PTRAIL

Release 0.7 Beta

Yaksh J Haranwala, Salman Haidri

Apr 08, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

PTRAIL is a state-of-the art Mobility Data Preprocessing Library that mainly deals with filtering data, generating features and interpolation of Trajectory Data.

The main features of PTRAIL are:

1. PTRAIL use primarily parallel computation based on python Pandas and numpy which makes it very fast as compared to other comparable libraries available.
2. PTRAIL harnesses the full power of the machine that it is running on by using all the cores available in the computer.
3. Four different kinds of Trajectory Interpolation techniques are offered by PTRAIL which is a first in the community.

CHAPTER

TWO

REFERENCES

2.1 ptrail

2.1.1 ptrail.core package

2.1.1.1 Submodules

2.1.1.2 ptrail.core.Datasets module

The Datasets.py module is used to load built-in datasets to variables. All the datasets loaded are stored and returned in a PTRAILDataFrame Currently, the library has the following datasets available to use:

1. Atlantic Hurricanes Dataset
2. Traffic Dataset (a smaller subset)
3. Geo-life Dataset (a smaller subset)
4. Seagulls Dataset
5. Ships Dataset (a smaller subset)
6. Starkey Animals Dataset
7. Starkey Habitat Dataset (accompanies the starkey dataset)

   The Starkey Habitat Dataset is not loaded is not loaded into a PTrailDataframe since it is not a movement dataset and rather contains contextual information about the starkey habitat. It is rather loaded into a pandas dataframe and returned as is.

Authors: Yaksh J Haranwala

class ptrail.core.Datasets.**Datasets**

Bases: `object`

static load_geo_life_sample()

Load the Geo-Life Sample dataset into the PTRAILDataFrame and return it.

Returns The geo-life sample dataset loaded into a PTrailDataFrame.

Return type *PTRAILDataFrame*

static load_hurricanes()

Load the Atlantic Hurricane dataset into the PTRAILDataFrame and return it.

Returns The atlantic hurricanes dataset loaded into a PTrailDataFrame.

Return type *PTRAILDataFrame*

`static load_seagulls()`

Load the Sea-Gulls dataset into the PTRAILDataFrame and return it.

Returns The seagulls dataset loaded into a PTrailDataFrame.

Return type *PTRAILDataFrame*

`static load_ships()`

Load the Sea-Gulls dataset into the PTRAILDataFrame and return it.

Returns The Ships dataset loaded into a PTrailDataFrame.

Return type *PTRAILDataFrame*

`static load_starkey()`

Load the Starkey dataset into the PTRAILDataFrame and return it.

Returns The Starkey dataset loaded into a PTrailDataFrame.

Return type *PTRAILDataFrame*

`static load_starkey_habitat()`

Load the Starkey dataset into a pandas dataframe and return it.

Returns The Starkey habitat dataset.

Return type *PTRAILDataFrame*

`static load_traffic_data()`

Load the Traffic dataset into the PTRAILDataFrame and return it.

Returns The traffic dataset loaded into a PTrailDataFrame.

Return type *PTRAILDataFrame*

2.1.1.3 ptrail.core.TrajectoryDF module

The TrajectoryDF module is the main module containing the PTRAILDataFrame Dataframe for storing the Trajectory Data with PTRAIL Library. The Dataframe has certain restrictions on what type of data is mandatory in order to be stored as a PTRAILDataFrame which is mentioned in the documentation of the constructor.

Authors: Yaksh J Haranwala, Salman Haidri

`class ptrail.core.TrajectoryDF.PTRAILDataFrame(*args: Any, **kwargs: Any)`

Bases: `pandas.DataFrame`

`__init__(data_set: Union[pandas.DataFrame, List, Dict], latitude: str, longitude: str, datetime: str, traj_id: str, rest_of_columns: Optional[List[str]] = None)`

Construct a trajectory dataframe to store and represent the Trajectory Data.

Note:

The mandatory columns in the dataset are:

1. DateTime
2. Trajectory ID

3. Latitude
4. Longitude

`rest_of_columns` makes sure that if the data_set is a list, it has appropriate headers that the user wants instead of the default numerical values.

Parameters

- **data_set** (*List, Dictionary or pandas DF.*) – The data provided by the user that needs to be represented and stored.
- **datetime** (*str*) – The header of the datetime column.
- **traj_id** (*str*) – The header of the Trajectory ID column.
- **latitude** (*str*) – The header of the latitude column.
- **longitude** (*str*) – The header of the longitude column.
- **rest_of_columns** (*Optional[list[Text]]*) – A list containing headers of the columns other than the mandatory ones.

property datetime

Accessor method for the DateTime column of the PTRAILDataFrame DataFrame.

Returns The Series containing all the DateTime values from the DataFrame.

Return type pandas.core.series.Series

Raises *MissingColumnsException* – DateTime column is missing from the data.

property latitude

Accessor method for the latitude column of the PTRAILDataFrame DataFrame.

Returns The Series containing all the latitude values from the DataFrame.

Return type pandas.core.series.Series

Raises *MissingColumnsException* – Latitude column is missing from the data.

property longitude

Accessor method for the longitude column of the PTRAILDataFrame DataFrame.

Returns The Series containing all the longitude values from the DataFrame.

Return type pandas.core.series.Series

Raises *MissingColumnsException* – Longitude column is missing from the data

set_default_index()

Set the Index of the dataframe back to traj_id and DateTime.

Raises *MissingColumnsException* – DateTime/traj_id column is missing from the dataset.

sort_by_traj_id_and_datetime(*ascending=True*)

Sort the trajectory in Ascending or descending order based on the following 2 columns in order:

1. Trajectory ID
2. DateTime

Parameters **ascending** (*bool*) – Whether to sort the values in ascending order or descending order.

Returns The sorted dataframe.

Return type *PTRAILDataFrame*

to_numpy(*dtype=None*, *copy: bool = False*, *na_value=pandas._libs.lib.no_default*) → numpy.ndarray

Convert the DataFrame to a NumPy array.By default, the dtype of the returned array will be the common dtype of all types in the DataFrame. For example, if the dtypes are float16 and float32, the results dtype will be float32. This may require copying data and coercing values, which may be expensive

Parameters

- **dtype** – The dtype to pass to `numpy.asarray()`.
- **copy** – Whether to ensure that the returned value is not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.
- **na_value** – The value to use for missing values. The default value depends on *dtype* and the dtypes of the DataFrame columns.

property traj_id

Accessor method for the Trajectory_ID column of the DaskTrajectoryDF.

Returns The Series containing all the Trajectory_ID values from the DataFrame.

Return type pandas.core.series.Series

Raises *MissingColumnsException* – traj_id column is missing from the data.

2.1.1.4 Module contents

2.1.2 ptrail.features package

2.1.2.1 Submodules

2.1.2.2 ptrail.features.contextual_features module

The semantic features module contains several semantic features like intersection of trajectories, stop and stay point detection. Moreover, features like distance from Point of Interests, water bodies and other demographic features related to the trajectory data are calculated. The demographic features are extracted with the help of the python osmnx library.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.features.contextual_features.**ContextualFeatures**

Bases: `object`

static nearest_poi(*coords: tuple*, *dist_threshold*, *tags: dict*)

Given a coordinate point and a distance threshold, find the Point of Interest which is the nearest to it within the given distance threshold.

Warning: The users are advised the be mindful about the tags being passed in as parameter. More the number of tags, longer will the OSMNx library take to download the information from the OpenStreet-Network maps. Moreover, an active internet connection is also required to execute this function.

Note: If several tags (POIs) are given in, then the method will find the closest one based on the distance and return it and will not given out the others that may or may not be present within the threshold of the given point.

Parameters

- **coords** (*tuple*) – The point near which the bank is to be found.
- **dist_threshold** – The maximum distance from the point within which the distance is to be calculated.
- **tags** (*dict*) – The dictionary containing tags of Points of interest.

