CerebralCortex Documentation

Release 3.3.0

Nasir Ali

Sep 04, 2020

Contents:

Cerebral Cortex is the big data cloud companion of mCerebrum designed to support population-scale data analysis, visualization, model development, and intervention design for mobile sensor data.

You can find more information about MD2K software on our software website or the MD2K organization on our MD2K website.

CerebralCortex Kernel is part of our CerebralCortex cloud platform. CerebralCortex-Kernel is capable of parallelizing tasks and scale a job to n-number of cores/machines. CerebralCortex Kernel offers some builtin features as follows:

CHAPTER 1

Installation

1.1 Dependencies

CerebralCortex Kernel requires `java 8` to run. Java 8 prior to version 8u92 support is deprecated as of CerebralCortex-Kernel 3.3.0. - check java version - `java -version` - set `JAVA_HOME` to `java 8` - OR start python shell with `JAVA_HOME=/path/to/java/Home python3`

1.2 Install using pip

CerebralCortex-Kernel requires minimum Python3.6. To install CerebralCortex-Kernel as an API:

```
pip3 install cerebralcortex-kernel
```

- Note: please use appropriate pip (e.g., pip, pip3, pip3.6 etc.) installed on your machine

1.3 Install from source code

- Clone repo - `git clone https://github.com/MD2Korg/CerebralCortex-Kernel.git`
- `cd CerebralCortex-Kernel`
- `python3 setup.py install`

CHAPTER 2

Usage

```
from cerebralcortex.kernel import Kernel
CC = Kernel(cc_configs="default")

# to view default configs
print(CC.config)

# default data storage path is
# /user/home/folder/cc_data
```

By default Kernel will load default configs. Please have a look at all available configurations for CerebralCortex-Kernel. You may also load config files as:

```
CC = Kernel(configs_dir_path="dir/path/to/configs/", new_study=True)
```

2.1 How to use builtin algorithms

Using builtin algorithms are as easy as loading data, passing it to algorithm and get the results. Below is an example on how to compute CGM Glucose Variability Metrics.

- Download Glucose Data. The device used to collect glucose was the Dexcom G6 Continuous Glucose Monitor
- Install Cerebral Cortex Kernel `pip install cerebralcortex-kernel`
- Open terminal and start python3 shell

2.2 Python Code

```
# import packages
from cerebralcortex.kernel import Kernel
from cerebralcortex.algorithms.glucose.glucose_variability_metrics import glucose_var
```

(continues on next page)

(continued from previous page)

```
# Create Kernel object
CC = Kernel(cc_configs="default", new_study=True)

# Read sample CSV data
ds = CC.read_csv("/path/of/the/downloaded/file/sample.csv", stream_name="cgm_glucose_
↪variability_metrics", header=True)

# view sample data
ds.show(2)

# Apply glucose_variability_metrics algorithm on the data
results = glucose_var(ds)

# view results
results.show(2)

# save computed data
CC.save_stream(results)
```

Please have a look at jupyter notebook for basic operation that could be perform on DataStream object.

2.3 Algorithms to Analyze Sensor Data

External CerebralCortex-Kernel offers following builtin algorithms to analyze sensor data.

- ECG sensor data quality
- ECG RR Interval Computation
- Heart Rate Variability Feature Computation
- CGM Glucose Variability Metrics
- GPS Data Clustering
- Sensor Data Interpolation
- Statistical Features Computation
- List of all available algorithms

2.4 Markers with ML Models

- Stress Detection using ECG data
- mContain Social Crowding
- Brushing Detection using Accelerometer and Gyro Data (TODO)

2.5 Visualization

- Basic Plots for Timeseries Data
- Plot GPS Clusters on Map

- Stress Visualization

2.6 Import and Document Data

- Import CSV Data in CerebralCortex-Kernel Format
- Document imported Data using MetaData Module

2.7 External CerebralCortex-Kernel Supported Platforms

- mProv
- mFlow

CHAPTER 3

Examples

- Jupyter Notebooks

CHAPTER 4

Documentation

CHAPTER 5

Deploy on Cloud

CerebralCortex-Kernel is a part of CerebralCortex cloud platform. To test the complete cloud platform, please visit CerebralCortex.

CHAPTER 6

FAQ

1 - Do I need whole CerebralCortex cloud platform to use CerebralCortex-Kernal?

No! If you want to use CerebralCortex-Kernel independently.

2 - How can I change NoSQL backend storage layer?

CerebralCortex-Kernel follows component based structure. This makes it easier to add/remove features. * Add a new class in Data manager-Raw. * New class must have read/write methods. Here is a sample skeleton class with mandatory methods required in the new class. * Create an object of new class in Data-Raw with appropriate parameters. * Add appropriate configurations in cerebralcortex.yml in (NoSQL Storage)[https://github.com/MD2Korg/CerebralCortex-Kernel/blob/master/conf/cerebralcortex.yml#L8] section.

3 - How can I replace MySQL with another SQL storage system?

- Add a new class in Data manager-SQL.
- New class must implement all of the methods available in stream_handler.py class.
- Create an object of new class in Data-SQL with appropriate parameters.
- Add appropriate configurations in cerebralcortex.yml in Relational Storage section.

4 - Where are all the backend storage related classes/methods?

In Data manager-Raw. You can add/change any backend storage.

CHAPTER 7

Contributing

Please read our Contributing Guidelines for details on the process for submitting pull requests to us.

We use the Python PEP 8 Style Guide.

Our Code of Conduct is the Contributor Covenant.

Bug reports can be submitted through JIRA.

Our discussion forum can be found here.

CHAPTER 8

Versioning

We use Semantic Versioning for versioning the software which is based on the following guidelines.

MAJOR.MINOR.PATCH (example: 3.0.12)

1. MAJOR version when incompatible API changes are made,
2. MINOR version when functionality is added in a backwards-compatible manner, and
3. PATCH version when backwards-compatible bug fixes are introduced.

For the versions available, see this repository's tags.

CHAPTER 9

Contributors

Link to the list of contributors who participated in this project.

CHAPTER 10

License

This project is licensed under the BSD 2-Clause - see the license file for details.

CHAPTER 11

Acknowledgments

- National Institutes of Health - Big Data to Knowledge Initiative
- Grants: R01MD010362, 1UG1DA04030901, 1U54EB020404, 1R01CA190329, 1R01DE02524, R00MD010468, 3UH2DA041713, 10555SC
- National Science Foundation
- Grants: 1640813, 1722646
- Intelligence Advanced Research Projects Activity
- Contract: 2017-17042800006

11.1 cerebralcortex package

11.1.1 Subpackages

cerebralcortex.algorithms package

Subpackages

cerebralcortex.algorithms.bluetooth package

Submodules

cerebralcortex.algorithms.bluetooth.encounter module

bluetooth_encounter(*data*, *st: datetime.datetime*, *et: datetime.datetime*, *distance_threshold=12*, *n_rows_threshold=8*, *time_threshold=600*, *ltime=True*)

Parameters

- **ds** – Input Datastream
- **st** – Start Time the time window in UTC
- **et** – End Time of time window in UTC
- **distance_threshold** – Threshold on mean distance per encounter
- **n_rows_threshold** – No of rows per group/encounter
- **time_threshold** – Minimum Duration of time per encounter
- **epsilon** – A simple threshold
- **count_threshold** – Threshold on count

