

SKRIPSI

PENERAPAN ALGORITMA ANONIMISASI DATA PADA LINGKUNGAN BIG DATA



Stephen Jordan

NPM: 2016730018

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2020

UNDERGRADUATE THESIS

**APPLICATION OF DATA ANONYMIZATION ALGORITHM
IN BIG DATA ENVIRONMENT**



Stephen Jordan

NPM: 2016730018

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2020**

ABSTRAK

Data mining umumnya digunakan untuk menganalisis pola-pola dari data yang dikumpulkan. Untuk mendapatkan hasil yang valid, data yang dianalisis harus sangat banyak. Oleh karena itu, teknologi *big data* muncul untuk menangani masalah tersebut. Sayangnya, proses *data mining* dapat menimbulkan masalah privasi. Privasi adalah hak seseorang untuk memiliki kendali atas bagaimana informasi pribadi dikumpulkan dan digunakan. Konsep PPDM dipakai untuk melindungi privasi setiap individu sebelum dilakukan proses *data mining*. Contoh dari metode PPDM adalah metode *k-anonymity* untuk melakukan anonimisasi data. *K-anonymity* adalah metode anonimisasi data dari PPDM untuk menjaga agar sebuah data tidak dapat dibedakan dengan $k - 1$ data lainnya. Karena metode anonimisasi menderita kehilangan informasi yang besar, maka data akan dikelompokan terlebih dahulu menggunakan algoritma *greedy k-member clustering*. Tujuan dari penelitian ini adalah melakukan implementasi algoritma *greedy k-member clustering* dan *k-anonymity* pada lingkungan *big data* dan menguji model *data mining* klasifikasi dan *clustering* sebelum dan setelah data dilakukan anonimisasi data.

Pada penelitian ini, telah dibangun tiga buah perangkat lunak dengan *framework* Spark. Perangkat lunak eksplorasi yang bertujuan mencari nilai unik sebuah kolom untuk dipakai dalam membuat pohon generalisasi. Perangkat lunak anonimisasi yang berisi implementasi algoritma *greedy k-member clustering* dan *k-anonymity*. Perangkat lunak pengujian untuk mengamati hasil pemodelan *data mining* sebelum dan setelah data dilakukan anonimisasi. Hasil perangkat lunak anonimisasi dipakai untuk tahap analisis. Analisis dilakukan dengan pengujian fungsional dan eksperimental. Pengujian fungsional bertujuan memeriksa hasil perangkat lunak. Pengujian eksperimental bertujuan mendapatkan waktu komputasi dari algoritma *greedy k-member clustering* dan *k-anonymity*, waktu komputasi pemodelan model *data mining* klasifikasi dan *clustering*, menghitung *total information loss*, evaluasi hasil *data mining*, dan mencari perbedaan hasil prediksi terbaik dari masing-masing model *data mining*.

Hasil pengujian kualitas informasi menunjukkan bahwa *total information loss* terendah dengan penggunaan kolom campuran, mengurangi jumlah *quasi-identifier* bertipe numerik, memakai ukuran data yang relatif kecil. Untuk waktu komputasinya, algoritma *greedy k-member clustering* membutuhkan waktu yang sangat lama untuk melakukan pengelompokan data, sedangkan algoritma *k-anonymity* dapat dilakukan komputasi dengan cepat. Saat dilakukan pengujian teknik *data mining*, dapat dilihat bahwa metode *clustering* memiliki perbedaan hasil pengelompokan yang cukup jauh antara sebelum dan setelah data dianonimisasi, sedangkan metode klasifikasi memiliki perbedaan hasil klasifikasi yang cukup dekat. Sehingga, pemodelan *data mining* yang cocok untuk anonimisasi data adalah klasifikasi.

Kata-kata kunci: *Data Mining*, *Big Data*, Privasi, *Privacy Preserving Data Mining (PPDM)*, K-Anonymity, Greedy K-Member Clustering

ABSTRACT

Data mining is used to analyze patterns of collected data. To get a valid result, the data to be analyzed must be very large. Hence, big data technology emerged to address this problem. Unfortunately, data mining process can create privacy concerns. Privacy is a person's right to have control over how personal information is collected and used. PPDM concept is used to protect the privacy of each individual before the data mining process is carried out. An example of the PPDM method is the k-anonymity. K-anonymity is a data anonymization method from PPDM to keep data indistinguishable from other $k - 1$ data. Since the anonymization method suffers from a large loss of information, the data will be grouped first using the greedy k-member clustering algorithm. The purpose of this research is to implement the greedy k-member clustering and k-anonymity algorithms in the big data environment and test the data mining classification and clustering models before and after the data is anonymized.

In this research, there are three pieces of Spark software. Exploration software aims to find the unique values of a column to create a generalization tree. Anonymization software contains the implementation of the greedy k-member clustering and k-anonymity algorithms. Testing software for observing the results of data mining modeling before and after the data is anonymized. The results of the anonymization software are used for the analysis stage. The analysis was carried out by functional and experimental testing. Functional aims to check software results. The experimental aims to get the computation time of the greedy k-member clustering and k-anonymity algorithms, the computation time of the data mining classification model and clustering, calculating total information loss, evaluate the results of data mining, and differences in the best predictions of each data mining model.

The results of the information quality test show that the lowest total information loss is with the use of mixed columns, reducing the number of quasi-identifier numeric types, using a relatively small data size. For computation time, the greedy k-member clustering algorithm takes a very long time to group data, while the k-anonymity algorithm can be computationally fast. When testing the data mining technique, it can be seen that the clustering method has quite a large difference in the results of the grouping between before and after the data is anonymized, while the classification method has quite close differences in classification results. Thus, data mining modeling suitable for data anonymization is classification.

Keywords: Data Mining, Big Data, Privacy, Privacy Preserving Data Mining, K-Anonymity, Greedy K-Member Clustering

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	xiii
DAFTAR TABEL	xvii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	4
2 LANDASAN TEORI	5
2.1 Privasi	5
2.2 <i>Data Mining</i>	6
2.2.1 Klasifikasi	7
2.2.2 <i>Clustering</i>	13
2.3 Tahapan Evaluasi <i>Data Mining</i>	18
2.3.1 Menghitung Tingkat Akurasi untuk Model Klasifikasi	18
2.3.2 Menghitung <i>Silhouette Score</i> untuk Model <i>Clustering</i>	18
2.4 <i>Privacy-Preserving Data Mining</i> (PPDM)	19
2.5 Metode Anonimisasi	19
2.6 <i>K-Anonymity</i>	20
2.7 <i>Domain Generalization Hierarchy</i> (DGH)	22
2.8 <i>Greedy K-Member Clustering</i>	23
2.9 Metrik <i>Distance</i> dan <i>Information Loss</i>	26
2.9.1 <i>Distance</i>	27
2.9.2 <i>Information Loss</i>	27
2.10 Sistem Terdistribusi	28
2.10.1 Penggunaan Sistem Terdistribusi	28
2.10.2 Komputasi Big Data Dengan Sistem Terdistribusi	29
2.11 Big Data	30
2.11.1 Karakteristik Big Data	30
2.11.2 Jenis Data Pada Big Data	31
2.11.3 Pekerjaan Terkait Big Data	31
2.12 Hadoop	32
2.12.1 HDFS	32
2.12.2 MapReduce	32
2.13 Spark	34
2.13.1 Arsitektur Spark	34
2.13.2 Jenis Instalasi pada Spark	35

2.13.3	Resilient Distributed Datasets (RDD)	36
2.13.4	<i>DataFrame</i>	37
2.13.5	Komponen Spark	38
2.14	Spark MLlib	39
2.14.1	Tipe Data pada Spark MLlib	40
2.14.2	<i>Data Mining</i> pada Spark MLlib	40
2.15	Scala	43
2.16	Format Penyimpanan Data	44
2.16.1	CSV	44
2.16.2	JSON	44
3	ANALISIS	45
3.1	Analisis Masalah	45
3.2	Gambaran Umum Perangkat Lunak	46
3.2.1	Diagram Aktifitas	47
3.2.2	Diagram Kelas	49
3.2.3	Pengenalan Karakteristik Data	53
3.2.4	<i>Personally Identifiable Information</i>	54
3.2.5	Perhitungan <i>Distance</i> dan <i>Total Information Loss</i>	54
3.2.6	<i>Greedy K-Member Clustering</i>	56
3.2.7	<i>Domain Generalization Hierarchy</i>	58
3.2.8	<i>K-Anonymity</i>	59
3.3	Eksplorasi Spark	60
3.3.1	Instalasi Spark	60
3.3.2	Membuat <i>Project</i> Spark pada IntelliJ	64
3.3.3	Menjalankan Program Spark pada Komputer Lokal	66
3.3.4	Menjalankan Program Spark pada Hadoop Cluster	66
3.4	Studi Kasus	67
3.4.1	Eksperimen Scala	67
3.4.2	Eksperimen Spark	70
3.4.3	Eksperimen Komponen Spark	74
3.4.4	Eksperimen Spark MLIB	79
4	PERANCANGAN	83
4.1	Penggunaan Spark Web UI	84
4.1.1	Menu Spark Jobs	85
4.1.2	Menu Stages	85
4.1.3	Menu Tasks	86
4.1.4	Menu Storage	86
4.1.5	Menu Environment	87
4.1.6	Menu SQL	87
4.1.7	Menu Executors	87
4.2	Perancangan Perangkat Lunak	88
4.3	Diagram Kelas Lengkap	89
4.3.1	Diagram Package	89
4.3.2	Diagram Kelas pada Package ExploratoryModel	90
4.3.3	Diagram Kelas pada Package AnonymizationModel	91
4.3.4	Diagram Kelas pada Package ExaminationModel	104
4.4	Masukan Perangkat Lunak	108
4.4.1	Masukan Perangkat Lunak Eksplorasi	109
4.4.2	Masukan Perangkat Lunak Anonimisasi	109
4.4.3	Masukan Perangkat Lunak Pengujian	112

5 IMPLEMENTASI DAN PENGUJIAN	115
5.1 Implementasi Antarmuka	115
5.1.1 Komputer Lokal dengan IntelIJ	115
5.1.2 Hadoop Cluster dengan Terminal Ubuntu	124
5.2 Pengujian	132
5.2.1 Pengujian Fungsional	132
5.2.2 Pengujian Eksperimental	138
6 KESIMPULAN DAN SARAN	151
6.1 Kesimpulan	151
6.2 Saran	152
DAFTAR REFERENSI	153
A KONFIGURASI LIBRARY SPARK	155
A.1 Perangkat Lunak Eksplorasi	155
A.2 Perangkat Lunak Anonimisasi	155
A.3 Perangkat Lunak Pengujian	155
B KODE PROGRAM PERANGKAT LUNAK EKSPLORASI	157
B.1 Kelas MainExploratory	157
C KODE PROGRAM PERANGKAT LUNAK ANONIMISASI	159
C.1 Kelas Node	159
C.2 Kelas BinaryTree	159
C.3 Kelas GreedyKMemberClustering	160
C.4 Kelas KAnonymity	164
C.5 Kelas LowestCommonAncestor	167
C.6 Kelas MainAnonymization	167
C.7 Kelas MainClusterization	169
C.8 Kelas MainLCATesting	171
D KODE PROGRAM PERANGKAT LUNAK PENGUJIAN	173
D.1 Kelas KMeansModel	173
D.2 Kelas MainExamination	174
D.3 Kelas NaiveBayesModel	181
E MASUKAN FILE JSON SELURUH PERANGKAT LUNAK	183
E.1 Masukan JSON Perangkat Lunak Eksplorasi	183
E.2 Masukan JSON Perangkat Lunak Anonimisasi	183
E.3 Masukan JSON Perangkat Lunak Pengujian	190
F HASIL PENGUJIAN FUNGSIONAL	193
F.1 Hasil Pengelompokan Greedy K-Member Clustering	193
F.2 Hasil Anonimisasi K-Anonymity	194
F.3 Hasil Clustering K-Means (Sebelum Anonimisasi)	195
F.4 Hasil Clustering K-Means (Setelah Anonimisasi)	196
F.5 Hasil Klasifikasi Naive Bayes (Sebelum Anonimisasi)	197
F.6 Hasil Klasifikasi Naive Bayes (Setelah Anonimisasi)	198

DAFTAR GAMBAR

2.1	Tahapan pada KDD	6
2.2	Ilustrasi Supervised Learning	7
2.3	Ilustrasi Supervised Learning	8
2.4	Ilustrasi Supervised Learning	8
2.5	Contoh <i>Logistic Regression</i>	9
2.6	Contoh <i>Hierarchical Clustering</i>	13
2.7	Contoh <i>Hierarchical Clustering</i>	14
2.8	Contoh <i>Partitional Clustering</i>	14
2.9	Contoh <i>Partitional Clustering</i>	15
2.10	<i>Privacy Preserving Data Mining</i> (PPDM)	19
2.11	<i>Privacy Preserving Data Mining</i> (PPDM)	20
2.12	<i>Privacy Preserving Data Mining</i> (PPDM)	21
2.13	Contoh Implementas DGH,VGH (ZIP)	22
2.14	Kelemahan Algoritma Greedy	23
2.15	Sistem Terdistribusi	28
2.16	Pemanfaatan Sistem Terdistribusi	29
2.17	Pemanfaatan Sistem Terdistribusi	30
2.18	Hadoop	32
2.19	Arsitektur HDFS	32
2.20	Proses Komputasi pada MapReduce	33
2.21	Arsitektur Spark	34
2.22	Arsitektur Spark	34
2.23	Arsitektur Spark	35
2.24	Beberapa Jenis Komponen Spark	38
2.25	Contoh Vektor Dense dan Sparse	39
2.26	Contoh Vektor Dense dan Sparse	40
2.27	Scala dan Java JVM	43
3.1	Flow Chart Penggunaan Perangkat Lunak	47
3.2	Diagram Kelas Anonimisasi Data	49
3.3	Diagram Aktifitas Perangkat Lunak Ekplorasi	51
3.4	Diagram Aktifitas Perangkat Lunak Anonimisasi	52
3.5	Diagram Aktifitas Perangkat Lunak Pengujian	52
3.6	Pohon DGH (<code>NAME_INCOME_TYPE</code>)	55
3.7	DGH dan VGH pada atribut ZIP	58
3.8	Environment Variables	61
3.9	Penambahan Variable Value	61
3.10	Perintah <code>java -version</code>	61
3.11	Environment Variable	62
3.12	Penambahan Variable Value	62
3.13	Spark 2.4.5	62
3.14	Instalasi IntelliJ	63

3.15 Plugins Scala	63
3.16 Memilih Bahasa Scala Berbasis sbt	64
3.17 Melakukan Konfigurasi Project Spark	64
3.18 Menambahkan Scala Class pada Project Spark	65
3.19 Memilih Tipe Object pada Scala Class	65
3.20 Hasil Naive Bayes Spark MLLib	80
3.21 Hasil K-Means Spark MLLib	82
4.1 Spark Web UI	83
4.2 Spark Web UI	84
4.3 Menu Spark Jobs	85
4.4 Bagian Scheduling Mode (Spark Jobs)	85
4.5 Menu Stage	86
4.6 Menu Task	86
4.7 Menu Storage	86
4.8 Menu Storage	87
4.9 Menu Storage	87
4.10 Menu Storage	88
4.11 Library Spark (build.sbt)	88
4.12 Diagram Kelas pada Package	89
4.13 Diagram Kelas pada Package ExploratoryModel	90
4.14 Diagram Kelas pada Package AnonymizationModel	91
4.15 Diagram Kelas pada ExaminationModel	104
5.1 IntelliJ	115
5.2 Tombol Selector MainExploratory	116
5.3 Konfigurasi Parameter Perangkat Lunak Ekplorasi	116
5.4 Cara Menjalankan Perangkat Lunak Ekplorasi	117
5.5 Log Perangkat Lunak Ekplorasi	117
5.6 Folder Output Perangkat Lunak Ekplorasi	118
5.7 Flowchart Penggunaan Perangkat Lunak	118
5.8 Tombol Selector MainAnonymization	118
5.9 Konfigurasi Parameter Perangkat Lunak Anonimisasi	119
5.10 Menjalankan Perangkat Lunak Anonimisasi	119
5.11 Log Perangkat Lunak Anonimisasi	120
5.12 Folder Output Greedy K-Member Clustering	120
5.13 Folder Output K-Anonymity	121
5.14 Flowchart Penggunaan Perangkat Lunak (Lanjutan)	121
5.15 Tombol Selector MainExamination	121
5.16 Konfigurasi Parameter Perangkat Lunak Ekplorasi	122
5.17 Menjalankan Perangkat Lunak Ekplorasi	122
5.18 Log Perangkat Lunak Ekplorasi	123
5.19 Folder Output Perangkat Lunak Pengujian	123
5.20 Folder Output Perangkat Lunak Pengujian	124
5.21 Spesifikasi Slaves Node pada Hadoop Cluster	124
5.22 File Input Eksplorasi HDFS	125
5.23 Log Perangkat Lunak Ekplorasi	126
5.24 Folder HDFS Hasil Ekplorasi	127
5.25 Folder HDFS Hasil Ekplorasi Atribut Race	127
5.26 File Input Anonimisasi HDFS	128
5.27 Log Perangkat Lunak Anonimisasi	129
5.28 Folder HDFS Hasil Pengelompokan Data	129

5.29	Folder HDFS Hasil Anonimisasi Data	129
5.30	File Input Pengujian HDFS	130
5.31	Log Perangkat Lunak Pengujian	131
5.32	Folder HDFS Hasil Pengelompokan K-Means	132
5.33	Folder HDFS Hasil Klasifikasi Naive Bayes	132
5.34	Waktu Pengelompokan Data (Jenis Kolom Bervariasi)	142
5.35	Waktu Anonimisasi Data (Jenis Kolom Bervariasi)	142
5.36	Total Information Loss (Jenis Kolom Bervariasi)	142
5.37	Waktu Pengelompokan Data (Jumlah QID bervariasi)	144
5.38	Waktu Anonimisasi Data (Jumlah QID bervariasi)	144
5.39	Total Information Loss (Jumlah QID bervariasi)	144
5.40	Waktu Pengelompokan Data (Ukuran data bervariasi)	146
5.41	Waktu Anonimisasi Data (Ukuran data bervariasi)	146
5.42	Total Information Loss (Ukuran data bervariasi)	146
5.43	Silhouette Score (K-means)	148
5.44	Waktu Komputasi (K-means)	148
5.45	Perbandingan Hasil Clustering Terhadap K-Value (K-means)	148
5.46	Perbandingan Hasil Clustering Terhadap Jumlah Data (K-means)	148
5.47	Perbandingan Tingkat Akurasi (Naive bayes)	149
5.48	Perbandingan Waktu Komputasi (Naive bayes)	149
5.49	Perbandingan Hasil Klasifikasi Terhadap Jumlah Data (Naive bayes)	150

DAFTAR TABEL

2.1	Contoh Kasus <i>PlayGolf</i>	10
2.2	Tabel Probabilitas pada Atribut <i>Outlook</i>	11
2.3	Tabel Probabilitas dari Atribut Temperature	12
2.4	Tabel Probabilitas dari Atribut Humidity	12
2.5	Tabel Probabilitas dari Atribut Wind	13
2.6	Tabel Dataset Mata Pelajaran	15
2.7	Hasil Pengelompokan Awal	16
2.8	Mencari Centroid Kelompok	16
2.9	Hasil Cluster Baru	17
2.10	Euclidean Distance Cluster 1, Cluster 2	17
2.11	Hasil Pengelompokan Akhir	17
2.12	Perbandingan Konsep Supresi, Generalisasi	22
3.1	Tabel Hasil Clustering Data pada Cluster 1	55
3.2	Data Credit Score	56
3.3	Hasil Pengelompokan Sementara (Greedy k-member clustering)	57
3.4	Hasil Pengelompokan Akhir (Greedy k-member clustering)	57
3.5	Hasil Pengelompokan Data (Greedy k-member clustering)	59
3.6	Hasil Anonimisasi Data (K-Anonymity)	59
5.1	Pengujian Kualitas Informasi	139
5.2	Pengujian Hasil Data Mining	139
5.3	Konfigurasi Pengujian	140
5.4	Konfigurasi Pengujian	140
5.5	Sampel Data Credit Score(Numerik)	141
5.6	Sampel Data Credit Score(Kategorikal)	141
5.7	Sampel Data Credit Score(Campuran)	141
5.8	Sampel Data Credit score ($ QID =2$)	143
5.9	Sampel Data Credit score ($ QID =3$)	143
5.10	Sampel Data Credit score ($ QID =4$)	143
5.11	Sampel Data Credit score (10k dan 30k)	145
5.12	Kesimpulan Pengujian Eksperimental	150

¹

BAB 1

²

PENDAHULUAN

³ 1.1 Latar Belakang

⁴ *Data mining* umumnya digunakan untuk menganalisis pola-pola dari data yang dikumpulkan. Untuk
⁵ mendapatkan hasil yang valid, data yang dianalisis harus sangat banyak. Oleh karena itu, teknologi
⁶ *big data* muncul untuk menangani masalah tersebut. Menurut Gartner, salah satu perusahaan riset
⁷ teknologi informasi di Amerika Serikat, *big data* adalah aset informasi bervolume tinggi,
⁸ cepat, dan beragam yang menuntut bentuk pemrosesan informasi yang hemat biaya dan inovatif
⁹ untuk meningkatkan wawasan dan pengambilan keputusan¹. Teknologi *big data* dapat membagi
¹⁰ pekerjaan pengolahan data ke beberapa komputer menggunakan konsep sistem terdistribusi. Sistem
¹¹ terdistribusi adalah solusi pengolahan *big data* karena terbukti dapat mengurangi biaya penyimpanan
¹² dan komputasi data dari pemrosesan data secara paralel.

¹³ Privasi adalah hak seseorang untuk memiliki kendali atas bagaimana informasi pribadi dikum-
¹⁴ pulkan dan digunakan. Privasi dapat terlanggar saat dilakukan proses *data mining*. *Privacy*
¹⁵ *Preserving Data Mining* (PPDM) berperan penting untuk memberi perlindungan privasi dalam
¹⁶ proses *data mining*. Konsep PPDM dapat dicapai dengan metode enkripsi dan anonimisasi. Menurut
¹⁷ Gartner, enkripsi adalah proses pengkodean aliran bit secara sistematis sebelum dilakukan transmisi
¹⁸ sehingga pihak yang tidak berwenang tidak dapat mengetahui arti pesan sebenarnya². Anonimisasi
¹⁹ adalah metode yang menyamarkan satu atau lebih nilai kolom data agar sebuah data tidak dapat
²⁰ saling dibedakan dengan data lainnya. Metode anonimisasi lebih unggul karena tidak perlu membuat
²¹ kunci untuk menjaga privasi data. Metode anonimisasi dapat diterapkan pada data *credit score*
²² yang menyimpan informasi pribadi dari pendaftar kartu kredit.

²³ Umumnya data yang disamarkan dengan metode anonimisasi mengalami kehilangan informasi
²⁴ yang cukup besar. Istilah ini lebih sering dikenal dengan nama *total information loss*. Hal ini
²⁵ mengakibatkan data yang telah dianonimisasi memiliki hasil prediksi atau pengelompokan data yang
²⁶ buruk jika mengguna model *data mining* klasifikasi/*clustering*. Salah satu cara untuk mencegah
²⁷ hal tersebut adalah dengan melakukan pengelompokan data terlebih dahulu sebelum dilakukan
²⁸ anonimisasi. Contoh algoritma pengelompokan data yang ingin diuji adalah *greedy k-member*
²⁹ *clustering*. Algoritma ini dipilih karena dapat mencari solusi paling optimal untuk mendapatkan
³⁰ kelompok data dengan nilai *information loss* yang rendah. Metode anonimisasi PPDM yang
³¹ digunakan adalah *k-anonymity*. Metode ini menjaga sebuah data tidak dapat dibedakan dengan
³² *k* – 1 data lainnya. Karena penelitian berkaitan dengan big data dan membutuhkan implementasi

¹<https://www.gartner.com/en/information-technology/glossary/big-data>

²<https://www.gartner.com/en/information-technology/glossary/encryption>

1 algoritma secara iteratif maka diperlukan teknologi untuk menangani komputasi secara paralel untuk
2 kinerja yang lebih efisien. Spark merupakan pilihan yang tepat untuk pemrosesan *big data*, karena
3 dapat memanfaatkan komputasi memori untuk komputasi yang lebih cepat dalam implementasi
4 algoritma iteratif seperti *greedy k-member clustering*.

5 Pada skripsi ini, akan dibuat tiga jenis perangkat lunak yaitu perangkat lunak eksplorasi,
6 perangkat lunak anonimisasi, dan perangkat lunak pengujian. Perangkat lunak eksplorasi bertujuan
7 mencari nilai unik untuk pembuatan pohon generalisasi yang digunakan ketika menghitung jarak
8 terdekat antar data kategorikal. Perangkat lunak anonimisasi bertujuan mengimplementasikan
9 proses pengelompokan data dengan algoritma *greedy k-member clustering* dan proses anonimisasi
10 data dengan metode *k-anonymity*. Perangkat lunak pengujian bertujuan membandingkan kualitas
11 hasil data mining sebelum dan setelah data dianonimisasi. Ketiga perangkat lunak ini dibuat dengan
12 bahasa Scala, berjalan di atas Spark, dan menggunakan penyimpanan HDFS untuk menyimpan
13 hasil komputasi sementara dari algoritma *greedy k-member clustering*. Algoritma *greedy k-member*
14 *clustering* dinilai tepat melakukan pengelompokan data karena terbukti pada penelitian sebelumnya
15 dapat meminimalkan *total information loss*. Kedua jenis perangkat lunak ini menerima data input
16 dalam format CSV. Penelitian ini memiliki tujuan utama yaitu membandingkan *silhouette score*
17 untuk model *clustering*, tingkat akurasi untuk model klasifikasi, mencari parameter terbaik pada
18 model *clustering* dan klasifikasi, dan terakhir mencari perbedaan hasil *clustering* dan klasifikasi
19 sebelum dan setelah dilakukan proses anonimisasi data.

20 1.2 Rumusan Masalah

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

- 22 1. Bagaimana cara kerja algoritma *greedy k-member clustering* untuk pengelompokan data?
- 23 2. Bagaimana cara kerja algoritma *k-anonymity* untuk anonimisasi data?
- 24 3. Bagaimana implementasi algoritma *greedy k-member clustering* pada Spark?
- 25 4. Bagaimana performa algoritma *greedy k-member clustering* dan *k-anonymity* untuk lingkungan
26 *big data* pada *hadoop cluster*?
- 27 5. Bagaimana performa pemodelan *data mining clustering* (*k-means*) dan klasifikasi (*naive bayes*)
28 untuk lingkungan *big data* pada komputer lokal?
- 29 6. Bagaimana kualitas informasi pada data yang telah dikelompokan dengan algoritma *greedy*
30 *k-member clustering* berdasarkan total information loss?
- 31 7. Bagaimana analisis kualitas hasil *data mining* terhadap pemodelan *clustering* sebelum dan
32 setelah dilakukan anonimisasi?
- 33 8. Bagaimana analisis kualitas hasil *data mining* terhadap pemodelan klasifikasi sebelum dan
34 setelah dilakukan anonimisasi?

1 1.3 Tujuan

- 2 Berdasarkan rumusan masalah di atas, tujuan dari skripsi ini adalah sebagai berikut:
- 3 1. Mempelajari cara kerja algoritma *greedy k-member clustering* untuk pengelompokan data.
- 4 2. Mempelajari cara kerja algoritma *k-anonymity* untuk anonimisasi data.
- 5 3. Mengimplementasi algoritma *greedy k-member clustering* dan *k-anonymity* pada Spark.
- 6 4. Menganalisis performa dari algoritma *greedy k-member clustering* dan *k-anonymity* untuk
- 7 lingkungan *big data* pada *hadoop cluster*.
- 8 5. Menganalisis performa dari pemodelan data mining *clustering* (k-means) dan klasifikasi (naive
- 9 bayes) untuk lingkungan *big data* pada komputer lokal.
- 10 6. Menganalisis kualitas informasi pada data yang telah dikelompokan dengan algoritma *greedy*
- 11 *k-member clustering* berdasarkan *total information loss*.
- 12 7. Menganalisis kualitas hasil metode *data mining clustering* berdasarkan *silhouette score*,
- 13 mencari parameter *clustering* terbaik, dan mencari persentase perbedaan hasil *clustering*
- 14 terbaik sebelum dan setelah dilakukan anonimisasi.
- 15 8. Menganalisis kualitas hasil metode *data mining klasifikasi* berdasarkan tingkat akurasi, mencari
- 16 parameter klasifikasi terbaik, dan mencari persentase perbedaan hasil klasifikasi terbaik
- 17 sebelum dan setelah dilakukan anonimisasi.

18 1.4 Batasan Masalah

- 19 Batasan masalah pada pengerjaan skripsi ini adalah sebagai berikut:
- 20 1. Perangkat lunak hanya menerima masukan dalam format JSON. Format masukan JSON
- 21 untuk masing-masing perangkat lunak berbeda dan dapat diisi sesuai kebutuhan. Contoh
- 22 format JSON untuk masing-masing perangkat lunak telah dicatumkan pada Subbab 4.3
- 23 2. Perangkat lunak hanya dapat mengeluarkan output dalam format CSV. Folder penyimpanan
- 24 output perangkat lunak dapat diubah pada data JSON.
- 25 3. Perangkat lunak anonimisasi dapat melakukan pengelompokan dan anonimisasi data hingga
- 26 100.000 data dengan waktu komputasi selama 2 hari. Disarankan untuk menggunakan jumlah
- 27 data dibawah 10.000 data untuk mendapat estimasi waktu komputasi kurang dari 5 jam.

28 1.5 Metodologi

- 29 Bagian-bagian pengerjaan skripsi ini adalah sebagai berikut:
- 30 1. Mempelajari dasar-dasar privasi data.
- 31 2. Mempelajari konsep *k-anonymity* pada algoritma *greedy k-member clustering*.

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

10 1.6 Sistematika Pembahasan

11 Pengeraaan skripsi ini tersusun atas enam bab sebagai berikut:

- 12 • Bab 1 Pendahuluan
Berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan.
- 13 • 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.
- 14 • 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).
- 15 • Bab 4 Perancangan
Berisi perancangan antarmuka perangkat lunak anonimisasi data dan analisis data, diagram kelas lengkap, masukan perangkat lunak anonimisasi data dan analisis data.
- 16 • Bab 5 Implementasi dan Pengujian
Berisi implementasi perangkat lunak anonimisasi data dan analisis data, pengujian fungsional, pengujian eksperimental, dan melakukan analisis terhadap hasil pengujian.
- 17 • Bab 6 Kesimpulan dan Saran
Berisi kesimpulan penelitian dan saran untuk penelitian selanjutnya.

1

BAB 2

2

LANDASAN TEORI

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

6 2.1 Privasi

- 7 Privasi adalah hak untuk dibiarkan sendiri, atau bebas dari gangguan atau gangguan ¹. Informasi
8 privasi adalah hak untuk memiliki kendali atas bagaimana informasi pribadi Anda dikumpulkan
9 dan digunakan. Privasi dapat berarti kemampuan satu atau sekelompok individu untuk menutupi
10 atau melindungi kehidupan dan urusan personalnya dari publik dengan mengontrol sumber-sumber
11 informasi mengenai diri mereka. Saat melakukan proses *data mining*, digunakan teknik anonimisasi
12 data untuk menyamarkan atribut sensitif untuk setiap data.

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

21

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

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

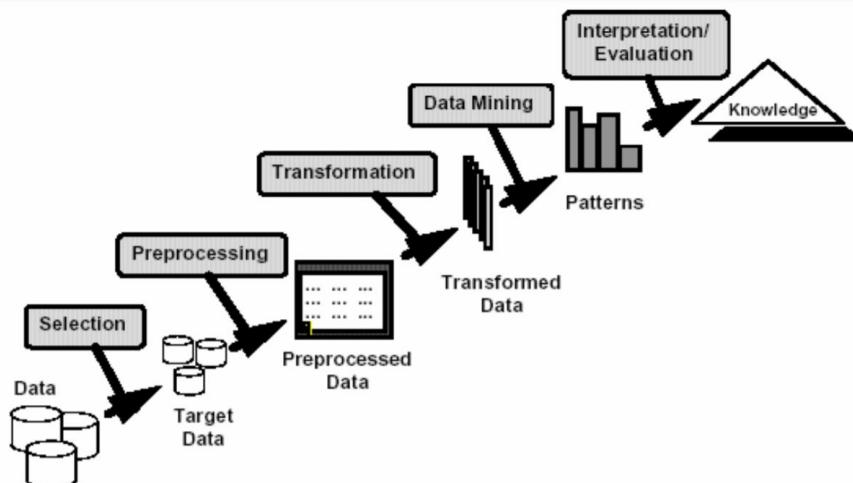
¹<https://iapp.org/about/what-is-privacy/>

- 1 Berikut adalah contoh informasi yang bersifat non-sensitif menurut standar PII:
- 2 • Rekaman medis
- 3 • Riwayat pendidikan
- 4 • Riwayat pekerjaan
- 5 • Informasi finasial
- 6 • Letak geografis

7 **2.2 Data Mining**

8 *Data mining* adalah proses menemukan pola yang menarik dan pengetahuan dari data dalam
 9 jumlah besar [1]. *Data mining* merupakan bagian dari *Knowledge Discovery in Databases* (KDD).
 10 *Knowledge Discovery in Databases* (KDD) adalah proses transformasi sekumpulan data menjadi
 11 informasi yang berguna dan dapat dimengerti.

12



Gambar 2.1: Tahapan pada KDD

- 13 Berikut ini adalah penjelasan tahapan pada KDD pada Gambar 2.1 sebagai berikut:
- 14 1. *Selection*: penyeleksian atau segmentasi data berdasarkan kriteria tertentu.
- 15 2. *Preprocessing*: tahap pembersihan dimana informasi yang tidak berguna akan dibuang. Pada
 16 tahap ini data dikonfigurasi ulang untuk menjamin format tetap konsisten.
- 17 3. *Transformation*: proses manipulasi data menggunakan konsep agregasi, generalisasi, normali-
 18 sasi, dan reduksi untuk kebutuhan analisis.
- 19 4. *Data mining*: proses ekstraksi pola dari data yang ada, contohnya model klasifikasi, *clustering*.
- 20 5. *Interpretation/evaluation*: proses interpretasi pola menjadi pengetahuan yang dapat digunakan
 21 untuk mendukung pengambilan keputusan

- 1 Berikut adalah beberapa jenis tipe data terkait hasil data mining:
- 2
 - *Binary*: tipe data kategorikal/numerik yang hanya memiliki 2 jenis kemungkinan nilai.
Contoh: nilai true/false dan 0/1.
 - *Nominal*: tipe data kategorikal/numerik yang memiliki lebih dari 2 jenis kemungkinan nilai.
Contoh: warna kuning, hijau, hitam, merah.
- 6 Tujuan dari penggunaan teknik *data mining* adalah sebagai berikut:
 - 7 • Prediksi: proses menggunakan nilai dari beberapa atribut yang sudah ada untuk memprediksi data baru. Contoh: klasifikasi.
 - 9 • Deskripsi: proses menemukan pola yang dapat merepresentasikan kelompok dari sebuah data.
Contoh: *clustering*.

11 2.2.1 Klasifikasi

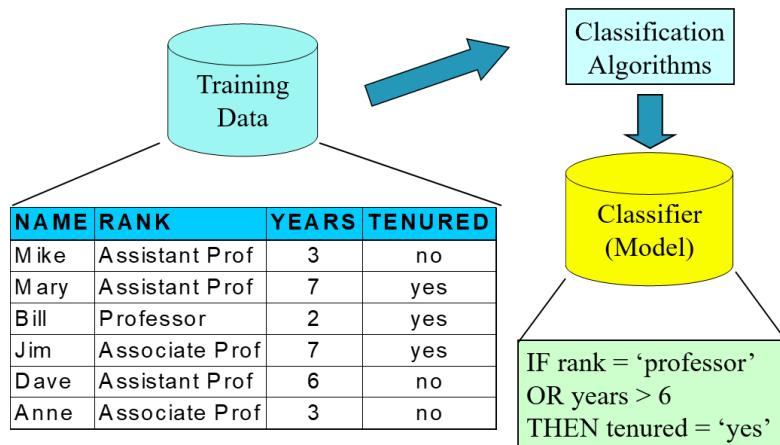
12 Klasifikasi adalah proses menemukan model (atau fungsi) yang menggambarkan dan membedakan
13 kelas data [1]. Model klasifikasi digunakan untuk memprediksi label kelas dari objek yang label
14 kelasnya tidak diketahui. Model klasifikasi merupakan bagian dari *supervised learning*, dimana pela-
15 tihan model dilakukan dengan memakai data yang label kelasnya diketahui. Untuk mempermudah
16 penjelasan mengenai *supervised learning*, maka diberikan ilustrasi sebagai berikut.

Jumlah Kaki	Punya Sayap?	Tinggi (Cm)	Makanan	Jenis
2	Y	12.3	buah	burung kutilang
4	T	23	campuran ikan	kucing
4	T	134	campuran tumbuhan	sapi
2	Y	3	biji-bijian	burung gereja
0	T	2	binatang kecil	ular
4	T	0.6	serangga	cicak

Gambar 2.2: Ilustrasi Supervised Learning

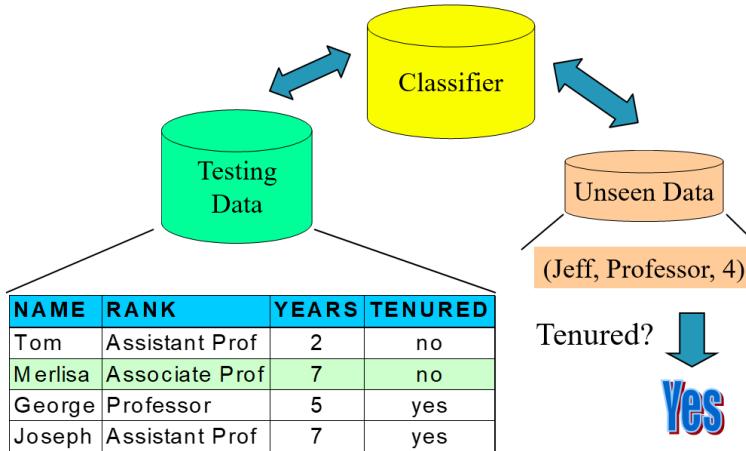
17 Gambar 2.2 adalah contoh ilustrasi yang diambil dari buku pengantar data science untuk menge-
18 nalkan contoh data yang dipakai *supervised learning*, dimana data mengandung kolom prediktor dan
19 kolom kelas. Jika model klasifikasi dilatih dengan data tersebut, nantinya model klasifikasi dapat di-
20 gunakan untuk memprediksi jenis binatang berdasarkan atribut prediktor yang dipilih yaitu jumlah
21 kaki, punya sayap, tinggi tubuh, jenis makanan. Hasil prediksi jenis binatang yang mungkin, berni-
22 lai salah satu dari yang tercantum pada kolom jenis yaitu burung kutilang, kucing, sapi, dan lain-lain.

- 1 Gambar 2.3 adalah tahapan pelatihan model klasifikasi:



Gambar 2.3: Ilustrasi Supervised Learning

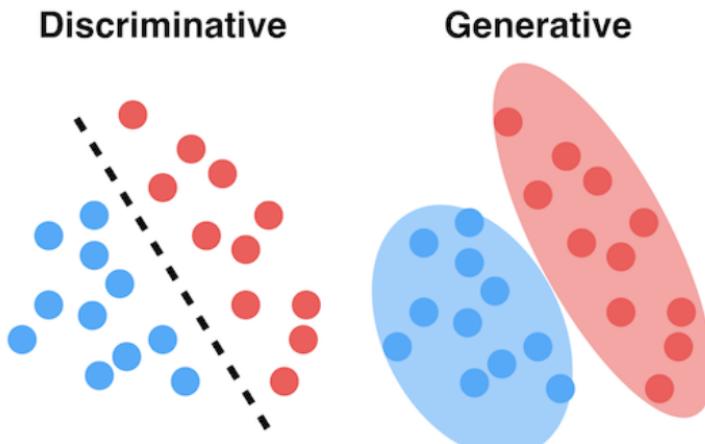
- 2 1. Mengambil data training dengan proporsi 70% dari total data keseluruhan.
- 3 2. Melakukan komputasi algoritma klasifikasi berdasarkan masukan data training.
- 4 3. Model klasifikasi dihasilkan dari komputasi algoritma klasifikasi.
- 5 4. Model klasifikasi menghasilkan aturan untuk melakukan prediksi data.
- 6 Gambar 2.4 adalah tahapan prediksi model klasifikasi:



Gambar 2.4: Ilustrasi Supervised Learning

- 7 1. Mengambil data testing dengan proporsi 30% dari total data keseluruhan.
- 8 2. Mengambil model klasifikasi dari tahapan pelatihan sebelumnya.
- 9 3. Memberikan data baru kepada model klasifikasi untuk dilakukan prediksi.
- 10 4. Hasil prediksi dari model klasifikasi berupa label kelas dari kolom tertentu

- 1 Berikut adalah kategori pemodelan klasifikasi :



Gambar 2.5: Contoh *Logistic Regression*

- 2 • *Discriminative*: model yang mempelajari batas pemberian label dalam kumpulan data.
- 3 Tujuannya untuk menemukan batasan keputusan yang dapat memisahkan satu kelas data
- 4 dari kelas data yang lain. Model ini memisahkan kelas berdasarkan probabilitas bersyarat.
- 5 Contoh dari model *discriminative* adalah *logistic regression*, *decision tree*, *random forest*.
- 6 • *Generative*: model yang mempelajari distribusi kelas individu dalam kumpulan data. Model
- 7 ini menggunakan estimasi probabilitas yang memungkinkan memodelkan sebuah data dengan
- 8 membuat label kelas yang berbeda untuk masing-masing data. Contoh dari model *generative*
- 9 adalah *naive bayes*.

- 10 Berikut adalah penjelasan model klasifikasi dari naive bayes:

11
12 *Naive Bayes* menerapkan klasifikasi dengan menggunakan metode probabilitas dan statistik. Pe-
13 modelan ini mencari nilai probabilitas tertinggi pada masing-masing kelas menggunakan teorema
14 *Bayes*. Kelas dengan probabilitas tertinggi akan dipilih sebagai hasil akhir. *Naive Bayes* mudah
15 untuk dibangun dan memiliki komputasi yang lebih cepat daripada model klasifikasi lainnya.
16
17 Teorema Bayes dapat digunakan untuk menghitung probabilitas kemunculan label terhadap sebuah
18 kejadian berdasarkan informasi yang telah diberikan. Teorema Bayes dapat dinyatakan secara
19 matematis melalui persamaan berikut

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

- 20 Hasil prediksi dari model *naive bayes* dapat dinyatakan sebagai label yang memiliki hasil proba-
21 bilitas teorema bayes tertinggi terhadap sebuah kejadian. Probabilitas tertinggi teorema bayes
22 dapat dinyatakan secara matematis melalui persamaan berikut:

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

- ¹ Keterangan:
- ² • $P(H|D)$ adalah probabilitas terjadi hipotesis H dan diketahui kejadian D .
 - ³ • $P(D|H)$ adalah probabilitas terjadi kejadian D dan diketahui hipotesis H .
 - ⁴ • $P(H)$ adalah probabilitas hipotesis H
 - ⁵ • $P(D)$ adalah probabilitas seluruh kejadian.
- ⁶ Pada bagian ini, dijelaskan lebih detil mengenai contoh perhitungan probabilitas teorema bayes
⁷ untuk menentukan prediksi kelas. Tabel 2.12 adalah data yang berisi kondisi cuaca saat bermain
⁸ golf. Data ini memiliki label kelas dengan nama *PlayGolf* yang menyatakan kondisi yang cocok
⁹ dalam bermain golf.

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

- ¹⁰ Berikut adalah penentuan atribut klasifikasi pada data PlayGolf:
- ¹¹ • Atribut prediktor
¹² Atribut prediktor adalah atribut yang digunakan untuk memprediksi label kelas. Atribut
¹³ prediktor pada contoh kasus ini terdiri dari atribut *Outlook*, *Temperature*, *Humidity*, dan
¹⁴ *Windy*.
 - ¹⁵ • Atribut kelas
¹⁶ Atribut kelas adalah atribut yang berisi nilai label kelas. Atribut kelas pada contoh kasus ini
¹⁷ diwakili oleh atribut *PlayGolf*.
- ¹⁸ Secara singkat, langkah kerja algoritma *Naive Bayes* dapat dijelaskan sebagai berikut².

²<https://www.geeksforgeeks.org/naive-bayes-classifiers/>

1 1. Merepresentasikan teorema *Bayes* terhadap vektor fitur.

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

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

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

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

5 Contoh: $X = (\text{Rainy}, \text{Hot}, \text{High}, \text{False})$, $y = \text{No}$

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

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

9 2. Menghitung probabilitas nilai dari sebuah atribut.

10 Sebagai contoh, nilai yang dimiliki atribut *Outlook* adalah $\{\text{Sunny}, \text{Overcast}, \text{Rainy}\}$. Tabel 2.2
11 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 $P(\text{No})$ untuk nilai *Sunny* pada atribut *Outlook*

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

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

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

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

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

16 4. Menghitung probabilitas bersyarat terhadap data baru yang diberikan.

17 Contoh: *today* = (*Sunny*, *Hot*, *Normal*, *NoWind*)

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

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

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

¹ 5. Melakukan normalisasi terhadap probabilitas besyarat.

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

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

² Ketika probabilitas tersebut dijumlahkan, maka hasilnya akan menjadi 1.

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

³ 6. Mencari probabilitas tertinggi.

⁴

Berdasarkan pernyataan berikut:

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

Tabel 2.5: Tabel Probabilitas dari Atribut Wind

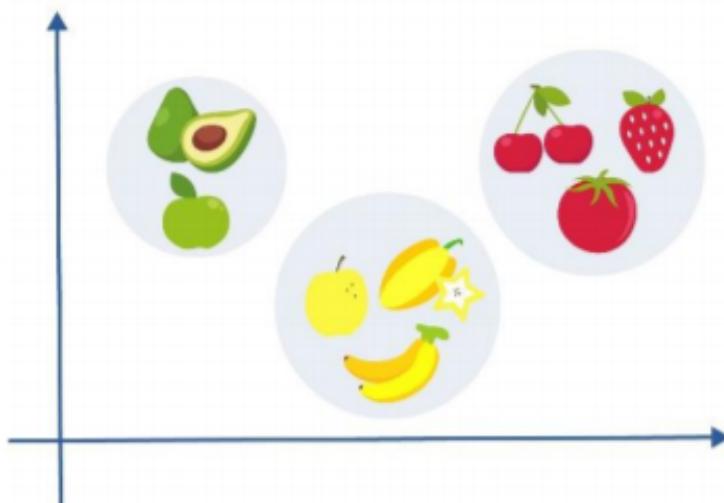
	Yes	No	P(Yes)	P(No)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100	100

1 Dapat disimpulkan bahwa data dengan nilai (*Sunny, Hot, Normal, NoWind*) dapat diklasifi-
 2 kasikan terhadap atribut *PlayGolf* dengan nilai label kelas Yes.

3 2.2.2 Clustering

4 *Clustering* adalah proses mempartisi sekumpulan objek data (atau observasi) menjadi subset. Setiap
 5 subset dapat disebut sebagai *cluster* [1]. Anggota di dalam suatu cluster memiliki tingkat kesamaan
 6 sifat atau fitur yang tinggi dengan anggota lainnya. Sebaliknya, tingkat kesamaan sifat atau fitur
 7 anggota suatu *cluster*, bernilai rendah dengan sifat atau fitur anggota *cluster* lain. Pada *data*
 8 *mining*, *clustering* digunakan untuk mendapatkan pemahaman terkait distribusi data, mengobservasi
 9 karakteristik tiap *cluster*, dan berfokus pada *cluster* tertentu saja untuk analisis lebih lanjut. Metode
 10 *clustering* dipakai apabila dataset tidak memiliki atribut label kelas, dengan kata lain metode ini
 11 merupakan bagian dari *unsupervised learning*. Untuk mempermudah penjelasan mengenai metode
 12 *clustering* diberikan ilustrasi sebagai berikut.

13

Gambar 2.6: Contoh *Hierarchical Clustering*

14 Gambar 2.7 adalah contoh kasus *clustering* sederhana[2]. Pada kasus ini, sekelompok buah ingin
 15 dikelompokkan ke dalam beberapa *cluster*. Salah satu kemiripan sifat yang ada pada buah adalah
 16 warna. Pada kasus ini, beberapa buah dengan warna mirip dikelompokkan menjadi satu *cluster*.
 17 Misalnya, alpukat dan apel hijau dikelompokkan menjadi satu *cluster* karena berwarna hijau. Jika
 18 buah dikelompokkan berdasarkan sifat atau fitur lain misalnya ukuran, maka hasil *cluster* yang
 19 didapat akan berbeda dari sebelumnya. Dari pernyataan ini dapat disimpulkan bahwa pemilihan
 20 fitur dapat mempengaruhi hasil *clustering*.

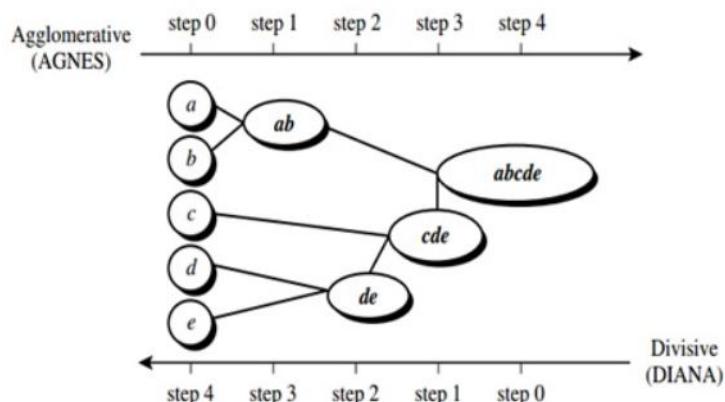
- 1
2 Berikut adalah tahapan *clustering* secara umum:
- 3 1. Perhitungan *distance*: proses untuk mengukur kesamaan antar objek dengan cara menghitung
4 *distance* antar 2 titik. Salah satu contoh metode pengukuran *distance* yang umum digunakan
5 adalah *Euclidean distance*. Metode ini terdiri atas variabel p_i , menyatakan kordinat titik data
6 dan variabel C_i , menyatakan kordinat titik *centroid* sebuah *cluster*.
- 7 Berikut adalah persamaan untuk menghitung *Euclidean distance*:

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

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

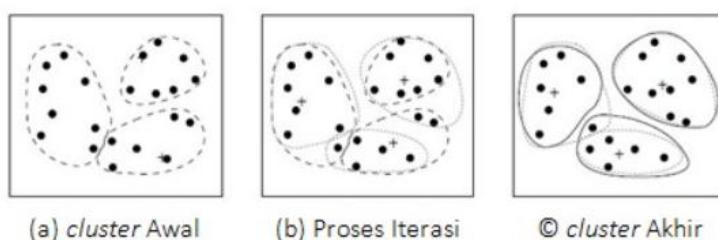
10 Berikut adalah kategori pemodelan *clustering*:

- 11 • *Hierarchical clustering*: pengelompokan data pada sebuah hirarki dengan cara menggabung
12 dua kelompok data terdekat maupun membagi data ke dalam *cluster* tertentu. Contoh
13 pemodelannya adalah *agglomerative*.



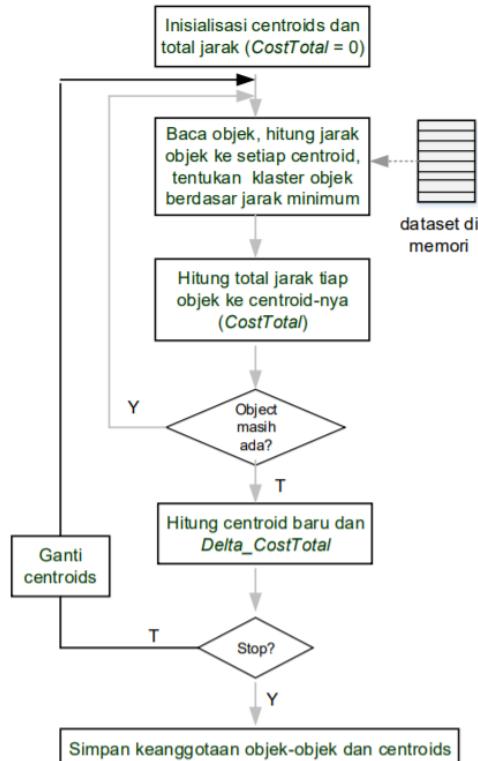
Gambar 2.7: Contoh *Hierarchical Clustering*

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



Gambar 2.8: Contoh *Partitional Clustering*

- 1 Berikut adalah penjelasan model clustering dari *k-means*:
- 2
- 3 *K-means* merupakan salah contoh pemodelan *clustering*. Algoritma ini menerima masukan himpunan data berformat vektor, dimana tiap objek memiliki fitur-fitur dan tiap fitur memiliki nilai yang merepresentasikan objek tersebut. *K-means* membutuhkan masukan nilai *k* yang menyatakan jumlah *cluster* yang ingin dibentuk. Untuk mempermudah penjelasan langkah kerja *k-means*, diberikan ilustrasi pada Gambar 2.9.

Gambar 2.9: Contoh *Partitional Clustering*

- 8 Pada bagian ini dijelaskan lebih detil mengenai contoh perhitungan jarak untuk mengelompokan data dengan *cluster* terdekat. Tabel 2.6 diberikan untuk mengelompokan mata pelajaran yang 10 sejenis berdasarkan data skor yang diperoleh dari individu A dan B. Data³ ini tidak memiliki kolom 11 label sehingga cara satu-satunya adalah membuat model *clustering*.

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

³<http://mnemstudio.org/clustering-k-means-example-1.htm>

- 1 Secara singkat, langkah kerja algoritma *k-means* dapat dijelaskan sebagai berikut:
- 2 1. Inisialisasi *k* dan titik *centroid* awal.
- 3 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.

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

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

7 Titik data akan dikelompokan ke *centroid* terdekat dengan memilih nilai *Euclidian distance* 8 paling kecil. Sebagai contoh, Tabel 2.8 pada Step 2 menyatakan data ke-1 lebih dekat dengan 9 data ke-2, karena memiliki nilai *Euclidean distance* paling kecil. Karena itu, data ke-2 10 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.

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

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

2 Iterasi paling terakhir di langkah sebelumnya akan dijadikan sebagai anggota dari *cluster* baru.
 3 Tabel 2.9 menunjukkan kelompok data sementara yang terbentuk pada masing-masing *cluster*.
 4 *Cluster 1* terdiri dari anggota data $\{1, 2, 3\}$, sedangkan *Cluster 2* terdiri dari anggota data
 5 $\{4, 5, 6, 7\}$. 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)

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

7 Belum dipastikan bahwa setiap individu telah dialokasikan dengan tepat. Oleh karena itu,
 8 langkah 1 sampai 4 perlu kembali diulang dengan menghitung kembali *Euclidean distance*.
 9 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

- 10 6. Menetapkan kelompok data akhir.

11 Apabila *cluster* tidak mengalami perubahan secara signifikan pada anggotanya selama periode
 12 iterasi tertentu, maka *cluster* yang terbentuk sudah sesuai. Tabel 2.11 menunjukkan anggota
 13 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 [1]

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

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

2.3.2 Menghitung *Silhouette Score* untuk Model *Clustering*

Hasil pemodelan *clustering* dapat dievaluasi menggunakan perhitungan *silhouette score*. *Silhouette score* bertujuan untuk menghitung seberapa dekat sebuah objek dengan *intra-cluster* dan mengukur seberapa jauh objek yang sama dengan *cluster* terdekat lainnya. Semakin tinggi nilai *silhouette score*, maka hasil pengelompokannya akan semakin baik. *Silhouette score* 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 nilai *silhouette score* 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 dikelompokan pada *cluster* yang salah.

Berikut adalah persamaan untuk menghitung *silhouette score* [1]:

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

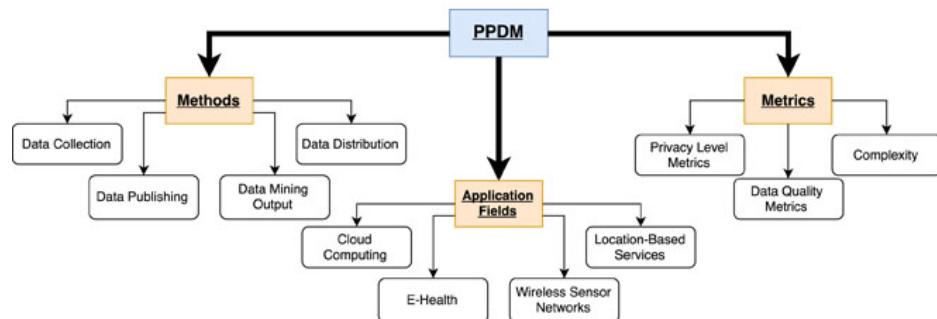
Keterangan:

- 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)

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 [6]. 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 2.12 menjelaskan garis besar pokok bahasan 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 2.10: Privacy Preserving Data Mining (PPDM)

2.5 Metode Anonimisasi

Anonimisasi adalah cara melindungi data pribadi atau sensitif dengan menghapus atau mengenkripsi informasi yang dapat diidentifikasi secara pribadi [6]. Anonimisasi data bertujuan melindungi aktivitas pribadi individu atau perusahaan sambil menjaga integritas data yang dikumpulkan dan dibagikan. Namun, dengan menghapus PII saja masih belum cukup karena penyerang masih dapat menggunakan metode de-anonimisasi untuk menelusuri kembali proses anonimisasi data. Karena data biasanya dibagikan melalui berbagai sumber, maka teknik de-anonimisasi dapat mereferensikan sumber dan mengungkapkan informasi pribadi.

Anonimisasi data dapat dilakukan dengan penghapusan, enkripsi, generalisasi, dan lain-lain. Perusahaan dapat menghapus informasi identitas pribadi (PII) atau mengenkripsi informasi ini dengan sandi yang kuat. Cara lain yang lebih efektif adalah menggunakan metode generalisasi informasi. Misalnya, tabel berisi pendapatan kotor yang diperoleh pekerja di sektor ritel. Diasumsikan pendapatan yang dicatat adalah \$520.000, \$230.000, \$109.000, \$875.000, dan \$124.000. Informasi ini dapat digeneralisasi menjadi "< \$500.000" dan ">= \$500.000". Meskipun datanya dikaburkan, data masih memiliki informasi secara umum.

¹ 2.6 *K-Anonymity*

² *K-anonymity* adalah konsep anonimisasi yang diperkenalkan PPDM untuk mengatasi risiko identifikasi ulang data yang dianonimkan melalui tautan ke kumpulan data lain⁴. Model privasi ³ *k-anonymity* pertama kali diusulkan pada tahun 1998 oleh Latanya Sweeney melalui penelitiannya. ⁴ Untuk melakukan metode *k-anonymity*, setidaknya *k* individu dalam kumpulan data harus memiliki ⁵ nilai atribut yang unik agar setiap individu dapat dikenali. Konsep *k-anonymity* menyebabkan ⁶ sebuah data tidak dapat dibedakan setidaknya dengan *k* – 1 data lainnya. ⁷

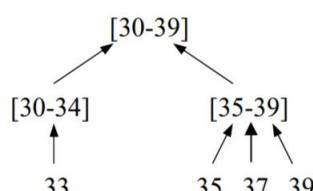
⁸

⁹ Berikut adalah atribut dari pemodelan *k-anonymity*, yaitu:

- ¹⁰ • *Identifier* (ID) adalah atribut yang unik dan secara langsung dapat mengidentifikasi seseorang seperti nama, nomor ID, dan nomor ponsel.
- ¹¹ • *Quasi-identifier* (QID) adalah kombinasi atribut yang mungkin terjadi untuk mengidentifikasi individu berdasarkan penggabungan informasi lain dari luar. Seluruh atribut data terkecuali ¹² atribut identifier dapat dianggap sebagai atribut quasi-identifier.
- ¹³ • *Sensitive Attribute* (SA) adalah atribut yang menyangkut informasi sensitif seseorang, biasanya ¹⁴ nilai atribut ini akan dirahasiakan saat distribusi data.
- ¹⁵ • *Non-sensitive Attribute* (NSA) adalah atribut yang tidak menyangkut informasi sensitif ¹⁶ seseorang, biasanya nilai atribut ini langsung ditampilkan saat distribusi data.

¹⁷ Berikut adalah tahapan anonimisasi pada model *k-anonymity*:

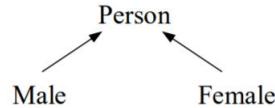
- ¹⁸ 1. Inisialisasi **result** dengan *dataframe* kosong, simpan hasil pengelompokan dalam *dataframe* pada **temp**, hitung jumlah *cluster* pada hasil pengelompokan.
- ¹⁹ 2. Pada kondisi jumlah *cluster* > 0, lakukan perulangan sebagai berikut:
 - ²⁰ (a) Mengambil seluruh anggota data sebuah *cluster* dari variabel **temp**
 - ²¹ (b) Lakukan perulangan untuk setiap kolom data sebagai berikut:
 - ²² i. Menyeleksi kolom data dengan mengambil beberapa atribut *quasi-identifier* yang dipilih sebelumnya.
 - ²³ ii. Jika kolom bertipe numerik, maka lakukan pencarian nilai maksimum dan minimum dari kolom tersebut untuk dilakukan anonimisasi berdasarkan rentang nilai. Misalnya pada kolom *Age*, nilai maksimum = 50 dan nilai minimum = 20, sehingga hasil anonimisasinya adalah [25-50].



Gambar 2.11: *Privacy Preserving Data Mining* (PPDM)

⁴<https://www.privitar.com/blog/k-anonymity-an-introduction/>

- 1 iii. Jika kolom bertipe kategorikal, maka lakukan pencarian nilai *root* terhadap pohon
 2 DGH untuk kolom tersebut. Misalnya pada kolom *Occupation*, diketahui bahwa nilai
 3 *root* untuk kolom *Occupation* adalah *Employee*, sehingga hasil anonimisasi kolom
 4 tersebut adalah *Employee*.



Gambar 2.12: Privacy Preserving Data Mining (PPDM)

- 5 (c) Karena seluruh kolom pada sebuah *cluster* berhasil dianonimisasi, langkah selanjutnya
 6 adalah menyimpan hasil anonimisasi pada variabel **result**.
 7 (d) Lakukan operasi SQL `except()` pada variabel temp untuk menghilangkan *cluster* yang
 8 pernah dilakukan proses anonimisasi di tahap sebelumnya.
 9 (e) Mengurang jumlah *cluster* dengan nilai 1 untuk setiap iterasi.
 10 3. Algoritma ini mengembalikan himpunan tabel anonimisasi dari seluruh *cluster* dengan pe-
 11 manggilan variabel **result**.

12 Meskipun *k-anonymity* dapat memberikan keamanan privasi, metode ini tetap dilengkapi dengan
 13 ketentuan berikut:

- 14 • **Kolom sensitif tidak boleh mengungkapkan identitas seseorang.**

15 Misalnya, penyakit tertentu bersifat unik untuk pria atau wanita yang kemudian dapat
 16 mengungkapkan atribut *Gender* yang dihapus.

- 17 • **Nilai di kolom sensitif tidak semuanya sama untuk grup k tertentu.**

18 Jika semua nilai sensitif sama untuk sekumpulan *k* record yang memiliki atribut kuasi-
 19 identifikasi, maka dataset ini masih rentan terhadap apa yang disebut serangan homogenitas.
 20 Dalam serangan homogenitas, penyerang memanfaatkan fakta bahwa cukup untuk menemukan
 21 grup record yang dimiliki individu jika semuanya memiliki nilai sensitif yang sama.

- 22 • **Jika keragaman tidak cukup maka identitas individu dapat dipelajari.**

23 Jika sekitar 90 persen dari semua data dalam kelompok memiliki nilai sensitif yang sama,
 24 penyerang setidaknya dapat menyimpulkan dengan pasti apa atribut sensitif individu tersebut.
 25 Penggunaan metode seperti *l-diversity* dan *t-closeness* dapat digunakan untuk menentukan
 26 bahwa di antara *k* data yang cocok harus ada sejumlah keragaman di antara nilai-nilai sensitif.

- 27 • **Dimensi data harus cukup rendah.**

28 Jika data berdimensi tinggi, seperti data deret waktu, memberikan jaminan privasi yang sama
 29 seperti data berdimensi rendah menjadi cukup sulit. Untuk jenis data seperti data transaksi
 30 atau lokasi, dimungkinkan untuk mengidentifikasi individu secara unik dengan merangkai
 31 beberapa titik data. Selain itu, seiring dengan meningkatnya dimensi data, titik data sering
 32 kali tersebar sangat jarang. Dengan menggunakan pendekatan ini maka hanya merilis kolom
 33 yang benar-benar dibutuhkan orang, dimensi dapat dikurangi ke tingkat yang dapat dikelola.

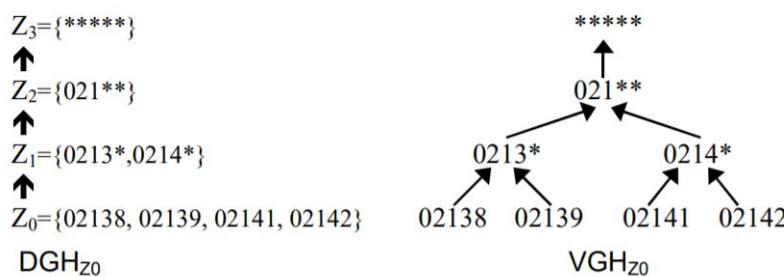
1 2.7 Domain Generalization Hierarchy (DGH)

2 Cara yang dipakai oleh metode *k-anonymity* untuk melakukan anonimisasi data adalah dengan menggunakan
 3 prinsip generalisasi dan supresi. Generalisasi adalah proses membuat nilai *quasi-identifier*
 4 menjadi kurang tepat, sehingga data dengan nilai berbeda dapat diubah (atau digeneralisasikan)
 5 menjadi data yang memiliki nilai yang sama. Ide generalisasi adalah membuat informasi menjadi
 6 kurang tepat agar memenuhi persyaratan privasi, akan tetapi tetap memungkinkan dilakukannya
 7 analisis data. *Generalization hierarchy* dapat dibuat jika beberapa data ingin dilakukan anonimisasi.

Tabel 2.12: Perbandingan Konsep Supresi, Generalisasi

Generalisasi	Supresi
merubah nilai tertentu menjadi bentuk nilai lain	merubah nilai tertentu dengan tanda bintang **
contoh: atribut umur berubah menjadi age = >50, age = <50	contoh: atribut nama berubah menjadi name = *
menderita kehilangan informasi yang cukup banyak	informasi sudah tidak dapat lagi diinterpretasi

8 *Domain Generalization Hierarchy* (DGH) adalah bagian implementasi generalization hierarchy
 9 yang menyimpan nilai generalisasi dan nilai sesungguhnya pada tingkatan domain tertentu⁵. DGH
 10 dapat menggunakan prinsip generalisasi dan supresi dalam melakukan anonimisasi. Sebagai contoh,
 11 pohon DGH untuk kode pos dibuat untuk menyamarkan satu atau lebih digit. Banyaknya digit
 12 yang disamarkan bergantung pada tingkatan nilai generalisasi yang ingin dibuat pada pohon DGH.
 13 Jika tingkatan domain semakin atas, maka nilai yang disimpan bersifat lebih umum. Sedangkan jika
 14 tingkatan domain semakin bawah, maka nilai yang disimpan semakin dekat dengan nilai sebenarnya.



Gambar 2.13: Contoh Implementas DGH,VGH (ZIP)

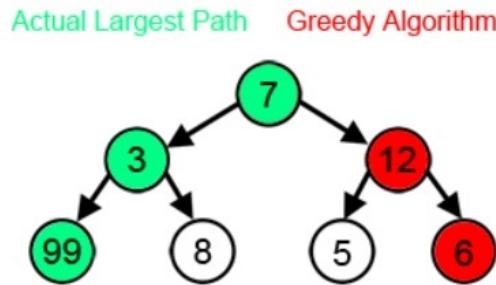
15 Gambar 3.2 menunjukkan contoh penggunaan DGH dan VGH. VGH singkatan dari *Value*
 16 *Generalization Hierarchy* dan merupakan bagian implementasi dari *generalization hierarchy*. Peng-
 17 gunaan VGH cukup jarang, sehingga pada contoh ini akan difokuskan untuk membahas DGH.
 18 Diketahui bahwa nilai ZIP {02138,02139,02141,02142} pada domain Z_0 telah dilakukan generalisasi
 19 menjadi {0213*,0214*} dengan menyamarkan satu digit di paling kanan. Diketahui bahwa nilai
 20 ZIP {0213*,0214*} pada domain Z_1 telah dilakukan generalisasi menjadi {021***}. Langkah ini
 21 dilanjutkan sampai ZIP bernilai {*****}. Melalui contoh ini dapat ditunjukan bahwa semakin atas
 22 tingkatan generalisasi maka semakin banyak informasi data yang hilang akibat proses anonimisasi.

⁵<https://sciencedirect.com/topics/computer-science/generalization-hierarchy>

¹ 2.8 Greedy K-Member Clustering

² Greedy adalah algoritma yang membuat pilihan optimal pada setiap langkah saat mencoba menemukan cara optimal secara keseluruhan untuk menyelesaikan seluruh masalah ⁶. Algoritma *greedy* cukup berhasil dalam beberapa masalah, seperti *Huffman encoding* terkait masalah kompresi data dan *Dijkstra* terkait masalah pencarian jalur terpendek.