Returns A pandas DF containing the info about the nearest bank from the given point.

Return type pandas.core.dataframe.DataFrame

Raises **JSONDecodeError:** – One or more given tags are invalid.

static traj_intersect_inside_polygon(*df1:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *df2:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *polygon:* *shapely.geometry.Polygon*)

Given a df1 and df2 containing trajectory data along with polygon, check whether the trajectory/trajectories are inside the polygon and if they are, whether the intersect at any point or not.

Warning: While creating a polygon, the format of the coordinates is: (longitude, latitude) instead of (latitude, longitude). Beware of that, otherwise the results will be incorrect.

Note: It is to be noted that df1 and df2 should only contain trajectory data of only one trajectory each. If they contain more than one trajectories, then the results might be unexpected.

Parameters

- **df1** (PTRAILDataFrame) – Trajectory Dataframe 1.
- **df2** (PTRAILDataFrame) – Trajectory Dataframe 2.
- **polygon** (*Polygon*) – The area inside which it is to be determined if the trajectories intersect or not.

Returns

- *PTRAILDataFrame* – A dataframe containing trajectories that are inside the polygon.
- *geopandas.GeoDataFrame* – An empty dataframe if both the trajectories do not intersect.

static trajectories_inside_polygon(*df:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *polygon: shapely.geometry.Polygon*)

Given a trajectory dataframe and a Polygon, find out all the trajectories that are inside the given polygon.

Warning: While creating a polygon, the format of the coordinates is: (longitude, latitude) instead of (latitude, longitude). Beware of that, otherwise the results will be incorrect.

Parameters

- **df** (PTRAILDataFrame) – The dataframe containing the trajectory data.
- **polygon** (*Polygon*) – The polygon inside which the points are to be found.

Returns A dataframe containing trajectories that are inside the polygon.

Return type *PTRAILDataFrame*

static visited_location(*df:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *geo_layers: Union[pandas.DataFrame, geopandas.GeoDataFrame]*, *visited_location_name: str*, *location_column_name: str*)

Create a column called visited_Location for all the pastures present in the dataset.

Warning: While using this method, make sure that the geo_layers parameter dataframe that is being passed into the method has Latitude and Longitude columns with columns named as 'lat' and 'lon' respectively. If this format is not followed then a KeyError will be thrown.

Note: It is to be noted that depending on the size of the dataset and the surrounding data passed in, this function will take longer time to execute if either of the datasets is very large. It has been parallelized to make it faster, however, it can still take a longer time depending on the size of the data being analyzed.

Parameters

- **df** (PTRAILDataFrame) – The dataframe containing the dataset.
- **geo_layers** (*Union[pd.DataFrame, gpd.GeoDataFrame]*) – The Dataframe containing the geographical layers near the trajectory data. It is to be noted
- **visited_location_name** (*Text*) – The location for which it is to be checked whether the objected visited it or not.
- **location_column_name** (*Text*) – The name of the column that contains the location to be checked.

Returns The Dataframe containing a new column indicating whether the animal has visited the pasture or not.

Return type *PTRAILDataFrame*

Raises **KeyError:** – The column or the location name does not exist.

static visited_poi(*df:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *surrounding_data: Union[geopandas.GeoDataFrame, pandas.DataFrame, ptrail.core.TrajectoryDF.PTRAILDataFrame]*, *dist_column_label: str*, *nearby_threshold: int*)

Given a surrounding data with information about the distance to the nearest POI source from a given coordinate, check whether the objects in the given trajectory data have visited/crossed those POIs or not

Warning: It is to be noted that for this method to work, the surrounding dataset NEEDS to have a column containing distance to the nearest POI. For more info, see the Starkey habitat dataset which has the columns like 'DistCWat' and 'DistEWat'.

Parameters

- **df** (PTRAILDataFrame) – The dataframe containing the trajectory data.
- **surrounding_data** (*Union[gpd.GeoDataFrame, pd.DataFrame]*) – The surrounding data that needs to contain the information of distance to the nearest water body.
- **dist_column_label** (*Text*) – The name of the column containing the distance information.
- **nearby_threshold** (*int*) – The maximum distance between the POI and the current location of the object within which the object is considered to be crossing/visiting the POI.

Returns The dataframe containing the new column indicating whether the object at that point is near.

Return type *PTRAILDataFrame*

2.1.2.3 ptrail.features.helper_functions module

This module contains all the helper functions for the parallel calculations in the spatial and temporal features classes.

Warning: These functions should not be used directly as they would result in a slower calculation and execution times. In some cases, these functions might even yield wrong results if used directly. They are meant to be used only as helpers. For calculation of features, use the ones in the features package.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.features.helper_functions.**Helpers**

Bases: object

static bearing_helper(*dataframe*)

This function is the helper function of the create_bearing_column(). The create_bearing_column() delegates the task of calculation of bearing between 2 points to this function because the original functions runs multiple instances of this function in parallel. This function does the calculation of bearing between 2 consecutive points in the entire DF and then creates a column in the dataframe and returns it.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the calculation is to be done.

Returns The dataframe containing the Bearing column.

Return type *PTRAILDataFrame*

static distance_between_consecutive_helper(*dataframe*)

This function is the helper function of the create_distance_between_consecutive_column() function. The create_distance_between_consecutive_column() function delegates the actual task of calculating the distance between 2 consecutive points. This function does the calculation and creates a column called Distance_prev_to_curr and places it in the dataframe and returns it.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which calculation is to be performed.

Returns The dataframe containing the resultant Distance_prev_to_curr column.

Return type *core.TrajectoryDF.PTRAILDataFrame*

References

Arina De Jesus Amador Monteiro Sanches. 'Uma Arquitetura E Imple-menta c ao Do M odulo De Pr e-processamento Para Biblioteca Pymove'.Bachelor's thesis. Universidade Federal Do Cear a, 2019.

static distance_from_given_point_helper(*dataframe*, *coordinates*)

This function is the helper function of the create_distance_from_point() function. The create_distance_from_point() function delegates the actual task of calculating distance between the given point to all the points in the dataframe to this function. This function calculates the distance and creates another column called 'Distance_to_specified_point'.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which calculation is to be done.
- **coordinates** (*tuple*) – The coordinates from which the distance is to be calculated.

Returns The dataframe containing the resultant Distance_from_(x, y) column.

Return type pandas.core.dataframe.DataFrame

static distance_from_start_helper(*dataframe*)

This function is the helper function of the create_distance_from_start_column() function. The create_distance_from_start_column() function delegates the actual task of calculating the distance between 2 the start point of the trajectory to the current point.This function does the calculation and creates a column called Distance_start_to_curr and places it in the dataframe and returns it.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which calculation is to be performed.

Returns The dataframe containing the resultant Distance_start_to_curr column.

Return type pandas.core.dataframe

static end_location_helper(*dataframe*, *ids_*)

This function is the helper function of the get_end_location(). The get_end_location() function delegates the task of calculating the end location of the trajectories in the dataframe because the original functions runs multiple instances of this function in parallel. This function finds the end location of the specified trajectory IDs the DF and then another returns dataframe containing end latitude, end longitude and trajectory ID for each trajectory

dataframe: PTRAILDataFrame The dataframe of which the locations are to be found.dataframe

ids_: list List of trajectory ids for which the end locations are to be calculated

Returns New dataframe containing Trajectory ID as index and latitude and longitude as other 2 columns.

Return type pandas.core.dataframe.Dataframe

static end_time_helper(*dataframe*, *ids_*)

This function is the helper function of the get_end_time(). The get_end_time() function delegates the task of calculating the end_time of the trajectories in the dataframe because the original functions runs multiple instances of this function in parallel. This function finds the end time of the specified trajectory IDs the DF and then another returns dataframe containing end latitude, end longitude, DateTime and trajectory ID for each trajectory

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing the original data.
- **ids** (*list*) – List of trajectory ids for which the end times are to be calculated

Returns New dataframe containing Trajectory ID as index end time of all trajectories.

