PEARLS API and Backend Documentation

i. Brief description of the PEARLS visualisation technique

The PEARLS data visualisation is used to provide a conceptual visualisation of high-dimensional data clusters, proposed by Mr. Nahil Jain in 2012. The algorithm aims to extract meaningful information from large quantities of data by clustering them using some already existing clustering algorithm, re-clustering the clusters to form 'pearls', which are then assigned a particular shape, and placed in 3D space with respect to the centroid of the cluster. For a more detailed description of the algorithm, refer to 'Visual Analysis of High Dimensional Real Data - Nahil Jain, 2012'.

ii. For whom is this documentation for

This documentation is written specifically for developers who wish to modify or extend the existing backend/frontend of the PEARLS software. This version of pearls (v0.1) is a very rudimentary one, where all files uploaded by all users are dumped into a common folder, there is no provision for security of files, and there might be some unhandled errors within the system. This serves as a guide about the files that exist in the current version of the backend and a brief structure of the barebones frontend written for v0.1, and the purpose of those functions/components, so that a developer can either use the same backend APIs, and develop a new frontend for the same, or add in a new clustering algorithm into the toolkit.

1. Setting up pearls locally:

Pre-requirements:

Make sure the following are installed in the system running

- Python (version >= 3.6) (For backend)
- npm (version >= 6.14.5) (For frontend)
- node (version >= 12.18.1) (For frontend)

In order to set up and run the PEARLS app locally, do the following:

For setting up the frontend, do the following in the pearls_frontend folder:

```
yarn install    # Only for the first time setting up the frontend
yarn start      # Hosts the frontend on localhost:3000
```

After running the following commands, open localhost:3000 on your browser to view the frontend.

For the backend, go to the PearlsAPI_Flask folder, and run the following

```
# Only for the first time setting up the backend
pip install -r requirements.txt
# Run the command below to start the backend
python server.py      # Hosts the frontend on localhost:5000
```

2. Details about the PEARLS backend API:

The pearls backend API as of now has the following calls incorporated in it:
(Note: all requests are POST requests unless explicitly mentioned, each POST call involves a JSON being sent to the backend (for parameters))

UploadCSV:

In PEARLS v0.1, the CSVs that are being uploaded from the frontend usually end up in a common folder named 'media'. This request entails a .csv file with it, which gets saved in the 'media' folder. Further versions will incorporate a login setup, where users can maintain a set of files for themselves.

URL for API call: **URL_of_server/upload**

Note: The request being sent to the server consists of a file object, which is accessed as requests.files['file']. The name of the file is accessed by requests.files['file'].filename

Parameters in input JSON: Apart from the file object, there are none.

Format of response:

```
{
  'response': 'Success',
  'fieldList' : # A list of all the column names in the csv file
}
```

ClusteringService:

This is used to cluster an existing file in the 'media' folder. On making this request, the server generates a JSON file with the same name as the CSV file with the cluster data stored as a json in the 'clustered_data' folder.

URL for API call: **URL_of_server/cluster**

Parameters in input JSON:

```
{
  clustering_algo:
  # The algorithm to be used for clustering
  pearl_clustering_algo:
  # The algorithm to be used for clustering
  number_of_clusters:
  # Number of clusters that need to be formed
  number_of_pearls:
  # Number of pearls that need to be formed
```

```
  #! NOTE: If the clustering algorithm doesn't need number of
     clusters as an input, it will simply ignored by the backend.

  binning_dimension: "None" # default argument
  # Dimension based on which binning should be done
  binning_criterion:
  # The criterion on which the bins will be created. There are
    two main options:
         ## binsize: creates bins of equal/near equal number of
                     data points
         ## range:   Divides the complete range of the inputted
                     binning dimension into equal sub-ranges, and
                     classifies
         ## bins_per_cluster: Number of bins each cluster's data
                     must be distributed into
         ## filtered_attributes: A boolean array, which indicates
                     whether a particular attribute is selected
                     or not. The order of the boolean array
                     corresponds to the order
}
```

Clustering algorithm-specific attributes:

1. KrNN:
   A. KrNN_k_for_clustering
   B. KrNN_k_for_pearling

Format of response:

```
{
  'response': 'Success',
  'fieldList' : # A list of all the column names in the csv file
}
```

RetrieveCluster:

This call is used to retrieve a particular cluster from a particular file that has already been uploaded to the media folder and been clustered (and hence, JSON file corresponding to the CSV file exists in the clustered_data folder).