Returns A Sparse representation of the Bluetooth Encounter

count_encounters_per_cluster(*ds*, *multiplier=10*)

get_encounter_count_all_user(*data_ds*, *user_list_ds*, *start_time*, *end_time*)

get_notification_messages(*ds*, *day*, *day_offset=5*)

Parameters

- **ds** – Input Datastream
- **day** – test date as datetime object
- **day_offset** – number of days to be considered before the test day

Returns

remove_duplicate_encounters(*ds*, *owner_name='user'*, *transmitter_name='participant_identifier'*, *start_time_name='start_time'*, *end_time_name='end_time'*, *centroid_id_name='centroid_id'*, *distance_threshold=12*)

Module contents

cerebralcortex.algorithms.ecg package

Submodules

cerebralcortex.algorithms.ecg.autosense_data_quality module

ecg_autosense_data_quality(*ecg*, *Fs=64*, *sensor_name='autosense'*, *outlier_threshold_high=4000*, *outlier_threshold_low=20*, *slope_threshold=100*, *range_threshold=50*, *eck_threshold_band_loose=400*, *window_size=3*, *acceptable_outlier_percent=34*)

Some desc..

Parameters

- **ecg** (DataStream) –
- **Fs** (*int*) –
- **sensor_name** (*str*) –
- **outlier_threshold_high** (*int*) –
- **outlier_threshold_low** (*int*) –

- **slope_threshold** (*int*) –
- **range_threshold** (*int*) –
- **eck_threshold_band_loose** (*int*) –
- **window_size** (*int*) –
- **acceptable_outlier_percent** (*int*) –

Returns DataStream - structure [timestamp, localtime, version. . . ..]

cerebralcortex.algorithms.ecg.autosense_rr_interval module

get_rr_interval(*ecg_data*, *Fs=64*)

Parameters

- **ecg_data** (DataStream) –
- **Fs** (*int*) –

Returns DataStream - timestamp, localtime, user, version . . . .

cerebralcortex.algorithms.ecg.hrv_features module

get_hrv_features(*rr_data*, *acceptable_percentage=50*, *window_length=60*)

Parameters

- **rr_data** (DataStream) –
- **acceptable_percentage** (*int*) –
- **window_length** (*int*) –

Returns:

Module contents

cerebralcortex.algorithms.ema package

Submodules

cerebralcortex.algorithms.ema.ema_random_features module

get_ema_random_features(*user_data*)

cerebralcortex.algorithms.ema.features module

ema_incentive(*ds*)

Parse stream name 'incentive–org.md2k.ema_scheduler–phone'. Convert json column to multiple columns.

Parameters **ds** – Windowed/grouped DataStream object

Returns Windowed/grouped DataStream object.

Return type ds

ema_logs(*ds*)
Convert json column to multiple columns.

Parameters **ds** (DataStream) – Windowed/grouped DataStream object

Returns:

Module contents

cerebralcortex.algorithms.glucose package

Submodules

cerebralcortex.algorithms.glucose.glucose_variability_metrics module

glucose_var(*ds*)
Compute CGM Glucose Variability Metrics:

This algorithm computes 23 clinically validated glucose variability metrics from continuous glucose monitor data.

Input: ds (DataStream): Windowed/grouped DataStream of CGM data

Returns DataStream with glucose variability metrics Glucose Variability Metrics include: Interday Mean Glucose Interday Median Glucose Interday Maximum Glucose Interday Minimum Glucose Interday Standard Deviation of Glucose Interday Coefficient of Variation of Glucose Intraday Standard Deviation of Glucose (mean, median, standard deviation) Intraday Coefficient of Variation of Glucose (mean, median, standard deviation) TIR (Time in Range of default 1 SD) TOR (Time outside Range of default 1 SD) POR (Percent outside Range of default 1 SD) MAGE (Mean Amplitude of Glucose Excursions, default 1 SD) MAGN (Mean Amplitude of Normal Glucose, default 1 SD) J-index LBGI (Low Blood Glucose Index) HBGI (High Blood Glucose Index) MODD (Mean of Daily Differences) CONGA24 (Continuous overall net glycemic action over 24 hours) ADRR (Average Daily Risk Range) GMI (Glucose Management Indicator) eA1c (estimated A1c according to American Diabetes Association) Q1G (intraday first quartile glucose) Q3G (intraday third quartile glucose) ** for more information on these glucose metrics see dbdp.org**

Module contents

cerebralcortex.algorithms.gps package

Submodules

cerebralcortex.algorithms.gps.clustering module

cluster_gps(*ds: cerebralcortex.core.datatypes.datastream.DataStream*, *epsilon_constant: int = 1000*, *km_per_radian: int = 6371.0088*, *geo_fence_distance: int = 30*, *minimum_points_in_cluster: int = 1*, *latitude_column_name: str = 'latitude'*, *longitude_column_name: str = 'longitude'*)
Cluster GPS data - Algorithm used to cluster GPS data is based on DBScan

Parameters

- **ds** (DataStream) – Windowed/grouped DataStream object
- **epsilon_constant** (*int*) –
- **km_per_radian** (*int*) –
- **geo_fence_distance** (*int*) –
- **minimum_points_in_cluster** (*int*) –
- **latitude_column_name** (*str*) –
- **longitude_column_name** (*str*) –

Returns DataStream object

impute_gps_data(*ds*, *accuracy_threashold: int = 100*)
Inpute GPS data

Parameters

- **ds** (DataStream) – Windowed/grouped DataStream object
- **accuracy_threashold** (*int*) –

Returns DataStream object

Module contents

cerebralcortex.algorithms.raw_byte_decode package

Submodules

cerebralcortex.algorithms.raw_byte_decode.motionsenseHRV module

Preprc(*raw_data: object*, *flag: object = 0*) → object
Function to compute the decoded values in motionsense HRV sensors and interploate the timestamps given the decoded sequence numbers :param raw_data: :param flag: :return:

convert_to_array(*vals*)

get_metadata()

motionsenseHRV_decode(*raw_data_with_diff*)

process_raw_PPG(*raw_data*)

Module contents

motionsenseHRV_decode(*raw_data_with_diff*)

cerebralcortex.algorithms.rr_intervals package

Submodules

cerebralcortex.algorithms.rr_intervals.rr_interval_feature_extraction module

combine_data(*window_col*)

compute_rr_interval_features()

get_windows(*data*)

heart_rate_power(*power: numpy.ndarray*, *frequency: numpy.ndarray*, *low_rate: float*, *high_rate: float*)
Compute Heart Rate Power for specific frequency range :param power: np.ndarray :param frequency: np.ndarray :param high_rate: float :param low_rate: float :return: sum of power for the frequency range

lomb(*time_stamps: List*, *samples: List*, *low_frequency: float*, *high_frequency: float*)