Return type pandas.core.dataframe.Dataframe

static number_of_location_helper(*dataframe*, *ids_*)

This is the helper function for the get_number_of_locations() function. The get_number_of_locations() delegates the actual task of calculating the number of unique locations visited by a particular object to this function. This function calculates the number of unique locations by each of the unique object and returns a dataframe containing the results.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing all the original data.
- **ids** (*list*) – The list of ids for which the number of unique locations visited is to be calculated.

Returns dataframe containing the results.

Return type pandas.core.dataframe.DataFrame

static point_within_range_helper(*dataframe*, *coordinates*, *dist_range*)

This is the helper function for create_point_within_range() function. The create_point_within_range_column()

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which the operation is to be performed.
- **coordinates** (*tuple*) – The coordinates from which the distance is to be checked.
- **dist_range** – The range within which the distance from the coordinates should lie.

Returns The dataframe containing the resultant Within_X_m_from_(x,y) column.

Return type pandas.core.dataframe.DataFrame

static start_location_helper(*dataframe*, *ids_*)

This function is the helper function of the get_start_location(). The get_start_location() function delegates the task of calculating the start location of the trajectories in the dataframe because the original functions runs multiple instances of this function in parallel. This function finds the start location of the specified trajectory IDs the DF and then another returns dataframe containing start latitude, start longitude and trajectory ID for each trajectory

dataframe: PTRAILDataFrame The dataframe of which the locations are to be found.dataframe

ids_: list List of trajectory ids for which the start locations are to be calculated

Returns New dataframe containing Trajectory as index and latitude and longitude

Return type pandas.core.dataframe.Dataframe

`static start_time_helper`(*dataframe*, *ids_*)

This function is the helper function of the get_start_time(). The get_start_time() function delegates the task of calculating the start_time of the trajectories in the dataframe because the original functions runs multiple instances of this function in parallel. This function finds the start time of the specified trajectory IDs the DF and then another returns dataframe containing start latitude, start longitude, DateTime and trajectory ID for each trajectory

dataframe: PTRAILDataFrame The dataframe containing the original data.

ids_: list List of trajectory ids for which the start times are to be calculated

Returns New dataframe containing Trajectory ID as index and start time of all trajectories.

Return type pandas.core.dataframe.Dataframe

`static traj_duration_helper`(*dataframe*, *ids_*)

Calculate the duration of the trajectory i.e. subtract the max time of the trajectory by the min time of the trajectory.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing all the original data.
- **ids** (*list*) – A list containing all the Trajectory IDs present in the dataset.

Returns The resultant dataframe containing all the trajectory durations.

Return type pandas.core.dataframe.DataFrame

`static visited_poi_helper`(*df*, *surrounding_data*, *dist_column_label*, *nearby_threshold*)

Given a Trajectory dataframe and another dataset with the surrounding data, find whether the given object is nearby a point of interest or not.

Parameters

- **df** – The dataframe containing the trajectory data.
- **surrounding_data** – The dataframe containing the data of the surroundings.
- **dist_column_label** (*Text*) – The label of the column containing the distance of the coords from the nearest POI.
- **nearby_threshold** (*int*) – The maximum distance between the POI and the current location of the object within which the object is considered to be crossing/visiting the POI.

Returns

- *The original dataframe with another column added to it indicating whether*
- *each point is within*

2.1.2.4 ptrail.features.kinematic_features module

The spatial_features module contains several functions of the library that calculates kinematic features based on the coordinates of points provided in the data. This module mostly extracts and modifies data collected from some existing dataframe and appends these information to them. Inspiration of lots of functions in this module is taken from the PyMove library.

Authors: Yaksh J Haranwala, Salman Haidri

References

Arina De Jesus Amador Monteiro Sanches. "Uma Arquitetura E Imple-menta c ao Do M odulo De Pr e-processamento Para Biblioteca Pymove".Bachelor's thesis. Universidade Federal Do Cear a, 2019

class ptrail.features.kinematic_features.**KinematicFeatures**

Bases: `object`

static create_acceleration_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column containing acceleration of the object from the previous to the current point.

Note: The acceleration calculated here is the acceleration between 2 consecutive points of the same trajectory. Furthermore, the acceleration yielded is in metres/second^2 (m/s^2).

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the calculation of acceleration is to be done.

Returns The dataframe containing the resultant Acceleration_prev_to_curr column.

Return type *PTRAILDataFrame*

static create_bearing_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column containing bearing between 2 consecutive points. Bearing is also referred as "Forward Azimuth" sometimes. Bearing/Forward Azimuth is defined as follows:

Bearing is the horizontal angle between the direction of an object and another object, or between the object and the True North.

Note: The bearing calculated here is the bearing between 2 consecutive points of the same trajectory. Furthermore, the bearing yielded is in degrees.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the bearing is to be calculated.

Returns The dataframe containing the resultant Bearing_from_prev column.

Return type *PTRAILDataFrame*

static create_bearing_rate_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Calculates the bearing rate of the consecutive points. And adding that column into the dataframe

Note: The bearing calculated here is the bearing between 2 consecutive points of the same trajectory. Furthermore, the bearing yielded is in degrees/second.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the bearing rate is to be calculated

Returns The dataframe containing the resultant Bearing_rate_from_prev column.

Return type *PTRAILDataFrame*

static create_distance_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column called Dist_prev_to_curr containing distance between 2 consecutive points. The distance calculated is the Great-Circle (Haversine) distance.

Note: When the trajectory ID changes in the data, then the distance calculation again starts from the first point of the new trajectory ID and the distance-value of the first point of the new Trajectory ID will be set to 0.

Note: The Distance calculated here is the distance between 2 consecutive points of the same trajectory. Furthermore, the distance yielded is in metres (m).

Parameters **dataframe** (PTRAILDataFrame) – The data where distance is to be calculated.

Returns The dataframe containing the resultant Distance_prev_to_curr column.

Return type *PTRAILDataFrame*

static create_distance_from_point_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *coordinates: tuple*)

Given a point, this function calculates the distance between that point and all the points present in the dataframe and adds that column into the dataframe.

Note: The distance yielded here is in metres.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which calculation is to be done.
- **coordinates** (*tuple*) – The coordinates from which the distance is to be calculated.

Returns The dataframe containing the resultant Distance_from_(x, y) column.

Return type *PTRAILDataFrame*

static create_distance_from_start_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column containing distance between the start location and the rest of the points using Haversine formula. The distance calculated is the Great-Circle distance.

Note: When the trajectory ID changes in the data, then the distance calculation again starts from the first point of the new trajectory ID and the first distance of the new trajectory ID will be set to 0.

Note: The Distance calculated here is the distance between the start point and the current points of the same trajectory. Furthermore, the distance yielded is in metres (m).

Parameters **dataframe** (PTRAILDataFrame) – The data where distance is to be calculated.

Returns The dataframe containing the resultant Distance_start_to_curr column.

Return type *PTRAILDataFrame*

static create_jerk_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column containing jerk of the object from previous to the current point.

Note: The jerk calculated here is the jerk between 2 consecutive points of the same trajectory. Furthermore, the jerk yielded is in metres/second^3 (m/s^3).

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the calculation of jerk is to be done.

Returns The dataframe containing the resultant jerk_prev_to_curr column.

Return type *PTRAILDataFrame*

static create_point_within_range_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *coordinates: tuple*, *dist_range: float*)

Check how many points are within the range of the given coordinate by first making a column containing the distance between the given coordinate and rest of the points in dataframe by calling create_distance_from_point() and then comparing each point using the condition if it's within the range and appending the values in a column and attaching it to the dataframe.

Note: The dist_range parameter is given in metres.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which the point within range calculation is to be done.
- **coordinates** (*tuple*) – The coordinates from which the distance is to be calculated.
- **dist_range** (*float*) – The range within which the resultant distance from the coordinates should lie.

