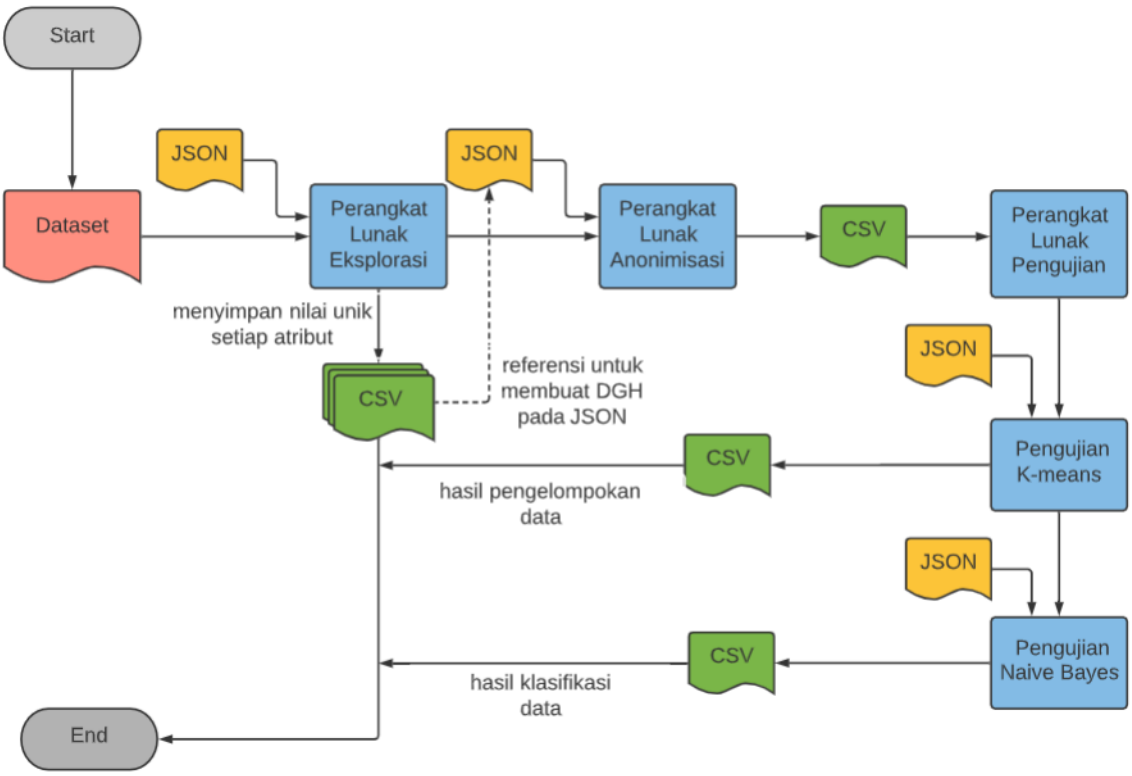
BAB 4

PERANCANGAN

Perancangan yang dibuat pada bab ini meliputi tiga perangkat lunak yaitu eksplorasi, anonimisasi, dan pengujian. Pada perangkat lunak eksplorasi akan dilakukan pencarian nilai unik untuk setiap atribut. Pada perangkat lunak anonimisasi akan dilakukan pemodelan algoritma greedy k-member clustering dan k-anonymity. Pada perangkat lunak pengujian akan dilakukan pemodelan k-means dan naive bayes. Pada tahap ini tidak dilakukan perancangan terhadap tampilan antarmuka karena program yang dihasilkan akan dijalankan dengan menggunakan command prompt.



Gambar 4.1: Flow Chart Penggunaaan Perangkat Lunak

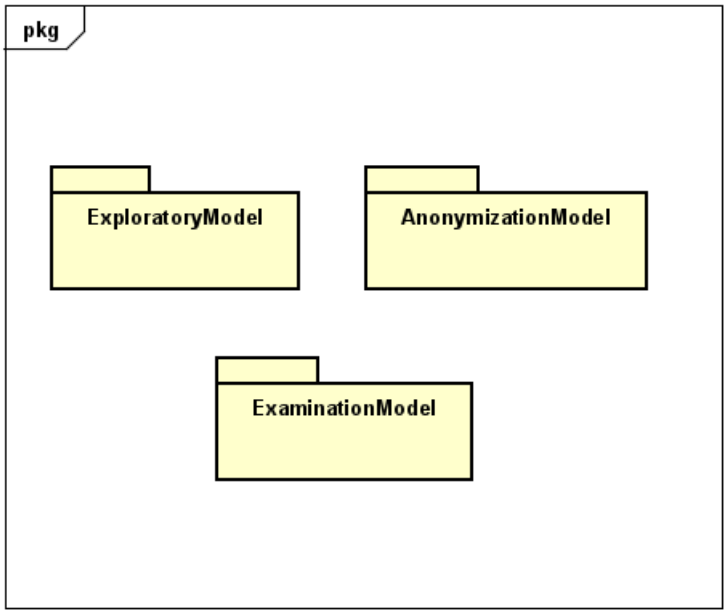
Pada Gambar 4.1 telah ditampilkan input dan output untuk masing-masing perangkat lunak. Persegi panjang berwarna biru diartikan sebagai program utama dan persegi panjang berwarna krem diartikan sebagai program pengujian. Bentuk dengan warna merah diartikan sebagai masukan perangkat lunak dalam bentuk CSV. Bentuk dengan warna hijau diartikan sebagai keluaran perangkat lunak dalam bentuk CSV. Bentuk dengan warna oranye diartikan sebagai input perangkat lunak dalam bentuk JSON. Pada bagian selanjutnya akan dijelaskan perancangan perangkat lunak dengan lebih detil. Penjelasan akan dibagi menjadi tiga bagian yaitu perancangan perangkat lunak eksplorasi, perancangan perangkat lunak anonimisasi, dan perancangan perangkat lunak pengujian.