⁶ Namun pada beberapa kasus, algoritma *greedy* tidak menghasilkan solusi yang optimal. Misalnya pada Gambar 2.14, algoritma *greedy* berupaya menemukan jalur dengan bobot paling besar. Pada tahap ini, algoritma *greedy* memilih bobot paling besar di setiap langkah sehingga gagal menemukan bobot terbesar karena hanya melihat bobot paling besar pada langkah paling awal, tanpa memperhatikan bobot paling besar secara keseluruhan.



Gambar 2.14: Kelemahan Algoritma Greedy

¹¹ *Greedy k-member clustering* merupakan bagian dari pengembangan algoritma *greedy*. Algoritma ini bertujuan melakukan pengelompokan data ke masing-masing *cluster* [8]. Selain itu, algoritma ini dapat meminimalkan nilai informasi yang hilang saat dilakukan anonimisasi data, karena data telah dikelompokan terlebih dahulu. Algoritma ini memiliki kompleksitas n^2 , sehingga memerlukan penanganan khusus jika digunakan untuk lingkungan *big data*.

¹⁶
¹⁷ Beberapa hal penting terkait algoritma *greedy k-means clustering*:

- ¹⁸ • Menetapkan tabel S
- ¹⁹ • Menetapkan nilai k
- Menetapkan jumlah *cluster* (m) yang ingin dibuat

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

²⁰ Berikut adalah langkah kerja algoritma *greedy k-means clustering* secara lengkap [8]:

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

⁶<https://brilliant.org/wiki/greedy-algorithm/>

- (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* numerik:

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

Berikut adalah rumus menghitung *distance* kategorikal:

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

- 1 (b) Membuang himpunan data variabel r pada variabel S
 2 (c) Mengisi data dari variabel r pada variabel c .
 3 (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:

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

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

- 4 ii. Membuang himpunan data dari variabel r pada variabel S .
 5 iii. Menambahkan himpunan data dari variabel r pada variabel c .
 6 iv. Menambahkan himpunan data dari variabel c pada variabel result.
 7 3. Pada kondisi $|S| \neq 0$, artinya jika masih terdapat data yang belum dimasukkan pada sebuah
 8 cluster dari tabel S , lakukan perulangan sebagai berikut:
 9 (a) Memilih data secara acak dari tabel S untuk disimpan pada variabel r .
 10 (b) Membuang himpunan data dari variabel r pada variabel S .
 11 (c) Memilih cluster terbaik untuk variabel c berdasarkan nilai *Information Loss* (IL) yang
 12 paling rendah. *Information Loss* (IL) dapat dicari menggunakan rumus berikut:

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

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

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

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

Algoritma 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 ∈ 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-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

```

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

Algoritma 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 ∈ 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-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

```

- 8 Algoritma 2 menerima input himpunan data *cluster* dan sebuah data dengan nilai *distance*
 9 tertinggi dari data terpilih acak. Algoritma ini menghitung selisih *distance* dari dua jenis data yang
 10 berbeda. Variabel *diff* pada algoritma ini adalah perbedaan *distance*, dicari dengan penjumlahan
 11 *information loss* pada sebuah *cluster* dengan *information loss* pada data ke-i, lalu hasil penjumlahan

- 1 tersebut dikurangi dengan *information loss* dari cluster. Output algoritma ini adalah data dengan
 2 nilai *cluster* terbaik, yaitu data ke-i dari dataset S dengan nilai *distance* paling kecil.

Algoritma 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:
12: while  $|S| \geq k$  do
13:     r = the furthest record from r
14:     S = S - {r}
15:     c = {r}
16:     while  $|c| < k$  do
17:         r = find_best_record(S,c)
18:         S = S - {r}
19:         c = c  $\cup$  {r}
20:     end while
21:     result = result  $\cup$  {c}
22: end while
23:
24: while  $S \neq \emptyset$  do
25:     r = a randomly picked record from S
26:     S = S - {r}
27:     c = find_best_cluster(result,r)
28:     c = c  $\cup$  {r}
29: end while
30: return result

```

- 3 Algoritma 3 menerima input himpunan data S dan nilai k . Algoritma ini mengeksekusi dua
 4 jenis fungsi yang berbeda yaitu fungsi *find_best_cluster* untuk mencari *cluster* dengan *distance*
 5 terkecil dan fungsi *find_best_record* untuk mencari data dengan *distance* terkecil. Output dari
 6 algoritma ini adalah himpunan data dari berbagai jenis *cluster* dengan nilai *distance* terkecil.

7 2.9 Metrik *Distance* dan *Information Loss*

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

1 2.9.1 *Distance*

2 *Distance* adalah perhitungan pada algoritma *greedy k-member clustering* yang bertujuan untuk
 3 mencari pengelompokan data terdekat. *Distance* yang dihitung pada algoritma *greedy k-member*
 4 *clustering* terdiri dari kolom numerik, kolom kategorikal, dan *record*. Pemilihan *distance* yang paling
 5 dekat dapat menghasilkan pengelompokan data yang lebih akurat.

6 *Distance Data Numerik*

7 *Distance* data numerik digunakan untuk menghitung jarak antar kolom numerik pada algoritma
 8 *greedy k-member clustering*. *Distance* numerik mencari rentang nilai berdasarkan perbedaan ni-
 9 lai maksimum dan minimum dari kolom numerik. *Distance* numerik menerima parameter $|D|$
 10 adalah jumlah data keseluruhan, v_1, v_2 adalah nilai kolom numerik. *Distance* numerik dihitung
 11 menggunakan rumus berikut [8]:

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

12 *Distance Data Kategorikal*

13 *Distance* kategorikal dipakai untuk menghitung jarak antar kolom kategorikal pada algoritma
 14 *greedy k-member clustering*. *Distance* kategorikal menggunakan pohon DGH dan metode *lowest*
 15 *common ancestor* untuk mencari *root* paling dekat yang mengandung kedua nilai data kategorikal.
 16 *Distance* kategorikal menerima parameter $|D|$ adalah jumlah data keseluruhan, TD adalah *pohon*
 17 *DGH* untuk domain D , $H(\Lambda(v_i, v_j))$ adalah tinggi *lowest common ancestor* yang mengandung nilai
 18 data kategorikal, $H(T_D)$ adalah tinggi dari *taxonomy tree*. *Distance* data kategorikal dihitung
 19 menggunakan rumus berikut [8]:

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

20 *Distance Record*

Distance record dipakai untuk menghitung total jarak seluruh kolom kategorikal dan kolom numerik
 pada algoritma *greedy k-member clustering*. *Distance record* menerima parameter $r_1[N_i]$, $r_2[N_i]$
 adalah nilai kolom numerik, $r_1[C_j]$, $r_2[C_j]$ adalah nilai kolom kategorikal, δ_N adalah *distance* data
 numerik, δ_C adalah *distance* kategorikal. *Distance record* dihitung dengan rumus berikut [8] :

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

21 2.9.2 *Information Loss*

22 *Information Loss* (IL) dipakai untuk mengevaluasi seberapa baik kinerja algoritma *k-anonymity*
 23 terhadap utilitas sebuah data. Dalam menghitung *Information Loss* (IL), perlu mendefinisikan
 24 beberapa atribut seperti *cluster* $e = r_1, \dots, r_k$ untuk *quasi-identifier* yang terdiri dari atribut
 25 numerik N_1, \dots, N_m dan atribut kategorikal C_1, \dots, C_n , T_{C_i} adalah *taxonomy tree* untuk domain

- ¹ kategorikal C_i , MIN_{N_i} dan MAX_{N_i} adalah nilai minimum dan maksimum pada cluster e untuk
² atribut N_i , \cup_{C_i} adalah sekumpulan nilai pada cluster e berdasarkan atribut C_i .

Information loss dihitung dengan rumus sebagai berikut:

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

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

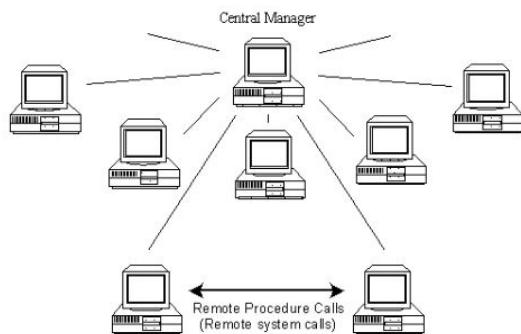
Total Information Loss dihitung dengan rumus sebagai berikut:

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

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

2.10 Sistem Terdistribusi

- 9 Sistem terdistribusi adalah teknik yang memungkinkan komputer dapat saling terhubung melintasi
10 wilayah geografis seolah-olah berada pada satu lingkungan [3]. Gambar 2.16 menyatakan bahwa
11 sistem terdistribusi dapat berbagi sumber daya mulai dari memori hingga jaringan dan penyimpanan.
12 Semua model sistem terdistribusi memiliki kesamaan sifat yaitu sekelompok komputer yang saling
13 terhubung dan bekerja sama untuk membagi beban komputasi.



Gambar 2.15: Sistem Terdistribusi

14 2.10.1 Penggunaan Sistem Terdistribusi

- ¹⁵ Berikut beberapa hal penting terkait sistem distribusi:

- **Tidak semua masalah pemrograman membutuhkan sistem terdistribusi.**
Jika waktu komputasi program untuk komputer lokal masih cepat, maka teknologi sistem distribusi masih belum dibutuhkan. Ketika memerlukan proses analisis data yang kompleks seperti pada lingkungan *big data*, maka dibutuhkan teknologi sistem terdistribusi yang dapat

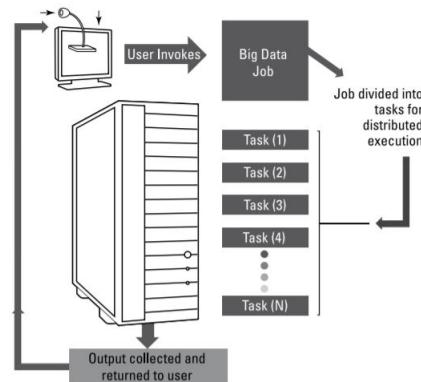
melakukan komputasi dengan cepat yang dapat memanfaatkan komputasi dengan skalabilitas yang tinggi, dimana pekerjaan dapat dibagi secara paralel.

- Penggunaan sistem terdistribusi dipengaruhi perkembangan perangkat keras dan perangkat lunak dalam manajemen data.**

Tingginya penggunaan perangkat komputer menyebabkan perangkat keras komputer mengalami peningkatan signifikan. Perkembangan perangkat keras mempengaruhi perkembangan perangkat lunak, khususnya dibidang manajemen data. Perangkat lunak sekarang dapat memanfaatkan perangkat keras untuk menyeimbangkan dan mengoptimalkan proses di sistem terdistribusi. Perangkat lunak dapat memahami untuk beban kerja tertentu, memerlukan jumlah sumber daya berapa banyak sehingga perangkat lunak dapat membagi pekerjaan dengan seimbang pada beberapa komputer.

2.10.2 Komputasi Big Data Dengan Sistem Terdistribusi

Hanya mengandalkan satu komputer dengan spesifikasi yang mumpuni tidak cukup memastikan kinerja yang menjadi lebih cepat untuk menangani *big data*. Cara yang tepat adalah membagi pekerjaan pemrosesan *big data* pada sistem terdistribusi. Gambar 2.16 menunjukkan tahapan melakukan komputasi *big data*. Langkah pertama, pengguna harus membuat pekerjaan *big data* pada komputer *master*. Langkah selanjutnya, komputer master akan memecah pekerjaan *big data* menjadi bagian-bagian kecil yang disebar ke beberapa komputer. Langkah terakhir, komputer *master* mengumpulkan semua hasil pekerjaan menjadi satu untuk dikembalikan kepada pengguna.



Gambar 2.16: Pemanfaatan Sistem Terdistribusi

Dalam lingkungan *big data*, beberapa komputer dirancang saling terhubung (*cluster*) untuk bekerja secara paralel dalam menyelesaikan pekerjaan. Dimulai dengan menggunakan beberapa komputer untuk pemrosesan data, lalu munculnya kebutuhan untuk memproses data yang lebih besar sehingga memerlukan untuk menambahkan lebih banyak resources agar mendapatkan kinerja yang lebih optimal. Untuk mengakomodasi kebutuhan tersebut, sebuah organisasi hanya perlu menambahkan lebih banyak komputer ke dalam *cluster* sehingga skalabilitas dapat meningkat untuk mengakomodasi kebutuhan pemrosesan data yang terus berkembang. Namun, itu tidak cukup dengan memperluas jumlah *node* di *cluster*. Pada kasus tertentu memerlukan untuk mengirim sebagian analisis data besar ke lingkungan fisik yang berbeda. Cara membagi dan mengelola data dapat membuat perbedaan yang signifikan terhadap kinerja sistem terdistribusi.

2.11 Big Data

Big data adalah kemampuan untuk mengelola sejumlah besar data yang berbeda, dengan kecepatan yang tepat, dan dalam kerangka waktu yang tepat untuk memungkinkan analisis dan reaksi waktu nyata [3]. *Big data* digunakan untuk menganalisis data perusahaan dalam jumlah besar secara real time dalam memberi solusi permasalahan bisnis. *Big data* dapat menggabungkan penyimpanan untuk semua jenis data termasuk data terstruktur, semi terstruktur, dan tidak terstruktur seperti email, media sosial, *text streaming*, dan lainnya.

8



Gambar 2.17: Pemanfaatan Sistem Terdistribusi

2.11.1 Karakteristik Big Data

Berikut adalah karakteristik 5V pada *big data* seperti Gambar 2.17:

- *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.

- *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.

¹ 2.11.2 Jenis Data Pada Big Data

² Berikut adalah jenis data yang umum ditemui pada *big data*:

- ³ • Data terstruktur
 - ⁴ Data terstruktur dapat diproses, disimpan, dan diambil dalam format tetap. Jenis data ini
 - ⁵ disimpan dalam bentuk tabel, baris dan kolom yang normalnya disimpan dalam *Excel* atau
 - ⁶ *Spreadsheet*, dimana informasi pada data sangat terorganisir dan dapat dengan mudah diakses
 - ⁷ dari database dengan algoritma mesin pencari sederhana.
- ⁸ • Semi-terstruktur
 - ⁹ Data semi-terstruktur merupakan jenis data yang dimasukan ke dalam sebuah tabel, tetapi
 - ¹⁰ skemanya tidak sama dengan tabel biasa yang hanya terdiri dari baris dan kolom. Data
 - ¹¹ semi-terstruktur mengandung format data terstruktur dan tidak terstruktur. Walaupun belum
 - ¹² diklasifikasi oleh repository tertentu (*database*), namun mengandung informasi yang penting.
 - ¹³ Contohnya adalah data CSV, XML, dan JSON.
- ¹⁴ • Tidak terstruktur
 - ¹⁵ Data terstruktur adalah data dengan bentuk yang tidak dikenal, harus disimpan dengan
 - ¹⁶ format khusus karena tidak memiliki struktur yang spesifik seperti jenis data structured.
 - ¹⁷ *Raw data* dari jenis data ini hanya dapat menghasilkan nilai setelah diproses dan dianalisa.
 - ¹⁸ Contohnya adalah data yang berformat foto/gambar, video, atau suara.

¹⁹ 2.11.3 Pekerjaan Terkait Big Data

²⁰ Berikut adalah jenis pekerjaan yang umum ditemui pada *big data*:

- ²¹ • *Data warehouse, data mart analytics*
 - ²² Untuk mencapai tujuan data warehouse seperti menghasilkan analitis yang lebih baik, melakukan
 - ²³ implementasi kecerdasan bisnis, melakukan operasi ETL sudah ditangani oleh *framework*
 - ²⁴ *big data* seperti Hadoop dan Spark yang memiliki operasi ETL dirancang khusus di lingkungan
 - ²⁵ *big data* ini. Hal ini dapat mempermudah data warehouse untuk mencapai tujuan yang
 - ²⁶ dimaksud dengan cepat dan dapat menurunkan biaya penambahan sumber daya.
- ²⁷ • *Big data analytics*
 - ²⁸ *Big data analytics* memiliki pekerjaan terkait pengumpulan data, pembuatan alogritma statistik
 - ²⁹ dan pemodelan *machine learning* untuk mengidentifikasi informasi berdasarkan riwayat data
 - ³⁰ yang pernah digunakan, penggunaan teknologi *data mining* dalam memeriksa data berskala
 - ³¹ besar untuk menemukan pola-pola di dalam sebuah data, pemanfaatan teknologi *big data*
 - ³² seperti Spark,Hadoop untuk kebutuhan tertentu,
- ³³ • Big data applications
 - ³⁴ *Big data application* dapat diterapkan di berbagai bidang seperti perbankan, pertanian, kimia,
 - ³⁵ penambangan data, cloud computing, keuangan, pemasaran, saham, kesehatan. *Big data*
 - ³⁶ *application* juga dapat dipakai untuk menjamin keamanan data. Pada penelitian ini, teknologi
 - ³⁷ *big data* digunakan untuk membuat perangkat yang dapat melindungi privasi saat dilakukan
 - ³⁸ proses *data mining*.

1 2.12 Hadoop

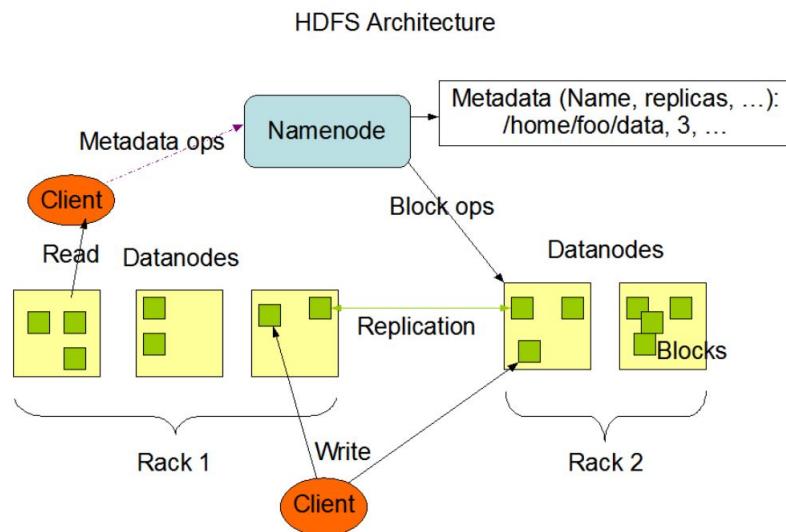
- 2 Hadoop adalah *framework* yang memanfaatkan beberapa komputer untuk menyelesaikan masalah
 3 yang melibatkan volume data sangat besar [3]. Hadoop memecah input dari pengguna menjadi
 4 beberapa blok data dan masing-masing blok data diproses menggunakan konsep *MapReduce* di
 5 mana data akan diproses secara paralel pada sistem terdistribusi.



Gambar 2.18: Hadoop

6 2.12.1 HDFS

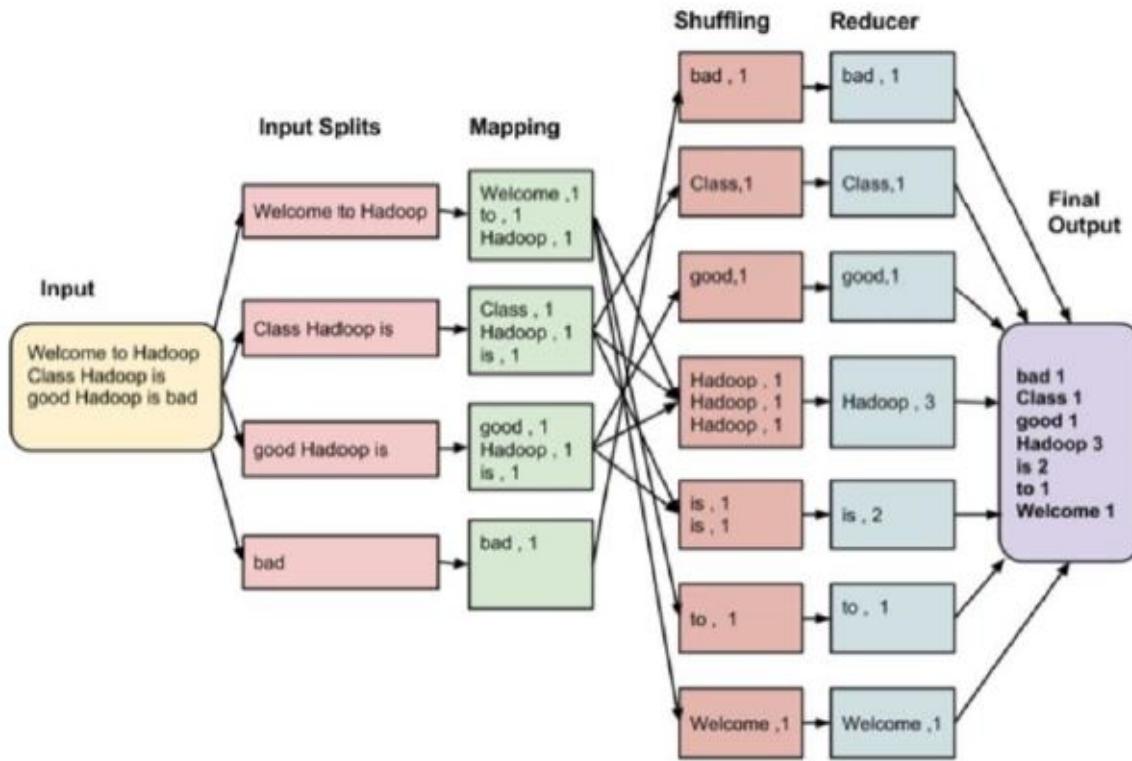
- 7 HDFS terdiri *master/slave node*. HDFS terdiri dari satu namenode (*master*) yang mengelola
 8 penamaan file system pada HDFS dan mengatur akses klien ke *file*. HDFS terdiri dari sejumlah
 9 *DataNodes* (*slave*) yang mengelola penyimpanan *file*. *File* yang disimpan pada HDFS terdiri dari
 10 beberapa blok data. *NameNodes* menjalankan operasi HDFS seperti membuka, menutup, dan
 11 mengganti nama file dan direktori. Sedangkan *DataNodes* bertanggung jawab untuk melayani
 12 permintaan baca dan tulis dari klien ke HDFS. *DataNodes* melakukan pembuatan blok, penghapusan,
 13 dan replikasi atas instruksi dari *NameNodes*.



Gambar 2.19: Arsitektur HDFS

14 2.12.2 MapReduce

- 15 *MapReduce* adalah model pemrograman untuk memproses data berukuran besar secara terdistribusi
 16 dan paralel pada *cluster* yang terdiri atas banyak komputer [3]. Dalam memproses data, secara
 17 garis besar, *MapReduce* dibagi menjadi dua jenis proses yaitu *map* dan *reduce*. Setiap fase memiliki
 18 pasangan *key-value* sebagai input dan output. Kedua jenis proses ini didistribusikan ke setiap
 19 komputer dalam suatu *cluster* dan berjalan secara paralel tanpa saling bergantung satu sama lain.



Gambar 2.20: Proses Komputasi pada MapReduce

1 Berikut adalah penjelasan masing-masing tahapan pada *MapReduce*:

2 • *Input*

3 Pada tahap ini, model *MapReduce* menerima input data secara utuh dari *file text/CSV*.

4 • *Input Splits*

5 Pada tahap ini, model *MapReduce* akan memecah input data menjadi blok-blok data dan
6 disebarluhan ke seluruh *cluster*.

7 • *Mapping*

8 *Mapping* bertujuan untuk memetakan blok-blok data ke dalam pasangan $\langle key, value \rangle$.
9 Key, Value pada contoh ini adalah jenis kata dan jumlah jenis kata pada sebuah blok data.

10 • *Shuffling*

11 *Shuffling* bertujuan untuk mengirim data dari *Mapping* ke *Reducer*, agar data dengan *key*
12 yang sama akan dikelompokan dan diolah oleh *Reducer* yang sama

13 • *Reducer*

14 Reducer bertujuan sebagai proses penggabungan *key,value* dari proses *shuffling* untuk dihitung
15 dan dikembalikan sebagai sebuah output

16 • *Output*

17 Pada tahap ini, pemodelan *MapReduce* telah selesai. Output siap untuk ditulis pada *file*
18 maupun ditampilkan pada *console*.

2.13 Spark

Spark adalah teknologi komputasi *cluster* yang dirancang untuk komputasi cepat [4]. 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*.



Gambar 2.21: Arsitektur Spark

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.

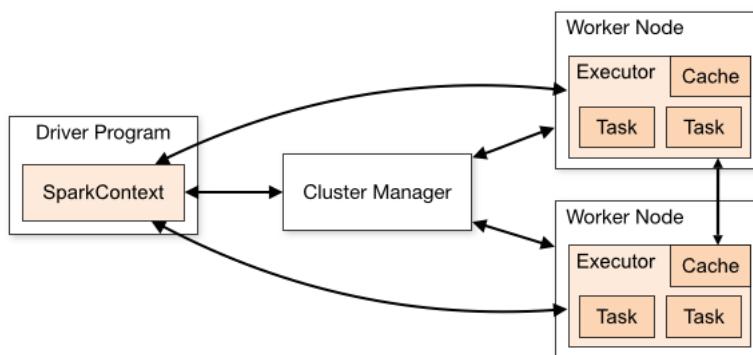
- Mudah untuk diatur

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 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.

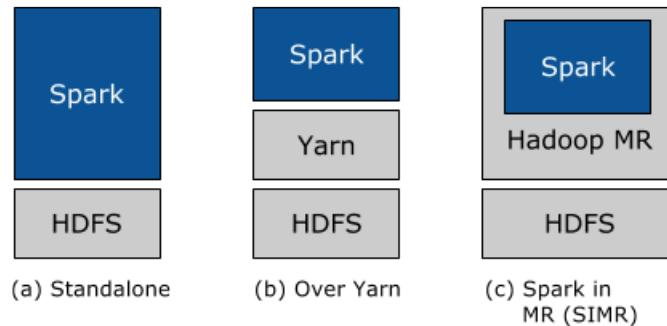
2.13.1 Arsitektur Spark



Gambar 2.22: Arsitektur Spark

- 1 Berdasarkan Gambar 2.22, berikut adalah beberapa hal penting terkait arsitektur Spark:
- 2 • *Driver Program* bertugas untuk menjalankan *Object main* pada *master node*. *Driver Program*
3 adalah tempat dimana *Spark Context* dibuat.
- 4 • *Spark Context* bertugas untuk menghubungkan pengguna dengan *cluster*. *Spark Context* juga
5 digunakan untuk membuat *RDD*, *accumulator*, dan *broadcast variable*.
- 6 • *Cluster Manager* bertugas untuk mengatur sumber daya pada sebuah *cluster*.
- 7 • *Executor* membantu memantau proses-proses yang berjalan pada *worker node* dan bertanggung
8 jawab untuk mengerjakan *task* yang diberikan.
- 9 • *Task* adalah satuan kerja pada Spark yang berisi perintah-perintah fungsi yang diserialisasi.

10 **2.13.2 Jenis Instalasi pada Spark**



Gambar 2.23: Arsitektur Spark

- 11 Berdasarkan Gambar 2.23, berikut adalah jenis-jenis instalasi pada Spark:

- 12 • *Standalone*
13 Spark dapat dipasang pada komputer lokal sehingga pemrosesan data nantinya akan dikerjakan
14 oleh satu komputer. *Standalone* umumnya dipakai jika ingin mempermudah melakukan
15 eksperimen program big data pada IDE khususnya untuk melakukan proses *debugging*. Data
16 yang dijalankan pada lingkungan *standalone* tidak dapat menangani data dalam jumlah besar.
- 17 • *Hadoop Yarn*
18 Spark dapat dipasang pada komputer *cluster* jika data yang diproses besar dan membutuhkan
19 komputasi paralel untuk memproses data secara efisien. *Hadoop yarn* dipakai jika program
20 sudah jadi dan siap untuk dicoba pada lingkungan data yang lebih besar. Dengan menjalankan
21 Spark pada lingkungan *hadoop yarn*, pemrosesan data dalam jumlah besar tidak lagi menjadi
22 masalah terhadap waktu komputasinya.
- 23 • *Spark In MapReduce (SIMR)*
24 SIMR digunakan untuk menjalankan pekerjaan Spark secara independen. Jenis instalasi ini
25 sudah tidak lagi berlaku untuk Spark versi 2.0 ke atas.

2.13.3 Resilient Distributed Datasets (RDD)

RDD adalah kumpulan partisi terdistribusi yang disimpan dalam memori atau *disk* pada beberapa *cluster* [4]. RDD tersebar menjadi beberapa partisi, sehingga partisi tersebut dapat disimpan dan diproses pada komputer yang berbeda.

5

Berikut adalah beberapa karakteristik yang dimiliki RDD:

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

11 Berikut adalah jenis operasi pada RDD:

- 12 • Fungsi *Transformation*

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

15

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.

- 17 • Fungsi *Action*

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

22

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.

2.13.4 DataFrame

1 *DataFrame* adalah kumpulan data yang didistribusikan, disusun dalam baris dan kolom [4]. Setiap
 2 kolom dalam *DataFrame* memiliki nama dan tipe terkait. *Dataframe* mirip dengan tabel database
 3 tradisional, yang terstruktur dan ringkas. Dengan menggunakan *DataFrame*, kueri SQL dapat
 4 dengan mudah diimplementasi pada *big data*.

6

7 Berikut adalah jenis operasi pada Dataframe:

- 8 • Operasi Select, Filter

9 Operasi *Select, Filter* adalah operasi SQL yang umum digunakan untuk mengambil dan
 10 menyeleksi data yang dibutuhkan pada proses analisis. Operasi *Select, Filter* pada DataFrame
 11 akan dijelaskan pada tabel dibawah ini.

12

Fungsi	Deskripsi
select()	Memilih kolom DataFrame yang ingin ditampilkan atau disimpan.
filter()	Menyeleksi baris dari DataFrame berdasarkan satu atau beberapa kondisi atau ekspresi SQL yang diberikan

14

- Operasi Inner-Join, Cross-Join

15 Operasi *Inner-Join, Cross-Join* adalah operasi SQL untuk menggabung satu atau lebih kolom
 16 data yang dibutuhkan pada proses analisis. Operasi *Inner-Join, Cross-Join* pada DataFrame
 17 akan dijelaskan pada tabel dibawah ini.

18

Fungsi	Deskripsi
join()	Melakukan penggabungan tabel berdasarkan nilai kolom masing-masing tabel yang saling beririsan
crossjoin()	Melakukan operasi perkalian cartesian antar tabel

20

- Operasi Agregat

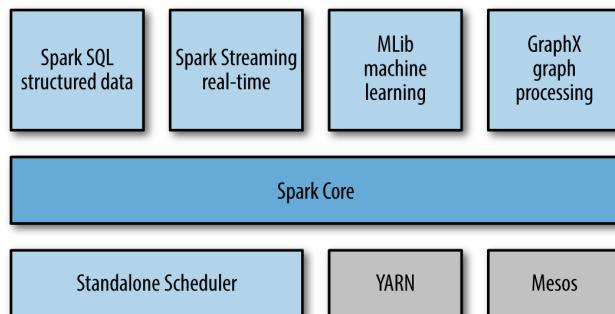
21 Operasi Agregat adalah operasi SQL untuk melakukan beberapa komputasi yang dilakukan
 22 untuk mendapatkan hasil analisis data. Operasi Agregat pada DataFrame akan dijelaskan
 23 pada tabel dibawah ini.

24

Fungsi	Deskripsi
sum()	Mengembalikan penjumlahan semua nilai dalam kolom.
collect_set()	Mengembalikan semua nilai dari kolom input dengan menghilangkan nilai yang bersifat duplikat.

1 2.13.5 Komponen Spark

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



Gambar 2.24: Beberapa Jenis Komponen Spark

5 • Spark Core

6 Spark Core adalah *library* Spark yang berisi fungsionalitas dasar Spark, termasuk komponen
 7 untuk penjadwalan tugas, manajemen memori, pemulihan kesalahan, dan berinteraksi dengan
 8 sistem penyimpanan. Spark Core menyediakan komputasi pada memori, fungsi *action* dan
 9 *transformation* untuk mengolah RDD.

10 Listing 2.1: Import Library Spark Core

```
11 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
12 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.3"
```

14 • Spark SQL

15 Spark SQL memungkinkan pemrosesan kueri SQL pada lingkungan *big data*. Spark SQL
 16 menyediakan fungsi untuk menghitung nilai statistik dasar seperti *mean*, *median*, *modus*, nilai
 17 maksimum dan nilai minimum.

18 Listing 2.2: Import Library Spark SQL

```
19 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
20 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.3"
```

22 • Spark MLlib

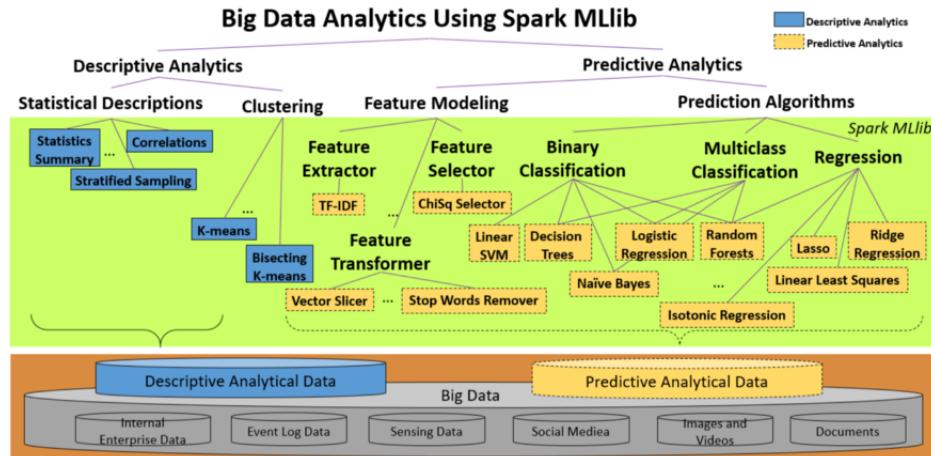
23 Spark MLlib adalah *library* Spark yang berisi fungsionalitas yang umum digunakan pada
 24 *machine learning*. Untuk mengimplementasikan teknik *data mining* pada lingkungan *big data*
 25 dibutuhkan *library* Spark MLlib. Spark MLlib menyediakan berbagai jenis algoritma *machine*
 26 *learning* termasuk klasifikasi dan pengelompokan/*clustering*.

27 Listing 2.3: Import Library Spark MLlib

```
28 // https://mvnrepository.com/artifact/org.apache.spark/spark-mllib
29 libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.4.3"
```

2.14 Spark MLlib

- Spark MLlib adalah *library machine learning* yang tersedia di Spark [4]. Tujuannya adalah membuat pembelajaran mesin praktis dapat diskalakan dan mudah. Spark MLlib mendukung berbagai jenis algoritma, yang disebutkan di bawah ini:



Gambar 2.25: Contoh Vektor Dense dan Sparse

- Berikut adalah jenis algoritma yang tersedia pada Spark MLlib:

- **mllib.regression:** *spark.mllib package* yang mendukung pemodelan regresi untuk menemukan hubungan dan ketergantungan antar variabel. Salah satu kegunaan dari regresi adalah untuk melakukan prediksi berdasarkan data-data yang telah dimiliki sebelumnya.
- **mllib.classification:** *spark.mllib package* yang mendukung pemodelan klasifikasi biner, klasifikasi multikelas dan analisis regresi. Beberapa algoritma paling populer dalam klasifikasi adalah Random Forest, Naive Bayes, Decision Tree, dll.
- **mllib.clustering:** *spark.mllib package* yang mendukung pemodelan clustering (supervised learning) untuk melakukan pengelompokan himpunan data yang memiliki kesamaan sifat menjadi cluster yang sejenis.

- Berikut alasan pemilihan Spark MLlib untuk membuat pemodelan *data mining*:

- Spark MLlib terintegrasi di dalam Spark, sehingga memudahkan pengembangan algoritma *machine learning* untuk skala besar dengan efisien.
- Skalabilitas, kesederhanaan, dan kompatibilitas Spark MLlib tersedia untuk Java, Scala, dan Python. Skalabilitas dapat membantu pemrosesan data dengan lebih cepat menggunakan komputasi paralel.
- Spark MLlib memberikan peningkatan performa pemodelan *machine learning* pada *big data* hingga 10-100 kali lebih cepat dibandingkan Hadoop.
- Spark MLlib mudah untuk dipakai dan tidak memerlukan instalasi apa pun. Hal ini berlaku jika Hadoop versi 2.0.0 sudah pernah dilakukan instalasi.

2.14.1 Tipe Data pada Spark MLlib

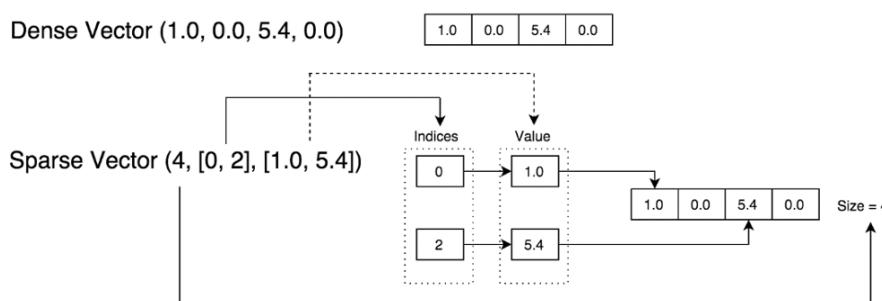
Tipe data yang digunakan untuk membuat model *machine learning* pada Spark MLlib adalah *local vector*, *labeled points*, *local matrix*, dan *distributed matrix*. *Local vector* digunakan untuk pembuatan model data mining (*k-means*, *naive bayes*). *Labeled point* digunakan sebagai training model pada supervised learning (*naive bayes*).

6

Berikut adalah beberapa jenis tipe data pada Spark MLlib:

- 8 • Local Vector

Vektor terdiri dari dua jenis yaitu vektor *dense* dan vektor *sparse*. Kelas vektor berada pada package `mllib.linalg.Vectors`. Gambar 2.27 adalah contoh vektor dense dan vektor sparse:



Gambar 2.26: Contoh Vektor Dense dan Sparse

11 – 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.

14 – 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*.

18 • *LabeledPoint*

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

21 • *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.

24 2.14.2 *Data Mining* pada Spark MLlib

Spark MLlib dapat menggunakan *library machine learning* untuk melakukan pemodelan *data mining*. Untuk melakukan pemodelan *naive bayes*, dapat menggunakan *library mllib.classification*. Untuk melakukan pemodelan *k-means*, dapat menggunakan *library mllib.clustering*. Berikut adalah tahapan implementasi Spark MLlib terhadap masing-masing model *data mining*.

1 1. Naive Bayes

2 2. Naive bayes adalah salah satu metode paling sederhana yang digunakan untuk klasifikasi. *Naive*
3 bayes pada Spark dapat dimodelkan menggunakan vektor fitur. Ciri khas dari naive bayes adalah
4 menganggap bahwa fitur secara independen berperan dalam menentukan kategori, sehingga tidak
5 peduli tentang korelasi antar setiap fitur.

6

7 7. Berikut adalah proses langkah demi langkah untuk membangun pengklasifikasi menggunakan
8 algoritma *naive bayes* dari Spark MLLib.

9 1. Membuat konfigurasi Spark.

Listing 2.4: Baris Kode Scala

```
10
11  SparkConf sparkConf = new SparkConf().setAppName("NaiveBayesClassifierExample");
12
```

14 2. Pisahkan data input berdasarkan *training* dan *test* data.

Listing 2.5: Baris Kode Scala

```
15
16  val training = json.select("naive_bayes.training_set").first().getDouble(0)
17  val test = json.select("naive_bayes.test_set").first().getDouble(0)
18  val pathModel = json.select("naive_bayes.model_path_anonym").first().
19      getString(0)
20  val Array(training, test) = table.randomSplit(Array(trainingSet,testSet))
```

22 3. Melatih model *naive bayes*.

Listing 2.6: Baris Kode Scala

```
23
24  val model = new NaiveBayes().setModelType("multinomial").setLabelCol(label+
25      _Index").fit(training)
```

27 4. Menggunakan model untuk memprediksi *test* data.

Listing 2.7: Baris Kode Scala

```
28
29  val prediction = model.transform(table)
```

31 5. Menghitung akurasi model *naive bayes*.

Listing 2.8: Baris Kode Scala

```
32
33  val evaluator = new MulticlassClassificationEvaluator().
34     setLabelCol(label+_Index").
35      setPredictionCol("prediction").
36      setMetricName("accuracy")
37  val accuracy = evaluator.evaluate(predictions)
```

6. Menyimpan model yang dilatih ke komputer lokal untuk penggunaan ke depannya.

Listing 2.9: Baris Kode Scala

```
model.write.overwrite.save(pathModel)
```

5 *K-Means*

K-means adalah salah satu metode paling sederhana yang digunakan untuk clustering. *K-means* pada Spark dapat dibuat menggunakan vektor fitur. Ciri khas dari *K-means* adalah data-data yang nilainya mirip satu sama lain akan dikelompokan ke dalam satu *cluster* yang sama.

¹⁰ Berikut adalah proses langkah demi langkah untuk membangun model *clustering* menggunakan
¹¹ algoritma *k-means* dari Spark MLLib.

- ## 12 1. Membuat konfigurasi Spark.

Listing 2.10: Baris Kode Scala

```
SparkConf sparkConf = new SparkConf().setAppName("NaiveBayesClassifierExample");
```

- ## 17 2. Melatih model *k-means*.

Listing 2.11: Baris Kode Scala

```
val k = json.select("k_means.k").first().getLong(0).toInt  
val kmeans = new KMeans().setK(k).setFeaturesCol("features")  
    .setPredictionCol("prediction")  
val model = kmeans.fit(table)
```

- 24 3. Menggunakan model untuk memprediksi *test* data.

Listing 2.12: Baris Kode Scala

```
val prediction = model.transform(table)
```

- #### 28 4. Menghitung akurasi model *k-means*.

Listing 2.13: Baris Kode Scala

```
val evaluator = new ClusteringEvaluator()  
val silhouette_score = evaluator.evaluate(predictions)  
return silhouette score
```

- 34 5. Menyimpan model yang dilatih ke komputer lokal untuk penggunaan ke depannya.

Listing 2.14: Baris Kode Scala

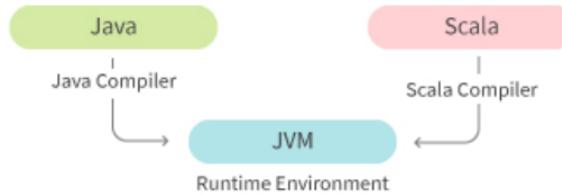
```
model.write.overwrite.save(pathModel)
```

1 2.15 Scala

2 Scala adalah bahasa pemrograman fungsional hybrid karena memiliki fitur pemrograman berorientasi
 3 objek dan pemrograman fungsional [5]. Sebagai bahasa pemrograman *Object-Oriented Programming*
 4 (OOP), Scala menganggap setiap nilai sebagai objek. Scala terdiri dari 2 jenis tipe atribut yaitu **val**
 5 dan **var**. **Val** dipakai jika atribut memiliki nilai yang tetap. Sedangkan, **var** dipakai jika atribut
 6 memiliki nilai yang berubah-ubah.

7

8 Scala dan Java adalah bahasa pemrograman populer yang dijalankan melalui JVM. JVM membuat
 9 bahasa ini ramah kerangka kerja. Bisa dibilang, Scala adalah level lanjutan dari Java.



Gambar 2.27: Scala dan Java JVM

- 10 Jika diberikan skenario dengan menambahkan metode kecil di kelas *User* yang mengembalikan
 11 daftar semua produk yang telah dipesan pengguna maka hasil implementasi adalah sebagai berikut.
 12 Listing 2.15 adalah baris kode program yang ditulis dengan bahasa Java.

Listing 2.15: Baris Kode Java

```

13 public List<Product> getProducts() {
14     List<Product> products = new ArrayList<Product>();
15     for (Order order : orders) {
16         products.addAll(order.getProducts());
17     }
18     return products;
19 }
20 }
21 
```

- 22 Di Scala, fungsi apapun dapat diimplementasi dengan cepat. Masalah ini dapat diselesaikan
 23 menggunakan **flatMap** untuk menggabungkan daftar produk dari setiap pesanan dalam satu daftar,
 24 lalu menyeleksi produk berdasarkan produk yang cocok dengan kategori. Listing 2.16 adalah
 25 perbandingan kode program yang ditulis dengan bahasa Scala.

Listing 2.16: Baris Kode Scala

```

26 def products = orders.flatMap(o => o.products)
27 
```

- 29 Untuk pengembangan Spark, penulisan sintaks Scala dianggap produktif untuk mengimplementa-
 30 sikan kode program. Satu baris kode program pada Scala dapat menggantikan 20 hingga 25 baris
 31 kode Java. Karena alasan tersebut, Scala menjadi bahasa pemrograman yang umum dipakai untuk
 32 pemrograman *big data* pada Spark.

2.16 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.

2.16.1 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 2.17: 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
```

2.16.2 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 2.18. 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 2.18: Format Penyimpanan JSON

```
{
    "firstName": "Rack",
    "lastName": "Jackson",
    "address": {
        "streetAddress": "126",
    },
    "phoneNumbers": [
        { "type": "home", "number": "7383627627" }
    ]
}
```

1

BAB 3

2

ANALISIS

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

5 **3.1 Analisis Masalah**

6 Untuk menghasilkan keputusan bisnis yang berkualitas, perusahaan menggunakan teknik *data*
7 *mining* untuk mengenali pola-pola tertentu pada data yang dikumpulkan. Beberapa penelitian telah
8 mengungkapkan bahwa teknik *data mining* dapat menyebabkan kasus pelanggaran privasi yang
9 tidak disadari oleh pemilik data. Pelanggaran privasi ini dapat terjadi ketika data sensitif seseorang
10 tidak dilakukan proses perlindungan privasi terlebih dahulu sebelum dilakukan pembuatan model
11 *data mining*. Oleh karena itu, diperlukan metode perlindungan data yang dapat menjaga privasi
12 meskipun telah dilakukan proses *data mining*. Karena data yang disimpan mencapai ukuran yang
13 sangat besar, maka implementasi metode perlindungan data membutuhkan teknologi big data agar
14 proses dapat diimplementasikan dengan efisien.

15 Solusi yang tepat untuk menjamin perlindungan data sebelum dilakukan proses data mining
16 adalah anonimisasi. Pada subbab 2.5, anonimisasi bertujuan untuk menyamarkan sebagian nilai
17 atribut data yang unik terhadap atribut data lain, khususnya untuk atribut yang termasuk dalam
18 kategori atribut privasi menurut PII. Pada subbab 2.4, *Privacy-preserving data mining* adalah sebuah
19 cara untuk melindungi data sebelum dilakukan *data mining* agar privasi dari hasil data mining
20 dapat terlindungi. Pada subbab 2.6, *K-anonymity* adalah salah satu metode agar *privacy-preserving*
21 *data mining* dapat dicapai dengan menyamarkan beberapa nilai atribut data. Tujuan utama dari
22 penelitian ini adalah mempelajari, menganalisis, melakukan eksperimen, membangun perangkat
23 lunak, dan melakukan pengujian hasil anonimisasi.

24 Pada penelitian ini, akan dibangun 3 jenis perangkat lunak. Pertama adalah perangkat lunak
25 eksplorasi yang digunakan untuk mencari nilai unik. Nilai unik atribut dipakai sebagai referensi
26 dalam membuat pohon DGH untuk proses anonimisasi data. Kedua adalah perangkat lunak
27 anonimisasi yang digunakan untuk proses anonimisasi data. Perangkat lunak ini melakukan imple-
28 mentasi algoritma *greedy k-member clustering* dan metode *k-anonymity* pada lingkungan *big data*.
29 Ketiga adalah perangkat lunak pengujian yang digunakan untuk menganalisis waktu komputasi
30 pengelompokan dan anonimisasi data, *total infomation loss*, waktu pembuatan model *naive bayes*
31 dan *k-means*, metrik klasifikasi (tingkat akurasi), metrik clustering (silhouette score), persentase
32 perbedaan hasil klasifikasi dan clustering.

33

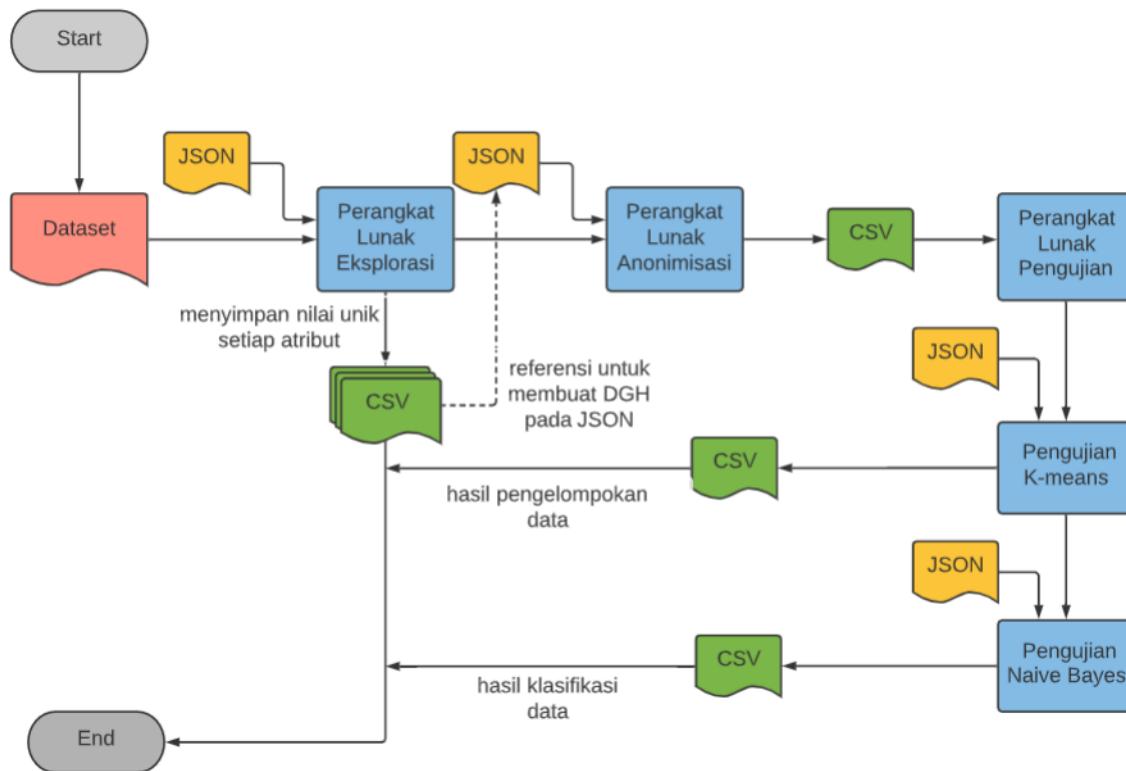
- 1 Berikut tahapan analisis masalah sebelum melakukan implementasi perangkat lunak:
- 2 1. Menjelaskan gambaran umum perangkat lunak.
- 3 2. Mengenali karakteristik data yang ingin dianonimisasi.
- 4 3. Melakukan identifikasi atribut yang menjadi *identifier*, *quasi-identifier*, dan *sensitive attribute* menggunakan konsep dari *Personally Identifiable Information* (PII).
- 5 4. Mempelajari cara perhitungan *distance* dan *information loss* untuk diimplementasikan pada algoritma *greedy k-member clustering* dalam mencari kelompok terbaik.
- 6 5. Mempelajari tahapan dan cara algoritma *greedy k-member clustering* mencari pengelompokan data terbaik sebelum dilakukan proses anonimisasi data.
- 7 6. Mempelajari tahapan dan cara metode *k-anonymity* melakukan anonimisasi data setelah dilakukan pengelompokan data dari tahap sebelumnya.
- 8 7. Melakukan eksplorasi Spark, dengan melakukan proses instalasi perangkat lunak yang dibutuhkan dan melakukan eksperimen terkait fungsi-fungsi pada Spark.

14 3.2 Gambaran Umum Perangkat Lunak

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

- 18 1. Perangkat lunak anonimisasi dapat mengimplementasikan algoritma *k-anonymity* dan algoritma *greedy k-member clustering*. Masukan dari perangkat lunak ini adalah data *credit score* dalam format CSV dan JSON yang berisi parameter dari algoritma *greedy k-member clustering* yaitu nilai *k* dan objek *Domain Generalization Hierarchy* (DGH). Keluaran dari perangkat lunak ini adalah hasil pengelompokan dan anonimisasi dalam format CSV.
- 19 2. Perangkat lunak pengujian dapat membandingkan waktu komputasi pengelompokan dan anonimisasi data, menghitung *information loss* dari pengelompokan data, menghasilkan pengelompokan data (*k-means*), menghasilkan klasifikasi data (*naive bayes*), menghitung tingkat akurasi pada model *naive bayes*, menghitung *silhouette score* pada model *k-means*, dan mencari persentase perbedaan hasil model *k-means*, *naive bayes* sebelum dan setelah dilakukan anonimisasi. Masukan dari perangkat lunak ini adalah dataset *Credit score* dalam format file CSV dan JSON yang berisi parameter pengujian. Untuk pemodelan *k-means* membutuhkan parameter tambahan seperti nilai *k* dan atribut prediktor yang ingin digunakan. Sedangkan untuk pemodelan *Naive Bayes* membutuhkan parameter tambahan seperti persentase antara *training* dan *testing* data, atribut prediktor yang ingin digunakan, dan atribut label. Keluaran dari perangkat lunak ini adalah hasil pemodelan *k-means* dan *naive bayes*, hasil perhitungan *total information loss*, hasil perhitungan tingkat akurasi dan *silhouette score*, dan perbedaan hasil klasifikasi, *clustering* sebelum dan setelah anonimisasi yang disimpan dalam format CSV.

- 1 3. Perangkat lunak eksplorasi dapat mencari nilai unik dari masing-masing atribut yang dipilih
 2 untuk digunakan sebagai referensi dalam mengisi atribut `domain_generalization_hierarchy`
 3 pada masukan JSON untuk perangkat lunak anonimisasi. Masukan dari perangkat lunak ini
 4 adalah data *credit score* dalam format file CSV dan file JSON yang berisi nama atribut yang
 5 ingin dicari nilai uniknya. Keluaran dari perangkat lunak ini adalah nilai unik setiap atribut
 6 yang disimpan dalam format CSV.



Gambar 3.1: Flow Chart Penggunaan Perangkat Lunak

- 7 Pada Gambar 3.1 telah ditampilkan input dan output untuk masing-masing perangkat lunak.
 8 Persegi panjang berwarna biru diartikan sebagai program utama dan persegi panjang berwarna
 9 krem diartikan sebagai program pengujian. Bentuk dengan warna merah diartikan sebagai masukan
 10 perangkat lunak dalam bentuk CSV. Bentuk dengan warna hijau diartikan sebagai keluaran
 11 perangkat lunak dalam bentuk CSV. Bentuk dengan warna oranye diartikan sebagai input perangkat
 12 lunak dalam bentuk JSON. Pada bagian selanjutnya akan dijelaskan perancangan perangkat lunak
 13 dengan lebih detil. Penjelasan akan dibagi menjadi tiga bagian yaitu perancangan perangkat lunak
 14 eksplorasi, perancangan perangkat lunak anonimisasi, dan perancangan perangkat lunak pengujian.

15 3.2.1 Diagram Aktifitas

- 16 Penelitian ini memiliki tiga jenis diagram aktivitas, yaitu diagram aktivitas untuk perangkat lunak
 17 eksplorasi, diagram aktivitas untuk perangkat lunak anonimisasi, dan diagram aktivitas untuk
 18 perangkat lunak pengujian. Tujuan dari pembangunan tiga jenis perangkat lunak antara lain untuk
 19 memisahkan perangkat lunak berdasarkan fungsionalitasnya.

1 Perangkat Lunak Anonimisasi Data

- 2 Perangkat lunak ini bertujuan untuk melakukan proses anonimisasi pada data *credit score* menggunakan algoritma *k-anonymity*. Diagram aktifitas dapat dilihat pada Gambar 3.4, berikut adalah tahapan yang terjadi pada perangkat lunak anonimisasi:
- 5 1. Pengguna perlu menjalankan hadoop terlebih dahulu.
- 6 2. Pengguna mengisi parameter perangkat lunak anonimisasi menggunakan JSON. File JSON disimpan pada sebuah folder input dengan nama tertentu di komputer lokal.
- 8 3. Pengguna membuat folder output di komputer lokal. Sebagai informasi tambahan, nama folder output harus sesuai dengan yang dicantumkan pada JSON.
- 10 4. Pengguna menjalankan perintah eksekusi Spark pada terminal *command prompt*. Pada tahap ini, perangkat lunak akan menampilkan *log* program pada terminal.
- 12 5. Pengguna menunggu eksekusi perangkat lunak sampai selesai. Perangkat lunak yang telah selesai ditandai dengan terminal yang berhenti menampilkan isi *log*.
- 14 6. Perangkat lunak menyimpan hasil eksekusi pada folder output. Perangkat lunak anonimisasi menyimpan hasil pengelompokan dan anonimisasi dalam format CSV.
- 16 7. Pengguna meninjau ulang hasil perangkat lunak. Bagian yang perlu ditinjau adalah apakah perangkat lunak telah tepat menyimpan output pada folder yang telah ditetapkan

18 Perangkat Lunak Pengujian

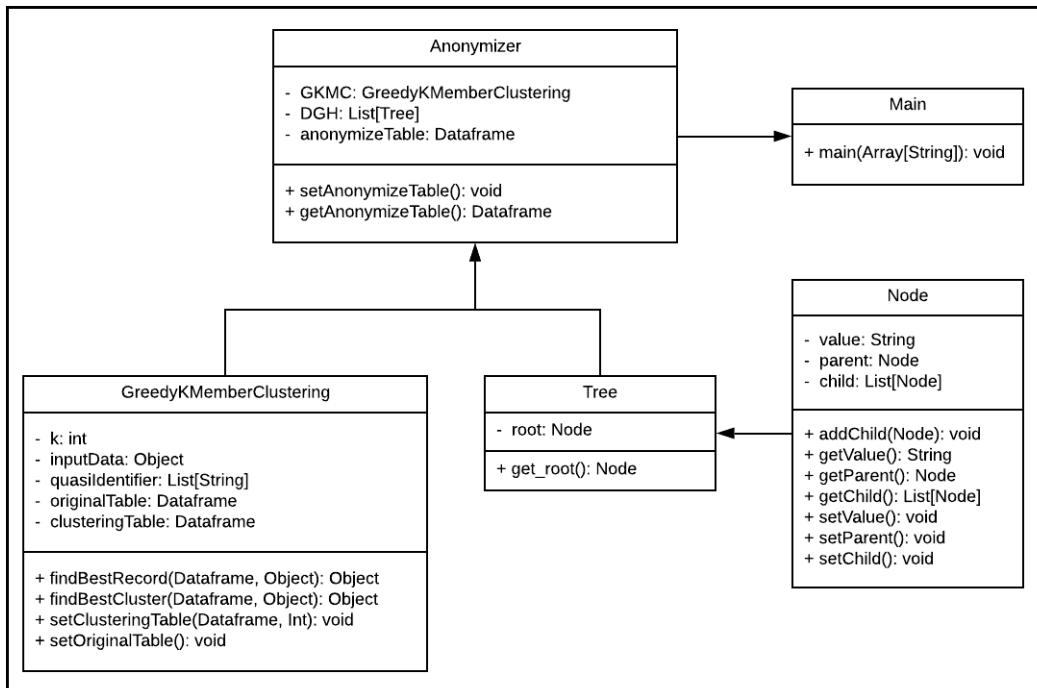
- 19 Perangkat lunak ini bertujuan untuk mencari perbandingan hasil sebelum dan setelah data dilakukan proses anonimisasi dengan metode *data mining*. Diagram aktifitas dapat dilihat pada Gambar 3.5, berikut adalah tahapan yang terjadi pada perangkat lunak pengujian:
- 22 1. Pengguna perlu menjalankan hadoop terlebih dahulu.
- 23 2. Pengguna mengisi parameter perangkat lunak pengujian menggunakan JSON. File JSON disimpan pada sebuah folder input dengan nama tertentu di komputer lokal.
- 25 3. Pengguna membuat folder output di komputer lokal. Sebagai informasi tambahan, nama folder output harus sesuai dengan yang dicantumkan pada JSON.
- 27 4. Pengguna menjalankan perintah eksekusi Spark pada terminal *command prompt*. Pada tahap ini, perangkat lunak akan menampilkan *log* program pada terminal.
- 29 5. Pengguna menunggu eksekusi perangkat lunak sampai selesai. Perangkat lunak yang telah selesai ditandai dengan terminal yang berhenti menampilkan isi *log*.
- 31 6. Perangkat lunak menyimpan hasil eksekusi pada folder output. Perangkat lunak pengujian menyimpan hasil *clustering* dan klasifikasi dalam format CSV.
- 33 7. Pengguna meninjau ulang hasil perangkat lunak. Bagian yang perlu ditinjau adalah apakah perangkat lunak telah tepat menyimpan output pada folder yang telah ditetapkan

1 Perangkat Lunak Eksplorasi

2 Perangkat lunak ini bertujuan mencari nilai unik masing-masing atribut untuk digunakan sebagai
 3 referensi dalam membuat pohon DGH pada proses anonimisasi. Diagram aktifitas dapat dilihat
 4 pada Gambar 3.3, berikut adalah tahapan yang terjadi pada perangkat lunak eksplorasi:

- 5 1. Pengguna perlu menjalankan hadoop terlebih dahulu.
- 6 2. Pengguna mengisi parameter perangkat lunak eksplorasi menggunakan JSON. *File JSON*
 7 disimpan pada sebuah folder input dengan nama tertentu di komputer lokal.
- 8 3. Pengguna membuat folder output di komputer lokal. Sebagai informasi tambahan, nama
 9 folder output harus sesuai dengan yang dicantumkan pada JSON.
- 10 4. Pengguna menjalankan perintah eksekusi Spark pada terminal command prompt. Pada tahap
 11 ini, perangkat lunak akan menampilkan *log* program pada terminal.
- 12 5. Pengguna menunggu eksekusi perangkat lunak sampai selesai. Perangkat lunak yang telah
 13 selesai ditandai dengan terminal yang berhenti menampilkan isi *log*.
- 14 6. Perangkat lunak menyimpan hasil eksekusi pada folder output. Perangkat lunak eksplorasi
 15 menyimpan nilai unik masing-masing atribut dalam format CSV.
- 16 7. Pengguna meninjau ulang hasil perangkat lunak. Bagian yang perlu ditinjau adalah apakah
 17 perangkat lunak telah tepat menyimpan output pada folder yang telah ditetapkan

18 3.2.2 Diagram Kelas



Gambar 3.2: Diagram Kelas Anonimisasi Data

1 Diagram kelas bertujuan untuk menggambarkan keterhubungan antar kelas. Pada penelitian ini
2 digambarkan diagram kelas untuk perangkat lunak anonimisasi data. Karena perangkat lunak ana-
3 lisis data hanya memiliki satu kelas saja, maka keterhubungan antar kelas tidak perlu digambarkan
4 dalam diagram kelas. Gambar 3.2 menggambarkan keterhubungan antar kelas pada perangkat
5 lunak anonimisasi data. Berikut adalah penjelasan lengkap mengenai deskripsi kelas dan method
6 pada perangkat lunak anonimisasi data:

7 • Kelas *Anonymizer* bertujuan untuk melakukan proses anonimisasi setelah data dikelompokan
8 menjadi beberapa *cluster*. Kelas *Anonymizer* memiliki 2 jenis variabel, yaitu:

- 9 – *GKMC* adalah objek dari kelas *GreedyKMemberClustering* yang berisi tabel hasil penge-
10 lompokan data berdasarkan algoritma *Greedy k-member clustering*.
- 11 – *DGH* adalah array 1 dimensi dari objek *Tree* yang berisi hasil anonimisasi untuk nilai
12 quasi-identifier yang unik agar menjadi nilai yang lebih umum.
- 13 – *anonymizeTable* adalah array 2 dimensi dari kelas *Object* untuk menyimpan tabel hasil
14 anonimisasi data.

15 Kelas *Anonymizer* memiliki 2 jenis method, yaitu:

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

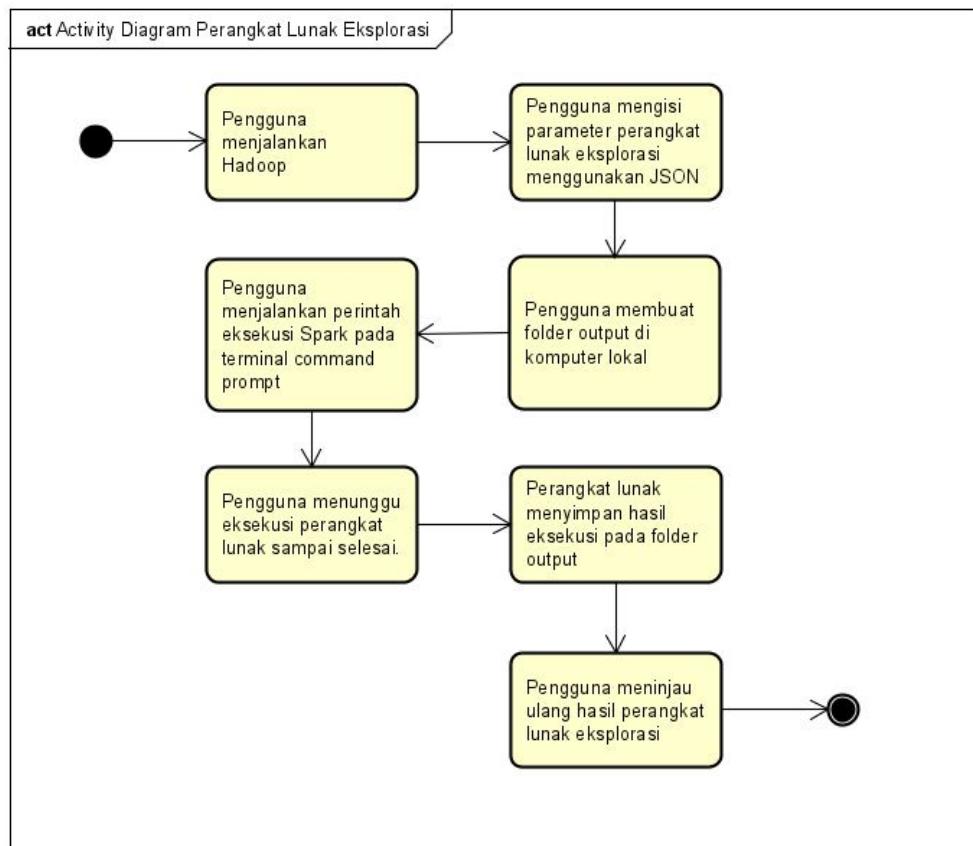
20 • Kelas *GreedyKMemberClustering* bertujuan untuk melakukan pengelompokan data menjadi
21 beberapa *cluster* berdasarkan sifat/nilai atribut yang dimiliki oleh masing-masing baris data.
22 Kelas *GreedyKMemberClustering* memiliki 5 jenis variabel, yaitu:

- 23 – *k* adalah variabel bertipe *Integer* untuk membatasi jumlah anggota pada sebuah *cluster*
24 agar memiliki jumlah yang tetap sebanyak jumlah tertentu.
- 25 – *inputData* adalah variabel untuk menyimpan seluruh baris data *file CSV*.
- 26 – *quasiIdentifier* adalah daftar dari nama-nama kolom yang akan dipilih untuk membuat
27 tabel baru yang digunakan pada proses anonimisasi data
- 28 – *originalTable* adalah tabel yang menyimpan seluruh baris data pada file CSV berdasarkan
29 jenis kolom yang terpilih pada variabel *quasiIdentifier*.
- 30 – *clusteringTable* adalah tabel yang menyimpan hasil pengelompokan baris data dari
31 algoritma *Greedy k-member clustering*.