Returns The dataframe containing the resultant Within_x_m_from_(x,y) column.

Return type *PTRAILDataFrame*

static create_rate_of_br_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Calculates the rate of bearing rate of the consecutive points. And then adding that column into the dataframe.

Note: The rate of bearing rate calculated here is the rate of bearing rate between 2 consecutive points of the same trajectory. Furthermore, the bearing yielded is in degrees.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the rate of bearing rate is to be calculated

Returns The dataframe containing the resultant Rate_of_bearing_rate_from_prev column

Return type *PTRAILDataFrame*

static create_speed_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column containing speed of the object from the previous point to the current point.

Note: When the trajectory ID changes in the data, then the speed calculation again starts from the first point of the new trajectory ID and the speed of the first point of the new trajectory ID will be set to 0.

Note: The Speed calculated here is the speed between 2 consecutive points of the same trajectory. Furthermore, the speed yielded is in metres/second (m/s).

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the calculation of speed is to be done.

Returns The dataframe containing the resultant Speed_prev_to_curr column.

Return type *PTRAILDataFrame*

static distance_travelled_by_date_and_traj_id(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *date*, *traj_id*)

Given a date and trajectory ID, calculate the total distance covered in the trajectory on that particular date.

Note: The distance yielded is in metres (m).

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe in which teh actual data is stored.
- **date** (*Text*) – The Date on which the distance covered is to be calculated.
- **traj_id** (*Text*) – The trajectory ID for which the distance covered is to be calculated.

Returns The total distance covered on that date by that trajectory ID.

Return type float

Raises **KeyError:** – Traj_id is not present in the arguments passed.

static generate_kinematic_features(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Generate all the Kinematic features with a single call of this function.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the features are to be generated.

Returns The dataframe enriched with Kinematic Features.

Return type *PTRAILDataFrame*

static get_bounding_box(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Return the bounding box of the Trajectory data. Essentially, the bounding box is of the following format:

(min Latitude, min Longitude, max Latitude, max Longitude).

Parameters **dataframe** (PTRAILDataFrame) – The dataframe containing the trajectory data.

Returns The bounding box of the trajectory

Return type tuple

static get_distance_travelled_by_traj_id(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: str*)

Given a trajectory ID, calculate the total distance covered by the trajectory. NOTE: The distance calculated is in metres.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing the entire dataset.
- **traj_id** (*Text*) – The trajectory ID for which the distance covered is to be calculated.

Returns The distance covered by the trajectory

Return type float

Raises **MissingTrajIDException:** – The Trajectory ID given by the user is not present in the dataset.

static get_end_location(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: Optional[str] = None*)

Get the ending location of an object's trajectory in the data.

Note: If the user does not give in any traj_id, then the library, by default gives out the end locations of all the unique trajectory ids present in the data.

Parameters

- **dataframe** (PTRAILDataFrame) – The PTRAILDataFrame storing the trajectory data.
- **traj_id** – The ID of the trajectory whose end location is to be found.

Returns

- *tuple* – The (lat, longitude) tuple containing the end location.
- *pandas.core.dataframe.DataFrame* – The dataframe containing start locations of all trajectory IDs.

static get_number_of_locations(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: Optional[str] = None*)

Get the number of unique coordinates in the dataframe specific to a trajectory ID.

Note: If no Trajectory ID is specified, then the number of unique locations in the visited by each trajectory in the dataset is calculated.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe of which the number of locations are to be computed
- **traj_id** (*Text*) – The trajectory id for which the number of unique locations are to be found

Returns

- *int* – The number of unique locations in the dataframe/trajectory id.
- *pandas.core.dataframe.DataFrame* – The dataframe containing start locations of all trajectory IDs.

static get_start_location(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id=None*)

Get the starting location of an object's trajectory in the data.

Note: If the user does not give in any traj_id, then the library, by default gives out the start locations of all the unique trajectory ids present in the data.

Parameters

- **dataframe** (PTRAILDataFrame) – The PTRAILDataFrame storing the trajectory data.
- **traj_id** – The ID of the object whose start location is to be found.

Returns

- *tuple* – The (lat, longitude) tuple containing the start location.
- *pandas.core.dataframe.DataFrame* – The dataframe containing start locations of all trajectory IDs.

2.1.2.5 ptrail.features.temporal_features module

1. The temporal_features module contains all the features of the library that calculates several features based on the DateTime provided in the data.
2. It is to be noted that most of the functions in this module calculate the features and then add the results to an entirely new column with a new column header.
3. It is to be also noted that a lot of these features are inspired from the PyMove library and we are crediting the PyMove creators with them.

Authors: Yaksh J Haranwala, Salman Haidri

References

Arina De Jesus Amador Monteiro Sanches. “Uma Arquitetura E Imple-menta c ao Do M odulo De Pr e-processamento Para Biblioteca Pymove”.Bachelor’s thesis. Universidade Federal Do Cear a, 2019

class ptrail.features.temporal_features.**TemporalFeatures**

Bases: object

static create_date_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

From the DateTime column already present in the data, extract only the date and then add another column containing just the date.

Parameters **dataframe** (PTRAILDataFrame) – The PTRAILDataFrame Dataframe on which the creation of the time column is to be done.

Returns The dataframe containing the resultant Date column.

Return type *PTRAILDataFrame*

static create_day_of_week_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column called Day_Of_Week which contains the day of the week on which the trajectory point is recorded. This is calculated on the basis of timestamp recorded in the data.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe containing the entire data on which the operation is to be performed

Returns The dataframe containing the resultant Day_of_week column.

Return type *PTRAILDataFrame*

static create_time_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

From the DateTime column already present in the data, extract only the time and then add another column containing just the time.

Parameters **dataframe** (PTRAILDataFrame) – The PTRAILDataFrame Dataframe on which the creation of the time column is to be done.

Returns The dataframe containing the resultant Time column.

Return type *PTRAILDataFrame*

static create_time_of_day_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a Time_Of_Day column in the dataframe using parallelization which indicates at what time of the day was the point data captured. Note: The divisions of the day based on the time are provided in the utilities.constants module.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the calculation is to be done.

Returns The dataframe containing the resultant Time_Of_Day column.

Return type *PTRAILDataFrame*

References

Arina De Jesus Amador Monteiro Sanches. 'Uma Arquitetura E Imple-menta c ao Do M odulo De Pr e-processamento Para Biblioteca Pymove'.Bachelor's thesis. Universidade Federal Do Cear a, 2019.

static create_weekend_indicator_column(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Create a column called Weekend which indicates whether the point data is collected on either a Saturday or a Sunday.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the operation is to be performed.

Returns The dataframe containing the resultant Weekend column.

Return type *PTRAILDataFrame*

References

Arina De Jesus Amador Monteiro Sanches. 'Uma Arquitetura E Imple-menta c ao Do M odulo De Pr e-processamento Para Biblioteca Pymove'.Bachelor's thesis. Universidade Federal Do Cear a, 2019.

static generate_temporal_features(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Generate all the temporal features with a single call of this function.

Parameters **dataframe** (PTRAILDataFrame) – The dataframe on which the features are to be generated.

Returns The dataframe enriched with Temporal Features.

Return type *PTRAILDataFrame*

static get_end_time(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: Optional[str] = None*)

Get the ending time of the trajectory.

Note: If the trajectory ID is not specified by the user, then by default, the ending times of all the trajectory IDs in the data are returned.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which the operations are to be performed.
- **traj_id** (*Optional[Text]*) – The trajectory for which the end time is required.

Returns

- *pandas.DateTime* – The end time of a single trajectory.
- *pandas.core.dataframe.DataFrame* – Pandas dataframe containing the end time of all the trajectories present in the data when the user hasn't asked for a particular trajectory's end time.

static get_start_time(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: Optional[str] = None*)

Get the starting time of the trajectory.