4.1 Diagram Kelas Lengkap

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

4.1.1 Diagram Package

Perangkat lunak ini mempunyai 3 buah package yang tidak saling berhubungan satu sama lain yaitu ExploratoryModel, AnonymizationModel, ExaminationModel. Ketika package ini memiliki fungsinya masing-masing. Diagram Package dapat dilihat pada Gambar 4.2



Gambar 4.2: Diagram Kelas pada Package

Package ExploratoryModel

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.

Package AnonymizationModel

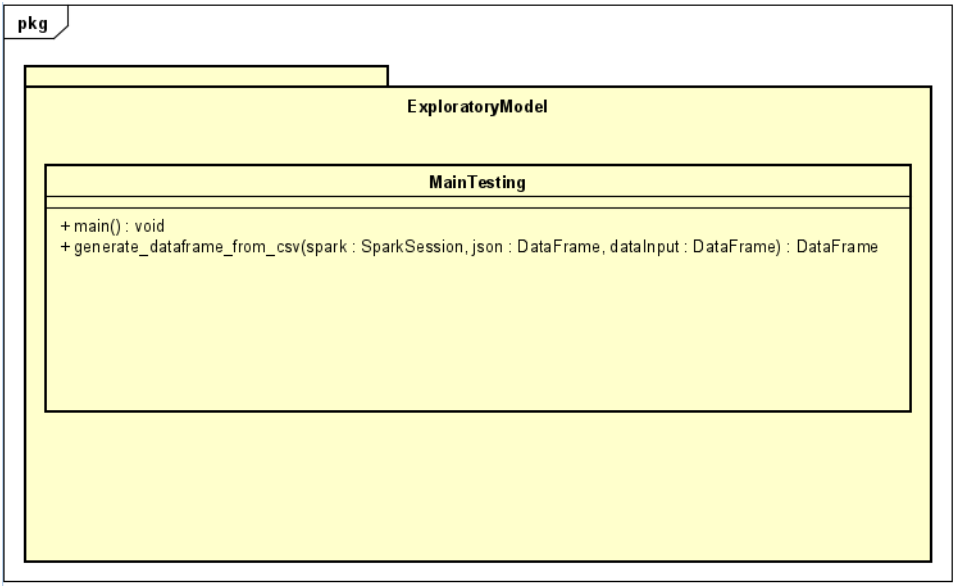
Package AnonymizationModel merupakan package yang menangani segala jenis fungsi yang berkaitan dengan masalah pengelompokan data dan anonimisasi data. Fungsi tersebut antara lain pengelompokan data dengan algoritma Greedy k-member clustering, pencarian record dan cluster terbaik, perhitungan distance numerik dan kategorikal, dan perhitungan information loss. Package ini juga mengimplementasikan berbagai macam operasi untuk membaca atribut DGH pada JSON dan menangani pembuatan binary tree berdasarkan atribut DGH. Operasi-operasi ini diimplementasikan karena adanya kebutuhan untuk membantu proses implementasi pengelompokan data dan anonimisasi data. Selain itu, package ini memiliki dua jenis kelas Main yaitu MainTesting dan MainLCATesting. Kelas MainTesting digunakan untuk melakukan proses pengelompokan dan anonimisasi data dan mengembalikan hasilnya ke dalam format CSV. Kelas MainLCATesting digunakan untuk menampilkan root terdekat dari kedua node.

Package ExaminationModel

Package ExaminationModel merupakan package yang menangani segala jenis fungsi yang berkaitan dengan masalah pengujian data. Fungsi tersebut antara lain pengelompokan data dengan pemodelan k-means beserta evaluasi modelnya dengan silhouette score dan pemodelan naive bayes beserta evaluasi modelnya dengan accuracy. Package ini juga mengimplementasikan berbagai macam operasi untuk membuat DataFrame berdasarkan atribut yang telah dipilih pada JSON dan mendapatkan nilai parameter k-means dan naive bayes dari JSON. Operasi-operasi ini diimplementasikan karena adanya kebutuhan untuk membantu proses pengujian data berdasarkan pemodelan yang dipilih (k-means/naive bayes). Selain itu, package ini memiliki satu jenis kelas Main yaitu MainTesting. Kelas MainTesting digunakan untuk melakukan pengujian data dengan pemodelan k-means/naive bayes dan mengembalikan hasilnya ke dalam format CSV.

