

# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Berkembangnya penggunaan sistem informasi di jaman sekarang mengakibatkan data dapat dihasilkan dalam jumlah yang sangat banyak. Data yang jumlahnya sangat banyak ini dikumpulkan dan disimpan dalam tabel basis data untuk keperluan analisis di masa yang akan datang. Data yang dikumpulkan dapat mengambil kapasitas penyimpanan yang besar, sehingga proses analisis data menjadi sangat lambat. Dampak yang ditimbulkan dari pertumbuhan data menyebabkan basis data konvensional menjadi kurang efektif untuk pengolahan data. Oleh karena itu, teknologi *big data* digunakan untuk mengurangi biaya penyimpanan dan komputasi data, sehingga kapasitas data dapat ditingkatkan dan data berukuran besar dapat lebih mudah untuk diolah.

*Big data* adalah kumpulan data yang telah dikumpulkan dalam jumlah yang sangat besar pada rentang waktu tertentu. *Big data* disimpan, diolah, dan dilakukan analisis agar menghasilkan informasi yang bermanfaat sebagai dasar pengambilan keputusan atau kebijakan yang lebih tepat berdasarkan data sebenarnya. Karena *Big data* memiliki ukuran data yang besar, maka proses analisis *big data* harus dilakukan secara paralel. Caranya adalah dengan membagi data ke beberapa komputer untuk diolah masing-masing komputer tersebut. Konsep ini disebut dengan sistem terdistribusi. Sistem terdistribusi adalah solusi pengolahan *big data* karena terbukti dapat mengurangi biaya penyimpanan dan komputasi data dari pemrosesan data secara paralel.

Untuk melakukan proses analisis data, diperlukan teknik untuk mencari tahu kesamaan sifat yang dimiliki oleh sekumpulan data. Salah satu teknik yang dapat digunakan adalah *data mining*. *Data mining* adalah teknik untuk melihat kesamaan sifat yang terbentuk dari sekumpulan data. Teknik *data mining* dapat membantu proses analisis data pada lingkungan *big data*. Pemodelan data mining untuk *big data* dijalankan pada sistem terdistribusi, sehingga waktu komputasi dapat diminimalkan. Hasil *data mining* dipakai untuk berbagai macam kebutuhan. Umumnya, sebuah perusahaan meminta data dari perusahaan lain untuk kebutuhan analisis. Hal ini dapat menimbulkan kasus pelanggaran privasi ketika perusahaan lain melakukan teknik *data mining* pada sekumpulan data yang masih banyak mengandung data privasi. Oleh karena itu, diperlukan teknik khusus agar perusahaan masih dapat mencari informasi yang berharga dari data yang diberikan dan privasi data tetap terlindungi meskipun data tersebut dilakukan proses *data mining*.

Perlindungan privasi pada *data mining* dapat dicapai dengan menggunakan metode enkripsi dan anonimisasi pada data yang akan diberikan. Enkripsi adalah metode yang memanfaatkan pola atau kunci tertentu untuk melindungi data yang sifatnya sensitif. Anonimisasi adalah metode yang menyamarkan satu atau lebih nilai atribut data agar data seseorang tidak dapat saling dibedakan dengan data lainnya. Salah satu kekurangan dari metode enkripsi dibandingkan metode anonimisasi adalah keamanan enkripsi dapat diretas melalui penalaran hubungan nilai atribut yang unik untuk setiap baris data. Penalaran ini dicapai dengan menggabungkan seluruh nilai atribut yang unik pada masing-masing baris data untuk membentuk sebuah pola kelompok data. Penalaran ini sangat berbahaya karena menghubungkan nilai atribut data yang secara tidak langsung dapat mengungkapkan entitas pemilik data. Dengan menerapkan konsep anonimisasi diharapkan nilai keterhubungan antaratribut data diperkecil sehingga privasi dapat terlindungi.

Dengan melakukan metode anonimisasi pada sebagian nilai atribut data untuk kelompok data yang sama, maka bobot informasi yang diperoleh akan semakin kecil. Bobot informasi menunjukkan seberapa besar peluang untuk mengetahui arti dari nilai data yang telah dianonimisasi. Oleh karena itu, semakin kecil bobot informasi yang diperoleh maka kelompok data yang dapat membentuk entitas data akan semakin kecil sehingga perlindungan privasi akan semakin aman. Akan tetapi dengan semakin kecil bobot informasi yang diperoleh, maka nilai akurasi yang dihasilkan oleh metode anonimisasi akan semakin kecil. Nilai akurasi menunjukkan seberapa tepat model dapat menentukan sebuah data merupakan anggota dari kelompok data lain, sehingga kualitas informasi yang diperoleh semakin buruk. Oleh karena itu diperlukan cara untuk menyeimbangkan keamanan dan nilai akurasi informasi. Permasalahan *k-anonymity* melibatkan pencarian solusi dalam menyeimbangkan nilai akurasi informasi yang diperoleh dengan nilai informasi yang dapat dilindungi.

Metode *k-anonymity* dapat diuji menggunakan pendekatan generalisasi dan supresi. Hasil yang didapat dari penggunaan metode *k-anonymity* dinilai masih kurang untuk mendapatkan nilai akurasi data yang lebih baik, karena tingginya jumlah informasi yang hilang. Berdasarkan penelitian-penelitian yang telah dilakukan sebelumnya, permasalahan metode *k-anonymity* dapat teratasi melalui penerapan algoritma *k-member clustering* untuk pengelompokan data. Penerapan algoritma *k-member clustering* pada algoritma *Greedy k-member clustering* menjadi baik karena algoritma *greedy* mencari solusi optimal untuk meminimalkan jumlah informasi yang hilang. Agar algoritma *Greedy k-member clustering* dapat dijalankan pada lingkungan *big data* dengan efisien, maka akan dipilih *framework* Spark untuk waktu komputasi yang lebih optimal.

Spark adalah *framework* yang tepat untuk melakukan proses anonimisasi data pada lingkungan *big data*, karena pekerjaan pengolahan data yang besar dapat dibagi ke beberapa komputer pada sistem terdistribusi. Penggunaan Spark dipilih karena Hadoop memiliki waktu pemrosesan *big data* yang lebih lama dari Spark karena melakukan komputasi pada *hardisk*, sedangkan Spark dapat melakukan komputasi pada memori. Selain itu Spark memiliki jenis *library* yang lebih beragam dibandingkan dengan Hadoop. Spark mampu melakukan pemrosesan teknik *data mining* pada lingkungan *big data* menggunakan *library* tambahan yaitu Spark MLlib. Spark MLlib memfasilitasi pemodelan *data mining* yaitu klasifikasi dan pengelompokan/*clustering*. Kekurangan dari Spark adalah tidak memiliki penyimpanan yang tetap, sehingga membutuhkan mekanisme penyimpanan Hadoop, agar hasil pemrosesan data dapat tersimpan dalam *hardisk* komputer.

Pada skripsi ini, akan dibuat dua jenis perangkat lunak yaitu perangkat lunak anonimisasi data dan perangkat lunak analisis data. Perangkat lunak anonimisasi data menggunakan konsep *k-anonymity* dengan algoritma *Greedy k-member clustering* agar sebuah data tidak dapat dibedakan dengan  $k - 1$  data lainnya. Perangkat lunak anonimisasi data dibuat dengan bahasa Scala dan berjalan di atas Spark untuk meminimalkan waktu komputasi proses anonimisasi di lingkungan *big data*. Algoritma *Greedy k-member clustering* dinilai tepat untuk melakukan pengelompokan data karena meminimalkan jumlah informasi yang hilang saat proses *data mining* yang terbukti pada penelitian sebelumnya. Kedua jenis perangkat lunak ini menerima data input dalam format CSV. Untuk tampilannya, kedua perangkat lunak ini akan dibuat menggunakan GUI dari *library* Scala-swing. Penelitian ini memiliki tujuan utama yaitu membandingkan nilai akurasi dari hasil teknik *data mining* sebelum dan setelah dilakukan proses anonimisasi data.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah pada skripsi ini adalah sebagai berikut:

1. Bagaimana cara kerja algoritma *Greedy k-member clustering* ?
2. Bagaimana implementasi algoritma *Greedy k-member clustering* pada Spark?
3. Bagaimana hasil *data mining* sebelum dan setelah dilakukan anonimisasi?

## 1.3 Tujuan

Berdasarkan rumusan masalah di atas, tujuan dari skripsi ini adalah sebagai berikut:

1. Mempelajari cara kerja algoritma *Greedy k-member clustering*.
2. Mengimplementasikan algoritma *Greedy k-member clustering* pada Spark.
3. Menganalisis hasil *data mining* sebelum dan setelah dilakukan anonimisasi.

## 1.4 Batasan Masalah

Batasan masalah pada pengerjaan skripsi ini adalah sebagai berikut:

1. Perangkat lunak dapat berjalan diatas Spark.
2. Perangkat lunak dapat menerapkan algoritma *Greedy k-member clustering*.
3. Perangkat lunak dapat diimplementasikan menggunakan *library* Scala-swing.
4. Perangkat lunak hanya menerima input data semi terstruktur CSV dan XML.
5. Menggunakan teknik *data mining* yang tersedia pada *library* Spark MLlib.
6. Membandingkan hasil *data mining* sebelum dan setelah dilakukan anonimisasi.

## 1.5 Metodologi

Bagian-bagian pengerjaan skripsi ini adalah sebagai berikut:

1. Mempelajari dasar-dasar privasi data.
2. Mempelajari konsep *k-anonymity* pada algoritma *Greedy k-member clustering*.
3. Mempelajari teknik-teknik dasar *data mining*.
4. Mempelajari konsep Hadoop, Spark, dan Spark MLlib.
5. Mempelajari bahasa pemrograman Scala pada Spark.
6. Melakukan analisis masalah dan mengumpulkan data studi kasus.
7. Mengimplementasikan algoritma *Greedy k-member clustering* pada Spark.
8. Mengimplementasikan tampilan perangkat lunak menggunakan *library* Scala-swing.
9. Mengimplementasikan teknik *data mining* menggunakan *library* Spark MLlib.
10. Melakukan pengujian fungsional dan experimental.
11. Melakukan analisis hasil *data mining* sebelum dan setelah dilakukan anonimisasi.
12. Menarik kesimpulan berdasarkan hasil eksperimen yang telah dilakukan.

## 1.6 Sistematika Pembahasan

Pengerjaan skripsi ini tersusun atas enam bab sebagai berikut:

- Bab 1 Pendahuluan  
Berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan.
- Bab 2 Landasan Teori  
Berisi landasan teori mengenai konsep privasi, teknik *data mining*, *privacy-preserving data mining*, *k-anonymity*, algoritma *greedy k-member clustering*, metrik *distance* dan *information loss*, teknologi *big data*, pemrograman scala, dan format penyimpanan data.
- Bab 3 Analisis  
Berisi analisis penelitian mengenai analisis masalah (dataset eksperimen, *personally identifiable information*, perhitungan *distance* dan *information loss*, algoritma *greedy k-member clustering*, *k-anonymity*, *domain generalization hierarchy*), eksplorasi spark (instalasi spark, pembuatan *project spark*, menjalankan program spark), studi kasus (eksperimen scala, eksperimen spark), dan gambaran umum perangkat lunak (diagram kelas dan diagram aktivitas).
- Bab 4 Perancangan  
Berisi perancangan antarmuka perangkat lunak anonimisasi data dan analisis data, diagram kelas lengkap, masukan perangkat lunak anonimisasi data dan analisis data.
- Bab 5 Implementasi dan Pengujian  
Berisi implementasi perangkat lunak anonimisasi data dan analisis data, pengujian fungsional, pengujian eksperimental, dan melakukan analisis terhadap hasil pengujian.
- Bab 6 Kesimpulan dan Saran  
Berisi kesimpulan penelitian dan saran untuk penelitian selanjutnya.

## BAB 2

### LANDASAN TEORI

Pada bab ini, akan dijelaskan konsep mengenai privasi, teknik *data mining*, *privacy-preserving data mining*, *k-anonymity*, algoritma *greedy k-member clustering*, metrik *distance* dan *information loss*, teknologi *big data*, pemrograman skala, dan format penyimpanan data sebagai landasan penelitian.

#### 2.1 Privasi

Privasi adalah suatu keadaan dimana kehidupan pribadi seseorang atau sekelompok orang terbebas dari pengawasan atau gangguan orang lain. Privasi juga dapat berarti kemampuan satu atau sekelompok individu untuk menutupi atau melindungi kehidupan dan urusan personalnya dari publik dengan mengontrol sumber-sumber informasi mengenai diri mereka. Untuk melakukan publikasi data dari satu perusahaan ke perusahaan lain, digunakan teknik anonimisasi data untuk melindungi dan menyamarkan atribut sensitif untuk setiap data.

*Personally Identifiable Information* (PII) adalah standar yang digunakan untuk menentukan apakah informasi yang ada dapat melakukan identifikasi entitas individu secara langsung atau tidak langsung. PII menjelaskan bahwa identifikasi entitas secara langsung dapat dilakukan menggunakan atribut sensitif. Sedangkan identifikasi entitas secara tidak langsung dapat dilakukan menggunakan penggabungan beberapa atribut non-sensitif. PII adalah atribut yang biasanya terjadi pelanggaran data dan pencurian identitas. Jika data perusahaan atau organisasi terungkap, maka sangat mungkin data pribadi seseorang akan terungkap. Informasi yang diketahui dapat dijual dan digunakan untuk melakukan pencurian identitas, menempatkan korban dalam risiko.

Berikut adalah contoh informasi yang bersifat sensitif menurut standar PII:

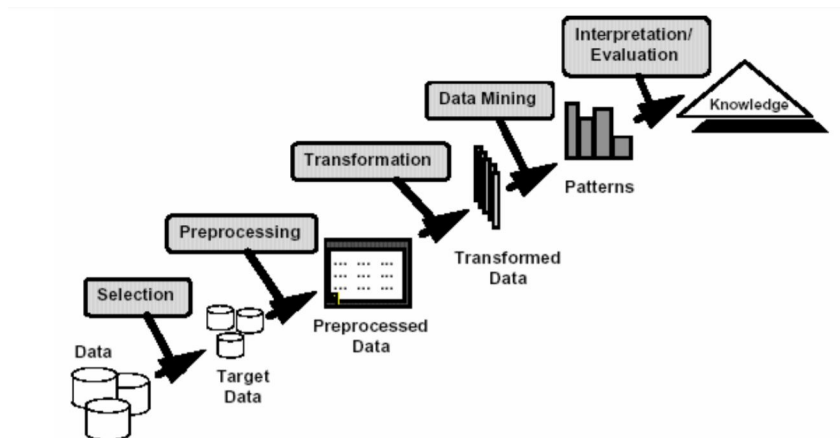
- Identitas diri  
Nama lengkap, tempat tanggal lahir, alamat rumah, alamat email.
- Nomor identitas diri  
NIK, nomor passport, nomor SIM, nomor wajib pajak, nomor rekening, nomor telepon, dan nomor kartu kredit.
- Data biometrik  
Pemindaian retina, jenis suara, dan geometri wajah.

Berikut adalah contoh informasi yang bersifat non-sensitif menurut standar PII:

- Rekaman medis
- Riwayat pendidikan
- Riwayat pekerjaan
- Informasi finansial
- Letak geografis

## 2.2 Data Mining

Data yang dikumpulkan bertambah banyak, sehingga perlu adanya cara untuk melakukan proses ekstraksi informasi pada sekumpulan data yang sangat banyak. Menurut Gartner, *data mining* adalah proses menemukan korelasi, pola, dan tren baru yang bermakna dengan menyaring sejumlah besar data yang disimpan menggunakan teknologi pengenalan pola serta teknik statistik dan matematika. *Data mining* merupakan bagian dari *Knowledge Discovery in Databases* (KDD). KDD adalah proses transformasi sekumpulan data yang disimpan pada basis data menjadi informasi yang berguna.



Gambar 2.1: Tahapan pada KDD

Berikut ini adalah penjelasan tahapan pada KDD pada Gambar 2.1 sebagai berikut:

1. *Selection*: proses mengambil data yang relevan terhadap analisis.
2. *Preprocessing*: proses pembersihan data dari data yang tidak konsisten dan integrasi data saat penggabungan data.
3. *Transformation*: proses manipulasi data menggunakan konsep agregasi, generalisasi, normalisasi, dan reduksi untuk kebutuhan analisis.
4. *Data mining*: proses ekstraksi informasi menggunakan metode pengenalan pola seperti klasifikasi, pengelompokan/*clustering*.
5. *Interpretation/evaluation*: proses interpretasi hasil pengolahan data menjadi sebuah grafik yang dapat dimengerti.

Berikut adalah beberapa jenis tipe data terkait teknik data mining:

- *Binary*: tipe data alphabet/numerik yang hanya memiliki 2 kemungkinan nilai.  
Contoh: nilai true/false dan 0/1.
- *Nominal*: tipe data alphabet/numerik yang memiliki lebih dari 2 kemungkinan nilai.  
Contoh: warna kuning, hijau, hitam, merah.

Tujuan dari penggunaan teknik *data mining* adalah sebagai berikut:

- *Prediksi*: proses menggunakan nilai dari beberapa atribut yang sudah ada untuk memprediksi nilai atribut di masa yang akan datang. Contoh: klasifikasi.
- *Deskripsi*: proses menemukan pola yang dapat merepresentasikan kelompok dari sebuah data. Contoh: *clustering*.

### 2.2.1 Klasifikasi

Klasifikasi adalah proses menemukan model (atau fungsi) yang cocok untuk mendeskripsikan dan membedakan sebuah kelas data dengan kelas data lain. Dalam pembelajaran mesin, klasifikasi sering dianggap sebagai contoh dari metode pembelajaran yang diawasi, yaitu menyimpulkan fungsi dari data pelatihan berlabel.

Berikut adalah tahapan klasifikasi secara umum:

1. Pelatihan: proses konstruksi model klasifikasi menggunakan algoritma tertentu. Algoritma digunakan untuk membuat model belajar menggunakan set pelatihan data yang tersedia. Model dilatih untuk menghasilkan prediksi yang akurat.
2. Klasifikasi: model yang digunakan untuk memprediksi label kelas dan menguji model yang dibangun pada data uji dan karenanya memperkirakan akurasi aturan klasifikasi.

Berikut adalah kategori pemodelan klasifikasi:

- *Discriminative*: pemodelan paling mendasar untuk menentukan satu kelas untuk setiap baris data. Pemodelan ini bergantung pada data yang diamati dan sangat bergantung pada kualitas data daripada distribusi data.

```
Student 1 : Test Score: 9/10, Grades: 8/10 Result: Accepted
Student 2 : Test Score: 3/10, Grades: 4/10, Result: Rejected
Student 3 : Test Score: 7/10, Grades: 6/10, Result: to be tested
```

Gambar 2.2: Contoh *Logistic Regression*

Contoh: *Logistic Regression*

Gambar 2.2 adalah penerimaan siswa pada sebuah Universitas, untuk mempertimbangkan *test score* dan *grades* terhadap keputusan seorang siswa diterima/tidak diterima.

- *Generative*: pemodelan ini memodelkan distribusi kelas individu dan mencoba mempelajari model yang menghasilkan data dengan memperkirakan asumsi dan distribusi model. Digunakan untuk memprediksi nilai data yang belum diketahui.

Contoh: *Naive Bayes*

Mendeteksi email spam dengan melihat data sebelumnya. Misalkan dari 100 email yang ada dibagi menjadi kategori Kelas A: 25% (Email spam) dan Kelas B: 75% (Email Non-Spam). Ingin diperiksa apakah email berisi spam atau bukan. Pada Kelas A, 20 dari 25 email adalah spam dan sisanya bukan spam. Pada Kelas B, 70 dari 75 email bukan spam dan sisanya adalah spam. Probabilitas email yang berisi spam termasuk pemodelan *Naive Bayes*.

Berikut adalah contoh pemodelan klasifikasi yang umum digunakan:

- *Decision Trees*
- *Naive Bayes*
- *Neural Networks*
- *K-Nearest Neighbour*
- *Linear Regression*

### 2.2.2 *Naive Bayes*

*Naive Bayes* menerapkan klasifikasi dengan menggunakan metode probabilitas dan statistik. Pe-modelan ini mencari nilai probabilitas tertinggi pada masing-masing kelas menggunakan teorema *Bayes*. Kelas dengan probabilitas tertinggi akan dipilih sebagai hasil akhir. *Naive Bayes* mudah untuk dibangun dan memiliki komputasi yang lebih cepat daripada model klasifikasi lainnya.

Teorema *Bayes* menemukan probabilitas suatu peristiwa terjadi mengingat probabilitas peristiwa lain yang telah terjadi. Teorema *Bayes* dinyatakan secara matematis melalui persamaan berikut:

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} \quad (2.1)$$

Dari perhitungan probabilitas teorema *Bayes*, akan dicari kelas dengan probabilitas maksimum. Probabilitas maksimum dapat dinyatakan secara matematis melalui persamaan berikut:

$$MAP(H) = \max(P(H|D)) \quad (2.2)$$

Keterangan:

- $P(H|D)$  adalah probabilitas posterior apabila diberika hipotesis  $H$  dan diketahui data  $D$ .
- $P(D|H)$  adalah probabilitas posterior data  $D$  jika hipotesis  $h$  adalah benar.
- $P(H)$  adalah probabilitas hipotesis  $h$  adalah benar
- $P(D)$  adalah probabilitas data.

Tabel 2.1 diberikan untuk menggambarkan kondisi cuaca saat bermain golf. Masing-masing data dikategorikan berdasarkan nilai atribut *PlayGolf*, yaitu cocok (*Yes*) atau tidak cocok (*No*).

Tabel 2.1: Contoh Kasus *PlayGolf*

Outlook	Temperature	Humidity	Windy	PlayGolf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Berikut adalah pengelompokan nilai berdasarkan dataset yang telah diberikan:

- Vektor fitur  
Vektor fitur adalah vektor yang mewakili nilai fitur untuk setiap baris dataset. Vektor fitur pada contoh kasus ini terdiri dari atribut *Outlook*, *Temperature*, *Humidity*, dan *Windy*.



- Vektor respon

Vektor respon adalah nilai prediksi dalam bentuk label kelas untuk setiap baris data. Vektor respon pada contoh kasus ini diwakili oleh atribut *PlayGolf*.

Secara singkat, langkah kerja algoritma *Naive Bayes* dapat dijelaskan sebagai berikut:

1. Merepresentasikan teorema *Bayes* terhadap vektor fitur.

Berdasarkan dataset, teorema *Bayes* dapat diubah seperti berikut:

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)} \quad (2.3)$$

Di mana  $y$  adalah label kelas dan  $X$  adalah vektor fitur (dengan ukuran  $n$ ), dinyatakan melalui persamaan berikut:

$$X = (x_1, x_2, x_3, \dots, x_n) \quad (2.4)$$

Contoh:  $X = (Rainy, Hot, High, False), y = No$

Diasumsikan teorema *Bayes* saling independen terhadap fitur-fiturnya. Berikut adalah persamaan teorema *Bayes* baru, jika memakai lebih dari satu nilai atribut:

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y) \dots P(x_n|y)P(y)}{P(x_1)P(x_2) \dots P(x_n)} \quad (2.5)$$

2. Menghitung probabilitas nilai dari sebuah atribut.

Sebagai contoh, nilai yang dimiliki atribut *Outlook* adalah  $\{Sunny, Overcast, Rainy\}$ . Tabel 2.2 berfungsi untuk mencatat frekuensi dan menghitung probabilitas nilai dari atribut *Outlook*.

Tabel 2.2: Tabel Probabilitas pada Atribut *Outlook*

	Yes	No	P(Yes)	P(No)
Sunny	3	2	3/9	2/5
Overcast	4	0	4/9	0/5
Rainy	2	3	2/9	3/5
Total	9	5	100	100

Berikut adalah contoh perhitungan  $P(No)$  untuk nilai *Sunny* pada atribut *Outlook*

$$P(No) = \frac{\text{frekuensi}(Sunny \cap No)}{\text{frekuensi}(No)} = \frac{2}{5}$$

Berikut adalah contoh perhitungan  $P(Yes)$  untuk nilai *Sunny* pada atribut *Outlook*

$$P(Yes) = \frac{\text{frekuensi}(Sunny \cap Yes)}{\text{frekuensi}(Yes)} = \frac{3}{9}$$

Perhitungan probabilitas  $P(Yes)$  dan  $P(No)$  berlaku untuk nilai lainnya pada atribut *Outcast* yaitu  $\{Overcast, Rainy\}$ , sehingga hasil akhirnya dapat dilihat pada Tabel 2.2.