Note: If the trajectory ID is not specified by the user, then by default, the starting times of all the trajectory IDs in the data are returned.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which the operations are to be performed.
- **traj_id** (*Optional[Text]*) – The trajectory for which the start time is required.

Returns

- *pandas.DateTime* – The start time of a single trajectory.
- *pandas.core.dataframe.DataFrame* – Pandas dataframe containing the start time of all the trajectories present in the data when the user hasn't asked for a particular trajectory's start time.

static get_traj_duration(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: Optional[str] = None*)

Accessor method for the duration of a trajectory specified by the user.

Note: If no trajectory ID is given by the user, then the duration of each unique trajectory is calculated.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing the resultant column if inplace is True.
- **traj_id** (*Optional[Text]*) – The trajectory id for which the duration is required.

Returns

- *pandas.TimeDelta* – The trajectory duration.
- *pandas.core.dataframe.DataFrame* – The dataframe containing the duration of all trajectories in the dataset.

2.1.2.6 Module contents

2.1.3 ptrail.GUI package

2.1.3.1 Submodules

2.1.3.2 ptrail.GUI.GUI_driver module

This file launches PTRAIL's GUI module.

Author: Yaksh J Haranwala

2.1.3.3 ptrail.GUI.InputDialog module

This class is an abstraction that can be used to create input dialog boxes for virtually any number of inputs.

Authors: Yaksh J Haranwala

class ptrail.GUI.InputDialog.**InputDialog**(**args: Any*, ***kwargs: Any*)

Bases: PyQt5.QtWidgets.QDialog

__init__(*labels: List[str]*, *title: str*, *placeHolders: List[str]*, *parent=None*)

getInputs()

2.1.3.4 ptrail.GUI.Table module

This python module is the abstract definition of the Table view for viewing the dataframe inside the GUI.

Authors: Yaksh J Haranwala

class ptrail.GUI.Table.**TableModel**(**args: Any*, ***kwargs: Any*)

Bases: PyQt5.QtCore.QAbstractTableModel

__init__(*data*)

columnCount(*parent=None*)

data(*index*, *role=PyQt5.QtCore.Qt.DisplayRole*)

headerData(*section*, *orientation*, *role=PyQt5.QtCore.Qt.DisplayRole*)

rowCount(*parent=None*)

2.1.3.5 ptrail.GUI.gui module

This module contains the design of PTRAIL's GUI module. It is to be noted that this class does not handle the functionalities, it is rather handled by the handler class.

Author: Yaksh J Haranwala

class ptrail.GUI.gui.**Ui_MainWindow**(**args: Any*, ***kwargs: Any*)

Bases: PyQt5.QtWidgets.QMainWindow

__init__(*OuterWindow*)

add_df_controller()

add_tree_options()

open_file()

Open the file and load the dataframe to perform operations.

Return type None

retranslateUi(*OuterWindow*)

Auto Generated method by PyQt Designer.

save_file()

Save the dataframe to a .csv file.

Return type None

setupUi(*OuterWindow*)

Set the main window of the GUI up and start the application.

Parameters **OuterWindow** (*PyQt5.QtWidgets.QOuterWindow'*) –

setup_command_palette()

Set up the pane that displays the command palette.

Return type None

setup_df_pane()

Set up the pane that displays the dataframe.

Return type None

setup_map_pane()

Set up the pane that displays the map.

Return type None

setup_menubar()

Create the menu bar of the window.

Return type None

setup_stats_palette()

Set up the pane that displays the statistics.

Return type None

setup_statusbar()

version_button_clicked()

Show the version info of the application.

Return type None

2.1.3.6 ptrail.GUI.handler module

This class is used to connect the PTRAIL GUI to PTRAIL backend. All the GUI's functionalities are handled in this class.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.GUI.handler.**GuiHandler**(*filename*, *window*)

Bases: object

__init__(*filename*, *window*)

add_column_drop_widget()

Add a List Widget to drop columns from the dataset. This widget is added to the CommandPalette.

Note: It is to be noted that the following columns are mandatory for PTrailDataFrame:

1. traj_id
2. DateTime
3. lat
4. lon

Hence, these columns are not presented as options for deletion.

---

display_df(*filename*)

Display the DataFrame on the DFPane of the GUI.

Parameters **filename** (*str*) – The name of the file. This is obtained from the GUI.

Raises **AttributeError:** – If the user gives incorrect column names, then we ask the user to enter them again.

draw_stats()

Handle the objects of the statistics pane from here.

drop_col()

Drop the columns based on the user selection.

generate_feature_imp_plot()

Take the input from the user and draw the mutual info plot.

redraw_map()

Redraw the map when the traj_id is changed from the DropDown list.

redraw_stat()

Redraw the statistics plot when the user changes the option from the Dropdown menu.

run_command()

When the user pushes the Run button, run the user's selected function on the dataset.

Return type None

update_dropCol_options()

Update the options in the QListWidget for dropping the columns.

2.1.3.7 Module contents

2.1.4 ptrail.preprocessing package

2.1.4.1 Submodules

2.1.4.2 ptrail.preprocessing.filters module

The filters module contains several data filtering functions like filtering the data based on time, date, proximity to a point and several others.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.preprocessing.filters.**Filters**

Bases: `object`

static filter_by_bounding_box(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *bounding_box: tuple*, *inside: bool = True*)

Given a bounding box, filter out all the points that are within/outside the bounding box and return a dataframe containing the filtered points.

Parameters

- **dataframe** (*PTRAILDataFrame*) – The dataframe from which the data is to be filtered out.
- **bounding_box** (*tuple*) – The bounding box which is to be used to filter the data.
- **inside** (*bool*) – Indicate whether the data outside the bounding box is required or the data inside it.

Returns The filtered dataframe.

Return type *PTRAILDataFrame*

static filter_by_date(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *start_date: Optional[str] = None*, *end_date: Optional[str] = None*)

Filter the dataset by user-given time range.

Note:

The following options are to be noted for filtering the data:

1. If the start_date and end_date both are not given, then entire dataset itself is returned.
2. If only start_date is given, then the trajectory data after (including the start date) the start date is returned.
3. If only end_date is given, then the trajectory data before (including the end date) the end date is returned.
4. If start_date and end_date both are given then the data between the start_date and end_date (included) are returned.

Parameters

- **dataframe** (*PTRAILDataFrame*) – The dataframe that is to be filtered.
- **start_date** (*Optional[Text]*) – The start date from which the points are to be filtered.

- **end_date** (*Optional[Text]*) – The end date before which the points are to be filtered.

Returns The filtered dataframe containing the resultant data.

Return type *PTRAILDataFrame*

Raises **ValueError:** – When the start date is later than the end date.

static filter_by_datetime(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *start_dateTime: Optional[str] = None*, *end_dateTime: Optional[str] = None*)

Filter the dataset by user-given time range.

Note:

The following options are to be noted for filtering the data.

1. If the start_dateTime and end_dateTime both are not given, then entire dataset itself is returned.
2. If only start_dateTime is given, then the trajectory data after (including the start datetime) the start date is returned.
3. If only end_dateTime is given, then the trajectory data before (including the end datetime) the end date is returned.
4. If start_dateTime and end_dateTime both are given then the data between the start_dateTime and end_dateTime (included) are returned.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe that is to be filtered.
- **start_dateTime** (*Optional[Text]*) – The start dateTime from which the points are to be filtered.
- **end_dateTime** (*Optional[Text]*) – The end dateTime before which the points are to be filtered.

Returns The filtered dataframe containing the resultant data.

Return type *PTRAILDataFrame*

Raises **ValueError:** – When the start datetime is later than the end datetime.

static filter_by_max_consecutive_distance(*dataframe*, *max_distance: float*)

Remove the points that have a distance between 2 consecutive points greater than a user specified value.

Note: max_distance is given in metres.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.
- **max_distance** (*float*) – The consecutive distance threshold above which the points are to be removed.