4.1.2 Diagram Kelas pada Package ExploratoryModel

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



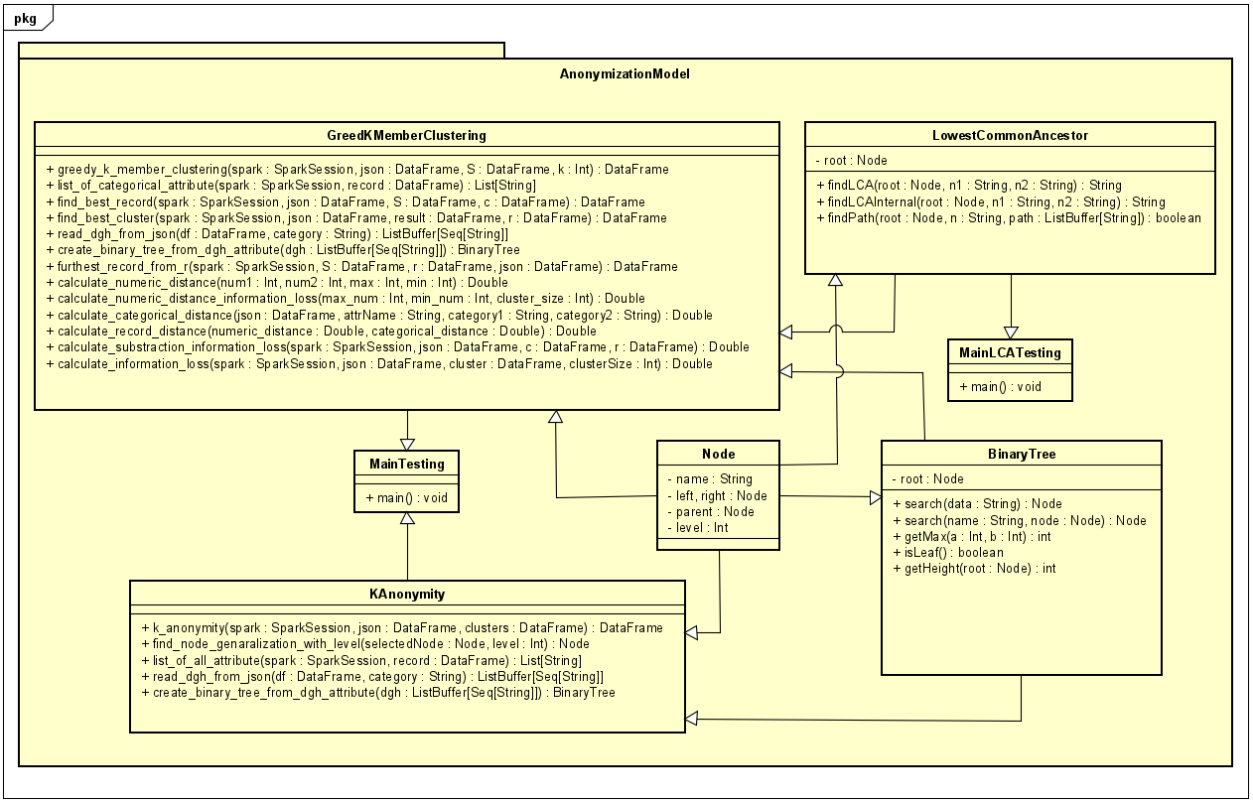
Gambar 4.3: Diagram Kelas pada Package ExploratoryModel

Kelas *MainTesting*

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

4.1.3 Diagram Kelas pada Package AnonymizationModel

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



Gambar 4.4: Diagram Kelas pada Package AnonymizationModel

Kelas *Node*

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*.

Berikut adalah penjelasan masing-masing atribut pada kelas *Node*:

- `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.
- `left` adalah variabel yang berfungsi untuk menyimpan node kiri dari node tersebut dengan tipe Node dan termasuk jenis variabel `var` sehingga atribut ini nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.
- `right` adalah variabel yang berfungsi untuk menyimpan node kanan dari node tersebut dengan tipe Node dan termasuk jenis variabel `var` sehingga atribut ini nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.
- `parent` adalah variabel yang berfungsi untuk menyimpan node `parent` dari node tersebut dengan tipe Node dan termasuk jenis variabel `var` sehingga atribut ini nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.
- `level` adalah variabel yang berfungsi untuk menyimpan posisi ketinggian BinaryTree untuk node tersebut dengan tipe Integer dan termasuk jenis variabel `var` sehingga atribut ini nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.

Kelas *BinaryTree*

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

Berikut deskripsi atribut pada kelas *BinaryTree*:

- `root` adalah variabel yang berfungsi untuk menyimpan node yang menjadi root pada objek BinaryTree dengan tipe Node dan termasuk jenis variabel *var* sehingga atribut *root* nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.

Berikut deskripsi atribut pada kelas *BinaryTree*:

- `search` dengan parameter `data(String)` adalah fungsi yang akan memanggil fungsi search lain dengan parameter *name(String)*, *node(Node)* untuk mengembalikan objek *Node* yang ingin dicari pada objek *BinaryTree*.
- `search` dengan parameter `name(String)`, *node(Node)* adalah fungsi yang bertujuan untuk mengembalikan objek *Node* yang ingin dicari pada objek *BinaryTree*.
- `getMax` dengan parameter `a(Integer)`, `b(Integer)` adalah fungsi yang bertujuan untuk mengembalikan nilai *Integer* paling besar antara parameter `a` dan `b`.
- `isLeaf` tanpa parameter adalah fungsi yang bertujuan untuk mengembalikan nilai *Boolean* untuk menyatakan apakah root dari node kiri dan kanan bernilai *null*. Jika kondisi terpenuhi maka nilainya *true*, apabila tidak terpenuhi maka nilainya *false*.
- `getHeight` dengan parameter `root(Node)` adalah fungsi yang bertujuan untuk mengembalikan nilai *Integer* untuk menyatakan ketinggian dari objek BinaryTree.