URL for API call: **URL_of_server/rcluster**

Parameters in input JSON:

```
{
  currentCluster: # Index of the cluster required
  filename: # Name of the file from where the cluster is required
}
```

Format of response:

```
{
  'response': 'Success',
  'fieldList' : # A list of all the column names in the csv file
}
```

RetrieveDomains:

This API call is used to retrieve the name of each column that is of a numerical (integer/ float type) and the range of the data in the column. This was created for providing input to the Parallel coordinates chart that appears for each cluster.

URL for API call: **URL_of_server/rdomains**

Parameters in input JSON:

```
{
  Filename : # Name of the csv file for which we want to retrieve
               domain information
}
```

Format of response:

```
{
   'domains' : # A list of all the numerical column names in the
csv file with their minimum and maximum values i.e. List of
objects of the form: [“Column name”, [minvalue, maxvalue].
 }
```

RetrieveHeaders

This API call returns the names of all the column names i.e the headers of the CSV file mentioned in the request.

URL for API call: **URL_of_server/rheaders**

Parameters in input JSON:

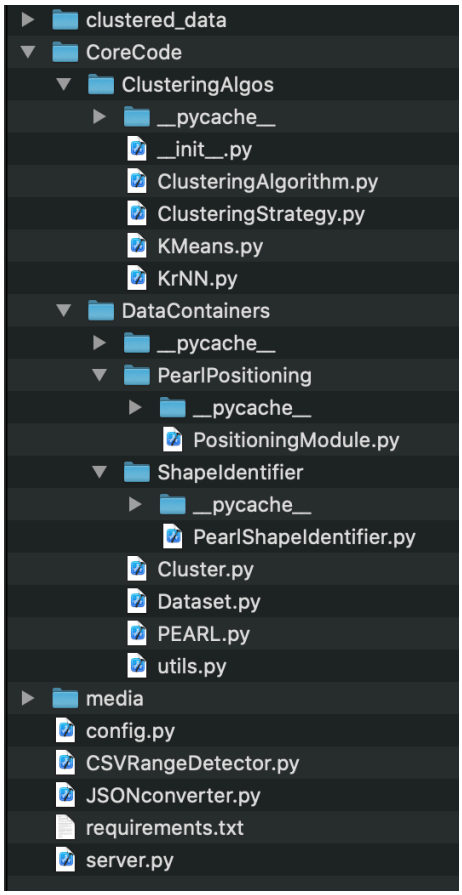
```
{
  Filename : # Name of the csv file for which we want to retrieve
               domain information
}
```

Format of response:

```
{
  'headers' : # A list of all the column names in the csv file.
}
```

3. Documentation of the backend

The file structure of the backend is as follows:



media:
This folder contains all the CSV files uploaded from the frontend.

clustered_data:
This folder contains the Clustered data JSON of the CSV files in the media folder if the cluster API call has been made over that particular CSV file.

config.py:
This file contains configurations of the flask server.

CSVRangeDetector.py:
This file contains a function that is used in the RetrieveDomains API call:

Function: returnDomainsJSON
Description:
A function to return the minimum and maximum values of all headers of numerical (int/float) type headers in the file.
Parameters:
- **Filename:** Name of the file for which we require the domain information

Output:
A json of the form `{domains: [A list of objects of the form [ColumnName:[MinValue, MaxValue]]]}`

JSONconvertor.py:
This file contains a class to convert a Dataset object (see the documentation of **Dataset.py** in the **CoreCode/DataContainers** folder.

Function: NpEncoder
Description:
A custom encoder function that allows the JSON library to serialise the int64 and special float types occurring in Pandas (which are not normally serialisable.