Returns The filtered dataframe.

Return type *PTRAILDataFrame*

static filter_by_max_distance_and_speed(*dataframe*, *max_distance: float*, *max_speed: float*)

Filter out values that have distance between consecutive points greater than a user-given distance and speed between consecutive points greater than a user-given speed

Note: The max_distance is given in metres

Note: The max_speed is given in metres/second (m/s).

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.
- **max_distance** (*float*) – The maximum distance between 2 consecutive points.
- **max_speed** (*float*) – The maximum speed between 2 consecutive points.

Returns The filtered dataframe.

Return type pandas.DataFrame

static filter_by_max_speed(*dataframe: ptrail.core.TrajectoryDF.PTRAILDataFrame*, *max_speed: float*)

Remove the data points which have speed more than a user given speed.

Note: The max_speed is given in the units m/s (metres per second).

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.
- **max_speed** (*float*) – The speed threshold above which the points are to be removed.

Returns PTRAILDataFrame Dataframe containing the resultant dataframe.

Return type *PTRAILDataFrame*

static filter_by_min_consecutive_distance(*dataframe*, *min_distance: float*)

Remove the points that have a distance between 2 consecutive points lesser than a user specified value.

Note: min_distance is given in metres.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.
- **min_distance** (*float*) – The consecutive distance threshold below which the points are to be removed.

Returns The filtered dataframe.

Return type *PTRAILDataFrame*

static filter_by_min_distance_and_speed(*dataframe*, *min_distance: float*, *min_speed: float*)

Filter out values that have distance between consecutive points lesser than a user-given distance and speed between consecutive points lesser than a user-given speed.

Note: The min_distance is given in metres.

Note: The min_speed is given in metres/second (m/s).

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.
- **min_distance** (*float*) – The minimum distance between 2 consecutive points.
- **min_speed** (*float*) – The minimum speed between 2 consecutive points.

Returns The filtered dataframe.

Return type *PTRAILDataFrame*

static filter_by_min_speed(*dataframe*, *min_speed: float*)

Remove the data points which have speed less than a user given speed.

Note: The min_speed is given in the units m/s (metres per second).

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.
- **min_speed** (*float*) – The speed threshold below which the points are to be removed.

Returns PTRAILDataFrame Dataframe containing the resultant dataframe.

Return type *PTRAILDataFrame*

static filter_by_traj_id(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *traj_id: str*)

Extract all the trajectory points of a particular trajectory specified by the trajectory's ID.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe on which the filtering by ID is to be done.
- **traj_id** (*Text*) – The ID of the trajectory which is to be extracted.

Returns The dataframe containing all the trajectory points of the specified trajectory.

Return type pandas.core.dataframe.DataFrame

Raises **MissingTrajIDException:** – This exception is raised when the Trajectory ID given by the user does not exist in the dataset.

static filter_outliers_by_consecutive_distance(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Check the outlier points based on distance between 2 consecutive points. Outlier formula:

Lower outlier = Q1 - (1.5*IQR)
Higher outlier = Q3 + (1.5*IQR)
IQR = Inter quartile range = Q3 - Q1

We need to find points between lower and higher outlier

Parameters **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.

Returns The dataframe which has been filtered.

Return type *PTRAILDataFrame*

static filter_outliers_by_consecutive_speed(*dataframe*)

Check the outlier points based on distance between 2 consecutive points. Outlier formula:

Lower outlier = Q1 - (1.5*IQR)
Higher outlier = Q3 + (1.5*IQR)
IQR = Inter quartile range = Q3 - Q1

We need to find points between lower and higher outlier

Parameters **dataframe** (PTRAILDataFrame) – The dataframe which is to be filtered.

Returns The dataframe which has been filtered.

Return type *PTRAILDataFrame*

static get_bounding_box_by_radius(*lat: float*, *lon: float*, *radius: float*)

Calculates bounding box from a point according to the given radius.

Parameters

- **lat** (*float*) – The latitude of centroid point of the bounding box.
- **lon** (*float*) – The longitude of centroid point of the bounding box.
- **radius** (*float*) – The max radius of the bounding box. The radius is given in metres.

Returns The bounding box of the user specified size.

Return type tuple

References

https://mathmesquita.dev/2017/01/16/filtrando-localizacao-em-um-raio.html

static hampel_outlier_detection(*dataframe*, *column_name: str*)

Use the hampel filter to remove outliers from the dataset on the basis of column specified by the user.

Warning: Do not use Hampel filter outlier detection and try to detect outliers with DateTime as it will raise a NotImplementedError as it has not been implemented yet by the original author of the Hampel filter.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe from which the outliers are to be removed.
- **column_name** (*Text*) – The column on te basis of which the outliers are to be detected.

Returns The dataframe with the outliers removed.

Return type *PTRAILDataFrame*

Raises **KeyError:** – The user-specified column is not present in the dataset.

References

Pedrido, M.O., "Hampel", (2020), GitHub repository, https://github.com/MichaelisTrofficus/hampel_filter

static remove_duplicates(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame)

Drop duplicates based on the four following columns:

1. Trajectory ID
2. DateTime
3. Latitude
4. Longitude

Duplicates will be dropped only when all the values in the above mentioned four columns are the same.

Returns The dataframe with dropped duplicates.

Return type *PTRAILDataFrame*

static remove_trajectories_with_less_points(*dataframe*, *num_min_points: Optional[int] = 3*)

Remove out the trajectories from the dataframe which have few points.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe from which trajectories with few points are to be removed.
- **num_min_points** (*Optional[int], default = 2*) – The minimum number of points that a trajectory should have if it is to be retained in the dataset.

Returns The filtered dataframe which does not contain the trajectories with few points anymore.

Return type *PTRAILDataFrame*

2.1.4.3 ptrail.preprocessing.helpers module

Warning:

1. None of the methods in this module should be used directly while performing operations on data.
2. These methods are helpers for the interpolation methods in the interpolation.py module and hence run linearly and not in parallel which will result in slower execution time.
3. All the methods in this module perform calculation on a single Trajectory ID due to which it will wrong results on data with multiple trajectories. Instead, use the interpolation.py methods for faster and reliable calculations.

The helpers class has the functionalities that interpolate a point based on the given data by the user. The class contains the following 4 interpolation calculators:

1. Linear Interpolation
2. Cubic Interpolation
3. Random-Walk Interpolation
4. Kinematic Interpolation

Besides the interpolation helpers, there are also general utilities which are used in splitting up dataframes for running the code in parallel.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.preprocessing.helpers.**Helpers**

Bases: object

static cubic_help(*df: Union[pandas.DataFrame,* ptrail.core.TrajectoryDF.PTRAILDataFrame*]*, *id_: str*, *sampling_rate: float*, *class_label_col*)

This method takes a dataframe and uses cubic interpolation to determine coordinates of location on Datetime where the time difference between 2 consecutive points exceeds the user-specified sampling_rate and inserts the interpolated point those between 2 points.

> **Warning:** This method should not be used for dataframes with multiple trajectory ids as it will yield wrong results and there might be a significant drop in performance.

Parameters

- **df** (*Union[pd.DataFrame, NumTrajDF]*) – The dataframe containing the original trajectory data.
- **id** (*Text*) – The Trajectory ID of the points in the dataframe.
- **sampling_rate** (*float*) – The maximum time difference between 2 points greater than which a point will be inserted between 2 points.

Returns The dataframe containing the trajectory enhanced with interpolated points.

Return type pandas.core.dataframe.DataFrame

static filt_df_by_date(*dataframe*, *start_date*, *end_date*)

static hampel_help(*df*, *column_name*)

This function is the helper function for the hampel_outlier_detection() function present in the filters module. The purpose of the function is to run the hampel filter on a single trajectory ID, remove the outliers and return the smaller dataframe.

> **Warning:** This function should not be used directly as it will result in a slower execution of the function and might result in removal of points that are actually not outliers.