Berikut implementasi method `search` pada kelas *BinaryTree*:

Algorithm 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
```

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

Kelas *LowestCommonAncestor*

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

Berikut deskripsi atribut pada kelas *LowestCommonAncestor*:

- `path1` adalah variabel yang berfungsi untuk menyimpan list node yang pernah dipilih sebelumnya dengan tipe ListBuffer[String] dan termasuk jenis variabel *var* sehingga atribut *path1* nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.
- `path2` adalah variabel yang berfungsi untuk menyimpan list node yang pernah dipilih sebelumnya dengan tipe ListBuffer[String] dan termasuk jenis variabel *var* sehingga atribut *path1* nilainya bersifat *mutable*/dapat diubah sesuai kebutuhan.

Berikut deskripsi method pada kelas *LowestCommonAncestor*:

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

Berikut implementasi method `findLCAInternal` pada kelas *LowestCommonAncestor*:

Algorithm 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))
   then
6:     return "not found"
7: end if
8:
9: i = 0
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)
```

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

Berikut implementasi method `findPath` pada kelas *LowestCommonAncestor*:

Algorithm 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
```

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

Kelas *GreedyKMemberClustering*

Kelas *GreedyKMemberClustering* merupakan kelas dengan tipe *class*. Hal ini dikarenakan kelas *GreedyKMemberClustering* berfungsi sebagai model untuk membuat atribut dan method pada objek *GreedyKMemberClustering*. Kelas *GreedyKMemberClustering* bertujuan untuk melakukan fungsi pengelompokan data. Kelas *GreedyKMemberClustering* tidak memiliki atribut.

Berikut deskripsi method pada kelas *GreedyKMemberClustering*:

- `greedy_k_member_clustering` dengan parameter `spark(SparkSession)`, `json(DataFrame)`, `S(DataFrame)`, `k(Integer)` adalah fungsi yang bertujuan untuk melakukan pengelompokan data dengan algoritma Greedy k-member clustering sebelum data dilakukan anonimisasi. Kelompok data yang terbentuk akan didasari pada selisih information loss paling minimum antar masing-masing data berdasarkan fungsi `find_best_record` dan fungsi `find_best_cluster`.
- `list_of_categorical_attribute` dengan parameter `spark(SparkSession`, `record (Data Frame)` adalah fungsi yang bertujuan untuk mendapatkan seluruh nama atribut dari tabel data. Fungsi lain seperti `furthest_record_from_r`, `calculate_information_loss` akan memanggil fungsi ini untuk mendapatkan tabel data yang berisi kolom-kolom bertipe kategorikal, sebelum tabel data dipakai untuk mencari record paling jauh dengan record r dan sebelum tabel data dipakai untuk menghitung nilai information loss pada kelompok data.
- `find_best_record` dengan parameter `spark(SparkSession)`, `json(DataFrame)`, `S(Data Frame)`, `c(DataFrame)` adalah fungsi yang bertujuan untuk mencari setiap record dari tabel data yang memiliki nilai information loss paling minimum terhadap record c.
- `find_best_cluster` dengan parameter `spark(SparkSession)`, `json(DataFrame)`, `S (Data Frame)`, `r(DataFrame)` adalah fungsi yang bertujuan untuk mencari setiap record dari tabel data yang telah dikelompokan dengan nilai information loss paling rendah terhadap sebuah record r, dimana r adalah record yang belum masuk kelompok data tertentu.
- `read_dgh_from_json` dengan parameter `df(DataFrame)`, `category(String)` adalah fungsi yang bertujuan untuk mengembalikan nilai `value,parent,level,position` untuk setiap nama atribut tabel data pada `domain_generaliation_hierarchy` JSON.
- `create_binary_tree_from_dgh_attribute` dengan parameter `dgh(ListBuffer[Seq[String ]])`, `category(String)` adalah fungsi yang bertujuan untuk membuat pohon DGH melalui representasi objek BinaryTree pada `domain_generaliation_hierarchy` JSON.
- `furthest_record_from_r` dengan parameter `spark(SparkSession)`, `S(DataFrame)`, `r (Data Frame)`, `json(DataFrame)` adalah fungsi yang bertujuan untuk menghitung distance record antara record r dengan masing-masing record pada tabel data. Setelah dilakukan perhitungan, record dengan nilai distance record tertinggi akan dikembalikan sebagai output dari fungsi ini.
- `calculate_numeric_distance` dengan parameter `num1(Int)`, `num2(Int)`, `max(Int)`, `min(Int )` adalah fungsi yang bertujuan untuk menghitung distance numerik antara 2 data numerik dengan atribut yang sejenis. Fungsi ini dipanggil oleh fungsi lain yaitu `furthest_record_from_r`, agar hasil perhitungan distance numerik dapat digunakan untuk menghitung distance record.
- `calculate_numeric_distance_information_loss` dengan parameter `max_num(Integer)`, `min_num(Integer)`, `cluster_size(Integer)` adalah fungsi yang bertujuan untuk menghitung distance numerik pada kasus information loss antar 2 data numerik di atribut yang sejenis. Fungsi ini dipanggil oleh fungsi lain yaitu `calculate_information_loss`, agar hasil perhitungan distance numerik dapat digunakan untuk menghitung nilai information loss pada sebuah kelompok data untuk mendapatkan hasil pengelompokan data yang terbaik.

- `calculate_categorical_distance` dengan parameter `json(DataFrame)`,`attrName(String)`, `category1(String)`, `category1(String)` adalah fungsi yang bertujuan untuk menghitung distance kategorikal antara 2 data kategori dengan atribut yang sejenis. Fungsi ini dipanggil oleh fungsi lain yaitu `furthest_record_from_r`, agar hasil perhitungan distance kategorikal dapat digunakan untuk menghitung distance record.
- `calculate_record_distance` dengan parameter `numeric_distance(Double)`, `categorical_distance (Double)` adalah fungsi yang bertujuan untuk menghitung distance record antara 2 record. Fungsi ini dipanggil oleh fungsi lain seperti `furthest_record_from_r`, untuk mencari record dengan distance record paling kecil. Distance record didapat dengan menjumlahkan total distance numerik dan total distance kategorikal.
- `calculate_substraction_information_loss` dengan parameter `spark (SparkSession)`, `json(DataFrame)`, `c(DataFrame)`, `r(DataFrame)` adalah fungsi yang bertujuan untuk mencari selisih information loss antara dua kelompok tabel data yang berbeda. Fungsi ini dipanggil oleh fungsi lain seperti `find_best_record` dan `find_best_cluster` untuk mencari record terbaik berdasarkan selisih information loss paling kecil.
- `calculate_information_loss` dengan parameter `spark(SparkSession)`, `json(DataFrame)`, `cluster(DataFrame)`, `clusterSize(Integer)` adalah fungsi yang bertujuan untuk menghitung nilai information loss pada kelompok tabel data tertentu. Fungsi ini dipanggil oleh fungsi lain seperti `calculate_substraction_information_loss` untuk mencari selisih information loss antara dua kelompok tabel data yang berbeda.

Berikut implementasi method `calculate_categorical_distance`:

Algorithm 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:
8:  if (dgh == null) then
9:      return result
10: end if
11:
12: binaryTree = create_binary_tree_from_dgh_attribute(dgh)
13: node1 = binaryTree.search(category1)
14: node2 = binaryTree.search(category2)
15:
16: if (node1 != null and node2 != null) then
17:     LCA = new LowestCommonAncestor()
18:     resultLCA = LCA.findLCA(binaryTree.root, node1.name, node2.name)
19:     H_subtree = binaryTree.search(resultLCA).level
20:     H_TD = binaryTree.getHeight(binaryTree.root).toDouble
21:     result = H_subtree / H_TD
22: end if
23:
24: result = BigDecimal(result)
25: result = result.setScale(2, BigDecimal.RoundingMode.HALF_UP)
26: result = result.toDouble
27: return result
```