: Lomb–Scargle periodogram implementation

param data List[DataPoint]

param high_frequency float

param low_frequency float

:return lomb-scargle pgram and frequency values

rr_feature_computation(*timestamp: list*, *value: list*, *low_frequency: float = 0.01*, *high_frequency: float = 0.7*, *low_rate_vlf: float = 0.0009*, *high_rate_vlf: float = 0.04*, *low_rate_hf: float = 0.15*, *high_rate_hf: float = 0.4*, *low_rate_lf: float = 0.04*, *high_rate_lf: float = 0.15*)
ECG Feature Implementation. The frequency ranges for High, Low and Very low heart rate variability values are derived from the following paper: 'Heart rate variability: standards of measurement, physiological interpretation and clinical use' :param high_rate_lf: float :param low_rate_lf: float :param high_rate_hf: float :param low_rate_hf: float :param high_rate_vlf: float :param low_rate_vlf: float :param high_frequency: float :param low_frequency: float :param datastream: DataStream :param window_size: float :param window_offset: float :return: ECG Feature DataStreams

rr_interval_feature_extraction(*data: object*) → object

Module contents

cerebralcortex.algorithms.signal_processing package

Submodules

cerebralcortex.algorithms.signal_processing.features module

complementary_filter(*ds*, *freq: int = 16*, *accelerometer_x: str = 'accelerometer_x'*, *accelerometer_y: str = 'accelerometer_y'*, *accelerometer_z: str = 'accelerometer_z'*, *gyroscope_x: str = 'gyroscope_x'*, *gyroscope_y: str = 'gyroscope_y'*, *gyroscope_z: str = 'gyroscope_z'*)
Compute complementary filter on gyro and accel data.

Parameters

- **ds** (DataStream) – Non-Windowed/grouped dataframe
- **freq** (*int*) – frequency of accel/gryo. Assumption is that frequency is equal for both gyro and accel.
- **accelerometer_x** (*str*) – name of the column

- **accelerometer_y** (*str*) – name of the column
- **accelerometer_z** (*str*) – name of the column
- **gyroscope_x** (*str*) – name of the column
- **gyroscope_y** (*str*) – name of the column
- **gyroscope_z** (*str*) – name of the column

compute_FFT_features(*ds, exclude_col_names: list = [], feature_names=['fft_centroid', 'fft_spread', 'spectral_entropy', 'fft_flux', 'spectral_falloff']*)

Transforms data from time domain to frequency domain.

Parameters

- **list** (*feature_names*) – name of the columns on which features should not be computed
- **list** – names of the features. Supported features are fft_centroid, fft_spread, spectral_entropy, spectral_entropy_old, fft_flux, spectral_falloff
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

Returns DataStream object with all the existing data columns and FFT features

compute_zero_cross_rate(*ds, exclude_col_names: list = [], feature_names=['zero_cross_rate']*)

Compute statistical features.

Parameters

- **ds** (DataStream) – Windowed/grouped dataframe
- **list** (*feature_names*) – name of the columns on which features should not be computed
- **list** – names of the features. Supported features are ['mean', 'median', 'stddev', 'variance', 'max', 'min', 'skew', 'kurt', 'sqr', 'zero_cross_rate'
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

Returns DataStream object

Module contents

cerebralcortex.algorithms.stats package

Submodules

cerebralcortex.algorithms.stats.features module

interpolate(*ds*, *freq=16*, *method='linear'*, *axis=0*, *limit=None*, *inplace=False*, *limit_direction='forward'*, *limit_area=None*, *downcast=None*)

Interpolate values according to different methods. This method internally uses pandas interpolation.

Parameters

- **ds** (DataStream) – Windowed/grouped DataStream object
- **freq** (*int*) – Frequency of the signal
- **method** (*str*) – default 'linear' - 'linear': Ignore the index and treat the values as equally spaced. This is the only method supported on MultiIndexes. - 'time': Works on daily and higher resolution data to interpolate given length of interval. - 'index', 'values': use the actual numerical values of the index. - 'pad': Fill in NaNs using existing values. - 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'spline', 'barycentric', 'polynomial': Passed to scipy.interpolate.interp1d. These methods use the numerical values of the index. Both 'polynomial' and 'spline' require that you also specify an order (int), e.g. df.interpolate(method='polynomial', order=5). - 'krogh', 'piecewise_polynomial', 'spline', 'pchip', 'akima': Wrappers around the SciPy interpolation methods of similar names. See Notes. - 'from_derivatives': Refers to scipy.interpolate.BPoly.from_derivatives which replaces 'piecewise_polynomial' interpolation method in scipy 0.18.
- **{0 or 'index', 1 or 'columns', None}** (*axis*) – default None. Axis to interpolate along.
- **limit** (*int*) – optional. Maximum number of consecutive NaNs to fill. Must be greater than 0.
- **inplace** (*bool*) – default False. Update the data in place if possible.
- **{'forward', 'backward', 'both'}** (*limit_direction*) – default 'forward'. If limit is specified, consecutive NaNs will be filled in this direction.
- **{None, 'inside', 'outside'}** (*limit_area*) – default None. If limit is specified, consecutive NaNs will be filled with this restriction. - None: No fill restriction. - 'inside': Only fill NaNs surrounded by valid values (interpolate). - 'outside': Only fill NaNs outside valid values (extrapolate).
- **optional, 'infer' or None** (*downcast*) – defaults to None
- ****kwargs** – Keyword arguments to pass on to the interpolating function.

Returns DataStream: interpolated data

magnitude(*ds*, *col_names=[]*)

Compute magnitude of columns

Parameters

- **ds** (DataStream) – Windowed/grouped DataStream object
- **col_names** (*list[str]*) – column names

Returns DataStream

statistical_features(*ds, exclude_col_names: list = [], feature_names=['mean', 'median', 'stddev', 'variance', 'max', 'min', 'skew', 'kurt', 'sqr']*)

Compute statistical features.

Parameters

- **ds** (DataStream) – Windowed/grouped DataStream object
- **list** (*feature_names*) – name of the columns on which features should not be computed
- **list** – names of the features. Supported features are ['mean', 'median', 'stddev', 'variance', 'max', 'min', 'skew', 'kurt', 'sqr', 'zero_cross_rate'

Returns DataStream object with all the existing data columns and FFT features

Module contents

cerebralcortex.algorithms.stress_prediction package

Submodules

cerebralcortex.algorithms.stress_prediction.ecg_stress module

compute_stress_probability(*stress_features_normalized*, *model_path='.'*, *feature_index=None*)

Parameters

- **stress_features_normalized** –
- **model_path** –
- **feature_index** –

Returns:

cerebralcortex.algorithms.stress_prediction.stress_episodes module

compute_stress_episodes(*ecg_stress_probability*, *macd_param_fast=7*, *macd_param_slow=19*, *macd_param_signal=2*, *threshold_stressed=0.36*, *threshold_not_stressed=0.36*)

Compute stress episodes using MACD

Parameters

- **ecg_stress_probability** (DataStream) –
- **macd_param_fast** (*int*) –
- **macd_param_slow** (*int*) –
- **macd_param_signal** (*into*) –
- **threshold_stressed** (*float*) –
- **threshold_not_stressed** (*float*) –

Returns with a column stress_episodes

Return type *DataStream*

cerebralcortex.algorithms.stress_prediction.stress_imputation module

forward_fill_data(*stress_data*, *output_stream_name='org.md2k.autosense.ecg.stress.probability.forward.filled'*, *minimum_points_per_day=60*)

Parameters

- **stress_data** (DataStream) –
- **output_stream_name** (*str*) –
- **minimum_points_per_day** (*int*) –

Returns:

get_metadata(*stress_imputed_data*, *output_stream_name*, *input_stream_name*)
generate metadata for a datastream.