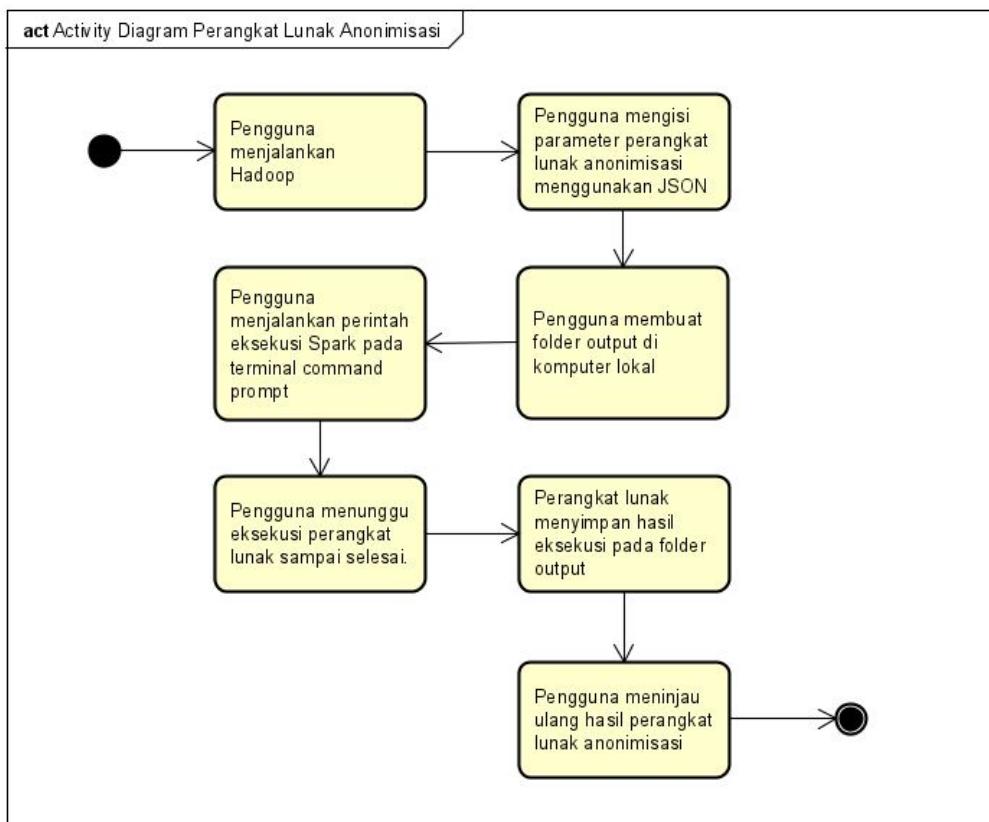
32 Kelas *GreedyKMemberClustering* memiliki 4 jenis method, yaitu:

- 33 – *findBestRecord()* bertujuan mencari sebuah baris data yang memiliki nilai *information*
34 *loss* yang paling minimal dengan baris data lainnya.
- 35 – *findBestCluster()* bertujuan mencari sebuah *cluster* data yang memiliki nilai *information*
36 *loss* yang paling minimal dengan *cluster* lainnya.

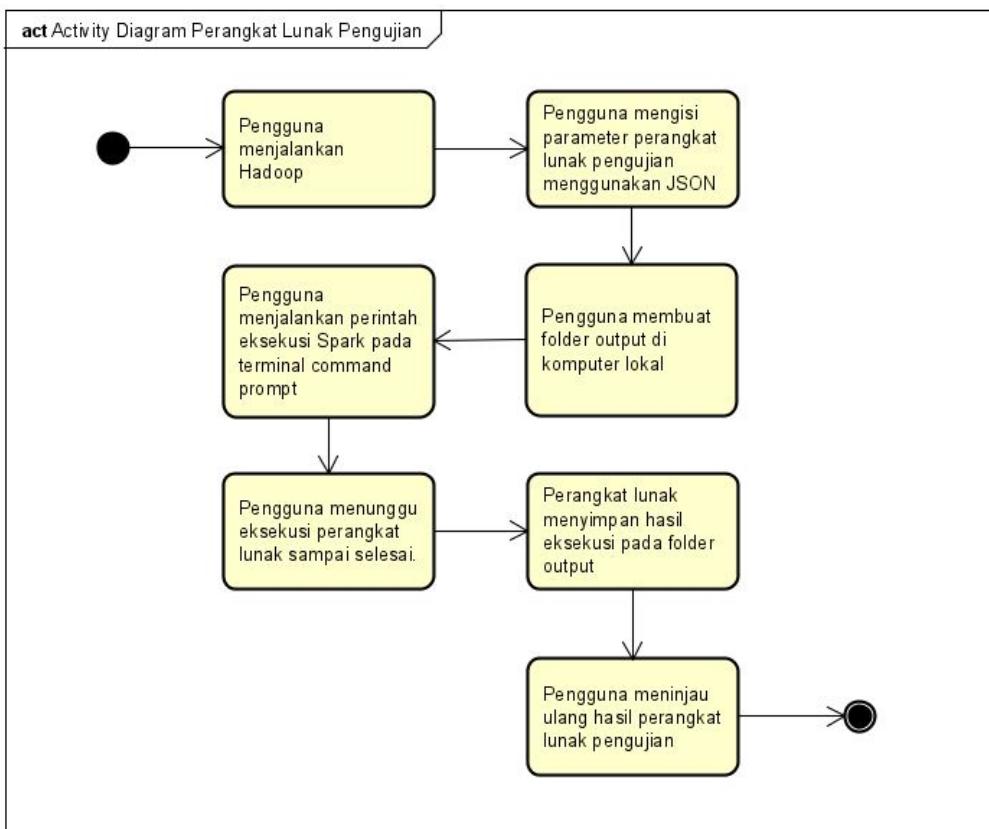
- 1 – `setClusteringTable()` bertujuan mengelompokkan data berdasarkan algoritma *Greedy k-member clustering* dan hasilnya disimpan pada variabel *clusteringTable*.
- 2 – `setOriginalTable()` bertujuan mengubah hasil pembacaan data input CSV menjadi tabel baru dan hasilnya disimpan pada variabel *originalTable*.
- 5 • Kelas *Tree* bertujuan untuk membuat pohon generalisasi berdasarkan jenis atribut *quasi-identifier* yang dipilih.
- 6 • Kelas *Node* bertujuan untuk menyimpan seluruh nilai *quasi-identifier* yang unik untuk masing-masing baris data.
- 9 • Kelas *Main* bertujuan untuk membuat tahapan anonimisasi dari awal sampai akhir dengan memanfaatkan pemanggilan *method* dari masing-masing objek kelas.



Gambar 3.3: Diagram Aktifitas Perangkat Lunak Eksplorasi



Gambar 3.4: Diagram Aktifitas Perangkat Lunak Anonimisasi



Gambar 3.5: Diagram Aktifitas Perangkat Lunak Pengujian

¹ 3.2.3 Pengenalan Karakteristik Data

² Data yang dipakai adalah *credit score*. Data ini diperoleh dari website Kaggle¹. Database ini disimpan
³ dalam format CSV seperti penjelasan pada bagian 2.16. Format CSV memisahkan nilai atribut
⁴ data melalui simbol koma. Data *credit score* dipilih, karena menggunakan informasi personal dari
⁵ pendaftar kartu kredit. Data ini digunakan oleh industri finansial untuk memprediksi kemungkinan
⁶ seseorang gagal bayar di masa depan dari pinjaman kartu kredit.

Listing 3.1: Dataset Credit Score

7	ID, CODE_GENDER, AMT_INCOME_TOTAL, NAME_INCOME_TYPE, DAYS_BIRTH, DAYS_EMPLOYED,
8	OCCUPATION_TYPE
9	5008804, M, 427500.0, Working, -12005, -4542,
10	5008805, M, 427500.0, Working, -12005, -4542,
11	5008806, M, 112500.0, Working, -21474, -1134, Security staff
12	5008808, F, 270000.0, Commercial associate, -19110, -3051, Sales staff
13	5008809, F, 270000.0, Commercial associate, -19110, -3051, Sales staff
14	5008810, F, 270000.0, Commercial associate, -19110, -3051, Sales staff
15	5008811, F, 270000.0, Commercial associate, -19110, -3051, Sales staff
16	5008812, F, 283500.0, Pensioner, -22464, 365243,
17	5008813, F, 283500.0, Pensioner, -22464, 365243,
18	5008814, F, 283500.0, Pensioner, -22464, 365243,
19	5008815, M, 270000.0, Working, -16872, -769, Accountants
20	5112956, M, 270000.0, Working, -16872, -769, Accountants
21	
22	

²³ Berikut deskripsi masing-masing atribut *credit score*:

- ²⁴ • ID: nomor klien.
²⁵ Contoh: 5008804, 5008805
- ²⁶ • CODE_GENDER: jenis kelamin klien.
²⁷ Contoh: M (*Male*), F (*Female*)
- ²⁸ • AMT_INCOME_TOTAL: pendapatan bulanan klien.
²⁹ Contoh: 427500, 270000
- ³⁰ • NAME_INCOME_TYPE: kategori pendapatan klien.
³¹ Contoh: *Commercial associate*, *Pensioner*
- ³² • DAYS_BIRTH: tanggal ulang tahun klien, dihitung mundur dari hari pendaftaran (0), (-1)
³³ artinya hari kemarin. Contoh: -12005, -19110
- ³⁴ • DAYS_EMPLOYED: tanggal klien mulai bekerja, dihitung mundur dari hari pendaftaran (0),
³⁵ bernilai positif jika klien sudah tidak bekerja. Contoh: -4542, -3051
- ³⁶ • OCCUPATION_TYPE: jenis pekerjaan klien.
³⁷ Contoh: *Security staff*, *Accountants*

¹<https://www.kaggle.com/rikdifos/credit-card-approval-prediction>

3.2.4 Personally Identifiable Information

Pada subbab 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 subbab 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 data *credit score* adalah ID. Atribut *quasi-identifier* adalah atribut yang dapat mengidentifikasi seseorang jika nilai sebuah atribut digabung dengan nilai atribut lain pada baris yang sama. Contoh *quasi-identifier* pada dataset *Credit score* adalah CODE_GENDER, NAME_INCOME_TYPE, DAYS_BIRTH, DAYS_EMPLOYED, OCCUPATION_TYPE. *Sensitive attribute* adalah nilai yang dirahasiakan. Contoh sensitive attribute pada dataset *credit score* adalah AMT_INCOME_TOTAL. Ketiga atribut ini dipakai pada proses anonimisasi.

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 terjamin.

3.2.5 Perhitungan Distance dan Total Information Loss

Pada bagian 2.9, telah dijelaskan 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.

Perhitungan 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.

Sebagai contoh, diambil 2 sampel data dari data *credit score* sebagai berikut:

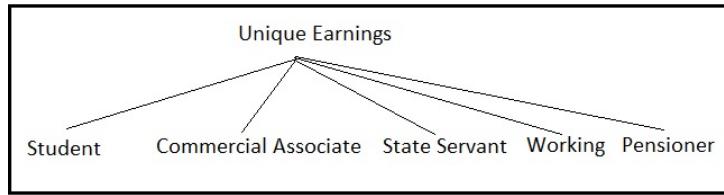
1. 5008811,F,270000.0,Commercial associate,-19110,-3051,Sales staff
2. 5008812,F,283500.0,Pensioner,-22464,365243,

Distance atribut numerik dapat dihitung sebagai berikut berdasarkan DAYS_BIRTH data pertama (v_1)= 39, DAYS_BIRTH data kedua (v_2)= 50, dan jumlah data (D)= 400.000 data.

$$\delta_n(v_1, v_2) = \frac{|v_1 - v_2|}{|D|} = \frac{|(-19110) - (-22464)|}{400.000} = \frac{3354}{400.000} = 0,008385$$

Distance atribut kategorikal dapat dihitung berdasarkan NAME_INCOME_TYPE data pertama (v_1)= Commercial associate, NAME_INCOME_TYPE data kedua (v_2)= Pensioner, tinggi *lowest common ancestor* (node root yang memiliki kedua data kategorikal paling dekat) $H(\Lambda(v_i, v_j))= 2$, dan tinggi pohon DGH ($H(T_D)$)= 2 seperti pada Gambar 3.6.

$$\delta_C(v_1, v_2) = \frac{H(\Lambda(v_i, v_j))}{H(T_D)} = \frac{2}{2} = 1$$



Gambar 3.6: Pohon DGH (NAME_INCOME_TYPE)

1 Perhitungan Total Information Loss

- 2 *Information Loss* (IL) bertujuan untuk mencari tahu kualitas dari hasil pengelompokan data.
- 3 Semakin kecil nilainya, maka hasil pengelompokannya semakin baik. Tabel 3.1 adalah contoh hasil
- 4 pengelompokan data pada dataset *Credit card*:

Tabel 3.1: Tabel Hasil Clustering Data pada Cluster 1

ID	DAYS_EMPLOYED	INCOME_TYPE	OCCUPATION_TYPE	INCOME_TOTAL	Cluster
1	-4542	Working	Security staff	427500	Cluster 1
2	-4542	Working	Security staff	427500	Cluster 1
3	-1134	Working	Sales staff	112500	Cluster 2
4	-3051	Pensioner	Sales staff	270000	Cluster 2
5	-3051	Pensioner	Sales staff	270000	Cluster 2

- 5 *Information Loss* (IL) dapat dihitung sebagai berikut, untuk atribut numerik diketahui jumlah anggota cluster (e) = 2, $MAX_{DAYS_EMPLOYED}$ = -1134, $MIN_{DAYS_EMPLOYED}$ = -4542, $N_{DAYS_EMPLOYED}$ = 5, sedangkan untuk atribut kategorikal diketahui tinggi *lowest common ancestor* $H(\Lambda(\cup_{C_j}))$ = 2 dan tinggi DGH $H(T_{C_j})$ = 2 pada atribut INCOME_TYPE, OCCUPATION_TYPE

$$\begin{aligned}
 D(e) &= \sum_{i=1}^m \frac{(MAX_{N_i} - MIN_{N_i})}{|N_i|} + \sum_{j=1}^n \frac{H(\Lambda(\cup_{C_j}))}{H(T_{C_j})} \\
 &= \frac{((-1134) - (-4542))}{5} + \frac{2}{2} + \frac{2}{2} + \frac{2}{2} + \frac{2}{2} = 681.6
 \end{aligned}$$

$$IL(e) = |e| \cdot D(e) = 2 \cdot 681.6 = 1363.2$$

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

$$\begin{aligned}
 Total - IL(AT) &= \sum_{e \in \varepsilon} IL(e) \\
 &= IL(Cluster1) + IL(Cluster2) + \dots + IL(ClusterN) \\
 &= 1363.2 + \dots + \dots
 \end{aligned}$$

3.2.6 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-identifier, dengan nilai QI = {*DAYS_EMPLOYED*, *INCOME_TYPE*, *OCCUPATION_TYPE*}. Jenis atribut anonimisasi yang kedua adalah Sensitive Attribute, dengan nilai SA = {*INCOME_TOTAL*}. Jika diberikan parameter $k = 2$ dan jumlah cluster yang dibentuk (m) = 2, maka algoritma ini siap ditelusuri lebih lanjut.

Tabel 3.2: Data Credit Score

ID	<i>DAYS_EMPLOYED</i>	<i>INCOME_TYPE</i>	<i>OCCUPATION_TYPE</i>	<i>INCOME_TOTAL</i>
t1	-4542	Working	Security staff	427500
t2	-4542	Working	Security staff	427500
t3	-1134	Working	Sales staff	112500
t4	-3051	Pensioner	Sales staff	270000
t5	-3051	Pensioner	Sales staff	270000

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

1. Nilai awal $r = \emptyset$, $r = \{t1\}$, $|S| = 5$, $k = 2$

2. Karena kondisi $|S| \geq k$ terpenuhi, maka dilakukan perulangan sebagai berikut:

(a) Nilai r diubah menjadi $r = \{t3\}$, karena terbukti data $t3$ memiliki $\Delta(t1, t3) = 683.6$ yang paling tinggi dari seluruh *distance* lain. Berikut adalah contoh perhitungannya:

$$\Delta(t_1, t_2) = 2.000$$

$$\Delta(t_1, t_3) = 683.6$$

$$\Delta(t_1, t_4) = 282.2$$

$$\Delta(t_1, t_5) = 282.2$$

(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 $IL(t3 \cup t4) = 385.4$ yang paling rendah. Berikut adalah contoh perhitungannya:

$$IL(t3 \cup t1) = 683.6$$

$$IL(t3 \cup t2) = 683.6$$

$$IL(t3 \cup t4) = 385.4$$

$$IL(t3 \cup t5) = 385.4$$

- 1 ii. Nilai S diubah menjadi $S = \{t1, t2, t5\}$, $|S| = 4$
 2 iii. Nilai c ditambahkan menjadi $c = \{t3, t4\}$, $|c| = 2$
 3 (e) Karena kondisi $|c| < k$ sudah tidak terpenuhi lagi, maka perulangan ini akan berhenti
 4 (f) Nilai $result$ akan ditambahkan menjadi $result = \{t3, t4\}$
 5 (g) Karena kondisi $|S| \geq k$ masih terpenuhi, maka perulangan akan tetap berlanjut sampai
 6 pada kondisi dimana $|S| < k$ sehingga hasil akhirnya adalah $result = \{\{t3, t4\}, \{t1, t2\}\}$,
 7 $S = \{t5\}$, $|S| = 1$.
 8 (h) Hasil sementara pengelompokan data ditunjukkan pada Tabel 3.3.

Tabel 3.3: Hasil Pengelompokan Sementara (Greedy k-member clustering)

ID	DAYs_EMPLOYED	INCOME_TYPE	OCCUPATION_TYPE	INCOME_TOTAL
t1	-4542	Working	Security staff	427500
t2	-4542	Working	Security staff	427500
t3	-1134	Working	Sales staff	112500
t4	-3051	Pensioner	Sales staff	270000

- 9 3. Karena kondisi $S \neq 0$ terpenuhi, maka dilakukan perulangan sebagai berikut:
 10 (a) Nilai r diubah menjadi $r = \{t1\}$
 11 (b) Nilai S diubah menjadi $S = \{\phi\}$, $|S| = 0$
 12 (c) Nilai c diubah menjadi $c = \{t3, t4\}$ karena terbukti $cluster c$ memiliki $IL(\{t1, t2\} \cup t5) = 300.2$ yang paling rendah dari seluruh $cluster$ lain. Berikut adalah contoh perhitungannya:

$$IL(\{t3, t4\} \cup t5) = 385.4$$

$$IL(\{t1, t2\} \cup t5) = 300.2$$

- 12 (d) Nilai c ditambahkan menjadi $c = \{t1, t2, t5\}$
 13 (e) Nilai c pada perulangan ini ditambahkan pada $result$
 14 (f) Nilai $|S|$ akan dikurangi dengan 1, karena data terakhir telah dikelompokan
 15 (g) Karena kondisi $S \neq 0$ sudah tidak terpenuhi lagi, maka perulangan ini akan berhenti.
 16 4. Hasil akhirnya adalah $result = \{\{t1, t2, t5\}, \{t3, t4\}\}$ dikembalikan sebagai output untuk
 17 algoritma *greedy k-member clustering* seperti pada Tabel 3.4 sebagai berikut:

Tabel 3.4: Hasil Pengelompokan Akhir (Greedy k-member clustering)

ID	DAYs_EMPLOYED	INCOME_TYPE	OCCUPATION_TYPE	INCOME_TOTAL	Cluster
t1	-4542	Working	Security staff	427500	Cluster 1
t2	-4542	Working	Security staff	427500	Cluster 1
t5	-3051	Pensioner	Sales staff	270000	Cluster 1
t3	-1134	Working	Sales staff	112500	Cluster 2
t4	-3051	Pensioner	Sales staff	270000	Cluster 2

3.2.7 Domain Generalization Hierarchy

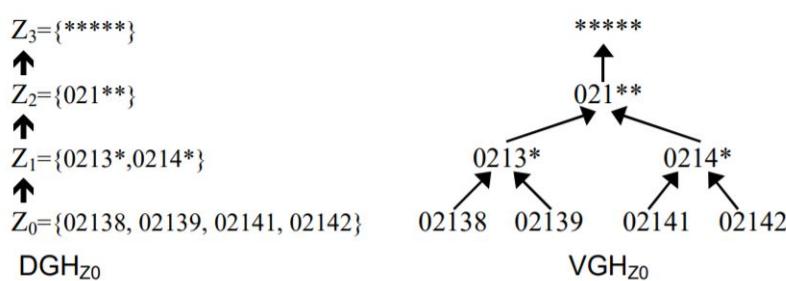
Pada subbab 2.7, telah dijelaskan konsep mengenai *Domain Generalization Hierarchy*. DGH adalah contoh penerapan dari *generalization hierarchy*. DGH bertujuan untuk melindungi data dengan cara menerapkan metode generalisasi terhadap nilai *quasi-identifier* yang bersifat unik untuk diubah menjadi nilai yang lebih umum. Nilai dari *sensitive attribute* tidak perlu dibuat pohon DGH. Berikut adalah penerapan DGH terhadap data *credit score*.

Diketahui kemungkinan nilai unik atribut pada data *credit score* sebagai berikut:

- DAYS_EMPLOYED = {-4542,-3051,-1134}
- INCOME_TYPE = {Working, Pensioner}
- OCCUPATION_TYPE = {Security staff, Sales staff}

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

- Domain dengan generalisasi rentang nilai
Domain ini dipilih apabila tujuannya adalah melakukan proses anonimisasi pada data numerik. Contohnya, nilai dari atribut DAYS_EMPLOYED yang semula adalah {-4542,-3051,-1134} telah diubah menjadi $[(-1134) - (-4542)]$ sehingga apabila dilakukan proses *data mining*, nilai data yang telah diubah menjadi rentang nilai akan sulit untuk dilakukan identifikasi karena di dalam satu data telah mengandung beberapa nilai.
- Domain dengan generalisasi nilai yang lebih umum
Domain ini dipilih apabila tujuannya adalah melakukan proses anonimisasi pada data kategorikal. Contohnya, nilai dari atribut INCOME_TYPE yang semula adalah {Working, Pensioner} telah diubah menjadi *Unique Earnings* sehingga apabila dilakukan proses *data mining*, nilai data yang telah diubah menjadi nilai yang lebih umum akan sulit untuk dilakukan identifikasi karena di dalam satu data telah mengandung beberapa nilai.
- Domain dengan supresi nilai. Domain ini dipilih jika lebih mengutamakan perlindungan data. Biasanya domain ini jarang dipilih, karena hasil anonimisasinya kurang informatif sehingga pemodelan *data mining* tidak akan dapat untuk mencari pola dari masing-masing data diakibatkan semua datanya serupa satu sama lain. Contoh domain ini dapat dilihat pada Gambar 3.7, dimana nilai ZIP diubah menjadi {*****}



Gambar 3.7: DGH dan VGH pada atribut ZIP

3.2.8 K-Anonymity

- 2 Pada bagian 2.5, dijelaskan konsep anonimisasi. *K-anonymity* bertujuan untuk menyamarkan nilai
 3 dari masing *quasi-identifier* yang unik pada kelompok *cluster* yang sama. Kata kuncinya adalah
 4 nilai unik pada kelompok *cluster* yang sama. Setelah dataset dilakukan anonimisasi, maka data
 5 privasi sudah terlindungi sehingga publikasi data dapat dilakukan dengan aman. Tabel 3.5 adalah
 6 kelompok data yang dihasilkan oleh algoritma *greedy k-member clustering*.

Tabel 3.5: Hasil Pengelompokan Data (Greedy k-member clustering)

ID	DAYS_EMPLOYED	INCOME_TYPE	OCCUPATION_TYPE	INCOME_TOTAL	Cluster
t1	-4542	Working	Security staff	427500	Cluster 1
t2	-4542	Working	Security staff	427500	Cluster 1
t5	-3051	Pensioner	Sales staff	270000	Cluster 1
t3	-1134	Working	Sales staff	112500	Cluster 2
t4	-3051	Pensioner	Sales staff	270000	Cluster 2

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

$$\text{DAYS_EMPLOYED} = [(-4542) - (-1134)]$$

INCOME_TYPE = Unique Earnings

OCCUPATION_TYPE = Employee

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

- 8 1. Diketahui *quasi-identifier* sebagai berikut $QI = \{\text{DAYS_EMPLOYED}, \text{INCOME_TYPE}, \text{OCCUPATION_TYPE}\}$
 9 dan *sensitive attribute* sebagai berikut $SA = \{\text{INCOME_TOTAL}\}$
- 10 2. Mencari nilai *quasi-identifier* yang unik pada kelompok *cluster* yang sama. Sebagai contoh,
 11 *cluster 2* memiliki nilai *quasi-identifier* yang unik sebagai berikut $QI = \{\text{INCOME_TYPE}\}$
- 12 3. Melakukan generalisasi DGH pada nilai *quasi-identifier* yang unik menjadi nilai yang lebih
 13 umum. Sebagai contoh, $QI = \{\text{INCOME_TYPE}, \text{DAYS_EMPLOYED}\}$ memiliki nilai yang unik
 14 sehingga diubah menjadi $\{\text{INCOME_TYPE}\} = \text{Unique Earnings}$, $\text{DAYS_EMPLOYED} = [(-4542) -$
 15 $(-1134)]$
- 16 4. *Sensitive attribute* tidak akan dilakukan generalisasi, karena *quasi-identifier* sudah dilakukan
 17 generalisasi sehingga seseorang akan sulit untuk menebak kepemilikan dari *sensitive attribute*.
- 18 5. Ulangi hal yang sama pada langkah sebelumnya untuk setiap *cluster*. Hasil akhir dari proses
 19 anonimisasi ada pada Tabel 3.6 sebagai berikut:

Tabel 3.6: Hasil Anonimisasi Data (K-Anonymity)

ID	DAYS_EMPLOYED	INCOME_TYPE	OCCUPATION_TYPE	INCOME_TOTAL
t1	[(-4542)-(-3051)]	Unique Earnings	Employee	427500
t2	[(-4542)-(-3051)]	Unique Earnings	Employee	427500
t5	[(-4542)-(-3051)]	Unique Earnings	Employee	270000
t3	[(-1134)-(-3051)]	Unique Earnings	Sales staff	112500
t4	[(-1134)-(-3051)]	Unique Earnings	Sales staff	270000

3.3 Eksplorasi Spark

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

4
5 Berikut adalah beberapa hal penting terkait Spark:

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

3.3.1 Instalasi Spark

20 Spark berjalan pada sistem operasi Windows, Linux, dan Mac OS. Spark dapat dijalankan secara
21 lokal menggunakan satu komputer, meskipun Spark tetap membutuhkan beberapa komputer untuk
22 pemrosesan data yang besar. Jenis instalasi Spark dijelaskan pada bagian [2.13.2](#). Pada penelitian
23 ini digunakan jenis instalasi Standalone untuk Spark versi 2.4.5 pada sistem operasi Windows.
24 Sebelum melakukan instalasi Spark, ada beberapa hal yang harus diperhatikan dan dipenuhi.

25
26 Berikut adalah beberapa hal yang harus dipenuhi:

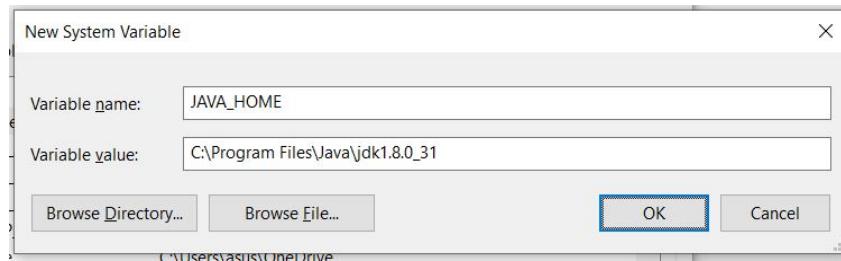
- 27 • Spark 2.4.5 dapat berjalan di Java 8, Python 2.7+/3.4+ dan R 3.1+
- 28 • Spark 2.4.5 dapat menggunakan Scala 2.12
- 29 • Spark 2.4.5 dapat menggunakan Hadoop 2.7

30
31 Berikut adalah tahapan instalasi Spark 2.4.5 secara umum:

- 32 1. Melakukan instalasi Java 8.
- 33 2. Melakukan instalasi Spark 2.4.5
- 34 3. Melakukan instalasi IntelliJ untuk Scala sbt.

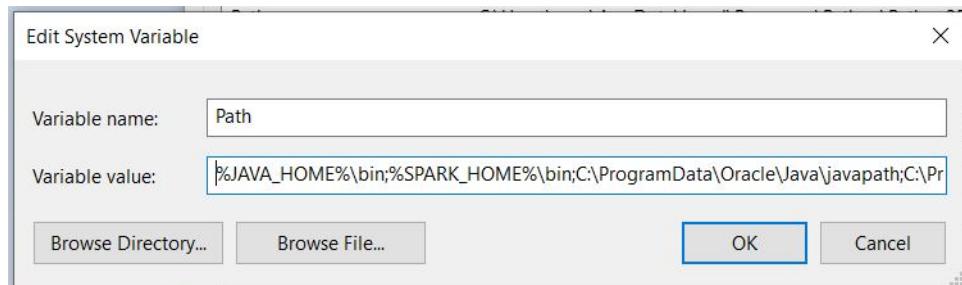
1 Instalasi Java 8

- 2 Berikut adalah tahapan instalasi Java 8 secara lengkap:
 - 3 1. Download Java SE Development Kit 8u31 pada link berikut <https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html>
 - 5 2. Lakukan instalasi Java SE Development Kit 8u31 seperti biasa.
 - 6 3. Pilih menu *Edit the system environment variables*.
 - 7 4. Buat *environment variables* baru seperti Gambar 3.8.



Gambar 3.8: Environment Variables

- 8 5. Tambahkan %JAVA_HOME%\bin; pada Path di System variables seperti Gambar 3.12.



Gambar 3.9: Penambahan Variable Value

- 9 Berikut adalah tahapan verifikasi terhadap instalasi Java 8:

- 10 1. Pilih menu *command prompt*.
- 11 2. Jalankan perintah `java -version` pada *command prompt*.

```
Command Prompt
Microsoft Windows [Version 10.0.17763.1158]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\asus>java -version
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
```

Gambar 3.10: Perintah `java -version`

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

1 Instalasi Spark 2.4.5

- 2 Berikut adalah tahapan instalasi Spark 2.4.5 secara lengkap:

3 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`

4

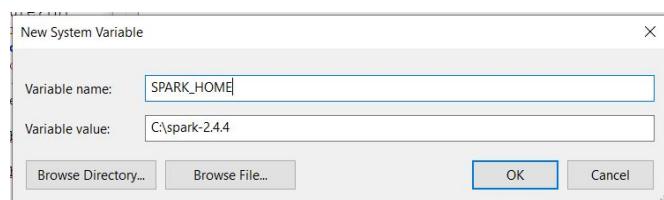
5 2. Download Spark 2.4.5 dari link <https://downloads.apache.org/spark/spark-3.0.0-preview2/spark-3.0.0-preview2-bin-hadoop2.7.tgz>

6

7 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.

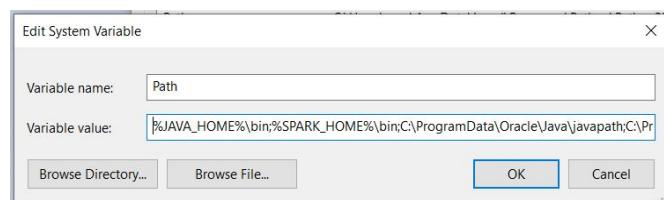
8

9 4. Buat *environment variables* baru seperti Gambar 3.11.



Gambar 3.11: Environment Variable

- 10 5. Tambahkan %SPARK_HOME%\bin; pada Path di System variables seperti Gambar 3.12

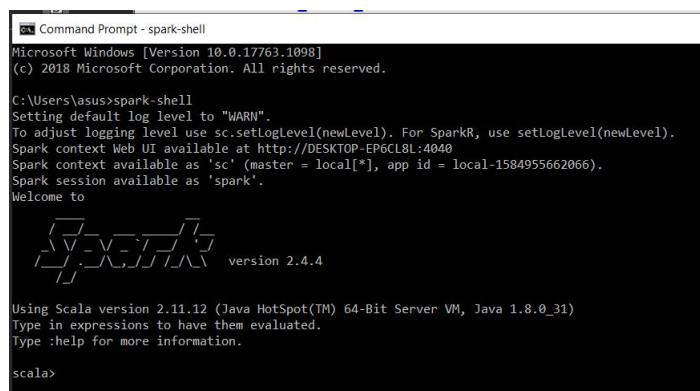


Gambar 3.12: Penambahan Variable Value

- ¹¹ Berikut adalah tahapan verifikasi terhadap instalasi Spark 2.4.5:

¹² 1. Jalankan perintah `spark-shell` pada *command prompt*.

¹³ 2. Apabila terminal menampilkan tampilan seperti pada Gambar 3.13, artinya Spark 2.4.5 sudah
¹⁴ dapat berjalan dengan baik pada komputer tersebut.



Gambar 3.13: Spark 2.4.5

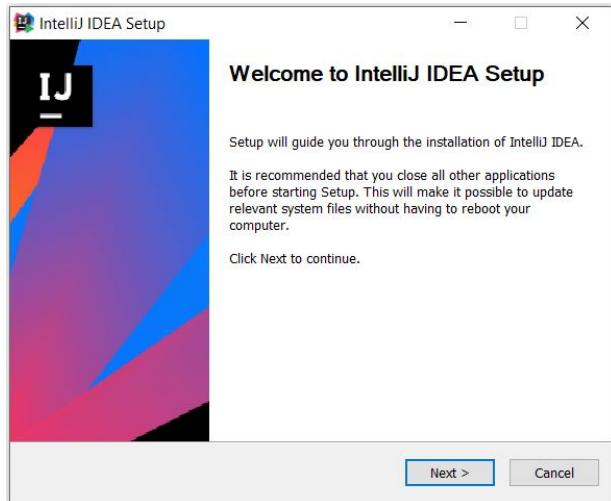
1 Instalasi IntelliJ untuk Scala SBT

2 Berikut adalah tahapan instalasi IntelliJ:

3 1. Download IntelliJ melalui link berikut

4 <https://www.jetbrains.com/idea/download/#section=windows>

5 2. Lakukan instalasi IntelliJ seperti biasa.

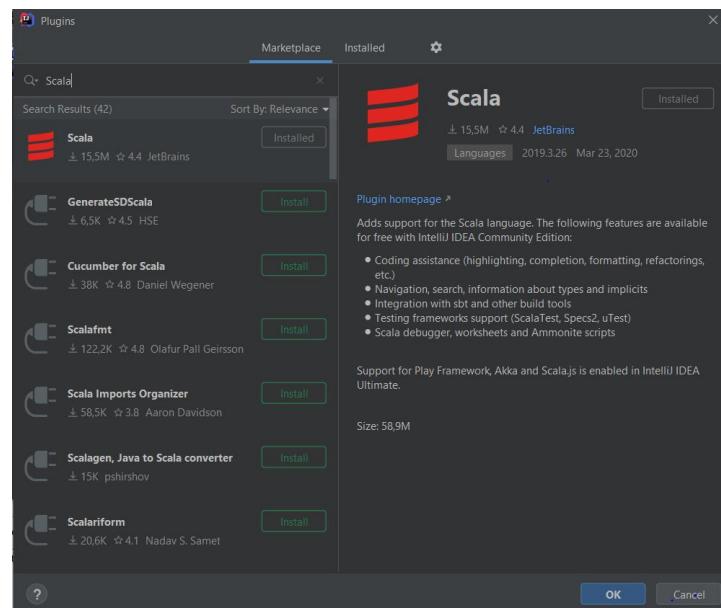


Gambar 3.14: Instalasi IntelliJ

6 Berikut adalah tahapan pemasangan *plugin* Scala pada IntelliJ.

7 1. Pilih menu *Configure* pada IntelliJ, lalu pilih menu *Plugins*.

8 2. Telusuri *plugin* Scala pada kolom pencarian seperti Gambar 3.15.



Gambar 3.15: Plugins Scala

9 3. Klik tombol *install*

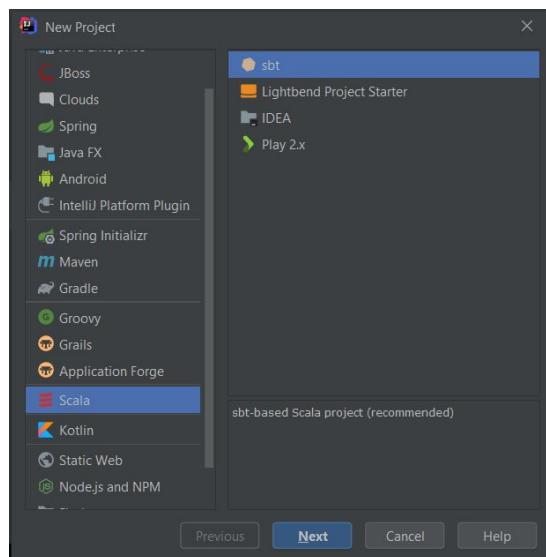
3.3.2 Membuat Project Spark pada IntelliJ

Untuk membuat program Spark, pertama-tama 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.

6

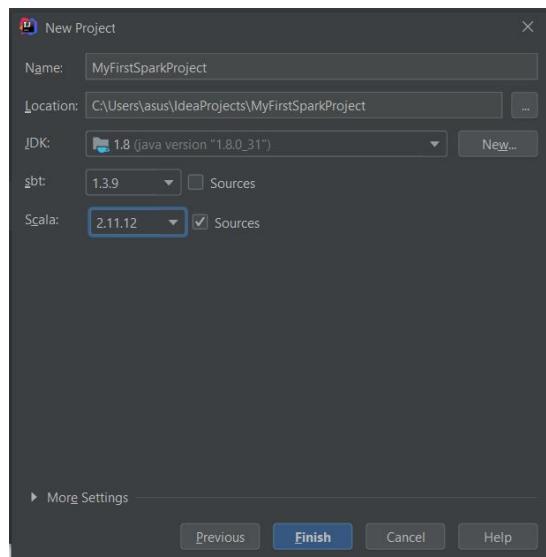
Berikut adalah tahapan pembuatan *project* Spark pada IntelliJ:

- 8 1. Memilih menu *Create New Project*
- 9 2. Menggunakan bahasa pemrograman Scala berbasis sbt seperti Gambar 3.16.



Gambar 3.16: Memilih Bahasa Scala Berbasis sbt

- 10 3. Melakukan konfigurasi pada project Spark baru seperti Gambar 3.17.



Gambar 3.17: Melakukan Konfigurasi Project Spark

- 1 4. Listing 3.2 adalah perintah *import libraryDependencies* pada file *build.sbt*

2 Contoh: spark-core, spark-sql, spark-mllib.

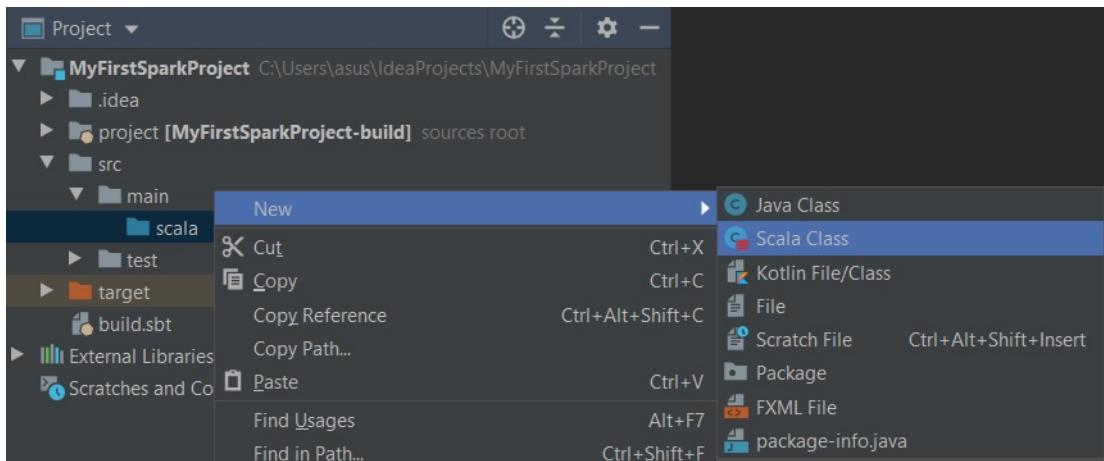
Listing 3.2: Melakukan Import Library Spark

```

3 name := "NamaProject"
4
5 version := "0.1"
6
7 scalaVersion := "2.11.12"
8
9 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
10 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
11 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
12 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.0"
13 // https://mvnrepository.com/artifact/org.apache.spark/spark-mllib
14 libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.4.3"

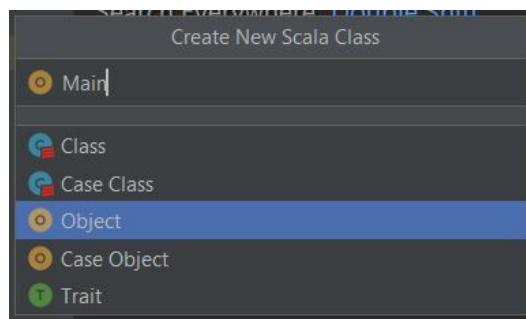
```

- 14 5. Menambahkan *Scala class* pada *src/main/scala* seperti Gambar 3.18.



Gambar 3.18: Menambahkan Scala Class pada Project Spark

- 15 6. Memilih tipe *Scala class* sebagai *Object* seperti Gambar 3.19.



Gambar 3.19: Memilih Tipe Object pada Scala Class

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

Listing 3.3: Menambahkan Main method pada Scala Class

```

1 object Main {
2     def main(args: Array[String]) {
3         //statement
4     }
5 }
6
7

```

3.3.3 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:
 - Membuka *command prompt* pada komputer lokal.
 - Menjalankan perintah eksekusi Spark sebagai berikut `spark-submit --class NamaMainClass --master local[*] lokasi_jar\nama_jar.jar` pada *command prompt*.
 - Menunggu proses eksekusi *file JAR* oleh komputer lokal, apabila terminal tidak menampilkan pesan error maka program Spark berhasil dijalankan dengan baik.
 - Output yang dihasilkan oleh program Spark akan ditampilkan pada terminal *command prompt*.

3.3.4 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*:
 - Membuka *command prompt* pada komputer lokal.
 - Menyambungkan jaringan komputer lokal dengan server *hadoop* menggunakan perintah `ssh hduser@10.100.69.101` pada *command prompt*.
 - 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*.
 - Menjalankan perintah eksekusi Spark sebagai berikut `spark-submit --class NamaMainClass --master yarn lokasi_jar\nama_jar.jar` pada *command prompt*.
 - Menunggu proses eksekusi *file JAR* oleh *hadoop cluster*, apabila terminal tidak menampilkan pesan *error* maka program Spark berhasil dijalankan dengan baik.

¹ 3.4 Studi Kasus

² Untuk memahami implementasi algoritma anonimisasi pada Spark, maka dilakukan studi kasus
³ terhadap fungsi Spark yang umum digunakan, seperti fungsi dasar pada Spark, fungsi dasar pada
⁴ komponen Spark, dan fungsi dasar pada Spark MLlib. Bentuk dari studi kasus yang akan dilakukan
⁵ adalah memberikan contoh kode program berikut penjelasan singkat mengenai tujuan pemanggilan
⁶ fungsi, parameter input, dan contoh output yang dikeluarkan oleh fungsi tersebut.

⁷ 3.4.1 Eksperimen Scala

⁸ Pada bagian [2.15](#) telah dijelaskan tujuan dari penggunaan bahasa Scala. Scala digunakan pada
⁹ penelitian ini karena sintaks yang sederhana untuk mengimplementasi beberapa baris kode pada
¹⁰ bahasa pemrograman Java. Berikut contoh beberapa eksperimen yang dilakukan pada bahasa Scala.

¹¹ Menentukan Jenis Variabel pada Scala

¹² Scala memiliki dua jenis variabel yaitu *immutable* variabel dan *mutable* variabel. *Immutable* variabel
¹³ adalah variabel yang nilainya tidak dapat diubah, sedangkan *mutable* variabel adalah variabel yang
¹⁴ nilainya dapat diubah. Implementasi *immutable* dan *mutable* memiliki implementasi sintaks yang
¹⁵ berbeda. *Immutable* variabel menggunakan sintaks `val` sedangkan *mutable* variabel menggunakan
¹⁶ sintaks `var`. Kode program dapat dilihat pada Listing [3.4](#) mengenai jenis variabel pada Scala.

Listing 3.4: Menentukan Jenis Variabel pada Scala

```
17
18
19 // Immutable Variabel
20 val donutsToBuy: Int = 5
21 donutsToBuy = 10
22 // Mutable Variabel
23 var favoriteDonut: String = "Glazed Donut"
24 favoriteDonut = "Vanilla Donut"
25
```

²⁶ Menentukan Jenis Tipe Data pada Scala

²⁷ Scala memiliki jenis tipe data yang mirip dengan tipe data pada bahasa pemrograman Java. Scala
²⁸ dapat menangani tipe data *Int*, *Long*, *Short*, *Double*, *Float*, *String*, *Byte*, *Char* dan *Unit*. Kode
²⁹ program dapat dilihat pada Listing [3.5](#) mengenai jenis tipe data pada Scala.

Listing 3.5: Menentukan Jenis Tipe Data pada Scala

```
30
31 val donutsBought: Int = 5
32 val bigNumberOfDonuts: Long = 10000000
33 val smallNumberOfDonuts: Short = 1
34 val priceOfDonut: Double = 2.50
35 val donutPrice: Float = 2.50f
36 val donutFirstLetter: Char = 'D'
37 val nothing: Unit = ()
```

1 Menentukan Struktur Data pada Scala

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

7 Listing 3.6: Membuat immutable collection pada Scala

```
8
9 // List
10 val list1: List[String] = List("Plain Donut", "Strawberry Donut", "Chocolate Donut
11   ")
12 println(s"Elements of list1 = $list1")
13
14 // Map
15 val map1: Map[String, String] = Map(("PD", "Plain Donut"), ("SD", "Strawberry Donut
16   "), ("CD", "Chocolate Donut"))
17 println(s"Elements of map1 = $map1")
18
```

19 Listing 3.7: Membuat mutable collection pada Scala

```
20
21 // Array
22 val array1: Array[String] = Array("Plain Donut", "Strawberry Donut", "Chocolate
23   Donut")
24 println(s"Elements of array1 = ${array1.mkString(", ")}")
25
26 // Map
27 val map1: Map[String, String] = Map(("PD", "Plain Donut"), ("SD", "Strawberry Donut
28   "), ("CD", "Chocolate Donut"))
29 println(s"Elements of map1 = $map1")
30
```

31 Membuat Kelas pada Scala

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

34 Listing 3.8: Membuat Kelas Object pada Scala

```
35 class AreaOfRectangle
36 {
37   var length = 20; // Variables
38   var height = 40;
39
40   def area() // Method which gives the area of the rectangle
```

```

1   {
2       var ar = length * height;
3       println("Area of the rectangle is :" + ar);
4   }
5 }

```

7 Membuat *Singleton Object* pada Scala

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

Listing 3.9: Membuat Kelas Object pada Scala

```

11 object Main
12 {
13     def main(args: Array[String])
14     {
15         // Creating object of AreaOfRectangle class
16         var obj = new AreaOfRectangle();
17         obj.area();
18     }
19 }
20
21

```

22 Membuat Fungsi Sederhana pada Scala

- 23 Scala menggunakan fungsi untuk menempatkan kode program berdasarkan tujuan masing-masing.
- 24 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 Sedehana pada Scala

```

27 def calculateDonutCost(donutName: String, quantity: Int): Double = {
28     println(s"Calculating cost for $donutName, quantity = $quantity")
29
30
31     // make some calculations ...
32     2.50 * quantity
33 }
34

```

35 Membuat Fungsi Percabangan

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

Listing 3.11: Membuat Fungsi Percabangan pada Scala

```

1 # If-Else statement
2 if(numberOfPeople > 10) {
3     println(s"Number of donuts to buy = ${numberOfPeople * donutsPerPerson}")
4 }
5 else if (numberOfPeople == 0) {
6     println("Number of people is zero.")
7     println("No need to buy donuts.")
8 }
9 else {
10     println(s"Number of donuts to buy = $defaultDonutsToBuy")
11 }
12 }
```

14 Membuat Fungsi Perulangan pada Scala

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

Listing 3.12: Membuat Fungsi Perulangan pada Scala

```

18
19 # For loop
20 for(numberOfDonuts <- 1 to 5){
21     println(s"Number of donuts to buy = $numberOfDonuts")
22 }
23 # While loop
24 while (numberOfDonutsToBake > 0) {
25     println(s"Remaining donuts to be baked = $numberOfDonutsToBake")
26     numberOfDonutsToBake -= 1
27 }
28 # Do-while loop
29 do {
30     numberOfDonutsBaked += 1
31     println(s"Number of donuts baked = $numberOfDonutsBaked")
32 }
33 while (numberOfDonutsBaked < 5)
34 }
```

35 3.4.2 Eksperimen Spark

36 Spark adalah teknologi yang digunakan untuk mengolah *big data* berdasarkan konsep dari bagian
37 [2.13](#). Spark membagi satu pekerjaan pada masing-masing *Worker Node* seperti pada bagian [2.13.1](#).
38 Oleh karena itu Spark memecah data partisi data agar data yang besar dapat distribusikan ke
39 masing-masing komputer. Berikut adalah beberapa fungsi dasar Spark untuk mengolah partisi data.

1 Melakukan Konfigurasi Spark

2 Berikut adalah tahapan konfigurasi Spark pada Main Class:

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

7 Listing 3.13: Konfigurasi Spark

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

13 Membuat RDD

14 Pada bagian [2.13.3](#), sudah dijelaskan konsep RDD. Pada bagian ini, akan dilakukan eksperimen
15 mengenai jenis-jenis cara untuk membuat RDD pada Spark.

16
17 Berikut adalah beberapa cara untuk membuat RDD:

- 18 • Membaca data eksternal pada Spark sebagai RDD
19 • Membuat RDD dari struktur data *List*
20 • Merubah DataFrame menjadi RDD

21 Listing 3.14: Cara Pembuatan RDD

```
22 # 1. Membaca data eksternal pada Spark sebagai RD  
23 rdd = sc.textFile("path")  
24 # 2. Membuat RDD dari struktur data list  
25 rdd = sc.parallelize(["id","name","3","5"])  
26 # 3. Merubah Dataframe menjadi RDD  
27 rdd = df.rdd
```

29 Membuat Dataframe

30 Pada bagian [2.13.4](#), sudah dijelaskan konsep *DataFrame*. Pada bagian ini, akan dilakukan eksperimen
31 mengenai jenis-jenis cara untuk membuat *DataFrame* pada Spark.

32
33 Berikut adalah beberapa cara untuk membuat *DataFrame*:

- 34 • Membaca data eksternal sebagai *DataFrame*

- 1 • Mengubah RDD menjadi *DataFrame* dengan nama kolom
- 2 • Mengubah RDD menjadi *DataFrame* dengan skema

Listing 3.15: Cara Pembuatan Dataframe

```

3 # 1. Membaca data eksternal sebagai Dataframe
4 # header and schema are optional
5 df = sqlContext.read.csv("path", header = True/False, schema=df_schema)
6 # 2.1 Mengubah RDD menjadi Dataframe dengan nama kolom
7 df = spark.createDataFrame(rdd, ["name", "age"])
8 # 2.2 Mengubah RDD menjadi Dataframe dengan skema
9 from pyspark.sql.types import *
10 df_schema = StructType([
11     ... StructField("name", StringType(), True),
12     ... StructField("age", IntegerType(), True)])
13 df = spark.createDataFrame(rdd, df_schema)
14
15

```

16 Memanggil Fungsi *Transformation*

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

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

Listing 3.16: Contoh Fungsi Transformation

```

25
26 # 1. select
27 df.select(df.name)
28 df.select("name")
29 # 2. filter/where
30 df.filter(df.age>20)
31 df.filter("age>20")
32 df.where("age>20")
33 df.where(df.age>20)
34 # 3. sort/orderBy
35 df.sort("age", ascending=False)
36 df.orderBy(df.age.desc())

```

```

1 # 4. groupBy dan agg
2 df.groupBy("gender").agg(count("name"),avg("age"))
3 # 5. join
4 df1.join(df2, (df1.x1 == df2.x1) & (df1.x2 == df2.x2),'left')

```

6 Memanggil Fungsi *Action*

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

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

Listing 3.17: Contoh Fungsi Action

```

14
15 # 1. show()
16 df.show(5)
17 # 2. take()
18 df.take(5)
19 # 3. collect()
20 df.collect()
21 # 4. count()
22 df.count()
23 # 6. printSchema()
24 df.printSchema()
25 # 7. transformation, action
26 df1.filter("age>10").join(df2,df1.x==df2.y).sort("age").show()
27

```

28 Memanggil Fungsi RDD

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

- 31 • repartition(n): membagi RDD menjadi n buah partisi
- 32 • cache(): menyimpan RDD pada penyimpanan memori.
- 33 • persist(): menyimpan RDD pada penyimpanan memori atau disk.
- 34 • unpersist(): menghapus RDD pada memori atau disk.

- 1 • foreach println: melakukan print seluruh baris data pada RDD
- 2 • saveAsTextFile(path): menyimpan RDD pada sebuah file

Listing 3.18: Contoh Fungsi RDD

```

3 rdd.repartition(4)
4 rdd.cache()
5 rdd.persist()
6 rdd.unpersist()
7 rdd.foreach(println)
8 rdd.saveAsTextFile(path)
9
10

```

11 Membuat Variabel Global

12 Listing 3.19 adalah perintah untuk membuat variabel global pada Spark.

Listing 3.19: Membuat Variabel Global

```

13 val broadcastVar = sc.broadcast(Array(1, 2, 3))
14
15

```

16 3.4.3 Eksperimen Komponen Spark

17 Pada bagian 2.13.5 dijelaskan mengenai jenis-jenis komponen Spark dan tujuan penggunaannya.

18 Pada bagian ini akan dilakukan eksperimen berdasarkan masing-masing jenis komponen Spark.

19 Spark Core

20 Berikut adalah langkah-langkah eksperimen dari Spark Core:

21 1. Listing 3.20 adalah membuat perintah SparkSession untuk inisialisasi project Spark

Listing 3.20: Membuat SparkSession

```

22
23 val spark: SparkSession = SparkSession.builder()
24   .master("local[3]")
25   .appName("SparkCoreAdults")
26   .getOrCreate()
27

```

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

Listing 3.21: Melihat dan Mengatur Partisi RDD

```

29
30 val sc = spark.sparkContext
31 val rdd: RDD[String] = sc.textFile("input/adult100k.csv")
32 println("initial partition count:" + rdd.getNumPartitions)
33 val reparRdd = rdd.repartition(4)
34 println("re-partition count:" + reparRdd.getNumPartitions)
35

```

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

Listing 3.22: Membuat Fungsi Transformation

```

2 //Transformation - flatMap
3 val rdd2 = rdd.flatMap(f => f.split(","))
4 rdd2.foreach(f => println(f))
5 //Transformation - map
6 val rdd3: RDD[(String, Int)] = rdd2.map(key => (key, 1))
7 rdd3.foreach(println)
8 //Transformation - filter
9 val rdd4 = rdd3.filter(a => a._1.startsWith("State-gov"))
10 rdd4.foreach(println)
11 //Transformation - reduceByKey
12 val rdd5 = rdd3.reduceByKey((x,y)=> x + y)
13 rdd5.foreach(println)
14 //Transformation - sortByKey
15 val rdd6 = rdd5.map(a => (a._2, a._1)).sortByKey()
16
17

```

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

Listing 3.23: Membuat Fungsi Action

```

19 //Action - count
20 println("Count : " + rdd6.count())
21 //Action - first
22 val firstRec = rdd6.first()
23 println("First Record : " + firstRec._1 + "," + firstRec._2)
24 //Action - max
25 val datMax = rdd6.max()
26 println("Max Record : " + datMax._1 + "," + datMax._2)
27 //Action - reduce
28 val totalWordCount = rdd6.reduce((a, b) => (a._1 + b._1, a._2))
29 println("dataReduce Record : " + totalWordCount._1)
30 //Action - take
31 val data3 = rdd6.take(3)
32 data3.foreach(f => {
33     println("data3 Key:" + f._1 + ", Value:" + f._2)
34 })
35 //Action - collect
36 val data = rdd6.collect()
37 data.foreach(f => {
38     println("Key:" + f._1 + ", Value:" + f._2)
39 })
40 //Action - saveAsTextFile
41

```

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

3 Spark SQL

4 Berikut adalah langkah-langkah eksperimen dari Spark Core:

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

Listing 3.24: Membuat Perintah SparkSession

```
6 val spark = SparkSession
7     .builder.master("local[*]")
8     .appName("SparkSQL")
9     .getOrCreate()
10
11
12
```

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

Listing 3.25: Membuat Dataframe

```
14 val peopleDF = spark.read
15     .format("csv")
16     .option("header", "true")
17     .option("delimiter", ",")
18     .load("input/adult100k.csv")
19
20 peopleDF.printSchema()
```

- 22 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

```
24 // Query
25 peopleDF.createTempView("tAdults")
26 val query = spark.sql(
27     "SELECT workclass,count(education) as count_people"+
28     "FROM tAdults " +
29     "WHERE lower(education) == 'bachelors'" +
30     "GROUP BY workclass " +
31     "ORDER BY count_people DESC " +
32     "LIMIT 10"
33 )
34 query.write.csv("C:/Users/asus/Desktop/resultquery1.csv")
35
36
37
```

- 38 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

```

1 // Statistika: count, mean, stddev, min, max
2 peopleDF.describe().show()
3
4

```

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

Listing 3.28: Mencari Nilai Median

```

6 // Median
7 val median = spark.sql(
8     "SELECT percentile_approx(age, 0.5) as Median " +
9     "FROM tAdults"
10    ).show()
11
12

```

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

Listing 3.29: Mencari Nilai Modus

```

14 // Modus
15 val modus = spark.sql(
16     "SELECT age as Modus " +
17     "FROM tAdults " +
18     "GROUP BY age " +
19     "ORDER BY COUNT(age) DESC " +
20     "LIMIT 1"
21    ).show()
22
23

```

24 Spark MLlib

- 25 Berikut adalah langkah-langkah eksperimen dari Spark Core:

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

Listing 3.30: Membuat Perintah SparkSession

```

27 val spark = SparkSession
28     .builder.master("local[*]")
29     .appName("SparkMLlib")
30     .getOrCreate()
31
32

```

34. 2. Listing 3.31 adalah perintah untuk membuat skema untuk *Dataframe*.

Listing 3.31: Membuat Skema Dataframe

```

35 val schema = StructType(
36     List(
37         StructField("age", IntegerType, true),
38

```

```

1      StructField("workclass", StringType, true),
2      StructField("fnlwgt", IntegerType, true),
3      StructField("education", StringType, true),
4      StructField("education-num", IntegerType, true),
5      StructField("marital-status", StringType, true),
6      StructField("occupation", StringType, true),
7      StructField("relationship", StringType, true),
8      StructField("race", StringType, true),
9      StructField("sex", StringType, true),
10     StructField("capital-gain", IntegerType, true),
11     StructField("capital-loss", IntegerType, true),
12     StructField("hours-per-week", IntegerType, true),
13     StructField("native-country", StringType, true),
14     StructField("salary", StringType, true)
15   )
16 )
17

```

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

Listing 3.32: Mengubah CSV Menjadi Dataframe

```

20
21 val adult100k_df = spark.read
22   .format("csv")
23   .option("header", "false")
24   .option("delimiter", ",")
25   .schema(schema)
26   .load("input/adult100k.csv")
27 adult100k_df.show()
28

```

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

Listing 3.33: Membuat Kolom Index

```

31
32 // Create vector based on stringIndexer and oneHotEncoder
33 val cols = adult100k_df.columns
34 val encodedFeatures = cols.flatMap{columnName =>
35   val stringIndexer = new StringIndexer()
36     .setInputCol(columnName)
37     .setOutputCol(columnName + "_Index")
38   val oneHotEncoder = new OneHotEncoderEstimator()
39     .setInputCols(Array(columnName + "_Index"))
40     .setOutputCols(Array(columnName + "_vec"))
41     .setDropLast(false)

```

```

1         Array(stringIndexer.setHandleInvalid("keep"), oneHotEncoder)
2     }

```

- 4 5. Listing 3.34 adalah perintah untuk menambahkan kolom vektor dan kolom indeks pada
 $DataFrame$.

Listing 3.34: Membuat Kolom Vektor

```

6 // Pipeline
7 val pipeline = new Pipeline().setStages(encodedFeatures)
8 val indexer_model = pipeline.fit(adult100k_df)
9
10

```

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

Listing 3.35: Memilih Jenis Vektor

```

12 // Sparse Vector
13 val df_transformed = indexer_model.transform(adult100k_df)
14 df_transformed.show()
15
16 // Dense Vector
17 val sparseToDense = udf((v: Vector) => v.toDense)
18 val df_denseVectors = df_transformed
19             .withColumn("dense_workclass_vec",
20                         sparseToDense(df_transformed("workclass_vec")))
21 df_denseVectors.show()
22

```

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

Listing 3.36: Membuat Vektor Fitur

```

24 // Final Result: Feature Vector
25 val vecFeatures = df_transformed.columns.filter(_.contains("vec"))
26 val vectorAssembler = new VectorAssembler()
27             .setInputCols(vecFeatures)
28             .setOutputCol("features")
29 val pipelineVectorAssembler = new Pipeline()
30             .setStages(Array(vectorAssembler))
31 val result_df = pipelineVectorAssembler
32             .fit(df_transformed)
33             .transform(df_transformed)
34 result_df.show()
35
36

```

37 3.4.4 Eksperimen Spark MLIB

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

1 *Naive Bayes*

2 Pada bagian 2.14.2 menjelaskan parameter pemodelan *Naive Bayes* pada Spark MLlib. Berikut
 3 adalah tahapan eksperimen pada Listing 3.37 untuk pemodelan *Naive Bayes*:

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

Listing 3.37: Eksperimen Naive Bayes Spark MLlib

```

8 // Split data into training (70%) and test (30%).
9 val Array(training, test) = result_df.randomSplit(Array(0.7, 0.3))
10 // Naive Bayes
11 val model = new NaiveBayes().setModelType("multinomial")
12           .setLabelCol("workclass_Index").fit(training)
13 // Predict model
14 val predictions = model.transform(test)
15 predictions.show()
16 // Accuracy
17 val evaluator = new MulticlassClassificationEvaluator()
18           .setLabelCol("workclass_Index")
19           .setPredictionCol("prediction")
20           .setMetricName("accuracy")
21 val accuracy = evaluator.evaluate(predictions)
22
23

```

24 Dataset yang dipakai untuk eksperimen *naive bayes* ini adalah sampel data dari dataset *credit score*
 25 yang sudah dijelaskan pada bagian. Data ini berjumlah 400.000 baris data dengan ukuran 11.000
 26 KB. Dataset ini akan dibagi menjadi data *test* dan data *training*. Jumlah data test adalah 30.000
 27 baris data, sedangkan jumlah data training adalah 70.000 data. Jumlah data training lebih banyak
 28 karena akan dipakai untuk pelatihan model *naive bayes*.

1	DAY_BIRTH, DAYS_EMPLOYED, OCCUPATION_TYPE, CODE_GENDER, NAME_INCOME_TYPE, prediction
2	-12005, -4542, "", M, Working, 1.0
3	-12005, -4542, "", M, Working, 1.0
4	-21474, -1134, Security staff, M, Working, 1.0
5	-19110, -3051, Sales staff, F, Commercial associate, 1.0
6	-19110, -3051, Sales staff, F, Commercial associate, 1.0
7	-19110, -3051, Sales staff, F, Commercial associate, 1.0
8	-19110, -3051, Sales staff, F, Commercial associate, 1.0
9	-22464, 365243, "", F, Pensioner, 0.0
10	-22464, 365243, "", F, Pensioner, 0.0

Gambar 3.20: Hasil Naive Bayes Spark MLlib

- 1 Hasil pemodelan *naive bayes* dari eksperimen ini adalah label data. *Cluster* ini terbentuk berdasarkan
 2 distance terdekat antara anggota data dengan titik *centroid* pada *cluster* tersebut. Sehingga data-
 3 data yang tergabung pada sebuah *cluster*, sudah dipastikan bahwa data-data tersebut lebih dekat
 4 dengan titik *centroid* pada cluster tersebut dibanding dengan titik *centroid* pada *cluster* lainnya.
 5 Gambar 3.20 menunjukan bahwa data dengan nilai umur yang sama (*age* = 17) memiliki kelompok
 6 *cluster* yang sama (*prediction* = 3.0).

7 **K-Means**

- 8 Pada bagian 2.14.2 menjelaskan parameter pemodelan *k-means* pada Spark MLlib. Berikut adalah
 9 tahapan eksperimen pada Listing 3.38 untuk pemodelan *k-means*:

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

Listing 3.38: Eksperimen K-Means Spark MLlib

```
15 // KMeans with 8 clusters
16 val kmeans = new KMeans()
17     .setK(8)
18     .setFeaturesCol("features")
19     .setPredictionCol("prediction")
20 val kmeansModel = kmeans.fit(result_df)
21 kmeansModel.clusterCenters.foreach(println)
22
23
24 // Predict model
25 val predictDf = kmeansModel.transform(result_df)
26 predictDf.show(10)
```

- 28 Dataset yang dipakai untuk eksperimen *k-means* ini adalah sampel data dari data *credit score* yang
 29 sudah dijelaskan pada bagian. Data ini berjumlah 400.000 baris data dengan ukuran 11.000 KB.
 30 Batas maksimum iterasi untuk fungsi *k-means* telah diatur sedemikian rupa oleh *library* Spark
 31 MLlib itu sendiri, sehingga parameter ini tidak perlu lagi diset manual oleh pengguna. Jumlah
 32 *cluster* yang akan dibentuk pada eksperimen ini berjumlah 8 *cluster*. Jumlah *cluster* pada fungsi
 33 *k-means* nantinya dapat diatur kembali sesuai kebutuhan eksperimen.

```
1 | DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,prediction
2 | -12005,-4542,"",M,Working,4
3 | -12005,-4542,"",M,Working,4
4 | -21474,-1134,Security staff,M,Working,4
5 | -19110,-3051,Sales staff,F,Commercial associate,0
6 | -19110,-3051,Sales staff,F,Commercial associate,0
7 | -19110,-3051,Sales staff,F,Commercial associate,0
8 | -19110,-3051,Sales staff,F,Commercial associate,0
9 | -22464,365243,"",F,Pensioner,6
10| -22464,365243,"",F,Pensioner,6
```

Gambar 3.21: Hasil K-Means Spark MLlib

- 1 Hasil pemodelan *k-means* dari eksperimen ini adalah *cluster* data. *Cluster* ini terbentuk berdasarkan
- 2 *distance* terdekat antara anggota data dengan titik *centroid* pada *cluster* tersebut. Sehingga data-
- 3 data yang tergabung pada sebuah *cluster*, sudah dipastikan bahwa data-data tersebut lebih dekat
- 4 dengan titik *centroid* pada *cluster* tersebut dibanding dengan titik *centroid* pada *cluster* lainnya.
- 5 Gambar 3.21 menunjukan bahwa data dengan atribut (*age* = 39) memiliki kelompok *cluster* yang
- 6 sama dengan dengan data lain dengan atribut (*age* = 38), karena berada pada kelompok data yang
- 7 sama dengan representasi nama kelompok (*prediction* = 5).

BAB 4

PERANCANGAN

- 3 Perancangan yang dibuat pada bab ini meliputi tiga perangkat lunak yaitu eksplorasi, anonimisasi,
4 dan pengujian. Pada perangkat lunak eksplorasi akan dilakukan pencarian nilai unik untuk setiap
5 atribut. Pada perangkat lunak anonimisasi akan dilakukan pemodelan algoritma *greedy k-member*
6 *clustering* dan *k-anonymity*. Pada perangkat lunak pengujian akan dilakukan pemodelan *k-means*
7 dan *naive bayes*. Pada subbab ini tidak dilakukan perancangan antarmuka perangkat lunak karena
8 pada umumnya aplikasi untuk mengolah *big data* tidak memerlukan tampilan antarmuka karena
9 dijalankan pada terminal. Perancangan perangkat lunak pada subbab ini diganti dengan perjelasan
10 antarmuka dari Spark Web UI.

Spark Stages Storage Environment Executors Spark shell application UI

Spark Stages

Total Duration: 58.8 m
Scheduling Mode: FIFO
Active Stages: 0
Completed Stages: 3
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
----------	-------------	-----------	----------	------------------------	--------------	---------------

Completed Stages (3)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
1	count at <console>:23	2014/10/01 19:47:49	378 ms	6/6		
2	map at <console>:23	2014/10/01 19:47:48	211 ms	6/6		74.5 KB
0	count at <console>:23	2014/10/01 19:45:54	623 ms	6/6		

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Shuffle Read	Shuffle Write
----------	-------------	-----------	----------	------------------------	--------------	---------------

Gambar 4.1: Spark Web UI

- Spark menyediakan Web UI yang dapat dibuka ketika aplikasi Spark sedang dijalankan. Tampilan antarmuka Spark bernama Spark Web UI. Spark dapat memantau status aplikasi yang sedang berjalan pada *hadoop cluster* melalui Spark Web UI. Gambar 4.1 dapat dibuka secara lokal dengan browser dengan melalui alamat <http://localhost:4040>. Spark menyediakan Web UI/*User Interface* yang terdiri dari menu *Jobs*, *Stages*, *Tasks*, *Storage*, *Environment*, *Executors*, dan *SQL* untuk memonitor status aplikasi Spark, melihat konsumsi sumber daya yang dipakai pada *hadoop cluster*, dan melihat konfigurasi yang digunakan untuk menjalankan aplikasi Spark. Untuk lebih memahami fungsi menu-menu yang tersedia pada Spark Web UI, pada subbab selanjutnya akan diberikan contoh menu beserta penjelasannya dengan lebih lengkap.