- Baris 6: baris ini mendapatkan nilai `value,parent,level,position` untuk setiap nama atribut pada `domain_generalization_hierarchy` JSON.
- Baris 8-10: baris ini memeriksa jika nilai pada `domain_generalization_hierarch` JSON kosong, maka perhitungan distance kategorikal tidak perlu dilakukan.
- Baris 12: baris ini membuat objek BinaryTree berdasarkan atribut `dgh` pada baris 6.
- Baris 13: baris ini mendapatkan objek Node pertama dengan parameter nama tertentu.
- Baris 14: baris ini mendapatkan objek Node kedua dengan parameter nama tertentu.
- Baris 16-22: baris ini menghitung *distance* kategorikal jika kedua node tidak kosong.
- Baris 19: baris ini menyimpan ketinggian `node root` dari objek `LowestCommonAncestor`
- Baris 20: baris ini menyimpan ketinggian maksimal dari objek `BinaryTree`
- Baris 24-26: baris ini mengembalikan hasil perhitungan *distance*kategorikal dalam format desimal, dengan ketelitian 2 angka dibelakang koma.

Berikut implementasi method `calculate_numeric_distance`:

Algorithm 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
```

- Baris 5: baris ini melakukan inisialisasi `v1` dengan nilai `num1`.
- Baris 6: baris ini melakukan inisialisasi `v2` dengan nilai `num2`.
- Baris 7: baris ini menghitung selisih perbedaan nilai antara `v1` dan `v2`.
- Baris 8: baris ini menghitung selisih range nilai antara `v1` dan `v2`.
- Baris 9: baris ini membagi selisih perbedaan nilai dengan selisih *range* nilai.

Berikut implementasi method `calculate_record_distance`:

Algorithm 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
```

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

Berikut implementasi method `furthest_record_from_r`:

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

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

Berikut implementasi method `calculate_information_loss`:

Algorithm 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
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
```

- Baris 7: baris ini mendapatkan seluruh nama atribut dari tabel data.
- Baris 10: baris ini mendapatkan nilai dari masing-masing kolom tabel data.
- Baris 12-34: baris ini melakukan perulangan untuk setiap kolom tabel data.
- Baris 13-16: baris ini menghitung *distance* numerik apabila kolom termasuk nilai numerik.
- Baris 17-33: baris ini menghitung *distance* kategorikal apabila kolom termasuk nilai kategori.
- 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*
- Baris 30-31: baris ini memeriksa jika nilai unik sebuah kolom kategorikal terdiri dari 1 atau lebih dari 2 nilai unik, maka *distance* kategorikal tidak dihitung.

Kelas *KAnonymity*

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

Berikut deskripsi method pada kelas *KAnonymity*:

- `k_anonymity` dengan parameter *spark(SparkSession)*, *json(DataFrame)*, *clusters(DataFrame)* adalah fungsi yang bertujuan untuk melakukan anonimisasi data dengan algoritma k-anonymity setelah tabel data berhasil dilakukan pengelompokan data.
- `find_node_genaralization_with_level` dengan parameter *selectedNode(Node)*, `level(Int)` adalah fungsi yang bertujuan untuk mencari node dengan tingkatan generalisasi tertentu pada objek BinaryTree. Tingkatan generalisasi ditentukan dari ketinggian node pada objek BinaryTree. Semakin kecil ketinggian node, maka nilainya semakin umum.
- `list_of_all_attribute` dengan parameter `spark(SparkSession)`, `record(DataFrame)` adalah fungsi yang bertujuan untuk mendapatkan seluruh nama kolom pada sebuah tabel data. Fungsi ini akan dipanggil pada fungsi lain, yaitu `k_anonymity` untuk mengganti nilai sesungguhnya dengan nilai yang lebih umum pada proses anonimisasi.
- `read_dgh_from_json` dengan parameter *df(DataFrame)*, `category(String)` adalah fungsi yang bertujuan untuk mengembalikan nilai `value, parent, level, position` untuk beberapa nama kolom pada domain_generaliation_hierarchy JSON.
- `create_binary_tree_from_dgh_attribute` dengan parameter `dgh(ListBuffer[Seq[String ]])` adalah fungsi yang bertujuan mengambil domain_generaliation_hierarchy pada JSON untuk membuat objek BinaryTree. Fungsi ini akan dipanggil pada fungsi lain, yaitu `k_anonymity` untuk mengambil nilai yang lebih umum pada pohon DGH.

Berikut implementasi method `k_anonymity`:

- Baris 7: baris ini mendapatkan seluruh nama atribut dari tabel data.
- Baris 9-48: baris ini melakukan perulangan selama `cluster_temp` tidak kosong.
- Baris 11-12: baris ini mendapatkan data berdasarkan nama cluster tertentu.
- Baris 16-40: baris ini melakukan perulangan untuk untuk setiap kolom tabel cluster tertentu.
- Baris 17: baris ini mendapatkan nilai unik dari sebuah kolom data pada nama cluster tertentu.
- Baris 21-25: melakukan anonimisasi kolom numerik dengan rentang nilai kolom pada cluster.
- Baris 26-38: melakukan anonimisasi kolom kategorikal dengan nama root pada pohon DGH.
- Baris 28-31: baris ini memeriksa jika nilai unik kategorikal pada sebuah kolom melebihi 1 nilai unik, maka hasil anonimisasinya adalah nama root dari pohon DGH.
- Baris 32-37: baris ini memeriksa jika nilai unik kategorikal pada sebuah kolom hanya 1 nilai unik, maka hasil anonimisasinya adalah nilai kategorikal tersebut.
- Baris 41-42: baris ini menginisialisasi baris data hasil anonimisasi untuk pertama kali.
- Baris 43-44: baris ini menggabungkan hasil anonimisasi pada baris-baris data sebelumnya dengan hasil anonimisasi pada baris data yang baru.

Algorithm 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
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
```

Kelas *MainTesting*

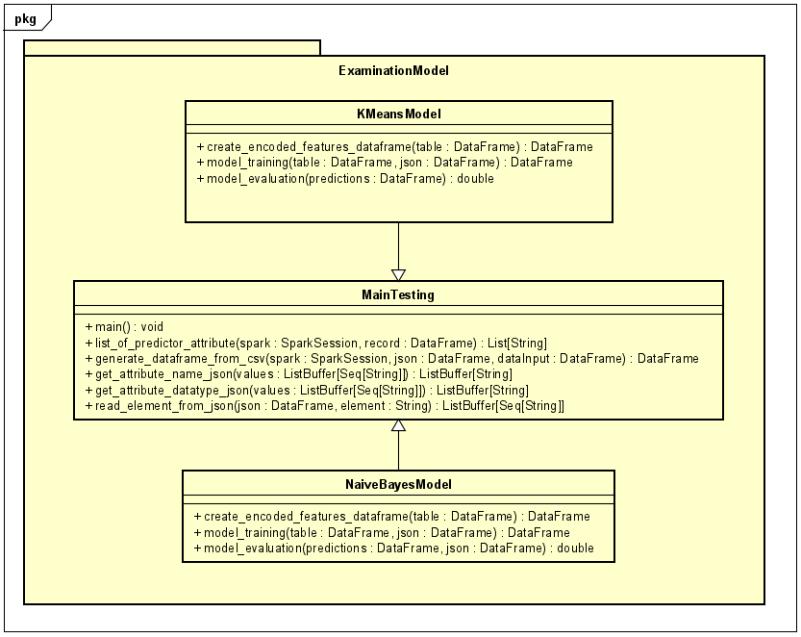
Kelas *MainTesting* merupakan kelas dengan tipe *object*. Hal ini dikarenakan kelas *MainTesting* berperan penting untuk melakukan eksekusi pengelompokan data dengan algoritma Greedy k-member clustering dan anonimisasi data dengan algoritma k-anonymity.