Parameters

- **stress_imputed_data** (DataStream) –
- **output_stream_name** (*str*) –

Returns:

impute_stress_likelihood(*stress_data*, *output_stream_name='org.md2k.autosense.ecg.stress.probability.imputed'*)

Parameters

- **stress_data** (DataStream) –
- **output_stream_name** (*str*) –

Returns:

cerebralcortex.algorithms.stress_prediction.stress_prediction module

stress_prediction(*data: object*) → object

Module contents

cerebralcortex.algorithms.utils package

Submodules

cerebralcortex.algorithms.utils.feature_normalization module

normalize_features(*data*, *index_of_first_order_feature=2*, *lower_percentile=20*, *higher_percentile=99*, *minimum_minutes_in_day=60*, *no_features=11*, *epsilon=1e-08*, *input_feature_array_name='features'*)

Parameters

- **data** –
- **index_of_first_order_feature** –

- **lower_percentile** –
- **higher_percentile** –
- **minimum_minutes_in_day** –
- **no_features** –
- **epsilon** –
- **input_feature_array_name** –

Returns:

cerebralcortex.algorithms.utils.mprov_helper module

CC_MProvAgg(*in_stream_name, op, out_stream_name, in_stream_key=['index'], out_stream_key=['index'], map=None, graph_name=None*)

CC_get_prov_connection(*graph_name=None*)

MProvAgg_empty()
This is an empty decorator. This will be applied if mprov server setting is OFF

write_metadata_to_mprov(*metadata*)

cerebralcortex.algorithms.utils.util module

update_metadata(*stream_metadata*, *stream_name*, *stream_desc*, *module_name*, *module_version*, *authors: list*, *input_stream_names: list = []*, *annotations: list = []*) → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
Create Metadata object with some sample metadata of phone battery data :param stream_metadata: :param stream_name: :param stream_desc: :param module_name: :param module_version: :param authors: List of authors names and emails ids in dict. For example, authors = [{"ali":"ali@gmail.com"}, {"nasir":"nasir@gmail.com"}] :type authors: list[dict] :param input_stream_names: :param annotations:

Returns metadata of phone battery stream

Return type *Metadata*

Module contents

cerebralcortex.algorithms.visualization package

Submodules

cerebralcortex.algorithms.visualization.visualization module

Module contents

Module contents

cerebralcortex.core package

Subpackages

cerebralcortex.core.config_manager package

Submodules

cerebralcortex.core.config_manager.config module

class Configuration(*config_dir: str, cc_configs: dict = '', config_file_name: str = 'cerebralcortex.yml'*)
Bases: *cerebralcortex.core.config_manager.config_handler.ConfigHandler*

cerebralcortex.core.config_manager.config_handler module

class ConfigHandler
Bases: object

load_file(*filepath: str, default_configs=False*)
Helper method to load a yaml file

Parameters **filepath** (*str*) – path to a yml configuration file

Module contents

cerebralcortex.core.data_manager package

Subpackages

cerebralcortex.core.data_manager.raw package

Submodules

cerebralcortex.core.data_manager.raw.data module

class RawData(*CC*)
Bases: *cerebralcortex.core.data_manager.raw.stream_handler.StreamHandler*, *cerebralcortex.core.data_manager.raw.filebased_storage.FileBasedStorage*

cerebralcortex.core.data_manager.raw.filebased_storage module

class FileBasedStorage
Bases: object

get_stream_versions(*stream_name: str*) → list
Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises `ValueError` – if stream_name is empty or None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

is_stream(*stream_name: str*) → bool
Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

is_study() → bool
Returns true if study_name exists.

Returns True if study_name exist False otherwise

Return type bool

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.is_study()
>>> True
```

list_streams() → List[str]
Get all the available stream names

Returns list of available streams names

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.list_streams()
```

list_users(*stream_name: str*, *version: int = 1*) → List[str]
Get all the available stream names with metadata

stream_name (str): name of a stream version (int): version of a stream

Returns list of available user-ids for a giving stream version

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.list_users()
```

read_file(*stream_name: str*, *version: str = 'latest'*, *user_id: str = None*) → object
Get stream data from storage system. Data would be return as pyspark DataFrame object :param stream_name: name of a stream :type stream_name: str :param version: version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all") :type version: str :param user_id: id of a user :type user_id: str

---

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Returns pyspark DataFrame object

Return type object

Raises `Exception` – if stream name does not exist.

search_stream(*stream_name*) → List[str]
Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

write_file(*stream_name: str*, *data: <property object at 0x7f8c8f4ec778>*, *file_mode: str*) → bool
Write pyspark DataFrame to a file storage system

Parameters

- **stream_name** (*str*) – name of the stream
- **data** (*object*) – pyspark DataFrame object
- **file_mode** (*str*) – write mode, append is currently supportes

Returns True if data is stored successfully or throws an Exception.

Return type bool

Raises `Exception` – if DataFrame write operation fails

write_pandas_to_parquet_file(*df: <module 'pandas' from '/home/docs/checkouts/readthedocs.org/user_builds/cerebralcortex-kernel/envs/3.3/lib/python3.6/site-packages/pandas/__init__.py'>*, *user_id: str*, *stream_name: str*, *stream_version: str*) → str

Convert pandas dataframe into pyarrow parquet format and store

Parameters

- **df** (*pandas*) – pandas dataframe
- **user_id** (*str*) – user id
- **stream_name** (*str*) – name of a stream

Returns file_name of newly create parquet file

Return type str

cerebralcortex.core.data_manager.raw.sample_code_for_soujanya module

cerebralcortex.core.data_manager.raw.storage_blueprint module

class BlueprintStorage(*obj*)

Bases: object

This is a sample reference class. If you want to add another storage layer then the class must have following methods in it. read_file() write_file()

get_stream_metadata_hash(*stream_name: str*) → list

Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes

Return type list[str]

Examples

```
>>> CC.get_stream_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_
↪SENSE_HRV--RIGHT_WRIST")
>>> ["00ab666c-afb8-476e-9872-6472b4e66b68", "15cc444c-dfb8-676e-3872-
↪8472b4e66b12"]
```

get_stream_name(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions (*stream_name: str*) → list
Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises ValueError – if stream_name is empty or None

Examples