4.1 Penggunaan Spark Web UI

Spark Web UI terdiri dari beberapa menu sebagai berikut:

1. Spark Jobs

2. Stages

3. Tasks

4. Storage

5. Environment

6. Executors

7. SQL

Berikut beberapa alamat penting terkait penulusuran aplikasi Spark:

- Spark Application UI: <http://localhost:4040/>

- Resource Manager: <http://localhost:9870/>

- Spark JobTracker: <http://localhost:8088/>

- Node Specific Info: <http://localhost:8042/>

Spark Jobs

Total Uptime: 2.2 min

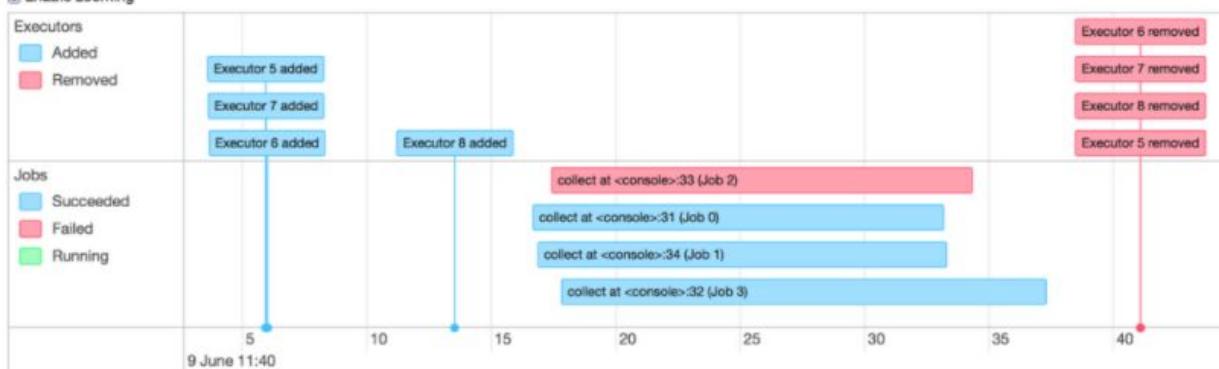
Scheduling Mode: FIFO

Completed Jobs: 3

Failed Jobs: 1

 Event Timeline

Enable zooming



Gambar 4.2: Spark Web UI

Gambar 4.2 adalah contoh halaman Spark Web UI yang dapat digunakan untuk menulusuri aplikasi Spark. Spark dapat dijalankan secara lokal (*Standalone*) maupun dengan *hadoop cluster* (YARN). Spark Web UI pada komputer lokal (*Standalone*) dapat diakses menggunakan <http://localhost:4040/>. Spark Web UI pada *cluster* diakses pada <http://10.100.69.101:4040/cluster>. Spark Web UI secara umum berjalan pada port 4040 dan dapat diganti sesuai kebutuhan.

1 4.1.1 Menu Spark Jobs

- 2 Detail pada menu ini yang perlu diketahui adalah *Scheduling Mode* dan *Completed Spark Jobs*.
 3 Gambar 4.3 merupakan isi dari menu Spark Jobs.

Spark Jobs (?)

User: sriramrimalapudi
 Total Uptime: 2.5 h
 Scheduling Mode: FIFO
Completed Jobs: 3

Gambar 4.3: Menu Spark Jobs

- 4 Berikut adalah penjelasannya:



Completed Jobs (3)					
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	count at SparkUIExample.scala:20 count at SparkUIExample.scala:20	2020/07/20 21:41:35	0.2 s	2/2	2/2
1	csv at SparkUIExample.scala:18 csv at SparkUIExample.scala:18	2020/07/20 21:41:35	0.3 s	1/1	1/1
0	csv at SparkUIExample.scala:18 csv at SparkUIExample.scala:18	2020/07/20 21:41:34	0.5 s	1/1	1/1

Gambar 4.4: Bagian Scheduling Mode (Spark Jobs)

- 5 • Scheduling Mode

6 Spark memiliki 3 Schedulling modes:

- 7 1. Standalone mode (Gambar 4.4)
 8 2. YARN mode
 9 3. Mesos

- 10 • Completed Spark Jobs

11 Jumlah pekerjaan Spark sama dengan jumlah pemanggilan fungsi action pada program Spark
 12 dan setiap pekerjaan Spark harus memiliki setidaknya satu Stage. Gambar 4.4, telah dilakukan
 13 3 pekerjaan Spark dengan Job Id (0,1,2).

- 14 1. Job 0: read the CSV file.
 15 2. Job 1: Inferschema from the file.
 16 3. Job 2: Count Check

17 4.1.2 Menu Stages

- 18 Menu Stages dapat dibuka dengan dua cara:

- 19 1. Pilih Description dari Spark Jobs untuk menampilkan Stages pada Spark Jobs yang dipilih.
 20 2. Di bagian menu Spark Web UI, pilih menu Stages untuk menampilkan semua Stages yang
 21 dieksekusi pada program Spark yang telah dijalankan.

- 1 Gambar 4.5 menunjukkan program terdiri 4 Stages.
- 2
- 3 Menu Stages menampilkan halaman ringkasan yang menunjukkan status saat ini dari semua tahapan
- 4 dari semua pekerjaan Spark di aplikasi spark. Jumlah tugas yang dapat Anda lihat di setiap tahap
- 5 adalah jumlah partisi yang akan dikerjakan oleh spark dan setiap tugas di dalam suatu tahap adalah
- 6 pekerjaan yang sama yang akan dilakukan oleh spark tetapi pada partisi data yang berbeda.
- 7

Stages for All Jobs								
Completed Stages: 4								
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	count at SparkUIExample.scala:20	+details 2020/07/20 21:41:35	57 ms	1/1			59.0 B	
2	count at SparkUIExample.scala:20	+details 2020/07/20 21:41:35	93 ms	1/1	296.6 KIB			59.0 B
1	csv at SparkUIExample.scala:18	+details 2020/07/20 21:41:35	0.2 s	1/1	296.6 KIB			
0	csv at SparkUIExample.scala:18	+details 2020/07/20 21:41:34	0.3 s	1/1	64.0 KIB			

Gambar 4.5: Menu Stage

8 4.1.3 Menu Tasks

- 9 Task terletak di paling bawah dari 0masing-masing tahap. Informasi yang dapat diketahui dengan
- 10 melihat halaman Task adalah:

- 11 1. Input Size: ukuran input pada Stage.
- 12 2. Shuffle Write: ukuran output dari proses Stage.

Tasks (75)												
Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Blocked Time	Shuffle Read Size / Records	Errors	
66	333	0	SUCCESS	NODE_LOCAL	0 / 192.168.65.1 <code>stdout</code> <code>stderr</code>	2016/10/17 20:11:22	2 ms		0 ms	126.0 B / 2		
0	334	0	SUCCESS	PROCESS_LOCAL	0 / 192.168.65.1 <code>stdout</code> <code>stderr</code>	2016/10/17 20:11:22	1 ms		0 ms	0.0 B / 0		
1	335	0	SUCCESS	PROCESS_LOCAL	0 / 192.168.65.1 <code>stdout</code> <code>stderr</code>	2016/10/17 20:11:22	2 ms		0 ms	0.0 B / 0		

Gambar 4.6: Menu Task

13 4.1.4 Menu Storage

- 14 Storage menampilkan ukuran RDD dan DataFrames yang disimpan dalam memori (jika ada).
- 15 Gambar 4.8 adalah contoh isi dari menu Storage:

Storage						
RDDs		Storage Level		Cached Partitions	Fraction Cached	Size in Memory
1	rdd	Memory Serialized	1x Replicated	5	100%	236.0 B 0.0 B
4	LocalTableScan [count#7, name#8]	Disk Serialized	1x Replicated	3	100%	0.0 B 2.1 kB

Gambar 4.7: Menu Storage

1 4.1.5 Menu Environment

- 2 Menu Environment menampilkan nilai untuk lingkungan yang berbeda dan variabel konfigurasi,
- 3 termasuk JVM, Spark, dan properti sistem.

Environment	
Runtime Information	
Name	Value
Java Home	/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/jre
Java Version	1.8.0_221 (Oracle Corporation)
Scala Version	version 2.12.8
Spark Properties	
Name	Value
spark.app.id	local-1565684968905
spark.app.name	Spark shell
spark.driver.host	10.12.221.7
spark.driver.port	58229
spark.executor.id	driver
spark.home	/Users/zrf/Dev/OpenSource/spark
spark.jars	
spark.master	local[*]
spark.repl.class.outputDir	/private/var/folders/vt/5svvtxjj6rz6syxf4ssyxlkm0000gn/T/spark-bae57fcf-54f9-48b5-8e71-cfb15c126e64/repl-e2551dc1-3350-47db-a5eb-b0148ef86325
spark.repl.class.uri	spark://10.12.221.7:58229/classes
spark.scheduler.mode	FIFO
spark.sql.catalogImplementation	in-memory
spark.submit.deployMode	client
spark.submit.pyFiles	
spark.ui.showConsoleProgress	true
Hadoop Properties	
System Properties	
Classpath Entries	

Gambar 4.8: Menu Storage

4 4.1.6 Menu SQL

- 5 Jika program menjalankan kueri Spark SQL, halaman SQL menampilkan informasi, seperti duration, jobs, physical and logical plans untuk kueri. Gambar 4.9 adalah isi dari halaman menu SQL:

SQL				
Completed Queries: 3				
Completed Queries (3)				
Page: 1		1 Pages. Jump to <input type="text" value="1"/> . Show <input type="text" value="100"/> items in a page.	<input type="button" value="Go"/>	
ID	Description	Submitted	Duration	Job IDs
2	show at <console>:24 +details	2019/08/04 15:23:15	2 s	[1][2][3][4][5]
1	createGlobalTempView at <console>:26 +details	2019/08/04 15:23:09	0.3 s	
0	count at <console>:26 +details	2019/08/04 15:23:01	2 s	[0]
Page: 1		1 Pages. Jump to <input type="text" value="1"/> . Show <input type="text" value="100"/> items in a page.	<input type="button" value="Go"/>	

Gambar 4.9: Menu Storage

7 4.1.7 Menu Executors

- 8 Menu Executor menampilkan informasi ringkas tentang pelaksana yang dibuat untuk aplikasi,
- 9 termasuk penggunaan memori dan disk serta informasi tugas dan acak. Kolom Memori Penyimpanan 10 menunjukkan jumlah memori yang digunakan dan dicadangkan untuk data cache.

Executors

[Show Additional Metrics](#)

- Select All
- On Heap Memory
- Off Heap Memory

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(3)	0	5.9 KiB / 1.1 GiB	0.0 B	2	0	0	5	5	4 s (0.2 s)	0.0 B	0.0 B	0.0 B	0
Total(3)	0	5.9 KiB / 1.1 GiB	0.0 B	2	0	0	5	5	4 s (0.2 s)	0.0 B	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries

Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Logs	Thread Dump
1	10.12.221.27:55834	Active	0	2 KiB / 366.3 MiB	0.0 B	1	0	0	3	3	2 s (0.1 s)	0.0 B	0.0 B	0.0 B	stdout	Thread Dump
0	10.12.221.27:55835	Active	0	2 KiB / 366.3 MiB	0.0 B	1	0	0	2	2	2 s (94.0 ms)	0.0 B	0.0 B	0.0 B	stdout	Thread Dump
driver	10.12.221.27:55827	Active	0	2 KiB / 366.3 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump

Showing 1 to 3 of 3 entries

Previous 1 Next

Gambar 4.10: Menu Storage

4.2 Perancangan Perangkat Lunak

- Untuk menjalankan program Spark, dibutuhkan konfigurasi tambahan untuk mencantumkan lokasi *library* Spark. Penggunaan *library* Spark dapat dipermudah dengan mencantumkan alamat dari *library* Spark yang akan dipakai. Konfigurasi *library* Spark ini nantinya disimpan pada file `build.sbt`. Gambar 4.11 adalah contoh konfigurasi yang dipakai pada pembangunan perangkat lunak eksplorasi, perangkat lunak anonimisasi, dan perangkat lunak pengujian.

7

```

1  name := "Skripsi"
2
3  version := "0.1"
4
5  scalaVersion := "2.11.12"
6
7  // https://mvnrepository.com/artifact/org.apache.spark/spark-core
8  libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.3"
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
11 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.3"
12
13 // https://mvnrepository.com/artifact/org.apache.spark/spark-mllib
14 libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.4.3"
15

```

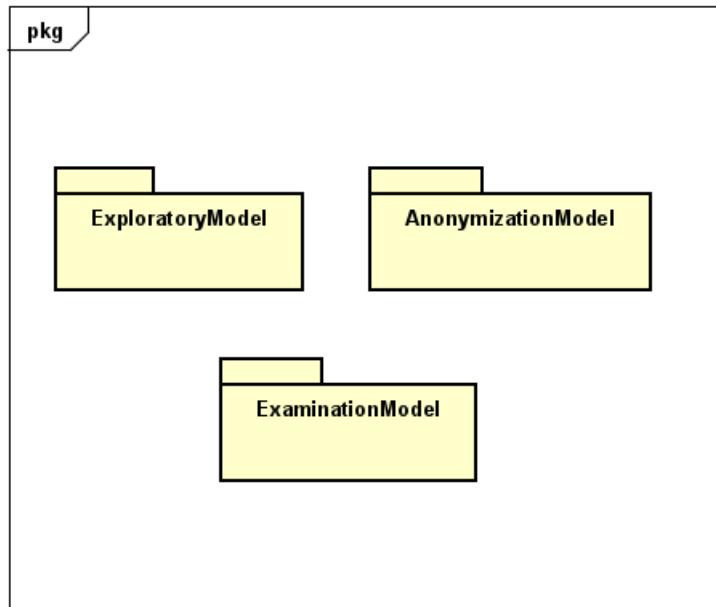
Gambar 4.11: Library Spark (build.sbt)

¹ 4.3 Diagram Kelas Lengkap

- ² Pada bagian ini, akan dibuat empat jenis diagram kelas, yaitu diagram kelas untuk package, diagram
- ³ kelas untuk perangkat lunak eksplorasi, diagram kelas untuk perangkat lunak anonimisasi, dan
- ⁴ diagram kelas untuk perangkat lunak pengujian.

⁵ 4.3.1 Diagram Package

- ⁶ Perangkat lunak ini mempunyai 3 buah *package* yang tidak saling berhubungan satu sama lain
- ⁷ yaitu *ExploratoryModel*, *AnonymizationModel*, *ExaminationModel*. Ketika package ini memiliki
- ⁸ fungsinya masing-masing. Diagram Package dapat dilihat pada Gambar 4.12



Gambar 4.12: Diagram Kelas pada Package

⁹ Package ExploratoryModel

- ¹⁰ *Package ExploratoryModel* merupakan *package* yang menangani pencarian nilai unik pada masing-masing atribut. *Package* ini hanya memiliki implementasi kelas **Main** untuk mencari nilai atribut yang unik dan mengembalikan nilai unik tersebut ke dalam tabel data yang disimpan dalam format CSV. Jumlah CSV yang dihasilkan bergantung pada jumlah atribut yang dicantumkan pada JSON.

¹⁴ Package AnonymizationModel

- ¹⁵ *Package AnonymizationModel* merupakan *package* yang menangani segala jenis fungsi yang berkaitan dengan masalah pengelompokan data dan anonimisasi data. Fungsi tersebut antara lain pengelompokan data dengan algoritma *greedy k-member clustering*, pencarian *record* dan *cluster* terbaik, perhitungan *distance* numerik dan kategorikal, dan perhitungan *information loss*. *Package* ini juga mengimplementasikan berbagai macam operasi untuk membaca atribut DGH pada JSON dan menangani pembuatan pohon generalisasi berdasarkan atribut DGH. Operasi-operasi ini diimplementasikan karena adanya kebutuhan untuk membantu proses implementasi pengelompokan

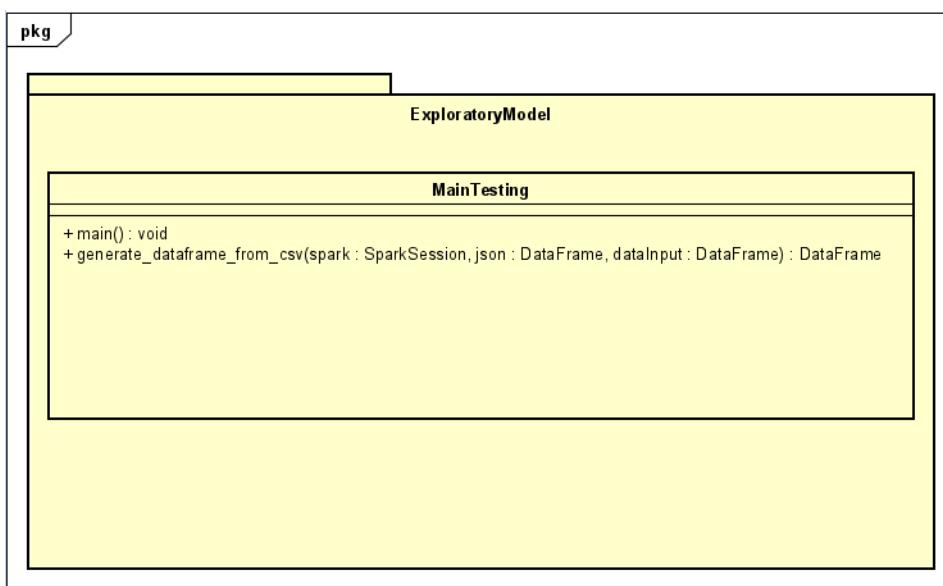
1 data dan anonimisasi data. Selain itu, *package* ini memiliki dua jenis kelas *Main* yaitu *MainTesting*
 2 dan *MainLCATesting*. Kelas *MainTesting* digunakan untuk melakukan proses pengelompokan
 3 dan anonimisasi data dan mengembalikan hasilnya ke dalam format CSV. Kelas *MainLCATesting*
 4 digunakan untuk menampilkan *root* terdekat dari kedua node.

5 **Package ExaminationModel**

6 *Package ExaminationModel* merupakan package yang menangani segala jenis fungsi yang berka-
 7itan dengan masalah pengujian data. Fungsi tersebut antara lain pengelompokan data dengan
 8 pemodelan *k-means* beserta evaluasi modelnya dengan *silhouette score* dan pemodelan *naive bayes*
 9 beserta evaluasi modelnya dengan tingkat akurasi. Package ini juga mengimplementasikan berbagai
 10 macam operasi untuk membuat DataFrame berdasarkan atribut yang telah dipilih pada JSON dan
 11 mendapatkan nilai parameter *k-means* dan *naive bayes* dari JSON. Operasi-operasi ini diimplemen-
 12 tasikan karena adanya kebutuhan untuk membantu proses pengujian data berdasarkan pemodelan
 13 yang dipilih (*k-means/naive bayes*). Selain itu, *package* ini memiliki satu jenis kelas *Main* yaitu
 14 *MainTesting*. Kelas *MainTesting* digunakan untuk melakukan pengujian data dengan pemodelan
 15 *k-means/naive bayes* dan mengembalikan hasilnya ke dalam format CSV.

16 **4.3.2 Diagram Kelas pada Package ExploratoryModel**

17 Perangkat lunak eksplorasi hanya memiliki satu jenis *package* dengan nama *ExploratoryModel*.
 18 *Package ExploratoryModel* terdiri dari satu kelas yaitu *MainTesting*. Kelas ini memiliki fungsi
 19 penting untuk menyelesaikan permasalahan pencarian nilai atribut yang unik.



Gambar 4.13: Diagram Kelas pada Package ExploratoryModel

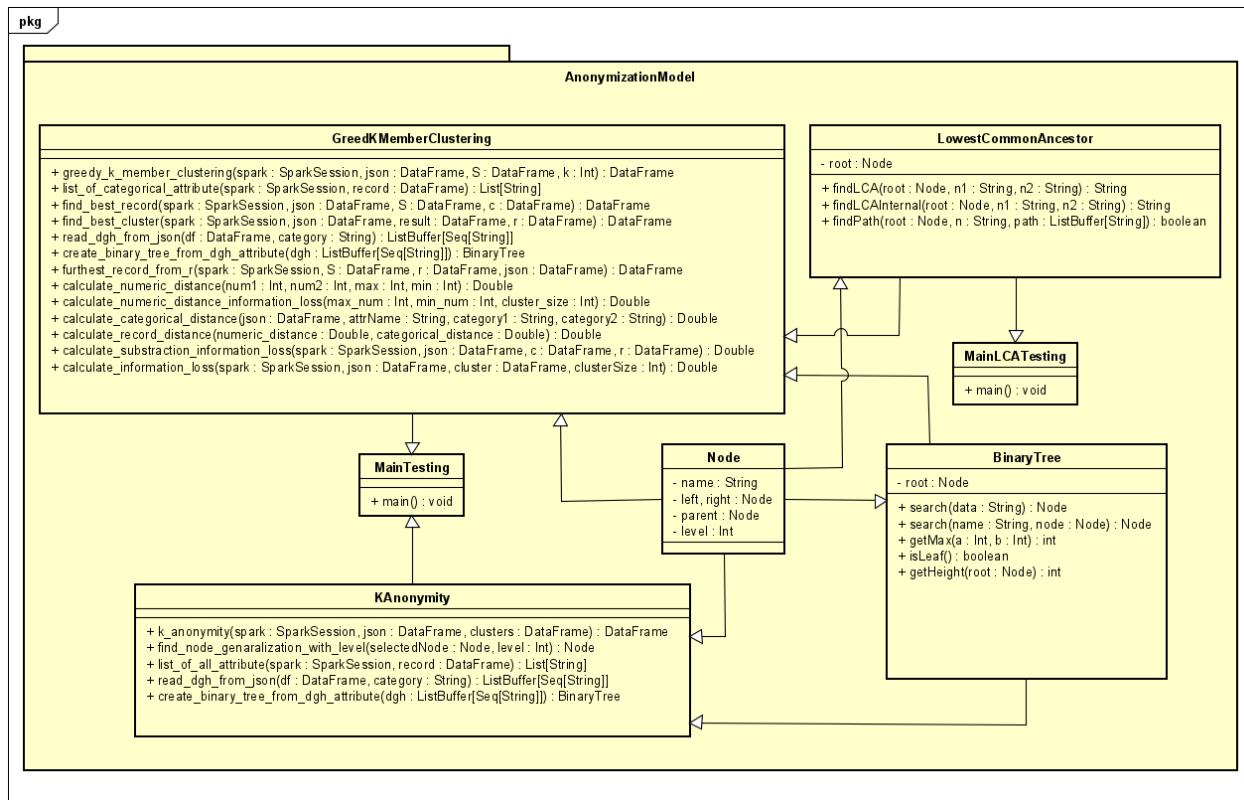
20 **Kelas *MainTesting***

21 Kelas *MainTesting* merupakan kelas dengan tipe *object*, karena kelas ini memiliki method *Main*
 22 untuk menjalankan eksekusi program. Kelas ini berperan penting untuk mencari nilai unik setiap
 23 atribut. Hasil eksekusi kelas ini akan menghasilkan output berupa beberapa jenis CSV yang

- 1 bergantung kepada jumlah atribut yang ingin diketahui nilai uniknya dan atribut tersebut telah dicantumkan pada format JSON. Perlu diketahui bahwa kelas ini tidak memiliki atribut dan hanya memiliki sebuah method dengan nama `generate_dataframe_from_csv` untuk membuat DataFrame berdasarkan atribut yang dipilih dan telah dicantumkan dalam format JSON.

5 4.3.3 Diagram Kelas pada Package AnonymizationModel

- 6 Perangkat lunak anonimisasi hanya memiliki satu jenis *package* bernama *AnonymizationModel*.
- 7 *Package AnonymizationModel* terdiri dari beberapa kelas. Masing-masing kelas memiliki fungsi penting menyelesaikan permasalahan pengelompokan dan anonimisasi data.



Gambar 4.14: Diagram Kelas pada Package AnonymizationModel

9 Kelas *Node*

- 10 Kelas *Node* merupakan kelas dengan tipe *class*, karena kelas ini berfungsi sebagai model untuk membuat atribut dan *method* pada objek *Node*. Kelas ini bertujuan untuk menyimpan informasi seperti nama *node*, menyatakan *node* kiri dan *node* kanan dari *node* tersebut, menyatakan *parent* dari *node* tersebut, dan *level* yang menyatakan posisi ketinggian *node* pada objek *BinaryTree*. Kelas *Node* nantinya akan dipakai untuk pembuatan objek *BinaryTree*.
- 15
- 16 Berikut adalah penjelasan masing-masing atribut pada kelas *Node*:
 - 17 • **name** adalah variabel yang berfungsi untuk menyimpan nilai atribut tertentu pada sebuah tabel data dengan tipe String dan termasuk jenis variabel **var** sehingga atribut ini nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.

- 1 • **left** adalah variabel yang berfungsi untuk menyimpan node kiri dari node tersebut dengan
2 tipe Node dan termasuk jenis variabel **var** sehingga atribut ini nilainya bersifat *mutable*/dapat
3 diubah sesuai kebutuhan.
- 4 • **right** adalah variabel yang berfungsi untuk menyimpan node kanan dari node tersebut dengan
5 tipe Node dan termasuk jenis variabel **var** sehingga atribut ini nilainya bersifat *mutable*/dapat
6 diubah sesuai kebutuhan.
- 7 • **parent** adalah variabel yang berfungsi untuk menyimpan node **parent** dari node tersebut
8 dengan tipe Node dan termasuk jenis variabel **var** sehingga atribut ini nilainya bersifat
9 *mutable*/dapat diubah sesuai kebutuhan.
- 10 • **level** adalah variabel yang berfungsi untuk menyimpan posisi ketinggian *BinaryTree* untuk
11 node tersebut dengan tipe Integer dan termasuk jenis variabel **var** sehingga atribut ini nilainya
12 bersifat *mutable*/dapat diubah sesuai kebutuhan.

13 Kelas *BinaryTree*

14 Kelas *BinaryTree* merupakan kelas dengan tipe *class*, karena kelas ini berfungsi sebagai model
15 untuk membuat atribut dan method pada objek *BinaryTree*. Kelas ini nantinya akan dipakai untuk
16 pembuatan objek *BinaryTree* berdasarkan *Domain Generalization Hierarchy* (DGH).

- 17
- 18 Berikut deskripsi atribut pada kelas *BinaryTree*:
- 19 • **root** adalah variabel yang berfungsi untuk menyimpan node yang menjadi root pada objek
20 *BinaryTree* dengan tipe Node dan termasuk jenis variabel *var* sehingga atribut *root* nilainya
21 bersifat *mutable*/dapat diubah sesuai kebutuhan.

22 Berikut deskripsi atribut pada kelas *BinaryTree*:

- 23
- 24 • **search** dengan parameter **data(String)** adalah fungsi yang akan memanggil fungsi *search*
25 lain dengan parameter **name(String)**, **node(Node)** untuk mengembalikan objek *Node* yang
26 ingin dicari pada objek *BinaryTree*.
- 27
- 28 • **search** dengan parameter **name(String)**, **node(Node)** adalah fungsi yang bertujuan untuk
29 mengembalikan objek *Node* yang ingin dicari pada objek *BinaryTree*.
- 30
- 31 • **getMax** dengan parameter **a(Integer)**, **b(Integer)** adalah fungsi yang bertujuan untuk
32 mengembalikan nilai *Integer* paling besar antara parameter **a** dan **b**.
- 33
- 34 • **isLeaf** tanpa parameter adalah fungsi yang bertujuan untuk mengembalikan nilai *Boolean*
35 untuk menyatakan apakah root dari node kiri dan kanan bernilai *null*. Jika kondisi terpenuhi
36 maka nilainya *true*, apabila tidak terpenuhi maka nilainya *false*.
- 37
- 38 • **getHeight** dengan parameter **root(Node)** adalah fungsi yang bertujuan untuk mengembalikan
39 nilai *Integer* untuk menyatakan ketinggian dari objek *BinaryTree*.

- 1 Berikut implementasi method `search` pada kelas *BinaryTree* di Algoritma 4:

Algoritma 4 Mencari Node dengan Nama Tertentu

```

1: Function search(name, node)
2: Input: a name of node (name) and a node.
3: Output: a node with selected name.
4:
5: if node != null then
6:   if node.name == name then
7:     return node
8:   else
9:     foundNode = search(name, node.left)
10:    if foundNode == null then
11:      foundNode = search(name, node.right)
12:    end if
13:    return foundNode
14:  end if
15: else
16:   return null
17: end if
```

- 3 • Baris 6-8: baris ini mengembalikan sebuah node, jika nama sebuah node sudah sesuai.
- 4 • Baris 9-14: baris ini melakukan proses rekursif untuk mencari nama node yang sesuai.

5 **Kelas *LowestCommonAncestor***

- 6 Kelas *LowestCommonAncestor* merupakan kelas dengan tipe *class*. Hal ini dikarenakan kelas *Lowes-*
7 *tCommonAncestor* berfungsi sebagai model untuk membuat atribut dan method-method pada objek
8 *LowestCommonAncestor*. Kelas *LowestCommonAncestor* bertujuan untuk melakukan pencarian
9 node root terdekat dari kedua node. Kelas *LowestCommonAncestor* nantinya akan dipakai untuk
10 mencari level node root terdekat untuk menghitung distance kategorikal.

- 11
- 12 Berikut deskripsi atribut pada kelas *LowestCommonAncestor*:

- 13 • `path1` adalah variabel yang berfungsi untuk menyimpan list node yang pernah dipilih sebe-
14 lumnya dengan tipe `ListBuffer[String]` dan termasuk jenis variabel `var` sehingga atribut `path1`
15 nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.
- 16 • `path2` adalah variabel yang berfungsi untuk menyimpan list node yang pernah dipilih sebe-
17 lumnya dengan tipe `ListBuffer[String]` dan termasuk jenis variabel `var` sehingga atribut `path1`
18 nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.

- 19 Berikut deskripsi method pada kelas *LowestCommonAncestor*:

- 20 • `findLCA` dengan parameter `root(Node)`, `n1(String)`, `n2(String)` adalah fungsi yang bertujuan
21 untuk mereset nilai `path1` dan `path2` dan memanggil fungsi `findLCAInternal` untuk mencari
22 nama root paling bawah yang mengandung kedua node.

- 1 • **findLCAInternal** dengan parameter *root(Node)*, *n1(String)*, *n2(String)* adalah fungsi yang
2 bertujuan untuk mencari nama root paling bawah yang mengandung kedua node. Caranya
3 dengan menambahkan indeks pada setiap iterasi, jika nama node pada path1 sama dengan
4 nama node pada path2. Indeks ini nantinya dipakai untuk mengambil nama node pada path1
5 sebagai hasil output dari algoritma Lowest Common Ancestor.
- 6 • **findPath** dengan parameter *root(Node)*, *n(String)*, *path(ListBuffer[String])* adalah fungsi
7 yang bertujuan untuk menambahkan node yang pernah diproses sebelumnya pada method ini
8 sebagai elemen untuk array **path 1** dan **path 2**. Method ini akan mengembalikan nilai true,
9 jika nama node yang diberikan sama dengan nama yang dicari.

10 Berikut implementasi **findLCAInternal** pada kelas *LowestCommonAncestor* di Algoritma 5:

Algoritma 5 Mencari Nama Root Terdekat dengan Kedua Node

```

1: Function findLCAInternal(root,namenode1,namenode2)
2: Input: root node (root), name of node 1 (n1), name of node 2 (n2).
3: Output: a name of root node.
4:
5: if (!findPath(root,namenode1,path1) or !findPath(root,namenode2,path2))
6:   then
7:     return "not found"
8:
9: end if
10:
11: while (i < path1.size and i < path2.size) do
12:   if !path1(i).equals(path2(i)) then
13:     i += 1
14:   end if
15: end while
16: return path1(i-1)
```

- 12 • Baris 5-7: baris ini memeriksa apakah nama node 1 dan nama node 2 ada pada objek Binary
13 Tree, jika tersedia maka method akan mengembalikan nilai true.
- 14 • Baris 9: baris ini digunakan untuk mengambil nama node yang pernah dikunjungi sebelumnya
15 pada indeks ke-*i* dari elemen array pada **path1** dan elemen array pada **path2**.
- 16 • Baris 11-15: baris ini melakukan perulangan untuk mencari nama node root paling rendah,
17 dengan objek node 1 dan node 2 sebagai anak dari node root tersebut.
- 18 • Baris 12-14: baris ini mencari nama root terdekat dengan menambahkan indeks pada setiap
19 iterasi, jika node 1 memiliki nama yang sama dengan node 2.
- 20 • Baris 16: pada baris ini, iterasi yang diperoleh akan dipakai untuk menyatakan nama dari
21 node root terdekat antara node 1 dan node 2.

22 Berikut implementasi method **findPath** pada kelas *LowestCommonAncestor* di Algoritma 6:

Algoritma 6 Mencari Node yang Pernah Dilalui Sebelumnya

```

1: Function findPath(root,namenode,path)
2: Input: root node (root), name of node (n), path of node.
3: Output: true/false.
4:
5: if (root == null) then
6:     return false
7: end if
8: path+ = root.name
9:
10: if (root.name == namenode) then
11:     return true
12: end if
13:
14: if (root.left != null and findPath(root.left, n, path)) then
15:     return true
16: end if
17:
18: if (root.right != null and findPath(root.right, n, path)) then
19:     return true
20: end if
21: path.remove(path.size - 1)
22: return false

```

- 2 • Baris 5-7: baris ini akan mengembalikan nilai *false* jika *node* sudah kosong.
- 3 • Baris 9: baris ini akan mencatat setiap *node* yang pernah dikunjungi ke dalam variabel *path*
- 4 • Baris 11-13: baris ini akan mengembalikan nilai *true* jika node yang dikunjungi memiliki
- 5 nama yang sama dengan nama yang ingin dicari
- 6 • Baris 15-17: baris ini akan memeriksa apakah setiap *node* kiri dari sebuah node memiliki
- 7 nama yang dicari, jika benar maka akan mengembalikan nilai *true*.
- 8 • Baris 19-21: baris ini akan memeriksa apakah setiap *node* kanan dari sebuah *node* memiliki
- 9 nama yang dicari, jika benar maka akan mengembalikan nilai *true*.
- 10 • Baris 23: setiap kali method ini dipanggil, maka *node* paling terakhir akan dihapus dari
- 11 kumpulan node yang sudah pernah dicatat sebelumnya pada variabel *path*.

12 Kelas *GreedyKMemberClustering*

- 13 Kelas *GreedyKMemberClustering* merupakan kelas dengan tipe *class*. Hal ini dikarenakan kelas
- 14 *GreedyKMemberClustering* berfungsi sebagai model untuk membuat atribut dan method pada objek
- 15 *GreedyKMemberClustering*. Kelas *GreedyKMemberClustering* bertujuan untuk melakukan fungsi
- 16 pengelompokan data. Kelas *GreedyKMemberClustering* tidak memiliki atribut.
- 17
- 18 Berikut deskripsi method pada kelas *GreedyKMemberClustering*:

- 1 • `greedy_k_member_clustering` dengan parameter `spark(SparkSession)`, `json(DataFrame)`,
2 `S(DataFrame)`, `k(Integer)` adalah fungsi yang bertujuan untuk melakukan pengelompokan
3 data dengan algoritma Greedy k-member clustering sebelum data dilakukan anonimisasi. Ke-
4 lompok data yang terbentuk akan didasari pada selisih information loss paling minimum antar
5 masing-masing data berdasarkan fungsi `find_best_record` dan fungsi `find_best_cluster`.

- 6 • `list_of_categorical_attribute` dengan parameter `spark(SparkSession, record (DataFrame))`
7 adalah fungsi yang bertujuan untuk mendapatkan seluruh nama atribut dari tabel
8 data. Fungsi lain seperti `furthest_record_from_r`, `calculate_information_loss` akan
9 memanggil fungsi ini untuk mendapatkan tabel data yang berisi kolom-kolom bertipe kate-
10 gorikal, sebelum tabel data dipakai untuk mencari record paling jauh dengan record r dan
11 sebelum tabel data dipakai untuk menghitung nilai information loss pada kelompok data.

- 12 • `find_best_record` dengan parameter `spark(SparkSession), json(DataFrame), S(Data-
13 Frame)` adalah fungsi yang bertujuan untuk mencari setiap record dari tabel
14 data yang memiliki nilai information loss paling minimum terhadap record c.

- 15 • `find_best_cluster` dengan parameter `spark(SparkSession), json(DataFrame), S (Data-
16 Frame), r(DataFrame)` adalah fungsi yang bertujuan untuk mencari setiap record dari tabel
17 data yang telah dikelompokan dengan nilai information loss paling rendah terhadap sebuah
18 record r, dimana r adalah record yang belum masuk kelompok data tertentu.

- 19 • `read_dgh_from_json` dengan parameter `df(DataFrame), category(String)` adalah fungsi
20 yang bertujuan untuk mengembalikan nilai `value, parent, level, position` untuk setiap
21 nama atribut tabel data pada `domain_generaliation_hierarchy` JSON.

- 22 • `create_binary_tree_from_dgh_attribute` dengan parameter `dgh(ListBuffer[Seq[String]]), category(String)` adalah fungsi yang bertujuan untuk membuat pohon DGH melalui
23 representasi objek `BinaryTree` pada `domain_generaliation_hierarchy` JSON.

- 24 • `furthest_record_from_r` dengan parameter `spark(SparkSession), S(DataFrame), r (Data-
25 Frame), json(DataFrame)` adalah fungsi yang bertujuan untuk menghitung distance record
26 antara record r dengan masing-masing record pada tabel data. Setelah dilakukan perhitungan,
27 record dengan nilai distance record tertinggi akan dikembalikan sebagai output dari fungsi ini.

- 28 • `calculate_numeric_distance` dengan parameter `num1(Int), num2(Int), max(Int), min(Int)`
29 adalah fungsi yang bertujuan untuk menghitung distance numerik antara 2 data numerik de-
30 ngan atribut yang sejenis. Fungsi ini dipanggil oleh fungsi lain yaitu `furthest_record_from_r`,
31 agar hasil perhitungan distance numerik dapat digunakan untuk menghitung distance record.

- 32 • `calculate_numeric_distance_information_loss` dengan parameter `max_num(Integer),
33 min_num(Integer), cluster_size(Integer)` adalah fungsi yang bertujuan untuk meng-
34 hitung distance numerik pada kasus information loss antar 2 data numerik di atribut yang
35 sejenis. Fungsi ini dipanggil oleh fungsi lain yaitu `calculate_information_loss`, agar hasil
36 perhitungan distance numerik dapat digunakan untuk menghitung nilai information loss pada
37 sebuah kelompok data untuk mendapatkan hasil pengelompokan data yang terbaik.

- 1 • `calculate_categorical_distance` dengan parameter `json(DataFrame)`, `attrName(String)`,
2 `category1(String)`, `category2(String)` adalah fungsi yang bertujuan untuk menghitung
3 distance kategorikal antara 2 data kategori dengan atribut yang sejenis. Fungsi ini dipanggil
4 oleh fungsi `furthest_record_from_r` untuk menghitung distance record.
- 5 • `calculate_record_distance` dengan parameter `numeric_distance(Double)`, `categorical_`
6 `distance (Double)` adalah fungsi yang bertujuan untuk menghitung distance record antara
7 2 record. Fungsi ini dipanggil oleh fungsi `furthest_record_from_r` untuk mencari record
8 dengan distance record paling kecil. Distance record didapat dengan menjumlahkan total
9 distance numerik dan total distance kategorikal.
- 10 • `calculate_subtraction_information_loss` dengan parameter `spark (SparkSession)`,
11 `json(DataFrame)`, `c(DataFrame)`, `r(DataFrame)` adalah fungsi yang bertujuan untuk mencari
12 selisih information loss antara dua kelompok tabel data yang berbeda. Fungsi ini dipanggil
13 oleh fungsi lain seperti `find_best_record` dan `find_best_cluster` untuk mencari record
14 terbaik berdasarkan selisih information loss paling kecil.
- 15 • `calculate_information_loss` dengan parameter `spark(SparkSession)`, `json(DataFrame)`,
16 `cluster(DataFrame)`, `clusterSize(Integer)` adalah fungsi yang bertujuan untuk menghi-
17 ting nilai information loss pada kelompok tabel data tertentu. Fungsi ini dipanggil oleh fungsi
18 lain seperti `calculate_subtraction_information_loss` untuk mencari selisih information
19 loss antara dua kelompok tabel data yang berbeda.

20 Berikut implementasi method `calculate_categorical_distance` di Algoritma 7:

Algoritma 7 Menghitung Distance Kategorikal

```

1: Function calculate_categorical_distance(json, attrName, cat1, cat2)
2: Input: numeric 1, numeric 2, maximum number, maximum number.
3: Output: numeric distance.
4:
5: result = 0.0
6: dgh = read_dgh_from_json(json, attrName)
7: if (dgh == null) then
8:     return result
9: end if
10: binaryTree = create_binary_tree_from_dgh_attribute(dgh)
11: node1 = binaryTree.search(category1)
12: node2 = binaryTree.search(category2)
13: if (node1 != null and node2 != null) then
14:     LCA = new LowestCommonAncestor()
15:     resultLCA = LCA.findLCA(binaryTree.root, node1.name, node2.name)
16:     H_subtree = binaryTree.search(resultLCA).level
17:     H_TD = binaryTree.getHeight(binaryTree.root).toDouble
18:     result = H_subtree / H_TD
19: end if
20: result = BigDecimal(result)
21: result = result.setScale(2, BigDecimal.RoundingMode.HALF_UP)
22: result = result.toDouble
23: return result

```

- 1 • Baris 6: baris ini mendapatkan nilai `value, parent, level, position` untuk setiap nama
- 2 atribut pada `domain_generalization_hierarchy` JSON.
- 3 • Baris 8-10: baris ini memeriksa jika nilai pada `domain_generalization_hierarch` JSON
- 4 kosong, maka perhitungan distance kategorikal tidak perlu dilakukan.
- 5 • Baris 12: baris ini membuat objek `BinaryTree` berdasarkan atribut `dgh` dari baris 6.
- 6 • Baris 16-22: baris ini menghitung *distance* kategorikal jika kedua node tidak kosong.
- 7 • Baris 19: baris ini menyimpan ketinggian `node root` dari objek `LowestCommonAncestor`
- 8 • Baris 24-26: baris ini mengembalikan hasil perhitungan *distance*kategorikal dalam format
- 9 desimal, dengan ketelitian 2 angka dibelakang koma.

10 Berikut implementasi method `calculate_numeric_distance` di Algoritma 8:

Algoritma 8 Menghitung Distance Numerik

```

1: Function calculate_numeric_distance(num1, num2, max, min)
2: Input: numeric 1, numeric 2, maximum number, maximum number.
3: Output: numeric distance.
4:
5: v1 = num1
6: v2 = num1
7: diff = Math.abs(v1-v2)
8: range = Math.abs(max-min)
9: result = diff/range
10: return result

```

- 12 • Baris 7: baris ini menghitung selisih perbedaan nilai antara `v1` dan `v2`.
- 13 • Baris 8: baris ini menghitung selisih range nilai antara `v1` dan `v2`.
- 14 • Baris 9: baris ini bagi selisih perbedaan nilai dengan selisih *range* nilai.

15 Berikut implementasi method `calculate_record_distance` di Algoritma 9:

Algoritma 9 Menghitung Distance Record

```

1: Function calculate_record_distance(num_dist, cat_dist)
2: Input: numeric 1, numeric 2, maximum number, maximum number.
3: Output: numeric distance.
4:
5: result = numeric_distance+categorical_distance
6: return result

```

- 17 • Baris 5: baris ini menjumlahkan total *distance* numerik masing-masing atribut dan total
- 18 *distance* kategorikal masing-masing atribut.
- 19 • Baris 6: baris ini mengembalikan hasil penjumlahannya sebagai *distance record*.

- 1 Berikut implementasi method `furthest_record_from_r` di Algoritma 10:

Algoritma 10 Mencari Record Paling Jauh dari Record Tertentu

```

1: Function furthest_record_from_r(spark,S,r,json)
2: Input: sparkSession(spark), all records(S), a record(r), json.
3: Output: furthest record from record r.
4:
5: list_record_distance = ()
6: domain = S.cache()
7: r_values = r.first().toSeq
8: categorical_name = list_of_categorical_attribute(spark,r)
9: index = 0
10: while (!domain.isEmpty) do
11:     selected_record = domain.limit(1).cache()
12:     record_values = selected_record.first().toSeq
13:     column_size = record_values.length-1
14:     record_distance = 0
15:     record_id = selected_record.select("id").first().getInt(0)
16:     for (i = 0 to column_size) do
17:         if (record_values(i).isInstanceOf[Int]) then
18:             num1 = record_values(i).toString.toInt
19:             num2 = r_values(i).toString.toInt
20:             max = domain.groupBy().max(domain.columns(i)).first().getInt(0)
21:             min = domain.groupBy().min(domain.columns(i)).first().getInt(0)
22:             record_distance += calculate_numeric_distance(num1,num2,max,min)
23:         else
24:             cat1 = record_values(i).toString
25:             cat2 = r_values(i).toString
26:             attrName = categorical_name(index)
27:             record_distance += calculate_categorical_distance(json,attrName,cat1,cat2)
28:             index += 1
29:         end if
30:     end for
31:     list_record_distance +=((record_id,record_distance))
32:     domain.unpersist()
33:     domain = domain.except(selected_record)
34:     index = 0
35: end while
36: return false

```

- 3 • Baris 8: baris ini mendapatkan nilai masing-masing kolom untuk *record r* yang dipilih.
- 4 • Baris 12-38: baris ini melakukan perulangan setiap baris data sampai domain kosong.
- 5 • Baris 19-33: baris ini melakukan perulangan setiap kolom dengan menghitung distance *record*.
- 6 • Baris 20-25: baris ini melakukan perhitungan distance numerik, lalu hasil perhitungannya akan ditambahkan pada nilai atribut `record_distance` sebelumnya.
- 7 • Baris 26-32: baris ini melakukan perhitungan *distance* kategorikal, lalu hasil perhitungannya akan ditambahkan pada nilai atribut `record_distance` sebelumnya.

- 1 Berikut implementasi method `calculate_information_loss` di Algoritma 11:

Algoritma 11 Menghitung Information Loss

```

1: Function calculate_information_loss(spark,json,cluster,clusterSize)
2: Input: numeric 1, numeric 2, maximum number, maximum number.
3: Output: numeric distance.
4:
5: cluster_temp = cluster.cache()
6: informationLoss = 0
7: categorical_name = list_of_categorical_attribute(spark,cluster_temp)
8: index = 0
9:
10: record_values = cluster_temp.first().toSeq
11: column_size = record_values.length-1
12: for (i <- 0 to column_size) do
13:     if (record_values(i).isInstanceOf[Int]) then
14:         max = cluster_temp.groupBy().max(cluster_temp.columns(i)).first().getInt(0)
15:         min = cluster_temp.groupBy().min(cluster_temp.columns(i)).first().getInt(0)
16:         informationLoss += calculate_numeric_distance_IL(max,min,clusterSize)
17:     else
2
18:         attrName = categorical_name(index)
19:         distinctValues = cluster_temp.select(attrName).distinct().cache()
20:         countDistinctValues = distinctValues.count()
21:         index += 1
22:         if (countDistinctValues == 2) then
23:             record1 = distinctValues.limit(1).cache()
24:             record2 = distinctValues.except(record1)
25:             cat1 = record1.first().getString(0)
26:             cat2 = record2.first().getString(0)
27:             informationLoss += calculate_categorical_distance(json,attrName,cat1,cat2)
28:             distinctValues.unpersist()
29:             record1.unpersist()
30:         else
31:             informationLoss += 0
32:         end if
33:     end if
34: end for
35: cluster_temp.unpersist()
36: return result

```

- 3
- Baris 7: baris ini mendapatkan seluruh nama atribut dari tabel data.
- 4
- Baris 10: baris ini mendapatkan nilai dari masing-masing kolom tabel data.
- 5
- Baris 12-34: baris ini melakukan perulangan untuk setiap kolom tabel data.
- 6
- Baris 13-16: baris ini menghitung *distance* numerik apabila kolom termasuk nilai numerik.
- 7
- Baris 17-33: baris ini menghitung *distance* kategorikal apabila kolom termasuk nilai kategori.
- 8
- Baris 22-29: baris ini memeriksa jika nilai unik sebuah kolom kategorikal terdiri dari 2 nilai unik, maka *distance* kategorikal dihitung menggunakan fungsi *Lowest Common Ancestor*.

1 **Kelas *KAnonymity***

2 Kelas *KAnonymity* merupakan kelas dengan tipe *class*. Hal ini dikarenakan kelas *GreedyKMember-*
3 *Clustering* berfungsi sebagai model untuk membuat atribut dan method pada objek *KAnonymity*.
4 Kelas *KAnonymity* bertujuan untuk melakukan anonimasasi k-anonymity pada data-data yang
5 sudah dikelompokan. Kelas *KAnonymity* tidak memiliki atribut.

6

7 Berikut deskripsi method pada kelas *KAnonymity* di Algoritma 12:

- 8 • **k_anonymity** dengan parameter *spark(SparkSession)*, *json(DataFrame)*, *clusters(DataFrame)*
9 adalah fungsi yang bertujuan untuk melakukan anonimisasi data dengan algoritma k-anonymity
10 setelah tabel data berhasil dilakukan pengelompokan data.
- 11 • **find_node_genaralization_with_level** dengan parameter *selectedNode(Node)*, *level(Int)*
12 adalah fungsi yang bertujuan untuk mencari node dengan tingkatan generalisasi tertentu
13 pada objek *BinaryTree*. Tingkatan generalisasi ditentukan dari ketinggian node pada objek
14 *BinaryTree*. Semakin kecil ketinggian node, maka nilainya semakin umum.
- 15 • **list_of_all_attribute** dengan parameter *spark(SparkSession)*, *record(DataFrame)* ada-
16 lah fungsi yang bertujuan untuk mendapatkan seluruh nama kolom pada sebuah tabel data.
17 Fungsi ini akan dipanggil pada fungsi lain, yaitu **k_anonymity** untuk mengganti nilai sesung-
18 guhnya dengan nilai yang lebih umum pada proses anonimisasi.
- 19 • **read_dgh_from_json** dengan parameter *df(DataFrame)*, *category(String)* adalah fungsi
20 yang bertujuan untuk mengembalikan nilai **value**, **parent**, **level**, **position** untuk beberapa
21 nama kolom pada domain **_generaliation_hierarchy** JSON.
- 22 • **create_binary_tree_from_dgh_attribute** dengan parameter *dgh(ListBuffer[Seq[String]]))*
23 adalah fungsi yang bertujuan mengambil domain **_generaliation_hierarchy** pada JSON un-
24 tuk membuat objek *BinaryTree*. Fungsi ini akan dipanggil pada fungsi lain, yaitu **k_anonymity**
25 untuk mengambil nilai yang lebih umum pada pohon DGH.

26 Berikut implementasi method **k_anonymity**:

- 27 • Baris 7: baris ini mendapatkan seluruh nama atribut dari tabel data.
- 28 • Baris 9-48: baris ini melakukan perulangan selama **cluster_temp** tidak kosong.
- 29 • Baris 11-12: baris ini mendapatkan data berdasarkan nama cluster tertentu.
- 30 • Baris 16-40: baris ini melakukan perulangan untuk untuk setiap kolom tabel cluster tertentu.
- 31 • Baris 17: baris ini mendapatkan nilai unik dari sebuah kolom data pada nama cluster tertentu.
- 32 • Baris 21-25: melakukan anonimisasi kolom numerik dengan rentang nilai kolom pada cluster.
- 33 • Baris 26-38: melakukan anonimisasi kolom kategorikal dengan nama root pada pohon DGH.

- 1 • Baris 28-31: baris ini memeriksa jika nilai unik kategorikal pada sebuah kolom melebihi 1
2 nilai unik, maka hasil anonimisasinya adalah nama root dari pohon DGH.
- 3 • Baris 32-37: baris ini memeriksa jika nilai unik kategorikal pada sebuah kolom hanya 1 nilai
4 unik, maka hasil anonimisasinya adalah nilai kategorikal tersebut.
- 5 • Baris 41-42: baris ini menginisialisasi baris data hasil anonimisasi untuk pertama kali.
- 6 • Baris 43-44: baris ini menggabungkan hasil anonimisasi pada baris-baris data sebelumnya
7 dengan hasil anonimisasi pada baris data yang baru.

8 Kelas *MainTesting*

9 Kelas *MainTesting* merupakan kelas dengan tipe *object*. Hal ini dikarenakan kelas *MainTesting* ber-
10 peran penting untuk melakukan eksekusi pengelompokan data dengan algoritma Greedy k-member
11 clustering dan anonimisasi data dengan algoritma k-anonymity.

12
13 Berikut adalah penjelasan masing-masing method pada kelas *MainTesting*:

- 14 • **main** adalah fungsi yang bertujuan untuk melakukan eksekusi perangkat lunak anonimisasi
15 dengan objek GreedyKMemberClustering dan objek KAnonymity.
- 16 • **generate_dataframe_from_csv** dengan parameter **spark(SparkSession)**, **json(DataFrame)**,
17 **dataInput(DataFrame)** adalah fungsi yang bertujuan untuk mengambil data **quasi_identifier**
18 dan **sensitive_attribute** pada JSON. Output dari fungsi ini adalah array 2 dimensi yang
19 berisi nama atribut dan tipe atribut (*category/numeric*).
- 20 • **get_attribute_name_json** dengan parameter **values(ListBuffer[Seq[String]])** adalah
21 fungsi yang bertujuan untuk mendapatkan nama-nama atribut berdasarkan output array 2
22 dimensi dari fungsi **generate_dataframe_from_csv**.
- 23 • **get_attribute_datatype_json** dengan parameter **values(ListBuffer[Seq[String]])** ada-
24 lah fungsi yang bertujuan untuk mengubah atribut dengan tipe "category" menjadi String,
25 sedangkan atribut dengan tipe "numeric" menjadi Integer.
- 26 • **read_element_from_json** dengan parameter **json(DataFrame)**, **elemen(String)** adalah
27 fungsi yang bertujuan untuk mendapatkan nilai atribut tertentu pada JSON. Fungsi ini
28 dipanggil pada fungsi lain yaitu **generate_dataframe_from_csv** untuk mengambil nilai dari
29 atribut **quasi_identifier** dan **sensitive_identifier**.

Algoritma 12 Melakukan Anonimisasi Data dengan K-Anonymity

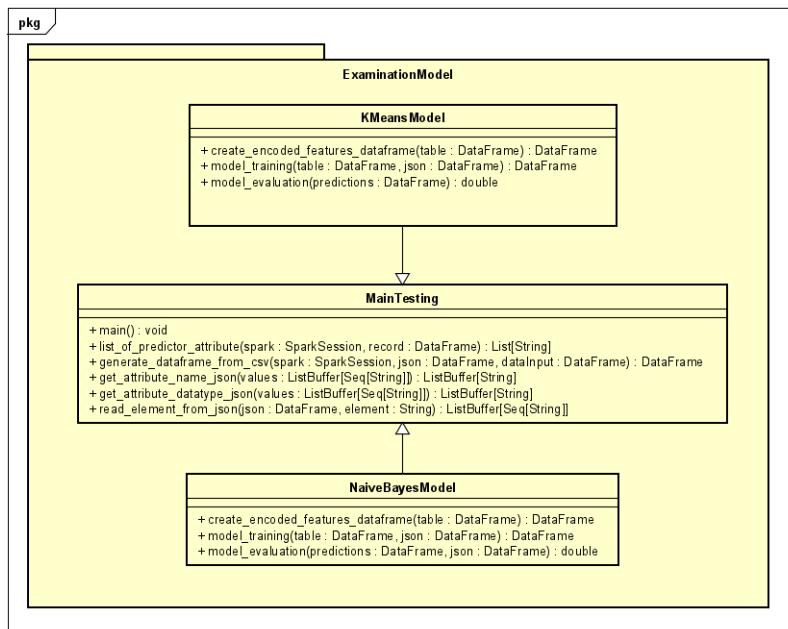
```

1: Function k_anonymity)
2: Input: spark, json, clusters.
3: Output: table with anonymization data.
4:
5: result = spark.emptyDataFrame
6: clusters_temp = clusters
7: columnName = list_of_all_attribute(spark,clusters_temp)
8: column_size = columnName.length-1
9: while (!clusters_temp.isEmpty) do
10:   clusterName = clusters_temp.select("Cluster").first().getString(0)
11:   clusterDF = clusters_temp.where(clusters_temp("Cluster"))
12:   clusterDF = clusterDF.contains(clusterName).cache()
13:   clusterAnonymization = clusterDF.select("id")
14:   recordDistinctValues = spark.emptyDataFrame
15:   numDistValues = 0
16:   for (i <- 0 to column_size) do
17:     recordDistinctValues = clusterDF.select(columnName(i)).distinct().cache()
18:     numDistinctValues = recordDistinctValues.count().toInt
19:     colName = columnName(i)
20:     colValues = lit(generalizationNumeric))
21:     if (numDistValues > 1 and recordDistinctValues.isInstanceOf[Int] then
22:       maxValue = recordDistinctValues.groupBy().max()
23:       minValue = recordDistinctValues.groupBy().min()
24:       generalizationNumeric = "["+minValue+"-"+maxValue+"]"
25:       clusterAnonym = clusterAnonym.withColumn(columnName, columnValues
1
26:     else
27:       dgh = read_dgh_from_json(json, columnName(i))
28:       if (numDistValues > 1 and recordDistinctValues.isInstanceOf[String] then
29:         binaryTree = create_binary_tree_from_dgh_attribute(dgh)
30:         generalizationCategorical = binaryTree.root.name
31:         clusterAnonym = clusterAnonym.withColumn(colName, colValues
32:       else
33:         column = clusterDF.select("id", columnName(i))
34:         column = column.withColumnRenamed("id", "id_temp")
35:         clusterAnonym = clusterAnonym.join(column)
36:         clusterAnonym = clusterAnonym.drop("id_temp")
37:       end if
38:     end if
39:     recordDistinctValues.unpersist()
40:   end for
41:   if (result.isEmpty) then
42:     result = clusterAnonym
43:   else
44:     result = result.union(clusterAnonym)
45:   end if
46:   clusters_temp.unpersist()
47:   clusters_temp = clusters_temp.except(clusterDF)
48: end while
49: result = result.drop("Cluster")
50: result = result.orderBy(asc("id"))
51: return result

```

1 4.3.4 Diagram Kelas pada Package ExaminationModel

- 2 Perangkat lunak anonimisasi hanya memiliki satu jenis *package* dengan nama *model_anonimisasi*.
- 3 *Package model_anonimisasi* terdiri dari beberapa kelas. Masing-masing kelas memiliki fungsi penting untuk menyelesaikan permasalahan pengelompokan dan anonimisasi data.



Gambar 4.15: Diagram Kelas pada ExaminationModel

5 Kelas *KMeansModel*

- 6 Kelas *KMeansModel* merupakan kelas dengan tipe *class*. Hal ini dikarenakan kelas *KMeansModel* berfungsi sebagai model untuk membuat atribut dan method pada objek *KMeansModel*. Kelas *KMeansModel* bertujuan membuat pemodelan *k-means* dan menghitung *silhouette score*.
- 9
- 10 Berikut deskripsi method pada kelas *KMeansModel*:

- 11 • `create_encoded_features_dataframe` dengan parameter `table(DataFrame)` adalah fungsi yang bertujuan untuk membuat tabel data baru yang berisi nilai aktual, label index, vektor masing-masing atribut, dan vektor dari masing-masing label index.
- 12 • `model_training` dengan parameter `table(DataFrame), json(DataFrame)` adalah fungsi yang bertujuan untuk membuat model pelatihan k-means dan menghasilkan tabel hasil pengelompokan data berdasarkan input vektor fitur pada setiap baris data.
- 13 • `model_evaluation` dengan parameter `predictions(DataFrame)` adalah fungsi yang bertujuan untuk mencari tahu seberapa baik model k-means yang dibuat dengan menghitung silhouette score. Jika silhouette score mendekati nilai 1, maka hasil pengelompokan sudah baik.

- 20 Berikut implementasi method `model_training` di Algoritma 13:

Algoritma 13 Membuat Pemodelan K-Means

```

1: Function model_training(table,json)
2: Input: table data, JSON
3: Output: table of cluster data.
4:
5: k = json.select("k_means.k").first().getLong(0).toInt
6: kmeans = new KMeans().setK(k).setFeaturesCol("features").setPredictionCol("prediction")
7: model = kmeans.fit(table)
8: predictions = model.transform(table)
9: return predictions

```

- 2 • Baris 5: baris ini mendapatkan nilai **k** dari atribut **k_means** JSON.
- 3 • Baris 6: baris ini membuat model KMeans menggunakan parameter **k** pada baris sebelumnya.
- 4 • Baris 7: baris ini melakukan pelatihan model k-means dengan parameter tabel data.
- 5 • Baris 8: baris ini melakukan prediksi model k-means terhadap parameter tabel data.
- 6 Berikut implementasi method **model_evaluation** di Algoritma 14:

Algoritma 14 Menghitung Silhouette Score

```

1: Function calculate_numeric_distance(num1,num2,max,min)
2: Input: numeric 1, numeric 2, maximum number, maximum number.
7: Output: numeric distance.
4:
5: evaluator = new ClusteringEvaluator()
6: silhouette_score = evaluator.evaluate(predictions)
7: return silhouette_score

```

- 8 • Baris 5: baris ini membuat model ClusteringEvaluator untuk evaluasi k-means.
- 9 • Baris 6: baris ini menghitung nilai *silhouette score* untuk pemodelan k-means.

10 **Kelas NaiveBayesModel**

11 Kelas *NaiveBayesModel* merupakan kelas dengan tipe *class*, karena kelas *NaiveBayesModel* hanya
12 berfungsi sebagai model untuk membuat atribut dan fungsi pada objek *NaiveBayesModel*. Kelas ini
13 bertujuan untuk melakukan pemodelan *naive bayes* dan menghitung *accuracy*.

14

15 Berikut deskripsi method pada kelas *NaiveBayesModel* di Algoritma 15:

- 16 • **create_encoded_features_dataframe** dengan parameter **table(DataFrame)** adalah fungsi
17 yang bertujuan untuk membuat tabel data baru yang berisi nilai aktual, label index, vektor
18 untuk masing-masing atribut, dan ditambah satu kolom baru untuk menyimpan vektor fitur
19 dari label index. Output fungsi ini menjadi input pada fungsi **model_training**
- 20 • **model_training** dengan parameter **table(DataFrame), json(DataFrame)** adalah fungsi yang
21 bertujuan untuk membuat model pelatihan naive bayes dan menghasilkan tabel klasifikasi
22 data berdasarkan input vektor fitur untuk setiap baris data.

- 1 • `model_evaluation` dengan parameter `predictions(DataFrame)`, `json(DataFrame)` adalah
2 fungsi yang bertujuan untuk mencari tahu seberapa baik model naive bayes yang dibuat
3 dengan menghitung accuracy score. Apabila nilainya mendekati 1, maka klasifikasi data yang
4 dibuat dengan model naive bayes sudah mendekati benar.

5 Berikut implementasi method `model_training`:

Algoritma 15 Membuat Pemodelan Naive Bayes

```

1: Function model_training(table,json)
2: Input: table data, JSON
3: Output: table of classification data.
4:
5: attrName = json.select("naive_bayes.label").first().getString(0)
6: trainingSet = json.select("naive_bayes.training_set").first().getDouble(0)
7: testSet = json.select("naive_bayes.test_set").first().getDouble(0)
8: Array(training, test) = table.randomSplit(Array(trainingSet, testSet))
9: model = new NaiveBayes().setModelType("multinomial")
10: model = model.setLabelCol(attrName+"_Index").fit(training)
11: predictions = model.transform(test)
12: return predictions

```

- 7 • Baris 5: baris ini mendapatkan nilai `label` dari atribut `naive_bayes` JSON.
- 8 • Baris 7: baris ini mendapatkan persentase `training_set` dari atribut `naive_bayes` JSON.
- 9 • Baris 8: baris ini mendapatkan persentase `test_set` dari atribut `naive_bayes` JSON.
- 10 • Baris 10: baris ini membagi data menjadi data training dan data test berdasarkan persentase
11 dari atribut `trainingSet` dan `testSet` pada baris sebelumnya.
- 12 • Baris 12: baris ini digunakan untuk membuat model NaiveBayes bertipe "multinomial".
- 13 • Baris 13: baris ini melakukan pelatihan model naive bayes dengan data training.
- 14 • Baris 15: baris ini melakukan prediksi model naive bayes dengan data test.

15 Berikut implementasi method `model_evaluation` di Algoritma 16:

Algoritma 16 Menghitung Silhouette Score

```

1: Function calculate_numeric_distance(num1,num2,max,min)
2: Input: numeric 1, numeric 2, maximum number, maximum number.
3: Output: numeric distance.
4:
5: attrName = json.select("naive_bayes.label").first().getString(0)
6: evaluator = new MulticlassClassificationEvaluator()
7: evaluator = evaluator.setLabelCol(attrName+"_Index")
8: evaluator = evaluator.setPredictionCol("prediction")
9: evaluator = evaluator.setMetricName("accuracy")
10: accuracy = evaluator.evaluate(predictions)
11: return accuracy

```

- 1 • Baris 5: baris ini mendapatkan nilai `label` dari atribut `naive_bayes` JSON.
- 2 • Baris 6-9: baris ini membuat model evaluasi `MulticlassClassificationEvaluator` untuk
- 3 mengetahui nilai akurasi dari hasil klasifikasi pemodelan naive bayes.
- 4 • Baris 10: baris ini mengembalikan hasil akurasi untuk pemodelan naive bayes.

5 **Kelas *MainTesting***

6 Kelas *MainTesting* merupakan kelas dengan tipe *object*. Hal ini dikarenakan kelas *MainTesting*
7 berperan penting untuk melakukan eksekusi perangkat lunak pengujian terhadap masalah penge-
8 lompokan data dengan k-means dan masalah klasifikasi data dengan naive bayes. Kelas ini juga
9 menangani eksekusi evaluasi kedua model tersebut.

10

11 Berikut deskripsi method pada kelas *MainTesting*:

- 12 • `main` adalah fungsi yang bertujuan untuk melakukan proses eksekusi perangkat lunak pengujian
13 dengan membuat objek `NaiveBayesModel` dan objek `KMeansModel`.
- 14 • `list_of_predictor_attribute` dengan parameter `spark(SparkSession)`, `record(DataFrame)`
15 adalah fungsi yang bertujuan untuk mendapatkan nilai kolom-kolom prediktor dari tabel data.
- 16 • `generate_dataframe_from_csv` dengan parameter `spark(SparkSession)`, `json(DataFrame)`,
17 `dataInput(DataFrame)` adalah fungsi yang bertujuan untuk mengambil data `quasi_identifier`
18 dan `sensitive_attribute` pada JSON. Output dari fungsi ini adalah array 2 dimensi berisi
19 informasi mengenai nama atribut dan kategori atribut (*category/numeric*).
- 20 • `get_attribute_name_json` dengan parameter `values(ListBuffer[Seq[String]])` adalah
21 fungsi yang bertujuan untuk mendapatkan nama-nama atribut berdasarkan output array 2
22 dimensi dari fungsi `generate_dataframe_from_csv`.
- 23 • `get_attribute_datatype_json` dengan parameter `values(ListBuffer[Seq[String]])` ada-
24 lah fungsi yang bertujuan untuk mengubah atribut dengan tipe "category" menjadi String,
25 sedangkan atribut dengan tipe "numeric" menjadi Integer.
- 26 • `read_element_from_json` dengan parameter `json(DataFrame)`, `element(String)` adalah
27 fungsi yang bertujuan untuk mendapatkan nilai atribut tertentu pada JSON. Fungsi ini
28 dipanggil pada fungsi lain yaitu `generate_dataframe_from_csv` untuk mengambil nilai dari
29 atribut `quasi_identifier` dan `sensitive_identifier`.

4.4 Masukan Perangkat Lunak

Perangkat lunak membutuhkan masukan berupa data input .csv yang berisi tabel privat beserta nama atributnya. Pada file .csv, baris pertama merupakan nama atribut dan baris berikutnya merupakan data. Setiap atribut dipisahkan dengan tanda koma (","), sedangkan data baru dipisahkan dengan baris. Format file .csv dapat dilihat pada Listing 5.4.

Listing 4.1: Dataset Adult

```

6 ID, CODE_GENDER, AMT_INCOME_TOTAL, NAME_EDUCATION_TYPE, DAYS_BIRTH, DAYS_EMPLOYED,
7 OCCUPATION_TYPE
8 5008804,M,427500.0,Higher education;-12005;-4542,
9 5008805,M,427500.0,Higher education;-12005;-4542,
10 5008806,M,112500.0,Secondary / secondary special;-21474;-1134,Security staff
11 5008808,F,270000.0,Secondary / secondary special;-19110;-3051,Sales staff
12 5008809,F,270000.0,Secondary / secondary special;-19110;-3051,Sales staff
13 5008810,F,270000.0,Secondary / secondary special;-19110;-3051,Sales staff
14 5008811,F,270000.0,Secondary / secondary special;-19110;-3051,Sales staff
15 5008812,F,283500.0,Higher education;-22464,365243,
16 5008813,F,283500.0,Higher education;-22464,365243,
17 5008814,F,283500.0,Higher education;-22464,365243,
18 5008815,M,270000.0,Higher education;-16872;-769,Accountants
19 5112956,M,270000.0,Higher education;-16872;-769,Accountants
20 6153651,M,270000.0,Higher education;-16872;-769,Accountants
21 5008819,M,135000.0,Secondary / secondary special;-17778;-1194,Laborers
22 5008820,M,135000.0,Secondary / secondary special;-17778;-1194,Laborers
23 5008821,M,135000.0,Secondary / secondary special;-17778;-1194,Laborers
24 5008822,M,135000.0,Secondary / secondary special;-17778;-1194,Laborers
25 5008823,M,135000.0,Secondary / secondary special;-17778;-1194,Laborers
26 5008824,M,135000.0,Secondary / secondary special;-17778;-1194,Laborers
27 5008825,F,130500.0,Incomplete higher;-10669;-1103,Accountants
28 5008826,F,130500.0,Incomplete higher;-10669;-1103,Accountants
29 5008830,F,157500.0,Secondary / secondary special;-10031;-1469,Laborers
30 5008831,F,157500.0,Secondary / secondary special;-10031;-1469,Laborers
31 5008832,F,157500.0,Secondary / secondary special;-10031;-1469,Laborers
32 5008834,F,112500.0,Secondary / secondary special;-10968;-1620,
33 5008835,F,112500.0,Secondary / secondary special;-10968;-1620,
34 6153712,F,112500.0,Secondary / secondary special;-10968;-1620,
35 5008836,M,270000.0,Secondary / secondary special;-12689;-1163,Laborers
36 5008837,M,270000.0,Secondary / secondary special;-12689;-1163,Laborers
37
38

```

4.4.1 Masukan Perangkat Lunak Eksplorasi

Perangkat lunak eksplorasi membutuhkan data masukan tambahan yaitu file .json yang berisi pohon klasifikasi serta tipe dan jenis atribut. File .json menerapkan format penyimpanan data dengan diawali oleh tanda ("{ }") dan diisi oleh nilai *key,value* yang dipisahkan oleh tanda titik dua (:). Format file .json dapat dilihat pada Listing 4.2.