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

- `main` adalah fungsi yang bertujuan untuk melakukan eksekusi perangkat lunak anonimisasi dengan objek GreedyKMemberClustering dan objek KAnonymity.
- `generate_dataframe_from_csv` dengan parameter `spark(SparkSession)`, `json(DataFrame)`, `dataInput(DataFrame)` adalah fungsi yang bertujuan untuk mengambil data `quasi_identifier` dan `sensitive_attribute` pada JSON. Output dari fungsi ini adalah array 2 dimensi yang berisi nama atribut dan tipe atribut (*category/numeric*).
- `get_attribute_name_json` dengan parameter `values(ListBuffer[Seq[String]])` adalah fungsi yang bertujuan untuk mendapatkan nama-nama atribut berdasarkan output array 2 dimensi dari fungsi `generate_dataframe_from_csv`.
- `get_attribute_datatype_json` dengan parameter `values(ListBuffer[Seq[String]])` adalah fungsi yang bertujuan untuk mengubah atribut dengan tipe "category" menjadi String, sedangkan atribut dengan tipe "numeric" menjadi Integer.
- `read_element_from_json` dengan parameter `json(DataFrame)`, `elemen(String)` adalah fungsi yang bertujuan untuk mendapatkan nilai atribut tertentu pada JSON. Fungsi ini dipanggil pada fungsi lain yaitu `generate_dataframe_from_csv` untuk mengambil nilai dari atribut `quasi_identifier` dan `sensitive_identifier`.

4.1.4 Diagram Kelas pada Package ExaminationModel

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



Gambar 4.5: Diagram Kelas pada ExaminationModel

Kelas *KMeansModel*

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*.

Berikut deskripsi method pada kelas *KMeansModel*:

- `create_encoded_features_dataframe` dengan parameter `table(DataFrame)` adalah fungsi yang bertujuan untuk membuat tabel data baru yang berisi nilai aktual, label index, vektor untuk masing-masing atribut, dan ditambah satu kolom baru untuk menyimpan vektor fitur dari label index. Output fungsi ini menjadi input pada fungsi `model_training`
- `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.
- `model_evaluation` dengan parameter `predictions(DataFrame)` adalah fungsi yang bertujuan untuk mencari tahu seberapa baik model k-means yang dibuat dengan menghitung silhouette score. Apabila silhouette score mendekati nilai 1, maka pengelompokan data yang dibuat dengan pemodelan k-means telah mendekati hasil yang benar.

Berikut implementasi method `model_training`:

Algorithm 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
```

- Baris 5: baris ini mendapatkan nilai `k` dari atribut `k_means` JSON.
- Baris 6: baris ini membuat model KMeans menggunakan parameter k pada baris sebelumnya.
- Baris 7: baris ini melakukan pelatihan model k-means dengan parameter tabel data.
- Baris 8: baris ini melakukan prediksi model k-means terhadap parameter tabel data.

Berikut implementasi method `model_evaluation`:

Algorithm 14 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: evaluator = new ClusteringEvaluator()
6: silhouette_score = evaluator.evaluate(predictions)
7: return silhouette_score
```

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

Kelas *NaiveBayesModel*

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

Berikut deskripsi method pada kelas *NaiveBayesModel*:

- `create_encoded_features_dataframe` dengan parameter `table(DataFrame)` adalah fungsi yang bertujuan untuk membuat tabel data baru yang berisi nilai aktual, label index, vektor untuk masing-masing atribut, dan ditambah satu kolom baru untuk menyimpan vektor fitur dari label index. Output fungsi ini menjadi input pada fungsi `model_training`
- `model_training` dengan parameter `table(DataFrame)`, `json(DataFrame)` adalah fungsi yang bertujuan untuk membuat model pelatihan naive bayes dan menghasilkan tabel klasifikasi data berdasarkan input vektor fitur untuk setiap baris data.
- `model_evaluation` dengan parameter `predictions(DataFrame)`, `json(DataFrame)` adalah fungsi yang bertujuan untuk mencari tahu seberapa baik model naive bayes yang dibuat dengan menghitung accuracy score. Apabila nilainya mendekati 1, maka klasifikasi data yang dibuat dengan model naive bayes sudah mendekati benar.

Berikut implementasi method `model_training`:

Algorithm 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:
7:  trainingSet = json.select("naive_bayes.training_set").first().getDouble(0)
8:  testSet = json.select("naive_bayes.test_set").first().getDouble(0)
9:
10: Array(training, test) = table.randomSplit(Array(trainingSet, testSet))
11:
12: model = new NaiveBayes().setModelType("multinomial")
13: model = model.setLabelCol(attrName+"_Index").fit(training)
14:
15: predictions = model.transform(test)
16: return predictions
```

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

Berikut implementasi method `model_evaluation`:

Algorithm 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
```

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

Kelas *MainTesting*

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

Berikut deskripsi method pada kelas *MainTesting*:

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

4.2 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 5.9.

Listing 4.1: Dataset Adult

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

4.2.1 Masukan Perangkat Lunak Eksplorasi

Perangkat lunak ekplorasi 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.