```
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

is_stream (*stream_name: str*) → bool
Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

list_streams () → List[str]
Get all the available stream names with metadata

Returns list of available streams metadata

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.list_streams()
```

read_file (*stream_name: str*, *version: str = 'all'*) → object
Get stream data from storage system. Data would be return as pyspark DataFrame object :param stream_name: name of a stream :type stream_name: str :param version: version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all") :type version: str

Returns pyspark DataFrame object

Return type object

Raises `Exception` – if stream name does not exist.

search_stream(*stream_name*)
Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

write_file(*stream_name: str*, *data: cerebralcortex.core.datatypes.datastream.DataStream*) → bool
Write pyspark DataFrame to a data storage system :param stream_name: name of the stream :type stream_name: str :param data: pyspark DataFrame object :type data: object

Returns True if data is stored successfully or throws an Exception.

Return type bool

Raises `Exception` – if DataFrame write operation fails

cerebralcortex.core.data_manager.raw.stream_handler module

class DataSet
Bases: `enum.Enum`

An enumeration.

COMPLETE = (1,)

ONLY_DATA = (2,)

ONLY_METADATA = 3

class StreamHandler
Bases: `object`

get_stream(*stream_name: str*, *version: str = 'latest'*, *user_id: str = None*, *data_type=<DataSet.COMPLETE: (1,)>*) → cerebralcortex.core.datatypes.datastream.DataStream
Retrieve a data-stream with it's metadata.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are latest, or a specific version of a stream (e.g., 2)
- **user_id** (*str*) – id of a user
- **data_type** (DataSet) – DataSet.COMPLETE returns both Data and Metadata. DataSet.ONLY_DATA returns only Data. DataSet.ONLY_METADATA returns only metadata of a stream. (Default=DataSet.COMPLETE)

Returns contains Data and/or metadata

Return type *DataStream*

Raises `ValueError` – if stream name is empty or None

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = CC.get_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV-
↪-RIGHT_WRIST")
>>> ds.data # an object of a dataframe
>>> ds.metadata # an object of MetaData class
>>> ds.get_metadata(version=1) # get the specific version metadata of a stream
```

save_stream(*datastream*, *overwrite=False*) → bool

Saves datastream raw data in selected NoSQL storage and metadata in MySQL.

Parameters

- **datastream** (`DataStream`) – a DataStream object
- **overwrite** (*bool*) – if set to true, whole existing datastream data will be overwritten by new data

Returns True if stream is successfully stored or throws an exception

Return type bool

Todo: Add functionality to store data in influxdb.

Raises `Exception` – log or throws exception if stream is not stored

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_stream(ds)
```

cerebralcortex.core.data_manager.raw.tedt module

cerebralcortex.core.data_manager.raw.util module

class filesystem_helper

Bases: `object`

create_dir(*dirpath: str*, *stream_name: str = None*, *version: int = None*, *user_id: str = None*)

Creates a directory if it does not exist.

Parameters

- **dirpath** (*str*) – base storage dir path
- **stream_name** (*str*) – name of a stream
- **version** (*int*) – version number of stream data
- **user_id** (*str*) – uuid of a user

ls_dir (*dirpath: str*, *stream_name: str = None*, *version: int = None*, *user_id: str = None*)
List the contents of a directory

Parameters

- **dirpath** (*str*) – base storage dir path
- **stream_name** (*str*) – name of a stream
- **version** (*int*) – version number of stream data
- **user_id** (*str*) – uuid of a user

Returns list of file and/or dir names

Return type list[str]

path_exist (*dirpath: str*, *stream_name: str = None*, *version: int = None*, *user_id: str = None*)
Checks if a path exist

Parameters

- **dirpath** (*str*) – base storage dir path
- **stream_name** (*str*) – name of a stream
- **version** (*int*) – version number of stream data
- **user_id** (*str*) – uuid of a user

Returns true if path exist, false otherwise

Return type bool

class hdfs_helper
Bases: `object`

create_dir (*dirpath: str*, *stream_name: str = None*, *version: int = None*, *user_id: str = None*)
Creates a directory if it does not exist.

Parameters

- **dirpath** (*str*) – base storage dir path
- **stream_name** (*str*) – name of a stream
- **version** (*int*) – version number of stream data
- **user_id** (*str*) – uuid of a user

hdfs_conn = ''

ls_dir (*dirpath: str*, *stream_name: str = None*, *version: int = None*, *user_id: str = None*)
List the contents of a directory

Parameters

- **dirpath** (*str*) – base storage dir path
- **stream_name** (*str*) – name of a stream

- **version** (*int*) – version number of stream data
- **user_id** (*str*) – uuid of a user

Returns list of file and/or dir names

Return type list[str]

path_exist(*dirpath: str*, *stream_name: str = None*, *version: int = None*, *user_id: str = None*)
Checks if a path exist

Parameters

- **dirpath** (*str*) – base storage dir path
- **stream_name** (*str*) – name of a stream
- **version** (*int*) – version number of stream data
- **user_id** (*str*) – uuid of a user

Returns true if path exist, false otherwise

Return type bool

class tmp
Bases: object

get_storage_path(*dirpath*, *stream_name*, *version*, *user_id*)

Module contents

cerebralcortex.core.data_manager.sql package

Submodules

cerebralcortex.core.data_manager.sql.data module

class SqlData(*CC*)
Bases: *cerebralcortex.core.data_manager.sql.stream_handler.StreamHandler*, *cerebralcortex.core.data_manager.sql.users_handler.UserHandler*

cerebralcortex.core.data_manager.sql.orm_models module

class Stream(*name*, *version*, *study_name*, *metadata_hash*, *stream_metadata*)
Bases: sqlalchemy.ext.declarative.api.Base

creation_date

metadata_hash

name

row_id

stream_metadata

study_name

version

class User(*user_id*, *username*, *password*, *study_name*, *token*, *token_issued*, *token_expiry*, *user_role='participant'*, *user_metadata={}*, *user_settings={}*, *active=1*)

Bases: `sqlalchemy.ext.declarative.api.Base`

active

creation_date

has_data

password

row_id

study_name

token

token_expiry

token_issued

user_id

user_metadata

user_role

user_settings

username

cerebralcortex.core.data_manager.sql.stream_handler module

class StreamHandler

Bases: `object`

get_stream_metadata_by_hash(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → List

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns [stream_name, metadata]

Return type List

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ["name" .....] # stream metadata and other information
```

get_stream_metadata_by_name(*stream_name: str*, *version: int*) → cerebralcortex.core.metadata_manager.stream.metadata.Metadata

Get a list of metadata for all versions available for a stream.

Parameters

- **stream_name** (*str*) – name of a stream

- **version** (*int*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")

Returns Returns an empty list if no metadata is available for a stream_name or a list of metadata otherwise.

Return type *Metadata*

Raises ValueError – stream_name cannot be None or empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.list_users("mperf")
>>> [Metadata] # list of MetaData class objects
```

get_stream_metadata_hash (*stream_name: str*) → List

Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes with name and versions

Return type list

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [["stream_name", "version", "metadata_hash"]]
```

get_stream_name (*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions (*stream_name: str*) → list

Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises `ValueError` – if stream_name is empty or None

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

is_stream(*stream_name: str*) → bool
Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

list_streams() → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]
Get all the available stream names with metadata

Returns list of available streams metadata [{name:"", metadata:""}...]