Warning: Do not use Hampel filter outlier detection and try to detect outliers with DateTime as it will raise a NotImplementedError as it has not been implemented yet by the original author of the Hampel filter.

Parameters

- **df** (*PTRAILDataFrame/pd.core.dataframe.DataFrame*) – The dataframe which the outliers are to be removed
- **column_name** (*Text*) – The column based on which the outliers are to be removed.

Returns The dataframe where the outlier points are removed.

Return type pd.core.dataframe.DataFrame

static kinematic_help(*dataframe: Union[pandas.DataFrame, ptrail.core.TrajectoryDF.PTRAILDataFrame]*, *id_: str*, *sampling_rate: float*, *class_label_col*)

This method takes a dataframe and uses kinematic interpolation to determine coordinates of location on Datetime where the time difference between 2 consecutive points exceeds the user-specified sampling_rate and inserts the interpolated point those between 2 points.

Warning: This method should not be used for dataframes with multiple trajectory ids as it will yield wrong results and there might be a significant drop in performance.

Parameters

- **dataframe** (*Union[pd.DataFrame, NumTrajDF]*) – The dataframe containing the original trajectory data.
- **id** (*Text*) – The Trajectory ID of the points in the dataframe.
- **sampling_rate** (*float*) – The maximum time difference between 2 points greater than which a point will be inserted between 2 points.

Returns The dataframe containing the trajectory enhanced with interpolated points.

Return type pandas.core.dataframe.DataFrame

References

Nogueira, T.O., "kinematic_interpolation.py", (2016), GitHub repository, https://gist.github.com/talespaiva/128980e3608f9bc5083b.js

static linear_help(*dataframe: Union[pandas.DataFrame, ptrail.core.TrajectoryDF.PTRAILDataFrame]*, *id_: str*, *sampling_rate: float*, *class_label_col*)

This method takes a dataframe and uses linear interpolation to determine coordinates of location on Datetime where the time difference between 2 consecutive points exceeds the user-specified sampling_rate and inserts the interpolated point those between 2 points.

Warning: This method should not be used for dataframes with multiple trajectory ids as it will yield wrong results and there might be a significant drop in performance.

Parameters

- **dataframe** (*Union[pd.DataFrame, NumTrajDF]*) – The dataframe containing the original trajectory data.
- **id** (*Text*) – The Trajectory ID of the points in the dataframe.
- **sampling_rate** (*float*) – The maximum time difference between 2 points greater than which a point will be inserted between 2 points.

Returns The dataframe containing the trajectory enhanced with interpolated points.

Return type pandas.core.dataframe.DataFrame

static random_walk_help(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *id_: str*, *sampling_rate: float*, *class_label_col*)

This method takes a dataframe and uses random-walk interpolation to determine coordinates of location on Datetime where the time difference between 2 consecutive points exceeds the user-specified sampling_rate and inserts the interpolated point those between 2 points.

> **Warning:** This method should not be used for dataframes with multiple trajectory ids as it will yield wrong results and there might be a significant drop in performance.

Parameters

- **dataframe** (*Union[pd.DataFrame, NumTrajDF]*) – The dataframe containing the original trajectory data.
- **id** (*Text*) – The Trajectory ID of the points in the dataframe.
- **sampling_rate** (*float*) – The maximum time difference between 2 points greater than which a point will be inserted between 2 points.

Returns The dataframe containing the trajectory enhanced with interpolated points.

Return type pandas.core.dataframe.DataFrame

References

Etemad, M., Soares, A., Etemad, E. et al. SWS: an unsupervised trajectory segmentation algorithm based on change detection with interpolation kernels. Geoinformatica (2020)

static split_traj_helper(*df*, *num_days*)

static stats_helper(*df*, *target_col_name*, *segmented*)

Generate the stats of the kinematic features present in the Dataframe.

Parameters

- **df** (*pandas.core.dataframe.DataFrame*) – The dataframe containing the trajectory data and their features.
- **target_col_name** (*str*) – This is the 'y' value that is used for ML tasks, this is asked to append the species back at the end.
- **segmented** (*Optional[bool]*) – Indicate whether the trajectory has segments or not.

Returns A dataframe containing the stats of the given trajectory.

Return type pd.core.dataframe.DataFrame

2.1.4.4 ptrail.preprocessing.interpolation module

This class interpolates dataframe positions based on Datetime. It provides the user with the flexibility to use linear or cubic interpolation. In general, the user passes the dataframe, time jum and the interpolation type, based on the type the proper function is mapped. And if the time difference exceeds the time jump, the interpolated point is added to the position with large jump with a time increase of time jump. This interpolated row is added to the dataframe.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.preprocessing.interpolation.**Interpolation**

Bases: `object`

static interpolate_position(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *sampling_rate: float*, *ip_type: Optional[str] = 'linear'*, *class_label_col: Optional[str] = ''*)

Interpolate the position of an object and create new points using one of the interpolation methods provided by the Library. Currently, the library supports the following 4 interpolation methods:

1. Linear Interpolation
2. Cubic-Spline Interpolation
3. Kinematic Interpolation
4. Random Walk Interpolation

Warning: The Interpolation methods will only return the 4 mandatory library columns because it is not possible to interpolate other data that may or may not be present in the dataset apart from latitude, longitude and datetime. As a result, other columns are dropped.

Note: The time-jump parameter specifies where the new points are to be inserted based on the time difference between 2 consecutive points. However, it does not guarantee that the dataset will be brought down to having difference between 2 consecutive points equal to or less than the user specified time jump.

Note: The time-jump is specified in seconds. Hence, if the user-specified time-jump is not sensible, then the execution of the method will take a very long time.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing the original dataset.
- **sampling_rate** (*float*) – The maximum time difference between 2 consecutive points.
- **ip_type** (*Optional[Text], default = linear*) – The type of interpolation that is to be used.
- **class_label_col** (*Optional[Text], default = ''*) – The column header which contains the class label of the point.

Returns The dataframe containing the interpolated trajectory points.

Return type *PTRAILDataFrame*

2.1.4.5 ptrail.preprocessing.statistics module

The statistics module has several functionalities that calculate kinematic statistics of the trajectory, split trajectories, pivot dataframes etc. The main purpose of this module is to get the dataframe ready for Machine Learning tasks such as clustering, calssification etc.

Author: Yaksh J Haranwala

class ptrail.preprocessing.statistics.**Statistics**

Bases: object

static generate_kinematic_stats(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *target_col_name: str*, *segmented: Optional[bool] = False*)

Generate the statistics of kinematic features for each unique trajectory in the dataframe.

Parameters

- **dataframe** (PTRAILDataFrame) – The dataframe containing the trajectory data.
- **target_col_name** (*str*) – This is the 'y' value that is used for ML tasks, this is asked to append the target_col back at the end.
- **segmented** (*Optional[bool]*) – Indicate whether the trajectory has segments or not.

Returns A pandas dataframe containing stats for all kinematic features for each unique trajectory in the dataframe.

Return type pandas.core.dataframe.DataFrame

static pivot_stats_df(*dataframe*, *target_col_name: str*, *segmented: Optional[bool] = False*)

Given a dataframe with stats present in it, melt the dataframe to make it ready for ML tasks. This is specifically for melting the type of dataframe generated by the generate_kinematic_stats() function of the kinematic_features module.

Check the kinematic_features module for further details about the dataframe expected.

Parameters

- **dataframe** (*pd.core.dataframe.DataFrame*) – The dataframe containing stats.
- **target_col_name** (*str*) – This is the 'y' value that is used for ML tasks, this is asked to append the target_col back at the end.
- **segmented** (*Optional[bool]*) – Indicate whether the trajectory has segments or not.

Returns The dataframe above which is pivoted and has rows converted to columns.

Return type pd.core.dataframe.DataFrame

static segment_traj_by_days(*dataframe:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *num_days*)

Given a dataframe containing trajectory data, segment all the trajectories by each week.