Class DatasetToJSON
Description:
A class that has all the utility functions required to convert the Dataset object to a JSON

Methods:
- **PEARL_to_JSON:**

Description:
A function to convert a PEARL object into JSON form
Parameters:
- **PEARL_object:** The PEARL object we need a JSON object of

Output:

A JSON object of the pearl.

- **cluster_to_JSON:**

Description:
A function to convert a Cluster object into JSON form
Parameters:
- cluster_object: The Cluster object we need a JSON object of
Output:
A JSON object of the Cluster.

- **convert_dataset_to_JSON:**
Description:
The main helper function to convert a Dataset object into JSON form
Parameters:
- PEARL_object: The PEARL object we need a JSON object of
Output:
A JSON object of the Dataset object.

(See 'Details about the clustered JSON for more information on the JSON structure)

CoreCode/DataContainers:

This contains all files containing functions and data structures pertaining to the pearl creation algorithm. The files are as follows:

utils.py

This file contains basic utility functions that are used across the entire codebase in the PEARL formation process.

Function: LP_norm
Description:
Calculate the LP-norm of a given vector
Parameters:
- **point:** The N-D vector whose norm needs to be calculated
- **p:** The p of the LP-norm

Output:
LP norm of point and given p.

Function: distance
Description:
Calculates the P-distance between two points, by default, P = 2
Parameters:
- **point0:** The first point
- **point1:** The second point
- **p:** The p of the LP-norm (2 by default)

Output:
P-distance between the two points, returns -1, if point0 and point1's dimensions don't match.

Function: find_centroid
Description:
Returns the mean of a dataframe that is completely numerical in nature, which also represents the centroid of the dataset passed as input.
Parameters:
- **dataframe :** A pandas dataframe object, whose columns are numerical in nature

Output:
A pandas series, with each attribute's value being the mean of the entire dataset's values for that attribute.

PEARL.py

This file consists of the PEARL class, an object used to represent a PEARL in the clustered data.

class PEARL
Description:
A class whose objects represent a single pearl under a cluster in the PEARLS algorithm The cluster is a container for all the pearls belonging to itself, and is responsible for identifying the coordinates of the pearl in 3D-space and triggering the shape identification algorithm for all pearls under it.
Attributes:
- **cluster_ID:** ID of the cluster to which the Pearl is assigned to
- **pearl_ID:** ID of the Pearl assigned within its parent cluster
- **data:** A Pandas dataframe which consists of the data belonging to the Pearl
- **centroid:** Centroid of all the data points contained in the Pearl
- **P:** Value of P of the Pearl, that is used to determine its shape

- **radius:** The radius of the Pearl, which will be used to render the Pearl in 3D space
- **centroid_3D_coords:** The relative position of the Pearl with respect to the cluster centroid

Methods:
(Note: This class only has getters and setters for the attributes listed above)

- **set_pearl_ID:** Setter for the pearl_ID
- **get_centroid:** Getter for the centroid of the Pearl
- **get_P:** Getter for the P of the pearl
- **get_radius:** Getter for the radius
- **get_data:** Getter for the data in the Pearl
- **set_3D_coords:** Setter for the centroid_3D_coords attribute
- **get_3D_coords:** Getter for the centroid_3D_coords attribute

Cluster.py
This file consists of the Cluster class, an object used to represent a Cluster in the clustered data.

class Cluster
Description:
Class whose objects represent a single cluster in the PEARLS algorithm. The cluster is a container for all the pearls belonging to itself, and is responsible for identifying the coordinates of the pearl in 3D-space and triggering the shape identification algorithm for all pearls under it.

