

## 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:

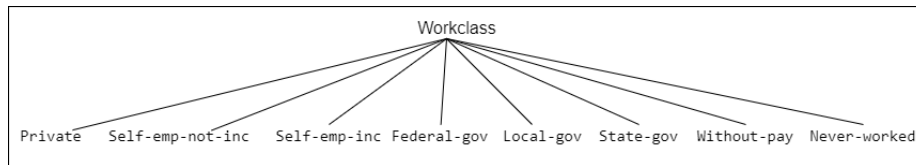
1. 39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
2. 50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 13, United-States, <=50K

*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

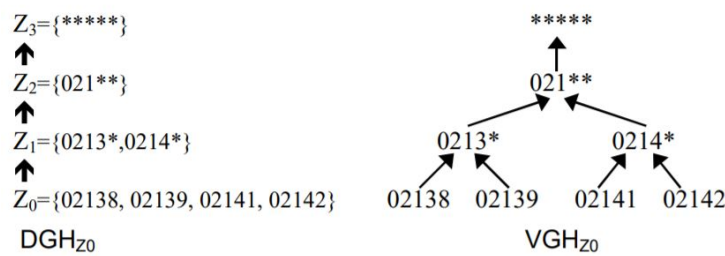
Pada bagian 2.7, telah dijelaskan konsep mengenai *Hierarchy Based Generalization*. DGH adalah contoh penerapan dari *Hierarchy Based Generalization*. DGH bertujuan untuk melindungi data dengan cara menerapkan metode generalisasi terhadap nilai atribut data yang bersifat unik, agar menjadi nilai yang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

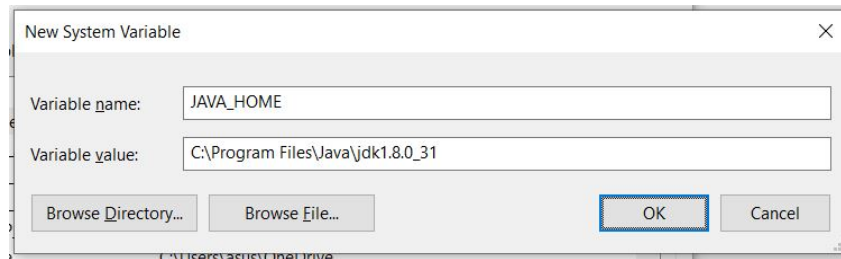
1. Melakukan instalasi Java 8.
2. Melakukan instalasi Spark 2.4.5
3. Melakukan instalasi IntelliJ untuk Scala sbt.



## Instalasi Java 8

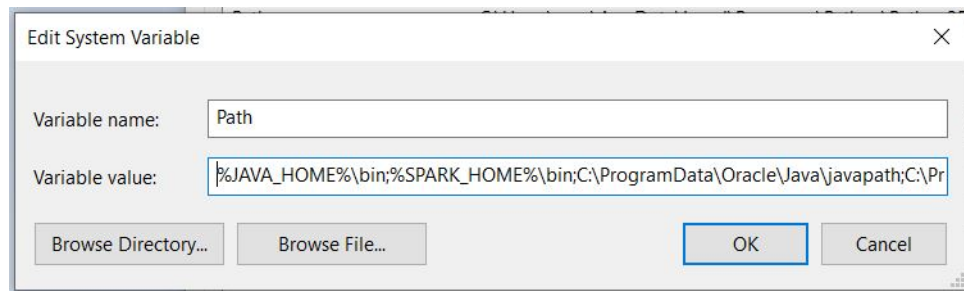
Berikut adalah tahapan instalasi Java 8 secara lengkap:

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

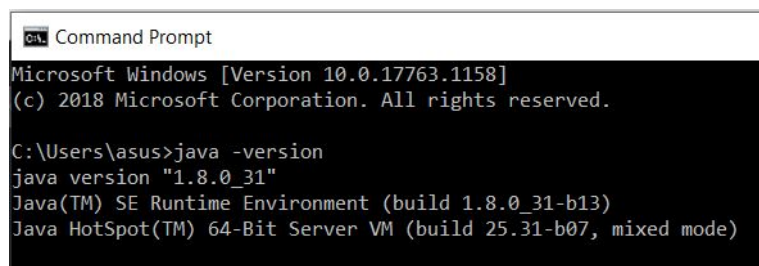
5. Tambahkan `%JAVA_HOME%\bin;` pada Path di System variables seperti Gambar 3.7.