6

Berikut spesifikasi input .json untuk perangkat lunak eksplorasi:

- **input_path** adalah lokasi penyimpanan data input perangkat lunak. Atribut ini akan memberi tahu Spark untuk mengambil data input pada lokasi ini.
- **output_path** adalah lokasi penyimpanan hasil output perangkat lunak. Atribut ini akan memberi tahu Spark untuk menyimpan hasil output pada lokasi ini.
- **selected_column** adalah array yang menyimpan informasi pemilihan atribut. Atribut ini menyimpan 2 jenis informasi penting dalam pemilihan atribut, yaitu
 - **attrName** adalah nama atribut yang ingin dipilih pada tabel data. Atribut ini akan memberi tahu Spark untuk mengambil atribut berdasarkan nama atribut.
 - **dataType** adalah jenis tipe data atribut yang telah dipilih. Atribut ini akan memberi tahu Spark untuk menyimpan atribut yang dipilih dalam format tertentu.

Listing 4.2: Input JSON untuk Eksplorasi Data

```

18 {
19
20   "input_path": "<path data input>",
21   "output_path": "<path data output>",
22   "selected_column": [
23     {
24       "attrName": "<nama atribut>",
25       "dataType": "<tipe data atribut>"
26     },
27     {
28       "attrName": "<nama atribut>",
29       "dataType": "<tipe data atribut>"
30     }
31   ]
32 }
```

4.4.2 Masukan Perangkat Lunak Anonimisasi

Perangkat lunak anonimisasi membutuhkan data masukan tambahan yaitu file .json yang berisi pohon klasifikasi serta tipe dan jenis atribut. File .json menerapkan format penyimpanan data dengan diawali oleh tanda ("{ }") dan diisi oleh nilai *key,value* yang dipisahkan oleh tanda titik dua (:).

1 (""). Format file .json dapat dilihat pada Listing 4.3.

2

3 Berikut spesifikasi input .json untuk perangkat lunak anonimisasi:

4 • **k** adalah nilai konstanta k-anonymity dan greedy k-member clustering. Atribut ini memiliki
5 tipe data Integer sehingga nilainya harus bilangan bulat.

6 • **num_sample_datas** adalah jumlah sample data. Atribut ini memiliki tipe data Integer sehingga
7 nilainya harus dinyatakan dalam bilangan bulat.

8 • **input_path** adalah lokasi penyimpanan data input perangkat lunak. Atribut ini akan memberi
9 tahu Spark untuk mengambil data input pada lokasi ini.

10 • **output_path** adalah lokasi penyimpanan hasil output perangkat lunak. Atribut ini akan
11 memberi tahu Spark untuk menyimpan hasil output pada lokasi ini.

12 • **identifiers** adalah array yang berisi nama-nama atribut yang dapat mengungkapkan identitas
13 sebuah data. Atribut ini akan dihilangkan pada tabel anonimisasi.

14 • **sensitive_identifiers** adalah array yang berisi nama-nama atribut yang nilainya bersifat
15 sensitif. Atribut ini akan tetap ada pada tabel dianonimisasi.

16 • **quasi_identifiers** adalah atribut yang nilainya dapat dipakai untuk mengungkap entitas
17 data. Atribut ini berisi pasangan nilai berupa nama atribut dan jenis atribut. Jenis atribut
18 diisi dengan nilai "category" untuk menyatakan atribut kategori, sedangkan untuk menyatakan
19 atribut numerik jenis atribut akan diisi dengan nilai "numeric".

20 • **domain_generalization_hierarchy** adalah array yang menyimpan informasi mengenai pembentukan
21 pohon DGH. Atribut ini menyimpan 5 jenis informasi penting dalam pembentukan
22 pohon DGH, yaitu:

23 – **attrName** adalah nama atribut yang ingin dipilih pada tabel data. Atribut ini akan
24 memberi tahu Spark untuk mengambil atribut berdasarkan nama atribut.

25 – **value** adalah nilai-nilai yang mungkin muncul untuk atribut yang terpilih. Atribut ini
26 akan digunakan untuk memberi nama pada sebuah node di pohon DGH.

27 – **parent** adalah nilai yang menyatakan nama dari sebuah node root. Atribut ini akan
28 digunakan untuk menyatakan parent dari sebuah node di pohon DGH.

29 – **level** adalah nilai yang menyatakan ketinggian node di pohon DGH. Atribut ini digunakan
30 untuk menempatkan node di ketinggian tertentu di pohon DGH.

31 – **position** adalah nilai yang menyatakan posisi node di pohon DGH. Atribut ini digunakan
32 untuk menempatkan node di posisi (kiri/kanan) dari node parent.

33 Berikut adalah hal penting yang perlu diperhatikan terkait input (.json):

34 • Nilai atribut yang dicantumkan pada **domain_generalization_hierarchy** di salah satu
35 atribut harus mencakup seluruh kemungkinan nilai untuk atribut tersebut. Apabila tidak
36 terpenuhi, maka beberapa nilai tidak dapat dianonimisasi.

- 1 • Semakin banyak atribut yang dicantumkan pada `quasi_identifier`, maka waktu komputasi
2 akan semakin lama untuk dijalankan pada komputer lokal.

Listing 4.3: Input JSON untuk Anonimisasi Data

```
3 {
4
5     "k": <konstanta untuk k-anonymity dan greedy k-member clustering>,
6     "num_sample_datas": <jumlah sampel data>,
7     "input_path": "<path data input>",
8     "output_path": "<path data output>",
9     "identifier": [
10         {
11             "attrName": "<nama atribut>",
12             "dataType": "<tipe data atribut>"
13         }
14     ],
15     "sensitive_identifier": [
16         {
17             "attrName": "<nama atribut>",
18             "dataType": "<jenis atribut>"
19         }
20     ],
21     "quasi_identifier": [
22         {
23             "attrName": "<nama atribut>",
24             "dataType": "<jenis atribut>"
25         },
26         {
27             "attrName": "<nama atribut>",
28             "dataType": "<jenis atribut>"
29         }
30     ],
31     "domain_generalization_hierarchy": {
32         "<nama atribut>": [
33             {
34                 "value": "<nilai atribut>",
35                 "parent": "<nilai atribut parent>",
36                 "level": "1",
37                 "position": "null"
38             },
39             {
40                 "value": "<nilai atribut>",
41                 "parent": "<nilai atribut parent>",
42                 "level": "2",
43             }
44         ]
45     }
46 }
```

```

1      "position": "left"
2    },
3    {
4      "value": "<nilai atribut>",
5      "parent": "<nilai atribut parent>",
6      "level": "2",
7      "position": "right"
8    }
9  ]
10 }
11 }
12 }
```

4.4.3 Masukan Perangkat Lunak Pengujian

Perangkat lunak anonimisasi membutuhkan data masukan tambahan yaitu file .json yang berisi pohon klasifikasi serta tipe dan jenis atribut. File .json menerapkan format penyimpanan data dengan diawali oleh tanda ("{ }") dan diisi oleh nilai *key,value* yang dipisahkan oleh tanda titik dua (":"). Format file .json dapat dilihat pada Listing 4.4.

Berikut spesifikasi input .json untuk perangkat lunak pengujian:

- **input_path** adalah lokasi penyimpanan data input perangkat lunak. Atribut ini akan memberi tahu Spark untuk mengambil data input pada lokasi ini.
- **output_path** adalah lokasi penyimpanan hasil output perangkat lunak. Atribut ini akan memberi tahu Spark untuk menyimpan hasil output pada lokasi ini.
- **model_name** adalah jenis model yang akan dipakai untuk pengujian. Atribut ini terdiri dari 2 jenis nilai yaitu **k_means**,**naive_bayes**.
- **selected_column** adalah array yang menyimpan informasi pemilihan atribut. Atribut ini menyimpan 2 jenis informasi penting dalam pemilihan atribut, yaitu:
 - **attrName** adalah nama atribut yang ingin dipilih pada tabel data. Atribut ini akan memberi tahu Spark untuk mengambil kolom berdasarkan nama atribut.
 - **dataType** adalah jenis tipe data atribut yang telah dipilih. Atribut ini akan memberi tahu Spark untuk menyimpan atribut yang dipilih dalam format tertentu.
- **k_means** adalah parameter untuk pemodelan k-means. Atribut ini terdiri dari 1 parameter untuk pemodelan k-means, yaitu:
 - **k** adalah konstanta k pada pemodelan k-means. Atribut ini digunakan untuk menentukan jumlah kelompok data yang ingin dibentuk.
- **naive_bayes** adalah parameter untuk pemodelan naive bayes. Atribut ini terdiri dari 3 parameter untuk pemodelan naive bayes, yaitu:

- 1 – **label** adalah nama atribut yang akan dianggap sebagai label untuk pemodelan naive
2 bayes. Atribut ini digunakan untuk mengklasifikasikan data berdasarkan nilai pada
3 atribut label.
- 4 – **training_set** adalah persentase pembagian data training. Umumnya, persentase yang
5 dipilih untuk pembagian data training adalah 0.7.
- 6 – **test_set** adalah persentase pembagian data test. Umumnya, persentase yang dipilih
7 untuk pembagian data test adalah 0.3.

8 Berikut adalah hal penting yang perlu diperhatikan terkait input (.json):

- 9 • Atribut **input_path** untuk dapat diisi menggunakan lokasi data setelah anonimisasi maupun
10 data sebelum anonimisasi untuk membandingkan hasil keduanya.
- 11 • Atribut **model_name** hanya boleh diisi oleh satu jenis model, tidak boleh lebih. Jika diisi lebih
12 dari satu nilai, maka program akan mengeluarkan pesan error.
- 13 • Atribut **training_set** dan **test_set** harus berada pada rentang 0 sampai dengan 1. Lalu
14 ketika jumlah **training_set** dan **test_set** harus benilai 1.

Listing 4.4: Input JSON untuk Pengujian Data

```
15 {
16
17   "input_path": "<path data input>",
18   "output_path": "<path data output>",
19   "model_name": "<model pengujian (k_means/naive_bayes)>",
20   "selected_column": [
21     {
22       "attrName": "<nama atribut>",
23       "dataType": "<jenis atribut>"
24     },
25     {
26       "attrName": "<nama atribut>",
27       "dataType": "<jenis atribut>"
28     }
29   ],
30   "k_means": {
31     "k": <nilai k untuk k_means>
32   },
33   "naive_bayes": {
34     "label": "<nama atribut>",
35     "training_set": <persentase training_set (0.7)>,
36     "test_set": <persentase training_set (0.3)>
37   }
38 }
```


1

BAB 5

2

IMPLEMENTASI DAN PENGUJIAN

- 3 Pada bab ini akan dijelaskan lebih detil mengenai hasil implementasi perangkat lunak eksplorasi,
4 perangkat lunak anonimisasi, dan perangkat lunak pengujian.

5 **5.1 Implementasi Antarmuka**

- 6 Implementasi antarmuka pada lingkungan *big data* terbagi menjadi dua jenis, yaitu menggunakan
7 terminal *command line* jika eksperimen ingin dieksekusi pada lab *hadoop cluster* dan menggunakan
8 IntelliJ jika eksperimen ingin dieksekusi pada komputer lokal (*standalone*). Penjelasan implementasi
9 antarmuka akan dijelaskan lebih detil pada bagian selanjutnya.

10 **5.1.1 Komputer Lokal dengan IntelliJ**

- 11 Perangkat lunak dapat dijalankan menggunakan komputer lokal (*standalone*) menggunakan IntelliJ
12 jika data yang diolah tidak terlalu besar. Perangkat lunak IntelliJ dipakai untuk melakukan eksekusi
13 perangkat lunak Spark, ketika perangkat lunak masih dalam tahap pengembangan. Umumnya untuk
14 menjalankan perangkat lunak Spark, perangkat lunak harus dikonversi terlebih dahulu menjadi
15 (.jar) jika ingin dijalankan menggunakan terminal. Penggunaan IntelliJ membuat perangkat lunak
16 dapat langsung dieksekusi tanpa perlu mengkonversi perangkat lunak menjadi (.jar), sehingga
17 eksekusi dapat dilakukan dengan lebih mudah.



Gambar 5.1: IntelliJ

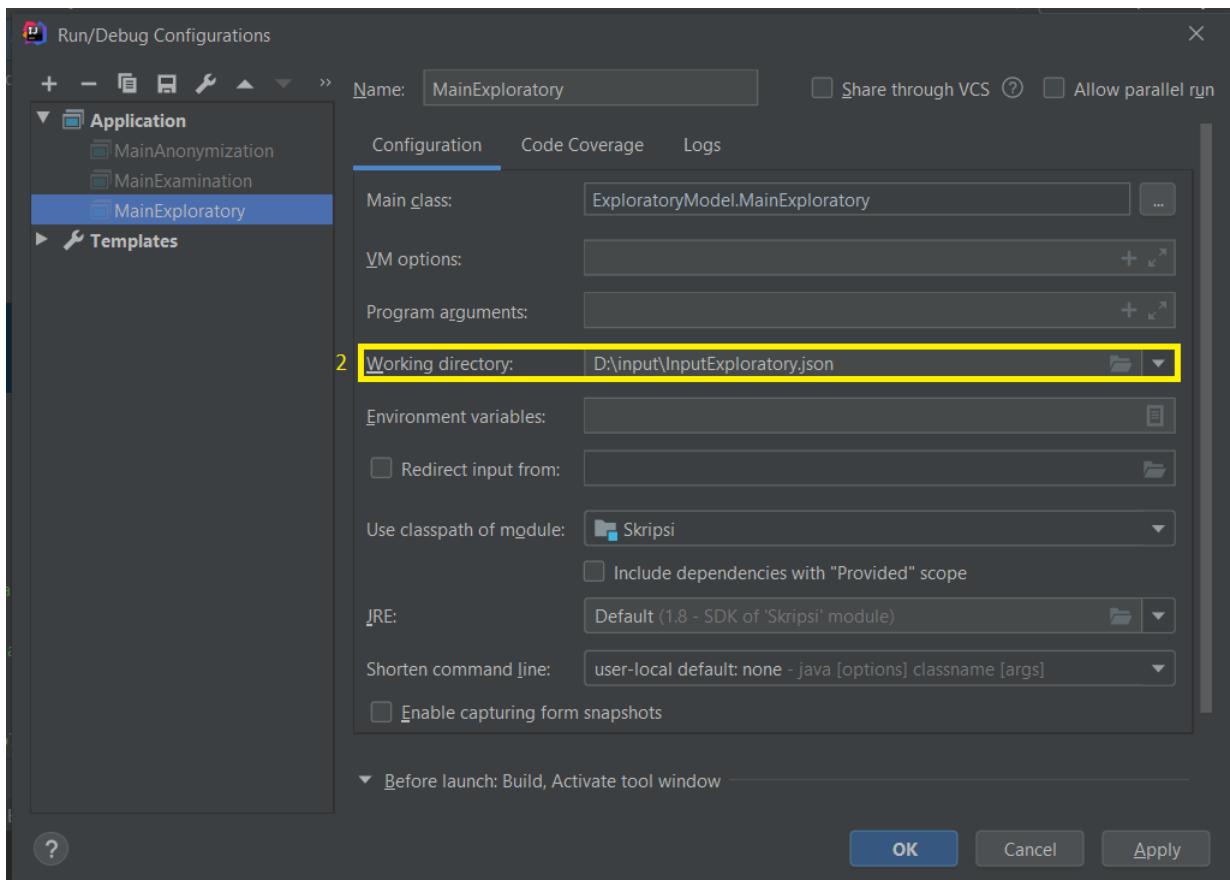
18 **Perangkat Lunak Eksplorasi**

- 19 Pada bagian ini, dijelaskan tahapan melakukan eksekusi pada perangkat lunak eksplorasi di IntelliJ.
20 Langkah pertama, mengisi lokasi folder JSON sebagai masukan dari perangkat lunak eksplorasi,
21 sebelum menjalankan perangkat lunak eksplorasi. Caranya adalah dengan memilih tombol *selector*
22 **Main class** pada sisi layar kanan atas dengan nama **MainExploratory**. Tombol *selector Main*
23 **class** ditandai dengan kotak kuning bermotor 1 pada Gambar 5.2.



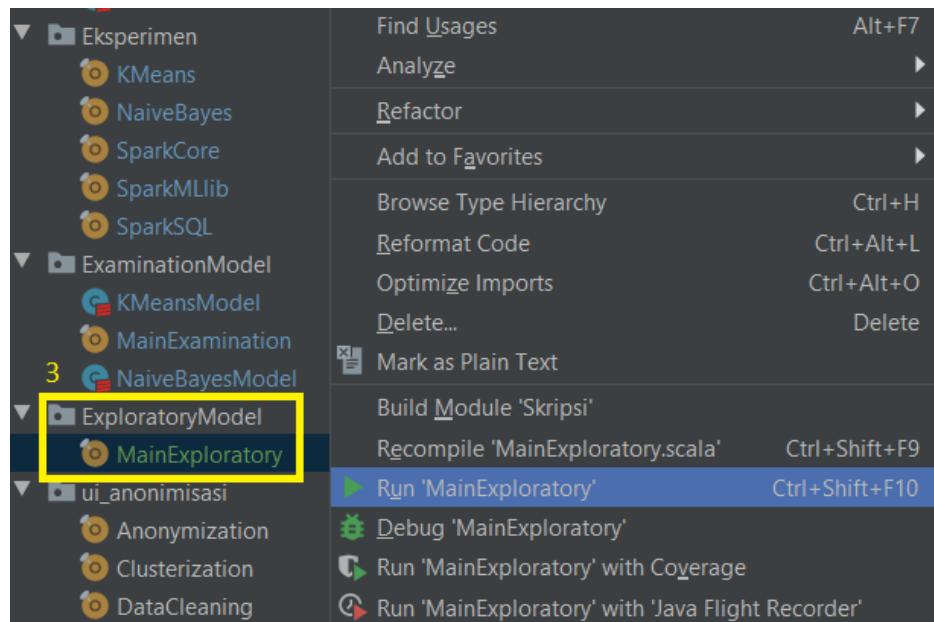
Gambar 5.2: Tombol Selector MainExploratory

- 1 Setelah itu, pilih menu **Edit Configurations...**, lalu isi parameter **Working directory** dengan lokasi **InputAnonymization.json**. Sebagai contoh, parameter lokasi JSON pada **Working directory** diisi dengan nilai **D:\input\InputAnonymization.json**. Pengisian parameter ditandai dengan kotak kuning bermotor 2 pada Gambar 5.3. Pada tahap ini langkah pertama telah selesai.



Gambar 5.3: Konfigurasi Parameter Perangkat Lunak Ekplorasi

- 5 Langkah kedua, menjalankan perangkat lunak ekplorasi dengan memilih **Main class** pada sisi layar kiri layar yaitu **\ExploratoryModel\MainExploratory**. **Main class** ditandai dengan kotak kuning bermotor 3 pada Gambar 5.4. Kemudian klik kanan pada **Main class** tersebut dan pilih menu **Run 'MainExploratory'**. Perangkat lunak akan menghasilkan output berupa tabel unik berdasarkan pengisian **selected_column** pada **InputExploratory.json**



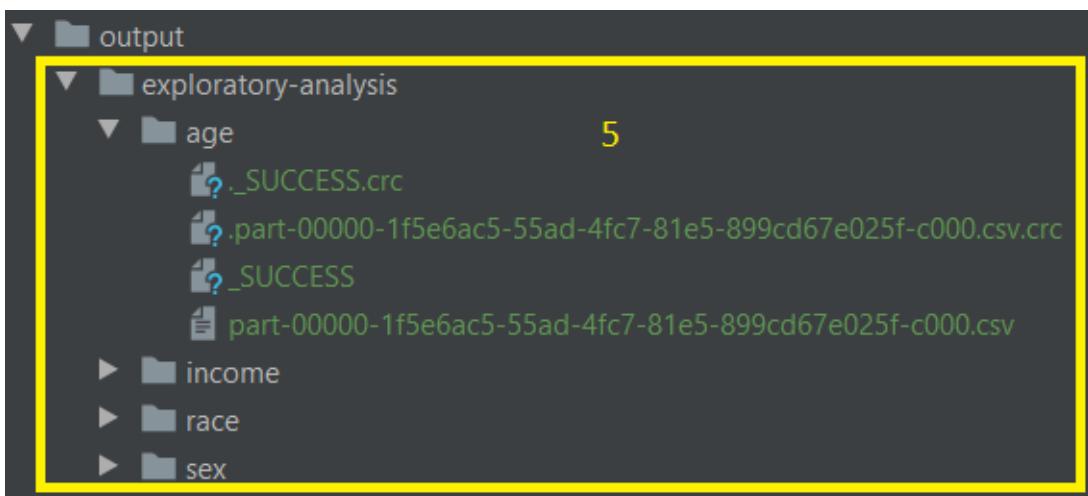
Gambar 5.4: Cara Menjalankan Perangkat Lunak Eksplorasi

1 Langkah terakhir adalah menunggu proses komputasi sampai dengan selesai. Sebagai catatan
 2 semakin banyak data yang diolah, maka waktu komputasi akan semakin lama. Perangkat lunak
 3 eksplorasi yang telah berhasil menyelesaikan proses komputasinya ditandai dengan baris log seperti
 4 berikut 'Process finished with exit code 0. Baris log ditandai dengan kotak kuning bernomor
 5 4 pada Gambar 5.5. Karena proses komputasi sudah selesai, output perangkat lunak eksplorasi
 6 sudah muncul berdasarkan output_path pada InputExploratory.json

```
20/10/25 12:45:54 INFO SparkUI: Stopped Spark web UI at http://DESKTOP-EP6CL8L:4040
20/10/25 12:45:54 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/10/25 12:45:54 INFO MemoryStore: MemoryStore cleared
20/10/25 12:45:54 INFO BlockManager: BlockManager stopped
20/10/25 12:45:54 INFO BlockManagerMaster: BlockManagerMaster stopped
20/10/25 12:45:54 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/10/25 12:45:54 INFO SparkContext: Successfully stopped SparkContext
20/10/25 12:45:54 INFO ShutdownHookManager: Shutdown hook called
20/10/25 12:45:54 INFO ShutdownHookManager: Deleting directory C:\Users\asus\AppData\Local\Temp\spark-1533322222
4
Process finished with exit code 0
```

Gambar 5.5: Log Perangkat Lunak Eksplorasi

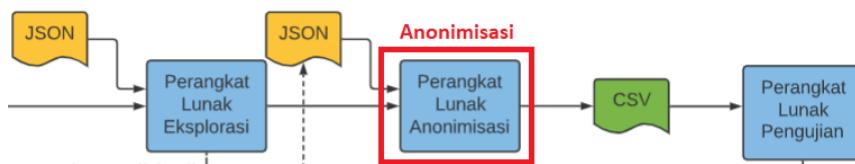
7 Output yang dihasilkan dari perangkat lunak eksplorasi dapat dilihat pada lokasi yang telah
 8 dicantumkan sebelumnya pada atribut output_path JSON. Sehingga ketika lokasi tersebut dibuka,
 9 maka akan ditampilkan seperti bagian kotak kuning bernomor 5 pada Gambar 5.6. Hasil eksplorasi
 10 nilai unik akan digunakan sebagai referensi mengisi nilai domain_generalization_hierarchy
 11 pada InputAnonymization.json. Eksplorasi dilakukan mengingat seluruh nilai atribut harus dapat
 12 diubah menjadi nilai anonimisasi. Output dapat dilihat ketika membuka file part-000X-YYYYY.csv.



Gambar 5.6: Folder Output Perangkat Lunak Ekplorasi

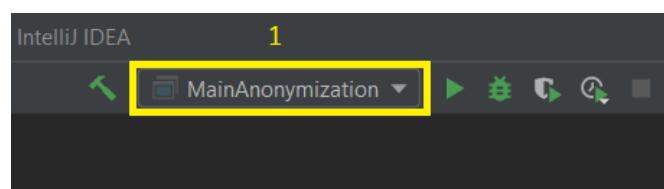
1 Perangkat Lunak Anonimisasi

- 2 Setelah mengeksekusi perangkat lunak eksplorasi untuk mencari tahu nilai unik pada setiap kolom *quasi-identifier*, pengujian dilanjutkan pada perangkat lunak anonimisasi seperti pada Gambar 5.7
- 3 untuk mencari tahu hasil pengelompokan dan anonimisasi data dengan algoritma greedy k-member clustering dan k-anonymity.



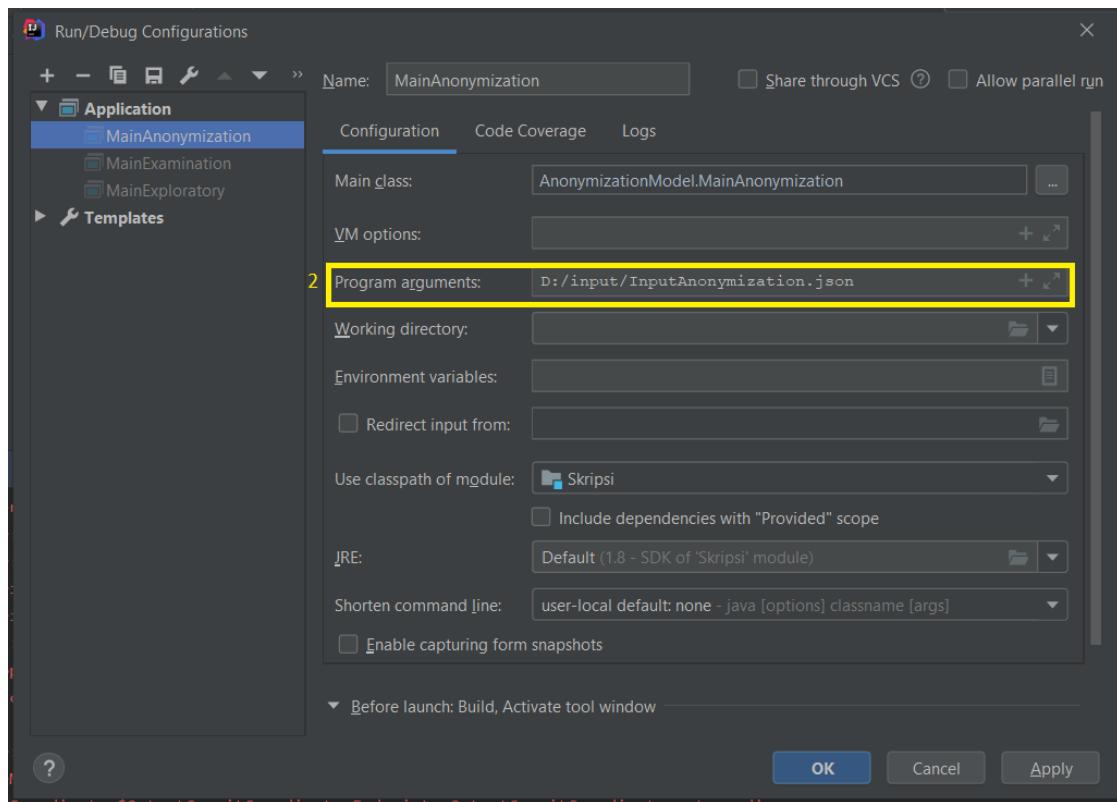
Gambar 5.7: Flowchart Penggunaan Perangkat Lunak

- 6 Pada bagian ini, dijelaskan tahapan melakukan eksekusi pada perangkat lunak eksplorasi di IntelliJ. Langkah pertama sebelum menjalankan perangkat lunak anonimisasi pada IntelliJ adalah mengisi parameter input. Caranya adalah dengan memilih tombol *selector Main class* pada sisi layar kanan atas dengan nama *MainAnonymization*. Tombol *selector Main class* ditandai dengan kotak kuning bernomor 1 pada Gambar 5.8.



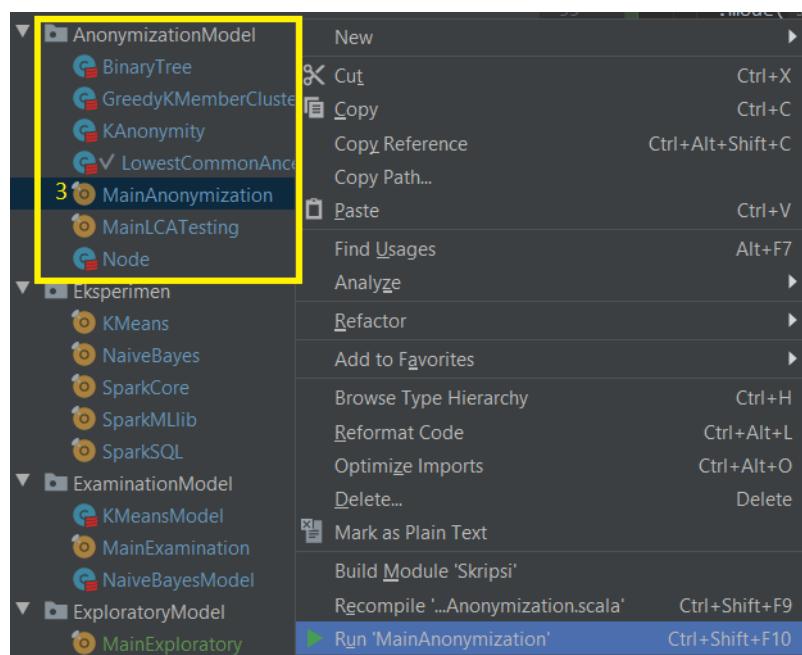
Gambar 5.8: Tombol Selector MainAnonymization

- 11 Setelah itu pilih menu *Edit Configurations...*, lalu isi parameter *Working directory* dengan lokasi *InputAnonymization.json*, contohnya pada Gambar ?? *Working directory* diisi dengan nilai *D:\input\InputAnonymization.json*. Pengisian parameter ditandai dengan kotak kuning bernomor 2 pada Gambar 5.9. Pada tahap ini langkah pertama telah selesai.



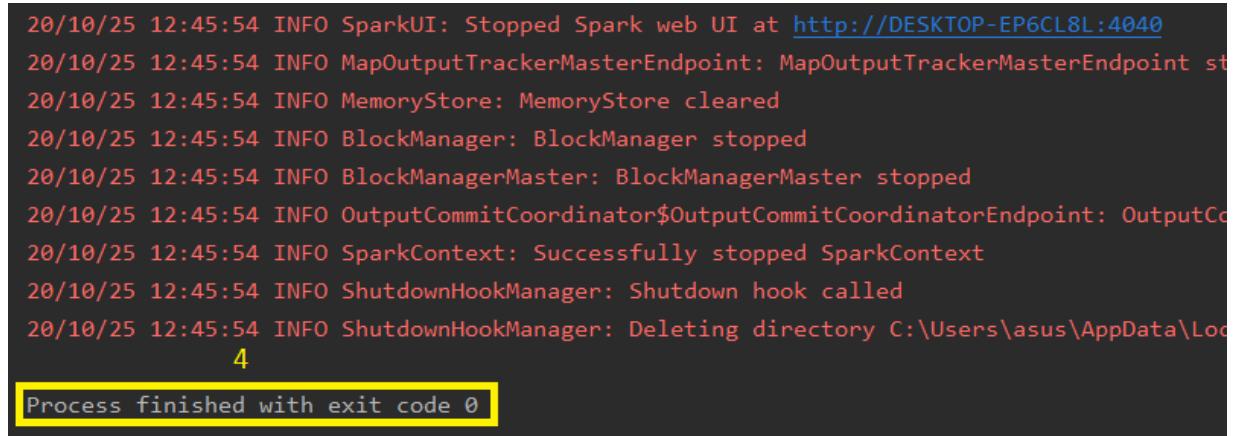
Gambar 5.9: Konfigurasi Parameter Perangkat Lunak Anonimisasi

- 1 Langkah kedua adalah menjalankan perangkat lunak anonimisasi pada IntelliJ dengan memilih
- 2 kelas pada `/AnonymizationModel/MainAnonymization`. Main class ditandai dengan kotak kuning bernomor 3 pada Gambar 5.10. Kemudian klik kanan pada `Main class` tersebut dan pilih menu `Run 'MainAnonymization'`. Perangkat lunak akan menghasilkan output berupa tabel unik berdasarkan pemilihan `selected_column` pada `InputAnonymization.json`



Gambar 5.10: Menjalankan Perangkat Lunak Anonimisasi

1 Langkah terakhir adalah menunggu proses komputasi sampai dengan selesai. Sebagai catatan
 2 semakin banyak data yang diolah, maka waktu komputasi akan semakin lama. Perangkat lunak
 3 anonimisasi yang telah berhasil menyelesaikan proses komputasinya ditandai dengan baris log seperti
 4 berikut 'Process finished with exit code 0. Baris log ditandai dengan kotak kuning bernomor
 5 4 pada Gambar 5.11. Karena proses komputasi sudah selesai, output perangkat lunak anonimisasi
 6 sudah muncul pada folder `output_path` yang dicantumkan di `InputAnonymization.json`

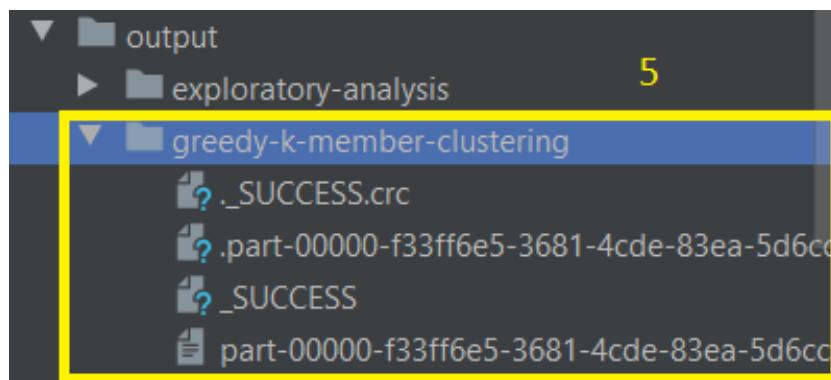


```

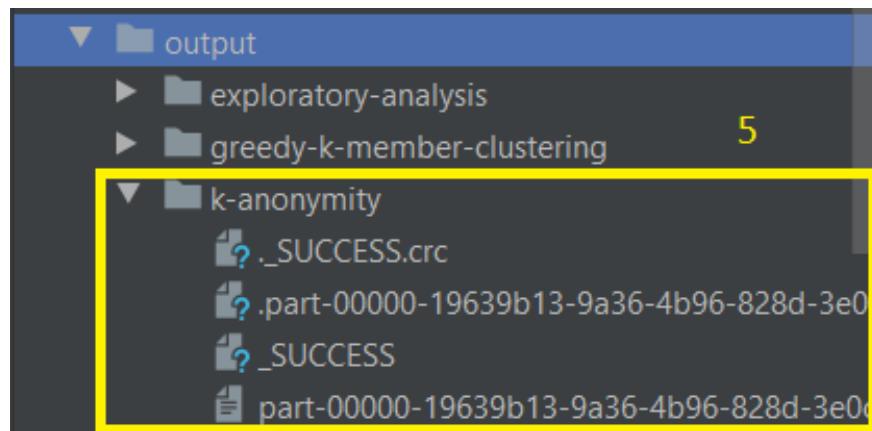
20/10/25 12:45:54 INFO SparkUI: Stopped Spark web UI at http://DESKTOP-EP6CL8L:4040
20/10/25 12:45:54 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint st
20/10/25 12:45:54 INFO MemoryStore: MemoryStore cleared
20/10/25 12:45:54 INFO BlockManager: BlockManager stopped
20/10/25 12:45:54 INFO BlockManagerMaster: BlockManagerMaster stopped
20/10/25 12:45:54 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCo
20/10/25 12:45:54 INFO SparkContext: Successfully stopped SparkContext
20/10/25 12:45:54 INFO ShutdownHookManager: Shutdown hook called
20/10/25 12:45:54 INFO ShutdownHookManager: Deleting directory C:\Users\asus\AppData\Loc
4
Process finished with exit code 0
  
```

Gambar 5.11: Log Perangkat Lunak Anonimisasi

7 Output yang dihasilkan dari perangkat lunak anonimisasi dapat dilihat pada lokasi yang telah
 8 dicantumkan sebelumnya pada nilai `output_path` JSON. Sehingga ketika lokasi tersebut dibuka,
 9 maka akan ditampilkan seperti bagian kotak kuning bernomor 5 pada Gambar 5.12, 5.13. Hasil
 10 anonimisasi digunakan pada pengujian *data mining* sebelum dan setelah anonimisasi. Sedangkan
 11 hasil pengelompokan digunakan untuk pengujian *total information loss*. Output dapat dilihat ketika
 12 membuka file `part-000X-YYYYYY.csv`.



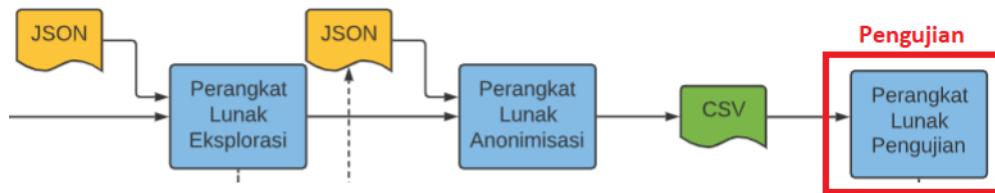
Gambar 5.12: Folder Output Greedy K-Member Clustering



Gambar 5.13: Folder Output K-Anonymity

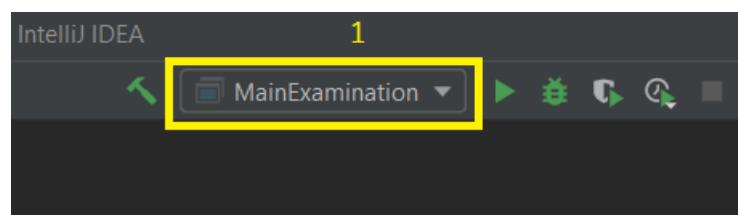
1 Perangkat Lunak Pengujian

- 2 Setelah mengeksekusi perangkat lunak anonimisasi untuk pengelompokan dan anonimisasi data kartu kredit, pengujian akan dilanjutkan pada perangkat lunak pengujian seperti pada Gambar 5.14
- 4 untuk mencari tahu kualitas hasil data mining sebelum dan setelah dilakukan proses anonimisasi.



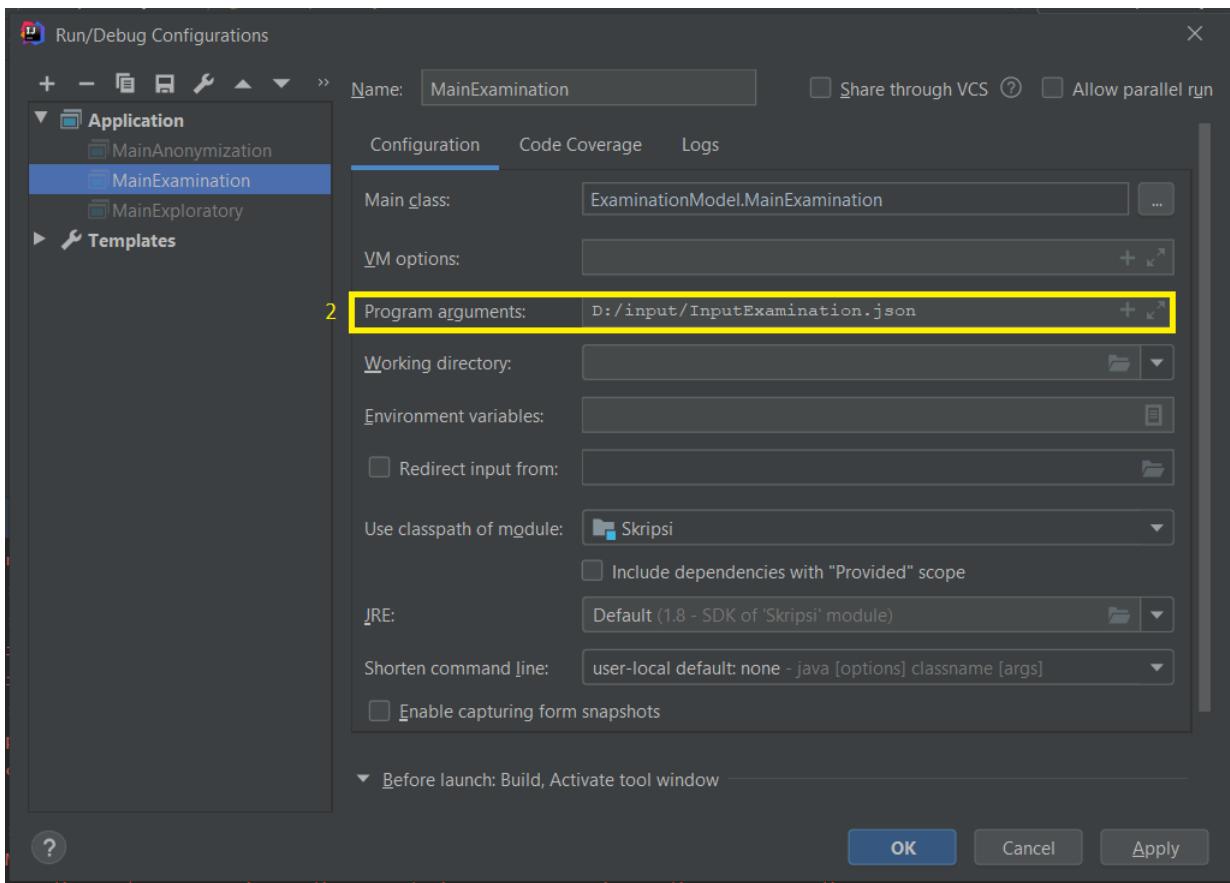
Gambar 5.14: Flowchart Penggunaan Perangkat Lunak (Lanjutan)

- 5 Pada bagian ini, dijelaskan tahapan melakukan eksekusi pada perangkat lunak pengujian di IntelliJ. Langkah pertama sebelum menjalankan perangkat lunak pengujian pada IntelliJ adalah mengisi parameter input. Caranya adalah dengan memilih tombol *selector Main class* pada sisi layar kanan atas dengan nama *MainExamination*. Tombol *selector Main class* ditandai dengan kotak kuning bernomor 1 pada Gambar 5.15.



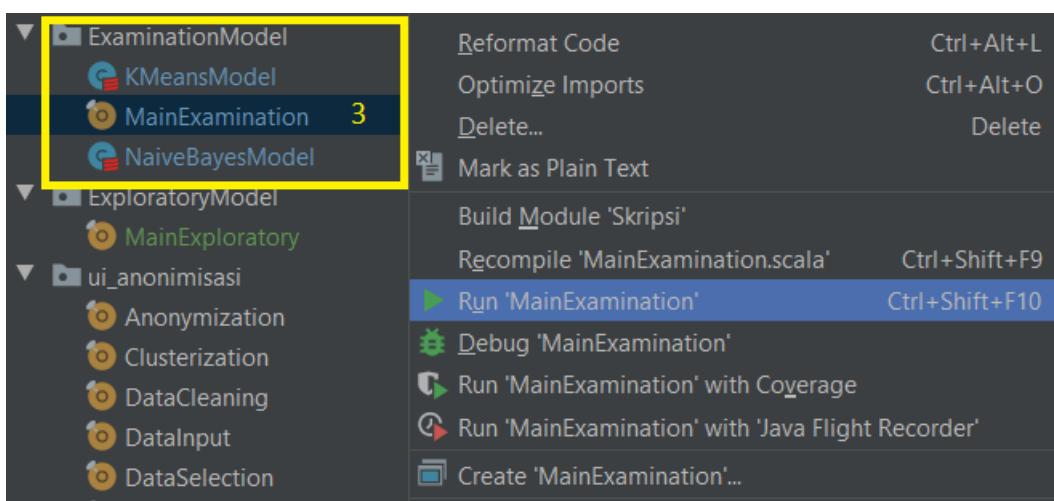
Gambar 5.15: Tombol Selector MainExamination

- 10 Setelah itu pilih menu *Edit Configurations...*, lalu isi parameter *Working directory* dengan lokasi *InputAnonymization.json*, contohnya pada Gambar ?? *Working directory* diisi dengan nilai *D:\input\InputExamination.json*. Pengisian parameter ditandai dengan kotak kuning bernomor 2 pada Gambar 5.16. Pada tahap ini langkah pertama telah selesai.



Gambar 5.16: Konfigurasi Parameter Perangkat Lunak Eksplorasi

- 1 Langkah kedua adalah menjalankan perangkat lunak anonimisasi pada IntelliJ dengan memilih
- 2 kelas pada `/ExaminationModel/MainExamination`. Main class ditandai dengan kotak kuning
- 3 bernomor 3 pada Gambar 5.17. Kemudian klik kanan pada `Main class` tersebut dan pilih menu
- 4 `Run 'MainExamination'`. Perangkat lunak menggunakan kolom prediktor pada proses *data mining*
- 5 berdasarkan pemilihan `selected_column` pada `InputExamination.json`



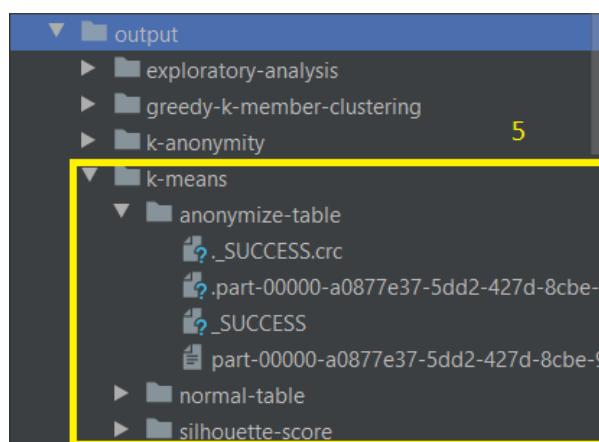
Gambar 5.17: Menjalankan Perangkat Lunak Eksplorasi

1 Langkah terakhir adalah menunggu proses komputasi sampai dengan selesai. Sebagai catatan
 2 semakin banyak data yang diolah, maka waktu komputasi akan semakin lama. Perangkat lunak
 3 pengujian yang telah berhasil menyelesaikan proses komputasinya ditandai dengan baris log seperti
 4 berikut 'Process finished with exit code 0. Baris log ditandai dengan kotak kuning bernomor
 5 4 pada Gambar 5.18. Karena proses komputasi sudah selesai, output perangkat lunak pengujian
 6 sudah muncul pada folder `output_path` yang dicantumkan di `InputExploratory.json`

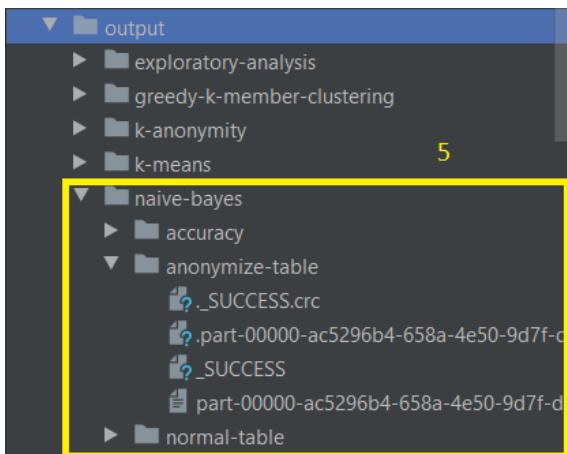
```
20/10/25 12:45:54 INFO SparkUI: Stopped Spark web UI at http://DESKTOP-EP6CL8L:4040
20/10/25 12:45:54 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint st
20/10/25 12:45:54 INFO MemoryStore: MemoryStore cleared
20/10/25 12:45:54 INFO BlockManager: BlockManager stopped
20/10/25 12:45:54 INFO BlockManagerMaster: BlockManagerMaster stopped
20/10/25 12:45:54 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCo
20/10/25 12:45:54 INFO SparkContext: Successfully stopped SparkContext
20/10/25 12:45:54 INFO ShutdownHookManager: Shutdown hook called
20/10/25 12:45:54 INFO ShutdownHookManager: Deleting directory C:\Users\asus\AppData\Loc
4
Process finished with exit code 0
```

Gambar 5.18: Log Perangkat Lunak Ekplorasi

7 Output yang dihasilkan dari perangkat lunak pengujian dapat dilihat pada lokasi yang telah
 8 dicantumkan sebelumnya pada nilai `output_path` JSON. Contoh JSON dapat dilihat pada Listing
 9 E.3. Ketika lokasi tersebut dibuka, maka akan ditampilkan seperti bagian kotak kuning bernomor
 10 5 pada Gambar 5.19, 5.20. Hasil pengujian digunakan untuk analisis lebih lanjut pada pengujian
 11 fungsional dan pengujian eksperimental di subbab selanjutnya. Pengujian dilakukan untuk mencari
 12 perbedaan kualitas hasil *data mining* sebelum dan setelah dilakukan anonimisasi. Output dapat
 13 dilihat ketika membuka file `part-000X-YYYYY.csv`.



Gambar 5.19: Folder Output Perangkat Lunak Pengujian



Gambar 5.20: Folder Output Perangkat Lunak Pengujian

5.1.2 Hadoop Cluster dengan Terminal Ubuntu

Pengujian ini dilakukan menggunakan Hadoop cluster dengan 10 slaves node yang dapat digunakan untuk melakukan proses komputasi big data. Penggunaan 10 slaves node dipilih karena resouces yang disediakan pada hadoop cluster lab hanya 10 komputer. Perangkat lunak yang ingin dieksusi pada hadoop cluster harus dikonversi menjadi (.jar).

6

Gambar 5.21 adalah spesifikasi *slaves node* pada pengujian Hadoop cluster:

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
slave6:50010 (10.100.69.213:50010)	0	In Service	907.92 GB	282.77 MB	63.57 GB	844.08 GB	78	282.77 MB (0.03%)	0	2.7.1
slave9:50010 (10.100.69.216:50010)	2	In Service	224.02 GB	357.2 MB	28.91 GB	194.76 GB	76	357.2 MB (0.16%)	0	2.7.1
slave3:50010 (10.100.69.210:50010)	2	In Service	146.65 GB	394.67 MB	34.37 GB	111.89 GB	76	394.67 MB (0.26%)	0	2.7.1
slave7:50010 (10.100.69.214:50010)	0	In Service	907.92 GB	452.92 MB	298.94 GB	608.54 GB	63	452.92 MB (0.05%)	0	2.7.1
slave2:50010 (10.100.69.209:50010)	0	In Service	146.65 GB	1.09 GB	37.82 GB	107.74 GB	67	1.09 GB (0.74%)	0	2.7.1
slave4:50010 (10.100.69.211:50010)	2	In Service	907.92 GB	670.63 MB	64.06 GB	843.21 GB	73	670.63 MB (0.07%)	0	2.7.1
slave5:50010 (10.100.69.212:50010)	2	In Service	911.68 GB	498.01 MB	65.04 GB	846.15 GB	81	498.01 MB (0.05%)	0	2.7.1
slave1:50010 (10.100.69.207:50010)	0	In Service	146.65 GB	440.79 MB	57.85 GB	88.36 GB	75	440.79 MB (0.29%)	0	2.7.1
slave10:50010 (10.100.69.217:50010)	0	In Service	146.65 GB	473.57 MB	26.74 GB	119.44 GB	82	473.57 MB (0.32%)	0	2.7.1
slave8:50010 (10.100.69.215:50010)	0	In Service	146.65 GB	650.28 MB	29.53 GB	116.48 GB	79	650.28 MB (0.43%)	0	2.7.1

Gambar 5.21: Spesifikasi Slaves Node pada Hadoop Cluster

- 8 • Node adalah nama dan alamat slaves node yang tersedia pada Hadoop cluster.
- 9 • Capacity adalah kapasitas penyimpanan data pada masing-masing slaves node.
- 10 • Used adalah jumlah kapasitas yang terpakai pada masing-masing slaves node.
- 11 • Remaining adalah sisa kapasitas penyimpanan pada masing-masing slaves node.

1 Perangkat Lunak Eksplorasi

2 Pada bagian ini, dijelaskan tahapan melakukan eksekusi pada perangkat lunak eksplorasi di hadoop
 3 cluster. Langkah pertama sebelum menjalankan perangkat lunak eksplorasi pada terminal Ubuntu
 4 adalah menyimpan data input pada sistem HDFS. Data input yang dipakai adalah `credit430k.csv`
 5 dan parameter program `InputExploratory.json`. Listing 5.1 adalah perintah untuk membuat
 6 folder dan menyimpan data input pada HDFS. Hasil folder dan file yang disimpan pada HDFS
 7 dapat dilihat menggunakan *browser* dengan alamat `http://10.100.69.101:50070/`.

Listing 5.1: Perintah Spark untuk Perangkat Lunak Eksplorasi

```

8 // Perintah untuk membuat folder di HDFS
9 hadoop fs -mkdir <nama folder HDFS>
10 // Perintah untuk menyimpan file adult100k.csv di HDFS
11 hadoop fs -put /home/hduser/<nama folder>/<nama file>.csv /<nama folder HDFS>/
12 // Perintah untuk menyimpan file InputAnonymization.json di HDFS
13 hadoop fs -put /home/hduser/<nama folder>/<nama file>.json /<nama folder HDFS>/
14 // Perintah untuk menghapus sebuah folder pada HDFS
15 hadoop fs -rm -r -f /<nama folder HDFS>
16
17

```

18 Langkah kedua adalah memastikan bahwa *file* sudah ditempatkan pada folder dengan nama
 19 yang sesuai. Hal ini perlu diperhatikan, karena jika terjadi kesalahan input pada nama *file* atau
 20 nama folder, program dapat menampilkan pesan *error*. Gambar 5.22 adalah contoh penempatan
 21 *file* pada folder HDFS dengan benar karena tidak ada kesalahan penulisan nama.

Browse Directory							
/skripsi-jordan							
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	4.46 KB	10/25/2020, 7:39:35 PM	3	128 MB	InputAnonymization.json
-rw-r--r--	hduser	hduser	4.47 KB	10/27/2020, 5:22:43 PM	3	128 MB	InputAnonymization100k.json
-rw-r--r--	hduser	hduser	639 B	10/25/2020, 7:10:35 PM	3	128 MB	InputExamination.json
-rw-r--r--	hduser	hduser	486 B	10/25/2020, 7:10:35 PM	3	128 MB	InputExploratory.json
-rw-r--r--	hduser	hduser	10.71 MB	10/25/2020, 7:42:57 PM	3	128 MB	adult100k.csv
-rw-r--r--	hduser	hduser	108.88 MB	10/27/2020, 5:08:03 PM	3	128 MB	adult1m.csv
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 6:27:28 PM	0	0 B	output
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 5:23:35 PM	0	0 B	output100k

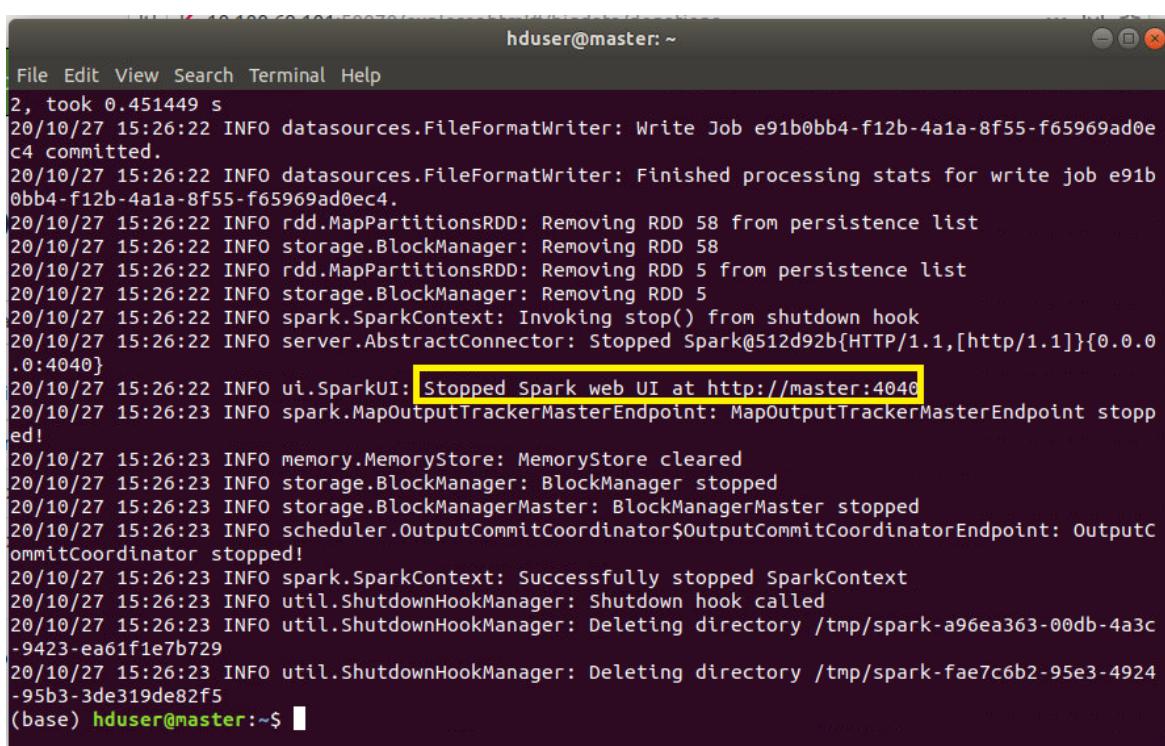
Gambar 5.22: File Input Eksplorasi HDFS

1 Langkah ketiga adalah melakukan eksekusi perangkat lunak eksplorasi menggunakan perintah
 2 Spark. Listing 5.23 adalah contoh perintah eksekusi pada Spark. Format --class ExploratoryModel.
 3 MainExploratory artinya menunjukkan Main class yang dieksekusi bernama MainExploratory
 4 pada package ExploratoryModel. Format --master yarn menunjukkan bahwa program dieksekusi
 5 pada sebuah komputer cluster. Format /home/hduser/skripsi-stephen/skripsi.jar menunjuk-
 6 an lokasi JAR pada komputer. Format /skripsi-jordan/InputExploratory.json menunjukkan
 7 lokasi JSON pada HDFS. Pada tahap ini, perintah Spark siap untuk dieksekusi.

Listing 5.2: Perintah Eksekusi Spark

```
8 spark-submit --class ExploratoryModel.MainExploratory --master yarn /home/hduser/
9   skripsi-stephen/skripsi.jar /skripsi-jordan/InputExploratory.json
10
11
```

12 Langkah terakhir adalah menunggu proses komputasi sampai dengan selesai. Perangkat lunak
 13 eksplorasi yang telah berhasil menyelesaikan proses komputasinya pada terminal Ubuntu ditandai
 14 dengan baris *log* seperti Gambar 5.23: 'Stopped Spark Web UI at http://master:4040'. Sete-
 15 lah proses komputasi selesai, maka output perangkat lunak eksplorasi sudah disimpan pada HDFS
 16 dengan lokasi sebagai berikut /skripsi-jordan/output/exploratory-analysis



```
File Edit View Search Terminal Help
2, took 0.451449 s
20/10/27 15:26:22 INFO datasources.FileFormatWriter: Write Job e91b0bb4-f12b-4a1a-8f55-f65969ad0e
c4 committed.
20/10/27 15:26:22 INFO datasources.FileFormatWriter: Finished processing stats for write job e91b
0bb4-f12b-4a1a-8f55-f65969ad0ec4.
20/10/27 15:26:22 INFO rdd.MapPartitionsRDD: Removing RDD 58 from persistence list
20/10/27 15:26:22 INFO storage.BlockManager: Removing RDD 58
20/10/27 15:26:22 INFO rdd.MapPartitionsRDD: Removing RDD 5 from persistence list
20/10/27 15:26:22 INFO storage.BlockManager: Removing RDD 5
20/10/27 15:26:22 INFO spark.SparkContext: Invoking stop() from shutdown hook
20/10/27 15:26:22 INFO server.AbstractConnector: Stopped Spark@512d92b{HTTP/1.1,[http/1.1]}{0.0.0
.0:4040}
20/10/27 15:26:22 INFO ui.SparkUI: Stopped Spark web UI at http://master:4040
20/10/27 15:26:23 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopp
ed!
20/10/27 15:26:23 INFO memory.MemoryStore: MemoryStore cleared
20/10/27 15:26:23 INFO storage.BlockManager: BlockManager stopped
20/10/27 15:26:23 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
20/10/27 15:26:23 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputC
ommitCoordinator stopped!
20/10/27 15:26:23 INFO spark.SparkContext: Successfully stopped SparkContext
20/10/27 15:26:23 INFO util.ShutdownHookManager: Shutdown hook called
20/10/27 15:26:23 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-a96ea363-00db-4a3c
-9423-ea61f1e7b729
20/10/27 15:26:23 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-fae7c6b2-95e3-4924
-95b3-3de319de82f5
(base) hduser@master:~$
```

Gambar 5.23: Log Perangkat Lunak Eksplorasi

17 Ketika lokasi output pada HDFS dibuka, maka akan tampak seperti Gambar 5.24. Hasil eksplor-
 18 rasi nilai unik akan digunakan sebagai referensi mengisi nilai domain_generalization_hierarchy
 19 pada InputExploratory.json. Eksplorasi perlu dilakukan mengingat seluruh nilai atribut harus
 20 dapat diubah menjadi nilai anonimisasi. Gambar 5.24 menujukan lokasi folder nilai unik un-
 21 tuk masing-masing atribut. Gambar 5.25 menujukan lokasi output disimpan /skripsi-jordan/
 22 exploratory-analysis/workclass/part-000X-YYYYYY.csv.

Browse Directory								
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 9:57:39 PM	0	0 B	age	
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 9:57:41 PM	0	0 B	income	
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 9:57:40 PM	0	0 B	race	
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 9:57:41 PM	0	0 B	sex	

Gambar 5.24: Folder HDFS Hasil Ekplorasi

Browse Directory								
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-r--r--	hduser	hduser	0 B	10/27/2020, 9:57:40 PM	3	128 MB	_SUCCESS	
-rw-r--r--	hduser	hduser	61 B	10/27/2020, 9:57:40 PM	3	128 MB	part-00000-clefb8e3-d4dc-48ca-b9f0-574031e46224-c000.csv	

Gambar 5.25: Folder HDFS Hasil Ekplorasi Atribut Race

1 Perangkat Lunak Anonimisasi

- 2 Pada bagian ini, dijelaskan tahapan melakukan eksekusi pada perangkat lunak anonimisasi di *hadoop cluster*. Langkah pertama sebelum menjalankan perangkat lunak ekplorasi pada terminal Ubuntu 4 adalah menyimpan data input pada sistem HDFS. Data input yang dibutuhkan antara lain dataset 5 *adult100k.csv* dan parameter program *InputAnonymization.json*. Listing 5.3 adalah perintah 6 untuk membuat folder dan menyimpan data input pada HDFS. Hasil folder dan file yang disimpan 7 pada HDFS dapat dilihat menggunakan *browser* dengan alamat <http://10.100.69.101:50070/>.

Listing 5.3: Perintah Spark untuk Perangkat Lunak Anonimisasi

```

8 // Perintah untuk membuat folder di HDFS
9 hadoop fs -mkdir /<nama folder HDFS>
10 // Perintah untuk menyimpan file adult100k.csv di HDFS
11 hadoop fs -put /home/hduser/<nama folder>/<nama file>.csv /<nama folder HDFS>/
12 // Perintah untuk menyimpan file InputAnonymization.json di HDFS
13 hadoop fs -put /home/hduser/<nama folder>/<nama file>.json /<nama folder HDFS>/
14 // Perintah untuk menghapus sebuah folder pada HDFS
15 hadoop fs -rm -r -f /<nama folder HDFS>
16

```

- 1 Langkah kedua adalah memastikan bahwa *file* sudah ditempatkan pada folder dengan nama
 2 yang sesuai. Hal ini perlu diperhatikan, karena jika terjadi kesalahan input pada nama *file* atau
 3 nama folder, program dapat menampilkan pesan *error*. Gambar ?? adalah contoh penempatan *file*
 4 pada folder HDFS dengan benar karena tidak ada kesalahan penulisan nama.

Browse Directory							
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	4.46 KB	10/25/2020, 7:39:35 PM	3	128 MB	InputAnonymization.json
-rw-r--r--	hduser	hduser	4.47 KB	10/27/2020, 5:22:43 PM	3	128 MB	InputAnonymization100k.json
-rw-r--r--	hduser	hduser	639 B	10/25/2020, 7:10:35 PM	3	128 MB	InputExamination.json
-rw-r--r--	hduser	hduser	486 B	10/25/2020, 7:10:35 PM	3	128 MB	InputExploratory.json
-rw-r--r--	hduser	hduser	10.71 MB	10/25/2020, 7:42:57 PM	3	128 MB	adult100k.csv
-rw-r--r--	hduser	hduser	108.88 MB	10/27/2020, 5:08:03 PM	3	128 MB	adult1m.csv
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 6:27:28 PM	0	0 B	output
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 5:23:35 PM	0	0 B	output100k

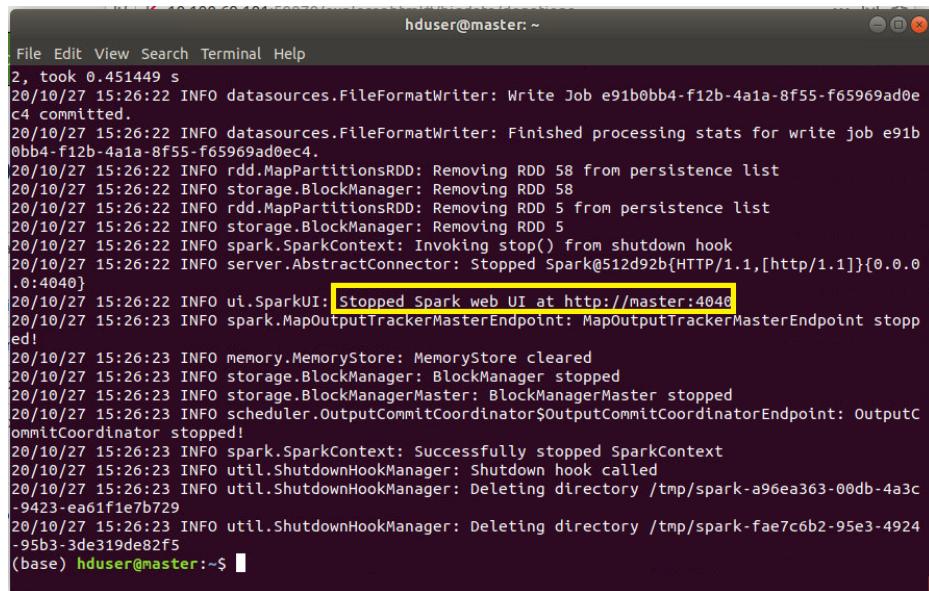
Gambar 5.26: File Input Anonimisasi HDFS

- 5 Langkah ketiga adalah melakukan eksekusi perangkat lunak eksplorasi menggunakan perintah
 6 Spark. Listing 1 adalah contoh perintah eksekusi pada Spark. Format `--class AnonymizationModel`.
 7 `MainAnonymization` artinya menunjukkan kelas Main yang dieksekusi bernama `MainAnonymization`
 8 pada `package AnonymizationModel`. Format `--master yarn` menunjukkan bahwa program diekse-
 9 kusi pada sebuah cluster komputer. Format `/home/hduser/skripsi-stephen/skripsi.jar` me-
 10 nunjukkan lokasi JAR pada komputer. Format `/skripsi-jordan/InputAnonymization100k.json`
 11 menunjukkan lokasi JSON pada HDFS. Pada tahap ini, perintah siap dieksekusi.

Listing 5.4: Perintah Eksekusi Spark

```
12
13 spark-submit --class AnonymizationModel.MainAnonymization --master yarn /home/
14   hduser/skripsi-stephen/skripsi.jar /skripsi-jordan/InputAnonymization100k.json
15
```

- 16 Langkah terakhir adalah menunggu proses komputasi sampai dengan selesai. Perangkat lu-
 17 nak eksplorasi yang telah berhasil menyelesaikan proses komputasinya pada terminal Ubuntu
 18 ditandai dengan baris *log* seperti Gambar 5.27: 'Stopped Spark Web UI at http://master:
 19 4040'. Setelah proses komputasi selesai, maka output perangkat lunak anonimisasi telah ter-
 20 simpan pada HDFS. Hasil pengelompokan data disimpan pada lokasi HDFS sebagai berikut
 21 `/skripsi-jordan/output/greedy-k-member-clustering`, sedangkan hasil anonimisasi data di-
 22 simpan pada lokasi HDFS sebagai berikut `/skripsi-jordan/output/k-anonymity`.