3. Membuat tabel probabilitas untuk atribut lainnya  $\{Temperature, Humidity, Windy\}$  dengan cara yang sama seperti langkah 2. Hasilnya akan menjadi Tabel 2.3, 2.4, 2.5

Tabel 2.3: Tabel Probabilitas dari Atribut Temperature

	Yes	No	P(Yes)	P(No)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Tabel 2.4: Tabel Probabilitas dari Atribut Humidity

	Yes	No	P(Yes)	P(No)
High	3	4	3/9	4/5
Normal	6	1	6/9	/5
Total	9	5	100	100

Tabel 2.5: Tabel Probabilitas dari Atribut Wind

	Yes	No	P(Yes)	P(No)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100	100

4. Menghitung probabilitas bersyarat terhadap data baru yang diberikan.

Contoh:  $today = (Sunny, Hot, Normal, NoWind)$

$$\begin{aligned}
 P(Yes|today) &= \frac{P(Sunny|Yes)P(Hot|Yes)P(Normal|Yes)P(NoWind|Yes)P(Yes)}{P(today)} \\
 &= \frac{3}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{2}{5} \cdot \frac{5}{14} = 0.0068
 \end{aligned}$$

$$\begin{aligned}
 P(No|today) &= \frac{P(Sunny|No)P(Hot|No)P(Normal|No)P(NoWind|No)P(No)}{P(today)} \\
 &= \frac{2}{9} \cdot \frac{2}{9} \cdot \frac{6}{9} \cdot \frac{6}{9} \cdot \frac{9}{14} = 0.0068
 \end{aligned}$$

5. Melakukan normalisasi terhadap probabilitas bersyarat.

$$P(Yes|today) = \frac{0.0141}{0.0141 + 0.0068} = 0.67$$

$$P(No|today) = \frac{0.0068}{0.0141 + 0.0068} = 0.33$$

Ketika probabilitas tersebut dijumlahkan, maka hasilnya akan menjadi 1.

$$P(Yes|today) + P(No|today) = 1 \quad (2.6)$$

6. Mencari probabilitas tertinggi.

Berdasarkan pernyataan berikut:

$$P(Yes|today) > P(No|today) \quad (2.7)$$

Dapat disimpulkan bahwa data dengan nilai (*Sunny, Hot, Normal, NoWind*) dapat diklasifikasi terhadap atribut *PlayGolf* dengan nilai label kelas *Yes*.

### 2.2.3 Clustering

*Clustering* merupakan proses mengelompokkan satu data ke dalam himpunan data yang disebut *cluster*. Objek di dalam *cluster* memiliki kemiripan karakteristik satu sama lain yang berbeda dengan *cluster* lainnya. *Clustering* sangat berguna untuk menemukan kelompok data yang tidak dikenal. *Clustering* sering disebut juga sebagai *outlier detection*.

Berikut adalah tahapan *clustering* secara umum:

1. Perhitungan *distance*: proses untuk mengukur kesamaan antar objek dengan cara menghitung *distance* antar 2 titik. Salah satu contoh metode pengukuran *distance* yang umum digunakan adalah *Euclidean distance*. Metode ini terdiri atas variabel  $p_i$ , menyatakan kordinat titik data dan variabel  $C_i$ , menyatakan kordinat titik *centroid* sebuah *cluster*.

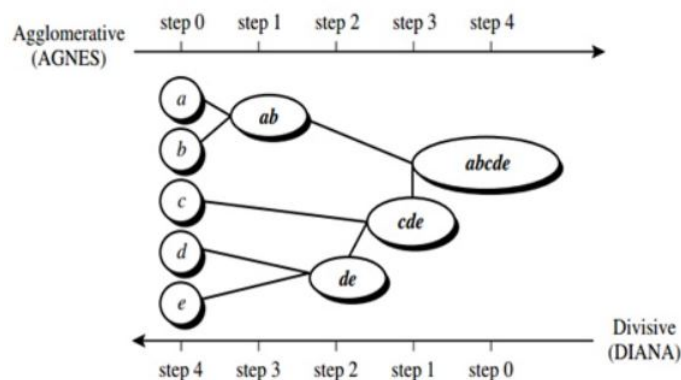
Berikut adalah persamaan untuk menghitung *Euclidean distance*:

$$EuclidDist(p_i, C_i) = \sqrt{(p_1 - C_1)^2 + (p_2 - C_2)^2 + \dots + (p_n - C_n)^2} \quad (2.8)$$

2. Pengelompokan data: proses pencarian anggota *cluster* dengan menghitung *distance* minimum antara masing-masing titik data dengan titik *centroid cluster* tersebut.

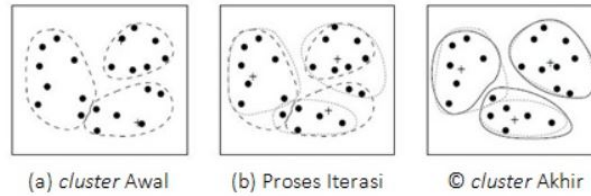
Berikut adalah kategori pemodelan *clustering*:

- *Hierarchical clustering*: pengelompokan data pada sebuah hirarki dengan cara menggabung dua kelompok data terdekat maupun membagi data ke dalam cluster tertentu. Contoh pemodelannya adalah *single linkage*, *complete linkage*, *average linkage*, *average group linkage*.



Gambar 2.3: Contoh *Hierarchical Clustering*

- *Partitional clustering*: pengelompokan data ke dalam sejumlah *cluster* tanpa adanya struktur hierarki. Pada metode ini, setiap *cluster* memiliki titik pusat cluster (*centroid*) dengan tujuan untuk meminimumkan (*dissimilarity distance*) seluruh data ke *centroid cluster* masing-masing. Contoh pemodelannya adalah *k-means*, *fuzzy k-means*, dan *mixture modeling*.



Gambar 2.4: Contoh *Partitional Clustering*

Berikut adalah contoh pemodelan *clustering* yang umum digunakan:

- *Agglomerative*
- *K-Means*

#### 2.2.4 *K-Means*

*K-means* merupakan metode clustering yang paling sederhana dan umum. *K-means* merupakan salah satu algoritma *clustering* dengan metode *partitional clustering* menggunakan titik *centroid* sebagai pusat kelompok data. *K-means* memerlukan tiga jenis paramater yaitu menentukan jumlah kelompok data ( $k$ ), inisialisasi titik *centroid* awal, dan mengetahui *distance* antar titik.

Tabel 2.6 diberikan untuk mengelompokan mata pelajaran yang sejenis berdasarkan data skor yang diperoleh dari individu A dan B:

Tabel 2.6: Tabel Dataset Mata Pelajaran

Subject	Person A	Person B
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Secara singkat, langkah kerja algoritma *k-means* dapat dijelaskan sebagai berikut:

1. Inisialisasi  $k$  dan titik *centroid* awal.

Pada contoh ini, jumlah *cluster* yang ingin dibentuk adalah dua kelompok data atau  $k = 2$  dengan inisialisasi titik *centroid* awal adalah (1.0, 1.0) dan (5.0, 7.0). Kedua titik ini dipilih secara acak. Data ini direpresentasikan pada Tabel 2.7 seperti berikut.

Tabel 2.7: Hasil Pengelompokan Awal

	Individual	Centroid
Cluster 1	1	(1.0, 1.0)
Cluster 2	4	(5.0, 7.0)

2. Melakukan perhitungan *distance*.

*Distance* digunakan untuk mencari jarak antara sebuah titik data dengan titik *centroid* dari *cluster* tertentu. Sebagai contoh, *distance* antara data ke-1 (1.5, 2.0) dengan titik *centroid* dari *Cluster 1* (1.0, 1.0) dihitung menggunakan *Euclidean distance* sebagai berikut.

$$\begin{aligned} EuclidDist(p_i, C_i) &= \sqrt{(p_1 - C_1)^2 + (p_2 - C_2)^2 + \dots + (p_n - C_n)^2} \\ &= \sqrt{(1.5 - 1.0)^2 + (2.0 - 1.0)^2} = 1.1180 \end{aligned}$$

3. Menempatkan setiap titik data ke titik *centroid* terdekat.

Titik data akan dikelompokkan ke *centroid* terdekat dengan memilih nilai Euclidian *distance* paling kecil. Sebagai contoh, Tabel 2.8 pada *Step 2* menyatakan data ke-1 lebih dekat dengan data ke-2, karena memiliki nilai Euclidean distance paling kecil. Karena itu, data ke-2 bergabung pada titik *centroid* data ke-1. Hal ini berlaku pada setiap langkah.

Tabel 2.8: Mencari Centroid Kelompok

Step	Cluster 1		Cluster 2	
	Individual	Mean Vector (centroid)	Individual	Mean Vector (centroid)
1	1	(1.0, 1.0)	4	(5.0, 7.0)
2	1, 2	(1.2, 1.5)	4	(5.0, 7.0)
3	1, 2, 3	(1.8, 2.3)	4	(5.0, 7.0)
4	1, 2, 3	(1.8, 2.3)	4, 5	(4.2, 6.0)
5	1, 2, 3	(1.8, 2.3)	4, 5, 6	(4.3, 5.7)
6	1, 2, 3	(1.8, 2.3)	4, 5, 6, 7	(4.1, 5.4)

Setelah pengelompokan data selesai dilakukan, selanjutnya perlu menghitung *Mean Vector* (*centroid*) sebagai titik *centroid* baru sebuah *cluster*. Sebagai contoh, *Mean Vector* pada *Cluster 1* untuk *Step 2* dapat dihitung sebagai berikut.

$$\begin{aligned} MeanVector(PersonA) &= \frac{1.0 + 1.5}{2} = 1.2 \\ MeanVector(PersonB) &= \frac{1.0 + 2.0}{2} = 1.5 \end{aligned}$$

4. Menetapkan kelompok data baru pada masing-masing *cluster*.

Iterasi paling terakhir di langkah sebelumnya akan dijadikan sebagai anggota dari *cluster* baru. Tabel 2.9 menunjukkan kelompok data sementara yang terbentuk pada masing-masing *cluster*. *Cluster 1* terdiri dari anggota data {1, 2, 3}, sedangkan *Cluster 2* terdiri dari anggota data {4, 5, 6, 7}. Kelompok data yang terbentuk saat ini masih sementara dan dapat berubah-ubah.

Tabel 2.9: Hasil Cluster Baru

	Individual	Mean Vector (centroid)
Cluster 1	1, 2, 3	(1.8, 2.3)
Cluster 2	4, 5, 6, 7	(4.1, 5.4)

5. Mencari titik *centroid* baru untuk setiap *cluster*.

Belum dipastikan bahwa setiap individu telah dialokasikan dengan tepat. Oleh karena itu, langkah 1 sampai 4 perlu kembali diulang dengan menghitung kembali *Euclidean distance*. Tabel 2.10 merupakan hasil perhitungan *distance* untuk setiap titik data.

Tabel 2.10: Euclidean Distance Cluster 1, Cluster 2

Individual	Distance centroid of Cluster 1	Distance centroid of Cluster 2
1	1.5	5.4
2	0.4	4.3
3	2.1	1.8
4	5.7	1.8
5	3.2	0.7
6	3.8	0.6
7	2.8	1.1

6. Menetapkan kelompok data akhir.

Apabila *cluster* tidak mengalami perubahan secara signifikan pada anggotanya selama periode iterasi tertentu, maka *cluster* yang terbentuk sudah sesuai. Tabel 2.11 menunjukkan anggota dari *cluster* yang sudah tetap, sehingga proses *k-means* dapat dihentikan.

Tabel 2.11: Hasil Pengelompokan Akhir

	Individual	Mean Vector (centroid)
Cluster 1	1, 2	(1.3, 1.5)
Cluster 2	3, 4, 5, 6, 7	(3.9, 5.1)

## 2.3 Tahapan Evaluasi *Data Mining*

Setelah melakukan implementasi model *data mining*, langkah selanjutnya adalah mengevaluasi hasil dari model *data mining* yang telah dibuat. Tahapan evaluasi sangat penting untuk mengukur seberapa tinggi tingkat akurasi model yang dipilih untuk menyelesaikan sebuah permasalahan data. Metode evaluasi untuk setiap model *data mining* berbeda satu sama lain. Oleh karena itu, perlu untuk mendefinisikan perhitungan evaluasi untuk masing-masing model.

### 2.3.1 Menghitung Tingkat Akurasi untuk Model Klasifikasi

Hasil pemodelan klasifikasi dapat dievaluasi menggunakan perhitungan tingkat akurasi. Semakin tinggi tingkat akurasi yang diperoleh, maka hasil pemodelan klasifikasi menjadi semakin baik. Akurasi dihitung dari rasio jumlah prediksi yang benar dengan jumlah total sampel input.

Berikut adalah persamaan untuk menghitung tingkat akurasi:

$$\text{Tingkat Akurasi} = \frac{\text{jumlah prediksi yang benar}}{\text{total sampel input}} \times 100\% \quad (2.9)$$

Kisaran skor untuk tingkat akurasi adalah  $[0\%, 100\%]$ .

### 2.3.2 Menghitung Koefisien *Silhouette* untuk Model *Clustering*

Hasil pemodelan *clustering* dapat dievaluasi menggunakan perhitungan koefisien *silhouette*. Koefisien *silhouette* bertujuan untuk menghitung seberapa dekat sebuah objek dengan *intra-cluster* dan mengukur seberapa jauh objek yang sama dengan *cluster* terdekat lainnya. Semakin tinggi nilai koefisien *silhouette*, maka hasil pengelompokannya akan semakin baik. Koefisien *silhouette* dihitung berdasarkan rata-rata *distance* antara setiap titik data dengan titik *centroid intra-cluster* dan rata-rata *distance* antara setiap titik data dengan titik *centroid cluster* lainnya.

Kisaran skor koefisien *silhouette* adalah  $[-1, 1]$ , berikut adalah analisisnya:

- Skor 1, artinya sampel berada jauh dari *cluster* tetangganya.
- Skor 0, artinya bahwa sampel sangat dekat dengan *cluster* tetangganya
- Skor -1, artinya sampel telah dikelompokkan pada *cluster* yang salah.

Berikut adalah persamaan untuk menghitung koefisien *silhouette*:

$$\text{Silhouette Score} = \frac{(p - q)}{\max(p, q)} \quad (2.10)$$

Keterangan:

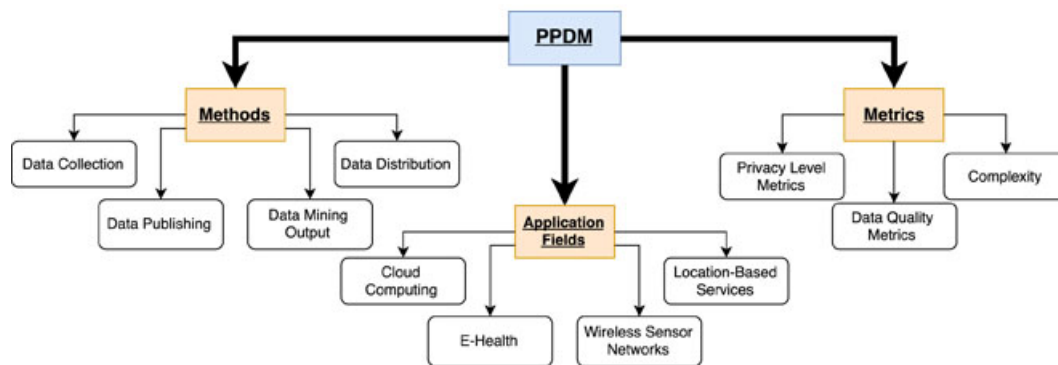
- $p$  = rata-rata *distance* setiap titik *intra-cluster* terhadap titik *centroid* pada *cluster* terdekat.
- $q$  = rata-rata *distance* setiap titik *intra-cluster* terhadap titik *centroid intra-cluster*.
- $\max(p, q)$  = nilai maksimum dari  $p$  dan  $q$ .

## 2.4 Privacy-Preserving Data Mining (PPDM)

Di era informasi saat ini, sistem informasi terus menghasilkan banyak informasi. Namun, banyak dari informasi yang dikumpulkan bersifat sensitif seperti data pribadi yang dapat menimbulkan masalah privasi. Untuk melindungi kebocoran informasi, metode preservasi privasi telah dikembangkan untuk melindungi entitas pemilik data dengan memodifikasi data asli menjadi data yang disamarkan atau data anonim. Akan tetapi metode ini memiliki kekurangan karena secara tidak langsung mengubah isi data, sehingga informasi yang didapat menjadi berkurang. Hal ini mengakibatkan ekstraksi informasi melalui proses *data mining* menjadi tidak akurat.

*Privacy Preserving Data Mining* (PPDM) adalah metodologi yang dirancang khusus untuk menjamin tingkat privasi pada level tertentu, sekaligus memaksimalkan utilitas data saat dilakukan proses data mining. PPDM mencakup semua teknik yang dapat digunakan untuk mengekstrak pengetahuan dari data dan secara bersama menjaga privasi dari data tersebut. Menurut IEEE, metodologi ini dapat dibagi menjadi beberapa bagian yaitu metode perlindungan privasi dan metrik perlindungan privasi yang akan dijelaskan pada bagian selanjutnya.

Gambar 2.5 menjelaskan garis besar pokok pembahasan dari PPDM yaitu metode, metrik, dan bidang penerapannya. Metode PPDM dapat diterapkan fase siklus data mulai dari pengumpulan data, publikasi data, hasil *data mining*, sampai kepada distribusi data. Selain itu, PPDM juga memiliki metrik untuk mengevaluasi dan membandingkan teknik-teknik yang digunakan agar mencapai tingkat privasi dan kualitas data tertentu. PPDM umumnya diaplikasikan pada bidang *cloud computing*, *e-health*, *wireless sensor*, dan *location-based service*.



Gambar 2.5: *Privacy Preserving Data Mining* (PPDM)

### 2.4.1 Metode pada PPDM

Beberapa metode PPDM telah diusulkan untuk menjamin perlindungan privasi pada masing-masing fase siklus hidup sebuah data di mana fase siklus tersebut sering terjadi pelanggaran privasi saat pengumpulan data, penerbitan data, distribusi data dan terhadap hasil *data mining*.

Berikut adalah metode perlindungan privasi pada PPDM:

- Pengumpulan data  
Pada fase ini, entitas yang bertugas mengumpulkan data tidak dapat dipercaya karena dapat mengumpulkan dan menggunakan data privasi individu secara tidak benar. Metode yang dapat digunakan untuk melindungi privasi saat pengumpulan data adalah *randomisation*.
- Publikasi data  
Pada fase ini, entitas ingin mempublikasikan datanya untuk keperluan penelitian atau analisis. Akan tetapi, ada beberapa individu yang mencoba melakukan tindakan deanonimisasi pada data-data privasi milik seseorang untuk tindakan kejahatan. Metode yang dapat digunakan untuk melindungi privasi saat publikasi data adalah *k-anonymity*.
- Hasil data mining  
Hasil data mining biasanya dibuat mudah diakses melalui aplikasi atau *interface*. Pada fase ini, entitas dapat melakukan kueri pada sistem untuk mencari informasi sensitif seseorang. Metode yang dapat digunakan untuk melindungi privasi pada hasil data mining adalah *association rule hiding*, *downgrading classifier effectiveness*, dan *query auditing and inference control*.
- Distribusi data  
Pada fase ini, beberapa entitas akan mencari wawasan umum dalam bentuk data statistik tanpa mengungkap informasi entitas lain. Akan tetapi ada beberapa entitas yang sengaja untuk menemukan celah keamanan privasi melalui pencocokan data dengan informasi umum yang diberikan secara publik. Hal ini dapat dicegah dengan memberlakukan *multiparty protocol* pada sistem distribusi data.

### 2.4.2 Metrik pada PPDM

Karena privasi tidak memiliki standar definisi yang jelas, maka diperlukan metrik untuk mengukur keamanan privasi. Melalui metode PPDM, beberapa metrik perlindungan privasi telah diusulkan. Metrik PPDM terbagi menjadi dua kategori utama yaitu privasi data dan kualitas data.



Berikut adalah metrik perlindungan privasi pada PPDM:

- **Tingkatan Privasi**  
Tingkatan privasi memberikan gambaran seberapa aman data diolah agar terhindar dari kasus pelanggaran privasi. Metrik ini dapat ditemukan pada model *k-anonymity*. Model *k-anonymity* berpengaruh terhadap tingkat privasi, dimana nilai *k* dapat diubah sesuai keinginan dalam menentukan tingkat keamanan data.
- **Kualitas Data**  
Kualitas data dinilai dari tiga parameter penting. *Accuracy*, untuk mengukur seberapa dekat nilai data yang disamarkan dengan data asli. *Completeness*, untuk mengevaluasi banyaknya data yang hilang. *Consistency*, untuk mengukur hilangnya korelasi data pada data yang disamarkan. Pada pemodelan *k-anonymity*, metrik yang dapat diukur untuk menyatakan kualitas data adalah *Information Loss* (IL).

### 2.4.3 Model Serangan pada PPDM

Menurut Dalenius (1977), perlindungan privasi tidak memberikan kesempatan bagi orang lain untuk mendapatkan informasi sensitif individu meskipun orang lain mengetahui informasi umum yang berhubungan dengan informasi sensitif individu tersebut. Secara umum, orang lain dapat menemukan sebuah cara untuk memetakan sebuah data ke dalam tabel yang telah dianonimisasi ketika data tersebut telah dipublikasi. Serangan ini dikenal dengan istilah *linkage attack*.

Berikut adalah jenis serangan dari *linkage attack*:

- **Record Linkage**  
*Record Linkage* mengacu serangan mencocokkan antara record korban yang dirilis dengan record korban yang telah diketahui untuk menebak atribut sensitif milik korban. Jika record tersebut cocok dengan record lain, artinya kedua data saling berkaitan. Serangan ini dapat dicegah dengan penggunaan model *k-anonymity*.
- **Attribute Linkage**  
*Attribute Linkage* mengacu serangan untuk mendapatkan beberapa informasi nilai atribut sensitif korban dengan mencari keterhubungan atribut. Tipe serangan ini mencari keterhubungan nilai atribut non sensitif untuk menebak atribut sensitif milik korban. Serangan ini dapat dicegah dengan penggunaan model *l-diversity*.

## 2.5 Anonimisasi

Anonimisasi data mengacu pada penghapusan atau enkripsi informasi pribadi atau data sensitif. Karena bisnis, pemerintah, sistem perawatan kesehatan, dan organisasi lain semakin banyak menyimpan informasi individu di server lokal atau cloud, anonimisasi data sangat penting untuk menjaga integritas data dan mencegah pelanggaran keamanan. Di sektor kesehatan dan keuangan, data dapat menjadi hal yang sangat sensitif. Data pasien atau entitas lain harus disembunyikan untuk memenuhi persyaratan perlindungan privasi.

Anonimisasi berupaya melindungi data pribadi atau atribut sensitif dengan menghapus atau mengenkripsi informasi yang dapat diidentifikasi secara pribadi dari database. Anonimisasi data dilakukan dengan tujuan melindungi aktivitas pribadi individu atau perusahaan sambil menjaga integritas data yang dikumpulkan dan dibagikan. Teknik anonimisasi data juga dikenal sebagai "*obfuscation data*", "*data masking*". Sedangkan *data de-anonymization* merupakan teknik yang digunakan pada proses *data mining* dengan tujuan untuk mengidentifikasi kembali informasi yang hilang pada data yang dienkripsi atau dianonimisasi.

## 2.6 *K-Anonymity*

*K-anonymity* merupakan model yang paling efektif untuk melindungi privasi saat melakukan publikasi data. *K-anonymity* adalah contoh pemodelan dari keamanan informasi yang bertujuan agar sebuah data tidak dapat dibedakan setidaknya dengan  $k - 1$  data lainnya. Dengan kata lain, penyerang tidak dapat mengidentifikasi identitas dari satu data karena  $k - 1$  data yang lain memiliki sifat yang sama. Dalam pemodelan *k-anonymity*, nilai  $k$  dapat digunakan sebagai tingkat keamanan privasi. Semakin tinggi nilai  $k$ , semakin sulit untuk mengidentifikasi sebuah data. Secara teori, probabilitas identifikasi sebuah data adalah  $1 / k$ . Namun, peningkatan nilai  $k$  juga dapat berpengaruh terhadap nilai informasi yang diperoleh dari sekumpulan data.