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

```
{
  "input_path": "<path data input>",
  "output_path": "<path data output>",
  "selected_column": [
      {
        "attrName": "<nama atribut>",
        "dataType": "<tipe data atribut>"
      },
      {
        "attrName": "<nama atribut>",
        "dataType": "<tipe data atribut>"
      }
  ]
}
```

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

Berikut spesifikasi input .json untuk perangkat lunak anonimisasi:

- `k` adalah nilai konstanta k-anonymity dan greedy k-member clustering. Atribut ini memiliki tipe data Integer sehingga nilainya harus bilangan bulat.
- `num_sample_datas` adalah jumlah sample data. Atribut ini memiliki tipe data Integer sehingga nilainya harus dinyatakan dalam bilangan bulat.
- `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.
- `identifiers` adalah array yang berisi nama-nama atribut yang dapat mengungkapkan identitas sebuah data. Atribut ini akan dihilangkan pada tabel anonimisasi.
- `sensitive_identifiers` adalah array yang berisi nama-nama atribut yang nilainya bersifat sensitif. Atribut ini akan tetap ada pada tabel dianonimisasi.
- `quasi_identifiers` adalah atribut yang nilainya dapat dipakai untuk mengungkap entitas data. Atribut ini berisi pasangan nilai berupa nama atribut dan jenis atribut. Jenis atribut diisi dengan nilai "category" untuk menyatakan atribut kategori, sedangkan untuk menyatakan atribut numerik jenis atribut akan diisi dengan nilai "numeric".
- `domain_generalization_hierarchy` adalah array yang menyimpan informasi mengenai pembentukan pohon DGH. Atribut ini menyimpan 5 jenis informasi penting dalam pembentukan pohon DGH, yaitu:
  - `attrName` adalah nama atribut yang ingin dipilih pada tabel data. Atribut ini akan memberi tahu Spark untuk mengambil atribut berdasarkan nama atribut.
  - `value` adalah nilai-nilai yang mungkin muncul untuk atribut yang terpilih. Atribut ini akan digunakan untuk memberi nama pada sebuah node di pohon DGH.
  - `parent` adalah nilai yang menyatakan nama dari sebuah node root. Atribut ini akan digunakan untuk menyatakan parent dari sebuah node di pohon DGH.
  - `level` adalah nilai yang menyatakan ketinggian node di pohon DGH. Atribut ini digunakan untuk menempatkan node di ketinggian tertentu di pohon DGH.
  - `position` adalah nilai yang menyatakan posisi node di pohon DGH. Atribut ini digunakan untuk menempatkan node di posisi (kiri/kanan) dari node parent.

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

- Nilai atribut yang dicantumkan pada `domain_generalization_hierarchy` di salah satu atribut harus mencakup seluruh kemungkinan nilai untuk atribut tersebut. Apabila tidak terpenuhi, maka beberapa nilai tidak dapat dianonimisasi.
- Semakin banyak atribut yang dicantumkan pada `quasi_identifier`, maka waktu komputasi akan semakin lama untuk dijalankan pada komputer lokal.

Listing 4.3: Input JSON untuk Anonimisasi Data

```
{
  "k": <konstanta untuk k-anonymity dan greedy k-member clustering>,
  "num_sample_datas": <jumlah sampel data>,
  "input_path": "<path data input>",
  "output_path": "<path data output>",
  "identifier": [
      {
        "attrName": "<nama atribut>",
        "dataType": "<tipe data atribut>"
      }
  ],
  "sensitive_identifier": [
      {
        "attrName": "<nama atribut>",
        "dataType": "<jenis atribut>"
      }
  ],
  "quasi_identifier": [
      {
        "attrName": "<nama atribut>",
        "dataType": "<jenis atribut>"
      },
      {
        "attrName": "<nama atribut>",
        "dataType": "<jenis atribut>"
      }
  ],
  "domain_generalization_hierarchy": {
      "<nama atribut>":[
          {
            "value": "<nilai atribut>",
            "parent": "<nilai atribut parent>",
            "level": "1",
            "position": "null"
          },
          {
            "value": "<nilai atribut>",
            "parent": "<nilai atribut parent>",
            "level": "2",
            "position": "left"
          },
          {
            "value": "<nilai atribut>",
            "parent": "<nilai atribut parent>",
            "level": "2",
            "position": "right"
          }
      ]
  }
}
```

4.2.3 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 **??**.

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:
  - `label` adalah nama atribut yang akan dianggap sebagai label untuk pemodelan naive bayes. Atribut ini digunakan untuk mengklasifikasikan data berdasarkan nilai pada atribut label.
  - `training_set` adalah persentase pembagian data training. Umumnya, persentase yang dipilih untuk pembagian data training adalah 0.7.
  - `test_set` adalah persentase pembagian data test. Umumnya, persentase yang dipilih untuk pembagian data test adalah 0.3.

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

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

Listing 4.4: Input JSON untuk Pengujian Data

```
{
  "input_path": "<path data input>",
  "output_path": "<path data output>",
  "model_name": "<model pengujian (k_means/naive_bayes)>",
  "selected_column": [
      {
        "attrName": "<nama atribut>",
        "dataType": "<jenis atribut>"
      },
      {
        "attrName": "<nama atribut>",
        "dataType": "<jenis atribut>"
      }
  ],
  "k_means": {
      "k": <nilai k untuk k_means>
  },
  "naive_bayes": {
      "label": "<nama atribut>",
      "training_set": <persentase training_set (0.7)>,
      "test_set": <persentase training_set (0.3)>
  }
}
```