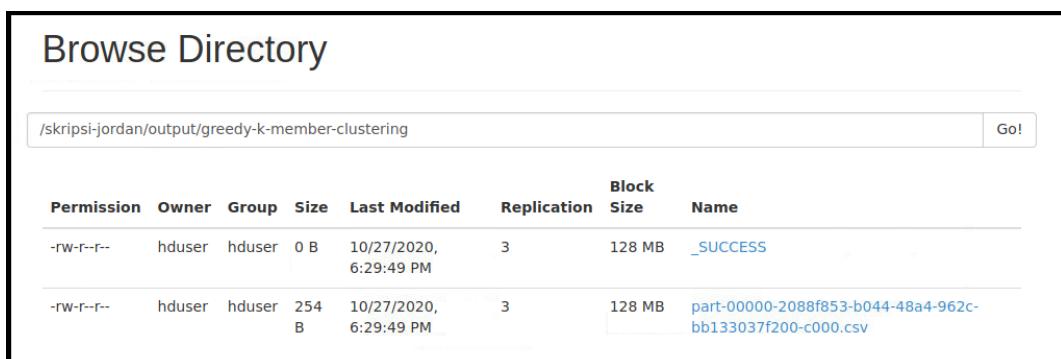
```

File Edit View Search Terminal Help
2, took 0.451449 s
20/10/27 15:26:22 INFO datasources.FileFormatWriter: Write Job e91b0bb4-f12b-4a1a-8f55-f65969ad0e
c4 committed.
20/10/27 15:26:22 INFO datasources.FileFormatWriter: Finished processing stats for write job e91b
0bb4-f12b-4a1a-8f55-f65969ad0ec4.
20/10/27 15:26:22 INFO rdd.MapPartitionsRDD: Removing RDD 58 from persistence list
20/10/27 15:26:22 INFO storage.BlockManager: Removing RDD 58
20/10/27 15:26:22 INFO rdd.MapPartitionsRDD: Removing RDD 5 from persistence list
20/10/27 15:26:22 INFO storage.BlockManager: Removing RDD 5
20/10/27 15:26:22 INFO spark.SparkContext: Invoking stop() from shutdown hook
20/10/27 15:26:22 INFO server.AbstractConnector: Stopped Spark@512d92b[HTTP/1.1,[http/1.1]]{0.0.0
.0:4040}
20/10/27 15:26:22 INFO ui.SparkUI: Stopped Spark web UI at http://master:4040
20/10/27 15:26:23 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopp
ed!
20/10/27 15:26:23 INFO memory.MemoryStore: MemoryStore cleared
20/10/27 15:26:23 INFO storage.BlockManager: BlockManager stopped
20/10/27 15:26:23 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
20/10/27 15:26:23 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputC
ommitCoordinator stopped!
20/10/27 15:26:23 INFO spark.SparkContext: Successfully stopped SparkContext
20/10/27 15:26:23 INFO util.ShutdownHookManager: Shutdown hook called
20/10/27 15:26:23 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-a96ea363-00db-4a3c
-9423-ea61fe7b729
20/10/27 15:26:23 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-fae7c6b2-95e3-4924
-95b3-3de319de82f5
(base) hduser@master:~$ 

```

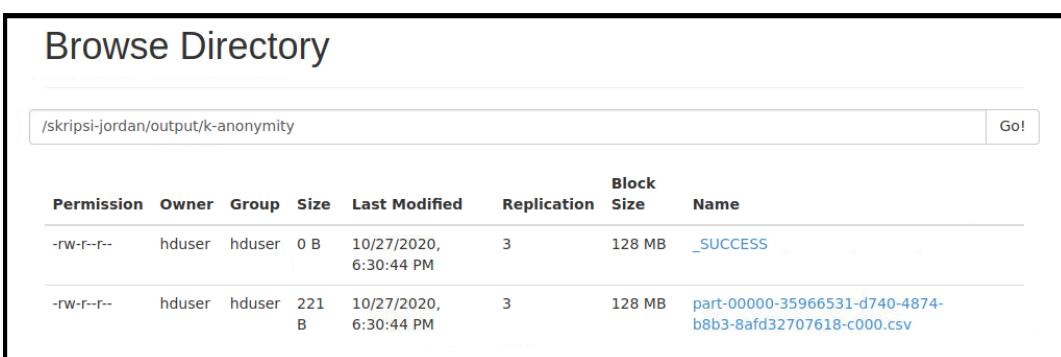
Gambar 5.27: Log Perangkat Lunak Anonimisasi

- 1 Ketika lokasi output pada HDFS dibuka, maka tampak seperti Gambar 5.28, 5.29. Hasil eksplorasi nilai unik akan digunakan sebagai referensi mengisi nilai `domain_generalization_hierarchy` pada `InputAnonymization.json`. Gambar 5.28 menunjukkan folder hasil pengelompokan data pada `/skripsi-jordan/greedy-k-member-clustering/part-000X-YYYYYY.csv`. Gambar 5.29 menunjukkan folder hasil pengelompokan pada `/skripsi-jordan/k-anonymity/part-000X-YYYYYY.csv`.
- 6 Untuk membuka hasil output, maka CSV harus diunduh terlebih dahulu.



Browse Directory							
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	0 B	10/27/2020, 6:29:49 PM	3	128 MB	_SUCCESS
-rw-r--r--	hduser	hduser	254 B	10/27/2020, 6:29:49 PM	3	128 MB	part-00000-2088f853-b044-48a4-962c-bb133037f200-c000.csv

Gambar 5.28: Folder HDFS Hasil Pengelompokan Data



Browse Directory							
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	0 B	10/27/2020, 6:30:44 PM	3	128 MB	_SUCCESS
-rw-r--r--	hduser	hduser	221 B	10/27/2020, 6:30:44 PM	3	128 MB	part-00000-35966531-d740-4874-b8b3-8af32707618-c000.csv

Gambar 5.29: Folder HDFS Hasil Anonimisasi Data

1 Perangkat Lunak Pengujian

2 Pada bagian ini, dijelaskan tahapan melakukan eksekusi pada perangkat lunak pengujian di *hadoop cluster*. Langkah pertama sebelum menjalankan perangkat lunak pengujian pada terminal Ubuntu 3 adalah menyimpan data input pada sistem HDFS. Data input yang dibutuhkan antara lain dataset 4 `credit430k.csv` dan parameter program `InputExamination.json`. Listing 5.5 adalah perintah 5 untuk membuat folder dan menyimpan data input pada HDFS. Hasil folder dan *file* yang disimpan 6 pada HDFS dapat dilihat menggunakan *browser* dengan alamat `http://10.100.69.101:50070/`. 7

Listing 5.5: Perintah Spark untuk Perangkat Lunak Pengujian

```
8 // Perintah untuk membuat folder di HDFS
9 hadoop fs -mkdir /<nama folder HDFS>
10 // Perintah untuk menyimpan file adult100k.csv di HDFS
11 hadoop fs -put /home/hduser/<nama folder>/<nama file>.csv /<nama folder HDFS>/
12 // Perintah untuk menyimpan file InputAnonymization.json di HDFS
13 hadoop fs -put /home/hduser/<nama folder>/<nama file>.json /<nama folder HDFS>/
14 // Perintah untuk menghapus sebuah folder pada HDFS
15 hadoop fs -rm -r -f /<nama folder HDFS>
16
17
```

18 Langkah kedua adalah memastikan bahwa *file* sudah ditempatkan pada folder dengan nama 19 yang sesuai. Hal ini perlu diperhatikan, karena jika terjadi kesalahan input pada nama *file* atau 20 nama folder, program dapat menampilkan pesan *error*. Gambar 5.30 adalah contoh penempatan 21 *file* pada folder HDFS dengan benar karena tidak ada kesalahan penulisan nama.

Browse Directory							
/skripsi-jordan							
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	4.46 KB	10/25/2020, 7:39:35 PM	3	128 MB	InputAnonymization.json
-rw-r--r--	hduser	hduser	4.47 KB	10/27/2020, 5:22:43 PM	3	128 MB	InputAnonymization100k.json
-rw-r--r--	hduser	hduser	639 B	10/25/2020, 7:10:35 PM	3	128 MB	InputExamination.json
-rw-r--r--	hduser	hduser	486 B	10/25/2020, 7:10:35 PM	3	128 MB	InputExploratory.json
-rw-r--r--	hduser	hduser	10.71 MB	10/25/2020, 7:42:57 PM	3	128 MB	adult100k.csv
-rw-r--r--	hduser	hduser	108.88 MB	10/27/2020, 5:08:03 PM	3	128 MB	adult1m.csv
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 6:27:28 PM	0	0 B	output
drwxr-xr-x	hduser	hduser	0 B	10/27/2020, 5:23:35 PM	0	0 B	output100k

Gambar 5.30: File Input Pengujian HDFS

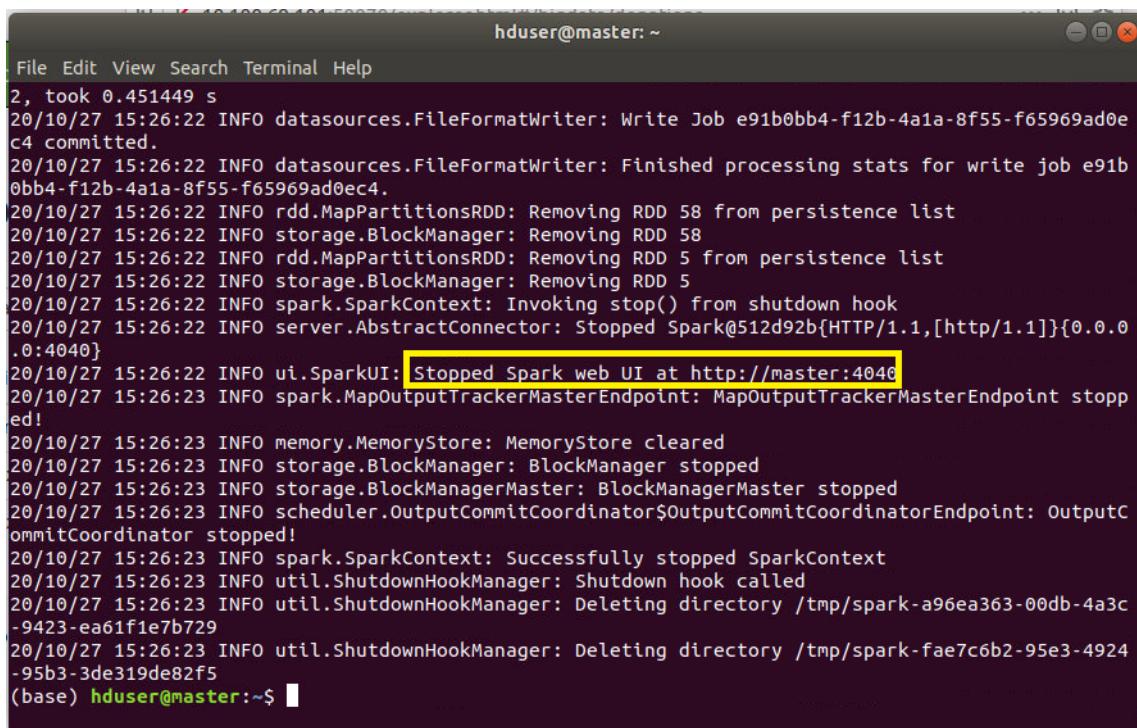
22 Langkah ketiga adalah melakukan eksekusi perangkat lunak pengujian menggunakan perintah 23 Spark. Listing 5.6 adalah contoh perintah eksekusi pada Spark. Format `--class ExaminationModel.` 24 `MainExamination` artinya menunjukkan `Main class` yang dieksekusi bernama `MainExamination` 25 pada package `ExaminationModel`. Format `--master yarn` menunjukkan bahwa program dieksekusi 26 pada sebuah *cluster* komputer. Format `/home/hduser/skripsi-stephen/skripsi.jar` menunjuk-

- 1 an lokasi JAR pada komputer. Format `/skripsi-jordan/InputKMeans.json` menunjukan lokasi
 2 JSON pada HDFS untuk pemodelan *k-means* dan format `/skripsi-jordan/InputNaiveBayes.`
 3 `json` menunjukan lokasi JSON pada HDFS untuk pemodelan *naive bayes*.

Listing 5.6: Perintah Eksekusi Spark

```
4 spark-submit --class ExaminationModel.MainExamination --master yarn /home/hduser/
5   skripsi-stephen/skripsi.jar /skripsi-jordan/InputExamination.json
6
```

- 8 Langkah terakhir adalah menunggu proses komputasi sampai dengan selesai. Perangkat lunak
 9 pengujian yang telah berhasil menyelesaikan proses komputasinya pada terminal Ubuntu ditandai
 10 dengan baris *log* seperti Gambar 5.31: 'Stopped Spark Web UI at <http://master:4040>'. Sete-
 11 lah proses komputasi selesai, maka hasil pengujian *k-means* disimpan sebagai format CSV pada
 12 lokasi HDFS berikut `/skripsi-jordan/output/k-means`, sedangkan hasil pemodelan *naive bayes*
 13 disimpan sebagai format CSV pada lokasi HDFS berikut `/skripsi-jordan/output/naive-bayes`.



```
hduser@master: ~
File Edit View Search Terminal Help
2, took 0.451449 s
20/10/27 15:26:22 INFO datasources.FileFormatWriter: Write Job e91b0bb4-f12b-4a1a-8f55-f65969ad0e
c4 committed.
20/10/27 15:26:22 INFO datasources.FileFormatWriter: Finished processing stats for write job e91b
0bb4-f12b-4a1a-8f55-f65969ad0e4.
20/10/27 15:26:22 INFO rdd.MapPartitionsRDD: Removing RDD 58 from persistence list
20/10/27 15:26:22 INFO storage.BlockManager: Removing RDD 58
20/10/27 15:26:22 INFO rdd.MapPartitionsRDD: Removing RDD 5 from persistence list
20/10/27 15:26:22 INFO storage.BlockManager: Removing RDD 5
20/10/27 15:26:22 INFO spark.SparkContext: Invoking stop() from shutdown hook
20/10/27 15:26:22 INFO server.AbstractConnector: Stopped Spark@512d92b{HTTP/1.1,[http/1.1]}{0.0.0
.0:4040}
20/10/27 15:26:22 INFO ui.SparkUI: Stopped Spark web UI at http://master:4040
20/10/27 15:26:23 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stop
ped!
20/10/27 15:26:23 INFO memory.MemoryStore: MemoryStore cleared
20/10/27 15:26:23 INFO storage.BlockManager: BlockManager stopped
20/10/27 15:26:23 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
20/10/27 15:26:23 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputC
ommitCoordinator stopped!
20/10/27 15:26:23 INFO spark.SparkContext: Successfully stopped SparkContext
20/10/27 15:26:23 INFO util.ShutdownHookManager: Shutdown hook called
20/10/27 15:26:23 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-a96ea363-00db-4a3c
-9423-ea61f1e7b729
20/10/27 15:26:23 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-fae7c6b2-95e3-4924
-95b3-3de319de82f5
(base) hduser@master:~$
```

Gambar 5.31: Log Perangkat Lunak Pengujian

- 14 Ketika lokasi output pada HDFS dibuka, maka akan tampak seperti Gambar 5.32, 5.33. Hasil eks-
 15 plorasi nilai unik akan digunakan sebagai referensi mengisi nilai `domain_generalization_hierarchy`
 16 sebuah atribut tabel data pada `InputAnonymization.json`. Pengujian perlu dilakukan untuk men-
 17 cari perbedaan kualitas hasil *data mining* sebelum dan setelah dilakukan anonimisasi. Gambar 5.32
 18 menujukan lokasi HDFS yang menyimpan hasil pengolompokan data pada CSV `/skripsi-jordan/`
 19 `output/k-means/part-000X-YYYYY.csv`. Gambar 5.33 menujukan lokasi HDFS yang menyimpan
 20 hasil klasifikasi data pada CSV `/skripsi-jordan/output/naive-bayes/part-000X-YYYYY.csv`.

Browse Directory							
/skripsi-jordan/output/k-means							Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	0 B	10/27/2020, 9:43:29 PM	3	128 MB	_SUCCESS
-rw-r--r--	hduser	hduser	2.49 MB	10/27/2020, 9:43:29 PM	3	128 MB	part-00000-a387a349-bc5b-426e-aa01-f4c7faaa9951-c000.csv

Gambar 5.32: Folder HDFS Hasil Pengelompokan K-Means

Browse Directory							
/skripsi-jordan/output/naive-bayes							Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	hduser	0 B	10/27/2020, 9:44:30 PM	3	128 MB	_SUCCESS
-rw-r--r--	hduser	hduser	814.79 KB	10/27/2020, 9:44:30 PM	3	128 MB	part-00000-c76ed676-c4ba-4e82-95c3-ef00aaa3606c-c000.csv

Gambar 5.33: Folder HDFS Hasil Klasifikasi Naive Bayes

5.2 Pengujian

Pengujian pada penelitian ini dibagi menjadi dua tahap, yaitu pengujian fungsional dan pengujian eksperimental. Pengujian fungsional bertujuan untuk melihat apakah fungsi-fungsi pada perangkat lunak sudah menghasilkan keluaran yang sesuai. Pengujian eksperimental bertujuan untuk melihat pengaruh data yang telah dianonimisasi pada proses *data mining*.

5.2.1 Pengujian Fungsional

Pengujian fungsional bertujuan untuk memastikan fungsionalitas dari perangkat lunak anonimisasi dan pengujian. Tahapan pengujian fungsional antara lain melakukan pemeriksaan output pengelompokan data (*greedy k-member clustering*), anonimisasi data (*k-anonymity*), pemodelan *data mining* dengan model *clustering* (*k-means*) dan model klasifikasi (*naive bayes*).

Data yang dipakai adalah data *credit score*, seperti yang sudah dijelaskan pada subbab 3.2.3. Data ini dipilih karena menggunakan informasi personal dari pendaftar kartu kredit. Data ini digunakan oleh industri finansial untuk memprediksi kemungkinan seseorang gagal bayar di masa depan dari pinjaman kartu kredit. Data ini digunakan pada proses pengelompokan data, anonimisasi data, dan pemodelan *data mining* dengan *k-means* dan *naive bayes*.

Listing 5.7: Sampel Data Credit Score

```

1 ID,DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,
2   AMT_INCOME_TOTAL
3 5008801,-19110,-3051,Sales staff,F,Commercial associate,270000
4 5008802,-19110,-3051,Sales staff,F,Commercial associate,270000
5 5008803,-21474,-1134,Security staff,M,Working,112500
6 5008804,-19110,-3051,Sales staff,F,Commercial associate,270000
7 5008805,-12005,-4542,"",M,Working,427500
8 5008806,-12005,-4542,"",M,Working,427500
9 5008807,-19110,-3051,Sales staff,Commercial associate,1
10 5008808,-22464,365243,"",Pensioner,0
11 5008809,-22464,365243,"",Pensioner,0
12
13

```

14 Pengelompokan Data dengan Greedy k-member clustering

15 Pengujian pengelompokan data dilakukan pada perangkat lunak anonimisasi. Pengujian ini meng-
 16 gunakan data set credit score yang berisi data pribadi pemohon kartu kredit untuk memprediksi
 17 kemungkinan gagal bayar dan pinjaman kartu kredit di masa depan. Dataset ini terdiri dari atribut
 18 numerik dan atribut kategorikal. Hasil pengelompokan data sudah disimpan dengan format CSV
 19 untuk mempermudah observasi. Listing 5.8 adalah contoh sampel data yang diambil dari data
 20 *credit score* yang terdiri dari 18 jenis atribut.

Listing 5.8: Sampel Data Credit Score

```

21 ID,DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,
22   AMT_INCOME_TOTAL
23 5008801,-19110,-3051,Sales staff,F,Commercial associate,270000
24 5008802,-19110,-3051,Sales staff,F,Commercial associate,270000
25 5008803,-21474,-1134,Security staff,M,Working,112500
26 5008804,-19110,-3051,Sales staff,F,Commercial associate,270000
27 5008805,-12005,-4542,"",M,Working,427500
28 5008806,-12005,-4542,"",M,Working,427500
29 5008807,-19110,-3051,Sales staff,Commercial associate,1
30 5008808,-22464,365243,"",Pensioner,0
31 5008809,-22464,365243,"",Pensioner,0
32 5008810,-22464,365243,"",Pensioner,0
33
34

```

35 Hasil yang ingin dicapai dari pengujian ini adalah perangkat lunak dapat mengeluarkan hasil
 36 pengelompokan data menggunakan algoritma *greedy k-member clustering*. Setiap baris pada data
 37 yang memiliki hubungan yang dekat dengan baris data lainnya telah dikelompokan ke dalam satu
 38 cluster dan jumlah anggota sebuah *cluster* bergantung pada nilai *k* yang ditetapkan sebelumnya.
 39 Selain itu, hasil pengelompokan data dan perhitungan *distance record* telah sesuai dengan perhi-
 40 tungan manual yang dilakukan pada studi kasus. Pada pengujian ini, nilai *k* yang digunakan adalah

- 1 2 sehingga masing-masing *cluster* yang terbentuk minimal terdiri dari 2 data . Listing 5.9 adalah
 2 hasil pengelompokan data dengan algoritma *greedy k-member clustering*.

Listing 5.9: Hasil Pengelompokan Greedy K-Member Clustering

```

3 id,DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,
4   AMT_INCOME_TOTAL,Cluster
5 2,-12005,-4542,"",M,Working,427500,Cluster 3
6 8,-22464,365243,"",F,Pensioner,283500,Cluster 3
7 5,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 1
8 1,-12005,-4542,"",M,Working,427500,Cluster 1
9 7,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 1
10 9,-22464,365243,"",F,Pensioner,283500,Cluster 3
11 3,-21474,-1134,Security staff,M,Working,112500,Cluster 2
12 4,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 2
13 10,-22464,365243,"",F,Pensioner,283500,Cluster 2
14 6,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 2
15
16

```

- 17 Berdasarkan hasil pengujian yang didapat, dapat disimpulkan bahwa perangkat lunak anonimisasi
 18 dapat melakukan perhitungan *infomation loss* dengan benar. Hal ini didasari dari subbab 3.2.5
 19 bahwa pengelompokan data dengan *information loss* yang kecil berpotensi tinggi menjadi satu
 20 kelompok dan telah dibuktikan pada Gambar 5.9 bahwa data-data yang memiliki nilai numerik
 21 atau kategorikal yang berdekatan telah dikelompokan ke dalam *clusteryang sama*. Hal ini terjadi
 22 karena data-data dengan nilai yang berdekatan memiliki bobot *information loss* yang relatif lebih
 23 kecil. Hasil pengelompokan data yang didapat dari program anonimisasi telah sesuai dengan hasil
 24 pengelompokan data yang dilakukan sebelumnya pada studi kasus subbab 3.2.6.

25 Anonimisasi Data dengan Metode K-Anonymity

- 26 Pengujian ini dilakukan pada perangkat lunak anonimisasi. Pengujian anomisasi data memakai
 27 input dari hasil pengelompokan data pada Listing 5.10, yang berisi kumpulan data-data pada *cluster*
 28 tertentu. Dataset ini terdiri dari atribut numerik dan atribut kategorikal. Pengelompokan data
 29 telah dilakukan sebelum melakukan tahap anonimisasi, sehingga diharapkan nilai informasi yang
 30 didapat masih cukup baik karena proses anonimisasi data dilakukan per masing-masing kelompok
 31 data. Hasil anonimisasi data telah disimpan dalam CSV untuk dilakukan observasi.

Listing 5.10: Sampel Pengelompokan Data

```

32 id,DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,
33   AMT_INCOME_TOTAL,Cluster
34 2,-12005,-4542,"",M,Working,427500,Cluster 3
35 8,-22464,365243,"",F,Pensioner,283500,Cluster 3
36 5,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 1
37 1,-12005,-4542,"",M,Working,427500,Cluster 1
38 7,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 1
39 9,-22464,365243,"",F,Pensioner,283500,Cluster 3
40

```

1	3,-21474,-1134,Security staff,M,Working,112500,Cluster 2
2	4,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 2
3	10,-22464,365243,"",F,Pensioner,283500,Cluster 2
4	6,-19110,-3051,Sales staff,F,Commercial associate,270000,Cluster 2

6 Hasil yang ingin dicapai dari pengujian ini adalah perangkat lunak dapat mengeluarkan hasil
7 anonimisasi data menggunakan metode *k-anonymity*. Dimana setiap baris pada data tidak dapat
8 dibedakan dengan $k - 1$ baris data lainnya, sehingga untuk mencapai pernyataan tersebut, maka
9 proses anonimisasi data dilakukan per kelompok data/*cluster*. Oleh karena itu, hasil anonimisasi
10 data antara kelompok data yang satu dengan yang kelompok data lainnya saling berbeda satu sama
11 lain. Selain itu, hasil pengujian anonimisasi data dan perhitungan *information loss* sudah sesuai
12 dengan perhitungan manual yang dilakukan pada studi kasus di Subbab 3.2.5. Pada pengujian ini,
13 nilai k yang digunakan adalah 2 sehingga masing-masing *cluster* yang terbentuk minimal terdiri
14 dari 2 data. Listing 5.11 adalah hasil anonimisasi data dengan metode *k-anonymity*.

Listing 5.11: Hasil Anonimisasi K-Anonymity

15	
16	<code>id,Anonym_DAYS_BIRTH,Anonym_DAYS_EMPLOYED,Anonym_OCCUPATION_TYPE,</code>
17	<code>Anonym_CODE_GENDER,Anonym_NAME_INCOME_TYPE,AMT_INCOME_TOTAL</code>
18	<code>1, [(-19110)-(-12005)], [(-4542)-(-3051)], Sales staff, Adult, Unique earnings, 427500</code>
19	<code>2, [(-21474)-(-12005)], [(-4542)-(-1134)], Employee, Adult, Unique earnings, 427500</code>
20	<code>3, [(-21474)-(-12005)], [(-4542)-(-1134)], Employee, Adult, Unique earnings, 112500</code>
21	<code>4, [(-21474)-(-12005)], [(-4542)-(-1134)], Employee, Adult, Unique earnings, 270000</code>
22	<code>5, [(-19110)-(-12005)], [(-4542)-(-3051)], Sales staff, Adult, Unique earnings, 270000</code>
23	<code>6, [(-21474)-(-12005)], [(-4542)-(-1134)], Employee, Adult, Unique earnings, 270000</code>
24	<code>7, [(-19110)-(-12005)], [(-4542)-(-3051)], Sales staff, Adult, Unique earnings, 270000</code>
25	<code>8,-22464,365243,"",F,Pensioner,283500</code>
26	<code>9,-22464,365243,"",F,Pensioner,283500</code>
27	<code>10,-22464,365243,"",F,Pensioner,283500</code>

29 Berdasarkan hasil anonimisasi data yang didapat, dapat disimpulkan bahwa perangkat lunak
30 anonimisasi sudah dapat melakukan proses anonimisasi data menggunakan metode *k-anonymity*.
31 Hal ini dibuktikan bahwa masing-masing nilai atribut telah dilakukan generalisasi ke nilai yang lebih
32 umum. Selain itu, hasil dari pengelompokan data juga sudah sesuai dengan konsep anonimisasi
33 data, karena setiap baris pada data tidak dapat dibedakan dengan $k - 1$ baris data lainnya. Jika
34 dilihat lebih seksama, nilai-nilai data pada Listing 5.11 terlihat mirip satu sama lain, sehingga
35 tujuan metode *k-anonymity* dapat dikatakan tercapai untuk hasil anonimisasi data.

36 Pembuatan Model Data Mining dengan K-Means

37 Pengujian ini dilakukan pada perangkat lunak pengujian. Pengujian pemodelan *k-means* data
38 dilakukan dengan memakai input sebelum dan setelah data dilakukan proses anonimisasi. Pengujian
39 ini dilakukan untuk membandingkan hasil pengelompokan data sebelum dan setelah dilakukan
40 anonimisasi. Selanjutnya, dilakukan perbandingan selisih *silhouette score* antara kedua model data

1 tersebut, untuk mengetahui jumlah informasi yang hilang selama data dilakukan anonimisasi. Hasil
 2 pemodelan *k-means* disimpan dalam CSV untuk dilakukan observasi lebih lanjut. Listing 5.14
 3 adalah contoh sampel data dari hasil pengelompokan data *k-means* sebelum anonimisasi.

4 Hasil yang ingin dicapai dari pengujian ini adalah perangkat lunak dapat mengeluarkan hasil
 5 pengelompokan data menggunakan model *k-means* dan menampilkan hasil evaluasi model menggu-
 6 nakan *silhouette score*. Dimana semakin kecil nilai selisih *silhouette score* antara kedua model maka
 7 model yang dibuat semakin mendekati informasi pada data yang belum dianonimisasi. Oleh karena
 8 itu, perlu dilakukan pengujian eksperimental untuk mencari model *k-means* terbaik dengan mencari
 9 nilai *k* terbaik. Pada pengujian ini, nilai *k* yang digunakan adalah 2 sehingga masing-masing *cluster*
 10 yang terbentuk, minimal terdiri dari 2 data. Listing 5.12 adalah hasil pengelompokan data dengan
 11 model *k-means*. Listing 5.13 adalah hasil *silhouette score* untuk model *k-means*. Listing 5.14 adalah
 12 persentase perbedaan hasil *clustering* sebelum dan setelah anonimisasi

Listing 5.12: Hasil Pengelompokan K-Means Sebelum Anonimisasi

```
13
14 DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,NAME_INCOME_TYPE,prediction
15 -12005,-4542,"",Working,0
16 -12005,-4542,"",Working,0
17 -21474,-1134,Security staff,Working,2
18 -19110,-3051,Sales staff,Commercial associate,1
19 -19110,-3051,Sales staff,Commercial associate,1
20 -19110,-3051,Sales staff,Commercial associate,1
21 -19110,-3051,Sales staff,Commercial associate,1
22 -22464,365243,"",Pensioner,0
23
```

Listing 5.13: Hasil Pengelompokan K-Means Setelah Anonimisasi

```
24
25 Anonym_DAYS_BIRTH,Anonym_DAYS_EMPLOYED,Anonym_OCCUPATION_TYPE,
26     Anonym_NAME_INCOME_TYPE,prediction
27 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
28 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
29 [(-22464)-(-19110)],[(-3051)-365243],Employee,Unique earnings,1
30 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
31 [(-22464)-(-19110)],[(-3051)-365243],Employee,Unique earnings,1
32 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
33 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
34 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
35 [(-22464)-(-12005)],[(-4542)-365243],Sales staff,Unique earnings,0
36
```

Listing 5.14: Perbedaan Silhouette Score

```
37
38 Info,Silhouette score
39 Normal table,0.7857142857142856
40 Anonymize table,1.0
41 How much different is Silhouette score,0.2142857142857144
42
```

Listing 5.15: Perbedaan Persentase Hasil Pengelompokan (%)

```

1 Percentage of Clustering Difference
2 0.5
3
4

```

Berdasarkan hasil pengelompokan data yang didapat, dapat disimpulkan bahwa perangkat lunak pengujian sudah dapat melakukan proses pengelompokan data menggunakan pemodelan *k-means*. Hal ini dibuktikan bahwa atribut prediktor berhasil menghasilkan kolom ***prediction*** yang merupakan hasil pengelompokan data. Selain itu, hasil dari pengelompokan data juga sudah sesuai dengan konsep penerapan *k-means* pada Subbab 2.2.1, karena setiap baris data yang mirip telah berhasil dikelompokan ke masing-masing *cluster* yang terbentuk. Jika dilihat lebih seksama, perbedaan *silhouette score* sebelum dan setelah anonimisasi pada Listing ?? cukup kecil, sehingga tujuan pemodelan *k-means* dapat dikatakan tercapai untuk anonimisasi data.

13 Pencarian Model Data Mining Terbaik dengan Naive Bayes

Pengujian ini dilakukan pada perangkat lunak pengujian. Pengujian pemodelan *naive bayes* dilakukan dengan memakai input sebelum dan setelah data dilakukan proses anonimisasi. Pengujian ini dilakukan untuk membandingkan hasil klasifikasi data sebelum dan setelah dilakukan anonimisasi. Selanjutnya, dilakukan perbandingan selisih tingkat akurasi antara kedua model data tersebut, untuk mengetahui jumlah informasi yang hilang selama data dilakukan anonimisasi. Hasil pemodelan *naive bayes* disimpan dalam CSV untuk dilakukan observasi.

Hasil yang ingin dicapai dari pengujian ini adalah perangkat lunak dapat mengeluarkan hasil pengelompokan data menggunakan model *naive bayes* dan menampilkan hasil evaluasi model menggunakan tingkat akurasi. Dimana semakin kecil selisih tingkat akurasi antara kedua model, maka model yang dibuat semakin mendekati informasi pada data yang belum dianonimisasi. Oleh karena itu, semakin banyak data yang digunakan untuk pembuatan model, maka diharapkan akurasinya semakin baik. Pada pengujian ini, data akan dibagi menjadi 70% data *training* dan 30% data *testing*. Listing 5.16, 5.17 adalah hasil klasifikasi data dengan model *naive bayes*. Listing 5.18 adalah selisih tingkat akurasi untuk model naive bayes.

Listing 5.16: Hasil Klasifikasi Naive Bayes Sebelum Anonimisasi

```

28
29 id,DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,NAME_INCOME_TYPE,prediction
30 1,-12005,-4542,"",Working,0.0
31 2,-12005,-4542,"",Working,0.0
32 3,-21474,-1134,Security staff,Working,1.0
33 4,-19110,-3051,Sales staff,Commercial associate,0.0
34 5,-19110,-3051,Sales staff,Commercial associate,0.0
35 6,-19110,-3051,Sales staff,Commercial associate,0.0
36 7,-19110,-3051,Sales staff,Commercial associate,0.0
37 8,-22464,365243,"",Pensioner,0.0
38 9,-22464,365243,"",Pensioner,0.0
39 10,-22464,365243,"",Pensioner,0.0
40

```

Listing 5.17: Hasil Klasifikasi Naive Bayes Setelah Anonimisasi

```

1 id,Anonym_DAYS_BIRTH,Anonym_DAYS_EMPLOYED,Anonym_OCCUPATION_TYPE,
2     Anonym_NAME_INCOME_TYPE,prediction
3 1, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
4 2, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
5 3, [(-22464)-(-19110)], [(-3051)-365243], Employee, Unique earnings, 1.0
6 4, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
7 5, [(-22464)-(-19110)], [(-3051)-365243], Employee, Unique earnings, 1.0
8 6, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
9 7, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
10 8, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
11 9, [(-22464)-(-12005)], [(-4542)-365243], Sales staff, Unique earnings, 0.0
12 10, [(-22464)-(-19110)], [(-3051)-365243], Employee, Unique earnings, 1.0
13
14

```

Listing 5.18: Perbedaan Tingkat Akurasi

```

15 Info,Accuracy
16 Normal table,0.70
17 Anonymize table,0.50
18 How much different is accuracy,0.20
19
20

```

Listing 5.19: Perbedaan Persentase Hasil Klasifikasi (%)

```

21 Percentage of Classification Difference
22 0.52
23
24

```

25 Berdasarkan hasil klasifikasi data yang didapat, dapat disimpulkan bahwa perangkat lunak pengujian
26 sudah dapat melakukan proses klasifikasi data menggunakan model *naive bayes*. Hal ini dibuktikan
27 bahwa atribut prediktor berhasil menghasilkan kolom *prediction* yang merupakan hasil klasifikasi
28 data. Selain itu, hasil dari klasifikasi data juga sudah sesuai dengan konsep *naive bayes* pada Subbab
29 [2.2.1](#), karena setiap baris pada data berhasil diprediksi nilai labelnya. Jika dilihat lebih seksama,
30 perbedaan tingkat akurasi sebelum dan setelah anonimisasi pada Listing [5.18](#) cukup kecil, sehingga
31 tujuan pemodelan klasifikasi dapat dikatakan tercapai untuk anonimisasi data.

32 5.2.2 Pengujian Eksperimental

33 Pengujian eksperimental bertujuan untuk menguji performa dari algoritma *greedy k-member clustering* dan *k-anonymity* di lingkungan *big data* pada *hadoop cluster*, menguji performa pemodelan
34 *data mining clustering (k-means)* dan klasifikasi (*naive bayes*) di lingkungan *big data* pada komputer
35 lokal, kualitas informasi pada data yang telah dikelompokan dengan algoritma *greedy k-member clustering* berdasarkan *total information loss*, melakukan analisis kualitas hasil *data mining* terhadap
36 pemodelan *clustering* dan klasifikasi sebelum dan setelah dilakukan anonimisasi. Hasil pengujian
37 ini dapat digunakan sebagai referensi untuk penelitian selanjutnya.

1 Lingkungan untuk menguji perangkat lunak anonimisasi adalah sistem terdistribusi pada *hadoop*
 2 *cluster* lab menggunakan 10 *slaves node*, dengan spesifikasi masing-masing *slaves node* antara lain
 3 prosesor Intel Core i3 CPU 550 @3.20GHz x4, 8GB RAM, Spark 2.4.3, Hadoop 2.7.1, dan
 4 Scala 2.11.12. Lingkungan untuk menguji perangkat lunak eksplorasi dan pengujian adalah
 5 laptop dengan spesifikasi antara lain prosesor Intel(R) Core(TM) i7-4720HQ @ 2.6 GHz, 4GB
 6 RAM, Java(TM) JDK 8, Spark 2.4.3, Hadoop 3.1.2, dan Scala 2.11.12.

7 Untuk melakukan pengujian eksperimental, digunakan data *credit score* yang didapat dari Kaggle.
 8 Data *credit score* adalah metode pengendalian risiko yang umum di industri keuangan. Data
 9 ini menggunakan informasi pribadi dari pemohon kartu kredit untuk memprediksi kemungkinan
 10 gagal bayar dan pinjaman kartu kredit di masa depan. Data *credit score* digunakan sebagai histori
 11 agar bank dapat memutuskan apakah kartu kredit diberikan/tidak kepada pemohon. Skor kredit
 12 dapat mengukur secara objektif besarnya risiko bank memberikan pinjaman kredit.

13

14 Tabel 5.1, 5.2 adalah jenis pengujian yang telah dianalisis yaitu pengujian kualitas informasi dan
 15 pengujian hasil data mining. Tabel ini menjelaskan garis besar kajian-kajian yang telah diuji.

Tabel 5.1: Pengujian Kualitas Informasi

Tahapan Pengujian Kualitas Informasi	Kajian
1. Pengaruh jenis kolom terhadap information loss <ul style="list-style-type: none"> • Waktu anonimisasi • Waktu pengelompokan • Total Information Loss 	numerik,kategorikal,campuran numerik,kategorikal,campuran numerik,kategorikal,campuran
2. Pengaruh jumlah quasi-identifier terhadap information loss <ul style="list-style-type: none"> • Waktu anonimisasi • Waktu pengelompokan • Total Information Loss 	1-QID,2-QID,3-QID 1-QID,2-QID,3-QID 1-QID,2-QID,3-QID
3. Pengaruh ukuran data terhadap information loss <ul style="list-style-type: none"> • Waktu anonimisasi • Waktu pengelompokan • Total Information Loss 	10.000 data, 30.000 data 10.000 data, 30.000 data 10.000 data, 30.000 data

Tabel 5.2: Pengujian Hasil Data Mining

Tahapan Pengujian Data Mining	Kajian
1. Pencarian model k-means terbaik untuk pengelompokan <ul style="list-style-type: none"> • Silhouette score • Waktu komputasi • Perbedaan hasil clustering (%) • Perbedaan hasil clustering (%) 	data sebelum, setelah anonimisasi data sebelum, setelah anonimisasi ($k = 25, n = 1000$), ($k = 100, n = 1000$) ($k = 100, n = 100$), ($k = 100, n = 1000$)
2. Pencarian model naive bayes terbaik untuk klasifikasi <ul style="list-style-type: none"> • Tingkat akurasi • Waktu komputasi • Perbedaan hasil klasifikasi (%) 	data sebelum, setelah anonimisasi data sebelum, setelah anonimisasi ($n = 100$), ($n = 1000$)

1 Tabel 5.3 adalah konfigurasi yang dipakai pada pengujian *total information loss*:

Tabel 5.3: Konfigurasi Pengujian

#	Eksperimen	Parameter	Dataset
1	<i>column</i>	$n = 100$, $QID = 2$, $column \in$ numerik, kategorikal, campuran, $k\text{-value} \in 25, 50, 75, 100$	<i>Credit score</i>
2	<i>QID</i>	$n = 100$, $QID \in [1..5]$, $k\text{-value} \in 25, 50, 75, 100$	<i>Credit score</i>
4	<i>size</i>	$n \in 10.000, 30.000$, $QID = 5$, $k\text{-value} \in 25, 50, 75, 100$	Credit score

2 Keterangan:

- 3 • $k\text{-value}$: nilai k pada *greedy k-member clustering* dan *k-anonymity*
- 4 • *column*: jenis kolom (numerik, kategorikal, numerik & kategorikal)
- 5 • *QID*: jumlah atribut *quasi-identifier*.
- 6 • *size*: jumlah data yang dipakai.

7 Tabel 5.4 adalah konfigurasi yang dipakai pada pengujian hasil *data mining*:

Tabel 5.4: Konfigurasi Pengujian

#	Eksperimen	Parameter	Dataset
1	k-means	$n \in 100, 1k$, $k = 25, 100$, $k\text{-value} \in 100$,	<i>Credit score</i> sebelum dan setelah anonimisasi
2	naive bayes	$n \in 100, 1000$, <i>label</i> = income, <i>trainset</i> = 0.7, <i>testset</i> = 0.3, $k\text{-value} \in 100$,	<i>Credit score</i> sebelum dan setelah anonimisasi

8 Keterangan:

- 9 • k : nilai k pada model *k-means*.
- 10 • *label*: nilai prediksi pada model *naive bayes*.
- 11 • $k\text{-value}$: nilai k pada *greedy k-member cluster* dan *k-anonymity*

1 Pengujian Eksperimental Total Information Loss

2 Tabel 5.5,5.6,5.7 adalah hasil setiap jenis pengujian eksperimental terhadap *total information loss*:

4 Jenis kolom bervariasi

6 Berikut adalah contoh sampel pengujian berdasarkan pemilihan kolom numerik, kategorikal, kategorikal dan numerik dari data *credit score*.

Tabel 5.5: Sampel Data Credit Score(Numerik)

DAY_S_BIRTH	DAY_S_EMPLOYED
-12005	-4542
-22464	365243

Tabel 5.6: Sampel Data Credit Score(Kategorikal)

OCCUPATION	GENDER
Sales staff	M
Security staff	F

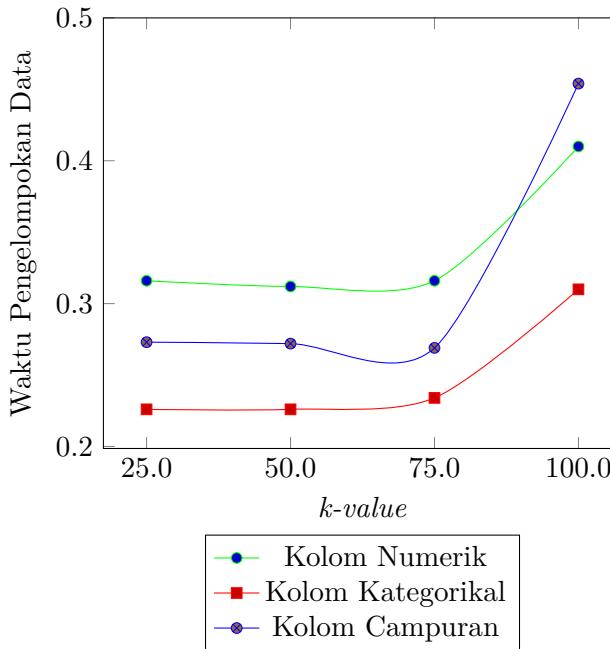
Tabel 5.7: Sampel Data Credit Score(Campuran)

DAY_S_BIRTH	OCCUPATION
-12005	Sales staff
-22464	Security staff

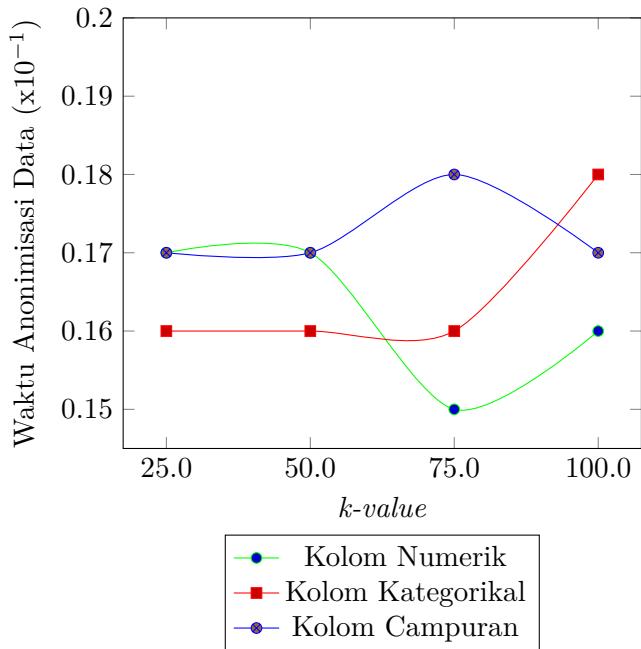
7 Pengujian ini ingin dibuktikan apakah hasil pengelompokan *greedy k-member clustering* dan anonimisasi *k-anonymity* dapat diterima jika jenis kolom bervariasi, karena umumnya data terdiri dari kolom numerik dan kategorikal. Pengujian ini menggunakan $QID \in [2]$ dengan pengujian pada 2 atribut numerik, 2 atribut kategorikal, dan 2 atribut campuran (numerik dan kategorikal), $k \in [25,50,75,100]$, dan $n = 100$. Berikut penjelasan jenis kajian yang diuji terhadap jumlah QID bervariasi:

- 9 • Waktu Pengelompokan.** Pengamatan ini dilakukan pada algoritma *greedy k-member 10 clustering*. Gambar 5.34 menunjukkan waktu pengelompokan data mengalami peningkatan 11 signifikan pada bobot *k-value* = 100. Dapat ditarik kesimpulan bahwa pengelompokan data 12 dapat dilakukan lebih cepat jika menggunakan kolom kategorikal dengan pembobotan *k-value* 13 kurang dari 100.
- 14 • Waktu Anonimisasi.** Pengamatan ini dilakukan pada algoritma *k-anonymity*. Gambar 5.35 15 menunjukkan waktu anonimisasi yang hampir mirip untuk setiap jenis kolom. Hal ini dapat 16 dilihat dari rentang waktu anonimisasi yang cukup sempit (0.15 – 0.18) detik. Berdasarkan 17 hasil pengujian ini, dapat ditarik kesimpulan bahwa setiap kolom tidak memberikan pengaruh 18 signifikan terhadap penambahan waktu anonimisasi. Proses anonimisasi memiliki waktu 19 komputasi yang lebih singkat dibandingkan pengelompokan data.

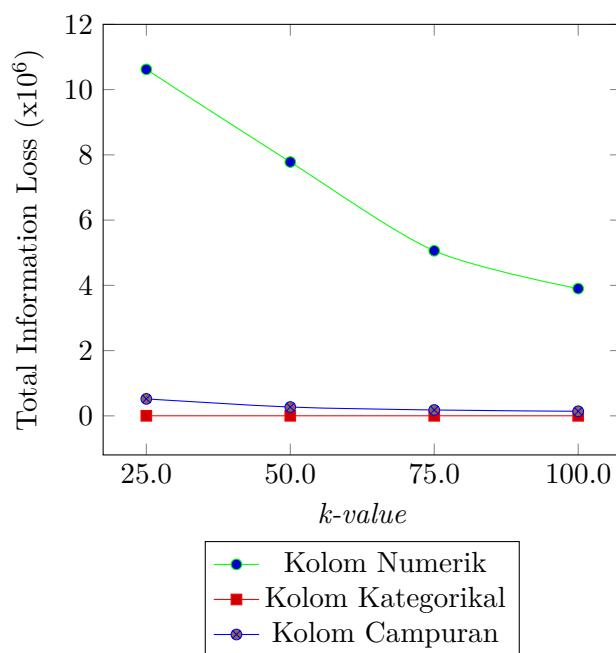
- Total Information Loss.** Gambar 5.36 menunjukkan *total information loss* pada kolom numerik lebih besar dibandingkan kolom campuran dan kategorikal, karena kolom numerik menghitung *information loss* berdasarkan perbedaan dua nilai numerik, sehingga menghasilkan bobot lebih besar dibandingkan kolom kategorikal. Melalui hasil pengujian ini, dapat ditarik kesimpulan bahwa pengelompokan data dapat menghasilkan kualitas pengelompokan yang lebih baik jika *total information loss* yang dihasilkan kecil dengan menggunakan kolom campuran dan pembobotan *k-value* yang lebih besar (*k-value* = 100).



Gambar 5.34: Perubahan Waktu Pengelompokan Data



Gambar 5.35: Perubahan Waktu Anonimisasi Data



Gambar 5.36: Perubahan Total Information Loss

1 Berdasarkan pengujian jumlah kolom bervariasi diatas, diketahui bahwa hasil pengelompokan
dengan algoritma *greedy k-member clustering* cukup baik jika menggunakan kolom campuran
dengan pembobotan *k-value* = 100, karena memiliki *total information loss* paling kecil dari
pengujian lainnya. Dari segi kecepatan komputasi, waktu pengelompokan data jauh lebih
lama dibandingkan waktu anonomisasi karena beberapa pekerjaan pengelompokan data pada
algoritma *greedy k-member clustering* tidak dapat dipecah secara paralel.

2 **Jumlah QID bervariasi**

3
4 Tabel 5.8, 5.9, 5.10 adalah contoh sampel pengujian berdasarkan pemilihan kolom numerik,
kategorikal, kategorikal & numerik dari data set *credit score*.

Tabel 5.8: Sampel Data Credit score ($|QID|=2$)

DAY_S_BIRTH	OCCUPATION
-12005	Security staff
-22464	Sales staff

Tabel 5.9: Sampel Data Credit score ($|QID|=3$)

DAY_S_BIRTH	OCCUPATION	GENDER
-12005	Security staff	M
-22464	Sales staff	F

Tabel 5.10: Sampel Data Credit score ($|QID|=4$)

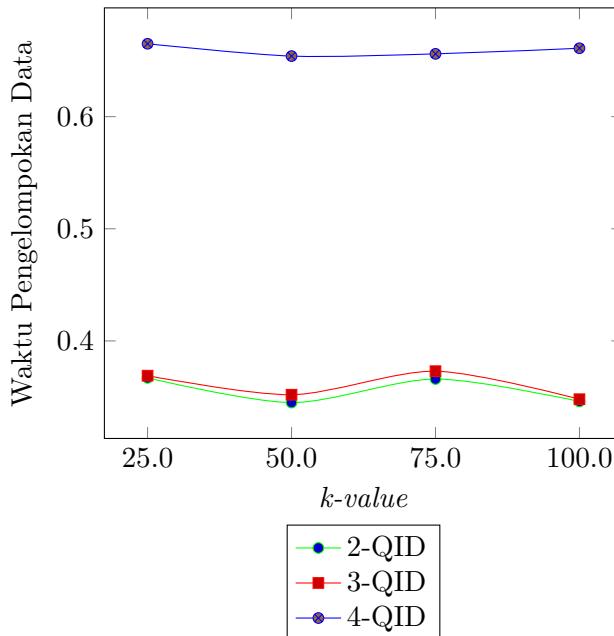
DAY_S_BIRTH	DAY_S_EMPLOYED	OCCUPATION	GENDER
-12005	-4542	Security staff	M
-22464	-4542	Sales staff	F

5 Pada pengujian ini, ingin dibuktikan apakah kualitas pengelompokan dan anonomisasi data
dari algoritma *greedy k-member clustering* dan *k-anonymity* dapat diterima jika jumlah *quasi-
identifier* bervariasi, karena umumnya data yang digunakan memiliki lebih dari 5 kolom.
Pengujian ini menggunakan $QID \in [2..4]$ dengan pengambilan 2 atribut numerik dan 2 atribut
kategorikal, $k \in [25,50,75,100]$ terhadap $n = 100$.

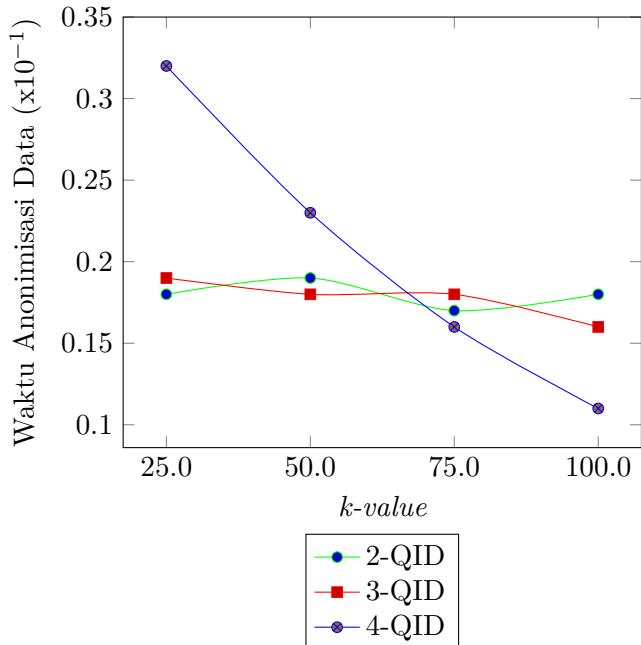
- 6
- 7 • **Waktu Pengelompokan.** Pengamatan ini dilakukan pada algoritma *greedy k-member
clustering*. Gambar 5.38 menunjukkan waktu pengelompokan data mengalami peningkatan
signifikan pada 3-QID dengan bobot *k-value* = 50. Melalui hasil pengujian ini, dapat ditarik
kesimpulan bahwa pengelompokan data dapat dilakukan lebih cepat jika menggunakan sampel
data dengan 2 *quasi-identifier* (2-QID) untuk setiap pembobotan *k-value*.
 - 8
 - 9
 - 10
 - 11
 - 12 • **Waktu Anonimisasi.** Pengamatan ini dilakukan pada algoritma *k-anonymity*. Gambar
5.38 menunjukkan waktu anonomisasi lebih lama pada penggunaan 4-QID. Hal ini dapat
dilihat dari rentang waktu anonomisasi yang cukup besar antara 4-QID dengan 2-QID, 3-QID.
Melalui hasil pengujian ini, dapat ditarik kesimpulan bahwa dengan bertambahnya kolom
numerik pada sampel data dengan 4 quasi-idntifier (4-QID) memberikan pengaruh signifikan
 - 13
 - 14
 - 15
 - 16

terhadap penambahan waktu anonimisasi. Proses anonimisasi juga memiliki waktu komputasi yang jauh lebih singkat dibandingkan pengelompokan data.

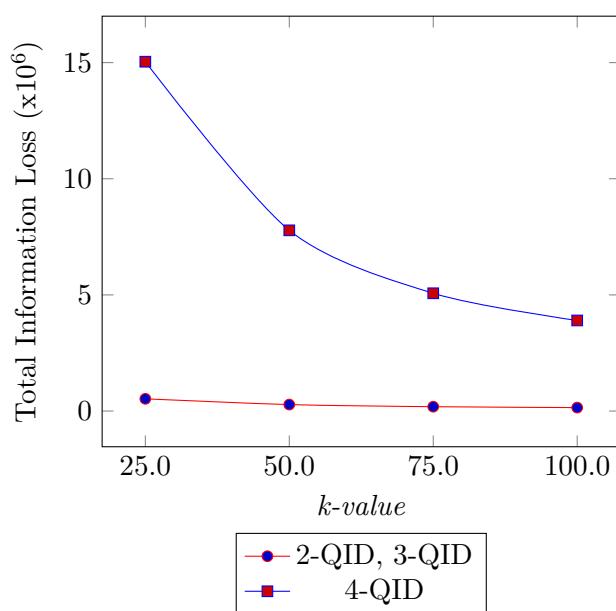
- **Total Information Loss.** Pengamatan ini dilakukan pada algoritma *greedy k-member clustering*. Gambar 5.39 menunjukkan *total information loss* yang dihasilkan oleh 4-QID jauh lebih besar dibandingkan 2-QID dan 3-QID. Melalui hasil pengujian ini, dapat ditarik kesimpulan bahwa pengelompokan data dapat menghasilkan kualitas pengelompokan yang lebih baik jika *total information loss* yang dihasilkan kecil dengan menggunakan 2-QID, 3-OID dengan jumlah kolom numerik = 1 dan pembobotan *k-value* = 100.



Gambar 5.37: Perubahan Waktu Pengelompokan Data



Gambar 5.38: Perubahan Waktu Anonimisasi Data



Gambar 5.39: Perubahan Total Information Loss

Berdasarkan pengujian jumlah QID bervariasi diatas, diketahui bahwa hasil pengelompokan dengan algoritma *greedy k-member clustering* cukup baik jika menggunakan $2\text{-QID}/3\text{-QID}$ dengan pembobotan $k\text{-value} = 100$, karena memiliki *total information loss* paling kecil dari pengujian lainnya. Dari segi kecepatan komputasi, waktu pengelompokan data jauh lebih lama dibandingkan waktu anonimisasi karena beberapa pekerjaan pengelompokan data pada algoritma *greedy k-member clustering* tidak dapat dikerjakan secara paralel.

Ukuran data bervariasi

Tabel 5.11 adalah sampel pengujian dengan memiliki nilai dari kolom numerik dan kolom kategorikal pada data Credit score, dimana *QID* adalah kolom *quasi-identifier* dan *SA* adalah kolom *sensitive attribute* untuk data set *credit score*.

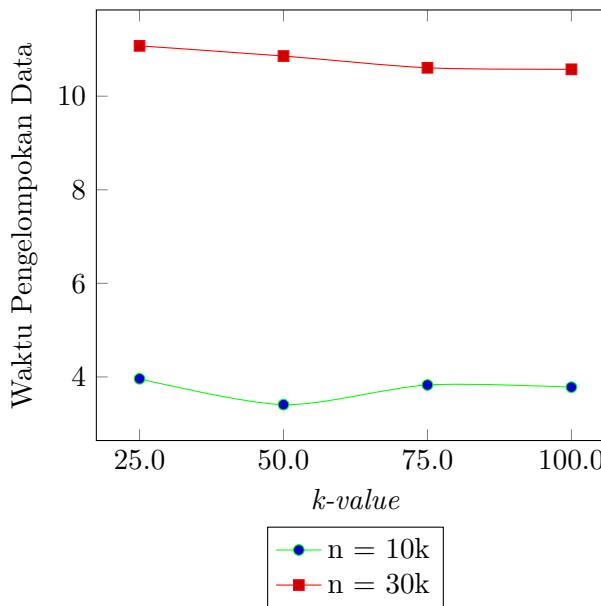
Tabel 5.11: Sampel Data Credit score (10k dan 30k)

Nama Atribut	Tipe Data	Keterangan
DAYS_BIRTH	numerik	QID
DAYS_EMPLOYED	numerik	QID
OCCUPATION_TYPE	kategorikal	QID
CODE_GENDER	kategorikal	QID
NAME_INCOME_TYPE	kategorikal	SA
NAME_EDUCATION_TYPE	kategorikal	QID
NAME_FAMILY_STATUS	kategorikal	QID
NAME_HOUSING_TYPE	kategorikal	QID

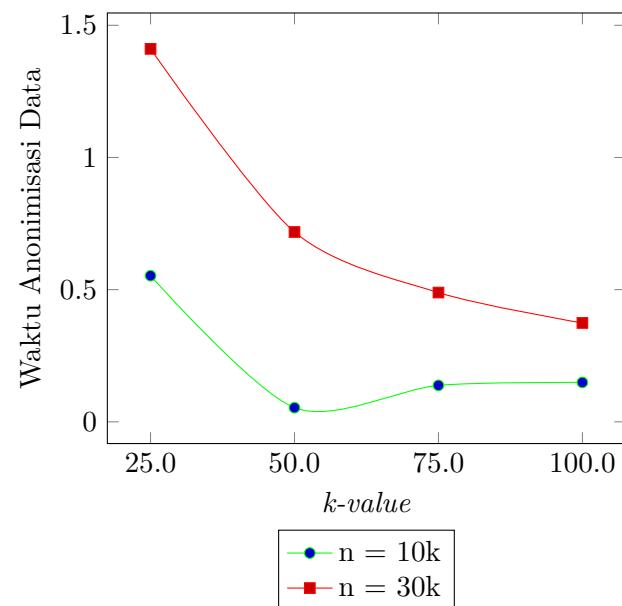
Pada pengujian ini, ingin dibuktikan apakah kualitas pengelompokan dan anonimisasi data dari algoritma *greedy k-member clustering* dan *k-anonymity* dapat diterima jika ukuran data bervariasi. Pengujian ini menggunakan $k \in [25, 50, 75, 100]$ dengan pengambilan atribut secara acak terhadap $n = 10.000$ dan $n = 30.000$. Berikut penjelasan jenis kajian yang diuji:

- **Waktu Pengelompokan.** Pengamatan ini dilakukan pada algoritma *greedy k-member clustering*. Gambar 5.40 menunjukkan waktu pengelompokan data dengan $n = 30.000$ membutuhkan waktu dua kali lebih lama dibandingkan pengelompokan data dengan $n = 10.000$, dengan total waktu pengelompokan data sekitar 10 jam. Selain itu, waktu pengelompokan data juga cenderung stabil seiring bertambahnya nilai $k\text{-value}$. Melalui hasil pengujian ini, dapat ditarik kesimpulan bahwa pengelompokan data dapat dilakukan lebih cepat jika menggunakan ukuran data yang relatif lebih kecil, yaitu $n = 10.000$ untuk setiap pembobotan $k\text{-value}$.
- **Waktu Anonimisasi.** Pengamatan ini dilakukan pada algoritma *k-anonymity*. Gambar 5.41 menunjukkan proses anonimisasi dilakukan lebih lama pada sampel data $n = 10.000$, dengan total waktu pengelompokan data sekitar 1.5 jam. Selain itu, $k\text{-value} = 25$ menempati posisi waktu anonimisasi data lebih tinggi dibandingkan dengan nilai $k\text{-value}$ lainnya. Melalui hasil pengujian ini, dapat ditarik kesimpulan bahwa proses anonimisasi data dapat dilakukan lebih cepat jika menggunakan sampel ukuran data yang relatif kecil, yaitu $n = 10.000$.

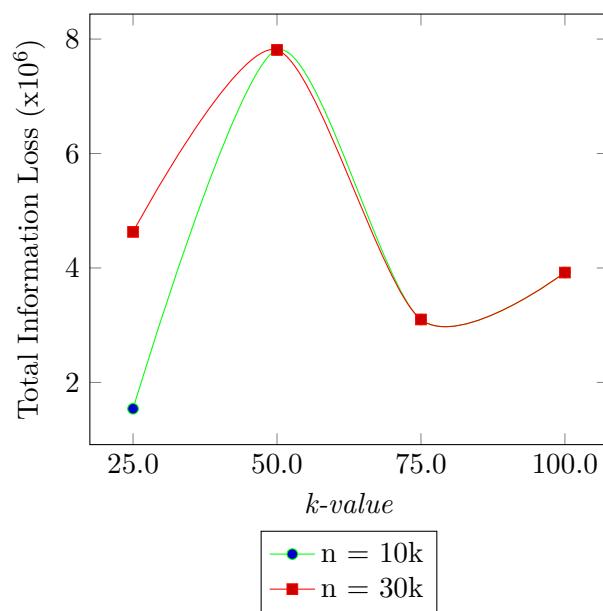
- **Total Information Loss.** Pengamatan ini dilakukan pada algoritma *greedy k-member clustering*. Gambar 5.42 menunjukkan *total information loss* yang dihasilkan oleh $n = 10.000$ dan $n = 30.000$ hampir sama. Perbedaan signifikan *total information loss* hanya ditunjukkan pada $k\text{-value} = 25$, dimana $n = 10.000$ menempati *total information loss* terkecil. Melalui hasil pengujian ini, dapat ditarik kesimpulan bahwa pengelompokan data dapat menghasilkan kualitas pengelompokan yang lebih baik jika total information loss yang dihasilkan kecil dengan menggunakan $n = 10k$ maupun $n = 30k$ untuk setiap pemilihan nilai $k\text{-value}$.



Gambar 5.40: Perubahan Waktu Pengelompokan Data



Gambar 5.41: Perubahan Waktu Anonimisasi Data



Gambar 5.42: Perubahan Total Information Loss

1 Berdasarkan pengujian jumlah QID bervariasi diatas, diketahui bahwa hasil pengelompokan
2 dengan algoritma *greedy k-member clustering* cukup baik jika menggunakan ukuran data
3 apapun dengan nilai $k\text{-value} = 25$, karena memiliki *total information loss* paling kecil dari
4 pengujian lainnya. Dari segi kecepatan komputasi, waktu pengelompokan data jauh lebih
5 lama dibandingkan waktu anonimisasi karena beberapa pekerjaan pengelompokan data pada
6 algoritma *greedy k-member clustering* tidak dapat dikerjakan secara paralel. Hal penting
7 yang perlu diketahui adalah komputasi pengelompokan data dengan $n > 10k$ membutuhkan
8 waktu lebih dari 3 jam sehingga diperlukan pertimbangan untuk menerapkan algoritma *greedy*
k-member clustering untuk lingkungan *big data*.

2 Pengujian Eksperimental Hasil Data Mining

3 Berikut adalah hasil setiap jenis pengujian eksperimental terhadap *total information loss*:

4

5 K-means

6

7 Pada pengujian ini, ingin dibuktikan apakah algoritma *greedy k-member clustering* dan *k-*
8 *anonymity* dapat diterima jika hasil anonimisasinya digunakan untuk *clustering*/pengelompokan
9 data. Pengujian ini menggunakan $QID \in [1..5]$ dengan pengambilan atribut secara acak
10 terhadap $n = 1000$ dan $k\text{-value} = 100$, karena pada pengujian sebelumnya, kondisi ini memiliki
11 *information loss* paling kecil. Berikut penjelasan jenis kajian yang diuji:

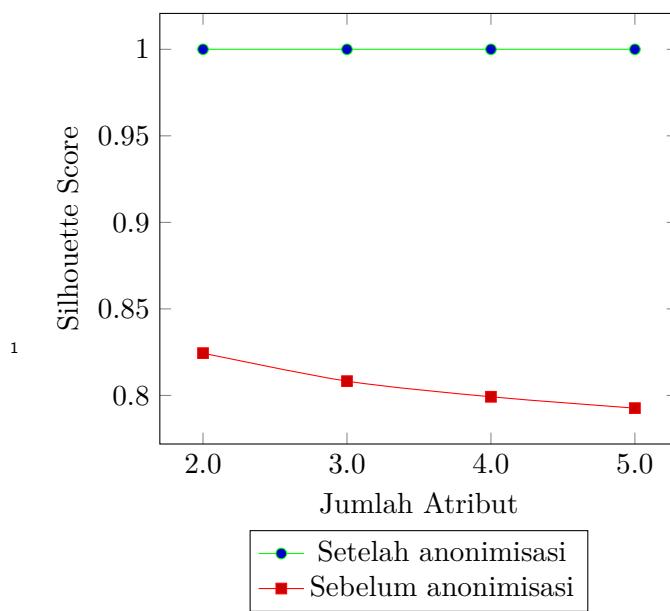
- 12
- **Silhouette score.** Pengamatan ini dilakukan dengan pemodelan *k-means* pada *library* Spark
13 MLlib. Gambar 5.43 menunjukkan *silhouette score* setelah anonimisasi stabil pada nilai 1.
14 Perilaku ini disebabkan karena hasil anonimisasi yang pada *cluster* yang sejenis bernilai sama
15 dan tidak dapat dibedakan satu sama lain. Hasil pengujian ini memberikan kesimpulan
16 bahwa hasil pengelompokan setelah dilakukan anonimisasi lebih baik dibandingkan sebelum
17 dilakukan anonimisasi, karena memiliki *silhouette score* yang lebih tinggi.

18

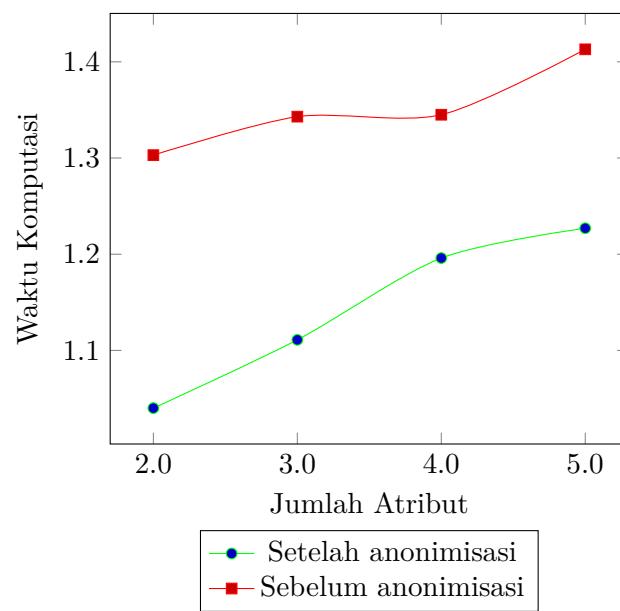
 - **Waktu komputasi.** Pengamatan ini dilakukan dengan komputer lokal (standalone). Gambar
19 5.44 menunjukkan waktu komputasi tercepat untuk pembuatan model *k-means* didapat oleh
20 data yang telah dianonimisasi. Perilaku ini disebabkan karena data yang telah dianonimisasi
21 umumnya memiliki lebih sedikit jumlah variasi nilai, sehingga dapat mempercepat klasifikasi
22 data. Hasil pengujian ini memberikan kesimpulan bahwa jumlah variasi nilai dapat memberi
23 pengaruh terhadap waktu komputasi.