Penelitian menunjukkan bahwa sebagian besar pemodelan *k-anonymity* menggunakan metode generalisasi dan supresi. Pendekatan tersebut menderita kehilangan informasi yang signifikan karena mereka sangat bergantung terhadap hubungan relasi antar atribut. Oleh karena itu, hasil anonimisasi menghasilkan nilai kehilangan informasi yang cukup tinggi. Selain itu, algoritma anonimisasi yang ada hanya berfokus pada perlindungan informasi pribadi dan mengabaikan utilitas data yang sebenarnya. Akibatnya, nilai utilitas pada data yang telah dianonimisasi memiliki nilai yang rendah. Beberapa algoritma telah diuji pada pemodelan *k-anonymity*.

Berikut algoritma yang dapat diterapkan pada pemodelan *k-anonymity*:

- Algoritma *k-means clustering* akan melakukan beberapa iterasi sampai *centroid* dari semua data tidak lagi berubah atau perubahannya kecil. Algoritma *k-means clustering* tidak mampu untuk menyelesaikan masalah pada atribut yang bernilai kategorikal. Kelebihan dari algoritma *k-means clustering* adalah memiliki hasil pengelompokan yang sudah baik. Kekurangan dari algoritma *k-means clustering* adalah pemilihan *centroid* awal *k-means* secara acak, sehingga setelah digeneralisasi hasil pengelompokannya mengakibatkan hilangnya informasi yang besar.
- Algoritma *k-member* dapat melakukan generalisasi atribut kategorikal dengan memperoleh kualitas informasi yang lebih baik daripada algoritma *k-means clustering*. Namun algoritma *k-member* masih memiliki masalah ketika melakukan pengelompokan data. Kekurangan dari algoritma *k-member* adalah hanya mempertimbangkan pengelompokan data pada baris yang terakhir tanpa memperhatikan pengelompokan yang dihasilkan pada proses sebelumnya sehingga menyebabkan distribusi kelompok data pada beberapa bagian menjadi kurang tepat.
- Untuk menghindari kekurangan pada algoritma *k-means* dan algoritma *k-member*, maka kedua pendekatan ini digabung menjadi algoritma baru yaitu *Greedy k-member clustering*. Algoritma *Greedy k-member clustering* mendapatkan hasil pengelompokan yang lebih tepat dan memiliki nilai informasi yang lebih baik meskipun dilakukan generalisasi. Hasil akhir dari algoritma *Greedy k-member clustering* adalah data-data yang sejenis telah dikelompokkan pada kelompok data yang sama. Untuk melakukan anonimisasi, digunakan konsep *Hierarchy Based Generalization*. *K-anonymity* menyamarkan data pada nilai quasi-identifier yang sama.

Berikut adalah atribut dari pemodelan *k-anonymity*, yaitu:

- *Identifier* (ID) adalah atribut yang unik dan secara langsung dapat mengidentifikasi seseorang seperti nama, nomor ID, dan nomor ponsel.
- *Quasi-identifier* (QID) adalah kombinasi atribut yang mungkin terjadi untuk mengidentifikasi individu berdasarkan penggabungan informasi lain dari luar. Seluruh atribut data terkecuali atribut identifier dapat dianggap sebagai atribut quasi-identifier.
- *Sensitive Attribute* (SA) adalah atribut yang menyangkut informasi sensitif seseorang, biasanya nilai atribut ini akan dirahasiakan saat distribusi data.
- *Non-sensitive Attribute* (NSA) adalah atribut yang tidak menyangkut informasi sensitif seseorang, biasanya nilai atribut ini langsung ditampilkan saat distribusi data.

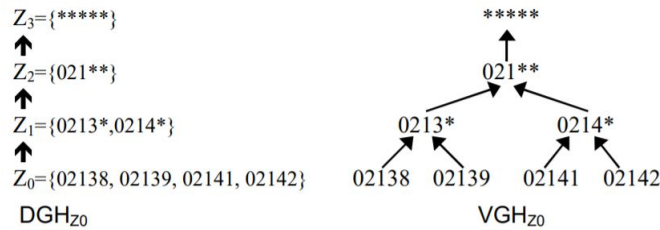
## 2.7 Hierarchy Based Generalization

*Hierarchy-based generalization* adalah tahapan anonimisasi pada pemodelan  $k$ -anonymity dimana *quasi-identifier* yang sama dikelompokkan ke dalam cluster yang sama. *Hierarchy-based generalization* menggunakan konsep generalisasi dan supresi dalam melakukan anonimisasi. *Hierarchy-based generalization* termasuk metode *full-domain generalization*. *Full-domain generalization* diusulkan oleh Samarati dan Sweeney untuk memetakan seluruh domain untuk masing-masing atribut *quasi-identifier* pada tabel ke domain yang lebih umum berdasarkan kategori tertentu.

*Full-domain generalization* dapat digunakan oleh model  $k$ -anonymity untuk menentukan generalisasi dan supresi dari sebuah nilai. *Full-domain generalization* menetapkan bahwa proses anonimisasi data dapat dilakukan dengan metode generalisasi apabila atribut *quasi-identifier* telah dipilih sejak awal. Sebagai contoh, dipilih atribut *quasi-identifier* sebagai berikut  $QI = \{A_1, \dots, A_n\}$ , dimana  $A$  adalah atribut pada tabel dataset. Terdapat dua jenis hierarki pada *Full-domain generalization*, yaitu Domain Generalization Hierarchy (DGH) dan Value Generalization Hierarchy (VGH). Jenis hierarki yang paling umum digunakan adalah DGH. DGH merupakan konsep sederhana dari penggantian nilai berdasarkan nilai yang kurang spesifik menjadi nilai lebih umum terhadap nilai aslinya. Tidak terdapat aturan khusus untuk memodelkan hierarki DGH. Beberapa nilai atribut harus digeneralisasi menggunakan DGH untuk mengakhiri proses anonimisasi data.

Pada Gambar 3.2, nilai ZIP  $\{02138, 02139\}$  dapat digeneralisasi menjadi  $\{0213*\}$  dengan menghilangkan digit paling kanan. Nilai yang telah digeneralisasi akan memiliki lingkup nilai yang lebih besar. Sebagai contoh, nilai ZIP adalah  $\{02138, 02139\}$  berada pada domain dasar yaitu  $Z_0$ . Untuk mencapai perlindungan data pada  $k$ -anonymity, maka kode ZIP yang sebelumnya unik harus diubah menjadi bentuk umum. Sebuah nilai dapat digeneralisasi jika memiliki domain yang lebih umum. Sedangkan domain yang kurang spesifik digunakan untuk mendeskripsikan garis besar nilai ZIP. Sebagai contoh dari domain kurang spesifik adalah  $Z_1$ , di mana digit terakhir diganti dengan simbol (\*). Berikut contoh pemetaan dari  $Z_0$  ke  $Z_1$  seperti berikut  $02139 \rightarrow 0213*$ .

Representasi generalisasi dapat diperluas, agar metode supresi dapat diterapkan dalam hirarki generalisasi. Elemen baru dengan nilai supresi maksimal (\*\*\*\*\* ) memiliki posisi lebih tinggi dibandingkan dengan elemen generalisasi ( $021**$ ). Ketinggian setiap hierarki generalisasi akan bertambah seiring bertambahnya kategori elemen generalisasi. Setelah elemen mencapai nilai supresi maksimal yang dapat digeneralisasi (\*\*\*\*\*), maka tidak akan ada lagi perubahan yang diperlukan untuk memasukkan elemen generalisasi yang baru. Gambar 3.2 dan 2.7 adalah contoh hierarki generalisasi DGH dan VGH pada atribut ZIP (kode pos) dan Race (warna kulit).



Gambar 2.6: DGH dan VGH pada Atribut ZIP



Gambar 2.7: DGH dan VGH pada Atribut Race

## 2.8 Greedy K-Member Clustering

Sebagian besar metode *k-anonymity* memakai metode generalisasi dan supresi sehingga data menderita kehilangan informasi yang signifikan. Masalah pengelompokan dipercaya dapat meminimalkan kehilangan informasi melalui implementasi algoritma *k-member clustering*. Akan tetapi algoritma tersebut memiliki kompleksitas eksponensial  $O(2^n)$ , sehingga perlu ditransformasi dengan algoritma *Greedy* dengan kompleksitas  $O(n \log n)$ . Algoritma *Greedy k-member clustering* bertujuan melakukan pengelompokan data ke masing-masing *cluster* dengan kompleksitas algoritma yang lebih baik dan dapat meminimalkan nilai informasi yang hilang saat dilakukan anonimisasi data.

**Teorema 1.** *Masalah pengambilan keputusan pada k-member clustering adalah NP-Hard. NP-Hard adalah suatu kelompok masalah dimana tidak ada algoritma yang dapat menemukan solusi optimal dengan kompleksitas lebih kecil dari polinomial.*

*Bukti.* Melalui percobaan Aggarwal et al dengan model *3-anonymity*, telah dibuktikan bahwa satu-satunya cara untuk melakukan anonimisasi atribut, yaitu dengan cara melakukan iterasi dari node paling atas (*root node*) ke node paling bawah (*leaf node*).  $\square$

**Teorema 2.** *N adalah total data dan k adalah parameter untuk anonimisasi. Setiap cluster yang ditemukan oleh algoritma Greedy k-member clustering memiliki jumlah tuple minimal sebanyak k, dan jumlah tuple tidak melebihi  $2k - 1$ .*

*Bukti.* S adalah himpunan data. Algoritma ini menemukan *cluster* selama jumlah data yang tersisa sama dengan atau lebih besar dari k, setiap *cluster* berisi k data. Jika total data pada S kurang dari k, maka sisa data akan dikelompokkan pada *cluster* yang sudah ada. Oleh karena itu, ukuran maksimum sebuah *cluster* adalah  $2k - 1$ .  $\square$

**Teorema 3.** *N adalah jumlah data dan k menjadi parameter anonimisasi yang ditentukan. Jika  $n > k$ , kompleksitas algoritma Greedy k-member clustering adalah  $O(n^2)$ .*

*Bukti.* Algoritma *Greedy k-member clustering* menghabiskan sebagian besar waktunya untuk memilih data dari S satu per satu hingga mencapai  $|S| = k$ . Karena ukuran set input berkurang satu pada setiap iterasi, total waktu eksekusi adalah  $O(n^2)$ .  $\square$

Beberapa hal penting terkait algoritma *Greedy k-means clustering*:

- Menetapkan tabel S
- Menetapkan nilai k
- Menetapkan jumlah cluster (m) yang ingin dibuat

$$m = \left\lfloor \frac{n}{k} \right\rfloor - 1 \quad (2.11)$$

Berikut adalah langkah kerja algoritma *Greedy k-means clustering* secara lengkap:

1. Melakukan inisialisasi variabel result dengan himpunan kosong dan variabel r dengan memilih data secara acak dari tabel S
2. Pada kondisi  $|S| \geq k$ , lakukan perulangan sebagai berikut:

- (a) Memilih data baru pada variabel  $r$  berdasarkan perbedaan  $distance$  tertinggi dari nilai  $r$  sebelumnya. Perbedaan  $distance$  dapat dicari menggunakan rumus berikut:

$$\Delta(r_1, r_2) = \sum_{i=1}^m \delta_N(r_1[N_i], r_2[N_i]) + \sum_{j=1}^n \delta_C(r_1[C_j], r_2[C_j])$$

Berikut adalah rumus menghitung *distance* antar data numerik:

$$\delta_n(v_1, v_2) = \frac{|v_1 - v_2|}{|D|}$$

Berikut adalah rumus menghitung  $distance$  antar data kategorikal:

$$\delta_C(v_1, v_2) = \frac{H(\Lambda(v_i, v_j))}{H(T_D)}$$

- (b) Membuang himpunan data variabel  $r$  pada variabel  $S$   
 (c) Mengisi data dari variabel  $r$  pada variabel  $c$ .  
 (d) Pada kondisi  $|c| \geq k$ , lakukan perulangan sebagai berikut:  
 i. Memilih data baru terbaik untuk variabel  $r$  berdasarkan nilai *Information Loss* (IL) yang paling rendah. *Information Loss* (IL) dapat dicari menggunakan rumus berikut:

$$IL(e) = |e| \cdot D(e)$$

$$D(e) = \sum_{i=1}^m \frac{(MAX_{N_i} - MIN_{N_i})}{|N_i|} + \sum_{j=1}^n \frac{H(\Lambda(\cup_{C_j}))}{H(T_{C_j})}$$

- ii. Membuang himpunan data dari variabel  $r$  pada variabel  $S$   
 iii. Menambahkan himpunan data dari variabel  $r$  pada variabel  $c$ .  
 iv. Menambahkan himpunan data dari variabel  $c$  pada variabel  $result$   
 3. Pada kondisi  $|S| \neq 0$ , artinya jika masih terdapat data yang belum dimasukkan pada sebuah *cluster* dari tabel  $S$ , lakukan perulangan sebagai berikut:

- (a) Memilih data secara acak dari tabel  $S$  untuk disimpan pada variabel  $r$   
 (b) Membuang himpunan data dari variabel  $r$  pada variabel  $S$   
 (c) Memilih *cluster* terbaik untuk variabel  $c$  berdasarkan nilai *Information Loss* (IL) yang paling rendah. *Information Loss* (IL) dapat dicari menggunakan rumus berikut:

$$IL(e) = |e| \cdot D(e)$$

$$D(e) = \sum_{i=1}^m \frac{(MAX_{N_i} - MIN_{N_i})}{|N_i|} + \sum_{j=1}^n \frac{H(\Lambda(\cup_{C_j}))}{H(T_{C_j})}$$

- (d) Menambahkan himpunan data dari variabel  $r$  pada variabel  $c$ .  
 4. Algoritma ini mengembalikan himpunan data berdasarkan jenis *cluster* yang berbeda-beda melalui variabel *result*.

Berikut adalah pseudocode secara lengkap dari algoritma *Greedy k-member clustering*:

---

**Algorithm 1** Find Best Record
 

---

```

1: Function find_best_record( $S, c$ )
2: Input: a set of records  $S$  and a cluster  $c$ .
3: Output: a record  $r \in S$  such that  $IL(c \cup \{r\})$  is minimal
4:
5:  $n = |S|$ 
6:  $min = \infty$ 
7:  $best = null$ 
8: for  $i = 1 \dots n$  do
9:    $r = i\text{-th record in } S$ 
10:   $diff = IL(c \cup \{r\}) - IL(c)$ 
11:  if  $diff < min$  then
12:     $min = diff$ 
13:     $best = r$ 
14:  end if
15: end for
16: return  $best$ 

```

---

Algoritma 1 menerima input himpunan data dataset dan sebuah data dengan nilai *distance* tertinggi dari data terpilih acak. Algoritma ini menghitung selisih *distance* dari dua jenis data yang berbeda. Variabel *diff* pada algoritma ini adalah perbedaan *distance*, dicari dengan penjumlahan *information loss* pada sebuah *cluster* dengan *information loss* pada data ke- $i$ , lalu hasil penjumlahan tersebut dikurangi dengan *information loss* dari *kluster*. Output algoritma ini adalah sebuah data dengan nilai terbaik, yaitu data ke- $i$  dari dataset  $S$  dengan nilai *distance* paling kecil.

---

**Algorithm 2** Find Best Cluster
 

---

```

1: Function find_best_cluster( $C, r$ )
2: Input: a set of cluster  $C$  and a record  $r$ .
3: Output: a cluster  $c \in C$  such that  $IL(c \cup \{r\})$  is minimal
4:
5:  $n = |C|$ 
6:  $min = \infty$ 
7:  $best = null$ 
8: for  $i = 1 \dots n$  do
9:    $c = i\text{-th cluster in } C$ 
10:   $diff = IL(c \cup \{r\}) - IL(c)$ 
11:  if  $diff < min$  then
12:     $min = diff$ 
13:     $best = c$ 
14:  end if
15: end for
16: return  $best$ 

```

---

Algoritma 2 menerima input himpunan data *cluster* dan sebuah data dengan nilai *distance* tertinggi dari data terpilih acak. Algoritma ini menghitung selisih *distance* dari dua jenis data yang berbeda. Variabel *diff* pada algoritma ini adalah perbedaan *distance*, dicari dengan penjumlahan *information loss* pada sebuah *cluster* dengan *information loss* pada data ke- $i$ , lalu hasil penjumlahan tersebut dikurangi dengan *information loss* dari *cluster*. Output algoritma ini adalah data dengan nilai *cluster* terbaik, yaitu data ke- $i$  dari dataset  $S$  dengan nilai *distance* paling kecil.

**Algorithm 3** Greedy K-Member Clustering

---

```

1: Function greedy_k_member_clustering( $S, k$ )
2: Input: a set of records  $S$  and a threshold value  $k$ 
3: Output: a set of clusters each of which contains at least  $k$  records.
4:
5: if  $S \leq k$  then
6:   return  $S$ 
7: end if
8:
9:  $result = \phi$ 
10:  $r$  = a randomly picked record from  $S$ 
11: while  $|S| \geq k$  do
12:    $r$  = the furthest record from  $r$ 
13:    $S = S - \{r\}$ 
14:    $c = \{r\}$ 
15:   while  $|c| < k$  do
16:      $r = \text{find\_best\_record}(S, c)$ 
17:      $S = S - \{r\}$ 
18:      $c = c \cup \{r\}$ 
19:   end while
20:    $result = result \cup \{c\}$ 
21: end while
22: while  $S \neq 0$  do
23:    $r$  = a randomly picked record from  $S$ 
24:    $S = S - \{r\}$ 
25:    $c = \text{find\_best\_cluster}(result, r)$ 
26:    $c = c \cup \{r\}$ 
27: end while
28: return  $result$ 

```

---

Algoritma 3 menerima input himpunan data  $S$  dan nilai  $k$ . Algoritma ini mengeksekusi dua jenis fungsi yang berbeda yaitu fungsi *find\_best\_cluster* untuk mencari *cluster* dengan *distance* terkecil dan fungsi *find\_best\_record* untuk mencari data dengan *distance* terkecil. Output dari algoritma ini adalah himpunan data dari berbagai jenis *cluster* dengan nilai *distance* terkecil.

## 2.9 Metrik *Distance* dan *Information Loss*

Konsep PPDM memberikan solusi untuk mengukur tingkat keamanan, fungsionalitas, dan utilitas data menggunakan beberapa jenis metrik. Beberapa metrik yang umum digunakan pada pengujian kualitas data yang telah dianonimisasi adalah *distance* dan *information loss*. Secara umum, pengukuran metrik dilakukan dengan membandingkan seberapa baik akurasi hasil data yang telah dianonimisasi dengan akurasi pada dataset sesungguhnya.

### 2.9.1 *Distance*

*Distance* adalah salah satu perhitungan untuk menyatakan akurasi terhadap utilitas sebuah data. *Distance* merupakan faktor yang paling penting untuk menentukan hasil pengelompokan data. Pemilihan *distance* yang baik dapat mencapai hasil klasifikasi dengan lebih optimal. Perhitungan *distance* dilakukan berdasarkan pengelompokan tipe data numerik atau kategorikal. Karena masalah *k-anonymity* menggunakan atribut numerik dan kategorikal, maka membutuhkan cara khusus untuk menghitung *distance* dari kedua jenis data pada saat yang sama.

### Distance Data Numerik

*Distance* data numerik direpresentasikan sebagai nilai rentang. Beberapa atribut pada *distance* numerik yaitu  $|D|$  adalah jumlah data pada sebuah domain berdasarkan satu atribut numerik,  $v_1, v_2$  adalah nilai atribut numerik. *Distance* data numerik dihitung menggunakan rumus berikut:

$$\delta_n(v_1, v_2) = \frac{|v_1 - v_2|}{|D|} \quad (2.12)$$

### Distance Data Kategorikal

*Distance* data kategorikal direpresentasikan sebagai *taxonomy tree*. Beberapa atribut pada *distance* kategorikal yaitu  $|D|$  adalah jumlah data pada domain kategorikal,  $TD$  adalah *taxonomy tree* untuk domain  $D$ ,  $H(\Lambda(v_i, v_j))$  adalah jarak dari satu *subtree* ke *subtree* lain,  $H(T_D)$  adalah tinggi dari *taxonomy tree*. *Distance* data kategorikal dihitung menggunakan rumus berikut:

$$\delta_C(v_1, v_2) = \frac{H(\Lambda(v_i, v_j))}{H(T_D)} \quad (2.13)$$

### Distance Record

Beberapa atribut pada *distance record* yaitu  $r_1[N_i], r_2[N_i]$  adalah nilai dari atribut numerik,  $r_1[C_j], r_2[C_j]$  adalah nilai dari atribut kategorikal,  $\delta_N$  adalah *distance* data numerik,  $\delta_C$  adalah *distance* data kategorikal. *Distance* record dihitung menggunakan rumus berikut:

$$\Delta(r_1, r_2) = \sum_{i=1}^m \delta_N(r_1[N_i], r_2[N_i]) + \sum_{j=1}^n \delta_C(r_1[C_j], r_2[C_j]) \quad (2.14)$$

## 2.9.2 Information Loss

*Information Loss* (IL) digunakan untuk mengevaluasi seberapa baik kinerja algoritma *k-anonymity* terhadap utilitas sebuah data. Dalam menghitung *Information Loss* (IL), perlu mendefinisikan beberapa atribut seperti *cluster*  $e = r_1, \dots, r_k$  untuk *quasi-identifier* yang terdiri dari atribut numerik  $N_1, \dots, N_m$  dan atribut kategorikal  $C_1, \dots, C_n$ ,  $T_{C_i}$  adalah *taxonomy tree* untuk domain kategorikal  $C_i$ ,  $MIN_{N_i}$  dan  $MAX_{N_i}$  adalah nilai minimum dan maksimum pada *cluster*  $e$  untuk atribut  $N_i$ ,  $\cup_{C_i}$  adalah sekumpulan nilai pada *cluster*  $e$  berdasarkan atribut  $C_i$ .

*Information loss* dihitung dengan rumus sebagai berikut:

$$IL(e) = |e| \cdot D(e) \quad (2.15)$$

$$D(e) = \sum_{i=1}^m \frac{(MAX_{N_i} - MIN_{N_i})}{|N_i|} + \sum_{j=1}^n \frac{H(\Lambda(\cup_{C_j}))}{H(T_{C_j})} \quad (2.16)$$

*Total Information Loss* dihitung dengan rumus sebagai berikut:

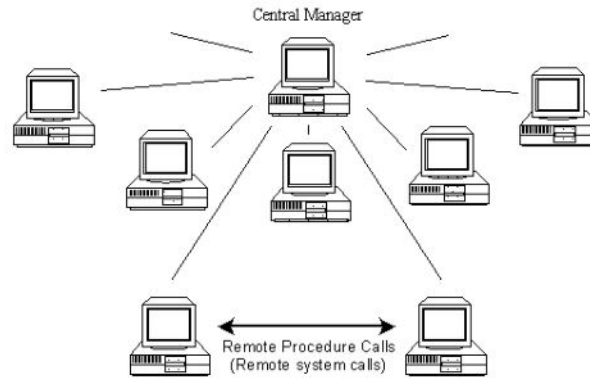
$$Total - IL(AT) = \sum_{e \in \epsilon} IL(e) \quad (2.17)$$

Semakin besar *total information loss* yang dihasilkan maka informasi yang dihasilkan semakin kurang akurat. Oleh karena itu perlu dilakukan beberapa eksperimen terhadap penentuan nilai  $k$  pada algoritma *Greedy k-member clustering* agar dihasilkan *total information loss* seminimal mungkin sehingga hasil *clustering* dan klasifikasi dengan nilai akurasi yang tinggi.



## 2.10 Sistem Terdistribusi

Sistem terdistribusi adalah kumpulan komputer berjalan secara independen dan saling terhubung dan saling bekerja sama untuk mencapai satu tujuan yang sama. Gambar 2.8 adalah ilustrasi dari cara kerja sistem terdistribusi secara paralel.



Gambar 2.8: Sistem Terdistribusi

### 2.10.1 Manfaat Sistem Terdistribusi

Berikut adalah manfaat penggunaan sistem terdistribusi:

- *Horizontal scalability*  
Sistem terdistribusi menawarkan kemampuan untuk melakukan pemrosesan komputasi skala besar pada big data dengan harga yang murah.
- *Reliability*  
Sistem terdistribusi dapat diandalkan karena proses komputasi pada sistem terdistribusi bergantung pada banyaknya komputer yang saling berkomunikasi satu sama lain untuk mencapai tujuan yang sama.
- *Performance*  
Sistem terdistribusi dapat menangani proses komputasi tugas secara efisien karena beban kerja sesungguhnya dibagi menjadi beberapa bagian dan tersebar di beberapa komputer.