Return type List[*Metadata*]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.list_streams()
```

save_stream_metadata(*metadata_obj*) → dict
Update a record if stream already exists or insert a new record otherwise.

Parameters **metadata_obj** (`Metadata`) – stream metadata

Returns {"status": True/False,"verion":version}

Return type dict

Raises `Exception` – if fail to insert/update record in MySQL. Exceptions are logged in a log file

search_stream(*stream_name*)
Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

cerebralcortex.core.data_manager.sql.users_handler module

class UserHandler

Bases: object

create_user(*username: str*, *user_password: str*, *user_role: str*, *user_metadata: dict*, *user_settings: dict*, *encrypt_password: bool = False*) → bool

Create a user in SQL storage if it doesn't exist :param username: Only alphanumeric usernames are allowed with the max length of 25 chars. :type username: str :param user_password: no size limit on password :type user_password: str :param user_role: role of a user :type user_role: str :param user_metadata: metadata of a user :type user_metadata: dict :param user_settings: user settings, mCerebrum configurations of a user :type user_settings: dict :param encrypt_password: encrypt password if set to True :type encrypt_password: bool

Returns True if user is successfully registered or throws any error in case of failure

Return type bool

Raises

- ValueError – if selected username is not available
- Exception – if sql query fails

encrypt_user_password(*user_password: str*) → str

Encrypt password

Parameters **user_password** (*str*) – unencrypted password

Raises ValueError – password cannot be None or empty.

Returns encrypted password

Return type str

gen_random_pass(*string_type: str*, *size: int = 8*) → str

Generate a random password

Parameters

- **string_type** – Accepted parameters are "varchar" and "char". (Default="varchar")
- **size** – password length (default=8)

Returns random password

Return type str

get_user_id(*user_name: str*) → str

Get the user id linked to user_name.

Parameters **user_name** (*str*) – username of a user

Returns user id associated to user_name

Return type str

Raises `ValueError` – User name is a required field.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_id("nasir_ali")
>>> '76cc444c-4fb8-776e-2872-9472b4e66b16'
```

get_user_metadata(*user_id: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'> = None*, *username: str = None*) → dict

Get user metadata by user_id or by username

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns user metadata

Return type dict

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_metadata(username="nasir_ali")
>>> {"study_name":"mperf".....}
```

get_user_settings(*username: str = None*, *auth_token: str = None*) → dict

Get user settings by auth-token or by username. These are user's mCerebrum settings

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – auth-token

Returns List of dictionaries of user metadata

Return type list[dict]

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_settings(username="nasir_ali")
>>> [{"mcerebrum":"some-conf".....}]
```

get_username(*user_id: str*) → str
Get the user name linked to a user id.

Parameters **user_name** (*str*) – username of a user

Returns user_id associated to username

Return type bool

Raises ValueError – User ID is a required field.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_username("76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> 'nasir_ali'
```

is_auth_token_valid(*username: str*, *auth_token: str*, *checktime: bool = False*) → bool
Validate whether a token is valid or expired based on the token expiry datetime stored in SQL

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – token generated by API-Server
- **checktime** (*bool*) – setting this to False will only check if the token is available in system. Setting this to true will check if the token is expired based on the token expiry date.

Raises ValueError – Auth token and auth-token expiry time cannot be null/empty.

Returns returns True if token is valid or False otherwise.

Return type bool

is_user(*user_id: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'> = None*, *user_name: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'> = None*) → bool
Checks whether a user exists in the system. One of both parameters could be set to verify whether user exist.

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns True if a user exists in the system or False otherwise.

Return type bool

Raises ValueError – Both user_id and user_name cannot be None or empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.is_user(user_id="76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> True
```

list_users() → List[list]
Get a list of all users part of a study.

Parameters **study_name** (*str*) – name of a study. If no study_name is provided then all users' list will be returned

Raises ValueError – Study name is a requied field.

Returns Returns empty list if there is no user associated to the study_name and/or study_name does not exist.

Return type list[list]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.list_users("mperf")
>>> [{"76cc444c-4fb8-776e-2872-9472b4e66b16": "nasir_ali"}] # [{user_id, user_
↪name}]
```

login_user(*username: str*, *password: str*, *encrypt_password: bool = False*) → dict

Authenticate a user based on username and password and return an auth token

Parameters

- **username** (*str*) – username of a user
- **password** (*str*) – password of a user
- **encrypt_password** (*str*) – is password encrypted or not. mCerebrum sends encrypted passwords

Raises ValueError – User name and password cannot be empty/None.

Returns return eturn {"status":bool, "auth_token": str, "msg": str}

Return type dict

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.connect("nasir_ali",
↪"2ksdfhoi2r2ljndf823hlkf8234hohwef0234hlkjwer98u234", True)
>>> True
```

update_auth_token(*username: str*, *auth_token: str*, *auth_token_issued_time: datetime.datetime*, *auth_token_expiry_time: datetime.datetime*) → bool

Update an auth token in SQL database to keep user stay logged in. Auth token valid duration can be changed in configuration files.

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – issued new auth token
- **auth_token_issued_time** (*datetime*) – datetime when the old auth token was issue
- **auth_token_expiry_time** (*datetime*) – datetime when the token will get expired

Raises ValueError – Auth token and auth-token issue/expiry time cannot be None/empty.

Returns Returns True if the new auth token is set or False otherwise.

Return type bool

username_checks(*username: str*)
No space, special characters, dash etc. are allowed in username. Only alphanumeric usernames are allowed with the max length of 25 chars.

Parameters **username** (*str*) –

Returns True if provided username comply the standard or throw an exception

Return type bool

Raises Exception – if username doesn't follow standards

Module contents

cerebralcortex.core.data_manager.time_series package

Submodules

cerebralcortex.core.data_manager.time_series.data module

class TimeSeriesData(*CC*)
Bases: *cerebralcortex.core.data_manager.time_series.influxdb_handler.InfluxdbHandler*

cerebralcortex.core.data_manager.time_series.influxdb_handler module

class InfluxdbHandler
Bases: object

save_data_to_influxdb(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*)
Save data stream to influxdb only for visualization purposes.

Parameters **datastream** (DataStream) – a DataStream object

Returns True if data is ingested successfully or False otherwise

Return type bool

Todo: This needs to be updated with the new structure. Should metadata be stored or not?

Example

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_data_to_influxdb(ds)
```

write_pd_to_influxdb(*user_id: str, username: str, stream_name: str, df: pandas.core.frame.DataFrame*)
Store data in influxdb. Influxdb is used for visualization purposes

Parameters

- **user_id** (*str*) – id of a user
- **username** (*str*) – username
- **stream_name** (*str*) – name of a stream
- **df** (*pandas*) – pandas dataframe

Raises `Exception` – if error occurs during storing data to influxdb

Module contents

Module contents

cerebralcortex.core.datatypes package

Submodules

cerebralcortex.core.datatypes.datastream module

class DataStream(*data: object = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*)

Bases: `pyspark.sql.dataframe.DataFrame`

agg(**exprs*)

Aggregate on the entire DataStream without groups

Parameters ***exprs** –

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.agg({"age": "max"}).collect()
>>> # Below example shows how to use pyspark functions in add method
>>> from pyspark.sql import functions as F
>>> ds.agg(F.min(ds.age)).collect()
```

alias(*alias*)

Returns a new DataStream with an alias set.

Parameters **alias** – string, an alias name to be set for the datastream.

Returns DataStream object

Return type object

Examples

```
>>> df_as1 = df.alias("df_as1")
>>> df_as2 = df.alias("df_as2")
```

approxQuantile(*col*, *probabilities*, *relativeError*)
Calculates the approximate quantiles of numerical columns of a DataStream.

The result of this algorithm has the following deterministic bound: If the DataStream has N elements and if we request the quantile at probability p up to error err, then the algorithm will return a sample x from the DataStream so that the exact rank of x is close to (p * N). More precisely,

floor((p - err) * N) <= rank(x) <= ceil((p + err) * N).

This method implements a variation of the Greenwald-Khanna algorithm (with some speed optimizations). The algorithm was first present in [[http://dx.doi.org/10.1145/375663.375670 Space-efficient Online Computation of Quantile Summaries]] by Greenwald and Khanna.

Note that null values will be ignored in numerical columns before calculation. For columns only containing null values, an empty list is returned.