24

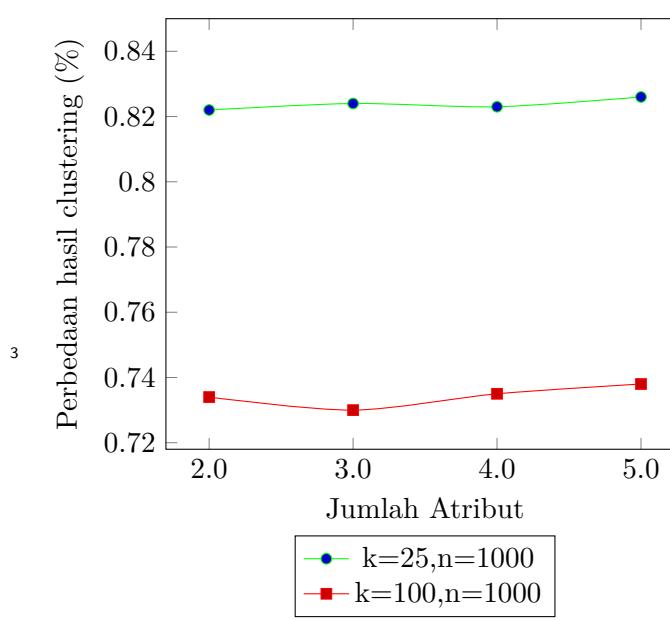
 - **Perbedaan hasil clustering (%)**. Pengamatan ini menghitung persentase perbedaan
25 hasil *clustering* sebelum dan setelah data dianonimisasi pada kondisi jumlah data dan k
26 yang beragam. Gambar 5.45 menunjukkan persentase perbedaan hasil klasifikasi tertinggi
berdasarkan nilai k diraih oleh $k = 25, n = 1000$. Gambar 5.46 menunjukkan persentase
perbedaan hasil klasifikasi tertinggi berdasarkan nilai k diraih oleh $k = 10, n = 1000$. Hasil
dari pengujian ini memberikan kesimpulan bahwa nilai k yang semakin kecil dan ukuran data
semakin besar dapat menyebabkan tingginya perbedaan hasil *clustering*.



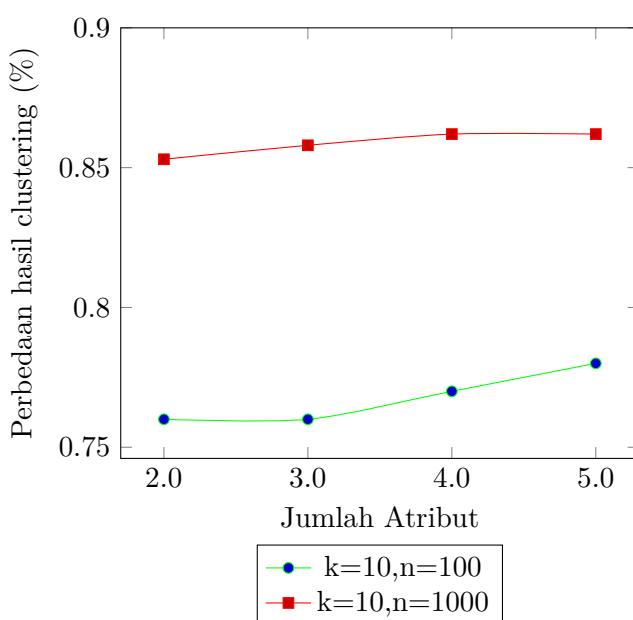
Gambar 5.43: Perbandingan Silhouette Score Model K-Means



Gambar 5.44: Perbandingan Waktu Komputasi Model K-Means



Gambar 5.45: Perbandingan Hasil Clustering Terhadap K-Value



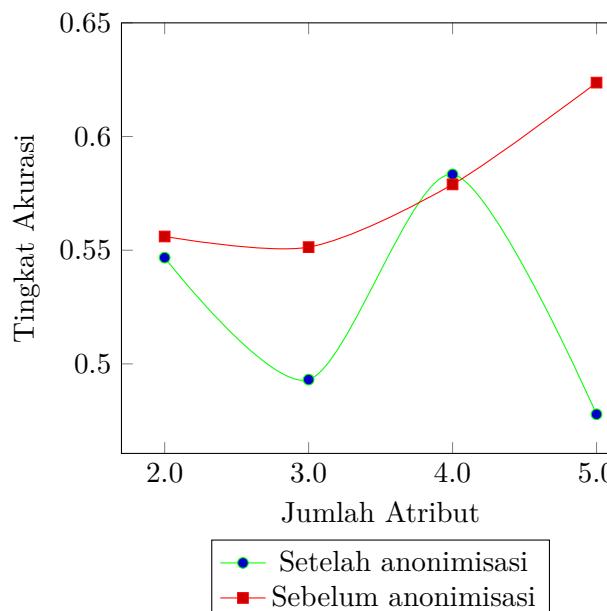
Gambar 5.46: Perbandingan Hasil Clustering Terhadap Jumlah Data

Pada percobaan ini dapat disimpulkan 3 hal penting. Pertama, penggunaan algoritma *greedy k-member clustering* dan *k-anonymity* pada data *credit score* dapat diterima, karena hasil pengelompokannya sudah baik dibuktikan dengan hasil anonomisasi memiliki *silhouette score* bernilai 1. Kedua, penggunaan model *k-means* pada *library Spark MLlib* dapat melakukan komputasi dengan cepat untuk ukuran 1000 data. Ketiga, perbedaan hasil *clustering* paling minimal dapat dicapai dengan memperkecil ukuran data dan memperbesar nilai *k*.

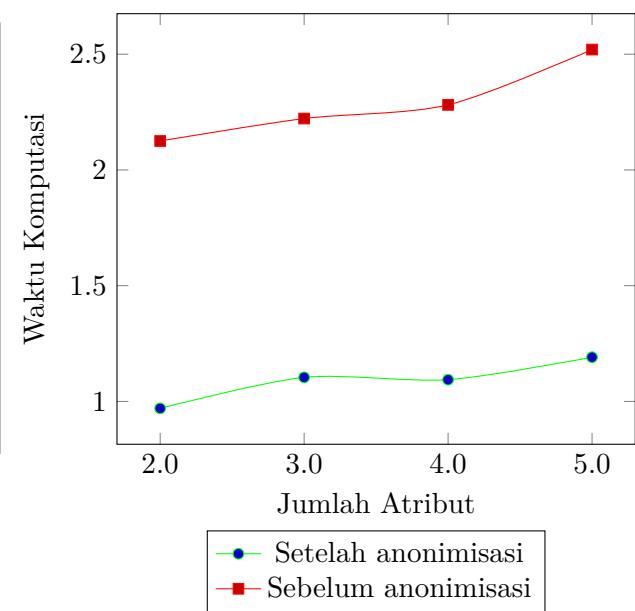
1 **Naive bayes**

2 Pada pengujian ini, ingin dibuktikan apakah algoritma *greedy k-member clustering* dan *k-anonymity* dapat diterima jika hasil anonimisasinya digunakan untuk klasifikasi data. Pengujian ini menggunakan $QID \in [1..5]$ dengan pengambilan atribut secara acak terhadap $n = 1000$ dan $k\text{-value} = 100$, karena pada pengujian sebelumnya, kondisi ini memiliki *total information loss* paling kecil. Berikut penjelasan jenis kajian yang diuji:

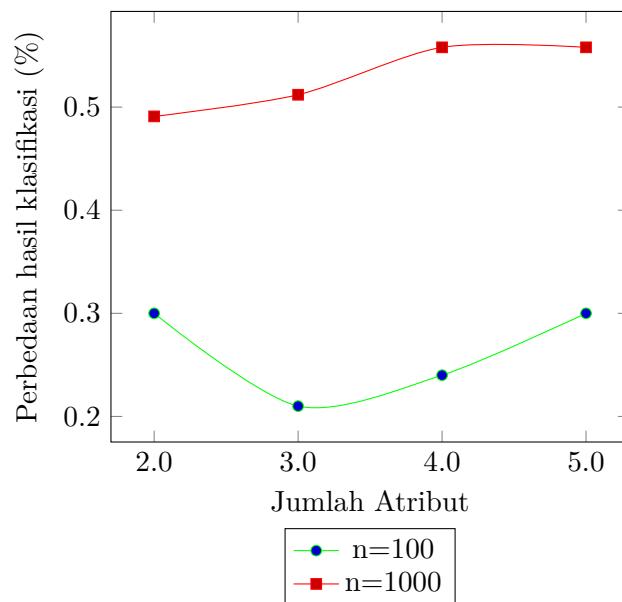
- 3 • **Tingkat akurasi.** Gambar 5.47 menunjukkan perbedaan akurasi yang cukup tinggi (pada
4 kondisi jumlah atribut berjumlah ganjil) dan perbedaan akurasi yang cukup dekat (pada kondisi
5 jumlah atribut berjumlah genap). Perilaku ini disebabkan karena penyisipan jenis atribut
6 bertipe numerik pada data dengan jumlah atribut genap. Hasil pengujian ini memberikan
7 kesimpulan bahwa untuk mendapatkan tingkat akurasi yang baik, data yang digunakan perlu
8 lebih banyak mengandung atribut numerik.
- 9 • **Waktu komputasi.** Gambar 5.48 menunjukkan waktu komputasi tercepat untuk pembuatan
10 model *naive bayes* diraih oleh data yang telah dianonimisasi. Perilaku ini disebabkan karena
11 data yang telah dianonimisasi umumnya memiliki lebih sedikit jumlah variasi nilai, sehingga
12 dapat mempercepat klasifikasi data. Hasil pengujian ini memberikan kesimpulan bahwa
13 jumlah variasi nilai dapat memberi pengaruh terhadap waktu komputasi.
- 14 • **Perbedaan hasil klasifikasi (%)**. Pengamatan ini menghitung persentase perbedaan hasil
15 klasifikasi saat sebelum dan setelah data dianonimisasi pada kondisi jumlah data yang beragam.
16 Gambar 5.49 menunjukkan persentase perbedaan hasil klasifikasi tertinggi diraih oleh $n = 1000$.
17 Hasil dari pengujian ini memberikan kesimpulan bahwa jumlah data dapat mempengaruhi
18 persentase perbedaan hasil klasifikasi, sehingga untuk mendapatkan perbedaan hasil klasifikasi
19 yang minimal perlu mengurangi jumlah data yang diuji.



Gambar 5.47: Perbandingan Tingkat Akurasi Model Naive Bayes



Gambar 5.48: Perbandingan Waktu Komputasi Model Naive Bayes



Gambar 5.49: Perbedaan Hasil Klasifikasi terhadap Jumlah Data

Pada percobaan ini dapat disimpulkan 3 hal penting. Pertama, penggunaan algoritma *greedy k-member clustering* dan *k-anonymity* pada data set *credit score* dapat diterima, karena pada kasus tertentu tingkat akurasi sebelum dan setelah anonimisasi memiliki perbedaan cukup dekat. Kedua, penggunaan model naive bayes pada *library* Spark MLLib dapat melakukan komputasi dengan cepat untuk ukuran 1000 data. Ketiga, perbedaan hasil klasifikasi paling minimal dapat dicapai dengan memperkecil ukuran data pengujian.

Kesimpulan Pengujian Eksperimental

Beberapa hal penting yang perlu dipertimbangkan untuk penelitian selanjutnya. Pertama, pengelompokan data dengan *greedy k-member clustering* sangat lama sehingga dapat diganti ke *library KMeans* milik Spark MLLib. Kedua, *total information loss* dapat diminimalisir dengan penggunaan nilai *k-value* yang cukup besar dan jumlah *quasi-identifier* secukupnya. Ketiga, metode *data mining* terbaik untuk hasil anonimisasi adalah *klasifikasi* karena menghasilkan perbedaan prediksi paling sedikit. Hasil pengujian dapat dilihat pada Tabel 5.12.

Tabel 5.12: Kesimpulan Pengujian Eksperimental

Hasil Pengujian	Kesimpulan
<i>Total Information Loss</i>	Untuk mendapatkan nilai <i>information loss</i> paling minimal, dipilih kolom campuran dengan penggunaan kolom numerik yang tidak terlalu banyak, dan jumlah QID yang tidak terlalu banyak.
<i>Hasil Data Mining</i>	Hasil anonimisasi data memiliki pemodelan data mining cukup baik karena memiliki metrik data mining (<i>silhouette score</i> , tingkat akurasi) yang dekat dengan data sebelum dilakukan anonimisasi. Selain itu persentase perbedaan hasil clustering, klasifikasi sebelum dan setelah dilakukan anonimisasi juga kecil.

1

BAB 6

2

KESIMPULAN DAN SARAN

3 Pada bab ini akan dijelaskan kesimpulan dari awal hingga akhir penelitian beserta saran untuk
4 penelitian selanjutnya.

5 6.1 Kesimpulan

6 Kesimpulan yang dapat diambil dari penelitian ini adalah sebagai berikut:

- 7 • Secara singkat tahapan *greedy k-member clustering* adalah mencari kandidat anggota sebuah
8 kelompok data menggunakan nilai *information loss* paling kecil
- 9 • Secara singkat tahapan *k-anonymity* adalah mengambil sebuah *cluster* dan melakukan proses
10 anonimisasi pada setiap kolom *cluster* yang bernilai unik menggunakan pohon DGH.
- 11 • Secara singkat, implementasi algoritma *greedy k-member clustering* dilakukan pada perangkat
12 lunak anonimisasi menggunakan *library* Spark SQL, karena banyak operasi pada algoritma
13 ini yang dapat diimplementasikan dengan kueri SQL. Implementasi algoritma ini dapat
14 menimbulkan permasalahan memori akibat prinsip *lazy evaluation* pada Spark.
- 15 • Performa algoritma *greedy k-member clustering* pada lingkungan *big data* sangat buruk
16 (mencapai waktu 5 jam untuk pengelompokan 10.000 data), sedangkan performa dari algoritma
17 *k-anonymity* pada lingkungan *big data* cukup baik (mencapai waktu kurang dari 1 jam untuk
18 melakukan anonimisasi 10.000 data).
- 19 • Performa pemodelan *data mining* menggunakan teknik *clustering (k-means)* dan teknik
20 klasifikasi (*naive bayes*) pada lingkungan *big data* sangat baik karena menggunakan *library*
21 bawaan Spark yaitu Spark MLlib.
- 22 • Kualitas informasi pengelompokan data dengan *greedy k-member clustering* cukup baik jika
23 menggunakan kolom campuran yang tidak menggunakan terlalu banyak kolom numerik dan
24 menggunakan jumlah *quasi-identifier* secukupnya. Hal ini terbukti pada pengujian kualitas
25 informasi, dimana kondisi tersebut memiliki *total information loss* yang paling rendah.
- 26 • Kualitas hasil *data mining* menggunakan pemodelan *clustering k-means* cukup baik untuk
27 diterapkan pada data anonimisasi, karena melalui pengujian hasil *data mining* yang dilakukan
28 sebelumnya, diketahui bahwa *silhouette score* dan perbedaan hasil *clustering* sebelum dan
29 setelah dilakukan anonimisasi cukup kecil.

- Kualitas hasil *data mining* menggunakan pemodelan klasifikasi *naive bayes* cukup baik untuk diterapkan pada data anonimisasi, karena melalui pengujian hasil *data mining* yang dilakukan sebelumnya, diketahui bahwa tingkat akurasi dan perbedaan hasil klasifikasi sebelum dan setelah dilakukan anonimisasi cukup kecil.

6.2 Saran

Saran untuk penelitian selanjutnya adalah sebagai berikut:

- Pada penelitian ini, diketahui bahwa algoritma *greedy k-member clustering* memiliki waktu komputasi yang sangat lama jika merancang algoritma pengelompokan secara mandiri. Solusi dari permasalahan pengelompokan data pada lingkungan *big data* adalah menggunakan *library* KMeans pada Spark MLlib agar waktu eksekusinya menjadi lebih efisien.
- Pada penelitian ini, pernah terjadi *error java.lang.OutOfMemoryError* terkait *lazy evaluation* pada Spark yang diterapkan pada algoritma yang iteratif. Dikutip dari Medium¹, masalah ini terjadi ketika fungsi *transformation filter()*, *union()* dipanggil pada setiap iterasi. Fungsi *transformation* adalah fungsi dengan biaya komputasi yang mahal, terutama jika dipanggil beberapa kali dalam satu iterasi. Konsep *lazy evaluation* pada Spark mirip dengan konsep rekursif, dimana fungsi *transformation* akan dijalankan saat fungsi *action* dipanggil. *Error* ini terjadi ketika fungsi *action* dipanggil pada iterasi tertentu yang membuat fungsi *transformation* pada iterasi sebelumnya juga ikut dijalankan. Solusi yang dapat diterima adalah menyimpan dan membaca hasil komputasi pada sistem penyimpanan HDFS.
- Struktur data pada pemrosesan *big data* perlu diperhatikan. Diusahakan untuk memilih struktur data Dataframe/RDD, karena hanya operasi tersebut yang dapat berjalan secara paralel. Selain itu diusahakan untuk tidak terlalu banyak menggunakan operasi perulangan. Pada kasus tertentu, operasi perulangan dapat diganti dengan operasi kueri SQL.

¹<https://medium.com/@cyneo41/memory-issues-caused-by-lazy-evaluation-in-spark-a419517c1732>

DAFTAR REFERENSI

- [1] Han, J., Kamber, M., dan Pei, J. (2012) *Data Mining: Concepts and Techniques*, 3rd edition. Morgan Kaufmann, Waltham, USA.
- [2] Veronica S. Moertini, M. T. A. (2020) *Pengantar Data Science dan Aplikasinya bagi Pemula*, 1st edition. Unpar Press, Bandung, Indonesia.
- [3] Judith Hurwitz, M. K., Alan Nugent (2013) *Big Data For Dummies*, 1st edition. John Wiley and Sons, Inc., Hoboken, New Jersey.
- [4] Holden Karau, P. W., Andy Konwinski (2015) *Learning Spark*, 1st edition. O'Reilly Media, Inc, Sebastopol, CA.
- [5] Martin Odersky, B. V., Lex Spoon (2008) *Programming in Scala*, 1st edition. Artima, Inc, Mountain View, California.
- [6] MENDES, R. dan VILELA, J. P. (2017) Privacy-preserving data mining: Methods, metrics, and applications. *IEEE Access*, **5**, 1–21.
- [7] LEI XU, J. W., CHUNXIAO JIANG (2014) Information security in big data: Privacy and data mining. *IEEE Access*, **2**, 1149–1176.
- [8] Byun, J.-W., Kamra, A., Bertino, E., dan Li, N. (2007) Efficient k-anonymity using clustering techniques. *CERIAS Tech Report 2006-10*, **1**, 1–12.
- [9] Sang Ni, Q. Q., Mengbo Xie (2017) Clustering based k-anonymity algorithm for privacy preservation. *International Journal of Network Security*, **1**, 1062–1071.
- [10] Vanessa Ayala Rivera, T. C., Patrick McDonagh (2014) A systematic comparison and evaluation of k-anonymization algorithms for practitioners. *Transaction on Data Privacy* **7**, **1**, 337–370.

LAMPIRAN A

KONFIGURASI LIBRARY SPARK

A.1 Perangkat Lunak Eksplorasi

Konfigurasi library Spark menggunakan build.sbt adalah sebagai berikut.

Listing A.1: build.sbt (Eksplorasi)

```
1 name := "PerangkatLunakEksplorasi"
2
3 version := "0.1"
4
5 scalaVersion := "2.11.12"
6
7 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
8 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.3"
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
11 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.3"
```

A.2 Perangkat Lunak Anonimisasi

Konfigurasi library Spark menggunakan build.sbt adalah sebagai berikut.

Listing A.2: build.sbt (Anonimisasi)

```
1 name := "PerangkatLunakAnonimisasi"
2
3 version := "0.1"
4
5 scalaVersion := "2.11.12"
6
7 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
8 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.3"
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
11 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.3"
```

A.3 Perangkat Lunak Pengujian

Konfigurasi library Spark menggunakan build.sbt adalah sebagai berikut.

Listing A.3: build.sbt (Pengujian)

```
1 name := "PerangkatLunakPengujian"
2
3 version := "0.1"
4
5 scalaVersion := "2.11.12"
6
7 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
8 libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.3"
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
11 libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.3"
12
13 // https://mvnrepository.com/artifact/org.apache.spark/spark-mllib
14 libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.4.3"
```


LAMPIRAN B

KODE PROGRAM PERANGKAT LUNAK EKSPLORASI

B.1 Kelas MainExploratory

Kode program untuk kelas *MainExploratory* adalah sebagai berikut.

Listing B.1: MainExploratory.scala

```
1 package ExploratoryModel
2
3 import ExaminationModel.MainExamination.{get_attribute_datatype_json, get_attribute_name_json, read_element_from_json}
4 import org.apache.spark.sql.types.{IntegerType, StringType}
5 import org.apache.spark.sql.{DataFrame, SparkSession}
6
7 import scala.collection.mutable.ListBuffer
8
9 object MainExploratory {
10   def main(args:Array[String]): Unit = {
11     val spark = SparkSession
12       .builder.master("local[*]")
13       .appName("SparkSQL")
14       .getOrCreate()
15
16   // Parameter Pengujian
17   val path_JSON = args(0)
18   val json: DataFrame = spark.read.option("multiline", "true").json(path_JSON).cache() // ganti di sini
19   val path_data_input = json.select("input_path").first().getString(0) // ganti di sini
20   val datasetInput = spark.read.format("csv").option("header", "true").load(path_data_input)
21   val columnSelectionWithID = generate_dataframe_from_csv(spark, json, datasetInput)
22
23   import org.apache.spark.sql.functions._
24   val outputPathExploratoryAnalysis = json.select("output_path").first().getString(0)
25   val columnsModify = columnSelectionWithID.columns.map(col).map(colName => {
26     val attrDF = columnSelectionWithID.select(colName).distinct()
27     attrDF.coalesce(1)
28       .write
29         .option("header", "true")
30         .option("sep", ",")
31         .mode("overwrite")
32         .csv(outputPathExploratoryAnalysis+colName)
33   })
34
35 }
36
37
38 def generate_dataframe_from_csv(spark:SparkSession, json:DataFrame, dataInput: DataFrame):DataFrame= {
39   val selectedColumn:ListBuffer[Seq[String]] = read_element_from_json(json, "selected_column")
40
41   val selectedColumnAttribute:List[String] = get_attribute_name_json(selectedColumn).toList
42   val selectedColumnDatatype:List[String] = get_attribute_datatype_json(selectedColumn).toList
43
44   var columnSelectionDF = dataInput.select(selectedColumnAttribute.head,selectedColumnAttribute.tail: _*).toDF() // ganti di sini
45
46   for(i <- 0 until selectedColumnAttribute.length){
47
48     val attrName = selectedColumnAttribute(i)
49     val datatype = selectedColumnDatatype(i)
50
51
52     if(datatype == "category"){
53       columnSelectionDF = columnSelectionDF.withColumn(attrName+"_new",columnSelectionDF(attrName).cast(StringType))
54       columnSelectionDF = columnSelectionDF.drop(attrName)
55     }
56     else{
57       columnSelectionDF = columnSelectionDF.withColumn(attrName+"_new",columnSelectionDF(attrName).cast(IntegerType))
58       columnSelectionDF = columnSelectionDF.drop(attrName)
59     }
60
61   }
62
63   val columnSelectionRenamedDF = columnSelectionDF.toDF(selectedColumnAttribute: _*)
64
65   return columnSelectionRenamedDF
66 }
67 }
```


LAMPIRAN C

KODE PROGRAM PERANGKAT LUNAK ANONIMISASI

C.1 Kelas Node

Kode program untuk kelas *Node* adalah sebagai berikut.

Listing C.1: Node.scala

```
1 package AnonymizationModel
2
3 class Node(value: String, lvl: Int) extends Serializable {
4   var name : String = value
5   var left, right : Node = null
6   var parent: Node = null
7   var level : Int = lvl
8 }
```

C.2 Kelas BinaryTree

Kode program untuk kelas *BinaryTree* adalah sebagai berikut.

Listing C.2: BinaryTree.scala

```
1 package AnonymizationModel
2
3 // https://www.geeksforgeeks.org/binary-tree-set-1-introduction/
4 class BinaryTree(key: String) extends Serializable {
5
6   var root: Node = null
7
8   def this(){
9     this(" ")
10    this.root = null
11  }
12
13 def search(data: String): Node = {
14   return search(data,root);
15 }
16
17 def search(name: String, node: Node): Node = {
18   if(node!=null){
19     if(node.name == name){
20       return node
21     }
22     else{
23       var foundNode = search(name,node.left)
24       if(foundNode == null){
25         foundNode = search(name,node.right)
26       }
27       return foundNode
28     }
29   } else{
30     return null
31   }
32 }
33
34 def getMax(a: Int, b: Int): Int ={
35   if(a>b) return a
36   else return b
37 }
38
39 def isLeaf():Boolean={
40   return (root.right == null && root.left ==null)
41 }
42
43 def getHeight(root: Node): Int ={
44   return this.root.level-1
45 }
46
47
48 }
```

C.3 Kelas GreedyKMemberClustering

Kode program untuk kelas *GreedyKMemberClustering* adalah sebagai berikut.

Listing C.3: GreedyKMemberClustering.scala

```

1 package AnonymizationModel
2
3 import org.apache.hadoop.fs.{FileSystem, Path}
4 import org.apache.spark.sql.expressions.UserDefinedFunction
5 import org.apache.spark.sql.functions.{max, _}
6 import org.apache.spark.sql.types.StructType
7 import org.apache.spark.sql.{Column, DataFrame, SparkSession}
8
9 import scala.collection.mutable.ListBuffer
10
11 class GreedyKMemberClustering extends Serializable {
12
13   def getNumPartitions(size:Int):Int = {
14     var numPartitions = 1
15     if(size >= 10) numPartitions = size/10
16     return numPartitions
17   }
18
19   def calculateNumericalDistance = udf( (num1: Int,num2: Int, max_num:Int, min_num:Int) => {
20     Math.abs(num1-num2)*1.0/Math.abs(max_num-min_num)
21   })
22
23   def calculateNumericalInformationLoss(cluster_size:Int) = udf( (max_num: Int,min_num: Int) => {
24     Math.abs(max_num-min_num)*1.0/cluster_size
25   })
26
27   def calculateNumericalInformationLossUnion(cluster_size:Int) = udf( (max_c:Int, min_c:Int, num:Int) => {
28     var max_value = max_c
29     var min_value = min_c
30     if(num > max_value){
31       max_value = num
32     }
33     if(num < min_value){
34       min_value = num
35     }
36     Math.abs(max_value-min_value)*1.0/cluster_size
37   })
38
39   def calculateCategoricalDistance(binaryTree: BinaryTree):UserDefinedFunction = udf( (category1: String, category2: String) => {
40     if(binaryTree!=null){
41       val node1 = binaryTree.search(category1)
42       val node2 = binaryTree.search(category2)
43       if(node1 != null && node2 != null && node1.name != node2.name){
44         val LCA = new LowestCommonAncestor()
45         val LCA_root_name = LCA.findLCA(binaryTree.root, node1.name, node2.name)
46         val H_subtree = binaryTree.search(LCA_root_name).level
47         val H_TD = binaryTree.getHeight(binaryTree.root).toDouble
48         BigDecimal(H_subtree*1.0 / H_TD).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
49       }
50     else{
51       1.0
52     }
53   }
54   else{
55     1.0
56   }
57 }
58 })
59
60   def calculateCategoricalInformationLoss(binaryTree: BinaryTree):UserDefinedFunction = udf( (listDistinctValues:Seq[Any]) => {
61   if(binaryTree!=null){
62
63     val numDistinctValues = listDistinctValues.length
64
65     if(numDistinctValues == 2){
66       val node1 = binaryTree.search(listDistinctValues(0).toString)
67       val node2 = binaryTree.search(listDistinctValues(1).toString)
68       if(node1 != null && node2 != null){
69         val LCA = new LowestCommonAncestor()
70         val LCA_root_name = LCA.findLCA(binaryTree.root, node1.name, node2.name)
71         val H_subtree = binaryTree.search(LCA_root_name).level
72         val H_TD = binaryTree.getHeight(binaryTree.root).toDouble
73         BigDecimal(H_subtree*1.0 / H_TD).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
74       }
75     else{
76       1.0
77     }
78   }
79   else if(numDistinctValues > 2){
80     val H_subtree = 1
81     val H_TD = binaryTree.getHeight(binaryTree.root).toDouble
82     BigDecimal(H_subtree*1.0 / H_TD).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
83   }
84   else{
85     0.0
86   }
87 }
88
89   else{
90     1.0
91   }

```

```

92 })
93 }
94
95 def calculateCategoricalInformationLossUnion(binaryTree: BinaryTree): UserDefinedFunction = udf( (check:Boolean, listCategory:
96   Array[String]) => {
97   if(binaryTree!=null){
98     var numDistinctValues = 0
99     if(check){
100       numDistinctValues = listCategory.length
101     }
102     else{
103       numDistinctValues = listCategory.length+1
104     }
105
106     if(numDistinctValues == 2){
107       val node1 = binaryTree.search(listCategory(0).toString)
108       val node2 = binaryTree.search(listCategory(1).toString)
109       if(node1 != null && node2 != null){
110         val LCA = new LowestCommonAncestor()
111         val LCA_root_name = LCA.findLCA(binaryTree.root, node1.name, node2.name)
112         val H_subtree = binaryTree.search(LCA_root_name).level
113         val H_TD = binaryTree.getHeight(binaryTree.root).toDouble
114         BigDecimal(H_subtree*1.0 / H_TD).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
115       }
116     else{
117       0.0
118     }
119   }
120 }
121 else if(numDistinctValues > 2){
122   val H_subtree = 1
123   val H_TD = binaryTree.getHeight(binaryTree.root).toDouble
124   BigDecimal(H_subtree*1.0 / H_TD).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
125 }
126 else{
127   0.0
128 }
129 }
130 }
131 else{
132   0.0
133 }
134 }
135 }
136 })
137
138 def sum_(cols: Column*) = cols.foldLeft(lit(0))(_ + _)
139
140 def furthest_record_from_r_optimize(json: DataFrame, S: DataFrame, r: DataFrame, listBinaryTree: ListBuffer[BinaryTree]):
141   DataFrame = {
142   val listColumnName = S.columns.toSeq
143   val r_temp = r.select(r.columns.map(c => col(c).alias(c+"_r")): _*)
144   var S_temp = S
145
146   S_temp = S_temp.crossJoin(r_temp)
147   var i = 1
148
149   S_temp.dtypes.foreach(element =>
150     if(!element._1.contains("_r") && !element._1.contains("id")){
151       if(element._2.contains("Integer")){
152         val max_value = S_temp.select(max(element._1).as("max_"+element._1))
153         val min_value = S_temp.select(min(element._1).as("min_"+element._1))
154         S_temp = S_temp.crossJoin(max_value)
155         S_temp = S_temp.crossJoin(min_value)
156         S_temp = S_temp.withColumn("Dist_" + element._1, calculateNumericalDistance(col(element._1), col(element._1+"_r"), col("max_"
157           + element._1), col("min_" + element._1) ))
158       i += 1
159     }
160     else{
161       S_temp = S_temp.withColumn("Dist_" + element._1, calculateCategoricalDistance(listBinaryTree(i))( S_temp(element._1), S_temp
162         (element._1+"_r") ))
163       i += 1
164     }
165   }
166
167   val distance = S_temp.select(S_temp.columns.filter(_.startsWith("Dist_")).map(S_temp(_)) : _*)
168   val columnstosum = distance.columns.toSeq.map(col _)
169
170   S_temp = S_temp.withColumn("DistRecord", sum_(columnstosum: _*))
171   S_temp = S_temp.orderBy(desc("DistRecord"))
172
173   val furthest_record_from_r = S_temp.select(listColumnName.head, listColumnName.tail: _*).limit(1)
174
175   return furthest_record_from_r
176 }
177
178 def calculate_substraction_information_loss_optimize(json: DataFrame, unionDF: DataFrame, c: DataFrame, cluster_size:Int):
179   DataFrame = {
180   var cluster_temp = unionDF
181
182   unionDF.dtypes.filter(!_.1.contains("_")).foreach(element =>
183     if(!element._1.contains("id") && !element._1.contains("Cluster") && !element._1.contains("Info")){
184       if(element._2.contains("Integer")){
185         cluster_temp = cluster_temp.withColumn("ILcluster_" + element._1, calculateNumericalInformationLoss(cluster_size)(col(""
186           max_ + element._1), col("min_" + element._1)))
187       }
188     }
189   }
190 }
```

```

185     cluster_temp = cluster_temp.withColumn("ILunion_" + element._1, calculateNumericalInformationLossUnion(cluster_size)(col
186         ("max_" + element._1), col("min_" + element._1), col(element._1)))
187     }
188     else{
189         cluster_temp = cluster_temp.withColumn("ILcluster_" + element._1, lit(1))
190         cluster_temp = cluster_temp.withColumn("ILunion_" + element._1, lit(1))
191     }
192   }
193   val infoloss_cluster = cluster_temp.select(cluster_temp.columns.filter(_.contains("ILcluster_"))).map(cluster_temp(_): _*)
194   val sum_infoloss_cluster = infoloss_cluster.columns.toSeq.map(col _)
195
196   val infoloss_union = cluster_temp.select(cluster_temp.columns.filter(_.contains("ILunion_"))).map(cluster_temp(_): _*)
197   val sum_infoloss_union = infoloss_union.columns.toSeq.map(col _)
198
199   cluster_temp = cluster_temp.withColumn("Total_ILcluster", sum_(sum_infoloss_cluster: _*) * cluster_size)
200   cluster_temp = cluster_temp.withColumn("Total_ILunion", sum_(sum_infoloss_union: _*) * cluster_size)
201
202   cluster_temp = cluster_temp.withColumn("Subs_IL", col("Total_ILunion") - col("Total_ILcluster"))
203   cluster_temp = cluster_temp.drop(cluster_temp.columns.filter(_.startsWith("IL")): _*)
204
205   return cluster_temp
206 }
207
208 def delete_folder_hdfs(pathName: String, hdfs: FileSystem) {
209   val path = new Path(pathName)
210   if (hdfs.exists(path)) {
211     hdfs.delete(path, true)
212   }
213 }
214
215
216 def greedy_k_member_clustering(spark: SparkSession, json: DataFrame, S: DataFrame, k: Int, numSampleDatas:Int, listBinaryTree:
217   ListBuffer[BinaryTree], hdfs: FileSystem, path_HDFS:String, path_delete_HDFS:String): DataFrame = {
218
219   var S_size = numSampleDatas
220   val schema = S.schema
221
222   if(S_size <= k){
223     return S
224   }
225
226   // Melakukan inisialisasi keseluruhan
227   this.delete_folder_hdfs(path_delete_HDFS + "/gkmc0_tmp/", hdfs)
228   S.write.option("header", "true").csv(path_HDFS + "/gkmc0_tmp/")
229   var S_temp: DataFrame = null
230   var clusters: DataFrame = null
231   var clusters_schema: StructType = null
232   var r: DataFrame = null
233
234   // Melakukan inisialisasi while (S_temp.count() >= k)
235   var c: DataFrame = null
236   var cluster_name: String = ""
237   var numCluster: Int = 0
238
239   // Apabila jumlah data >= k, maka lakukan perulangan
240   while (S_size >= k){
241
242     // Mengambil 1 record secara acak
243     this.delete_folder_hdfs(path_delete_HDFS + "/gkmc0/", hdfs)
244     hdfs.rename(new Path(path_HDFS + "/gkmc0_tmp"), new Path(path_HDFS + "/gkmc0"))
245     S_temp = spark.read.option("header", "true").schema(schema).csv(path_HDFS + "/gkmc0/").repartition(45)
246     r = S_temp.orderBy(rand()).limit(1)
247
248     // Mencari record tabel S terjauh dengan record r
249     r = furthest_record_from_r_optimize(json, S_temp, r, listBinaryTree).cache() // cache gk boleh dihilangkan
250
251     // Membuang record r dari tabel S (1)
252     S_size -= 1
253     S_temp = S_temp.except(r).repartition(45)
254
255     // Melakukan overwrite S_temp pada HDFS (1)
256     S_temp.coalesce(1).write.option("header", "true").csv(path_HDFS + "/gkmc0_tmp/")
257     S_temp.unpersist()
258
259     // Membuat penamaan sebuah cluster
260     numCluster += 1
261     cluster_name = "Cluster_" + numCluster
262
263     // Membuat record terjauh sebagai centroid cluster baru
264     c = r
265     var cluster_size = 1
266     val min_max_column = min_max_cluster(c)
267
268     // Menyimpan cluster sementara pada HDFS
269     this.delete_folder_hdfs(path_delete_HDFS + "/gkmc1_tmp/", hdfs)
270     c.coalesce(1).write.option("header", "true").csv(path_HDFS + "/gkmc1_tmp/")
271
272     // Mencari kelompok data pada cluster terdekat (c)
273     while ( cluster_size < k ) {
274
275       // Membaca file HDFS
276       this.delete_folder_hdfs(path_delete_HDFS + "/gkmc1/", hdfs)
277       hdfs.rename(new Path(path_HDFS + "/gkmc1_tmp"), new Path(path_HDFS + "/gkmc1"))
278       c = spark.read.option("header", "true").schema(c.schema).csv(path_HDFS + "/gkmc1/").repartition(45)
279
280       // Mencari record terbaik sebagai anggota cluster (c)
281       this.delete_folder_hdfs(path_delete_HDFS + "/gkmc0/", hdfs)
282       hdfs.rename(new Path(path_HDFS + "/gkmc0_tmp"), new Path(path_HDFS + "/gkmc0"))

```

```

282     S_temp = spark.read.option("header", "true").schema(schema).csv(path_HDFS+"/gkmc0/").repartition(45)
283     r = find_best_record(spark,json,S_temp,c,cluster_size,min_max_column)
284
285     // Mengelompokan data terhadap c -> find best record
286     c = c.union(r)
287     cluster_size += 1
288
289     // Menyimpan cluster sementara pada HDFS
290     c.coalesce(1).write.option("header", "true").csv(path_HDFS+"/gkmc1_tmp/")
291
292     // Membuang record r dari tabel S (2)
293     S_size -= 1
294     S_temp = spark.read.option("header", "true").schema(schema).csv(path_HDFS+"/gkmc0/").repartition(45)
295     S_temp = S_temp.except(r)
296
297     // Melakukan overwrite S_temp pada HDFS (2)
298     this.delete_folder_hdfs(path_delete_HDFS+"/gkmc0_tmp/",hdfs)
299     S_temp.coalesce(1).write.option("header", "true").csv(path_HDFS+"/gkmc0_tmp/")
300
301 }
302
303 this.delete_folder_hdfs(path_delete_HDFS+"/gkmc1/",hdfs)
304 hdfs.rename(new Path(path_HDFS+"/gkmc1_tmp"), new Path(path_HDFS+"/gkmc1"))
305 c = spark.read.format("csv").option("header", "true").schema(c.schema).load(path_HDFS+"/gkmc1/").repartition(45)
306 val min_max = min_max_cluster(c)
307 c = c.crossJoin(min_max).repartition(45)
308 c = c.withColumn("Cluster",lit(cluster_name))
309
310 // Mengelompokan data
311 if(clusters == null) {
312     clusters = c
313     clusters_schema = clusters.schema
314     this.delete_folder_hdfs(path_delete_HDFS+"/gkmc2_tmp/",hdfs)
315     clusters.coalesce(1).write.option("header", "true").csv(path_HDFS+"/gkmc2_tmp/")
316 }
317 else {
318     this.delete_folder_hdfs(path_delete_HDFS+"/gkmc2/",hdfs)
319     hdfs.rename(new Path(path_HDFS+"/gkmc2_tmp"), new Path(path_HDFS+"/gkmc2"))
320     clusters = spark.read.format("csv").option("header", "true").schema(clusters.schema).load(path_HDFS+"/gkmc2/").repartition(45)
321     clusters = clusters.union(c)
322     clusters.coalesce(1).write.option("header", "true").csv(path_HDFS+"/gkmc2_tmp/")
323 }
324
325 if(S_size == 0){
326     this.delete_folder_hdfs(path_delete_HDFS+"/gkmc0/",hdfs)
327     hdfs.rename(new Path(path_HDFS+"/gkmc0_tmp"), new Path(path_HDFS+"/gkmc0"))
328     this.delete_folder_hdfs(path_delete_HDFS+"/gkmc2/",hdfs)
329     hdfs.rename(new Path(path_HDFS+"/gkmc2_tmp"), new Path(path_HDFS+"/gkmc2"))
330 }
331
332 spark.sqlContext.clearCache()
333
334 }
335
336 // Jika S_temp masih ada sisa (masih ada data yang belum di kelompokan)
337 var initialize = true
338
339 while (S_size > 0){
340
341     // Mengambil record yang tersisa secara acak
342     this.delete_folder_hdfs(path_delete_HDFS+"/gkmc0/",hdfs)
343     hdfs.rename(new Path(path_HDFS+"/gkmc0_tmp"), new Path(path_HDFS+"/gkmc0"))
344     S_temp = spark.read.format("csv").option("header", "true").schema(S_temp.schema).load(path_HDFS+"/gkmc0/").repartition(45)
345     r = S_temp.orderBy(rand()).limit(1).cache()
346
347     // Membuang record r dari tabel S (3)
348     S_size -= 1
349     S_temp = S_temp.except(r)
350     S_temp.coalesce(1).write.option("header", "true").csv(path_HDFS+"/gkmc0_tmp/")
351     if(S_size == 0){
352         this.delete_folder_hdfs(path_delete_HDFS+"/gkmc0/",hdfs)
353         hdfs.rename(new Path(path_HDFS+"/gkmc0_tmp"), new Path(path_HDFS+"/gkmc0"))
354     }
355
356     // Membaca clusters dari file HDFS (1)
357     clusters = null
358     if(initialize){
359         this.delete_folder_hdfs(path_delete_HDFS+"/gkmc2/",hdfs)
360         hdfs.rename(new Path(path_HDFS+"/gkmc2_tmp"), new Path(path_HDFS+"/gkmc2"))
361         clusters = spark.read.format("csv").option("header", "true").schema(clusters_schema).load(path_HDFS+"/gkmc2/").repartition(45)
362         initialize = false
363     }
364     else{
365         this.delete_folder_hdfs(path_delete_HDFS+"/gkmc2/",hdfs)
366         hdfs.rename(new Path(path_HDFS+"/gkmc2_tmp"), new Path(path_HDFS+"/gkmc2"))
367         clusters = spark.read.format("csv").option("header", "true").schema(clusters_schema).load(path_HDFS+"/gkmc2/").repartition(45)
368     }
369
370     // Mencari cluster terbaik (c) untuk record r
371     val min_max_column = clusters.select(clusters.columns.filter(x => x.contains("_") || x.contains("Cluster"))).map(clusters(_))
372     :_*).distinct()
373     c = find_best_cluster(spark,json,clusters,r,numCluster,min_max_column)
374
375     // Menghilangkan sebuah cluster dalam clusters
376 }
```

```

377 var cluster = clusters.filter(clusters("Cluster").contains(c.first().getString(0))) // mengandung nama cluster c
378 clusters = clusters.except(cluster)
379
380 // Membuang kolom max_attr dan min_attr pada cluster
381 cluster = cluster.select(cluster.columns.filter(colName => !colName.startsWith("max_") && !colName.contains("min_") && !
382   colName.contains("Cluster")).map(cluster(_)) : _*)
383
384 // Mencari min max value dari cluster baru
385 cluster = cluster.union(r)
386 val minmax_cluster = min_max_cluster(cluster)
387 var updated_cluster_temp = cluster.crossJoin(minmax_cluster) // join dengan min max
388 updated_cluster_temp = updated_cluster_temp.crossJoin(c) // join dengan nama cluster
389
390 // Menulis ulang cluster yang baru pada HDFS
391 clusters = clusters.union(updated_cluster_temp).cache()
392 clusters.coalesce(1).write.option("header", "true").csv(path_HDFS+"/gkmc2_tmp/")
393 if(S.size == 0){
394   this.delete.folder_hdfs(path_delete_HDFS+"/gkmc2/",hdfs)
395   hdfs.rename(new Path(path_HDFS+"/gkmc2_tmp"), new Path(path_HDFS+"/gkmc2"))
396 }
397
398 spark.sqlContext.clearCache()
399
400 clusters = spark.read.format("csv").option("header", "true").schema(clusters_schema).load(path_HDFS+"/gkmc2/").repartition(45)
401 return clusters
402
403 }
404
405 def list_of_all_attribute(spark: SparkSession, record: DataFrame): List[String] = {
406   record.createOrReplaceTempView("tAdults")
407   val test = spark.catalog.listColumns("tAdults").select("name", "datatype")
408   val df = test.select("name")
409   val columnName = df.collect().map(_(0)).toList.asInstanceOf[List[String]]
410   return columnName
411 }
412
413
414 def find_best_record(spark: SparkSession, json: DataFrame, S: DataFrame, c: DataFrame, cluster_size: Int,min_max_column: DataFrame): DataFrame = {
415   val S_temp = S
416   val cluster_union_r = S_temp.crossJoin(min_max_column)
417   val subs_infoloss = calculate_subtraction_information_loss_optimize(json,cluster_union_r,c,cluster_size)
418   val subs_infoloss_ordered = subs_infoloss.orderBy(asc("Subs_IL")).limit(1)
419   val best = subs_infoloss_ordered.select(subs_infoloss_ordered.columns.filter(colName =>
420     !colName.startsWith("max_") && !colName.startsWith("min_") &&
421     !colName.startsWith("Total_") && !colName.startsWith("Subs_")).map(subs_infoloss_ordered(_)): _*)
422
423   return best
424 }
425
426 def min_max_cluster(c: DataFrame):DataFrame = {
427   var result:DataFrame = null
428   c.dtypes.filter(element => element._2 == "IntegerType").foreach{element =>
429     if(element._1 != "id" && element._2.contains("Integer")){
430       if(result == null){
431         result = c.select(max(element._1).as("max_" + element._1),min(element._1).as("min_" + element._1))
432       } else{
433         val c_temp = c.select(max(element._1).as("max_" + element._1),min(element._1).as("min_" + element._1))
434         result = result.crossJoin(c_temp)
435       }
436     }
437   }
438
439 }
440
441
442   return result
443 }
444
445 def find_best_cluster(spark: SparkSession, json: DataFrame, clusters: DataFrame, r: DataFrame, cluster_size: Int,min_max_column: DataFrame): DataFrame = {
446   val cluster_union_r = r.crossJoin(min_max_column)
447   val subs_infoloss = calculate_subtraction_information_loss_optimize(json,cluster_union_r,clusters,cluster_size)
448   val subs_infoloss_ordered = subs_infoloss.orderBy(asc("Subs_IL")).limit(1)
449   val best = subs_infoloss_ordered.select("Cluster")
450   return best
451 }
452
453 }
```

C.4 Kelas KAnonymity

Kode program untuk kelas *KAnonymity* adalah sebagai berikut.

Listing C.4: KAnonymity.scala

```

1 package AnonymizationModel
2
3 import org.apache.hadoop.fs.{FileSystem, Path}
4 import org.apache.spark.sql.functions.udf
5 import org.apache.spark.sql.types.StructType
6 import org.apache.spark.sql.DataFrame, SparkSession
```

```

7
8 import scala.collection.mutable.ListBuffer
9
10 class KAnonymity {
11
12   def k_anonymity(spark: SparkSession, json: DataFrame, clusters: DataFrame, listBinaryTree: ListBuffer[BinaryTree], hdfs: FileSystem,
13     path_HDFS: String, path_delete_function_HDFS: String, sensitive_identifier: String): DataFrame ={
14     import org.apache.spark.sql.functions._
15
16     val clusters_schema: StructType = clusters.schema
17     var clusters_temp: DataFrame = null
18     var result: DataFrame = null
19
20     var numClusters = clusters.select("Cluster").distinct().count().toInt
21     this.delete_folder_hdfs(path_delete_function_HDFS+"/kanonymity1_tmp/",hdfs)
22     clusters.coalesce(1).write.option("header", "true")
23     csv(path_HDFS+"/kanonymity1_tmp/")
24
25 // Perulangan untuk setiap cluster
26 while(numClusters > 0){ //looping
27
28
29 // Baca clusters dari file HDFS
30 this.delete_folder_hdfs(path_delete_function_HDFS+"/kanonymity1/",hdfs)
31 hdfs.rename(new Path(path_HDFS+"/kanonymity1_tmp"),
32             new Path(path_HDFS+"/kanonymity1"))
33 clusters_temp = spark.read.format("csv").option("header", "true").schema(clusters_schema).
34             load(path_HDFS+"/kanonymity1/")
35
36 // Mengambil sebuah cluster dari clusters
37 val clusterName = "Cluster_" + numClusters
38 var clusterDF = clusters_temp.where(clusters_temp("Cluster").contains(clusterName))
39 val clusterDF_temp = clusterDF
40
41 this.delete_folder_hdfs(path_delete_function_HDFS+"/kanonymity2_tmp/",hdfs)
42 clusterDF.coalesce(1).write.format("parquet").save(path_HDFS+"/kanonymity2_tmp/")
43
44 // Melakukan iterasi untuk anonimisasi data
45 var i = 1
46 clusterDF.dtypes.foreach { element => // looping
47
48   if(element._1 != "id" && !element._1.contains("Cluster") && !element._1.contains("min_") && !element._1.contains("max_")
49     && element._1 != sensitive_identifier) {
50
51     if (element._2.contains("Integer")) {
52       // Membaca file HDFS
53       this.delete_folder_hdfs(path_delete_function_HDFS+"/kanonymity2/",hdfs)
54       hdfs.rename(new Path(path_HDFS+"/kanonymity2_tmp"),
55                   new Path(path_HDFS+"/kanonymity2"))
56       clusterDF = spark.read.parquet(path_HDFS+"/kanonymity2/")
57
58       // Menulis ke file HDFS
59       clusterDF = clusterDF.withColumn("Anonym_" + element._1, convert_to_numeric_anonymous(
60         col("max_" + element._1), col("min_" + element._1)))
61       clusterDF.coalesce(1).write.format("parquet").save(path_HDFS+"/kanonymity2_tmp/")
62
63     i += 1
64   } else {
65     val binaryTree = listBinaryTree(i)
66
67     // Membaca file HDFS
68     this.delete_folder_hdfs(path_delete_function_HDFS+"/kanonymity2/",hdfs)
69     hdfs.rename(new Path(path_HDFS+"/kanonymity2_tmp"),
70                 new Path(path_HDFS+"/kanonymity2"))
71     clusterDF = spark.read.parquet(path_HDFS+"/kanonymity2/")
72
73     // Menulis file ke HDFS
74     val distinctColumn = clusterDF.select(collect_set(element._1).as("Dist_" + element._1))
75
76     if (binaryTree != null) {
77       clusterDF = clusterDF.crossJoin(distinctColumn)
78       clusterDF = clusterDF.withColumn("Anonym_" + element._1,
79         convert_to_category_anonymous(binaryTree)(col("Dist_" + element._1)) )
80       clusterDF = clusterDF.drop("Dist_" + element._1)
81       clusterDF.coalesce(1).write.format("parquet").save(path_HDFS+"/kanonymity2_tmp/")
82
83     i += 1
84   } else {
85     clusterDF = clusterDF.crossJoin(distinctColumn)
86     clusterDF = clusterDF.withColumn("Anonym_" + element._1,
87       col("Dist_" + element._1).getItem(0) )
88     clusterDF = clusterDF.drop("Dist_" + element._1)
89     clusterDF.coalesce(1).write.format("parquet").save(path_HDFS+"/kanonymity2_tmp/")
90     i += 1
91   } // end else
92 } // end else
93
94 } // end if
95
96 } // end loop
97
98 // Membaca file HDFS
99 this.delete_folder_hdfs(path_delete_function_HDFS+"/kanonymity2/",hdfs)
100 hdfs.rename(new Path(path_HDFS+"/kanonymity2_tmp"),
101             new Path(path_HDFS+"/kanonymity2"))

```

```

104 clusterDF = spark.read.parquet(path_HDFS+ "/kanonymity2/")
105 val clusterDF_temp2 = clusterDF.select("id",sensitive_identifier,"Cluster").
106   withColumnRenamed("id","id_temp")
107
108 // Mengambil nama column diawali dengan "Anonym"
109 clusterDF = clusterDF.select(clusterDF.columns.
110   filter(colName => colName.startsWith("Anonym") ||
111   colName.contains("id"))).
112   map(clusterDF(_)) : _*
113
114 clusterDF = clusterDF.join(clusterDF_temp2,clusterDF("id") === clusterDF_temp2("id_temp"),"inner")
115 clusterDF = clusterDF.drop("id_temp")
116
117 // Menyimpan hasil anonimisasi
118 if(result == null) {
119   result = clusterDF
120   this.delete_folder_hdfs(path_delete_function_HDFS+ "/kanonymity3_tmp/",hdfs)
121   result.coalesce(1).write.option("header", "true").
122   csv(path_HDFS+ "/kanonymity3_tmp/")
123 }
124
125 else{
126   // Membaca result dari file HDFS
127   this.delete_folder_hdfs(path_delete_function_HDFS+ "/kanonymity3/",hdfs)
128   hdfs.rename(new Path(path_HDFS+ "/kanonymity3_tmp"),
129             new Path(path_HDFS+ "/kanonymity3"))
130   result = spark.read.format("csv").option("header", "true").
131   schema(result.schema).load(path_HDFS+ "/kanonymity3/")
132
133 // Menggabungkan result
134 result = result.union(clusterDF)
135 delete_folder_hdfs(path_delete_function_HDFS+ "/kanonymity3_tmp/",hdfs)
136 result.coalesce(1).write.option("header", "true").
137   csv(path_HDFS+ "/kanonymity3_tmp/")
138 }
139
140 // Menyimpan anonimisasi ke HDFS
141 numClusters = 1
142 if(numClusters == 0){
143   this.delete_folder_hdfs(path_delete_function_HDFS+ "/kanonymity3/",hdfs)
144   hdfs.rename(new Path(path_HDFS+ "/kanonymity3_tmp"),
145             new Path(path_HDFS+ "/kanonymity3"))
146 }
147
148 // Membuang cluster yang sudah pernah dianonimisasi
149 clusters_temp = clusters_temp.except(clusterDF_temp).cache()
150 clusters_temp.coalesce(1).write.option("header", "true").
151   csv(path_HDFS+ "/kanonymity1_tmp/")
152 if(numClusters == 0){
153   this.delete_folder_hdfs(path_delete_function_HDFS+ "/kanonymity1/",hdfs)
154   hdfs.rename(new Path(path_HDFS+ "/kanonymity1_tmp"),
155             new Path(path_HDFS+ "/kanonymity1"))
156 }
157
158 result = spark.read.format("csv").option("header", "true").schema(result.schema).
159   load(path_HDFS+ "/kanonymity3/")
160 result = result.orderBy(asc("id"))
161 result = result.drop("Cluster")
162
163 return result
164 }
165
166 def convert_to_numeric_anonymous = udf ( (maxValue: Int, minValue: Int) => {
167   if(maxValue == minValue){
168     maxValue.toString
169   }
170   else{
171     var maxValueTemp = maxValue.toString
172     var minValueTemp = minValue.toString
173     if(maxValue < 0){
174       maxValueTemp = "(" + maxValueTemp +
175     }
176     if(minValue < 0){
177       minValueTemp = "(" + minValueTemp +
178     }
179     "[" + minValueTemp + "-" + maxValueTemp + "]"
180   }
181 }
182 })
183
184 def convert_to_category_anonymous(binaryTree: BinaryTree) = udf ( (listCategory: Seq[String]) => {
185   if(listCategory.length > 1){
186     binaryTree.root.name
187   }
188   else if (listCategory.length == 1){
189     listCategory.head.toString()
190   }
191   else{
192     null
193   }
194 })
195
196 def delete_folder_hdfs(pathName: String,hdfs:FileSystem) {
197   val path = new Path(pathName)
198   if (hdfs.exists(path)) {
199     hdfs.delete(path, true)
200   }
201 }
```

```

203
204
205 def getNumPartitions(size:Int):Int = {
206   var numPartitions = 1
207   if(size >= 9) numPartitions = size/9
208   return numPartitions
209 }
210
211 }
```

C.5 Kelas LowestCommonAncestor

Kode program untuk kelas *LowestCommonAncestor* adalah sebagai berikut.

Listing C.5: LowestCommonAncestor.scala

```

1 package AnonymizationModel
2
3 import scala.collection.mutable.ListBuffer
4
5 class LowestCommonAncestor extends Serializable {
6   var path1: ListBuffer[String] = ListBuffer[String]()
7   var path2: ListBuffer[String] = ListBuffer[String]()
8
9   // untuk Binary Tree (numeric/categorical)
10  def findLCA(root: Node, n1:String, n2: String):String = {
11    path1.clear()
12    path2.clear()
13    return findLCAInternal(root,n1,n2)
14  }
15
16  def findLCAInternal(root: Node, n1: String, n2: String):String= {
17    if(!findPath(root,n1,path1) || !findPath(root,n2,path2)){
18      println(if(!path1.isEmpty) "n1_is_present" else "n1_is_missing")
19      println(if(!path2.isEmpty) "n2_is_present" else "n2_is_missing")
20      return "not_found"
21    }
22
23    import util.control.Breaks._
24    var i = 0
25    breakable {
26      while (i < path1.size && i < path2.size) {
27        if (!path1(i).equals(path2(i))) break
28        i += 1
29      }
30    }
31
32    return path1(i - 1)
33  }
34
35  def findPath(root:Node, n: String, path>ListBuffer[String]):Boolean={
36    if (root == null) {
37      return false;
38    }
39
40    path += root.name
41
42    if (root.name == n) {
43      return true;
44    }
45
46    if (root.left != null && findPath(root.left, n, path)) {
47      return true;
48    }
49
50    if (root.right != null && findPath(root.right, n, path)) {
51      return true;
52    }
53
54    path.remove(path.size - 1)
55
56    return false
57  }
58}
59 }
```

C.6 Kelas MainAnonymization

Kode program untuk kelas *MainAnonymization* adalah sebagai berikut.

Listing C.6: MainAnonymization.scala

```

1 package AnonymizationModel
2
3 import AnonymizationModel.MainClusterization.create_list_binary_tree_attribute
4 import org.apache.spark.sql.{DataFrame, Row, SparkSession}
5
```

```

6| import scala.collection.mutable.ListBuffer
7|
8| object MainAnonymization {
9|
10|   // val path_HDFS_CLUSTER = "hdfs://master:9070/skripsi-jordan/temp"
11|   // val path_HDFS_LOCAL = "hdfs://localhost:50071/skripsi"
12|   // val path_delete_HDFS_CLUSTER = "/skripsi-jordan/temp"
13|   // val path_delete_HDFS_LOCAL = "/skripsi"
14|
15|   def main(args:Array[String]): Unit = {
16|
17|     val spark = SparkSession.builder
18|       .master("local[*]")
19|       .appName("K-Anonymity")
20|       .getOrCreate()
21|
22|     // Parameter JSON
23|     val path_JSON = args(0)
24|     val json:DataFrame = spark.read.option("multiline", "true").json(path_JSON).cache() // ganti di sini
25|     val sensitive_identifier_json:ListBuffer[Seq[String]] = read_element_from_json(json, "sensitive_identifier")
26|
27|     // Parameter Greedy K-Member Clustering
28|     val hdfs_name = json.select("hdfs").first().getString(0)
29|     val temp_files = json.select("temp_files").first().getString(0)
30|     val output_path = json.select("output_path").first().getString(0)
31|     val sensitive_identifier = get_attribute_name_json(sensitive_identifier_json)(0)
32|
33|     // Inisialisasi path HDFS untuk membaca csv dan delete folder
34|     val path_HDFS = hdfs_name + temp_files
35|     val path_delete_function_HDFS = temp_files
36|
37|     // Membaca input k-anonymity dari HDFS
38|     val clusters = spark.read.parquet(path_HDFS+"/gkmc_output/")
39|     val hadoopConf = new org.apache.hadoop.conf.Configuration()
40|     val hdfs = org.apache.hadoop.fs.FileSystem.get(new java.net.URI(hdfs_name), hadoopConf)
41|
42|     // Melakukan anonymisasi pada data yang telah dikelompokan menggunakan k-anonymity
43|     val KAnonymity = new KAnonymity()
44|     val GKMC = clusters.select(clusters.columns.
45|       filter(colName => !colName.startsWith("max_") &&
46|         !colName.contains("min_")).map(clusters(_)) : _*)
47|     val listBinaryTree = create_list_binary_tree.attribute(GKMC, json)
48|     val kanonymityDF = KAnonymity.k_anonymity(spark, json, clusters, listBinaryTree, hdfs, path_HDFS, path_delete_function_HDFS,
49|       sensitive_identifier)
50|
51|     // Menyimpan hasil pengelompokan data ke dalam CSV
52|     kanonymityDF.coalesce(1)
53|       .write
54|         .option("header", "true")
55|         .option("sep", ",")
56|         .mode("overwrite")
57|         .csv(output_path+"k-anonymity")
58|
59|   def get_attribute_name_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
60|     var result = ListBuffer[String]()
61|     for(value <- values) {
62|       result += value(0)
63|     }
64|     return result
65|   }
66|
67|   def get_attribute_datatype_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
68|     var result = ListBuffer[String]()
69|     for(value <- values) {
70|       result += value(1)
71|     }
72|     return result
73|   }
74|
75|   def read_element_from_json(json: DataFrame, element: String): ListBuffer[Seq[String]] = {
76|     var result = ListBuffer[Seq[String]]()
77|
78|     try{
79|       val quasiIdentifier = json.select(element).collect()
80|       val quasiIdentifierArr = quasiIdentifier.map(row => row.getSeq[Row](0))
81|       quasiIdentifierArr.foreach(quasiIdentifierVariables => {
82|         quasiIdentifierVariables.map(row => {
83|           result += row.toSeq.asInstanceOf[Seq[String]]
84|         })
85|       })
86|       return result
87|     }
88|     catch {
89|       case x: Exception => {
90|         result = null
91|         return result
92|       }
93|     }
94|   }
95|
96| }
97|
98| }
```

C.7 Kelas MainClusterization

Kode program untuk kelas *MainClusterization* adalah sebagai berikut.

Listing C.7: MainClusterization.scala

```

1 package AnonymizationModel
2
3 import org.apache.hadoop.fs.{FileSystem, Path}
4 import org.apache.spark.sql.expressions.Window
5 import org.apache.spark.sql.types.{IntegerType, StringType}
6 import org.apache.spark.sql.{DataFrame, Row, SparkSession}
7
8 import scala.collection.mutable
9 import scala.collection.mutable.ListBuffer
10
11 object MainClusterization {
12
13     //    val path_HDFS_CLUSTER= "hdfs://master:9070/skripsi-jordan/temp"
14     //    val path_HDFS_LOCAL= "hdfs://localhost:50071/skripsi"
15     //    val path_delete_HDFS_CLUSTER= "/skripsi-jordan/temp"
16     //    val path_delete_HDFS_LOCAL= "/skripsi"
17
18     def main(args:Array[String]): Unit = {
19
20         val spark = SparkSession.builder
21             .master("local[*]")
22             .appName("GKMC")
23             .getOrCreate()
24
25         // Parameter JSON
26         val path_JSON = args(0)
27         val json:DataFrame = spark.read.option("multiline", "true").json(path_JSON).cache() // ganti di sini
28
29         // Parameter Greedy K-Member Clustering
30         val k = json.select("k").first().getLong(0).toInt
31         val num_sample_datas = json.select("num_sample_datas").first().getLong(0).toInt
32         val hdfs_name = json.select("hdfs").first().getString(0)
33         val temp_files = json.select("temp_files").first().getString(0)
34         val input_path = json.select("input_path").first().getString(0)
35         val output_path = json.select("output_path").first().getString(0)
36
37         // Konfigurasi HDFS
38         val hadoopConf = new org.apache.hadoop.conf.Configuration()
39         val hdfs = org.apache.hadoop.fs.FileSystem.get(new java.net.URI(hdfs_name), hadoopConf)
40
41         // Dataset dengan ID
42         val datasetInput = spark.read.format("csv").option("header", "true").load(input_path)
43         val columnSelectionWithID = generate_dataframe_from_csv(spark, json, datasetInput)
44
45         // Membatasi record yang dipakai untuk eksperimen
46         val S = columnSelectionWithID.where("id<=" + num_sample_datas).repartition(45).cache()
47         S.coalesce(1).write.option("header", "true").option("sep", ",")
48             mode("overwrite").csv(output_path+"normal-table")
49
50
51         S.rdd.mapPartitionsWithIndex((x,y) => {println(s"partitions_${x}_has_${y.length}_records");y.map(a => a+"")}).collect.foreach(
52             println)
53         // Inisialisasi path HDFS untuk membaca csv dan delete folder
54         val path_HDFS = hdfs_name + temp_files
55         val path_delete_function_HDFS = temp_files
56
57         // Melakukan pengelompokan data dengan algoritma Greedy k-member clustering
58         val GKMC = new GreedyKMemberClustering() // 4 menit
59         val listBinaryTree = create_list_binary_tree_attribute(S, json)
60         val gkmcDF = GKMC.greedy_k_member_clustering(spark, json, S, k, num_sample_datas, listBinaryTree,
61             hdfs, path_HDFS, path_delete_function_HDFS).cache()
62
63         // Menyimpan hasil pengelompokan data ke dalam CSV
64         val kanonymity_input = spark.read.format("csv").option("header", "true").
65             schema(gkmcDF.schema).load(path_HDFS+"/gkmc2/")
66         val gkmc_output = kanonymity_input.select(kanonymity_input.columns.
67             filter(colName => !colName.startsWith("max_") &&
68                 !colName.contains("min_")).map(kanonymity_input(_)) : _*)
69
70         // Menyimpan input k-anonymity di HDFS dan menyimpan pengelompokan data di local path
71         this.delete_folder_hdfs(path_HDFS+"/gkmc_output",hdfs)
72
73         kanonymity_input.coalesce(1).
74             write.format("parquet")
75             .save(path_HDFS+"/gkmc_output")
76
77         gkmc_output.coalesce(1).
78             write.
79             option("header", "true").
80             option("sep", ",").
81             mode("overwrite").
82             csv(output_path+"greedy-k-member-clustering")
83
84         S.unpersist()
85         json.unpersist()
86     }
87
88     def delete_folder_hdfs(pathName: String,hdfs:FileSystem) {
89         val path = new Path(pathName)
90         if (hdfs.exists(path)) {

```

```

91     hdfs.delete(path, true)
92   }
93 }
94
95 def read_dgh_from_json(json: DataFrame, category: String): ListBuffer[Seq[String]] = {
96   var result = ListBuffer[Seq[String]]()
97
98   try{
99     val treeName = json.select("domain_generalization_hierarchy." + category + ".tree").collect()(0).getString(0)
100    val dgh_json = json.select("domain_generalization_hierarchy." + category + ".generalization")
101    val dgh = dgh_json.collect()
102    val dghArr = dgh.map(row => row.getSeq[Row](0))
103    result += Seq[String](treeName)
104    dghArr.foreach(dgh_variables => {
105      dgh_variables.map(row => {
106        result += row.toSeq.asInstanceOf[Seq[String]]
107      })
108    })
109    return result
110  }
111  catch {
112    case x: Exception => {
113      result = null
114      return result
115    }
116  }
117 }
118 }
119
120 def create_list_binary_tree_attribute(S: DataFrame, json: DataFrame): ListBuffer[BinaryTree] = {
121   S.columns.foreach{colName =>
122     val dgh = read_dgh_from_json(json,colName)
123     if(dgh!=null){
124       val treeName = dgh.remove(0)(0)
125       listBinaryTree += create_binary_tree_from_dgh_attribute(dgh)
126     }
127     else{
128       listBinaryTree += null
129     }
130   }
131   return listBinaryTree
132 }
133 }
134
135 def create_binary_tree_from_dgh_attribute(dgh: ListBuffer[Seq[String]]):BinaryTree = {
136   val tree = new BinaryTree
137   val queue = new mutable.Queue[Node]()
138   var currentNode = new Node("",0)
139   var initialize = true
140
141   ////////////////////////////////baris ini diganti/////////////////////////////
142   dgh.foreach { attribute => // looping
143     val level = attribute(0).toInt
144     val parent = attribute(1)
145     val position = attribute(2)
146     val value = attribute(3).toString
147
148     if(queue.size == 0){
149       var newNode = new Node(value,level) // ngasih nilai
150       tree.root = newNode
151       queue.enqueue(newNode)
152     }
153     else {
154       if(parent != currentNode.name || initialize){
155         currentNode = queue.dequeue()
156       }
157       if(position == "left"){
158         currentNode.left = new Node(value,level)
159         queue.enqueue(currentNode.left)
160       }
161       else{
162         currentNode.right = new Node(value,level)
163         queue.enqueue(currentNode.right)
164       }
165       initialize = false
166     }
167   }
168   ////////////////////////////////baris ini diganti/////////////////////////////
169   return tree
170 }
171
172
173 def generate_dataframe_from_csv(spark:SparkSession, json:DataFrame, dataInput: DataFrame):DataFrame= {
174   val quasiIdentifier:ListBuffer[Seq[String]] = read_element_from_json(json,"quasi_identifier")
175   val sensitiveIdentifier:ListBuffer[Seq[String]] = read_element_from_json(json,"sensitive_identifier")
176
177   val quasiIdentifierAttribute:List[String] = get_attribute_name_json(quasiIdentifier).toList
178   val quasiIdentifierDatatype:List[String] = get_attribute_datatype_json(quasiIdentifier).toList
179
180   val sensitiveIdentifierAttribute = get_attribute_name_json(sensitiveIdentifier)
181   val sensitiveIdentifierDatatype = get_attribute_datatype_json(sensitiveIdentifier)
182
183   val selectedColumnAttribute:List[String] = quasiIdentifierAttribute ++ sensitiveIdentifierAttribute distinct
184   val selectedColumnDatatype = quasiIdentifierDatatype ++ sensitiveIdentifierDatatype
185
186   var columnSelectionDF = dataInput.select(selectedColumnAttribute.head,selectedColumnAttribute.tail: _*).toDF() // ganti di sini
187
188   // Buat bikin schema DataFrame

```

```

189     for(i <- 0 until selectedColumnAttribute.length){
190
191         val attrName = selectedColumnAttribute(i)
192         val datatype = selectedColumnDatatype(i)
193
194         if(datatype == "category"){
195             columnSelectionDF = columnSelectionDF.withColumn(attrName + "_new", columnSelectionDF(attrName).cast(StringType))
196             columnSelectionDF = columnSelectionDF.drop(attrName)
197         }
198         else{
199             columnSelectionDF = columnSelectionDF.withColumn(attrName + "_new", columnSelectionDF(attrName).cast(IntegerType))
200             columnSelectionDF = columnSelectionDF.drop(attrName)
201         }
202     }
203
204     val columnSelectionRenamedDF = columnSelectionDF.toDF(selectedColumnAttribute: _*)
205
206     import org.apache.spark.sql.functions._
207
208     val columnSelectionWithID = columnSelectionRenamedDF.withColumn("id_temp", row_number().over(Window.orderBy(
209         monotonically_increasing_id())))
210     var columnID = columnSelectionWithID.select("id_temp")
211     columnID = columnID.withColumnRenamed("id_temp", "id")
212     var result = columnID.join(columnSelectionWithID, columnID("id") === columnSelectionWithID("id_temp"), "inner")
213     result = result.drop("id_temp")
214
215     return result
216 }
217
218 def get_attribute_name_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
219     var result = ListBuffer[String]()
220     for(value <- values) {
221         result += value(0)
222     }
223     return result
224 }
225
226 def get_attribute_datatype_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
227     var result = ListBuffer[String]()
228     for(value <- values) {
229         result += value(1)
230     }
231     return result
232 }
233
234 def read_element_from_json(json: DataFrame, element: String): ListBuffer[Seq[String]] = {
235     var result = ListBuffer[Seq[String]]()
236
237     try{
238         val quasiIdentifier = json.select(element).collect()
239         val quasiIdentifierArr = quasiIdentifier.map(row => row.getSeq[Row](0))
240         quasiIdentifierArr.foreach(quasiIdentifierVariables => {
241             quasiIdentifierVariables.map(row => {
242                 result += row.toSeq.asInstanceOf[Seq[String]]
243             })
244         })
245         return result
246     }
247     catch {
248         case x: Exception => {
249             result = null
250             return result
251         }
252     }
253 }
254
255 }
256 }
```

C.8 Kelas MainLCATesting

Kode program untuk kelas *MainLCATesting* adalah sebagai berikut.