Attributes:
- **cluster_ID:** ID of the cluster in the final clustered data
- **data:** A Pandas dataframe which consists of the data belonging to the Cluster
- **cluster_centroid:** Centroid of all the data points contained in the Cluster
- **column_filter:** List of columns that need to be removed from the Cluster data before clustering for Pearls
- **pearls:** list of Pearl objects contained by the Pearl
- **pearling_metadata_keys:** All the keys in the clustering metadata details sent from the frontend that are relevant to the Pearl creation process

Methods:

• **set_metadata:**
Description:
A function to set metadata from the input received from the enclosing Dataset object.
Parameters:
- **metadata:** The metadata required for the pearling process, in JSONform, to be extracted into python dictionaries.
Output:
None

• **create_PEARLS:**
Description:
A Function to trigger the formation of pearls of the concerned cluster. This function calls appropriate functions to set the pearling algorithm, create bin labels (to divide the data into bins), create the pearls, and calculate their 3D projected coordinates. After the pearls are created, it erases the cluster data points (since they are stored in the pearls already).
Parameters:

- **attr_filter:** A boolean array that corresponds to whether the particular attribute at that index must be added or not

Output:
None

- **clear_data:**

Description:
A function to erase the data of the cluster from memory after the clustering process is done to avoid duplicate data in the dataset object.

Parameters:
None

Output:
None

- **add_PEARLS_to_list:**

Description:
Function that triggers the actual pearl formation, which results in pearls added to the pearl list.

Parameters:
- **attr_filter:** A boolean array that corresponds to whether the particular attribute at that index must be added or not

Output:
None

- **set_clustering_algorithm:**

Description:
Function that sets the desired pearl clustering algorithm from the clustering strategy(see **ClusteringStrategy.py**). The function creates a new object attribute, **return_pearls**, which is the clustering algorithm function that will be used.

Parameters:
None

Output:
None

- **create_bin_labels:**

Description:
Function to create bin labels to divide data into multiple bins. Keeps data in one single bin if binning criterion is set to "None". It adds a new column in the data called as **bin_number** temporarily to assist the cluster formation process.

Parameters:
None

Output:
None

- **dataframe_from_list:**

Description:
Function to convert a list of Pandas series to a dataframe.

Parameters:
- **bin_list:** A list of Pandas Series

Output:
A Pandas dataframe consisting of all the points in the bin_list

- **get_clusterID:** A getter function for the cluster_ID attribute
- **get_centroid:** A getter function for the cluster_centroid attribute

- **get_upper_bounds_of_bins:**

Description:
In the case of binning by range of a particular binning dimension, the function divides the entire range of the dimension into the number of bins required.

Parameters:
None

Output:
A list with the upper bound of bins (A data element whose attribute value < upper_bound_of_bin[i] belongs to bin number `i`)

- **set_PEARL_coordinates:**

Description:
Function that sets the 3D-projected coordinates of each pearl in the pearl list with respect to the cluster centroid.

Parameters:
None

Output:
None

- **drop_attributes:**

Description:
Function to drop the attributes the user does not require from the input dataframe.

Parameters:
- **df: The dataframe whose attributes need to be filtered**
- **filter_:** A boolean array that corresponds to whether the particular attribute at that index must be added or not

Output:
A Pandas dataframe consisting of all the points in df, but with the unwanted columns dropped.

Dataset.py
The file that contains the Dataset class, the parent class that encloses clusters in it during the cluster and pearl formation process.

class Dataset

Description:
Basic container for the file fed in; it stores clusters and pearls. The Dataset class is responsible for the creation of the entire dataset-cluster-pearls tree-like data structure from the file that is being inputed. It takes care of tasks such as reclustering and attribute filtering as well.

Attributes:
- **current_path:** The path of the file for which clustering is required
- **attribute_list:** A list of column headers in the file
- **selected_attributes:**A boolean array that corresponds to whether the particular attribute at that index must be added or not
- **clusters:** List of cluster objects obtained after clustering
- **df:** The Pandas dataframe obtained by loading the relevant CSV file
- **clustering_metadata:** Python dictionary that contains all the metadata regarding the clustering process
- **pearling_metadata:** Python dictionary that contains all the metadata regarding the pearl creation process within each cluster

- **return_clusters:** The clustering algorithm function that will be used to create clusters out of the dataset
- **number_of_clusters:**The number of clusters that need to be created (in the case of a supervised clustering algorithm)
- **clustering_metadata_keys:** All the keys in the clustering metadata details sent from the frontend that are relevant to the cluster creation process
- **pearling_metadata_keys:** All the keys in the clustering metadata details sent from the frontend that are relevant to the Pearl creation process

Methods:

- **load_dataset:**

Description:
Function that loads the dataset, extracts attribute list, number of dimensions (attributes) and drops rows with missing values.