Parameters

- **col** (*str[list]*) – Can be a single column name, or a list of names for multiple columns.
- **probabilities** – a list of quantile probabilities Each number must belong to [0, 1]. For example 0 is the minimum, 0.5 is the median, 1 is the maximum.
- **relativeError** – The relative target precision to achieve (>= 0). If set to zero, the exact quantiles are computed, which could be very expensive. Note that values greater than 1 are accepted but give the same result as 1.

Returns the approximate quantiles at the given probabilities. If the input col is a string, the output is a list of floats. If the input col is a list or tuple of strings, the output is also a list, but each element in it is a list of floats, i.e., the output is a list of list of floats.

colRegex(*colName*)
Selects column based on the column name specified as a regex and returns it as Column.

Parameters **colName** (*str*) – column name specified as a regex.

Returns

Return type *DataStream*

Examples

```
>>> ds.colRegex("colName")
```

collect()
Collect all the data to master node and return list of rows

Returns rows of all the dataframe

Return type List

Examples

```
>>> ds.collect()
```

compute(*udfName*, *windowDuration: int = None*, *slideDuration: int = None*, *groupByColumnName: List[str] = []*, *startTime=None*)
Run an algorithm. This method supports running an udf method on windowed data

Parameters

- **udfName** – Name of the algorithm
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

corr(*col1*, *col2*, *method=None*)

Calculates the correlation of two columns of a DataStream as a double value. Currently only supports the Pearson Correlation Coefficient.

Parameters

- **col1** (*str*) – The name of the first column
- **col2** (*str*) – The name of the second column
- **method** (*str*) – The correlation method. Currently only supports "pearson"

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.corr("cal1", "col2", "pearson").collect()
```

count()

Returns the number of rows in this DataStream.

Examples

```
>>> ds.count()
```

cov(*col1*, *col2*)

Calculate the sample covariance for the given columns, specified by their names, as a double value.

Parameters

- **col1** (*str*) – The name of the first column
- **col2** (*str*) – The name of the second column

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.cov("cal1", "col2", "pearson").collect()
```

create_windows(*window_length='hour'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

crossJoin(*other*)
Returns the cartesian product with another DataStream

Parameters **other** – Right side of the cartesian product.

Returns DataStream object with joined streams

Examples

```
>>> ds.crossJoin(ds2.select("col_name")).collect()
```

crosstab(*col1*, *col2*)
Computes a pair-wise frequency table of the given columns. Also known as a contingency table. The number of distinct values for each column should be less than 1e4. At most 1e6 non-zero pair frequencies will be returned. The first column of each row will be the distinct values of col1 and the column names will be the distinct values of col2. The name of the first column will be $col1_$col2. Pairs that have no occurrences will have zero as their counts.

Parameters

- **col1** (*str*) – The name of the first column. Distinct items will make the first item of each row.
- **col2** (*str*) – The name of the second column. Distinct items will make the column names of the DataStream.

Returns DataStream object

Examples

```
>>> ds.crosstab("col_1", "col_2")
```

data
get stream data

Returns (DataFrame):

describe(**cols*)
Computes basic statistics for numeric and string columns. This include count, mean, stddev, min, and max. If no columns are given, this function computes statistics for all numerical or string columns.

Parameters ***cols** –

Examples

```
>>> ds.describe(['col_name']).show()
>>> ds.describe().show()
```

distinct()
Returns a new DataStream containing the distinct rows in this DataStream.

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.distinct().count()
```

drop(**cols*)
Returns a new Datastream that drops the specified column. This is a no-op if schema doesn't contain the given column name(s).

Parameters ***cols** – a string name of the column to drop, or a Column to drop, or a list of string name of the columns to drop.

Returns

Return type Datastream

Examples

```
>>> ds.drop('col_name')
```

dropDuplicates(*subset=None*)
Return a new DataStream with duplicate rows removed, optionally only considering certain columns.

Parameters **subset** – optional list of column names to consider.

Returns

Return type Datastream

Examples

```
>>> ds.dropDuplicates().show()
>>> # Example on how to use it with params
>>> ds.dropDuplicates(['col_name1', 'col_name2']).show()
```

dropna(*how='any'*, *thresh=None*, *subset=None*)
Returns a new DataStream omitting rows with null values.

Parameters

- **how** – 'any' or 'all'. If 'any', drop a row if it contains any nulls. If 'all', drop a row only if all its values are null.

- **thresh** – int, default None If specified, drop rows that have less than thresh non-null values. This overwrites the how parameter.
- **subset** – optional list of column names to consider.

Returns

Return type Datastream

Examples

```
>>> ds.dropna()
```

exceptAll(*other*)

Return a new DataStream containing rows in this DataStream but not in another DataStream while preserving duplicates.

Parameters **other** – other DataStream object

Returns

Return type Datastream

Examples

```
>>> ds1.exceptAll(ds2).show()
```

explain(*extended=False*)

Prints the (logical and physical) plans to the console for debugging purpose.

Parameters **extended** – boolean, default False. If False, prints only the physical plan.

Examples

```
>>> ds.explain()
```

fillna(*value*, *subset=None*)

Replace null values

Parameters

- **value** – int, long, float, string, bool or dict. Value to replace null values with. If the value is a dict, then subset is ignored and value must be a mapping from column name (string) to replacement value. The replacement value must be an int, long, float, boolean, or string.
- **subset** – optional list of column names to consider. Columns specified in subset that do not have matching data type are ignored. For example, if value is a string, and subset contains a non-string column, then the non-string column is simply ignored.

Returns

Return type Datastream

Examples

```
>>> ds.fill(50).show()
>>> ds.fill({'col1': 50, 'col2': 'unknown'}).show()
```

filter(*condition*)
Filters rows using the given condition

Parameters **condition** – a Column of types.BooleanType or a string of SQL expression.

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.filter("age > 3")
>>> df.filter(df.age > 3)
```

filter_user(*user_ids: List*)
filter data to get only selective users' data

Parameters **user_ids** (*List[str]*) – list of users' UUIDs

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

filter_version(*version: List*)
filter data to get only selective users' data

Parameters **version** (*List[str]*) – list of stream versions

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Todo: Metadata version should be return with the data

---

first()
Returns the first row as a Row.

Returns First row of a DataStream

Examples

```
>>> ds.first()
```

foreach(*f*)
Applies the f function to all Row of DataStream. This is a shorthand for df.rdd.foreach()

Parameters **f** – function

Returns DataStream object

Examples

```
>>> def f(person):
...     print(person.name)
>>> ds.foreach(f)
```

freqItems(*cols*, *support=None*)
Finding frequent items for columns, possibly with false positives. Using the frequent element count algorithm described in "http://dx.doi.org/10.1145/762471.762473, proposed by Karp, Schenker, and Papadimitriou".

Returns

Return type *DataStream*

Examples

```
>>> ds.freqItems("col-name")
```

get_metadata() → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
get stream metadata

Returns single version of a stream

Return type *Metadata*

Raises Exception – if specified version is not available for the stream

groupby(**cols*)
Groups the DataFrame using the specified columns, so we can run aggregation on them. This method will return pyspark.sql.GroupedData object.

Parameters **of columns to group by. Each element should be a column name** (*list*) –

Returns:

head(*n=None*)
Returns the first n rows.