### 2.10.2 Tantangan Sistem Terdistribusi

Berikut adalah tantangan pada sistem terdistribusi:

- *Penjadwalan*  
Kekuatan komputasi ada batasnya, sehingga sistem terdistribusi harus dapat memutuskan pekerjaan mana yang harus dikerjakan lebih dulu.
- *Latensi*  
Dengan pertukaran data antara perangkat keras dan perangkat lunak menggunakan jalur komunikasi jaringan, sehingga nilai latensi menjadi sangat tinggi.
- *Observasi*  
Ketika sistem terdistribusi menjadi kompleks, kemampuan pengamatan untuk memahami kegagalan pada sistem terdistribusi merupakan tantangan besar komputer.

## 2.11 Big Data

*Big data* adalah data yang besar dan kompleks sehingga tidak mungkin sistem tradisional dapat memproses dan bekerja pada lingkungan data yang besar secara maksimal. Data dapat dikategorikan sebagai data besar berdasarkan berbagai faktor. Konsep utama yang umum dalam semua faktor adalah jumlah data.

Berikut adalah karakteristik 5V pada *big data*:

- *Volume*  
*Volume* mengacu pada jumlah data yang sangat besar. Data tumbuh begitu besar sehingga sistem komputasi tradisional tidak lagi dapat menanganinya seperti yang kita inginkan.
- *Velocity*  
*Velocity* mengacu pada kecepatan di mana data dihasilkan. Setiap hari, sejumlah besar data dihasilkan, disimpan, dan dianalisis. Data dihasilkan dengan kecepatan kilat di seluruh dunia. Teknologi *big data* memungkinkan untuk mengeksplorasi data, saat data itu dihasilkan.
- *Variety*  
*Variety* mengacu pada berbagai jenis data. Data terutama dikategorikan ke dalam data terstruktur dan tidak terstruktur. Faktanya, lebih dari 75 persen data dunia ada dalam bentuk yang tidak terstruktur.
- *Veracity*  
*Veracity* mengacu pada kualitas data. Ketika menyimpan beberapa data yang besar, apabila tidak ada gunanya di masa depan, maka membuang-buang sumber daya untuk menyimpan data tersebut. Jadi, kita harus memeriksa kepercayaan data sebelum menyimpannya.
- *Value*  
*Value* adalah bagian terpenting dari *big data*. Organisasi menggunakan data besar untuk menemukan nilai informasi baru. Menyimpan sejumlah besar data sampai pada ekstraksi informasi yang bermakna dari sekumpulan data tersebut.

*Big data* memerlukan teknologi tertentu untuk melakukan komputasi. Pada bagian selanjutnya akan dijelaskan konsep-konsep terkait penggunaan *framework* beserta komponen-komponen penting pada *framework* tersebut terkait komputasi pada lingkungan *big data*. *Framework* tersebut antara lain Hadoop dan Spark. Masing-masing *framework* akan diteliti lebih lanjut, untuk dipilih pada penelitian ini berdasarkan kecepatan waktu komputasi.

## 2.12 Hadoop

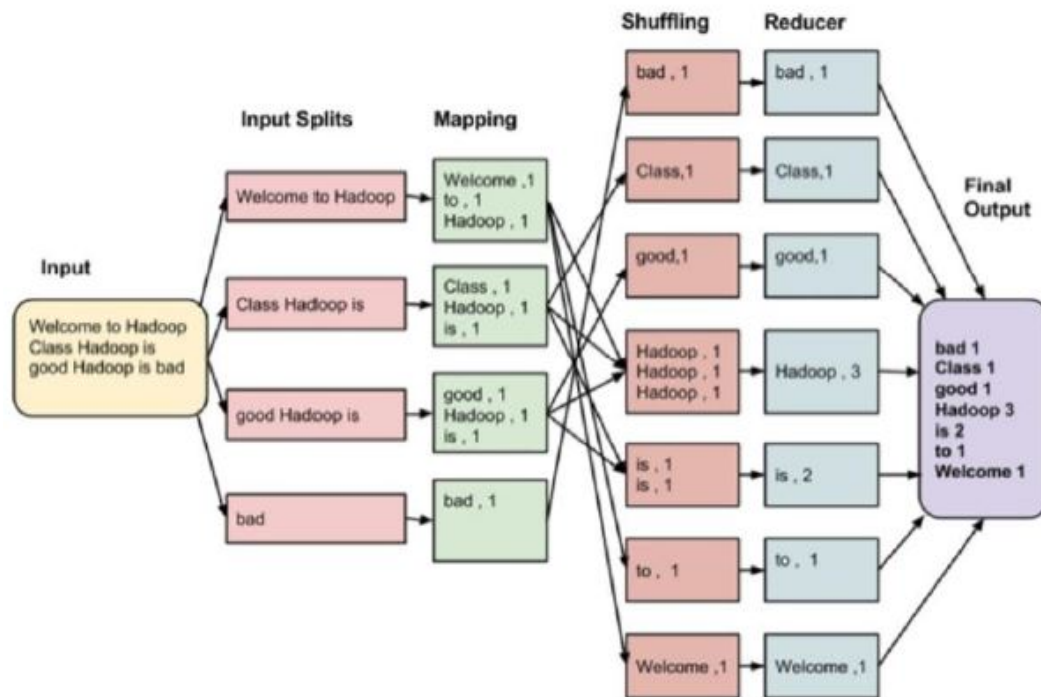
Hadoop adalah *framework* yang memanfaatkan beberapa komputer untuk menyelesaikan masalah yang melibatkan volume data sangat besar. Hadoop memecah input dari pengguna menjadi beberapa blok data dan masing-masing blok data diproses menggunakan konsep *MapReduce* di mana data akan diproses secara paralel pada sistem terdistribusi.

### 2.12.1 HDFS

HDFS adalah sistem *file* terdistribusi pada Hadoop yang menyediakan penyimpanan data yang handal dengan mendukung partisi data dan toleran terhadap kesalahan pada hardware. HDFS bekerja erat dengan MapReduce dengan mendistribusikan penyimpanan dan perhitungan di seluruh *cluster* dengan menggabungkan seluruh penyimpanan data menjadi terpusat.

### 2.12.2 MapReduce

*MapReduce* adalah model pemrograman untuk memproses data berukuran besar secara terdistribusi dan paralel pada cluster yang terdiri atas banyak komputer. Dalam memproses data, secara garis besar, MapReduce dibagi menjadi dua jenis proses yaitu map dan reduce. Setiap fase memiliki pasangan key-value sebagai input dan output. Kedua jenis proses ini didistribusikan ke setiap komputer dalam suatu cluster dan berjalan secara paralel tanpa saling bergantung satu sama lain.



Gambar 2.9: Proses Komputasi pada MapReduce

Berikut adalah penjelasan masing-masing tahapan pada MapReduce:

- **Input**  
Pada tahap ini, model MapReduce menerima input data secara utuh dari file text/CSV.
- **Input Splits** Pada tahap ini, model MapReduce akan memecah input data menjadi blok-blok data dan disebarkan ke seluruh cluster.
- **Mapping**  
*Mapping* bertujuan untuk memetakan blok-blok data ke dalam pasangan  $\langle key, value \rangle$ . Key, Value pada contoh ini adalah jenis kata dan jumlah jenis kata pada sebuah blok data.
- **Shuffling**  
Shuffling bertujuan untuk mengirim data dari Mapping ke Reducer, agar data dengan key yang sama akan dikelompokkan dan diolah oleh Reducer yang sama
- **Reducer**  
Reducer bertujuan sebagai proses penggabungan key,value dari proses shuffling untuk dihitung dan dikembalikan sebagai sebuah output
- **Output**  
Pada tahap ini, pemodelan MapReduce telah selesai. Output siap untuk ditulis pada file maupun ditampilkan pada console.

## 2.13 Spark

Spark adalah teknologi komputasi *cluster* yang dirancang untuk komputasi cepat. Spark adalah paradigma pemrosesan data berukuran besar yang dikembangkan oleh para peneliti *University of California di Berkeley*. Spark adalah alternatif dari Hadoop MapReduce untuk mengatasi keterbatasan pemrosesan input output yang tidak efisien pada disk, dengan menggunakan memori. Fitur utama Spark adalah melakukan komputasi di dalam memori sehingga waktu komputasi menjadi lebih singkat dibandingkan waktu komputasi di dalam *disk*.

Berikut adalah karakteristik dari Spark:

- Kecepatan  
Spark adalah alat komputasi *cluster* tujuan umum. Ini menjalankan aplikasi hingga 100 kali lebih cepat dalam memori dan 10 kali lebih cepat pada *disk* daripada Hadoop. Spark mengurangi jumlah operasi baca/tulis pada *disk* dan menyimpan data dalam memori.
- Mudah untuk diatur  
Spark dapat melakukan pemrosesan *batch*, analisis data secara interaktif, *machine learning*, dan *streaming data*. Semuanya pemrosesan tersebut dikerjakan pada satu komputer yang sama. Fungsi ini menjadikan Apache Spark sebagai mesin analisis data yang lengkap.
- Analisis secara *real-time*  
Spark dapat dengan mudah memproses data *real-time*, misalnya *streaming* data secara *real-time* untuk ribuan peristiwa/detik. Contoh dari sumber *streaming* data adalah Twitter, Facebook, Instagram. *Streaming* data dapat diproses secara efisien oleh Spark.

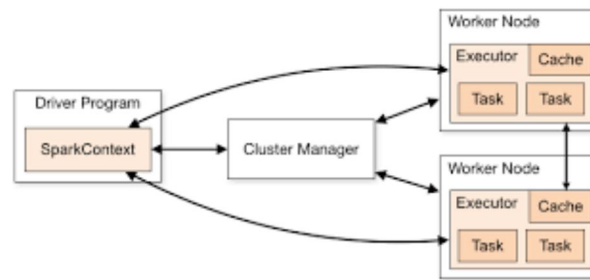
### 2.13.1 Ekosistem Spark



Gambar 2.10: Ekosistem Spark

Gambar 2.10 menunjukkan bahwa Spark bekerja sama dengan teknologi *big data* lain untuk memenuhi berbagai macam kebutuhan dalam pengolahan *big data*. Masing-masing warna pada Gambar 2.10 mewakili jenis teknologi yang dipakai pada Spark. Spark SQL, Spark Streaming, Spark MLlib adalah *library* tambahan pada Spark. Cassandra, Kafka, dan Elasticsearch adalah *framework* untuk melakukan pengumpulan data secara *streaming*. Scala, Java, dan Python adalah bahasa pemrograman yang dapat digunakan pada Spark.

### 2.13.2 Arsitektur Spark

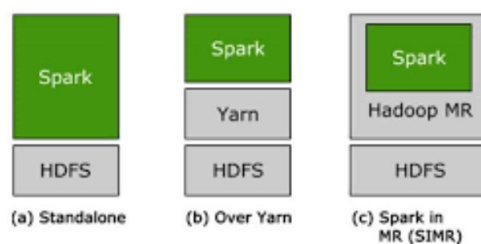


Gambar 2.11: Arsitektur Spark

Berdasarkan Gambar 2.11, berikut adalah beberapa hal penting terkait arsitektur Spark:

- *Driver Program* bertugas untuk menjalankan *Object main* pada *master node*. *Driver Program* adalah tempat dimana *Spark Context* dibuat.
- *Spark Context* bertugas untuk menghubungkan pengguna dengan *cluster*. *Spark Context* juga digunakan untuk membuat RDD, *accumulator*, dan *broadcast variable*.
- *Cluster Manager* bertugas untuk mengatur sumber daya pada sebuah *cluster*.
- *Executor* membantu memantau proses-proses yang berjalan pada *worker node* dan bertanggung jawab untuk mengerjakan *task* yang diberikan.
- *Task* adalah satuan kerja pada Spark yang berisi perintah-perintah fungsi yang diserialisasi.

### 2.13.3 Jenis Instalasi pada Spark



Gambar 2.12: Arsitektur Spark

Berdasarkan Gambar 2.12, berikut adalah jenis-jenis instalasi pada Spark:

- *Standalone*  
Spark berdiri diatas HDFS Hadoop. Spark memungkinkan untuk mengakses data pada HDFS Hadoop untuk membaca input dan menulis output.
- *Hadoop Yarn*  
Spark dapat berjalan pada Hadoop Yarn tanpa memerlukan instalasi atau meminta hak akses *root* apapun. Hadoop Yarn membantu integrasi Spark pada ekosistem Hadoop.
- *Spark In MapReduce (SIMR)*  
SIMR digunakan untuk menjalankan pekerjaan Spark secara independen. Jenis instalasi ini sudah tidak lagi berlaku untuk Spark versi 2.0

### 2.13.4 Resilient Distibuted Datasets (RDD)

RDD adalah kumpulan partisi terdistribusi yang disimpan dalam memori atau *disk* pada beberapa *cluster*. RDD tersebar menjadi beberapa partisi, sehingga partisi tersebut dapat disimpan dan diproses pada komputer yang berbeda.

Berikut adalah beberapa karakteristik yang dimiliki RDD:

- *Lazy evaluation*: operasi pada Spark hanya akan dilakukan ketika memanggil fungsi *Action*.
- *Immutability*: data yang disimpan dalam RDD tidak dapat diubah nilainya.
- *In-memory computation*: RDD menyimpan data secara langsung dalam memori.
- *Partitioning*: dapat membagi pekerjaan RDD pada beberapa komputer.

Berikut adalah jenis operasi pada RDD:

- Fungsi *Transformation*

Fungsi *transformation* dilakukan secara *lazy*, sehingga hanya akan dikerjakan apabila dipanggil pada fungsi *action*. Fungsi *transformation* pada RDD akan dijelaskan pada tabel dibawah ini.

Fungsi	Deskripsi
map()	Mengembalikan RDD baru dengan menerapkan fungsi pada setiap elemen data
filter()	Mengembalikan RDD baru yang dibentuk dengan memilih elemen-elemen sumber di mana fungsi mengembalikan true
reduceByKey()	Menggabungkan nilai-nilai kunci menggunakan fungsi

- Fungsi *Action*

Fungsi *Action* adalah operasi yang mengembalikan nilai output ke dalam terminal atau melakukan penulisan data pada sistem penyimpanan eksternal. Fungsi *Action* memaksa evaluasi pada RDD yang akan dipanggil, untuk menghasilkan output. Fungsi *Action* pada RDD akan dijelaskan pada tabel dibawah ini.

Fungsi	Deskripsi
count()	Mendapat jumlah elemen data dalam RDD
reduce()	Agregat elemen data ke dalam RDD dengan mengambil dua argumen dan mengembalikan satu
foreach(operation)	Menjalankan operasi untuk setiap elemen data dalam RDD

### 2.13.5 Dataframe

*Dataframe* adalah kumpulan data yang didistribusikan, disusun dalam baris dan kolom. Setiap kolom dalam *Dataframe* memiliki nama dan tipe terkait. *Dataframe* mirip dengan tabel database tradisional, yang terstruktur dan ringkas. Dengan menggunakan *Dataframe*, kueri SQL dapat dengan mudah diimplementasi pada *big data*.

Berikut adalah beberapa karakteristik yang dimiliki *Dataframe*:

- Terdiri atas baris dan kolom seperti tabel.
- Memiliki skema untuk penyimpanan data
- Data yang dapat disimpan berupa numerik dan kategorikal.
- Dapat dilakukan pemrosesan kueri SQL.

### 2.13.6 Komponen Spark

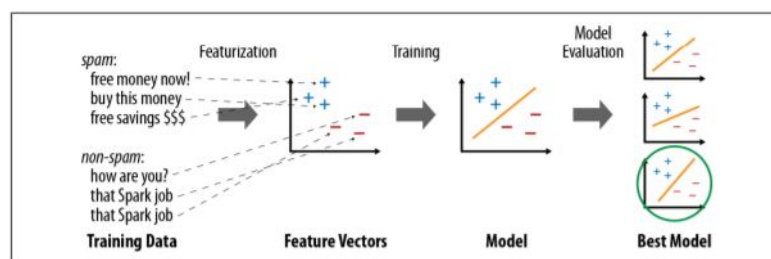
Komponen Spark adalah library tambahan pada Spark untuk melakukan proses komputasi pada lingkungan big data berdasarkan jenis-jenis kebutuhan pengolahan data. Berikut adalah penjelasan singkat mengenai komponen pada Spark:

- **Spark Core**  
Spark Core adalah *library* Spark yang berisi fungsionalitas dasar Spark, termasuk komponen untuk penjadwalan tugas, manajemen memori, pemulihan kesalahan, dan berinteraksi dengan sistem penyimpanan. Spark Core menyediakan komputasi pada memori, fungsi *action* dan *transformation* untuk mengolah RDD.
- **Spark SQL**  
Spark SQL memungkinkan pemrosesan kueri SQL pada lingkungan big data. Spark SQL menyediakan fungsi untuk menghitung nilai statistik dasar seperti *mean*, *median*, *modus*, nilai maksimum dan nilai minimum.
- **Spark Streaming**  
Spark Streaming adalah salah satu komponen Apache Spark, yang memungkinkan Spark dapat memproses data *streaming* secara *real-time*. Spark Streaming menyediakan API untuk memanipulasi aliran data yang cocok dengan RDD. Hal ini memungkinkan analisis data untuk beralih melalui sumber aplikasi yang memberikan data secara *real-time*.
- **Spark MLlib**  
Spark MLlib adalah *library* Spark yang berisi fungsionalitas yang umum digunakan pada *machine learning*. Untuk mengimplementasikan teknik *data mining* pada lingkungan *big data* dibutuhkan *library* Spark MLlib. Spark MLlib menyediakan berbagai jenis algoritma *machine learning* termasuk klasifikasi dan pengelompokan/*clustering*.

## 2.14 Spark MLlib

Spark MLlib adalah library pembelajaran mesin berdasarkan komputasi secara paralel. MLlib terdiri dari algoritma pembelajaran umum seperti klasifikasi, pengelompokan/*clustering*. Secara garis besar, MLlib melakukan data *preprocessing*, pelatihan model, dan membuat prediksi.

### 2.14.1 Machine Learning pada Spark MLlib



Gambar 2.13: Tahapan Pembelajaran Machine Learning

*Machine learning* bertujuan membuat prediksi label/kelompok data berdasarkan jenis model yang dipakai. Pemodelan *machine learning* mencakup model dari *data mining*. Pemodelan *machine learning* membutuhkan input berupa vektor fitur. Vektor fitur adalah nilai masing-masing atribut yang digunakan pada pelatihan data.

Gambar 2.13 adalah tahapan *machine learning* pada Spark MLlib, berikut adalah penjelasan singkat dari masing-masing tahapan:

1. *Featurization*

Pemodelan *machine learning* hanya dapat menerima input berupa vektor. Oleh karena itu, nilai atribut pada tabel akan diubah ke representasi numerik dalam bentuk vektor.

2. *Training*

Pemodelan *machine learning* melakukan pelatihan agar model yang dipakai memberikan hasil yang tepat untuk menentukan label atau kelompok data. Oleh karena itu, pemodelan *machine learning* memerlukan pelatihan model beberapa kali untuk mendapatkan model terbaik.

3. *Model Evaluation*

Pada akhir pelatihan, model yang terbentuk dapat diputuskan baik atau tidak melalui perhitungan nilai akurasi. Semakin besar nilai akurasi, maka model dapat digunakan untuk memprediksi nilai label atau kelompok data secara tepat.

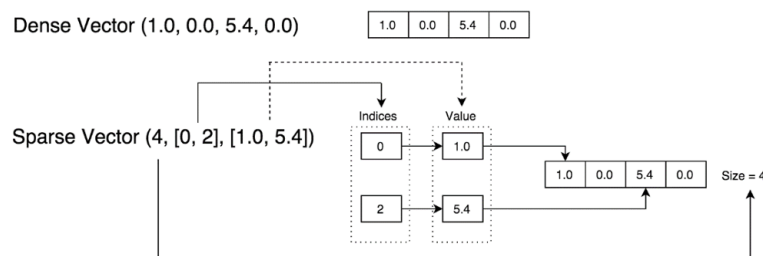
### 2.14.2 Tipe Data pada Spark MLlib

Seperti yang sudah dijelaskan pada bagian 2.14.1, pemodelan *machine learning* menerima input berupa vektor fitur. Tipe data yang disediakan pada Spark MLlib terdiri dari beberapa jenis yaitu vektor, *labeledpoint*, dan *various model class*.

Berikut adalah beberapa jenis tipe data pada Spark MLlib:

- Vektor

Vektor terdiri dari dua jenis yaitu vektor dense dan vektor sparse. Kelas vektor berada pada *package* `mllib.linalg.Vectors`. Gambar 2.14 adalah contoh vektor dense dan vektor sparse:



Gambar 2.14: Contoh Vektor Dense dan Sparse

- Vektor *dense*

Vektor *dense* adalah vektor yang menyimpan setiap nilai fitur dataset. Jumlah elemen pada vektor *dense* akan memiliki jumlah yang sama dengan jumlah fitur pada dataset.

- Vektor *sparse*

Vektor *sparse* adalah vektor yang menyimpan setiap nilai fitur yang bukan nol pada dataset, sehingga jumlah elemen yang disimpan pada vektor *sparse* lebih sedikit dibandingkan dengan jumlah elemen yang disimpan pada vektor *dense*.

- *LabeledPoint*

*LabeledPoint* digunakan pada algoritma *supervised learning* yaitu klasifikasi dan regresi. Kelas *LabeledPoint* terletak pada *package* `mllib.regress`.

- *Various Model class*

*Various Model classes* adalah tipe data yang dihasilkan dari pemodelan *machine learning*. Tipe data ini memiliki fungsi `predict()` untuk melakukan prediksi label dan kelompok data.



### 2.14.3 *Data Mining* pada Spark MLlib

Data mining pada Spark MLlib menggunakan tahapan pemodelan pada *machine learning* yang dijelaskan pada bagian 2.14.1 untuk menghasilkan tabel hasil pengelompokan dan klasifikasi. Pada bagian ini akan dijelaskan parameter dari pemodelan Spark MLlib.

#### *Naive Bayes*

*Naive Bayes* menjadi pemodelan klasifikasi yang umum digunakan. *Naive Bayes* dapat dilatih dengan sangat efisien karena prosesnya hanya menghitung probabilitas bersyarat. *Naive Bayes* memiliki parameter masukan sebagai berikut:

- *randomSplit* adalah membagi *training* dan *test* data berdasarkan persentase.
- *setModelType* adalah memilih model yang tersedia (*multinomial/bernoulli*)
- *setLabelCol* adalah memilih jenis atribut yang menjadi label kelas.

#### *K-Means*

*K-means* menjadi pemodelan pengelompokan/*clustering* yang paling umum digunakan untuk mengelompokkan titik-titik data menjadi sejumlah kelompok yang telah ditentukan. *K-means* memiliki parameter masukan sebagai berikut:

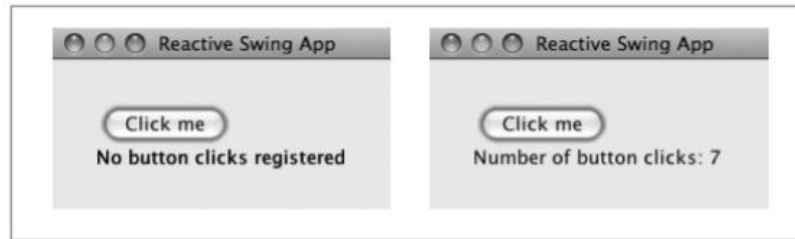
- *k* adalah jumlah cluster yang diinginkan.
- *maxIterations* adalah jumlah iterasi maksimum yang harus dijalankan.
- *initializationMode* menentukan inisialisasi centroid secara acak.
- *initializationSteps* menentukan jumlah langkah dalam algoritma *k-means*.
- *initialModel* adalah menentukan nilai centroid saat inisialisasi.

## 2.15 Scala

Scala adalah bahasa pemrograman berbasis open source, dibuat oleh Profesor Martin Odersky. Scala adalah bahasa pemrograman multi-paradigma dan mendukung paradigma fungsional serta berorientasi objek. Untuk pengembangan Spark, penulisan sintaks Scala dianggap produktif untuk mengimplementasikan kode program. Pemrograman pada Scala mempertahankan prinsip keseimbangan antara produktivitas pengembangan program dan kinerja program. Pemrograman pada Scala tidak serumit pemrograman pada Java. Satu baris kode program pada Scala dapat menggantikan 20 hingga 25 baris kode Java. Karena alasan terbut, Scala menjadi bahasa pemrograman yang sangat diminati untuk melakukan pemrosesan *big data* pada Spark.