Parameters:
None

Output:
None

- **set_metadata:**

Description:
A function to set clustering and pearling metadata from the input received from the user.

Parameters:
- **metadata:** The metadata required for the pearling process, in JSONform, to be extracted into python dictionaries.

Output:
None

- **set_clustering_algorithm:**

Description:
A function that sets the desired clustering algorithm from the clustering strategy.

Parameters:
None

Output:
None

- **create_clusters:**

Description:
Function to trigger the cluster and pearl formation of the dataset. It additionally sets the **number_of_clusters** variable in the case that the algorithm doesn't take numbers of clusters as an input. After cluster creation, the function triggers the pearl creation for each cluster as well.

Parameters:
None

Output:
None

- **get_attribute_list:**

Description:
Function that returns list of attributes of the dataset

Parameters:
None

Output:

A list consisting of attributes of the dataset.

- **filter_attributes:**

Description:
Function that sets the selected_attributes based on a boolean array inputted to it.

Parameters:
- **selected_attributes:** A boolean array that corresponds to whether the particular attribute at that index must be added or not

Output:
None

- **drop_attributes:**

Description:
A function that drops the attributes the user does not require from the input dataframe.

Parameters:
- **df:** The dataframe whose attributes need to be filtered

Output:
A dataframe without the columns that are not required.

PearlPositioning/PositioningModule.py
This class contains the Positioning Module class.

class PositioningModule

Description:
A class that serves as a container class for multiple static methods that are required in calculating the 3D-projected coordinates of a point with respect to a given cluster centroid.

Methods: (Note: All methods are static methods)

- **variance:**

Description:
Function that calculates the "variance"/squared distance of each dimension between two vectors

Parameters:
- **point0**
- **point1**

The above two parameters are the two points for which the squared distance has to be calculated.

Output:
A numpy array that represents the squared distance vector between the two points

- **get_cos_phi:**

Description:
Function that calculates the cosine of the projection angle phi of the pearl centre

Parameters:
- **pearl:** The pearl whose centre's 3D-projected coordinates have to be calculated
- **cluster_centroid:** The centroid of the cluster enclosing the current pearl. The above two parameters are the two points for which the squared distance has to be calculated.

Output:
The value of cos_phi of the centroid of the pearl, a decimal.

- **project_point_in_3D:**

Description:
Function that calculates the 3D-projected coordinates of the point provided as input with respect to the cluster centroid being the origin.

Parameters:

point: The point whose centre's 3D-projected coordinates have to be calculated
cluster_centroid: The centroid of the cluster enclosing the current point
cos_phi: The value of cos(phi), where phi refers to the inclination angle of the point (in spherical coordinates)
p: The "p" to calculate the (p-)distance between two points (by default p = 2)

Output:
The value of cos_phi of the centroid of the pearl, a decimal.

ShapeIdentifier/PearlShapeIdentifier.py
This class contains the shape identification module.

class PearlShapeIdentifier
Description:
A class that serves as a container class for multiple static methods that are required in calculating the shape of a particular pearl.

Methods: (Note: All methods are static methods)

- **find_distances_from_centroid:**

Description:
Function that finds the p-distances of all points in the given data from the centroid of the data.
Parameters:
- **data:** The multidimensional data for whose data points the p-distance has to be calculated
- **centroid:** The centroid of the data
- **p:** The value of p to calculate the norm to calculate the distance between a data point and the centroid

Output:
A numpy array that represents the squared distance vector between the two points

- **remove_10_percent_points:**

Description:
Function that removes the farthest 10% of points from the data to avoid outliers during shape calculation.
Parameters:
- **data:** The data from which the points must be removed
- **centroid:** The centroid of the data being considered.

Output:
90% of the points chosen in such a way that they are closer to the centroid than the remaining 10% points that are being removed.