Parameters

- **df** (PTRAILDataFrame) – The dataframe containing trajectory data.
- **num_days** (*int*) – The number of days that each segment is supposed to have.

Returns The dataframe containing segmented trajectories with a new column added called segment_id

Return type pandas.core.dataframe.DataFrame

2.1.4.6 Module contents

2.1.5 ptrail.utilities package

2.1.5.1 Submodules

2.1.5.2 ptrail.utilities.DistanceCalculator module

DistanceCalculator module contains various types of distance formulas that can be used to calculate distance between 2 points on the surface of earth depending on the CRS being used.

Authors: Salman Haidri, Yaksh J Haranwala

class ptrail.utilities.DistanceCalculator.**FormulaLog**

Bases: object

static bearing_calculation(*lat1*, *lon1*, *lat2*, *lon2*)

Calculates bearing between 2 points. Bearing can be defined as direction or an angle, between the north-south line of earth or meridian and the line connecting the target and the reference point.

Parameters

- **lat1** – The latitude value of point 1.
- **lon1** – The longitude value of point 1.
- **lat2** – The latitude value of point 2.
- **lon2** – The longitude value of point 2.

Returns Bearing between 2 points

Return type float

static haversine_distance(*lat1*, *lon1*, *lat2*, *lon2*)

The haversine formula calculates the great-circle distance between 2 points. The great-circle distance is the shortest distance over the earth's surface.

Parameters

- **lat1** (*float*) – The latitude value of point 1.
- **lon1** (*float*) – The longitude value of point 1.
- **lat2** (*float*) – The latitude value of point 2.
- **lon2** (*float*) – The longitude value of point 2.

Returns The great-circle distance between the 2 points.

Return type float

2.1.5.3 ptrail.utilities.constants module

Contains all the default constants needed for initialization. All the constant are of the type string.

2.1.5.4 ptrail.utilities.conversions module

The conversions modules contains various available methods that can be used to convert given data into another format.

Authors: Yaksh J Haranwala, Salman Haidri

class ptrail.utilities.conversions.**Conversions**

Bases: object

static convert_directions_to_degree_lat_lon(*data*, *latitude: str*, *longitude: str*)

Convert the latitude and longitude format from degrees (NSEW) to float values. This is used for datasets like the Atlantic Hurricane dataset where the coordinates are not given as float values but are instead given as degrees.

References

"Arina De Jesus Amador Monteiro Sanches. "Uma Arquitetura E Imple-menta c ao Do M odulo De Pr e-processamento Para Biblioteca Pymove".Bachelor's thesis. Universidade Federal Do Cear a, 2019"

2.1.5.5 ptrail.utilities.exceptions module

This file contains all the custom designed exception headers. There is nothing here but the exception headers and pass written inside them. The purpose of the file is to store all exceptions in one place.

Authors: Yaksh J Haranwala, Salman Haidri

exception ptrail.utilities.exceptions.**DataTypeMismatchException**

Bases: Exception

exception ptrail.utilities.exceptions.**MandatoryColumnException**

Bases: Exception

exception ptrail.utilities.exceptions.**MissingColumnsException**

Bases: Exception

exception ptrail.utilities.exceptions.**MissingTrajIDException**

Bases: Exception

exception ptrail.utilities.exceptions.**NoHeadersException**

Bases: Exception

exception ptrail.utilities.exceptions.**NotAllowedError**

Bases: Exception

2.1.5.6 Module contents

2.1.6 ptrail.visualization package

2.1.6.1 Submodules

2.1.6.2 ptrail.visualization.HydrationTrends module

This File contains the visualization that is a Radar Scatter plot showing how many number of days around each Running Water Body has an individual element spent. It is an interactive visualization as such the user can change the water body and check the distribution of animals around it.

Warning: The visualizations in this module are currently developed with a focus around the starkey.csv data as it has been developed as a side project by the developers. It will further be integrated into the library as a general class of visualizers in the time to come. Some of the visualization types may or may not work with other datasets.

Authors: Yaksh J Haranwala

class ptrail.visualization.HydrationTrends.**HydrationTrends**

Bases: `object`

static show_hydration_trends(*trajectories:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *habitat: pandas.DataFrame*, *dist_from_water: int*)

Plot the interactive plotly Radar chart that shows the number of days spent by animals around a specific water body.

Note: The water bodies in the original dataset do not have any specific names. Hence, they are just given names such as Water-body #1, Water-body #2 and so on.

Parameters

- **trajectories** (PTRAILDataFrame) – The dataframe containing the trajectory data.
- **habitat** (*pd.DataFrame*) – The dataframe containing the habitat data.
- **dist_from_water** (*int*) – The maximum distance from the water water body that the animal should be in.

Return type None

2.1.6.3 ptrail.visualization.InteractiveDonut module

This File contains the visualization that is a Donut chart depicting the breakdown of animals by each pasture. The user can change the pasture to see the breakdown of individual pastures.

Warning: The visualizations in this module are currently developed with a focus around the starkey.csv data as it has been developed as a side project by the developers. It will further be integrated into the library as a general class of visualizers in the time to come. Some of the visualization types may or may not work with other datasets.

Authors: Yaksh J Haranwala

class ptrail.visualization.InteractiveDonut.**InteractiveDonut**

Bases: object

static animals_by_pasture(*trajectories:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *habitat: pandas.DataFrame*)

Plot a donut chart that shows the proportion of animals for each pasture.

Parameters

- **trajectories** (PTRAILDataFrame) – The dataframe that contains trajectory data.
- **habitat** (*pd.DataFrame*) – The dataframe that contains habitat data.

Return type None

static plot_area_donut(*habitat: pandas.DataFrame*)

Given the trajectories and the habitat dataset, plot a donut plot which shows the area of each individual pasture as a ring and then has an interactive element that shows the distribution of animals upon clicking the pasture ring.

Parameters **habitat** (*pd.core.dataframe.DataFrame*) – The dataset containing the habitat data.

Return type None

2.1.6.4 ptrail.visualization.TrajPlotter module

This File contains TrajectoryPlotter for the Starkey dataset. An interactive experience is added to this plot in order to view the trajectory of an individual or multiple animals together.

Warning: The visualizations in this module are currently developed with a focus around the starkey.csv data as it has been developed as a side project by the developers. It will further be integrated into the library as a general class of visualizers in the time to come.

Authors: Yaksh J Haranwala

class ptrail.visualization.TrajPlotter.**TrajectoryPlotter**

Bases: object

static show_trajectories(*dataset*, *weight: float = 3*, *opacity: float = 0.8*)

Use folium to plot the trajectory on a map.

Parameters

- **dataset** –
- **weight** (*float*) – The weight of the trajectory line on the map.
- **opacity** (*float*) – The opacity of the trajectory line on the map.

Returns The map with plotted trajectory.

Return type folium.folium.Map

2.1.6.5 ptrail.visualization.statViz module

This File contains static visualizations i.e the ones that do not require the use of ipywidgets.

Warning: The visualizations in this module are currently developed with a focus around the starkey.csv data as it has been developed as a side project by the developers. It will further be integrated into the library as a general class of visualizers in the time to come. Some of the visualization types may or may not work with other datasets.

Authors: Yaksh J Haranwala

class ptrail.visualization.statViz.**StatViz**

Bases: object

static trajectory_distance_treemap(*dataset:* ptrail.core.TrajectoryDF.PTRAILDataFrame, *path: list*)

Plot a treemap of distance travelled by the moving object on a particular date.

Parameters

- **dataset** (PTRAILDataFrame) – The dataframe containing all the trajectory data.
- **map_date** (*str*) – The date for which the TreeMap is to be plotted.
- **path** (*list*) – The hierarchy of the treemap. This is passed directly into plotly's Treemap API.

Returns Treemap depicting the distance travelled.

Return type plotly.graph_objects.Figure

2.1.6.6 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

T

U

V