## 2.16 Scala Swing

Scala Swing adalah program berbasis *Graphical User Interface* (GUI) sehingga memiliki perbedaan dengan program Spark yang dieksekusi dengan terminal. Scala Swing bertujuan untuk memberi tampilan program sehingga hasil program diharapkan menjadi lebih interaktif. Scala menyediakan akses langsung terhadap kelas GUI pada Java menggunakan *library* Scala Swing. Dengan menggunakan Scala, penggunaan Scala Swing dapat memenuhi kebutuhan perancangan *User Interface* melalui berbagai macam komponen GUI pada umumnya. Gambar 2.15 adalah contoh implementasi GUI sederhana pada Scala Swing.



Gambar 2.15: GUI Sederhana pada Scala Swing

### 2.16.1 Panel dan Layout

*Panel* adalah tempat untuk menampilkan semua komponen GUI dengan beberapa aturan tata letak yang harus dipenuhi. Salah satu bagian tersulit pada perancangan aplikasi berbasis GUI adalah mengatur penempatan layout dengan benar. *Layout* terdiri dari beberapa komponen GUI seperti *Frame*, *Panel*, *Label* atau *Button*. Masing-masing komponen GUI pada *layout* memiliki nilai properti sendiri (warna, ukuran, posisi) yang dapat diatur secara manual.

### 2.16.2 Handling Event

*Handling event* adalah pekerjaan yang dilakukan masing-masing komponen. Komponen akan menerima aksi langsung dari pengguna aplikasi. Mekanisme ini dikenal sebagai *handling event*, yang dieksekusi ketika suatu peristiwa terjadi. *Handling event* memiliki *listener*. *Listener* adalah sebuah komponen memberi tahu sebuah aksi kepada komponen tertentu. *Listener* harus dibuat untuk masing-masing objek *handling event*.

## 2.17 Format Penyimpanan Data

Spark dapat melakukan aksi membaca dan menulis pada data terstruktur dan semi terstruktur. Contoh data terstruktur yang umum digunakan adalah CSV, sedangkan contoh data semi terstruktur yang umum digunakan adalah JSON. Berikut adalah penjelasan lengkap mengenai format penyimpanan data CSV dan JSON.

### 2.17.1 CSV

CSV (*Comma Separated Values*) menjadi format yang sangat umum digunakan untuk menyimpan nilai pada tabel data yang terstruktur. CSV menggunakan format ekstensi (.csv) saat berdiri sendiri. Hasil penyimpanan dengan format CSV umum digunakan untuk menyimpan data saat ingin menyimpan tabel dari basis data. CSV memisahkan nilai atribut yang satu dengan yang lainnya menggunakan tanda koma. CSV dapat memisahkan data yang satu dengan data lainnya berdasarkan penempatan data pada baris yang berbeda. Listing 3.1 adalah contoh format penyimpanan CSV.

Listing 2.1: Format Penyimpanan CSV

```
age,workclass,zip,education,year_of_education,marital_status,occupation
39,State-gov,77516,Bachelors,13,Never-married,Adm-clerical
50,Self-emp-not-inc,83311,Bachelors,13,Married-civ-spouse,Exec-managerial
38,Private,215646,HS-grad,9,Divorced,Handlers-cleaners
53,Private,234721,11th,7,Married-civ-spouse,Handlers-cleaners
28,Private,338409,Bachelors,13,Married-civ-spouse,Prof-specialty
37,Private,284582,Masters,14,Married-civ-spouse,Exec-managerial
49,Private,160187,9th,5,Married-spouse-absent,Other-service
52,Self-emp-not-inc,209642,HS-grad,9,Married-civ-spouse,Exec-managerial
```

### 2.17.2 JSON

JSON (*JavaScript Object Notation*) adalah format untuk pertukaran data. JSON menggunakan format ekstensi (.json) saat berdiri sendiri. JSON diturunkan dari bahasa pemrograman JavaScript. Walaupun diturunkan dari bahasa pemrograman lain, JSON tidak bergantung pada bahasa pemrograman apapun. Oleh karena itu, format JSON sangat mudah dipakai untuk pertukaran data antar bahasa pemrograman. JSON memiliki format penyimpanan *key-value* seperti pada Listing 2.2. JSON menyimpan enam jenis tipe data yaitu *string*, *number*, *object*, *array*, *boolean*, *null*. Menulis format JSON dalam beberapa baris akan lebih mudah dibaca terutama saat datanya sudah banyak.

Listing 2.2: Format Penyimpanan JSON

```
{
  "firstName": "Rack",
  "lastName": "Jackon",
  "gender": "man",
  "age": 24,
  "address": {
    "streetAddress": "126",
    "city": "San Jone",
    "state": "CA",
    "postalCode": "394221"
  },
  "phoneNumbers": [
    { "type": "home", "number": "7383627627" }
  ]
}
```



## BAB 3

### ANALISIS

Pada bab ini akan dijelaskan analisis masalah penelitian ini. Analisis ini meliputi analisis masalah, eksplorasi spark, studi kasus, dan gambaran umum perangkat lunak.

#### 3.1 Analisis Masalah

Berkembangnya penggunaan teknologi informasi menyebabkan data bertumbuh dengan sangat pesat. Istilah data yang memiliki ukuran yang besar dikenal sebagai *big data*. *Data mining* adalah cara untuk mengekstraksi sebuah informasi dari sekumpulan data untuk mendukung pengambilan keputusan atau pernyataan tertentu. Hasil *data mining* yang mengandung data individu apabila disebarkan kepada pihak lain untuk kebutuhan tertentu tanpa dilakukan perlindungan privasi terlebih dahulu maka dapat melanggar hak privasi seseorang. Apabila informasi pribadi seseorang dapat diketahui oleh orang lain, maka mengakibatkan munculnya tindak kejahatan yang mengatasnamakan privasi orang bersangkutan. Oleh karena itu, perlu adanya sebuah cara untuk melindungi privasi seseorang sebelum dilakukan distribusi data.

Solusi yang tepat untuk menjamin perlindungan data sebelum dilakukan distribusi data adalah anonimisasi. Anonimisasi bertujuan untuk menyamarkan sebagian nilai atribut data yang unik terhadap atribut data lain, khususnya untuk atribut yang termasuk dalam kategori atribut privasi menurut PII. *Privacy-preserving data mining* adalah sebuah cara untuk melindungi data sebelum dilakukan data mining agar privasi dari hasil data mining dapat terlindungi. *K-anonymity* adalah salah satu metode agar *privacy-preserving data mining* dapat dicapai dengan menyamarkan beberapa nilai atribut data. Tujuan utama dari penelitian ini adalah mempelajari, menganalisis, melakukan eksperimen, membuat perangkat lunak terkait anonimisasi pada lingkungan big data, dan menguji hasilnya agar privasi data dapat terjaga. Berikut beberapa kajian yang akan dianalisis terkait teknik anonimisasi pada lingkungan *big data*.

##### 3.1.1 Dataset Eksperimen

Dataset yang dipakai adalah *Adult*. Dataset ini diperoleh dari website Kaggle. Dataset ini disimpan dalam format CSV seperti penjelasan pada bagian 2.17. Format CSV memisahkan nilai atribut data melalui simbol koma. Dataset *Adult* dipilih, karena pernah digunakan sebelumnya untuk eksperimen algoritma *k-anonymity*. Dataset ini berisi sampel sensus penduduk di Amerika Serikat pada tahun 1990. Penelitian ini melibatkan 10 juta baris data dengan ukuran data sebesar 1.2 GB.

Listing 3.1: Dataset Adult

```
age,workclass,zip,education,year_of_education,marital_status,occupation
39,State-gov,77516,Bachelors,13,Never-married,Adm-clerical
50,Self-emp-not-inc,83311,Bachelors,13,Married-civ-spouse,Exec-managerial
38,Private,215646,HS-grad,9,Divorced,Handlers-cleaners
53,Private,234721,11th,7,Married-civ-spouse,Handlers-cleaners
28,Private,338409,Bachelors,13,Married-civ-spouse,Prof-specialty
```

Berikut adalah kemungkinan nilai untuk masing-masing jenis atribut dalam dataset:

- Age: numerik
- Workclass: *Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.*
- Education: *Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.*
- Years of education: numerik
- Marital status: *Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, MarriedAF-spouse*
- Occupation: *Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspect, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, ArmedForces.*
- Relationship: *Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried*
- Race: *White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black*
- Sex: *Male, Female*
- Capital gain: numerik
- Capital loss: numerik
- Hours per week: numerik
- Native country: *United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US, India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad and Tobago, Peru, Hong, HollandNetherlands*
- Income:  $\leq 50K$ ,  $> 50K$

### 3.1.2 Personally Identifiable Information

Pada bagian 2.1, telah dijelaskan mengenai konsep *Personally Identifiable Information* (PII). PII digunakan untuk mengelompokkan nilai atribut berdasarkan kategori atribut yang digunakan pada proses anonimisasi data. Berdasarkan bagian 2.5, atribut pada proses anonimisasi dapat dikategorikan sebagai *identifier*, *quasi-identifier*, dan *sensitive attribute*.

Atribut *identifier* adalah atribut yang dapat mengidentifikasi individu secara langsung. Contoh dari atribut *identifier* pada dataset *Adult* adalah nama, tempat tanggal lahir, alamat rumah, nomor KTP. Atribut *quasi-identifier* adalah atribut yang dapat mengidentifikasi seseorang apabila nilai sebuah atribut digabung dengan nilai atribut lain pada baris data yang sama. Contoh *quasi-identifier* pada dataset *Adult* adalah *age*, *zip*, *education*, *years of education*, *occupation*, *race*, *sex*, *native country*. *Sensitive attribute* adalah nilai yang ingin dirahasiakan. Contoh *sensitive attribute* pada dataset *Adult* adalah *workclass*, *marital status*, *relationship*, *income*.

Atribut *identifier* nantinya akan dihilangkan sebelum dilakukan proses anonimisasi, karena nilai dari atribut *identifier* dapat langsung mengidentifikasi seseorang. Sedangkan *sensitive attribute* nilainya tidak akan dihapus karena akan melalui proses anonimisasi bersamaan dengan nilai dari *quasi-identifier* sehingga *sensitive attribute* milik individu tidak dapat dibedakan satu sama lain pada hasil tabel akhir anonimisasi sehingga keamanan distribusi data terjaga.

### 3.1.3 Perhitungan *Distance* dan *Information Loss*

Pada bagian 2.9, telah dijelaskan konsep mengenai penggunaan *distance* dan *information loss*. *Distance* dan *Information Loss* digunakan oleh algoritma *Greedy k-member clustering* untuk mencari kelompok data terbaik sehingga menghasilkan pengelompokan data yang tepat.

#### *Distance*

*Distance* bertujuan untuk menentukan hasil pengelompokan data pada algoritma *Greedy k-member clustering*. Pemilihan *distance* yang baik dapat mencapai hasil klasifikasi yang lebih optimal.

Akan diambil 2 sampel data dari dataset *Adult* sebagai berikut:

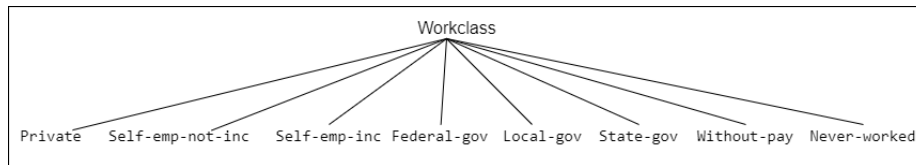
1. 39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
2. 50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 13, United-States, <=50K

*Distance* atribut numerik dapat dihitung sebagai berikut berdasarkan umur data pertama ( $v_1$ )= 39, umur data kedua ( $v_2$ )= 50, dan jumlah data ( $D$ )= 10.000.000 data.

$$\delta_n(v_1, v_2) = \frac{|v_1 - v_2|}{|D|} = \frac{|39 - 50|}{10.000.000} = \frac{11}{10.000.000} = 0.0000011$$

*Distance* atribut kategorikal dapat dihitung sebagai berikut berdasarkan *workclass* data pertama ( $v_1$ )= *State-gov*, *workclass* data kedua ( $v_2$ )= *Self-emp-not-inc*, jumlah subtree ( $H(\Lambda(v_i, v_j))$ )= 1, dan tinggi taxonomy tree ( $H(T_D)$ )= 1 seperti pada Gambar 3.1.

$$\delta_C(v_1, v_2) = \frac{H(\Lambda(v_i, v_j))}{H(T_D)} = \frac{1}{1} = 1$$



Gambar 3.1: Taxonomy Tree (Workclass)

#### *Information Loss*

*Information Loss* (IL) bertujuan untuk mengevaluasi seberapa baik kinerja algoritma *k-anonymity* terhadap nilai informasi sebuah data. Tabel 3.1 adalah contoh hasil pengelompokan data pada dataset *Adult*:

Tabel 3.1: Tabel Hasil Clustering Data pada Cluster 1

Age	Workclass	Education	Occupation	Sex	Income	Cluster Name
39	State-gov	Bachelors	Adm-clerical	Male	<=50K	Cluster 1
50	Self-emp-not-inc	Bachelors	Exec-managerial	Male	<=50K	Cluster 1
38	Private	HS-grad	Handlers-cleaners	Male	<=50K	Cluster 1
53	Private	11th	Handlers-cleaners	Male	<=50K	Cluster 1
28	Private	Bachelors	Prof-specialty	Female	<=50K	Cluster 1

*Information Loss* (IL) dapat dihitung sebagai berikut berdasarkan atribut numerik yaitu jumlah anggota  $cluster$  ( $e$ ) = 5,  $MAX_{Age}$  = 53,  $MIN_{Age}$  = 28,  $N_{Age}$  = 5 mencakup atribut  $Age$  dan atribut kategorikal yaitu  $H(\Lambda(\cup_{C_j}))$  = 1,  $H(T_{C_j})$  = 1.

$$\begin{aligned} D(e) &= \sum_{i=1}^m \frac{(MAX_{N_i} - MIN_{N_i})}{|N_i|} + \sum_{j=1}^n \frac{H(\Lambda(\cup_{C_j}))}{H(T_{C_j})} \\ &= \frac{(53 - 28)}{5} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} = 10 \end{aligned}$$

$$\begin{aligned} IL(e) &= |e| \cdot D(e) \\ &= 5 \cdot 10 = 50 \end{aligned}$$

Total *Information Loss* dihitung dari jumlah *Information Loss* masing-masing *cluster*.

$$\begin{aligned} Total - IL(AT) &= \sum_{e \in \epsilon} IL(e) \\ &= IL(cluster1) + IL(cluster2) + \dots + IL(clusterN) \end{aligned}$$

### 3.1.4 Greedy K-Member Clustering

Algoritma *Greedy k-member clustering* telah dijelaskan pada bagian 2.8. Algoritma ini bertujuan untuk membagi seluruh data pada tabel terhadap masing-masing *cluster* untuk kompleksitas yang lebih baik dan mendukung nilai utilitas informasi yang lebih baik dibandingkan algoritma *clustering* lain. Pada bagian ini, akan dilakukan eksperimen sederhana untuk mencari tahu langkah kerja algoritma *Greedy k-member clustering* secara konseptual.

Melalui sampel data pada Tabel 3.2, akan diputuskan nilai dari setiap atribut anonimisasi. Jenis atribut anonimisasi yang pertama adalah Quasi-identifikasi, dengan nilai  $QI = \{Age, Education, Occupation, Sex, Income\}$ . Jenis atribut anonimisasi yang kedua adalah Sensitive Attribute, dengan nilai  $SA = \{Workclass\}$ . Jika telah diketahui tabel data seperti diatas,  $k = 2$ , dan jumlah cluster ( $m$ ) = 2, maka algoritma ini siap ditelusuri lebih lanjut.

Tabel 3.2: Dataset Adults

ID	Age	Workclass	Education	Occupation	Sex	Income
t1	39	State-gov	Bachelors	Adm-clerical	Male	<=50K
t2	50	Self-emp-not-inc	Bachelors	Exec-managerial	Male	<=50K
t3	38	Private	HS-grad	Handlers-cleaners	Male	<=50K
t4	53	Private	11th	Handlers-cleaners	Male	<=50K
t5	28	Private	Bachelors	Prof-specialty	Female	<=50K

Berikut adalah tahapan yang terjadi pada algoritma *Greedy k-member clustering*:

1. Nilai awal result =  $\emptyset$ ,  $r = \{t1\}$ ,  $|S| = 5$ ,  $k = 2$
2. Karena kondisi  $|S| \geq k$  terpenuhi, maka dilakukan perulangan sebagai berikut:
  - (a) Nilai  $r$  diubah menjadi  $r = \{t3\}$ , karena terbukti data  $t3$  memiliki  $\Delta(t1, t3) = 1.7189$  yang paling tinggi dari seluruh *distance* lain. Berikut adalah contoh perhitungannya:

$$\Delta(t_1, t_2) = 1.715$$

$$\Delta(t_1, t_3) = 2.431$$

$$\Delta(t_1, t_4) = 2.122$$

$$\Delta(t_1, t_5) = 1.621$$



- (b) Nilai awal  $S = \{t1, t2, t4, t5\}$
- (c) Nilai awal  $c = \{t3\}$ ,  $|c| = 1$
- (d) Karena kondisi  $|c| < k$  terpenuhi, maka dilakukan perulangan sebagai berikut:
  - i. Nilai  $r$  diubah menjadi  $r = \{t3, t4\}$ , karena terbukti data  $t4$  memiliki  $IL(t3 \cup t4) = 0.330$  yang paling rendah dari seluruh data lain. Berikut adalah contoh perhitungannya:

$$IL(t3 \cup t1) = 0.479$$

$$IL(t3 \cup t2) = 0.515$$

$$IL(t3 \cup t4) = 0.330$$

$$IL(t3 \cup t5) = 0.367$$

- ii. Nilai  $S$  diubah menjadi  $S = \{t1, t2, t5\}$ ,  $|S| = 4$
- iii. Nilai  $c$  ditambahkan menjadi  $c = \{t3, t4\}$ ,  $|c| = 2$
- (e) Karena kondisi  $|c| < k$  sudah tidak terpenuhi lagi, maka perulangan ini akan berhenti
- (f) Nilai *result* akan ditambahkan menjadi  $result = \{t3, t4\}$
- (g) Karena kondisi  $|S| \geq k$  masih terpenuhi, maka perulangan akan tetap berlanjut sampai pada kondisi dimana  $|S| < k$  sehingga hasil akhirnya adalah  $result = \{\{t3, t4\}, \{t2, t5\}\}$ ,  $S = \{t1\}$ ,  $|S| = 1$

3. Karena kondisi  $S \neq 0$  terpenuhi, maka dilakukan perulangan sebagai berikut:

- (a) Nilai  $r$  diubah menjadi  $r = \{t1\}$
- (b) Nilai  $S$  diubah menjadi  $S = \{\phi\}$ ,  $|S| = 0$
- (c) Nilai  $c$  diubah menjadi  $c = \{t3, t4\}$  karena terbukti *cluster*  $c$  memiliki  $IL(\{t3, t4\} \cup t1) = 0.279$  yang paling rendah dari seluruh *cluster* lain. Berikut adalah contoh perhitungannya:

$$IL(\{t3, t4\} \cup t1) = 0.279$$

$$IL(\{t2, t5\} \cup t1) = 0.515$$

- (d) Nilai  $c$  ditambahkan menjadi  $c = \{t1, t3, t4\}$
  - (e) Nilai  $c$  pada perulangan ini tidak akan ditambahkan pada *result*, karena telah ditetapkan  $k = 2$  sedangkan jumlah datanya ganjil, sehingga sisa data tersebut tidak akan dicatat pada variabel *result* agar menjaga masing-masing *cluster* hanya memiliki 2 anggota saja.
  - (f) Karena kondisi  $S \neq 0$  sudah tidak terpenuhi lagi, maka perulangan ini akan berhenti.
4. Hasil akhirnya adalah  $result = \{\{t3, t4\}, \{t2, t5\}\}$  dikembalikan sebagai output untuk algoritma *Greedy k-member clustering* seperti pada Tabel 3.3 sebagai berikut:

Tabel 3.3: Tabel Hasil Greedy K-Member Clustering

ID	Age	Workclass	Education	Occupation	Sex	Income
t3	38	Private	HS-grad	Handlers-cleaners	Male	<=50K
t4	53	Private	11th	Handlers-cleaners	Male	<=50K
t2	50	Self-emp-not-inc	Bachelors	Exec-managerial	Male	<=50K
t5	28	Private	Bachelors	Prof-specialty	Female	<=50K

### 3.1.5 Domain Generalization Hierarchy

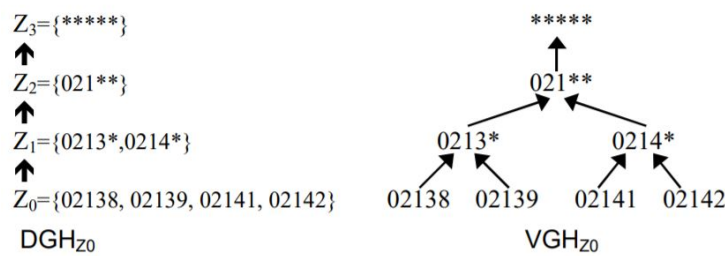
Pada bagian 2.7, telah dijelaskan konsep mengenai *Hierarchy Based Generalization*. DGH adalah contoh penerapan dari *Hierarchy Based Generalization*. DGH bertujuan untuk melindungi data dengan cara menerapkan metode generalisasi terhadap nilai atribut data yang bersifat unik, agar menjadi nilai yang lebih umum. Berikut adalah penerapan DGH terhadap dataset *Adult*.

Diketahui kemungkinan nilai unik atribut pada dataset *Adult* sebagai berikut:

- Age = {33,36,38,40,42,43,46,49}
- ZIP = {77516,77517,77526,77527}
- Sex = {Male,Female}

Nilai atribut ZIP, akan dibangun tiga jenis domain sebagai berikut:

- Domain dengan nilai kurang spesifik  
Domain ini dipilih apabila tujuannya adalah lebih mengutamakan hasil informasi yang diperoleh dengan cara melakukan sedikit anonimisasi pada nilai data. Contohnya atribut ZIP akan diubah menjadi {7751\*,7752\*} apabila satu digit terakhir memiliki nilai yang berbeda dan digit sisanya memiliki nilai yang sama.
- Domain dengan nilai yang lebih umum  
Domain ini dipilih apabila tujuannya adalah menyeimbangkan nilai informasi yang diperoleh dengan tingkat perlindungan data yang didapat dengan cara meningkatkan level anonimisasi nilai data. Contohnya atribut ZIP akan diubah menjadi {775\*\*} apabila kedua digit terakhir memiliki nilai yang berbeda dan digit sisanya memiliki nilai yang sama.
- Domain dengan nilai yang umum. Domain ini dipilih apabila tujuannya adalah mengutamakan perlindungan data. Biasanya jenis domain ini jarang dipilih, karena hasil anonimisasinya tidak dapat digunakan untuk proses data mining (memiliki nilai akurasi yang rendah apabila dilakukan pemodelan *data mining*). Contohnya atribut ZIP akan diubah menjadi {\*\*\*\*\*}



Gambar 3.2: DGH dan VGH pada atribut ZIP

Nilai atribut *Age* akan dibangun berdasarkan ketentuan berikut:

- Nilai atribut *Age* akan diubah menjadi rentang nilai. Contohnya nilai 33 diubah menjadi [30-39], karena 33 termasuk pada rentang nilai tersebut.

Nilai atribut *Age* dan *Sex* akan dibangun berdasarkan ketentuan berikut:

- Nilai atribut *Sex* termasuk nilai kategorikal, sehingga akan diubah menjadi nilai yang lebih umum. Contohnya nilai *Male/Female* diubah menjadi *Person* (bentuk umum).

### 3.1.6 *K-Anonymity*

Pada bagian 2.5, dijelaskan konsep anonimisasi. *K-anonymity* bertujuan untuk menyamarkan nilai dari masing *quasi-identifier* yang unik pada kelompok *cluster* yang sama. Kata kuncinya adalah nilai unik pada kelompok *cluster* yang sama. Setelah dataset dilakukan anonimisasi, maka data privasi sudah terlindungi sehingga publikasi data dapat dilakukan dengan aman. Tabel 3.4 adalah kelompok data yang dihasilkan oleh algoritma *Greedy k-member clustering*.