- **project_point_in_3D:**

Description:
Function that calculates the 3D-projected coordinates of the point provided as input with respect to the cluster centroid being the origin.
Parameters:
- **point:** The point whose centre's 3D-projected coordinates have to be calculated
- **cluster_centroid:** The centroid of the cluster enclosing the current point
- **cos_phi:** The value of cos(phi), where phi refers to the inclination angle of the point (in spherical coordinates)
- **p:** The "p" to calculate the (p-)distance between two points (by default p = 2)

Output:
The value of cos_phi of the centroid of the pearl, a decimal.

- **find_farthest_distance:**

Description:
Function to find the farthest distance of all points in the data to calculate the radius of the pearl.

Parameters:
- **data:** The pearl whose centre's 3D-projected coordinates have to be calculated
- **centroid:** The centroid of the cluster enclosing the current pearl. The above two parameters are the two points for which the squared distance has to be calculated.
- **p:** The value of p to calculate the norm to calculate the distance between a data point and the centroid

Output:
The maximum of distances of all points in the data.

- **calculate_volume_constant:**

Description:
Function that calculates the volume of the pearl (according to the algorithm in the pearls thesis).

Parameters:
- **point_tuple:** This consists of (farthest_distance, P), where farthest_distance is the the maximum of distances of all points in the data, and P is the P (from the P_list of calculate_shape)
- **n_dim:** The number of dimensions of the data being considered.

Output:
The value of cos_phi of the centroid of the pearl, a decimal.

- **calculate_shape:**

Description:
Function that calculates the shape and radius of a pearl

Parameters:
- **pearl_data:**The data of the pearl being considered
- **pearl_centroid:** The centroid of the pearl being considered.

Output:
A best_P (whose value determines the shape of the pearl), and the best-fit radius of the pearl

ClusteringAlgos:

This folder consists of all files related to the clustering algorithms that are involved in the cluster/ pearl formation process.

ClusteringAlgorithm.py

This file consists of the template any custom clustering algorithm to be added to the systems.

class ClusteringAlgoTemplate:

Description:

Abstract class defining the template of any clustering algorithm class. Any clustering algorithm class must inherit from this class as the base class.

The class must have atleast one method: **create_clusters.**

create_clusters must have the following parameters:

- **Data:** A pandas dataframe that needs to be classified
- **metadata:** Parameters required for the clustering algorithm as a dictionary (has to be extracted before using)
- **cluster_or_pearl:** can take the value either "Cluster" or “Pearl" depending on what object is being formed, so that the appropriate parameters can be extracted from the metadata.

Output:

The output should be of the form

return cluster_labels, number_of_clusters

Where cluster_labels is a list of integer labels (starting from 0) for each point (in the order of the points in the data), and the number of pearls is an integer, denoting the number of clusters. In case the clustering algorithm can result in labelling of points as outliers, the outliers should have a label of -1.

ClusteringStrategy.py

A file containing the clustering strategy pattern class.

class ClusteringStrategy

Description:

The strategy-pattern class that helps in setting the clustering algorithm for either the clustering or pearling process. The function that returns a function based on the algorithm that the user demands.

Methods:

set_clustering_algorithm:

Parameters:

algorithm : A string key that refers to a certain clustering algorithm (context variable in the strategy pattern)

Output:

The function that the user has requested for via the algorithm variable. This is done via a dictionary of clustering algorithms. For details about how to add a new clustering algorithm into the system go to ‘How to add a new clustering algorithm to the system’.

Clustering algorithms:

KMeans.py
A file containing the class for the KMeans algorithm.

class KMeansClustering
Description:
The KMeans algorithm used here is the version implemented in scikit-learn. For details about the create_clusters method, check the **ClusteringAlgoTemplate** class.
Methods:
create_clusters

KrNN.py
A file containing the class for the KrNN algorithm.
Helper methods:

- **depth_first_search**

Description:
A simple depth first search function that builds a depth first search stack for the strongly connected components algorithm.
Parameters:
- **graph:** A graph object for which the DFS must be run
- **starting_node:** The starting node index for the DFS
- **stack:** The DFS stack that is being constructed

Output:
None

- **strongly_connected_components**

Description:
Function to find the strongly connected components of a graph
Parameters:
graph: A graph object for which the strongly connected components need to be generated.
Output:
A list of lists, each list corresponding to the indices of vertices belonging to a single strongly connected component.

class KrNNClustering
Description:
A class that represents a graph data structure.