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.



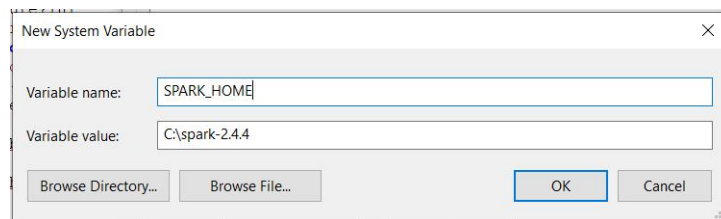
Gambar 3.5: Perintah `java -version`

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

### Instalasi Spark 2.4.5

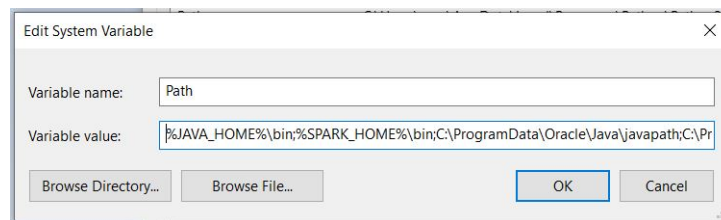
Berikut adalah tahapan instalasi Spark 2.4.5 secara lengkap:

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

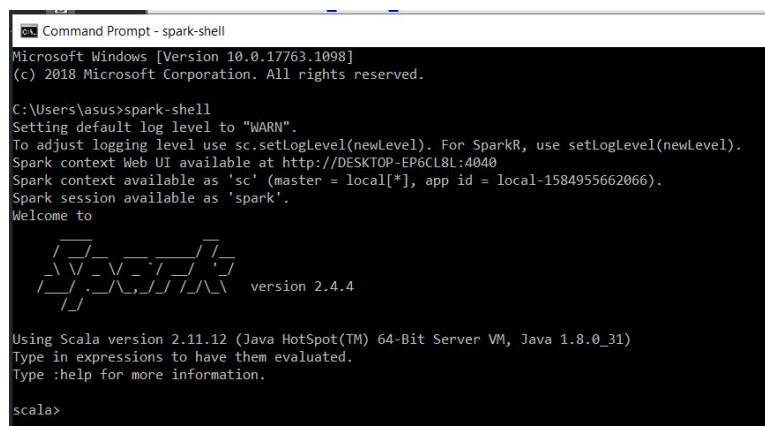
5. Tambahkan %SPARK\_HOME%\bin; pada Path di System variables seperti Gambar 3.7



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.

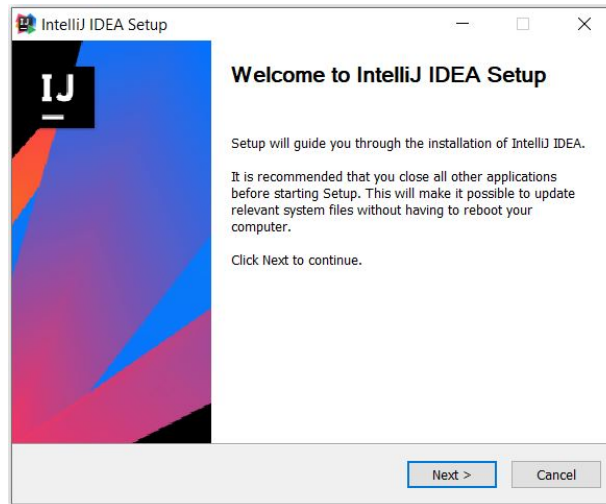


Gambar 3.8: Spark 2.4.5

## Instalasi IntelliJ untuk Scala SBT

Berikut adalah tahapan instalasi IntelliJ:

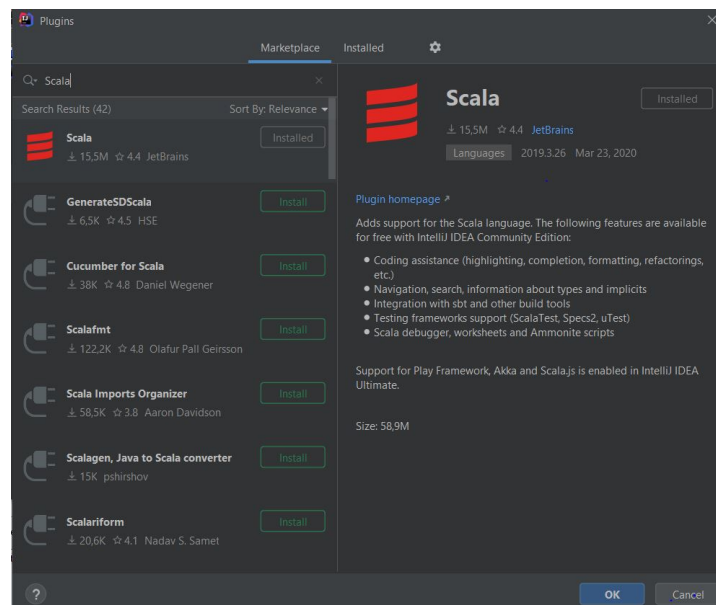
1. Download IntelliJ melalui link berikut  
<https://www.jetbrains.com/idea/download/#section=windows>
2. Lakukan instalasi IntelliJ seperti biasa.



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

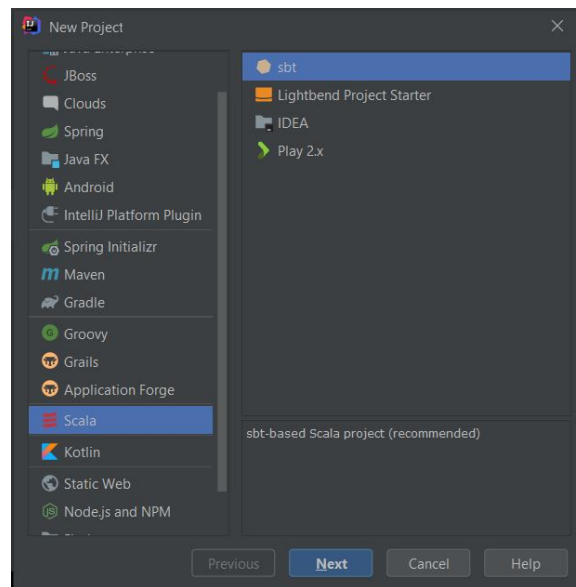
3. Klik tombol *install*

### 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.

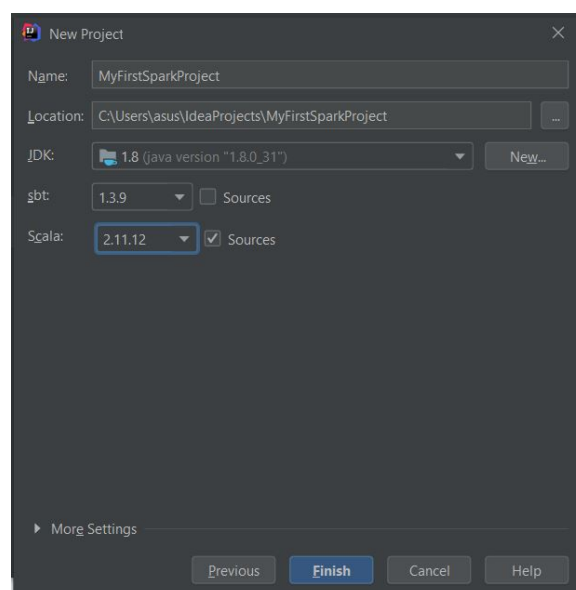
Berikut adalah tahapan pembuatan *project* Spark pada IntelliJ:

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.