Tabel 3.4: Tabel Hasil Clustering

ID	Age	Workclass	Education	ZIP	Sex	Hours/week	Cluster Name
t3	32	Private	HS-grad	77516	Male	30	Cluster 1
t4	32	Private	11th	77541	Female	30	Cluster 1
t2	34	Self-emp-not-inc	Bachelors	77526	Male	34	Cluster 2
t5	50	Private	Bachelors	77526	Male	37	Cluster 2
t1	47	Local-gov	Bachelors	77581	Male	54	Cluster 3
t6	50	Federal-gov	HS-grad	77532	Male	57	Cluster 3

Diketahui bentuk generalisasi berdasarkan *Domain Generalization Hierarchy* sebagai berikut:

$$\begin{aligned}
 \text{Age} &= \{[20 - 30], [40 - 50]\} \\
 \text{ZIP} &= \{775 **\} \\
 \text{Sex} &= \{Person\} \\
 \text{Hours/week} &= \{[12 - 18], [33 - 37], [53 - 61]\}
 \end{aligned}$$

Berikut adalah tahapan proses anonimisasi dengan model *k-anonymity*:

1. Diketahui *quasi-identifier* sebagai berikut  $QI = \{Age, ZIP, Sex, Hours/week\}$  dan *sensitive attribute* sebagai berikut  $SA = \{Workclass, Education\}$
2. Mencari nilai *quasi-identifier* yang unik pada kelompok *cluster* yang sama. Sebagai contoh, *cluster 2* memiliki nilai *quasi-identifier* yang unik sebagai berikut  $QI = \{Age, Hours/week\}$
3. Melakukan generalisasi DGH pada nilai *quasi-identifier* yang unik menjadi bentuk. Sebagai contoh,  $QI = \{Age, Hours/week\}$  memiliki nilai yang unik, sehingga diubah menjadi  $Age = \{[40-50]\}$ ,  $Hours/week = \{[33-37]\}$
4. *Sensitive attribute* tidak akan dilakukan generalisasi, karena *quasi-identifier* sudah dilakukan generalisasi sehingga seseorang akan sulit untuk menebak kepemilikan dari *sensitive attribute*.
5. Ulangi hal yang sama pada langkah sebelumnya untuk setiap *cluster*. Hasil akhir dari proses anonimisasi ada pada Tabel 3.5 sebagai berikut:

Tabel 3.5: Tabel Hasil Anonimisasi

ID	Age	Workclass	Education	ZIP	Sex	Hours/week	Cluster Name
t3	32	Private	HS-grad	775**	Person	30	Cluster 1
t4	32	Private	11th	775**	Person	30	Cluster 1
t2	[40-50]	Self-emp-not-inc	Bachelors	77526	Male	[33-37]	Cluster 2
t5	[40-50]	Private	Bachelors	77526	Male	[33-37]	Cluster 2
t1	[40-50]	Local-gov	Bachelors	775**	Male	[53-61]	Cluster 3
t6	[40-50]	Federal-gov	HS-grad	775**	Male	[53-61]	Cluster 3

## 3.2 Eksplorasi Spark

Pada bagian ini akan dilakukan penelusuran lebih lanjut mengenai beberapa hal penting terkait Spark sebelum melakukan eksperimen metode anonimisasi pada Spark.

Berikut adalah beberapa hal penting terkait Spark:

- Spark bekerja sama dengan komponen lain seperti JDK, SBT, HDFS sehingga instalasi Spark untuk masing-masing sistem operasi dapat berbeda. Pada penelitian ini, akan dilakukan instalasi Spark melalui sistem operasi Windows.
- Spark dapat bekerja dengan bahasa pemrograman Scala. Scala dipilih karena memiliki efektivitas yang baik pada penulisan kode program. Scala dapat menyederhanakan perintah pada Spark menjadi baris yang lebih sedikit.
- Program Spark dijalankan dengan cara membuat jar sebelum perintah eksekusi dijalankan. Hal ini menghambat pekerjaan pada tahap implementasi perangkat lunak. Intel IJ adalah sebuah *Integrated Development Environment* (IDE) yang memfasilitasi pemrograman Scala pada Spark dan menampilkan hasil pemrosesan Spark secara langsung.
- Spark menyediakan konfigurasi untuk mengatur jumlah resource yang dibutuhkan (jumlah pemakaian RAM, *core* CPU) pada pemrosesan data. Konfigurasi ini bertujuan agar Spark dapat mengolah data yang besar secara maksimal dengan menggunakan jumlah *resource* yang tersedia. Konfigurasi ini ditulis pada perintah eksekusi Spark.

### 3.2.1 Instalasi Spark

Spark berjalan pada sistem operasi Windows, Linux, dan Mac OS. Spark dapat dijalankan secara lokal menggunakan satu komputer, meskipun Spark tetap membutuhkan beberapa komputer untuk pemrosesan data yang besar. Jenis instalasi Spark dijelaskan pada bagian 2.13.3. Pada penelitian ini digunakan jenis instalasi Standalone untuk Spark versi 2.4.5 pada sistem operasi Windows. Sebelum melakukan instalasi Spark, ada beberapa hal yang harus diperhatikan dan dipenuhi.

Berikut adalah beberapa hal yang harus diperhatikan:

- Java 7, Python 2.6 telah dihilangkan pada implementasi Spark 2.2.0 ke atas.
- Scala 2.10 sudah usang apabila dipakai pada Spark 2.4.1 ke atas.
- Hadoop 2.6.5 telah dihilangkan pada implementasi Spark 2.2.0 ke atas.

Berikut adalah beberapa hal yang harus dipenuhi:

- Spark 2.4.5 dapat berjalan di Java 8, Python 2.7+/3.4+ dan R 3.1+
- Spark 2.4.5 dapat menggunakan Scala 2.12
- Spark 2.4.5 dapat menggunakan Hadoop 2.7

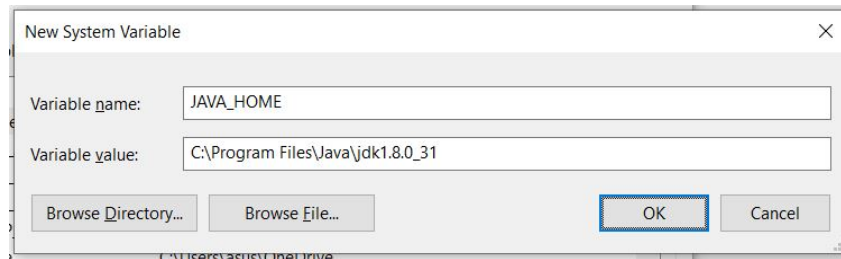
Berikut adalah tahapan instalasi Spark 2.4.5 secara umum:

1. Melakukan instalasi Java 8.
2. Melakukan instalasi Spark 2.4.5
3. Melakukan instalasi IntelliJ untuk Scala sbt.

## Instalasi Java 8

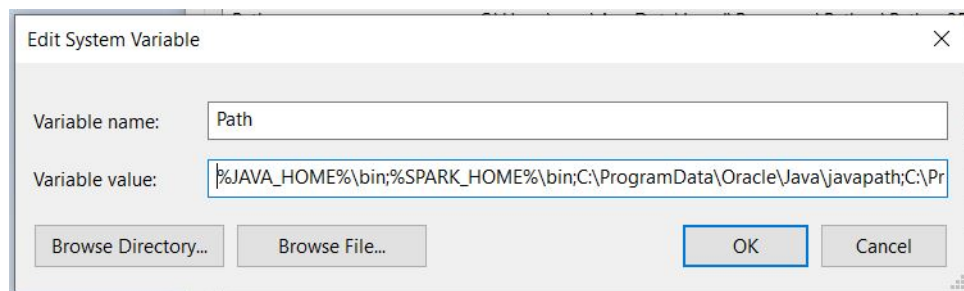
Berikut adalah tahapan instalasi Java 8 secara lengkap:

1. Download Java SE Development Kit 8u31 pada link berikut <https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html>
2. Lakukan instalasi Java SE Development Kit 8u31 seperti biasa.
3. Pilih menu *Edit the system environment variables*.
4. Buat *environment variables* baru seperti Gambar 3.3.



Gambar 3.3: Environment Variables

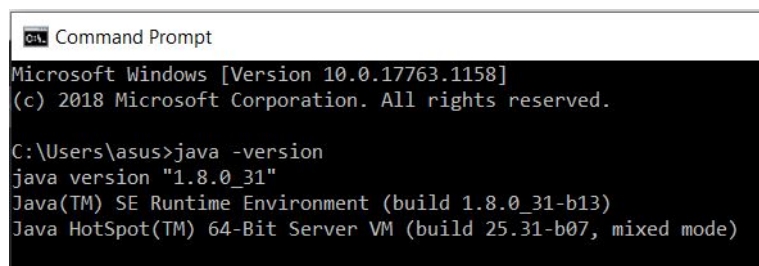
5. Tambahkan `%JAVA_HOME%\bin;` pada Path di System variables seperti Gambar 3.7.



Gambar 3.4: Penambahan Variable Value

Berikut adalah tahapan verifikasi terhadap instalasi Java 8:

1. Pilih menu *command prompt*.
2. Jalankan perintah `java -version` pada Command Prompt.



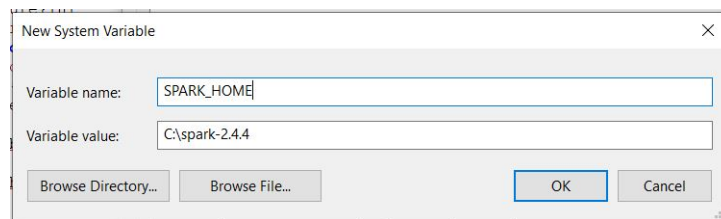
Gambar 3.5: Perintah `java -version`

3. Apabila sistem tidak menampilkan pesan error, maka Java 8 sudah terpasang dengan baik.

### Instalasi Spark 2.4.5

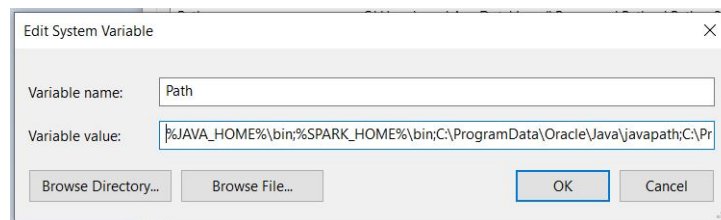
Berikut adalah tahapan instalasi Spark 2.4.5 secara lengkap:

1. Download winutils.exe dari link <https://github.com/steveloughran/winutils/tree/master/hadoop-2.7.1/bin>, tempatkan winutils.exe pada C:\winutils\bin
2. Download Spark 2.4.5 dari link <https://downloads.apache.org/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-hadoop2.7.tgz>
3. Buat folder sebagai berikut C:\spark-2.4.4 dan ekstraksi file spark-2.4.5-bin-hadoop2.7.tgz di dalam folder tersebut.
4. Buat *environment variables* baru seperti Gambar 3.6.



Gambar 3.6: Environment Variable

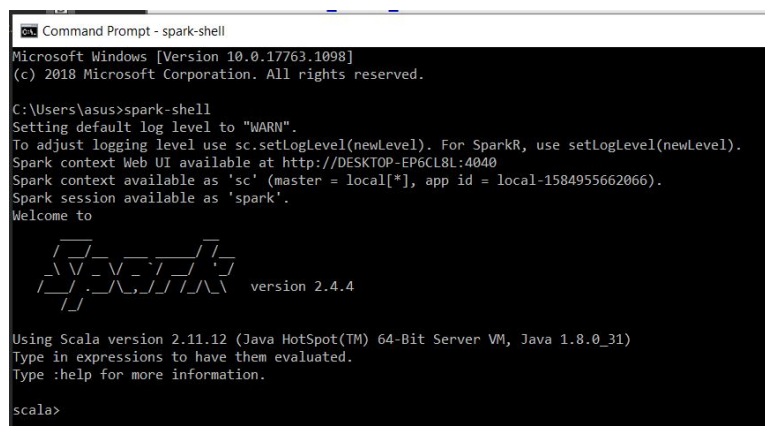
5. Tambahkan %SPARK\_HOME%\bin; pada Path di System variables seperti Gambar 3.7



Gambar 3.7: Penambahan Variable Value

Berikut adalah tahapan verifikasi terhadap instalasi Spark 2.4.5:

1. Jalankan perintah **spark-shell** pada *command prompt*.
2. Apabila terminal menampilkan tampilan seperti pada Gambar 3.8, artinya Spark 2.4.5 sudah dapat berjalan dengan baik pada komputer tersebut.

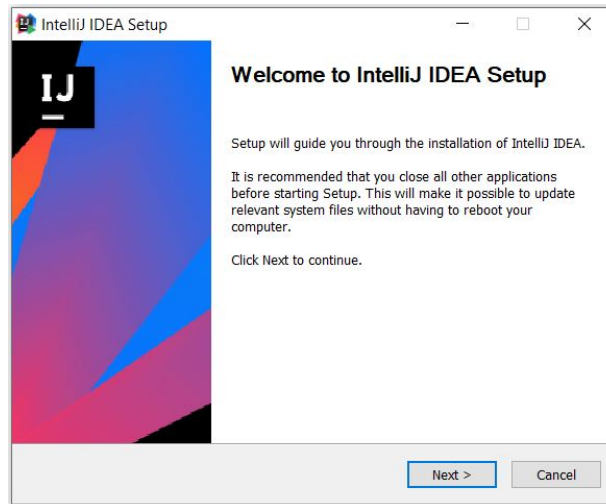


Gambar 3.8: Spark 2.4.5

## Instalasi IntelliJ untuk Scala SBT

Berikut adalah tahapan instalasi IntelliJ:

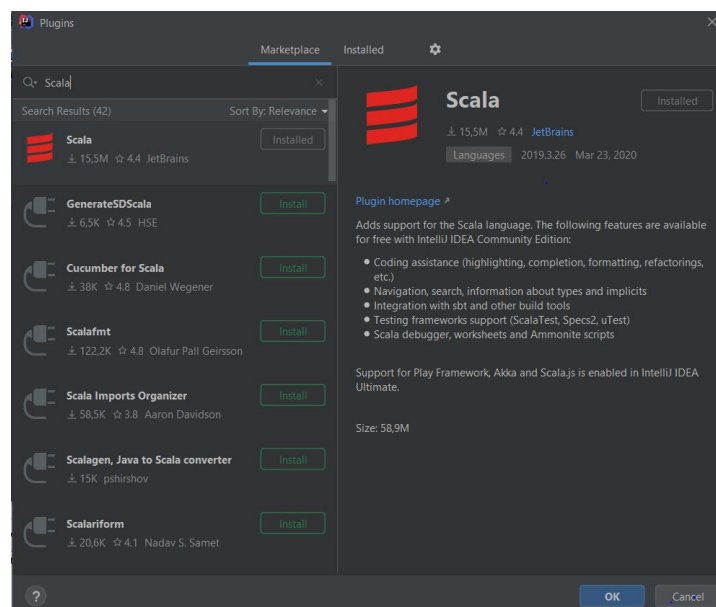
1. Download IntelliJ melalui link berikut  
<https://www.jetbrains.com/idea/download/#section=windows>
2. Lakukan instalasi IntelliJ seperti biasa.



Gambar 3.9: Instalasi IntelliJ

Berikut adalah tahapan pemasangan *plugin* Scala pada IntelliJ.

1. Pilih menu *Configure* pada IntelliJ, lalu pilih menu *Plugins*.
2. Telusuri *plugin* Scala pada kolom pencarian seperti Gambar 3.10.



Gambar 3.10: Plugins Scala

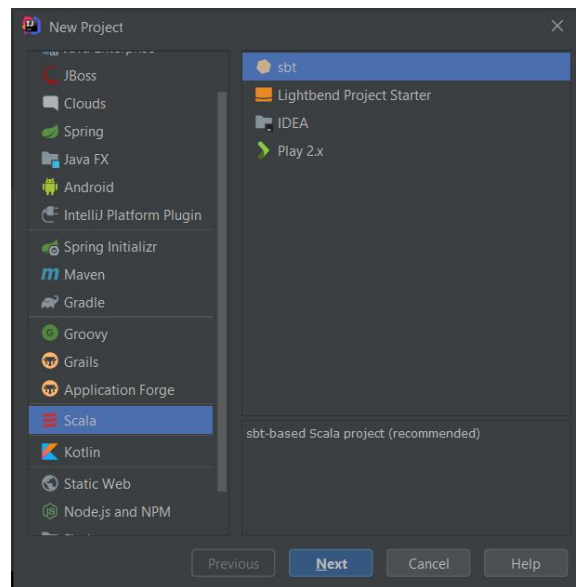
3. Klik tombol *install*

### 3.2.2 Membuat *Project* Spark pada IntelliJ

Untuk membuat program Spark, pertama-tam perlu membuat project Spark baru untuk merancang kelas-kelas yang dibutuhkan pada eksekusi Spark. Beberapa hal yang perlu diperhatikan adalah menggunakan versi Scala sbt, memilih versi sbt 1.3.9, memilih versi Scala 2.11.12, dan melakukan *import libraryDependencies* Spark sesuai kebutuhan.

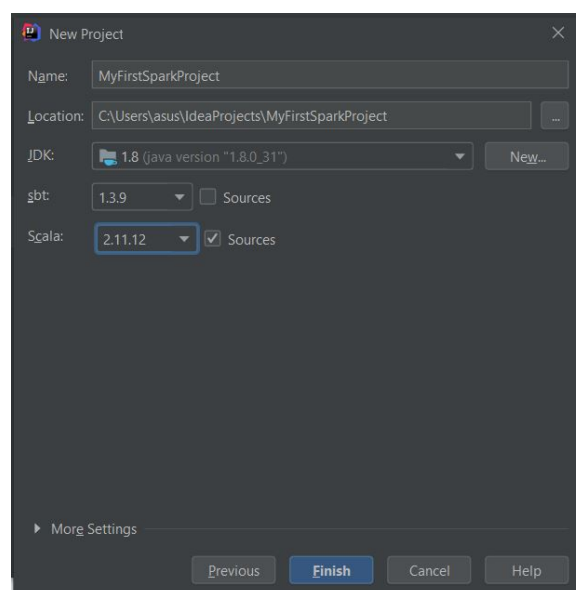
Berikut adalah tahapan pembuatan *project* Spark pada IntelliJ:

1. Memilih menu *Create New Project*
2. Menggunakan bahasa pemrograman Scala berbasis sbt seperti Gambar 3.11.



Gambar 3.11: Memilih Bahasa Scala Berbasis sbt

3. Melakukan konfigurasi pada project Spark baru seperti Gambar 3.12.



Gambar 3.12: Melakukan Konfigurasi Project Spark

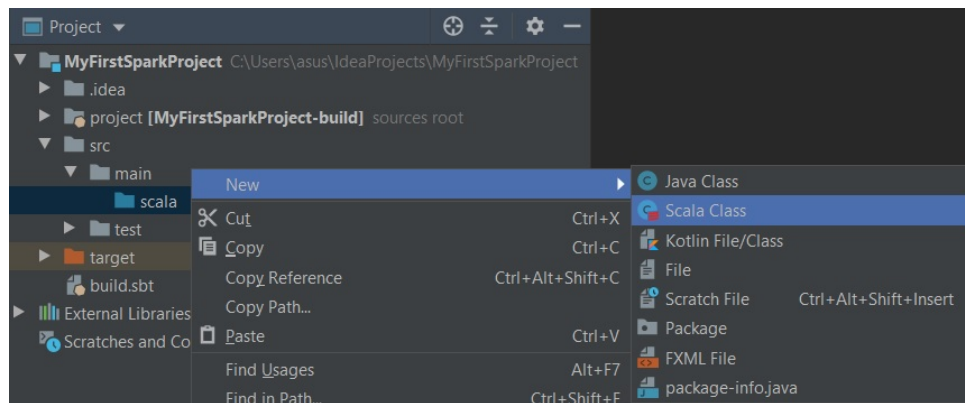


4. Listing 3.2 adalah perintah *import libraryDependencies* pada file `build.sbt`  
Contoh: `spark-core`, `spark-sql`, `spark-mllib`.

Listing 3.2: Melakukan Import Library Spark

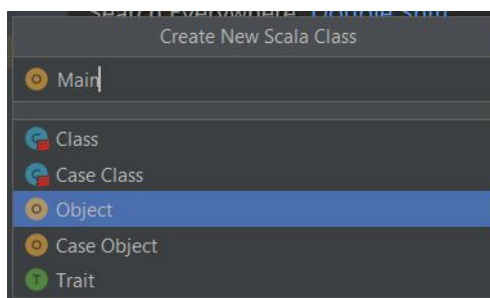
```
name := "NamaProject"
version := "0.1"
scalaVersion := "2.11.12"
// https://mvnrepository.com/artifact/org.apache.spark/spark-core
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
// https://mvnrepository.com/artifact/org.apache.spark/spark-sql
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.0"
// https://mvnrepository.com/artifact/org.apache.spark/spark-mllib
libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.4.3"
```

5. Menambahkan *Scala class* pada `src/main/scala` seperti Gambar 3.13.



Gambar 3.13: Menambahkan Scala Class pada Project Spark

6. Memilih tipe *Scala class* sebagai *Object* seperti Gambar 3.14.



Gambar 3.14: Memilih Tipe Object pada Scala Class

7. Listing 3.3 adalah perintah untuk menambahkan *main method* pada *Scala class*.

Listing 3.3: Menambahkan Main method pada Scala Class

```
object Main {
  def main(args: Array[String]) {
    //statement
  }
}
```

### 3.2.3 Membuat *File JAR* pada Command Prompt

Sebelum menjalankan program Spark pada Hadoop Cluster, program Spark yang sudah jadi harus dibuat menjadi file JAR terlebih dahulu. Hal ini disebabkan karena perintah Spark hanya menerima input berupa kode program dalam format (.jar).

Berikut adalah tahapan untuk membuat File JAR:

1. Membuka folder pengerjaan *project* Spark `\IdeaProjects\NamaProject`
2. Membuka command prompt pada folder *project*.
3. Mengeksekusi perintah `sbt package` pada command prompt
4. Menunggu proses pembuatan file JAR oleh sistem, apabila terminal tidak menampilkan pesan *error* maka file JAR telah berhasil dibuat dan tersimpan pada folder tertentu.
5. File JAR yang telah dibuat akan tersimpan pada `NamaProject\target\scala-2.11`

### 3.2.4 Menjalankan Program Spark pada Komputer Lokal

Apabila ukuran data input eksperimen kecil, maka program Spark dapat dijalankan pada komputer lokal menggunakan perintah dari Command Prompt. Waktu komputasi pada komputer lokal akan jauh lebih lama dibandingkan dijalankan pada server Hadoop Cluster.

Berikut adalah tahapan menjalankan program Spark pada komputer lokal:

1. Membuka *command prompt* pada komputer lokal.
2. Menjalankan perintah eksekusi Spark sebagai berikut `spark-submit --class NamaMainClass --master local[*] lokasi_jar\nama_jar.jar` pada command prompt
3. Menunggu proses eksekusi file JAR oleh komputer lokal, apabila terminal tidak menampilkan pesan error maka program Spark berhasil dijalankan dengan baik.
4. Output yang dihasilkan oleh program Spark akan ditampilkan pada terminal *command prompt*.

### 3.2.5 Menjalankan Program Spark pada Hadoop Cluster

Karena ukuran data input eksperimen terbilang besar yaitu mencapai 1GB, maka akan lebih efektif apabila komputasi dilakukan secara paralel melalui Hadoop cluster. Hadoop cluster terdiri dari beberapa perangkat komputer yang dapat saling bekerja sama, sehingga proses komputasi dapat dilakukan lebih cepat.

Berikut adalah tahapan menjalankan program Spark pada Hadoop cluster:

1. Membuka *command prompt* pada komputer lokal.
2. Menyambungkan jaringan komputer lokal dengan server Hadoop cluster menggunakan perintah `ssh hduser@10.100.69.101` pada command prompt.
3. Melakukan *upload file* JAR dari komputer lokal ke folder Hadoop cluster menggunakan perintah `scp nama_jar.jar hduser@10.100.69.101:nama_folder` pada command prompt.
4. Menjalankan perintah eksekusi Spark sebagai berikut `spark-submit --class NamaMainClass --master yarn lokasi_jar\nama_jar.jar` pada command prompt
5. Menunggu proses eksekusi *file* JAR oleh Hadoop cluster, apabila terminal tidak menampilkan pesan error maka program Spark berhasil dijalankan dengan baik.

## 3.3 Studi Kasus

Untuk memahami implementasi algoritma anonimisasi pada Spark, maka dilakukan studi kasus terhadap fungsi Spark yang umum digunakan, seperti fungsi dasar pada Spark, fungsi dasar pada komponen Spark, dan fungsi dasar pada Spark MLlib. Bentuk dari studi kasus yang akan dilakukan adalah memberikan contoh kode program berikut penjelasan singkat mengenai tujuan pemanggilan fungsi, parameter input, dan contoh output yang dikeluarkan oleh fungsi tersebut.