Attributes:
- **graph_dict:** An adjacency list of the graph
- **number_of_nodes:** Number of nodes in the graph
- **visited_list:** A list that says whether a node was visited, for the strongly connected components algorithm

Methods:

reversed_graph:
Description:
Function that returns a graph, which has the same nodes and edges as the original graph, but the edges are reversed in direction.
Parameters:
None

Output:
A Graph object with the edges directing in the opposite directions of the original graph

class KrNNClustering
Description:
The KrNN algorithm in this code was written from scratch. For more details on the algorithm, refer to "Reverse Nearest Neighbors Driven Clustering and Visualization of High Dimensional Data - Soujanya Vadapalli, 2010".

Methods:

- **construct_KrNN_list**

Description:
Function to create the reverse neighbour list given the neighbour list
Parameters:
neighbour_index_list: A dictionary which consists of the neighbour list of all points in the graph
Output:
A dictionary which consists of the reverse-neighbour list of all points in the graph

- **identify_densepoints_outliers**

Description:
Function that segregates points in data as dense points or outliers on the basis of the length of their reverse-neighbour list.
Method parameters:
- **KrNN_list:** A dictionary which consists of the reverse-neighbour list of all points in the graph
- **k:** The parameter k in the KrNN clustering algorithm

Output:
Two dictionaries, corresponding to dense points and their adjacency lists, and outliers and their adjacency lists.

- **create_clusters**

4. Details about the clustered json data

Structure of the cluster data that is being returned:

```
{
      "cluster_number": 0,
      "centroid": centroid_obj,
      "pearl_list": [List of pearl objects]
}
```

`centroid_obj:` It is a JSON object denoting the centroid of the cluster. The list of keys in centroid_obj consists of all numerical (int/float-valued columns) with the corresponding value.

Structure of a pearl object in the `pearl_list` is as follows:

```
{
      "pearl_number": The index of the pearl,
      "centroid": Centroid of the pearl, similar to "centroid" in the cluster object ,
      "pearl_P": The value of 'P' calculated by the algorithm, which is an indicator of the
shape of the pearl,
      "pearl_radius": The radius of the pearl calculated by the algorithm, a decimal,
```

"pearl_centroid_3D": A list consisting of the x, y, z coordinates of the pearl's centroid coordinates projected in 3D with respect to its parent cluster's centroid as origin,

"pearl_list":
A list of objects of the form
{"Index_of_row_in_table": {...}}

The value of that JSON object within a pearl_list constituent object has the keys as list of all attributes in the table, and the values are the entries in the row corresponding to the index given in "Index_of_row_in_table".

5. How to add a new clustering algorithm to the system

In the backend:

1. First, add a class that inherits from the ClusteringAlgoTemplate class, following all conventions given in the documentation for the same. Add the file to the ClusteringAlgos folder.
2. Add the clustering algorithm to the ClusteringStrategy class's algorithm set in the **`set_clustering_algorithm`** function.
3. Finally, add the appropriate keys for your algorithm in the below shown attributes of the Cluster and Dataset class.

```
self.pearling_metadata_keys = [
    'pearl_clustering_algo',
    'number_of_pearls',
    'KrNN_k_for_pearling',
    'binning_dimension',
    'binning_criterion',
    'bins_per_cluster',
]
```

```
# Keys related to the clustering algorithm
self.clustering_metadata_keys = [
    'clustering_algo',
    'number_of_clusters',
    'KrNN_k_for_clustering',
]

# Keys related to the pearl formation algorithm
self.pearling_metadata_keys = [
    'pearl_clustering_algo',
    'number_of_pearls',
    'KrNN_k_for_pearling',
    'binning_dimension',
    'binning_criterion',
    'bins_per_cluster',
]
```

The pearling_metadata_keys in the Cluster class (left), and the clustering_metadata_keys and pearling_metadata_keys in the Dataset class (right)