Listing C.8: MainLCATesting.scala

```

1 package AnonymizationModel
2
3 object MainLCATesting {
4     def main(args:Array[String]): Unit = {
5         val tree:BinaryTree = new BinaryTree() // Buat binary tree secara manual
6         tree.root = new Node("Person",1)
7         tree.root.left = new Node("Oriental",2)
8         tree.root.right = new Node("General",2)
9         tree.root.left.left = new Node("Amer-Indian-Eskimo",3)
10        tree.root.left.right = new Node("Asian",3)
11        tree.root.right.left = new Node("White",3)
12        tree.root.right.right = new Node("Black",3)
13        val lca = new LowestCommonAncestor() // Buat model Lowest Common Ancestor
14        val root = lca.findLCA(tree.root, "White", "Black")
15        println("LCA(4,5): " + root)
16        println("Level: " + tree.search(root).level)
17    }
18 }
```


LAMPIRAN D

KODE PROGRAM PERANGKAT LUNAK PENGUJIAN

D.1 Kelas KMeansModel

Kode program untuk kelas *KMeansModel* adalah sebagai berikut.

Listing D.1: KMeansModel.scala

```
1 package ExaminationModel
2
3 import org.apache.spark.ml.Pipeline
4 import org.apache.spark.ml.clustering.KMeans
5 import org.apache.spark.ml.evaluation.ClusteringEvaluator
6 import org.apache.spark.ml.feature.{OneHotEncoderEstimator, StringIndexer, VectorAssembler}
7 import org.apache.spark.sql.{DataFrame, SparkSession}
8
9 class KMeansModel {
10
11   def create_encoded_features_dataframe(table: DataFrame): DataFrame = {
12     // 1. Create label index and vector for each column
13     val cols = table.columns
14     val encodedFeatures = cols.flatMap { columnName =>
15       // Estimator: StringIndexer (create label index for each column from column value)
16       val stringIndexer = new StringIndexer()
17         .setInputCol(columnName)
18         .setOutputCol(columnName + "_Index")
19
20       // Transformer: OneHotEncoderEstimator (create vector for each column from label index)
21       val oneHotEncoder = new OneHotEncoderEstimator()
22         .setInputCols(Array(columnName + "Index"))
23         .setOutputCols(Array(columnName + "vec"))
24         .setDropLast(false)
25
26       Array(stringIndexer.setHandleInvalid("keep"), oneHotEncoder)
27     }
28
29     // 2. Pipeline: chains multiple Transformers and Estimators together to specify an ML workflow.
30     val pipeline = new Pipeline().setStages(encodedFeatures)
31     val indexer_model = pipeline.fit(table)
32
33     // 3. Create vector feature
34
35     // Example: Sparse Vector
36     val sparseVector = indexer_model.transform(table)
37
38     // Transformer: Vector Assembler (create feature vectors)
39     val vecFeatures = sparseVector.columns.filter(_.contains("Index"))
40     val vectorAssembler = new VectorAssembler().setInputCols(vecFeatures).setOutputCol("features")
41     val pipelineVectorAssembler = new Pipeline().setStages(Array(vectorAssembler))
42     val result_df = pipelineVectorAssembler.fit(sparseVector).transform(sparseVector) // hasil dataframe pipeline
43
44     return result_df
45   }
46
47
48   def model_training(table: DataFrame, json: DataFrame): DataFrame = {
49     val k = json.select("k_means.k").first().getLong(0).toInt
50     val kmeans = new KMeans().setK(k).setFeaturesCol("features").setPredictionCol("prediction")
51     val model = kmeans.fit(table)
52     val predictions = model.transform(table)
53     return predictions
54   }
55
56
57   def model_evaluation(predictions: DataFrame): Double = {
58     if (predictions.select("prediction").distinct().count() > 1) {
59       val evaluator = new ClusteringEvaluator()
60       val silhouette_score = evaluator.evaluate(predictions)
61       return silhouette_score
62     } else {
63       return 1
64     }
65   }
66
67 }
```

```

69 |     def comparing_model_evaluation_csv(spark:SparkSession,silhouetteScoreNormalTable:Double,silhouetteScoreAnonymizeTable:Double):
70 |         DataFrame_={
71 |             import spark.implicits._
72 |             val output = Seq(
73 |                 ("Normal_table",silhouetteScoreNormalTable),
74 |                 ("Anonymize_table",silhouetteScoreAnonymizeTable),
75 |                 ("How_much_different_is_Silhouette_score",Math.abs(silhouetteScoreNormalTable-silhouetteScoreAnonymizeTable))
76 |             ).toDF("Info","Silhouette_score")
77 |
78 |             return output
79 |         }
80 |     }

```

D.2 Kelas MainExamination

Kode program untuk kelas *MainExamination* adalah sebagai berikut.

Listing D.2: MainExamination.scala

```

1 package ExaminationModel
2
3
4 import AnonymizationModel.{BinaryTree, LowestCommonAncestor}
5 import org.apache.hadoop.fs.{Filesystem, Path}
6 import org.apache.spark.ml.classification.NaiveBayesModel
7 import org.apache.spark.sql.expressions.{UserDefinedFunction, Window}
8 import org.apache.spark.sql.functions...
9 import org.apache.spark.sql.types.{IntegerType, StringType}
10 import org.apache.spark.sql.{Column, DataFrame, Row, SparkSession}
11
12 import scala.collection.mutable.ListBuffer
13
14 object MainExamination {
15
16     def main(args:Array[String]): Unit = {
17         val spark = SparkSession
18             .builder.master("local[*]")
19             .appName("Examination")
20             .getOrCreate()
21
22         // Parameter Pengujian
23         val path_JSON = args(0)
24         val json: DataFrame = spark.read.option("multiline", "true").json(path_JSON).cache() // ganti di sini
25         val num_sample_datas = json.select("num_sample_data").first().getLong(0).toInt
26         val path_data_input_normal_table = json.select("input_path_normal").first().getString(0) // ganti di sini
27         val path_data_input_anonymize_table = json.select("input_path_anonymize").first().getString(0)
28         val path_data_input_cluster_table = json.select("input_path_cluster").first().getString(0)
29         val outputPath = json.select("output_path").first().getString(0)
30         val model_name = json.select("model_name").first().getString(0)
31
32         // K-Means
33         if(model_name.contains("k_means")){
34
35             // Membuat model k-means
36             val k_means_model = new KMeansModel()
37
38             ///////////////////////////////
39             // NORMAL TABLE K-MEANS //
40             //////////////////////////////
41
42             // Membaca file HDFS
43             val normalTable = spark.read.format("csv").option("header", "true").load(path_data_input_normal_table)
44
45             // Membuat dataframe dengan skema
46             val normalTableWithID = generate_dataframe_from_csv(spark,json,normalTable, false).
47                 where("id_<=" + num_sample_datas).cache()
48
49             // Edit column normalTableWithID
50             val features: Array[String] = json.select("k_means.features").first().getList(0).toArray().map(_.toString)
51             val normalTableWithID_edited = normalTableWithID.select(features.head, features.tail: _*)
52
53             // K-Means: Normal Table
54             val encodedDFNormalTable = k_means_model.create_encoded_features_dataframe(normalTableWithID_edited)
55             val startTimerKMNormal = System.currentTimeMillis()
56             val modelDFNormalTable = k_means_model.model_training(encodedDFNormalTable,json)
57             val endTimerKMNormal = System.currentTimeMillis()
58             val modelEvaluationNormalTable = k_means_model.model_evaluation(modelDFNormalTable)
59             val executionTimeKMNormal = ((endTimerKMNormal-startTimerKMNormal)/1000.0)
60
61             // List predictor Normal Table
62             val listPredictorAttributeNormalTable = list_of_predictor_attribute(spark,modelDFNormalTable)
63             val predictionDFNormalTable = modelDFNormalTable.select(listPredictorAttributeNormalTable.head,
64                         listPredictorAttributeNormalTable.tail: _*)
65
66             ///////////////////////////////
67             // ANONYM TABLE K-MEANS //
68             //////////////////////////////
69
70             // Membaca file HDFS
71             val anonymizeTable = spark.read.format("csv").option("header", "true").load(path_data_input_anonymize_table)

```

```

72 // Membuat dataframe dengan skema
73 val anonymizeTableWithID = anonymizeTable.where("id_<=" + num_sample_datas).cache()
74
75 // Edit column anonymizeTableWithID
76 val features2:Array[String] = json.select("k_means.features").first().getList(0).toArray().map(x=> "Anonym_" + x.toString)
77 val anonymizeTableWithID_edited = anonymizeTableWithID.select(features2.head, features2.tail: _*)
78
79 // K-Means: Anonymize Table
80 val encodedDFAnonymizeTable = k_means_model.create_encoded_features_dataframe(anonymizeTableWithID_edited)
81 val startTimerKMAuthorize = System.currentTimeMillis()
82 val modelDFAnonymizeTable = k_means_model.model_training(encodedDFAnonymizeTable, json)
83 val endTimerKMAuthorize = System.currentTimeMillis()
84 val modelEvaluationAnonymizeTable = k_means_model.model_evaluation(modelDFAnonymizeTable)
85 val executionTimeKMAuthorize = ((endTimerKMAuthorize - startTimerKMAuthorize) / 1000.0)
86
87 // List predictor Anonymize Table
88 val listPredictorAttributeAnonymizeTable = list_of_predictor_attribute(spark, modelDFAnonymizeTable)
89 val predictionDFAnonymizeTable = modelDFAnonymizeTable.select(listPredictorAttributeAnonymizeTable.head,
90 listPredictorAttributeAnonymizeTable.tail: _*)
91
92 /////////////////
93 // K-MEANS EVALUATION //
94 /////////////////
95
96 // K-Means: Time Evaluation
97 import spark.implicits._
98 val execTime = Seq(
99   ("Execution_time_(Normal)", executionTimeKNormal),
100  ("Execution_time_(Anonym)", executionTimeKMAuthorize)
101 ).toDF("Info", "Accuracy")
102
103 // K-Means: Model Evaluation
104 var model_evaluation = k_means_model.comparing_model_evaluation_csv(spark, modelEvaluationNormalTable,
105   modelEvaluationAnonymizeTable)
106 model_evaluation = model_evaluation.union(execTime)
107
108 // CSV
109 predictionDFNormalTable.coalesce(1) //Normal Table
110 .write
111 .option("header", "true")
112 .option("sep", ",")
113 .mode("overwrite")
114 .csv(outputPath+"normal-table")
115 predictionDFAnonymizeTable.coalesce(1) //Anonymize Table
116 .write
117 .option("header", "true")
118 .option("sep", ",")
119 .mode("overwrite")
120 .csv(outputPath+"anonymize-table")
121 model_evaluation.coalesce(1) //Silhouette Table
122 .write
123 .option("header", "true")
124 .option("sep", ",")
125 .mode("overwrite")
126 .csv(outputPath+"silhouette-score")
127 }
128 // Naive Bayes
129 else if(model_name.contains("naive_bayes")){
130
131 // Membuat model naive-bayes
132 val label_json = json.select("naive_bayes.label").first().getString(0)
133 val naive_bayes_model = new NaiveBayesModel()
134 val label = label_json + "_label"
135
136 /////////////////
137 // NORMAL TABLE NAIVE BAYES //
138 /////////////////
139
140 // Membaca file HDFS
141 val normalTable = spark.read.format("csv").option("header", "true").load(path_data_input_normal_table)
142
143 // Membuat dataframe dengan skema
144 var normalTableWithID = generate_dataframe_from_csv(spark, json, normalTable, false).
145   where("id_<=" + num_sample_datas).cache()
146
147 // Edit column normalTableWithID
148 var features:Array[String] = Array("id")
149 features = features ++ json.select("naive_bayes.features").first().getList(0).toArray().map(_.toString)
150 features = features :+ label_json
151 normalTableWithID = normalTableWithID.select(features.head, features.tail: _*)
152
153 // Check label is numeric or not
154 val dt:List[String] = normalTableWithID.select(label_json).dtypes.map(_._2).toList
155 val cek = dt.head.contains("Integer")
156 if(cek){
157   normalTableWithID = naive_bayes_model.create_label_column(spark, normalTableWithID, label_json, "")
158   normalTableWithID = normalTableWithID.drop(label_json)
159 }
160
161 //Naive Bayes: Normal Table
162 val normalTableWithoutLabel = normalTableWithID.drop(label)
163 val encodedDFNormalTable = naive_bayes_model.create_encoded_features_dataframe(normalTableWithID, normalTableWithoutLabel,
164   label, false)
165 val startTimerNBNormal = System.currentTimeMillis()
166 val modelDFNormalTable = naive_bayes_model.model_training(encodedDFNormalTable, json, label)
167 val endTimerNBNormal = System.currentTimeMillis()
168 val modelEvaluationNormalTable = naive_bayes_model.model_evaluation(modelDFNormalTable, json, label)

```

```

168  val executionTimeNBNormal = ((endTimerNBNormal-startTimerNBNormal)/1000.0)
169
170 ///////////////////////////////////////////////////////////////////
171 // ANONYMIZATION TABLE NAIVE BAYES //
172 ///////////////////////////////////////////////////////////////////
173
174 // Membaca file HDFS
175 val anonymizeTable = spark.read.format("csv").option("header", "true").load(path_data_input_anonymize_table)
176
177 // Membuat dataframe dengan skema
178 var anonymizeTableWithID = anonymizeTable.where("id_<=" + num_sample_datas).cache()
179
180 // Edit column anonymizeTableWithID
181 var features2:Array[String] = Array("id")
182 features2 = features2 ++ json.select("naive_bayes.features").first().getList(0).toArray().map(x=> "Anonym_" + x.toString)
183 features2 = features2 :+ label_json
184 anonymizeTableWithID = anonymizeTableWithID.select(features2.head, features2.tail: _*)
185
186 // Check label is numeric or not
187 if(cek){
188   anonymizeTableWithID = naive_bayes_model.create_label_column(spark,anonymizeTableWithID,label_json,"Anonym_")
189   anonymizeTableWithID = anonymizeTableWithID.drop(label_json)
190 }
191
192 //Naive Bayes: Anonymize Table
193 val anonymizeTableWithoutLabel = anonymizeTableWithID.drop("Anonym_" + label)
194 val encodedDFAnonymizeTable = naive_bayes_model.create_encoded_features_dataframe(anonymizeTableWithID,
195   anonymizeTableWithoutLabel,label,true)
196 val startTimerNBAnonymize = System.currentTimeMillis()
197 val modelDFAnonymizeTable = naive_bayes_model.model_training_anonym(encodedDFAnonymizeTable,json,label)
198 val endTimerNBAnonymize = System.currentTimeMillis()
199 val modelEvaluationAnonymizeTable = naive_bayes_model.model_evaluation_anonym(modelDFAnonymizeTable,json,label)
200 val executionTimeNBAnonymize = ((endTimerNBAnonymize-startTimerNBAnonymize)/1000.0)
201
202
203 ///////////////////////////////////////////////////////////////////
204 // NAIVE BAYES EVALUATION //
205 ///////////////////////////////////////////////////////////////////
206
207 // Naive Bayes: Time Evaluation
208 import spark.implicits._
209 val execTime = Seq(
210   ("Execution_time_(Normal)",executionTimeNBNormal),
211   ("Execution_time_(Anonym)",executionTimeNBAnonymize)
212 ).toDF("Info","Accuracy")
213
214 // Naive Bayes: Model Evaluation
215 var model_evaluation = naive_bayes_model.comparing_model_evaluation_csv(spark,modelEvaluationNormalTable,
216   modelEvaluationAnonymizeTable)
217 model_evaluation = model_evaluation.union(execTime)
218
219 // CSV
220 val outputPath = json.select("output_path").first().getString(0)
221 val pathModelNormal:String = json.select("naive_bayes.model_path_normal").first().getString(0)
222 val pathModelAnonym:String = json.select("naive_bayes.model_path_anonym").first().getString(0)
223 val loadModelNormalTable = NaiveBayesModel.load(pathModelNormal)
224 val loadModelAnonymTable = NaiveBayesModel.load(pathModelAnonym)
225
226 val normalPredictionTable = normalTableWithID.drop(label)
227 val anonymPredictionTable = anonymizeTableWithID.drop("Anonym_" + label)
228
229 val encodedNormalPredictionTable = naive_bayes_model.create_encoded_features_dataframe(normalTableWithID,
230   normalPredictionTable,label,false)
231 val encodedAnonymPredictionTable = naive_bayes_model.create_encoded_features_dataframe(anonymizeTableWithID,
232   anonymPredictionTable,label,true)
233
234 val predictionDFNormalTable = loadModelNormalTable.transform(encodedNormalPredictionTable).drop(label)
235 val predictionDFAnonymizeTable = loadModelAnonymTable.transform(encodedAnonymPredictionTable).drop("Anonym_" + label)
236
237 val predictionNormalCSV = predictionDFNormalTable.select(predictionDFNormalTable.columns.filter(colName =>
238   !colName.contains("_Index") && !colName.contains("_vec") &&
239   !colName.contains("rawPrediction") && !colName.contains("probability") &&
240   !colName.contains("id_temp") && !colName.contains("features")).map(predictionDFNormalTable(_)) : _*)
241
242 val predictionAnonymizeCSV = predictionDFAnonymizeTable.select(predictionDFAnonymizeTable.columns.filter(colName =>
243   !colName.contains("_Index") && !colName.contains("_vec") &&
244   !colName.contains("rawPrediction") && !colName.contains("probability") &&
245   !colName.contains("id_temp") && !colName.contains("features")).map(predictionDFAnonymizeTable(_)) : _*)
246
247 predictionNormalCSV.coalesce(1) //Normal Table
248   .write
249   .option("header", "true")
250   .option("sep", ",")
251   .mode("overwrite")
252   .csv(outputPath+"normal-table")
253 predictionAnonymizeCSV.coalesce(1) //Anonymize Table
254   .write
255   .option("header", "true")
256   .option("sep", ",")
257   .mode("overwrite")
258   .csv(outputPath+"anonymize-table")
259 model_evaluation.coalesce(1) //Accuracy
260   .write
261   .option("header", "true")
262   .option("sep", ",")

```

```

263     .mode("overwrite")
264     .csv(outputPath+"accuracy")
265   }
266   else if(model_name.contains("total_information_loss")){
267     // Membaca file HDFS dan membuat dataframe dengan skema
268     val clusterTable = spark.read.format("csv").option("header", "true").load(path_data_input_cluster_table)
269     val clusterTableWithID = generate_dataframe_from_csv(spark,json,clusterTable, true).
270       where("id_<=" +num_sample.datas).cache()
271
272   // Parameter Total Information Loss
273   var total_information_loss:Double = 0.0
274
275   // Parameter JSON
276   val path_json = json.select("total_information_loss.path_json").first().getString(0)
277   val json2 = spark.read.option("multiline", "true").json(path_json).cache()
278   val hdfs_name = json2.select("hdfs").first().getString(0)
279   val temp_files = json2.select("temp_files").first().getString(0)
280
281   // Inisialisasi path HDFS untuk membaca csv dan delete folder
282   val path_HDFS = hdfs_name + temp_files
283   val path_delete_function_HDFS = temp_files
284
285   // Membaca input k-anonymity dari HDFS
286   val hadoopConf = new org.apache.hadoop.conf.Configuration()
287   val hdfs = org.apache.hadoop.fs.FileSystem.get(new java.net.URI(hdfs_name), hadoopConf)
288
289   // Mengambil file GKMC
290   val clusters_schema = clusterTableWithID.schema
291
292   // Tulis clusters ke file HDFS
293   var numClusters = clusterTableWithID.select("Cluster").distinct().count().toInt
294   this.delete_folder_hdfs(path_delete_function_HDFS+"/totalIL1_tmp/",hdfs)
295   clusterTableWithID.coalesce(1).write.option("header", "true").
296     csv(path_HDFS+"/totalIL1_tmp/")
297
298   // Loop
299   while(numClusters > 0){
300
301     // Baca clusters dari file HDFS
302     this.delete_folder_hdfs(path_delete_function_HDFS+"/totalIL1/",hdfs)
303     hdfs.rename(new Path(path_HDFS+"/totalIL1_tmp"), new Path(path_HDFS+"/totalIL1"))
304     var clusters_temp = spark.read.format("csv").option("header", "true").
305       schema(clusters_schema).load(path_HDFS+"/totalIL1/")
306
307     // Mengambil sebuah cluster
308     val clusterName = "Cluster_" +numClusters
309     val clusterDF = clusters_temp.where(clusters_temp("Cluster").contains(clusterName))
310     val cluster_size = clusterDF.count().toInt
311     val clusterDF_temp = clusterDF
312
313     // Menulis cluster pada file HDFS
314     this.delete_folder_hdfs(path_delete_function_HDFS+"/totalIL2_tmp/",hdfs)
315     clusterDF.coalesce(1).write.option("header", "true").csv(path_HDFS+"/totalIL2_tmp/")
316
317     // Melakukan perhitungan Total Information Loss
318     total_information_loss += calculate_total_information_loss_optimize(json,clusterDF,cluster_size)
319
320     // Membuang cluster yang sudah pernah dianonimisasi
321     clusters_temp = clusters_temp.except(clusterDF_temp).cache()
322     clusters_temp.coalesce(1).write.option("header", "true").
323       csv(path_HDFS+"/totalIL1_tmp/")
324
325     if(numClusters == 0){
326       this.delete_folder_hdfs(path_delete_function_HDFS+"/totalIL1/",hdfs)
327       hdfs.rename(new Path(path_HDFS+"/totalIL1_tmp"), new Path(path_HDFS+"/totalIL1"))
328     }
329
330     numClusters -= 1
331
332   }
333
334   import spark.implicits._
335
336   val result = Seq((total_information_loss)).toDF("total_information_loss")
337
338   result.coalesce(1) //Accuracy
339     .write
340     .option("header", "true")
341     .option("sep", ",")
342     .mode("overwrite")
343     .csv(outputPath+"total-infloss")
344
345   }
346   else if(model_name.contains("perbedaan_hasil_clustering")){
347     val path_input_normal_clustering = json.select("perbedaan_hasil_clustering.path_input_normal_clustering").first().getString(0)
348     val path_input_anonym_clustering = json.select("perbedaan_hasil_clustering.path_input_anonym_clustering").first().getString(0)
349
350     var normal_clustering = spark.read.format("csv").option("header", "true").load(path_input_normal_clustering)
351     var anonym_clustering = spark.read.format("csv").option("header", "true").load(path_input_anonym_clustering)
352
353     normal_clustering = normal_clustering.withColumn("id", row_number().over(Window.orderBy(monotonically_increasing_id())))
354     anonym_clustering = anonym_clustering.withColumn("id", row_number().over(Window.orderBy(monotonically_increasing_id())))
355
356     normal_clustering = normal_clustering.select("id", "prediction")
357     anonym_clustering = anonym_clustering.select("id", "prediction").
358       withColumnRenamed("id", "id2").
359

```

```

360         withColumnRenamed("prediction", "prediction2")
361
362     var num_diff = 0
363     var numClusters = normal_clustering.select("prediction").distinct().count().toInt-1
364     while(numClusters >= 0) { //looping
365       val x = normal_clustering.filter(normal_clustering("prediction") === numClusters)
366       val x1 = x.join(anonym_clustering,x("id") === anonym_clustering("id2"), "inner")
367       val x2 = x1.groupBy("prediction2").count()
368       val x4 = x3.select("count").first().getLong(0).toInt
369       val x5 = x3.select(sum("count")).first().getLong(0).toInt
370       num_diff += x5-x4
371       numClusters -= 1
372     }
373
374
375     import spark.implicits._
376     val persentase_perbedaan = num_diff*1.0/num_sample_datas
377     val result = Seq((persentase_perbedaan)).toDF("Percentage_of_Clustering_Difference")
378
379 //     var result = normal_clustering.join(anonym_clustering, normal_clustering("id") === anonym_clustering("id2"), "inner")
380 //     result = result.withColumn("diff", isValueSimilar(col("prediction"), col("prediction2")))
381 //     result = result.filter(col("diff").contains(true)).select( (count("diff")*1.0/num_sample_datas).as("Percentage of
382 //     Clustering Difference"))
383
383     result.coalesce(1) //Accuracy
384       .write
385         .option("header", "true")
386         .option("sep", ",")
387         .mode("overwrite")
388         .csv(outputPath+"percentage-clustering-difference")
389   }
390   else if(model_name.contains("perbedaan_hasil_klasifikasi")){
391     val path_input_normal_classification = json.select("perbedaan_hasil_klasifikasi.path_input_normal_classification").first().
392       getString(0)
393     val path_input_anonym_classification = json.select("perbedaan_hasil_klasifikasi.path_input_anonym_classification").first().
394       getString(0)
395
396     var normal_classification = spark.read.format("csv").option("header", "true").load(path_input_normal_classification)
397     var anonym_classification = spark.read.format("csv").option("header", "true").load(path_input_anonym_classification)
398
399     normal_classification = normal_classification.withColumn("id", row_number().over(Window.orderBy(monotonically_increasing_id
400       ())))
401     anonym_classification = anonym_classification.withColumn("id", row_number().over(Window.orderBy(monotonically_increasing_id
402       ())))
403
404     normal_classification = normal_classification.select("id", "prediction")
405     anonym_classification = anonym_classification.select("id", "prediction").
406       withColumnRenamed("id", "id2").
407       withColumnRenamed("prediction", "prediction2")
408
409     var result = normal_classification.join(anonym_classification, normal_classification("id") === anonym_classification("id2"),
410       "inner")
411     result = result.withColumn("diff", isValueSimilar(col("prediction"), col("prediction2")))
412     result = result.filter(col("diff").contains(true)).select( (count("diff")/num_sample_datas).as("Percentage_of_Classification
413       _Difference"))
414
415     result.coalesce(1) //Accuracy
416       .write
417         .option("header", "true")
418         .option("sep", ",")
419         .mode("overwrite")
420         .csv(outputPath+"percentage-classification-difference")
421   }
422
423 def isValueSimilar = udf ( (value1: String,value2: String) => {
424   value1 != value2
425 })
426
427 def list_of_predictor_attribute(spark: SparkSession, record: DataFrame): List[String] = {
428   record.createOrReplaceTempView("tAdults")
429   val test = spark.catalog.listColumns("tAdults").select("name", "datatype")
430   val df = test.select("name").
431     where("datatype like 'string'" +
432       " or datatype like 'int'" +
433       " or datatype like 'double'" +
434       " and name not like '%Index%'")
435   val columnName = df.collect().map(_(0)).toList.asInstanceOf[List[String]]
436
437   return columnName
438 }
439
440 def generate_dataframe_from_csv(spark:SparkSession, json:DataFrame, dataInput: DataFrame, isCluster: Boolean):DataFrame={
441   val selectedColumn:ListBuffer[Seq[String]] = read_element_from_json(json, 'selected_column')
442   val selectedColumnAttribute:List[String] = get_attribute_name_json(selectedColumn).toList
443   val selectedColumnDatatype:List[String] = get_attribute_datatype_json(selectedColumn).toList
444
445   var columnSelectionDF = dataInput.select(selectedColumnAttribute.head,selectedColumnAttribute.tail: _*).toDF() // ganti di
446   sini
447
448   for(i <- 0 until selectedColumnAttribute.length){
449     val attrName = selectedColumnAttribute(i)
450     val datatype = selectedColumnDatatype(i)

      if(datatype == "category"){
        columnSelectionDF = columnSelectionDF.withColumn(attrName+"_new", columnSelectionDF(attrName).cast(StringType))
      }
    }
  }
}

```

```

451     columnSelectionDF = columnSelectionDF.drop(attrName)
452   }
453   else{
454     columnSelectionDF = columnSelectionDF.withColumn(attrName+"_new",columnSelectionDF(attrName).cast(IntegerType))
455     columnSelectionDF = columnSelectionDF.drop(attrName)
456   }
457 }
458 }
459 val columnSelectionRenamedDF = columnSelectionDF.toDF(selectedColumnAttribute: _*)
460
461 import org.apache.spark.sql.functions.-
462
463 val columnSelectionWithID = columnSelectionRenamedDF.withColumn("id_temp", row_number().over(Window.orderBy(
464   monotonically_increasing_id())))
465 var columnID = columnSelectionWithID.select("id_temp")
466 columnID = columnID.withColumnRenamed("id_temp","id")
467 var result = columnID.join(columnSelectionWithID, columnID("id") === columnSelectionWithID("id_temp"),"inner")
468 result = result.drop("id_temp")
469
470 if(isCluster){
471   val columnCluster = dataInput.select("id","Cluster").withColumnRenamed("id","id_temp")
472   result = result.join(columnCluster,result("id")===columnCluster("id_temp"),"inner")
473   result = result.drop("id_temp")
474 }
475
476 return result
477 }
478
479
480
481 def get_attribute_name_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
482   var result = ListBuffer[String]()
483   for(value <- values) {
484     result += value(0)
485   }
486   return result
487 }
488
489 def get_anonymize_attribute_name_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
490   var result = ListBuffer[String]()
491   for(value <- values) {
492     result += "Anonym_" + value(0)
493   }
494   return result
495 }
496
497 def get_attribute_datatype_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
498   var result = ListBuffer[String]()
499   for(value <- values) {
500     result += value(1)
501   }
502   return result
503 }
504
505 def get_anonymize_attribute_datatype_json(values: ListBuffer[Seq[String]]): ListBuffer[String] ={
506   var result = ListBuffer[String]()
507   for(value <- values) {
508     result += "Anonym_" + value(1)
509   }
510   return result
511 }
512
513 def read_element_from_json(json: DataFrame,element: String): ListBuffer[Seq[String]] = {
514   var result = ListBuffer[Seq[String]]()
515
516   try{
517     val selectedColumn = json.select(element).collect()
518     val selectedColumnArr = selectedColumn.map(row => row.getSeq[Row](0))
519     selectedColumnArr.foreach(selectedColumnVariables => {
520       selectedColumnVariables.map(row => {
521         result += row.toSeq.asInstanceOf[Seq[String]]
522       })
523     })
524     return result
525   }
526   catch {
527     case x: Exception =>
528       result = null
529       return result
530   }
531 }
532
533 }
534
535 def calculate_total_information_loss_optimize(json: DataFrame, cluster: DataFrame, cluster_size:Int): Double = {
536   if(!cluster.isEmpty){
537
538     val check = cluster.drop("id").dtypes.filter(element => element._2.startsWith("Integer"))
539     var cluster_temp:DataFrame = null
540
541     if(check.length == 0){
542       cluster_temp = cluster
543     }
544     else{
545       cluster_temp = min_max_cluster(cluster)
546       cluster_temp = cluster_temp.na.fill(0)
547     }
548   }

```

```

549     cluster.dtypes.foreach(element =>
550       if(!element._1.startsWith("id") && !element._1.startsWith("Cluster")) {
551         if(element._2.contains("Integer")){
552           cluster_temp = cluster_temp.withColumn("IL_" + element._1, calculateNumericalInformationLossUnion(cluster_size)(col("max_" + element._1), col("min_" + element._1)))
553         }
554         else{
555           cluster_temp = cluster_temp.withColumn("IL_" + element._1, lit(1)*1.0)
556         }
557       }
558     )
559
560   val infoloss_union = cluster_temp.select(cluster_temp.columns.filter(_.contains("IL_")).map(cluster_temp(_)): _*)
561
562   val sum_infoloss_union = infoloss_union.columns.toSeq.map(col _)
563   val x = cluster_temp.withColumn("TotalIL", sum_(sum_infoloss_union: _*) * cluster_size * 1.0)
564
565 //   cluster_temp = cluster_temp.drop(cluster_temp.columns.filter(_.startsWith("IL")): _*)
566
567   var result:Double = 0
568
569   result = x.select("TotalIL").first().getDouble(0)
570
571   if(result == 0 || result == null ){
572     print()
573   }
574
575   return result
576 }
577 else{
578   return 0
579 }
580
581
582 }
583
584 def calculateNumericalInformationLossUnion(cluster_size:Int) = udf ( (max_c:Int, min_c:Int) => {
585   Math.abs(max_c-min_c)*1.0/cluster_size
586 })
587
588 def calculateCategoricalDistance(binaryTree: BinaryTree):UserDefinedFunction = udf( (category1: String, category2: String) => {
589   if(binaryTree!=null){
590     val node1 = binaryTree.search(category1)
591     val node2 = binaryTree.search(category2)
592     if(node1 != null && node2 != null && node1.name != node2.name){
593       val LCA = new LowestCommonAncestor()
594       val LCA_root_name = LCA.findLCA(binaryTree.root, node1.name, node2.name)
595       val H_subtree = binaryTree.search(LCA_root_name).level
596       val H_TD = binaryTree.getHeight(binaryTree.root).toDouble
597       BigDecimal(H_subtree*1.0 / H_TD).setScale(2, BigDecimal.RoundingMode.HALF_UP).toDouble
598     }
599     else{
600       1.0
601     }
602   }
603   else{
604     1.0
605   }
606 }
607 )
608
609 def sum_(cols: Column*) = cols.foldLeft(lit(0))(_ + _)
610
611 def min_max_cluster(c: DataFrame):DataFrame = {
612   var result:DataFrame = null
613   c.dtypes.filter(element => element._2 == "IntegerType").foreach{element =>
614     if(element._1 != "id" && element._2.contains("Integer")){
615
616       if(result == null){
617         result = c.select(max(element._1).as("max_" + element._1),min(element._1).as("min_" + element._1))
618       }
619       else{
620         val c_temp = c.select(max(element._1).as("max_" + element._1),min(element._1).as("min_" + element._1))
621         result = result.crossJoin(c_temp)
622       }
623     }
624   }
625
626   return result
627 }
628 }
629
630
631 def delete_folder_hdfs(pathName: String,hdfs:FileSystem) {
632   val path = new Path(pathName)
633   if (hdfs.exists(path)) {
634     hdfs.delete(path, true)
635   }
636 }
637
638
639 }
640

```

D.3 Kelas NaiveBayesModel

Kode program untuk kelas *NaiveBayesModel* adalah sebagai berikut.

Listing D.3: NaiveBayesModel.scala

```

1 package ExaminationModel
2
3 import org.apache.spark.ml.Pipeline
4 import org.apache.spark.ml.classification.NaiveBayes
5 import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
6 import org.apache.spark.ml.feature.{OneHotEncoderEstimator, StringIndexer, VectorAssembler}
7 import org.apache.spark.sql.functions.udf
8 import org.apache.spark.sql.{DataFrame, SparkSession}
9
10 class NaiveBayesModel {
11   def create_encoded_features_dataframe(anonymizeTable: DataFrame, anonymizeTableWithoutLabel: DataFrame, label: String, isAnonym: Boolean): DataFrame = {
12     // 1. Create label index and vector for each column
13     val cols = anonymizeTableWithoutLabel.drop("id").columns
14     val encodedFeatures = cols.flatMap { columnName =>
15       // Estimator: StringIndexer (create label index for each column from column value)
16       val stringIndexer = new StringIndexer().setInputCol(columnName).setOutputCol(columnName + "_Index")
17
18       // Transformer: OneHotEncoderEstimator (create vector for each column from label index)
19       val oneHotEncoder = new OneHotEncoderEstimator().setInputCols(Array(columnName + "_Index"))
20         .setOutputCols(Array(columnName + ".vec"))
21         .setDropLast(false)
22
23       Array(stringIndexer.setHandleInvalid("keep"), oneHotEncoder)
24
25     }
26
27   }
28
29
30   // 2. Pipeline: chains multiple Transformers and Estimators together to specify an ML workflow.
31   val pipeline = new Pipeline().setStages(encodedFeatures)
32   val indexer_model = pipeline.fit(anonymizeTableWithoutLabel)
33
34   // 3. Create vector feature
35
36   // Example: Sparse Vector
37   val sparseVector = indexer_model.transform(anonymizeTableWithoutLabel)
38
39   // Transformer: Vector Assembler (create feature vectors)
40   val vecFeatures = sparseVector.columns.filter(_.contains("Index"))
41   val vectorAssembler = new VectorAssembler().setInputCols(vecFeatures).setOutputCol("features")
42   val pipelineVectorAssembler = new Pipeline().setStages(Array(vectorAssembler))
43   var result_df = pipelineVectorAssembler.fit(sparseVector).transform(sparseVector) // hasil dataframe pipeline
44
45   var name = label
46   if(isAnonym){
47     name = "Anonym_" + label
48   }
49
50   val x = anonymizeTable.select("id", name).withColumnRenamed("id", "id_temp")
51   val indexer = new StringIndexer().setInputCol(name).setOutputCol(name + "_Index").fit(x)
52   val y = indexer.transform(x)
53
54   result_df = result_df.join(y, result_df("id") === x("id_temp"))
55   result_df = result_df.drop("id", "id_temp")
56
57   return result_df
58 }
59
60
61 def create_label_column(spark: SparkSession, table: DataFrame, label: String, name: String): DataFrame = {
62   table.createOrReplaceTempView("tCredit")
63   val median = spark.sql("SELECT CAST(percentile_approx("+label+", 0.5) AS DOUBLE) AS Q2 FROM tCredit").first().getDouble(0).
64     toInt
65   val result = table.withColumn(name+label+"_label", getLabelFromNumeric(median)(table(label)))
66   return result
67 }
68
69 def getLabelFromNumeric(median: Int) = udf((num: Int) => {
70   if(num <= median){
71     "<=" + median/1000 + "k"
72   } else{
73     ">" + median/1000 + "k"
74   }
75 })
76
77 def model_training(table: DataFrame, json: DataFrame, label: String): DataFrame = {
78   val trainingSet = json.select("naive_bayes.training_set").first().getDouble(0)
79   val testSet = json.select("naive_bayes.test_set").first().getDouble(0)
80   val pathModel: String = json.select("naive_bayes.model_path_normal").first().getString(0)
81
82   val Array(training, test) = table.randomSplit(Array(trainingSet, testSet))
83   val model = new NaiveBayes().setModelType("multinomial").setLabelCol(label + "_Index").fit(training)
84   model.write.overwrite.save(pathModel) // tulis model ke file
85
86   val predictions = model.transform(test)
87   return predictions
88 }
89

```

```
90 |  
91 | def model_training_anonym(table: DataFrame, json: DataFrame,label:String):DataFrame={  
92 |     val trainingSet = json.select("naive_bayes.training_set").first().getDouble(0)  
93 |     val testSet = json.select("naive_bayes.test_set").first().getDouble(0)  
94 |     val pathModel:String = json.select("naive_bayes.model_path_anonym").first().getString(0)  
95 |  
96 |     val Array(training, test) = table.randomSplit(Array(trainingSet, testSet))  
97 |     val model = new NaiveBayes().setModelType("multinomial").setLabelCol("Anonym_"+label+"_Index").fit(training)  
98 |     model.write.overwrite.save(pathModel)  
99 |  
100|     val predictions = model.transform(test)  
101|     return predictions  
102| }  
103|  
104| def model_evaluation(predictions:DataFrame, json: DataFrame,label:String):Double={  
105|     val evaluator = new MulticlassClassificationEvaluator()  
106|         .setLabelCol(label+"_Index")  
107|         .setPredictionCol("prediction")  
108|         .setMetricName("accuracy")  
109|     val accuracy = evaluator.evaluate(predictions)  
110|     return accuracy  
111| }  
112|  
113| def model_evaluation_anonym(predictions:DataFrame, json: DataFrame,label:String):Double={  
114|     val evaluator = new MulticlassClassificationEvaluator()  
115|         .setLabelCol("Anonym_"+label+"_Index")  
116|         .setPredictionCol("prediction")  
117|         .setMetricName("accuracy")  
118|     val accuracy = evaluator.evaluate(predictions)  
119|     return accuracy  
120| }  
121|  
122| def comparing_model_evaluation_csv(spark:SparkSession,accuracyNormalTable:Double,accuracyAnonymizeTable:Double):DataFrame={  
123|     import spark.implicits._  
124|     val output = Seq(  
125|         ("Normal_table",accuracyNormalTable),  
126|         ("Anonymize_table",accuracyAnonymizeTable),  
127|         ("Diff_accuracy",Math.abs(accuracyNormalTable-accuracyAnonymizeTable))  
128|     ).toDF("Info","Accuracy")  
129|  
130|     return output  
131| }  
132|  
133| }
```

LAMPIRAN E

MASUKAN FILE JSON SELURUH PERANGKAT LUNAK

E.1 Masukan JSON Perangkat Lunak Eksplorasi

Berikut masukan (.json) untuk perangkat lunak eksplorasi.

Listing E.1: Data JSON untuk Perangkat Lunak Eksplorasi

```
1 {
2     "input_path": "D:/input/credit430k.csv",
3     "output_path": "D:/output/exploratory-analysis/",
4     "selected_column": [
5         {
6             "attrName": "CODE_GENDER",
7             "dataType": "category"
8         },
9         {
10            "attrName": "OCCUPATION_TYPE",
11            "dataType": "category"
12        },
13        {
14            "attrName": "NAME_INCOME_TYPE",
15            "dataType": "category"
16        },
17        {
18            "attrName": "NAME_EDUCATION_TYPE",
19            "dataType": "category"
20        }
21    ]
22 }
```

E.2 Masukan JSON Perangkat Lunak Anonimisasi

Berikut masukan (.json) untuk perangkat lunak anonimisasi.

Listing E.2: Data JSON untuk Perangkat Lunak Eksplorasi

```
1 {
2     "k": 2,
3     "num_sample_datas": 5,
4     "input_path": "D:/input/adult100k.csv",
5     "output_path": "D:/output/",
6     "identifier": [
7         {
```

```
8         "attrName": "ID",
9         "dataType": "category"
10    }
11 ],
12 "sensitive_identifier": [
13   {
14     "attrName": "AMT_INCOME_TOTAL",
15     "dataType": "numeric"
16   }
17 ],
18 "quasi_identifier": [
19   {
20     "attrName": "DAYS_BIRTH",
21     "dataType": "numeric"
22   },
23   {
24     "attrName": "DAYS_EMPLOYED",
25     "dataType": "numeric"
26   },
27   {
28     "attrName": "OCCUPATION_TYPE",
29     "dataType": "category"
30   },
31   {
32     "attrName": "CODE_GENDER",
33     "dataType": "category"
34   }
35 ],
36 "domain_generalization_hierarchy": {
37   "NAME_EDUCATION_TYPE":{
38     "tree": "n-ary-tree",
39     "generalization": [
40       {
41         "value": "Educated",
42         "parent": "Person",
43         "level": "2",
44         "position": "root"
45       },
46       {
47         "value": "Academic degree",
48         "parent": "Person",
49         "level": "1",
50         "position": "child"
51       },
52       {
53         "value": "Incomplete higher",
54         "parent": "Person",
55         "level": "1",
56         "position": "child"
57       },
58       {
```

```
59         "value": "Secondary / secondary special",
60         "parent": "Person",
61         "level": "1",
62         "position": "child"
63     },
64     {
65         "value": "Lower secondary",
66         "parent": "Person",
67         "level": "1",
68         "position": "child"
69     },
70     {
71         "value": "Higher education",
72         "parent": "Person",
73         "level": "1",
74         "position": "child"
75     }
76   ]
77 },
78 "CODE_GENDER": {
79   "tree": "n-ary-tree",
80   "generalization": [
81     {
82       "value": "Adult",
83       "parent": "Adult",
84       "level": "2",
85       "position": "null"
86     },
87     {
88       "value": "M",
89       "parent": "Adult",
90       "level": "1",
91       "position": "left"
92     },
93     {
94       "value": "F",
95       "parent": "Adult",
96       "level": "1",
97       "position": "right"
98     }
99   ]
100 },
101 "NAME_INCOME_TYPE": {
102   "tree": "n-ary-tree",
103   "generalization": [
104     {
105       "value": "Unique earnings",
106       "parent": "Unique earnings",
107       "level": "2",
108       "position": "root"
109     },

```

```
110    {
111        "value": "Student",
112        "parent": "Unique earnings",
113        "level": "1",
114        "position": "child"
115    },
116    {
117        "value": "Commercial associate",
118        "parent": "Unique earnings",
119        "level": "1",
120        "position": "child"
121    },
122    {
123        "value": "State servant",
124        "parent": "Unique earnings",
125        "level": "1",
126        "position": "child"
127    },
128    {
129        "value": "Working",
130        "parent": "Unique earnings",
131        "level": "1",
132        "position": "child"
133    },
134    {
135        "value": "Pensioner",
136        "parent": "Unique earnings",
137        "level": "1",
138        "position": "child"
139    }
140 ]
141 },
142 "NAME_HOUSING_TYPE": {
143     "tree": "n-ary-tree",
144     "generalization": [
145         {
146             "value": "Unique housing",
147             "parent": "Unique housing",
148             "level": "2",
149             "position": "root"
150         },
151         {
152             "value": "House / apartment",
153             "parent": "Unique housing",
154             "level": "1",
155             "position": "child"
156         },
157         {
158             "value": "Municipal apartment",
159             "parent": "Unique housing",
160             "level": "1",
```

```
161         "position": "child"
162     },
163     {
164         "value": "Co-op apartment",
165         "parent": "Unique housing",
166         "level": "1",
167         "position": "child"
168     },
169     {
170         "value": "Rented apartment",
171         "parent": "Unique housing",
172         "level": "1",
173         "position": "child"
174     },
175     {
176         "value": "Office apartment",
177         "parent": "Unique housing",
178         "level": "1",
179         "position": "child"
180     },
181     {
182         "value": "With parents",
183         "parent": "Unique housing",
184         "level": "1",
185         "position": "child"
186     }
187 ]
188 },
189 "NAME_FAMILY_STATUS": {
190     "tree": "n-ary-tree",
191     "generalization": [
192         {
193             "value": "Unique status",
194             "parent": "Unique status",
195             "level": "2",
196             "position": "root"
197         },
198         {
199             "value": "Separated",
200             "parent": "Unique status",
201             "level": "1",
202             "position": "child"
203         },
204         {
205             "value": "Married",
206             "parent": "Unique status",
207             "level": "1",
208             "position": "child"
209         },
210         {
211             "value": "Single / not married",
```

```
212         "parent": "Unique status",
213         "level": "1",
214         "position": "child"
215     },
216     {
217         "value": "Widow",
218         "parent": "Unique status",
219         "level": "1",
220         "position": "child"
221     },
222     {
223         "value": "Civil marriage",
224         "parent": "Unique status",
225         "level": "1",
226         "position": "child"
227     }
228 ]
229 },
230 "OCCUPATION_TYPE": {
231     "tree": "n-ary-tree",
232     "generalization": [
233         {
234             "value": "Employee",
235             "parent": "Employee",
236             "level": "2",
237             "position": "root"
238         },
239         {
240             "value": "Managers",
241             "parent": "Employee",
242             "level": "1",
243             "position": "child"
244         },
245         {
246             "value": "HR staff",
247             "parent": "Employee",
248             "level": "1",
249             "position": "child"
250         },
251         {
252             "value": "Medicine staff",
253             "parent": "Employee",
254             "level": "1",
255             "position": "child"
256         },
257         {
258             "value": "Accountants",
259             "parent": "Employee",
260             "level": "1",
261             "position": "child"
262         },
```

```
263    {
264        "value": "Laborers",
265        "parent": "Employee",
266        "level": "1",
267        "position": "child"
268    },
269    {
270        "value": "Cleaning staff",
271        "parent": "Employee",
272        "level": "1",
273        "position": "child"
274    },
275    {
276        "value": "Private service staff",
277        "parent": "Employee",
278        "level": "1",
279        "position": "child"
280    },
281    {
282        "value": "Drivers",
283        "parent": "Employee",
284        "level": "1",
285        "position": "child"
286    },
287    {
288        "value": "Sales staff",
289        "parent": "Employee",
290        "level": "1",
291        "position": "child"
292    },
293    {
294        "value": "Realty agents",
295        "parent": "Employee",
296        "level": "1",
297        "position": "child"
298    },
299    {
300        "value": "IT staff",
301        "parent": "Employee",
302        "level": "1",
303        "position": "child"
304    },
305    {
306        "value": "Security staff",
307        "parent": "Employee",
308        "level": "1",
309        "position": "child"
310    },
311    {
312        "value": "Secretaries",
313        "parent": "Employee",
```

```

314        "level": "1",
315        "position": "child"
316    },
317    {
318        "value": "Low-skill Laborers",
319        "parent": "Employee",
320        "level": "1",
321        "position": "child"
322    },
323    {
324        "value": "Core staff",
325        "parent": "Employee",
326        "level": "1",
327        "position": "child"
328    },
329    {
330        "value": "Cooking staff",
331        "parent": "Employee",
332        "level": "1",
333        "position": "child"
334    },
335    {
336        "value": "High skill tech staff",
337        "parent": "Employee",
338        "level": "1",
339        "position": "child"
340    },
341    {
342        "value": "Waiters/barmen staff",
343        "parent": "Employee",
344        "level": "1",
345        "position": "child"
346    }
347 ]
348 }
349 }
350 }
```

E.3 Masukan JSON Perangkat Lunak Pengujian

Berikut masukan (.json) untuk perangkat lunak pengujian.

Listing E.3: Data JSON untuk Perangkat Lunak Eksplorasi

```

1 {
2     "num_sample_data": 1000,
3     "input_path_normal": "D:/input/credit430k.csv",
4     "input_path_anonymize": "D:/input/AnonymizationTable.csv",
5     "input_path_cluster": "D:/input/ClusterizationTable.csv",
6     "output_path": "D:/output/k-means/",
7     "model_name": "k_means",
8     "selected_column": [
```

```
9      {
10        "attrName": "DAYS_BIRTH",
11        "dataType": "numeric"
12      },
13      {
14        "attrName": "DAYS_EMPLOYED",
15        "dataType": "numeric"
16      },
17      {
18        "attrName": "OCCUPATION_TYPE",
19        "dataType": "category"
20      },
21      {
22        "attrName": "CODE_GENDER",
23        "dataType": "category"
24      },
25    ],
26    "k_means": {
27      "k": 25,
28      "features": ["OCCUPATION_TYPE"],
29      "model_path_normal": "D:/model/k-means/normal/",
30      "model_path_anonym": "D:/model/k-means/anonym/"
31    },
32    "naive_bayes": {
33      "label": "AMT_INCOME_TOTAL",
34      "features": ["OCCUPATION_TYPE"],
35      "training_set": 0.8,
36      "test_set": 0.2,
37      "model_path_normal": "D:/model/naive-bayes/normal/",
38      "model_path_anonym": "D:/model/naive-bayes/anonym/"
39    },
40    "total_information_loss": {
41      "path_json": "D:/input/InputAnonymization.json"
42    },
43    "perbedaan_hasil_clustering": {
44      "path_input_normal_clustering": "D:/input/NormalClusteringTable.csv",
45      "path_input_anonym_clustering": "D:/input/AnonymClusteringTable.csv"
46    },
47    "perbedaan_hasil_klasifikasi": {
48      "path_input_normal_classification": "D:/input/NormalClassification.csv",
49      "path_input_anonym_classification": "D:/input/AnonymClassification.csv"
50    }
51 }
```


LAMPIRAN F

HASIL PENGUJIAN FUNGSIONAL

F.1 Hasil Pengelompokan Greedy K-Member Clustering

Berikut adalah hasil pengelompokan data dengan data set Credit score.

Listing F.1: Hasil Pengelompokan Data Credit Score

1	ID, DAYS_BIRTH, DAYS_EMPLOYED, OCCUPATION_TYPE, CODE_GENDER, AMT_INCOME_TOTAL, Cluster
2	396, -19082, -5243, Core staff, F, 74250, Cluster 12
3	864, -14906, -6826, Core staff, M, 445500, Cluster 4
4	198, -18926, -6276, Laborers, M, 166500, Cluster 2
5	73, -12411, -1773, Managers, F, 126000, Cluster 9
6	923, -15901, -130, Security staff, M, 180000, Cluster 5
7	129, -15519, -3234, Laborers, F, 297000, Cluster 10
8	518, -14718, -5578, Managers, F, 450000, Cluster 3
9	999, -10710, -2351, Sales staff, F, 157500, Cluster 7
10	54, -15761, -3173, Laborers, F, 135000, Cluster 4
11	568, -23480, 365243, "", F, 76500, Cluster 2
12	354, -20953, -8684, Accountants, F, 292500, Cluster 9
13	404, -9957, -921, Drivers, M, 135000, Cluster 4
14	160, -16027, -889, "", M, 157500, Cluster 6
15	861, -14118, -3174, Laborers, M, 135000, Cluster 8
16	832, -14657, -4392, Core staff, F, 99000, Cluster 11
17	885, -11813, -3266, Sales staff, F, 202500, Cluster 1
18	493, -19063, -7404, Managers, F, 900000, Cluster 3
19	100, -15519, -3234, Laborers, F, 297000, Cluster 8
20	357, -20953, -8684, Accountants, F, 292500, Cluster 10
21	828, -19341, -7314, Managers, F, 270000, Cluster 3
22	625, -11706, -1164, Drivers, M, 270000, Cluster 11
23	595, -19702, -9870, "", F, 157500, Cluster 1
24	360, -20953, -8684, Accountants, F, 292500, Cluster 10
25	80, -17016, -1347, Core staff, F, 247500, Cluster 10
26	101, -15519, -3234, Laborers, F, 297000, Cluster 9
27	167, -22319, 365243, "", F, 112500, Cluster 2
28	259, -20531, -5730, Drivers, M, 90000, Cluster 2
29	206, -15444, -3112, "", F, 216000, Cluster 9
30	268, -12197, -1194, Managers, M, 765000, Cluster 3
31	188, -18926, -6276, Laborers, M, 166500, Cluster 1
32	520, -14718, -5578, Managers, F, 450000, Cluster 8
33	781, -21688, 365243, "", F, 90000, Cluster 8
34	521, -14718, -5578, Managers, F, 450000, Cluster 11

F.2 Hasil Anonimisasi K-Anonymity

Berikut adalah hasil anonimisasi data dengan data set Credit score.

Listing F.2: Hasil Anonimisasi Data Credit Score

1	ID, DAYS_BIRTH, DAYS_EMPLOYED, OCCUPATION_TYPE, CODE_GENDER, AMT_INCOME_TOTAL, Cluster
2	1, [(-22319) - (-10628)], [(-7465) - 365243], Employee, Adult, 427500
3	2, [(-21688) - (-8489)], [(-10936) - 365243], Employee, Adult, 427500
4	3, [(-22464) - (-8488)], [(-9748) - 365243], Employee, Adult, 112500
5	4, [(-23299) - (-9430)], [(-9870) - 365243], Employee, Adult, 270000
6	5, [(-22464) - (-8488)], [(-9748) - 365243], Employee, Adult, 270000
7	6, [(-22002) - (-9087)], [(-8987) - 365243], Employee, Adult, 270000
8	7, [(-20140) - (-9087)], [(-4816) - 365243], Employee, Adult, 270000
9	8, [(-23480) - (-8488)], [(-10843) - 365243], Employee, Adult, 283500
10	9, [(-22464) - (-8488)], [(-9748) - 365243], Employee, Adult, 283500
11	10, [(-23187) - (-9087)], [(-7465) - 365243], Employee, Adult, 283500
12	11, [(-23480) - (-8657)], [(-6218) - 365243], Employee, Adult, 270000
13	12, [(-22002) - (-9087)], [(-8987) - 365243], Employee, Adult, 270000
14	13, [(-22464) - (-8488)], [(-9748) - 365243], Employee, Adult, 270000
15	14, [(-22061) - (-10276)], [(-9870) - 365243], Employee, Adult, 135000
16	15, [(-22331) - (-9087)], [(-4816) - 365243], Employee, Adult, 135000
17	16, [(-21635) - (-10266)], [(-8684) - 365243], Employee, Adult, 135000
18	17, [(-21688) - (-10031)], [(-10936) - 365243], Employee, Adult, 135000
19	18, [(-23532) - (-10611)], [(-9748) - 365243], Employee, Adult, 135000
20	19, [(-21316) - (-9736)], [(-9870) - 365243], Employee, Adult, 135000
21	20, [(-23187) - (-9380)], [(-10936) - 365243], Employee, Adult, 130500
22	21, [(-22002) - (-9087)], [(-8987) - 365243], Employee, Adult, 130500
23	22, [(-22464) - (-8488)], [(-9748) - 365243], Employee, Adult, 157500
24	23, [(-21688) - (-9489)], [(-9870) - 365243], Employee, Adult, 157500
25	24, [(-21688) - (-10031)], [(-10936) - 365243], Employee, Adult, 157500
26	25, [(-22464) - (-8488)], [(-9748) - 365243], Employee, Adult, 112500
27	26, [(-21688) - (-8489)], [(-10936) - 365243], Employee, Adult, 112500
28	27, [(-21688) - (-9430)], [(-9748) - 365243], Employee, Adult, 112500
29	28, [(-22331) - (-9087)], [(-4816) - 365243], Employee, Adult, 270000
30	29, [(-21688) - (-9430)], [(-9748) - 365243], Employee, Adult, 270000
31	30, [(-21688) - (-9430)], [(-9748) - 365243], Employee, Adult, 405000
32	31, [(-22061) - (-10276)], [(-9870) - 365243], Employee, Adult, 405000
33	32, [(-23187) - (-9380)], [(-10936) - 365243], Employee, Adult, 405000
34	33, [(-23299) - (-9890)], [(-9748) - 365243], Employee, Adult, 405000
35	34, [(-23187) - (-9380)], [(-10936) - 365243], Employee, Adult, 405000

F.3 Hasil Clustering K-Means (Sebelum Anonimisasi)

Berikut adalah hasil pemodelan clustering k-means sebelum dilakukan anonimisasi.

Listing F.3: Hasil Anonimisasi Data Credit Score

1	DAYs_BIRTH, DAYs_EMPLOYED, OCCUPATION_TYPE, CODE_GENDER, NAME_INCOME_TYPE, prediction
2	-12005, -4542, "", M, Working, 4
3	-12005, -4542, "", M, Working, 4
4	-21474, -1134, Security staff, M, Working, 4
5	-19110, -3051, Sales staff, F, Commercial associate, 0
6	-19110, -3051, Sales staff, F, Commercial associate, 0
7	-19110, -3051, Sales staff, F, Commercial associate, 0
8	-19110, -3051, Sales staff, F, Commercial associate, 0
9	-22464, 365243, "", F, Pensioner, 6
10	-22464, 365243, "", F, Pensioner, 6
11	-22464, 365243, "", F, Pensioner, 6
12	-16872, -769, Accountants, M, Working, 0
13	-16872, -769, Accountants, M, Working, 0
14	-16872, -769, Accountants, M, Working, 0
15	-17778, -1194, Laborers, M, Commercial associate, 5
16	-17778, -1194, Laborers, M, Commercial associate, 5
17	-17778, -1194, Laborers, M, Commercial associate, 5
18	-17778, -1194, Laborers, M, Commercial associate, 5
19	-17778, -1194, Laborers, M, Commercial associate, 5
20	-17778, -1194, Laborers, M, Commercial associate, 5
21	-10669, -1103, Accountants, F, Working, 4
22	-10669, -1103, Accountants, F, Working, 4
23	-10031, -1469, Laborers, F, Working, 0
24	-10031, -1469, Laborers, F, Working, 0
25	-10031, -1469, Laborers, F, Working, 0
26	-10968, -1620, "", F, Working, 0
27	-10968, -1620, "", F, Working, 0
28	-10968, -1620, "", F, Working, 0
29	-12689, -1163, Laborers, M, Working, 4
30	-12689, -1163, Laborers, M, Working, 4
31	-11842, -2016, Managers, M, Commercial associate, 8
32	-11842, -2016, Managers, M, Commercial associate, 8
33	-11842, -2016, Managers, M, Commercial associate, 8
34	-11842, -2016, Managers, M, Commercial associate, 8
35	-11842, -2016, Managers, M, Commercial associate, 8

F.4 Hasil Clustering K-Means (Setelah Anonimisasi)

Berikut adalah hasil pemodelan clustering k-means setelah dilakukan anonimisasi.

Listing F.4: Hasil Anonimisasi Data Credit Score

```

1 DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,prediction
2 [(-23299)-(-8861)], [(-10936)-365243], Employee, Adult, Unique earnings, 4
3 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 3
4 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 7
5 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 7
6 [(-23187)-(-8182)], [(-10936)-365243], Employee, Adult, Unique earnings, 0
7 [(-23299)-(-8489)], [(-10936)-365243], Employee, Adult, Unique earnings, 5
8 [(-23480)-(-8657)], [(-9748)-365243], Employee, Adult, Unique earnings, 8
9 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2
10 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2
11 [(-23532)-(-9087)], [(-10936)-365243], Employee, Adult, Unique earnings, 9
12 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2
13 [(-23532)-(-9087)], [(-10936)-365243], Employee, Adult, Unique earnings, 9
14 [(-23480)-(-8488)], [(-10936)-365243], Employee, Adult, Unique earnings, 6
15 [(-23299)-(-8861)], [(-10936)-365243], Employee, Adult, Unique earnings, 4
16 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 3
17 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 7
18 [(-23299)-(-8489)], [(-10936)-365243], Employee, Adult, Unique earnings, 5
19 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 3
20 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2
21 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2
22 [(-23299)-(-8489)], [(-10936)-365243], Employee, Adult, Unique earnings, 5
23 [(-23480)-(-8488)], [(-10936)-365243], Employee, Adult, Unique earnings, 6
24 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 7
25 [(-23532)-(-9087)], [(-10936)-365243], Employee, Adult, Unique earnings, 9
26 [(-23299)-(-8861)], [(-10936)-365243], Employee, Adult, Unique earnings, 4
27 [(-23187)-(-8182)], [(-10936)-365243], Employee, Adult, Unique earnings, 0
28 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 3
29 [(-23480)-(-8657)], [(-9748)-365243], Employee, Adult, Unique earnings, 8
30 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2
31 [(-23187)-(-8182)], [(-10936)-365243], Employee, Adult, Unique earnings, 0
32 [(-22319)-(-8842)], [(-10843)-365243], Employee, Adult, Unique earnings, 1
33 [(-22319)-(-8842)], [(-10843)-365243], Employee, Adult, Unique earnings, 1
34 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 3
35 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 2

```

F.5 Hasil Klasifikasi Naive Bayes (Sebelum Anonimisasi)

Berikut adalah hasil pemodelan klasifikasi naive bayes sebelum dilakukan anonimisasi.. Berikut adalah hasil pemodelan clustering k-means sebelum dilakukan anonimisasi.

Listing F.5: Hasil Anonimisasi Data Credit Score

```

1 DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,prediction
2 DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,prediction
3 -12005,-4542,"",M,Working,1.0
4 -12005,-4542,"",M,Working,1.0
5 -21474,-1134,Security staff,M,Working,1.0
6 -19110,-3051,Sales staff,F,Commercial associate,1.0
7 -19110,-3051,Sales staff,F,Commercial associate,1.0
8 -19110,-3051,Sales staff,F,Commercial associate,1.0
9 -19110,-3051,Sales staff,F,Commercial associate,1.0
10 -22464,365243,"",F,Pensioner,0.0
11 -22464,365243,"",F,Pensioner,0.0
12 -22464,365243,"",F,Pensioner,0.0
13 -16872,-769,Accountants,M,Working,1.0
14 -16872,-769,Accountants,M,Working,1.0
15 -16872,-769,Accountants,M,Working,1.0
16 -17778,-1194,Laborers,M,Commercial associate,0.0
17 -17778,-1194,Laborers,M,Commercial associate,0.0
18 -17778,-1194,Laborers,M,Commercial associate,0.0
19 -17778,-1194,Laborers,M,Commercial associate,0.0
20 -17778,-1194,Laborers,M,Commercial associate,0.0
21 -17778,-1194,Laborers,M,Commercial associate,0.0
22 -10669,-1103,Accountants,F,Working,1.0
23 -10669,-1103,Accountants,F,Working,1.0
24 -10031,-1469,Laborers,F,Working,1.0
25 -10031,-1469,Laborers,F,Working,1.0
26 -10031,-1469,Laborers,F,Working,1.0
27 -10968,-1620,"",F,Working,1.0
28 -10968,-1620,"",F,Working,1.0
29 -10968,-1620,"",F,Working,1.0
30 -12689,-1163,Laborers,M,Working,1.0
31 -12689,-1163,Laborers,M,Working,1.0
32 -11842,-2016,Managers,M,Commercial associate,1.0
33 -11842,-2016,Managers,M,Commercial associate,1.0
34 -11842,-2016,Managers,M,Commercial associate,1.0
35 -11842,-2016,Managers,M,Commercial associate,1.0
36 -11842,-2016,Managers,M,Commercial associate,1.0

```

F.6 Hasil Klasifikasi Naive Bayes (Setelah Anonimisasi)

Berikut adalah hasil pemodelan klasifikasi naive bayes setelah dilakukan anonimisasi.. Berikut adalah hasil pemodelan clustering k-means sebelum dilakukan anonimisasi.

Listing F.6: Hasil Anonimisasi Data Credit Score

```

1 DAYS_BIRTH,DAYS_EMPLOYED,OCCUPATION_TYPE,CODE_GENDER,NAME_INCOME_TYPE,prediction
2 [(-23299)-(-8861)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
3 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
4 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 0.0
5 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 0.0
6 [(-23187)-(-8182)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
7 [(-23299)-(-8489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
8 [(-23480)-(-8657)], [(-9748)-365243], Employee, Adult, Unique earnings, 0.0
9 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
10 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
11 [(-23532)-(-9087)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
12 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
13 [(-23532)-(-9087)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
14 [(-23480)-(-8488)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
15 [(-23299)-(-8861)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
16 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
17 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 0.0
18 [(-23299)-(-8489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
19 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
20 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
21 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
22 [(-23299)-(-8489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
23 [(-23480)-(-8488)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
24 [(-23480)-(-8861)], [(-9748)-365243], Employee, Adult, Unique earnings, 0.0
25 [(-23532)-(-9087)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
26 [(-23299)-(-8861)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
27 [(-23187)-(-8182)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
28 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
29 [(-23480)-(-8657)], [(-9748)-365243], Employee, Adult, Unique earnings, 0.0
30 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
31 [(-23187)-(-8182)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
32 [(-22319)-(-8842)], [(-10843)-365243], Employee, Adult, Unique earnings, 0.0
33 [(-22319)-(-8842)], [(-10843)-365243], Employee, Adult, Unique earnings, 0.0
34 [(-23480)-(-7489)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0
35 [(-23532)-(-9430)], [(-10936)-365243], Employee, Adult, Unique earnings, 0.0

```