### 3.3.1 Eksperimen Scala

Pada bagian 2.15 telah dijelaskan tujuan dari penggunaan bahasa Scala. Scala digunakan pada penelitian ini karena sintaks yang sederhana untuk mengimplementasi beberapa baris kode pada bahasa pemrograman Java. Berikut adalah beberapa contoh eksperimen yang dilakukan pada bahasa Scala.

#### Menentukan Jenis Variabel pada Scala

Scala memiliki dua jenis variabel yaitu *immutable* variabel dan *mutable* variabel. *Immutable* variabel adalah variabel yang nilainya tidak dapat diubah, sedangkan *mutable* variabel adalah variabel yang nilainya dapat diubah. Implementasi *immutable* dan *mutable* memiliki implementasi sintaks yang berbeda. *Immutable* variabel menggunakan sintaks `val`, sedangkan *mutable* variabel menggunakan sintaks `var`. Kode program dapat dilihat pada Listing 3.4 mengenai jenis variabel pada Scala.

Listing 3.4: Menentukan Jenis Variabel pada Scala

```
// Immutable Variabel
val donutsToBuy: Int = 5
donutsToBuy = 10

// Mutable Variabel
var favoriteDonut: String = "Glazed Donut"
favoriteDonut = "Vanilla Donut"
```

#### Menentukan Jenis Tipe Data pada Scala

Scala memiliki jenis tipe data yang mirip dengan tipe data pada bahasa pemrograman Java. Scala dapat menangani tipe data *Int*, *Long*, *Short*, *Double*, *Float*, *String*, *Byte*, *Char* dan *Unit*. Kode program dapat dilihat pada Listing 3.5 mengenai jenis tipe data pada Scala.

Listing 3.5: Menentukan Jenis Tipe Data pada Scala

```
val donutsBought: Int = 5
val bigNumberOfDonuts: Long = 100000000
val smallNumberOfDonuts: Short = 1
val priceOfDonut: Double = 2.50
val donutPrice: Float = 2.50f
val donutStoreName: String = "allaboutscala Donut Store"
val donutByte: Byte = 0xa
val donutFirstLetter: Char = 'D'
val nothing: Unit = ()
```

## Menentukan Struktur Data pada Scala

Scala memiliki dua jenis struktur data yaitu *immutable* dan *mutable collection*. *Immutable collection* adalah struktur data yang nilainya tidak dapat diubah, sedangkan *mutable collection* adalah struktur data yang nilainya dapat diubah. Implementasi *immutable* dan *mutable collection* memiliki jenis struktur data yang berbeda satu sama lain. Kode program dapat dilihat pada Listing 3.6 mengenai *immutable collection* pada Scala dan Listing 3.7 mengenai *mutable collection* pada Scala.

Listing 3.6: Membuat immutable collection pada Scala

```
// List
val list1: List[String] = List("Plain Donut","Strawberry Donut","Chocolate Donut")
println(s"Elements of list1 = $list1")

// Map
val map1: Map[String, String] = Map(("PD","Plain Donut"),("SD","Strawberry Donut"),("CD","Chocolate Donut"))
println(s"Elements of map1 = $map1")
```

Listing 3.7: Membuat mutable collection pada Scala

```
// Array
val array1: Array[String] = Array("Plain Donut","Strawberry Donut","Chocolate Donut")
println(s"Elements of array1 = ${array1.mkString(", ")")

// Map
val map1: Map[String, String] = Map(("PD","Plain Donut"),("SD","Strawberry Donut"),("CD","Chocolate Donut"))
println(s"Elements of map1 = $map1")
```

## Membuat Kelas pada Scala

Kelas pada Scala memiliki fungsi kelas yang sama pada Java yaitu untuk menyimpan variabel dan method. Kode program dapat dilihat pada Listing 3.8 mengenai cara membuat kelas pada Scala.

Listing 3.8: Membuat Kelas Object pada Scala

```
class AreaOfRectangle
{
    var length = 20; // Variables
    var height = 40;

    def area() // Method which gives the area of the rectangle
    {
        var ar = length * height;
        println("Area of the rectangle is :" + ar);
    }
}
```

### Membuat *Singleton Object* pada Scala

Scala tidak memiliki variabel statik seperti pada Java, sehingga fungsinya digantikan oleh *singleton object*. *Singleton object* adalah objek yang mendefinisikan method main dari kelas-kelas pada Scala. Kode program dapat dilihat pada Listing 3.9 mengenai cara membuat *singleton object* pada Scala.

Listing 3.9: Membuat Kelas Object pada Scala

```
object Main
{
    def main(args: Array[String])
    {
        // Creating object of AreaOfRectangle class
        var obj = new AreaOfRectangle();
        obj.area();
    }
}
```

### Membuat Fungsi Sederhana pada Scala

Scala menggunakan fungsi untuk menempatkan kode program berdasarkan tujuan masing-masing. Perlu diperhatikan bahwa hasil akhir dari fungsi langsung dikembalikan tanpa memanggil perintah *return*, seperti pada Java. Kode program dapat dilihat pada Listing 3.10 mengenai pembuatan fungsi pada Scala.

Listing 3.10: Membuat Fungsi Sederhana pada Scala

```
def calculateDonutCost(donutName: String, quantity: Int): Double = {
    println(s"Calculating cost for $donutName, quantity = $quantity")

    // make some calculations ...
    2.50 * quantity
}
```

### Membuat Fungsi Percabangan

Scala memiliki jenis implementasi percabangan yang sama dengan Java. Percabangan digunakan untuk melakukan eksekusi pada baris *statement* yang sesuai berdasarkan kondisi tertentu. Kode program dapat dilihat pada Listing 3.11 mengenai percabangan pada Scala.

Listing 3.11: Membuat Fungsi Percabangan pada Scala

```
# If-Else statement
if(numberOfPeople > 10) {
    println(s"Number of donuts to buy = ${numberOfPeople * donutsPerPerson}")
}
else if (numberOfPeople == 0) {
    println("Number of people is zero.")
    println("No need to buy donuts.")
}
else {
    println(s"Number of donuts to buy = $defaultDonutsToBuy")
}
```

### Membuat Fungsi Perulangan pada Scala

Scala memiliki jenis implementasi perulangan yang sama dengan Java. Perulangan digunakan untuk mengulangi eksekusi pada baris statement yang sama berdasarkan kondisi tertentu. Kode program dapat dilihat pada Listing 3.12 mengenai perulangan pada Scala.

Listing 3.12: Membuat Fungsi Perulangan pada Scala

```
# For loop
for(numberOfDonuts <- 1 to 5){
  println(s"Number of donuts to buy = $numberOfDonuts")
}

# While loop
while (numberOfDonutsToBake > 0) {
  println(s"Remaining donuts to be baked = $numberOfDonutsToBake")
  numberOfDonutsToBake -= 1
}

# Do-while loop
do {
  numberOfDonutsBaked += 1
  println(s"Number of donuts baked = $numberOfDonutsBaked")
}
while (numberOfDonutsBaked < 5)
```

### 3.3.2 Eksperimen Spark

Spark adalah teknologi yang digunakan untuk mengolah *big data* berdasarkan konsep dari bagian 2.13. Spark membagi satu pekerjaan pada masing-masing *Worker Node* seperti pada bagian 2.13.2. Oleh karena itu Spark memecah data partisi data agar data yang besar dapat distribusikan ke masing-masing komputer. Berikut adalah beberapa fungsi dasar Spark untuk mengolah partisi data.

#### Melakukan Konfigurasi Spark

Berikut adalah tahapan konfigurasi Spark pada Main Class:

- Membuat objek SparkConf untuk inisialisasi project Spark
- Menetapkan jumlah *core* CPU yang bekerja pada perintah setMaster()
- Menetapkan nama program Spark pada perintah setAppName()
- Membuat objek SparkContext untuk membuat RDD.

Listing 3.13: Konfigurasi Spark

```
val conf = new SparkConf()
conf.setMaster("local[2]")
conf.setAppName("Tutorial Spark")
val sc = new SparkContext(conf)
```

## Membuat RDD

Pada bagian 2.13.4, sudah dijelaskan konsep RDD. Pada bagian ini, akan dilakukan eksperimen mengenai jenis-jenis cara untuk membuat RDD pada Spark.

Berikut adalah beberapa cara untuk membuat RDD:

- Membaca data eksternal pada Spark sebagai RDD
- Membuat RDD dari struktur data *List*
- Merubah Dataframe menjadi RDD

Listing 3.14: Cara Pembuatan RDD

```
# 1. Membaca data eksternal pada Spark sebagai RD
rdd = sc.textFile("path")

# 2. Membuat RDD dari struktur data list
rdd = sc.parallelize(["id","name","3","5"])

# 3. Merubah Dataframe menjadi RDD
rdd = df.rdd
```

## Membuat Dataframe

Pada bagian 2.13.5, sudah dijelaskan konsep *DataFrame*. Pada bagian ini, akan dilakukan eksperimen mengenai jenis-jenis cara untuk membuat *DataFrame* pada Spark.

Berikut adalah beberapa cara untuk membuat *DataFrame*:

- Membaca data eksternal sebagai *DataFrame*
- Mengubah RDD menjadi *DataFrame* dengan nama kolom
- Mengubah RDD menjadi *DataFrame* dengan skema

Listing 3.15: Cara Pembuatan Dataframe

```
# 1. Membaca data eksternal sebagai Dataframe
# header and schema are optional
df = sqlContext.read.csv("path", header = True/False, schema=df_schema)

# 2.1 Mengubah RDD menjadi Dataframe dengan nama kolom
df = spark.createDataFrame(rdd,["name","age"])

# 2.2 Mengubah RDD menjadi Dataframe dengan skema
from pyspark.sql.types import *
df_schema = StructType([
...     StructField("name", StringType(), True),
...     StructField("age", IntegerType(), True)])
df = spark.createDataFrame(rdd,df_schema)
```

### Memanggil Fungsi *Transformation*

Pada bagian ?? dijelaskan jenis-jenis fungsi *Transformation* pada RDD. Listing 3.16 adalah contoh penerapan jenis-jenis fungsi *Transformation* pada Spark, berikut adalah penjelasan singkat dari masing-masing fungsi:

- `select()`: menampilkan isi RDD berdasarkan nama variabel yang menyimpan RDD.
- `filter()/where()`: menyeleksi isi RDD berdasarkan kondisi tertentu
- `sort()/orderBy()`: mengurutkan isi RDD berdasarkan keterurutan atribut tertentu
- `groupBy()` dan `agg()`: mengelompokkan isi RDD dan melakukan agregasi.
- `join()`: menggabungkan dua RDD yang berbeda berdasarkan kesamaan nilai atribut.

Listing 3.16: Contoh Fungsi Transformation

```
# 1. select
df.select(df.name)
df.select("name")

# 2. filter/where
df.filter(df.age>20)
df.filter("age>20")
df.where("age>20")
df.where(df.age>20)

# 3. sort/orderBy
df.sort("age",ascending=False)
df.orderBy(df.age.desc())

# 4. groupBy dan agg
df.groupBy("gender").agg(count("name"),avg("age"))

# 5. join
df1.join(df.2, (df1.x1 == df2.x1) & (df1.x2 == df2.x2),'left')
```

### Memanggil Fungsi *Action*

Pada bagian ?? dijelaskan jenis-jenis fungsi *Action* pada RDD. Listing 3.17 adalah contoh penerapan jenis-jenis fungsi *Action* pada Spark, berikut adalah penjelasan singkat dari masing-masing fungsi:

- `show()`: menampilkan n baris pertama dari *DataFrame* atau RDD
- `take()`: menampilkan beberapa baris dari *DataFrame* atau RDD
- `collect()`: mengumpulkan seluruh data dari *DataFrame* atau RDD
- `count()`: menghitung jumlah baris
- `printSchema()`: menampilkan nama kolom dan tipe data



Listing 3.17: Contoh Fungsi Action

```
# 1. show()
df.show(5)

# 2. take()
df.take(5)

# 3. collect()
df.collect()

# 4. count()
df.count()

# 6. printSchema()
df.printSchema()

# 7. transformation, action
df1.filter("age>10").join(df2,df1.x==df2.y).sort("age").show()
```

### Memanggil Fungsi RDD

Listing 3.18 adalah contoh penerapan jenis-jenis fungsi RDD pada Spark, berikut adalah penjelasan singkat dari masing-masing fungsi RDD:

- `repartition(n)`: membagi RDD menjadi n buah partisi
- `cache()`: menyimpan RDD pada penyimpanan memori.
- `persist()`: menyimpan RDD pada penyimpanan memori atau disk.
- `unpersist()`: menghapus RDD pada memori atau disk.
- `foreach(println)`: melakukan print seluruh baris data pada RDD
- `saveAsTextFile(path)`: menyimpan RDD pada sebuah file

Listing 3.18: Contoh Fungsi RDD

```
rdd.repartition(4)
rdd.cache()
rdd.persist()
rdd.unpersist()
rdd.foreach(println)
rdd.saveAsTextFile(path)
```

### Membuat Variabel Global

Listing 3.19 adalah perintah untuk membuat variabel global pada Spark.

Listing 3.19: Membuat Variabel Global

```
val broadcastVar = sc.broadcast(Array(1, 2, 3))
```

### 3.3.3 Eksperimen Komponen Spark

Pada bagian 2.13.6 dijelaskan mengenai jenis-jenis komponen Spark dan tujuan penggunaannya. Pada bagian ini akan dilakukan eksperimen berdasarkan masing-masing jenis komponen Spark.

#### Spark Core

Berikut adalah langkah-langkah eksperimen dari Spark Core:

1. Listing 3.20 adalah membuat perintah SparkSession untuk inialisasi project Spark

Listing 3.20: Membuat SparkSession

```
val spark: SparkSession = SparkSession.builder()
    .master("local[3]")
    .appName("SparkCoreAdults")
    .getOrCreate()
```

2. Listing 3.21 adalah melihat dan mengatur partisi RDD berdasarkan data input CSV

Listing 3.21: Melihat dan Mengatur Partisi RDD

```
val sc = spark.sparkContext

val rdd: RDD[String] = sc.textFile("input/adult100k.csv")
println("initial partition count:" + rdd.getNumPartitions)

val reparRdd = rdd.repartition(4)
println("re-partition count:" + reparRdd.getNumPartitions)
```

3. Listing 3.22 adalah membuat jenis-jenis fungsi *transformation*.

Listing 3.22: Membuat Fungsi Transformation

```
//Transformation - flatMap
val rdd2 = rdd.flatMap(f => f.split(","))
rdd2.foreach(f => println(f))

//Transformation - map
val rdd3: RDD[(String, Int)] = rdd2.map(key => (key, 1))
rdd3.foreach(println)

//Transformation - filter
val rdd4 = rdd3.filter(a => a._1.startsWith("State-gov"))
rdd4.foreach(println)

//Transformation - reduceByKey
val rdd5 = rdd3.reduceByKey((x,y)=> x + y)
rdd5.foreach(println)

//Transformation - sortByKey
val rdd6 = rdd5.map(a => (a._2, a._1)).sortByKey()
```

4. Listing 3.23 adalah membuat jenis-jenis fungsi *action*.

Listing 3.23: Membuat Fungsi Action

```
//Action - count
println("Count : " + rdd6.count())

//Action - first
val firstRec = rdd6.first()
println("First Record : " + firstRec._1 + "," + firstRec._2)

//Action - max
val datMax = rdd6.max()
println("Max Record : " + datMax._1 + "," + datMax._2)

//Action - reduce
val totalWordCount = rdd6.reduce((a, b) => (a._1 + b._1, a._2))
println("dataReduce Record : " + totalWordCount._1)

//Action - take
val data3 = rdd6.take(3)
data3.foreach(f => {
    println("data3 Key:" + f._1 + ", Value:" + f._2)
})

//Action - collect
val data = rdd6.collect()
data.foreach(f => {
    println("Key:" + f._1 + ", Value:" + f._2)
})

//Action - saveAsTextFile
rdd5.saveAsTextFile("c:/tmp/wordCount")
```

## Spark SQL

Berikut adalah langkah-langkah eksperimen dari Spark Core:

1. Listing 3.24 adalah perintah untuk membuat SparkSession saat inisialisasi *project* Spark

Listing 3.24: Membuat Perintah SparkSession

```
val spark = SparkSession
    .builder.master("local[*]")
    .appName("SparkSQL")
    .getOrCreate()
```

2. Listing 3.25 adalah perintah untuk membuat *DataFrame* dari data input CSV.

Listing 3.25: Membuat Dataframe

```
val peopleDF = spark.read
    .format("csv")
    .option("header", "true")
    .option("delimiter", ",")
    .load("input/adult100k.csv")

peopleDF.printSchema()
```

3. Listing 3.26 adalah perintah untuk membuat tabel sementara, melakukan kueri, dan menyimpan hasil kueri pada file CSV.

Listing 3.26: Membuat Tabel Sementara

```
// Query
peopleDF.createTempView("tAdults")
val query = spark.sql(
    "SELECT workclass,count(education) as count_people"+
    "FROM tAdults " +
    "WHERE lower(education) == 'bachelors'" +
    "GROUP BY workclass " +
    "ORDER BY count_people DESC " +
    "LIMIT 10"
)
query.write.csv("C:/Users/asus/Desktop/resultquery1.csv")
```

4. Listing 3.27 adalah perintah untuk mencari nilai statistik seperti jumlah data, *mean*, standar deviasi, nilai minimum dan maksimum.

Listing 3.27: Mencari Nilai Statistik

```
// Statistika: count, mean, stddev, min, max
peopleDF.describe().show()
```

5. Listing 3.28 adalah perintah untuk mencari nilai *median*.

Listing 3.28: Mencari Nilai Median

```
// Median
val median = spark.sql(
    "SELECT percentile_approx(age, 0.5) as Median " +
    "FROM tAdults"
).show()
```

6. Listing 3.29 adalah perintah untuk mencari nilai *modus*.

Listing 3.29: Mencari Nilai Modus

```
// Modus
val modus = spark.sql(
    "SELECT age as Modus " +
    "FROM tAdults " +
    "GROUP BY age " +
    "ORDER BY COUNT(age) DESC " +
    "LIMIT 1"
).show()
```

## Spark MLlib

Berikut adalah langkah-langkah eksperimen dari Spark Core:

1. Listing 3.30 adalah perintah untuk membuat `SparkSession` saat inisialisasi *project* Spark

Listing 3.30: Membuat Perintah `SparkSession`

```
val spark = SparkSession
    .builder.master("local[*]")
    .appName("SparkMLlib")
    .getOrCreate()
```

2. Listing 3.31 adalah perintah untuk membuat skema untuk *DataFrame*.

Listing 3.31: Membuat Skema *Dataframe*

```
val schema = StructType(
    List(
        StructField("age", IntegerType, true),
        StructField("workclass", StringType, true),
        StructField("fnlwgt", IntegerType, true),
        StructField("education", StringType, true),
        StructField("education-num", IntegerType, true),
        StructField("marital-status", StringType, true),
        StructField("occupation", StringType, true),
        StructField("relationship", StringType, true),
        StructField("race", StringType, true),
        StructField("sex", StringType, true),
        StructField("capital-gain", IntegerType, true),
        StructField("capital-loss", IntegerType, true),
        StructField("hours-per-week", IntegerType, true),
        StructField("native-country", StringType, true),
        StructField("salary", StringType, true)
    )
)
```

3. Listing 3.32 adalah perintah untuk mengubah data input CSV menjadi *DataFrame* berdasarkan skema.

Listing 3.32: Mengubah CSV Menjadi Dataframe

```
val adult100k_df = spark.read
    .format("csv")
    .option("header", "false")
    .option("delimiter", ",")
    .schema(schema)
    .load("input/adult100k.csv")

adult100k_df.show()
```

4. Listing 3.33 adalah perintah untuk menggunakan fungsi *stringIndexer* untuk membuat kolom indeks dan fungsi *oneHotEncoder* untuk membuat kolom vektor.

Listing 3.33: Membuat Kolom Index

```
// Create vector based on stringIndexer and oneHotEncoder
val cols = adult100k_df.columns
val encodedFeatures = cols.flatMap{columnName =>
    val stringIndexer = new StringIndexer()
        .setInputCol(columnName)
        .setOutputCol(columnName + "_Index")
    val oneHotEncoder = new OneHotEncoderEstimator()
        .setInputCols(Array(columnName + "_Index"))
        .setOutputCols(Array(columnName + "_vec"))
        .setDropLast(false)
    Array(stringIndexer.setHandleInvalid("keep"), oneHotEncoder)
}
```

5. Listing 3.34 adalah perintah untuk menambahkan kolom vektor dan kolom indeks pada *DataFrame*.

Listing 3.34: Membuat Kolom Vektor

```
// Pipeline
val pipeline = new Pipeline().setStages(encodedFeatures)
val indexer_model = pipeline.fit(adult100k_df)
```

6. Listing 3.35 adalah perintah untuk memilih jenis implementasi vektor yang akan digunakan.

Listing 3.35: Memilih Jenis Vektor

```
// Sparse Vector
val df_transformed = indexer_model.transform(adult100k_df)
df_transformed.show()

// Dense Vector
val sparseToDense = udf((v: Vector) => v.toDense)
val df_denseVectors = df_transformed
    .withColumn("dense_workclass_vec",
        sparseToDense(df_transformed("workclass_vec")))
df_denseVectors.show()
```

7. Listing 3.36 adalah perintah untuk membuat vektor fitur dari setiap baris data pada *DataFrame*.

Listing 3.36: Membuat Vektor Fitur

```
// Final Result: Feature Vector
val vecFeatures = df_transformed.columns.filter(_.contains("vec"))
val vectorAssembler = new VectorAssembler()
    .setInputCols(vecFeatures)
    .setOutputCol("features")
val pipelineVectorAssembler = new Pipeline()
    .setStages(Array(vectorAssembler))
val result_df = pipelineVectorAssembler
    .fit(df_transformed)
    .transform(df_transformed)
result_df.show()
```

### 3.3.4 Eksperimen Spark MLIB

Pada bagian 2.14 telah dijelaskan mengenai konsep dan contoh pemodelan pada Spark MLlib. Pada penelitian ini akan digunakan pemodelan *Naive Bayes* untuk permasalahan klasifikasi pada bagian 2.2.2 dan *k-means* untuk permasalahan clustering pada bagian 2.2.4.

#### *Naive Bayes*

Pada bagian 2.14.3 menjelaskan parameter pemodelan *Naive Bayes* pada Spark MLlib. Berikut adalah tahapan eksperimen pada Listing 3.37 untuk pemodelan *Naive Bayes*:

1. Membagi data input CSV menjadi training data dan test data.
2. Melakukan pelatihan data pada pemodelan Naive Bayes.
3. Mengembalikan hasil klasifikasi dalam bentuk tabel.
4. Menghitung akurasi dari klasifikasi label kelas.

Listing 3.37: Eksperimen Naive Bayes Spark MLlib

```
// Split data into training (70%) and test (30%).
val Array(training, test) = result_df.randomSplit(Array(0.7, 0.3))

// Naive Bayes
val model = new NaiveBayes().setModelType("multinomial")
    .setLabelCol("workclass_Index").fit(training)

// Predict model
val predictions = model.transform(test)
predictions.show()

// Accuracy
val evaluator = new MulticlassClassificationEvaluator()
    .setLabelCol("workclass_Index")
    .setPredictionCol("prediction")
    .setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
```

Dataset yang dipakai untuk eksperimen *naive bayes* ini adalah sampel data dari dataset Adult yang sudah dijelaskan pada bagian. Data ini berjumlah 100.000 baris data dengan ukuran 11.000 KB. Dataset ini akan dibagi menjadi data test dan data training. Jumlah data test adalah 30.000 baris data, sedangkan jumlah data training adalah 70.000 data. Jumlah data training lebih banyak karena akan dipakai untuk pelatihan model *naive bayes*.

```

+---+-----+
| age|prediction|
+---+-----+
| null|      8.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
|  17|      3.0|
+---+-----+
only showing top 10 rows

```

Gambar 3.15: Hasil Naive Bayes Spark MLlib

Hasil pemodelan *naive bayes* dari eksperimen ini adalah label data. Cluster ini terbentuk berdasarkan distance terdekat antara anggota data dengan titik centroid pada cluster tersebut. Sehingga data-data yang tergabung pada sebuah cluster, sudah dipastikan bahwa data-data tersebut lebih dekat dengan titik centroid pada cluster tersebut dibanding dengan titik centroid pada cluster lainnya. Gambar 3.15 menunjukkan bahwa data dengan nilai umur yang sama (*age* = 17) memiliki kelompok cluster yang sama (*prediction* = 3.0).