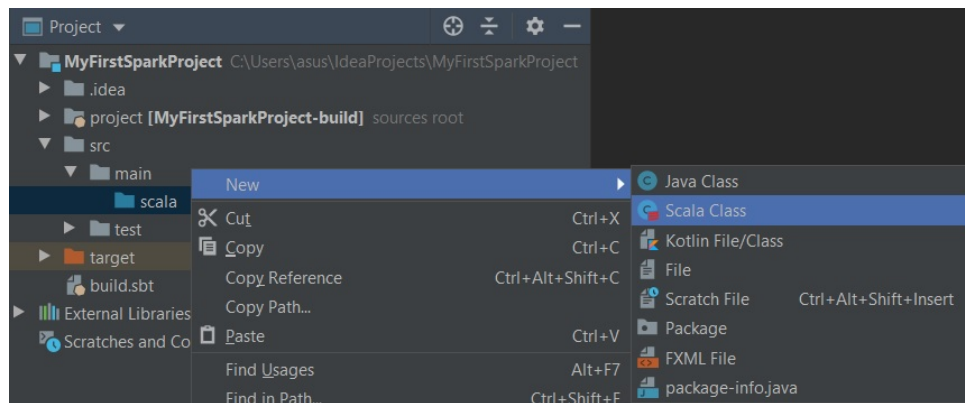
Gambar 3.12: Melakukan Konfigurasi Project Spark

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

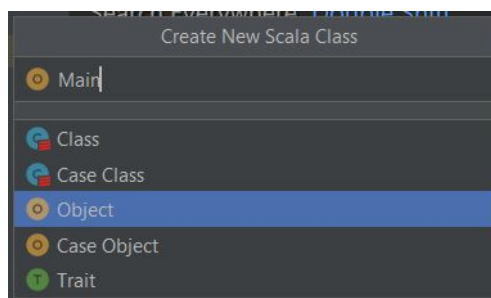
```
name := "NamaProject"
version := "0.1"
scalaVersion := "2.11.12"
// https://mvnrepository.com/artifact/org.apache.spark/spark-core
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
// https://mvnrepository.com/artifact/org.apache.spark/spark-sql
libraryDependencies += "org.apache.spark"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

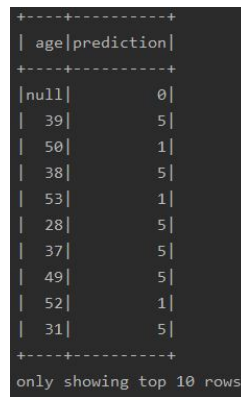
```

// 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.

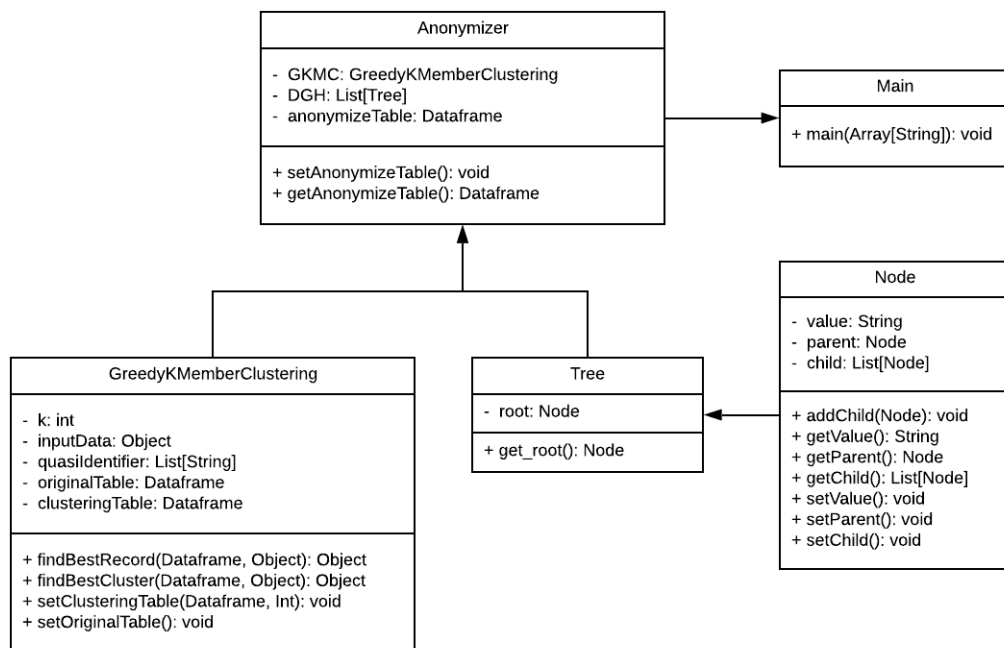
#### Perangkat Lunak Analisis Data

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:

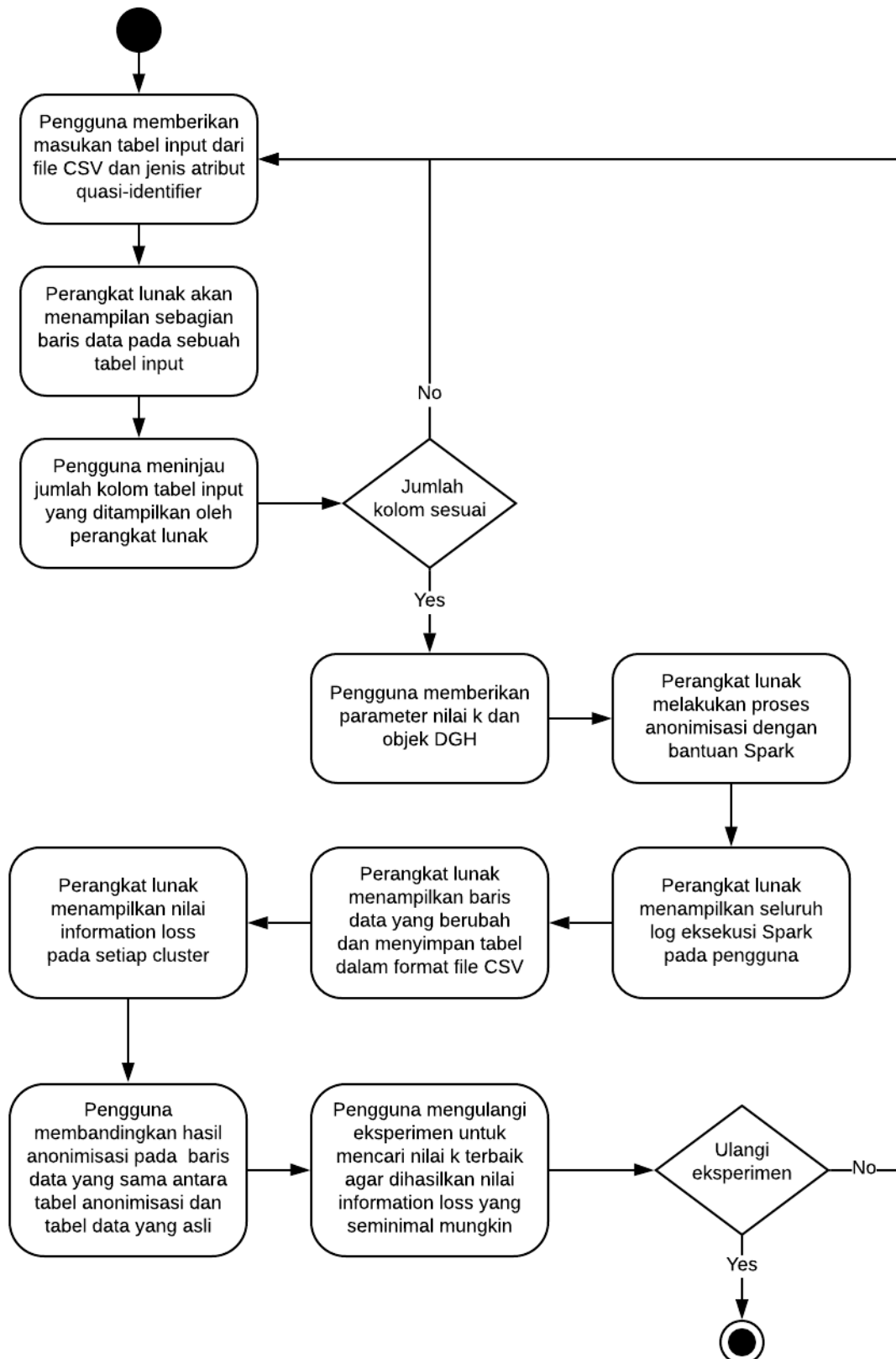
- Kelas *Anonymizer* bertujuan untuk melakukan proses anonimisasi setelah data dikelompokkan menjadi beberapa *cluster*. Kelas *Anonymizer* memiliki 2 jenis variabel, yaitu:
  - *GKMC* adalah objek dari kelas *GreedyKMemberClustering* yang berisi tabel hasil pengelompokan data berdasarkan algoritma *Greedy 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:

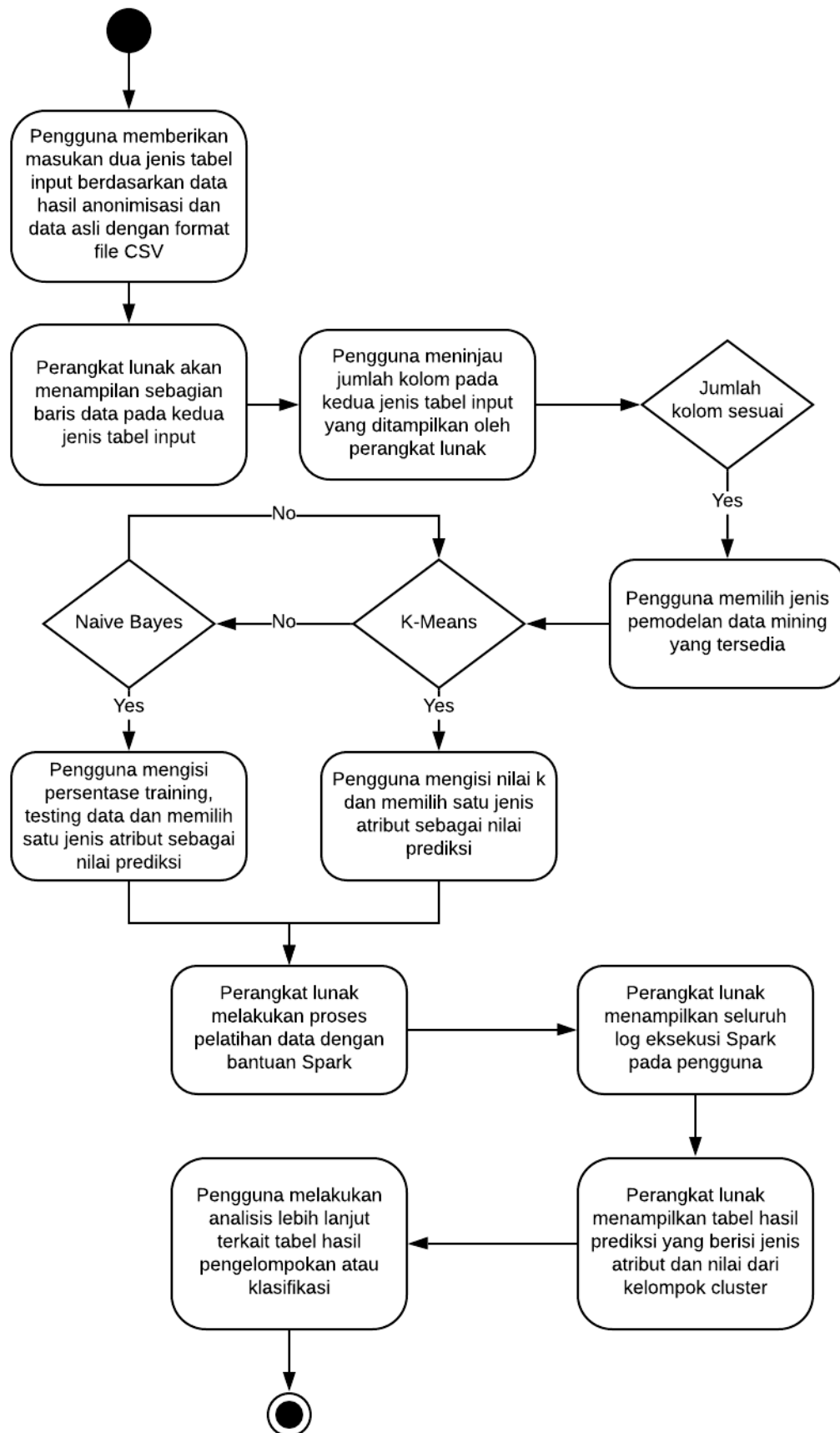
- *setAnonymizeTable()* bertujuan untuk melakukan proses anonimisasi pada masing-masing baris data yang tergabung dalam sebuah *cluster*, berdasarkan perbedaan nilai dari beberapa *quasi-identifier*.
- *getAnonymizeTable()* bertujuan untuk mengambil nilai pada atribut *anonymizeTable*.
- Kelas *GreedyKMemberClustering* 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