### ***K-Means***

Pada bagian 2.14.3 menjelaskan parameter pemodelan *k-means* pada Spark MLlib. Berikut adalah tahapan eksperimen pada Listing 3.38 untuk pemodelan *k-means*:

1. Membuat model *k-means* menggunakan Spark MLlib
2. Menentukan jumlah cluster (*k*) untuk pemodelan *k-means*.
3. Melakukan pelatihan data pada pemodelan *k-means*.
4. Mencari nilai *centroid* dari masing-masing *cluster*.
5. Mengembalikan hasil *clustering* dalam bentuk tabel.

Listing 3.38: Eksperimen K-Means Spark MLlib

```

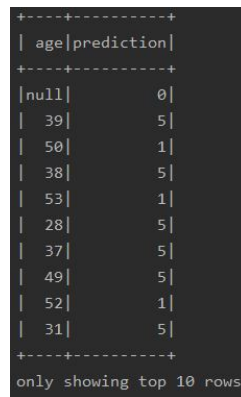
// KMeans with 8 clusters
val kmeans = new KMeans()
    .setK(8)
    .setFeaturesCol("features")
    .setPredictionCol("prediction")
val kmeansModel = kmeans.fit(result_df)
kmeansModel.clusterCenters.foreach(println)

// Predict model
val predictDf = kmeansModel.transform(result_df)
predictDf.show(10)

```



Dataset yang dipakai untuk eksperimen *k-means* ini adalah sampel data dari dataset Adult yang sudah dijelaskan pada bagian. Data ini berjumlah 100.000 baris data dengan ukuran 11.000 KB. Batas maksimum iterasi untuk fungsi *k-means* telah diatur sedemikian rupa oleh library Spark MLlib itu sendiri, sehingga parameter ini tidak perlu lagi diset manual oleh pengguna. Jumlah cluster yang akan dibentuk pada eksperimen ini berjumlah 8 cluster. Jumlah cluster pada fungsi *k-means* nantinya dapat diatur kembali sesuai kebutuhan eksperimen.



```

+----+-----+
| age|prediction|
+----+-----+
| null|    0|
|  39|    5|
|  50|    1|
|  38|    5|
|  53|    1|
|  28|    5|
|  37|    5|
|  49|    5|
|  52|    1|
|  31|    5|
+----+-----+
only showing top 10 rows

```

Gambar 3.16: Hasil K-Means Spark MLlib

Hasil pemodelan *k-means* dari eksperimen ini adalah *cluster* data. Cluster ini terbentuk berdasarkan distance terdekat antara anggota data dengan titik centroid pada cluster tersebut. Sehingga data-data yang tergabung pada sebuah cluster, sudah dipastikan bahwa data-data tersebut lebih dekat dengan titik centroid pada cluster tersebut dibanding dengan titik centroid pada cluster lainnya. Gambar 3.16 menunjukkan bahwa data dengan atribut (*age* = 39) memiliki kelompok *cluster* yang sama dengan data lain dengan atribut (*age* = 38), karena berada pada kelompok data yang sama dengan representasi nama kelompok (*prediction* = 5).

### 3.4 Gambaran Umum Perangkat Lunak

Penelitian ini menghasilkan dua jenis perangkat lunak dengan tujuan yang berbeda satu sama lain, untuk menyelesaikan permasalahan penerapan algoritma *Greedy k-member clustering* pada lingkungan *big data*. Berikut adalah deskripsi perangkat lunak yang akan dibuat:

1. Perangkat lunak dapat mengimplementasikan algoritma *k-anonymity*. Algoritma *k-anonymity* yang dimaksud adalah algoritma *Greedy k-member clustering*. Masukan dari perangkat lunak ini adalah dataset *Adult* dalam format file CSV, tabel atribut quasi-identifier (QID), dan parameter dari algoritma *Greedy k-member clustering* yaitu nilai *k* dan objek *Domain Generalization Hierarchy* (DGH). Keluaran dari perangkat lunak ini adalah hasil anonimisasi dari dataset *Adult* yang disimpan dalam format file CSV.
2. Perangkat lunak dapat membandingkan hasil anonimisasi algoritma *k-anonymity* melalui metode *data mining* yang telah disediakan. Metode *data mining* yang akan disediakan adalah klasifikasi dengan pemodelan *Naive Bayes* dan pengelompokan/*clustering* dengan pemodelan *k-means*. Masukan dari perangkat lunak ini adalah dataset *Adult* dalam format file CSV, baik dataset pernah dilakukan proses anonimisasi maupun dataset asli. Untuk pemodelan *k-means* membutuhkan parameter tambahan seperti nilai *k* dan jenis atribut yang akan diprediksi nilainya. Sedangkan untuk pemodelan *Naive Bayes* membutuhkan parameter tambahan seperti persentase antara *training* dan *testing* data, jenis atribut yang akan diprediksi nilainya. Keluaran dari perangkat lunak ini untuk pemodelan *Naive Bayes* dan *k-means* memiliki hasil yang sama, yaitu jenis nilai dari atribut yang telah dipilih dan jenis *cluster*.

### 3.4.1 Diagram Aktifitas

Penelitian ini memiliki dua jenis diagram aktivitas, yaitu diagram aktivitas untuk perangkat lunak anonimisasi data dan diagram aktivitas untuk perangkat lunak analisis data. Tujuan dari membuat dua jenis perangkat lunak antara lain untuk memisahkan perangkat lunak dari fungsionalitas yang berbeda. Fungsionalitas tersebut antara lain melakukan proses anonimisasi data pada dataset dan membandingkan hasil antara dataset asli dengan dataset yang telah dilakukan proses anonimisasi untuk mencari tahu seberapa baik kinerja algoritma *Greedy k-member clustering* untuk mendapatkan hasil yang informatif.

#### Perangkat Lunak Anonimisasi Data

Perangkat lunak ini bertujuan untuk melakukan proses anonimisasi pada dataset *Adult* menggunakan algoritma *k-anonymity*. Diagram aktifitas dapat dilihat pada Gambar 3.18, berikut adalah tahapan yang terjadi pada perangkat lunak saat melakukan proses anonimisasi data:

1. Pengguna memberi masukan dalam format file CSV dan beberapa jenis atribut *quasi-identifier* untuk menjadi tabel input pada proses anonimisasi.
2. Perangkat lunak menampilkan sebagian baris data dari tabel input karena baris data yang akan digunakan pada eksperimen akan berjumlah sangat banyak .
3. Pengguna akan meninjau ulang apakah jumlah kolom yang ditampilkan sudah sesuai dengan jumlah atribut *quasi-identifier* yang akan dipakai.
4. Menggunakan memberikan parameter tambahan seperti rentang nilai k untuk menentukan jumlah anggota *cluster* dan objek DGH untuk proses anonimisasi.
5. Perangkat lunak akan melakukan proses anonimisasi dengan bantuan Spark pada tabel input berdasarkan parameter tambahan yang diberikan sebelumnya.
6. Perangkat lunak mengembalikan seluruh isi *log* yang dihasilkan selama proses eksekusi Spark berlangsung kepada pengguna untuk deteksi *error*.
7. Perangkat lunak hanya menampilkan baris data yang berubah akibat proses anonimisasi pada GUI dan hasil keseluruhannya dalam format *file* CSV.
8. Perangkat lunak mengembalikan nilai *information loss* pada masing-masing *cluster* yang terbentuk agar pengguna dapat mencari hasil yang optimal.
9. Pengguna dapat membandingkan hasil anonimisasi antara baris data yang berubah akibat proses anonimisasi dengan baris data yang ada pada tabel asli.
10. Pengguna dapat mengulangi eksperimen untuk mencari nilai k terbaik agar dihasilkan *information loss* seminimal mungkin pada proses anonimisasi.

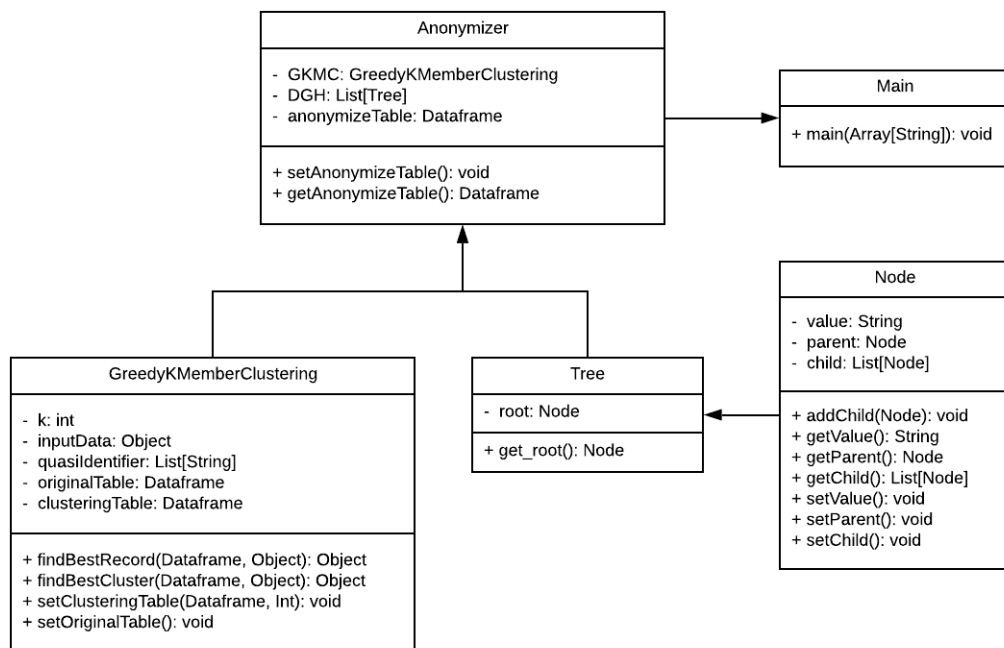
#### Perangkat Lunak Analisis Data

Perangkat lunak ini bertujuan untuk mencari perbandingan hasil sebelum dan setelah data dilakukan proses anonimisasi dengan metode *data mining*. Diagram aktifitas dapat dilihat pada Gambar 3.19, berikut adalah tahapan yang terjadi pada perangkat lunak saat melakukan pemodelan *data mining*:

1. Pengguna memberi dua jenis masukan yaitu data asli dan data hasil anonimisasi dalam format *file* CSV untuk menjadi tabel input pada proses analisis data.
2. Perangkat lunak hanya menampilkan sebagian baris data dari dua jenis tabel input karena input baris data pada eksperimen berjumlah sangat banyak

3. Pengguna meninjau kembali apakah jumlah kolom yang ditampilkan pada kedua jenis tabel memiliki jumlah kolom atribut yang sama.
4. Pengguna memilih jenis pemodelan data mining yang tersedia pada eksperimen, yaitu klasifikasi dengan *Naive Bayes* atau pengelompokan/*clustering* dengan *k-means*.
5. Pengguna mengisi parameter pada pemodelan yang dipilih. Contoh pada *k-means* adalah nilai *k* dan satu jenis atribut. Sedangkan pada *Naive Bayes* adalah persentase *training*, *testing* data dan satu jenis atribut.
6. Perangkat lunak akan melakukan proses pelatihan data pada Spark untuk menemukan klasifikasi/pengelompokan yang sesuai berdasarkan jenis pemodelan yang dipilih.
7. Perangkat lunak mengembalikan seluruh isi *log* yang dihasilkan selama proses eksekusi Spark berlangsung kepada pengguna untuk deteksi *error*.
8. Perangkat lunak menampilkan sebagian hasil prediksi *cluster* untuk masing-masing data dan menyimpan hasil keseluruhannya dalam format *file* CSV.
9. Pengguna melakukan analisis lebih lanjut terkait pengelompokan dan klasifikasi kelompok data yang terbentuk dari proses pemodelan *data mining*.

### 3.4.2 Diagram Kelas



Gambar 3.17: Diagram Kelas Anonimisasi Data

Diagram kelas bertujuan untuk menggambarkan keterhubungan antar kelas. Pada penelitian ini digambarkan diagram kelas untuk perangkat lunak anonimisasi data. Karena perangkat lunak analisis data hanya memiliki satu kelas saja, maka keterhubungan antar kelas tidak perlu digambarkan dalam diagram kelas. Gambar 3.17 menggambarkan keterhubungan antar kelas pada perangkat lunak anonimisasi data. Berikut adalah penjelasan lengkap mengenai deskripsi kelas dan method pada perangkat lunak anonimisasi data:

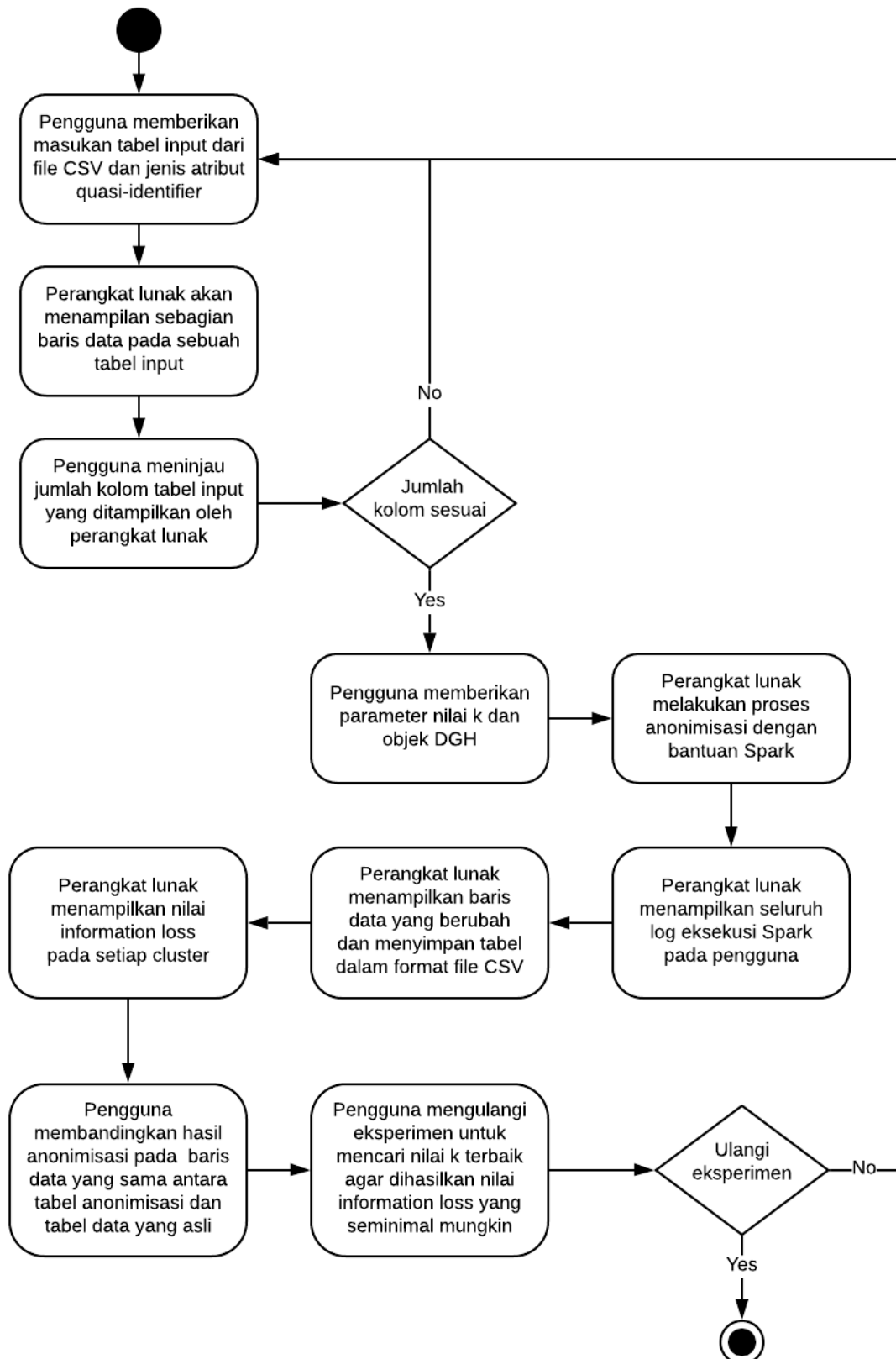
- Kelas *Anonymizer* bertujuan untuk melakukan proses anonimisasi setelah data dikelompokkan menjadi beberapa *cluster*. Kelas *Anonymizer* memiliki 2 jenis variabel, yaitu:
  - *GKMC* adalah objek dari kelas *GreedyKMemberClustering* yang berisi tabel hasil pengelompokan data berdasarkan algoritma *Greedy k-member clustering*.
  - *DGH* adalah array 1 dimensi dari objek *Tree* yang berisi hasil anonimisasi untuk nilai *quasi-identifier* yang unik agar menjadi nilai yang lebih umum.
  - *anonymizeTable* adalah array 2 dimensi dari kelas *Object* untuk menyimpan tabel hasil anonimisasi data.

Kelas *Anonymizer* memiliki 2 jenis method, yaitu:

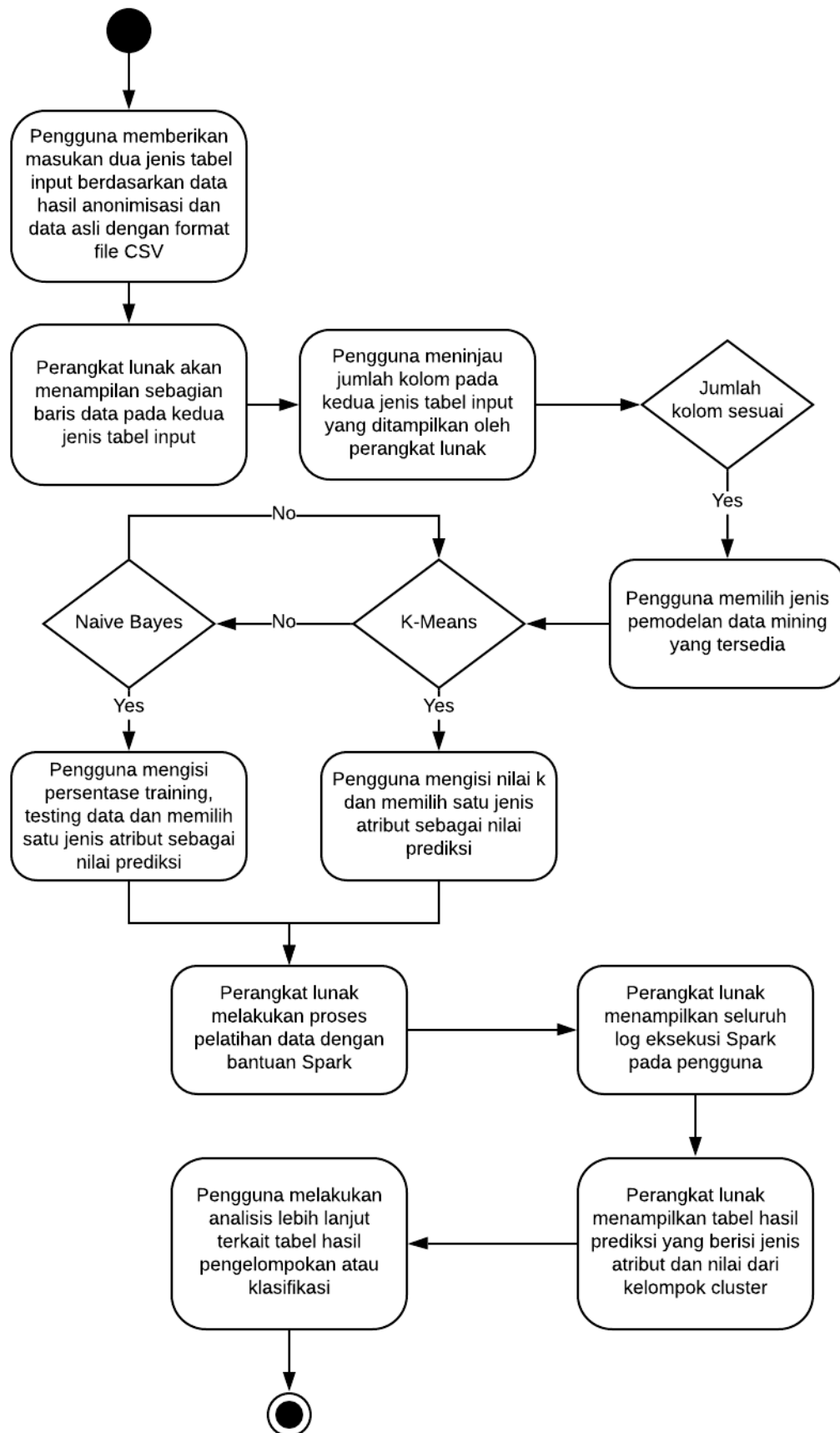
- *setAnonymizeTable()* bertujuan untuk melakukan proses anonimisasi pada masing-masing baris data yang tergabung dalam sebuah *cluster*, berdasarkan perbedaan nilai dari beberapa *quasi-identifier*.
- *getAnonymizeTable()* bertujuan untuk mengambil nilai pada atribut *anonymizeTable*.
- Kelas *GreedyKMemberClustering* bertujuan untuk melakukan pengelompokan data menjadi beberapa *cluster* berdasarkan sifat/nilai atribut yang dimiliki oleh masing-masing baris data. Kelas *GreedyKMemberClustering* memiliki 5 jenis variabel, yaitu:
  - *k* adalah variabel bertipe *Integer* untuk membatasi jumlah anggota pada sebuah *cluster* agar memiliki jumlah yang tetap sebanyak jumlah tertentu.
  - *inputData* adalah variabel untuk menyimpan seluruh baris data *file* CSV.
  - *quasiIdentifier* adalah daftar dari nama-nama kolom yang akan dipilih untuk membuat tabel baru yang digunakan pada proses anonimisasi data
  - *originalTable* adalah tabel yang menyimpan seluruh baris data pada *file* CSV berdasarkan jenis kolom yang terpilih pada variabel *quasiIdentifier*.
  - *clusteringTable* adalah tabel yang menyimpan hasil pengelompokan baris data dari algoritma *Greedy k-member clustering*.

Kelas *GreedyKMemberClustering* memiliki 4 jenis method, yaitu:

- *findBestRecord()* bertujuan mencari sebuah baris data yang memiliki nilai *information loss* yang paling minimal dengan baris data lainnya.
- *findBestCluster()* bertujuan mencari sebuah *cluster* data yang memiliki nilai *information loss* yang paling minimal dengan *cluster* lainnya.
- *setClusteringTable()* bertujuan mengelompokkan data berdasarkan algoritma *Greedy k-member clustering* dan hasilnya disimpan pada variabel *clusteringTable*.
- *setOriginalTable()* bertujuan mengubah hasil pembacaan data input CSV menjadi tabel baru dan hasilnya disimpan pada variabel *originalTable*.
- Kelas *Tree* bertujuan untuk membuat pohon generalisasi berdasarkan jenis atribut *quasi-identifier* yang dipilih.
- Kelas *Node* bertujuan untuk menyimpan seluruh nilai *quasi-identifier* yang unik untuk masing-masing baris data.
- Kelas *Main* bertujuan untuk membuat tahapan anonimisasi dari awal sampai akhir dengan memanfaatkan pemanggilan *method* dari masing-masing objek kelas.



Gambar 3.18: Diagram Aktifitas Anonimisasi Data



Gambar 3.19: Diagram Aktifitas Analisis Data

# LAMPIRAN A

## KODE PROGRAM

Listing A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4
5 #include<stdio.h>
6
7 void myFunction( int input, float* output ) {
8     switch ( array[i] ) {
9         case 1: // This is silly code
10             if ( a >= 0 || b <= 3 && c != x )
11                 *output += 0.005 + 20050;
12             char = 'g';
13             b = 2^n + ~right_size - leftSize * MAX_SIZE;
14             c = (--aaa + &daa) / (bbb++ - ccc % 2 );
15             strcpy(a,"hello_$@?");
16         }
17         count = ~mask | 0x00FF00AA;
18     }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Listing A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id; //id of the set
8     protected MyEdge FurthestEdge; //the furthest edge
9     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID; //store the ID of all vertices
12    protected ArrayList<Double> closeDist; //store the distance of all vertices
13    protected int totaltrj; //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35
36 }

```





## LAMPIRAN B

### HASIL EKSPERIMEN

Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4