Parameters **n** (*int*) – default 1. Number of rows to return.

Returns If n is greater than 1, return a list of Row. If n is 1, return a single Row.

Notes

This method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver's memory.

Examples

```
>>> ds.head(5)
```

intersect(*other*)
Return a new DataFrame containing rows only in both this frame and another frame. This is equivalent to INTERSECT in SQL.

Parameters **other** (*int*) – DataStream object

Returns If n is greater than 1, return a list of Row. If n is 1, return a single Row.

Examples

```
>>> ds.intersect(other=ds2)
```

intersectAll(*other*)
Return a new DataFrame containing rows in both this dataframe and other dataframe while preserving duplicates.

Parameters **other** (*int*) – DataStream object

Returns If n is greater than 1, return a list of Row. If n is 1, return a single Row.

Examples

```
>>> ds.intersectAll(ds2).show()
```

join(*other*, *on=None*, *how=None*)
Joins with another DataStream, using the given join expression.

Parameters

- **other** (DataStream) – Right side of the join
- **– a string for the join column name, a list of column names, a join expression** (*on*) –
- **how** (*str*) – inner, cross, outer, full, full_outer, left, left_outer, right, right_outer, left_semi, and left_anti.

Examples

```
>>> ds.join(ds2, 'user', 'outer').show()
```

Returns DataStream object with joined streams

join_stress_streams(*dataStream*, *propagation='forward'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

limit(*num*)
Limits the result count to the number specified.

Parameters **num** –

Returns

Return type Datastream

map_stream(*window_ds*)
Map/join a stream to a windowed stream

Parameters **window_ds** (*Datastream*) – windowed datastream object

Returns joined/mapped stream

Return type Datastream

metadata
return stream metadata

Returns

Return type *Metadata*

orderBy(**cols*)
order by column name

Parameters ***cols** –

Returns

Return type Datastream

printSchema()
Prints out the schema in the tree format.

Examples

```
>>> ds.printSchema()
```

repartition(*numPartitions*, **cols*)
Returns a new DataStream partitioned by the given partitioning expressions. The resulting DataStream is hash partitioned.

numPartitions can be an int to specify the target number of partitions or a Column. If it is a Column, it will be used as the first partitioning column. If not specified, the default number of partitions is used.

Parameters

- **numPartitions** –
- ***cols** –

Returns:

replace(*to_replace*, *value*, *subset=None*)
Returns a new DataStream replacing a value with another value. Values to_replace and value must have the same type and can only be numerics, booleans, or strings. Value can have None. When replacing, the new value will be cast to the type of the existing column. For numeric replacements all values to be replaced should have unique floating point representation. In case of conflicts (for example with {42: -1, 42.0: 1}) and arbitrary replacement will be used.

Parameters

- **to_replace** – bool, int, long, float, string, list or dict. Value to be replaced. If the value is a dict, then value is ignored or can be omitted, and to_replace must be a mapping between a value and a replacement.

- **value** – bool, int, long, float, string, list or None. The replacement value must be a bool, int, long, float, string or None. If value is a list, value should be of the same length and type as to_replace. If value is a scalar and to_replace is a sequence, then value is used as a replacement for each item in to_replace.
- **subset** – optional list of column names to consider. Columns specified in subset that do not have matching data type are ignored. For example, if value is a string, and subset contains a non-string column, then the non-string column is simply ignored.

Returns

Return type Datastream

Examples

```
>>> ds.replace(10, 20).show()
>>> ds.replace('some-str', None).show()
>>> ds.replace(['old_val1', 'new_val1'], ['old_val2', 'new_val2'], 'col_name
↪').show()
```

select(**cols*)

Projects a set of expressions and returns a new DataStream :param cols: list of column names (string) or expressions (Column). If one of the column names is '*', that column is expanded to include all columns in the current DataStream :type cols: str

Returns this will return a new datastream object with selected columns

Return type *DataStream*

Examples

```
>>> ds.select('*')
>>> ds.select('name', 'age')
>>> ds.select(ds.name, (ds.age + 10).alias('age'))
```

selectExpr(**expr*)

This is a variant of select() that accepts SQL expressions. Projects a set of expressions and returns a new DataStream

Parameters **expr** (*str*) –

Returns this will return a new datastream object with selected columns

Return type *DataStream*

Examples

```
>>> ds.selectExpr("age * 2")
```

show(*n=20*, *truncate=True*, *vertical=False*)

Parameters

- **n** – Number of rows to show.
- **truncate** – If set to `True`, truncate strings longer than 20 chars by default.

- **set to a number greater than one, truncates long strings to length truncate** (*If*) –
- **align cells right.** (*and*) –
- **vertical** – If set to `True`, print output rows vertically (one line
- **column value)** (*per*) –

Returns:

sort (**cols*, ***kwargs*)

Returns a new DataStream sorted by the specified column(s).

Parameters

- **cols** – list of Column or column names to sort by.
- **ascending** – boolean or list of boolean (default True). Sort ascending vs. descending. Specify list for multiple sort orders. If a list is specified, length of the list must equal length of the cols.

Returns DataStream object

Return type object

Examples

```
>>> ds.sort("col_name", ascending=False)
```

summary (**statistics*)

Computes specified statistics for numeric and string columns. Available statistics are: - count - mean - stddev - min - max - arbitrary approximate percentiles specified as a percentage (eg, 75%) If no statistics are given, this function computes count, mean, stddev, min, approximate quartiles (percentiles at 25%, 50%, and 75%), and max.

Parameters ***statistics** –

Examples

```
>>> ds.summary().show()
>>> ds.summary("count", "min", "25%", "75%", "max").show()
>>> # To do a summary for specific columns first select them:
>>> ds.select("col1", "col2").summary("count").show()
```

take (*num*)

Returns the first num rows as a list of Row.

Returns row(s) of a DataStream

Return type Row(list)

Examples

```
>>> ds.take()
```

toPandas ()

This method converts pyspark dataframe into pandas dataframe.

Notes

This method will collect all the data on master node to convert pyspark dataframe into pandas dataframe. After converting to pandas dataframe datastream objects helper methods will not be accessible.

Returns this will return a new datastream object with blank metadata

Return type Datastream (*Metadata*, pandas.DataFrame)

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = CC.get_stream("STREAM-NAME")
>>> new_ds = ds.toPandas()
>>> new_ds.data.head()
```

union(*other*)

Return a new Datastream containing union of rows in this and another frame.

This is equivalent to UNION ALL in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by distinct().

Also as standard in SQL, this function resolves columns by position (not by name).

Parameters **other** (DataStream) –

Returns

Return type Datastream

Examples

```
>>> ds.union(ds2).collect()
```

unionByName(*other*)

Returns a new Datastream containing union of rows in this and another frame.

This is different from both UNION ALL and UNION DISTINCT in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by distinct().

The difference between this function and union() is that this function resolves columns by name (not by position):

Parameters **other** (DataStream) –

Returns

Return type Datastream

Examples

```
>>> ds.unionByName(ds2).show()
```

where(*condition*)

where() is an alias for filter().

Parameters **condition** –

Returns

Return type Datastream

Examples

```
>>> ds.filter("age > 3").collect()
```

window(*windowDuration: int = None*, *groupByColumnName: List[str] = []*, *slideDuration: int = None*, *startTime=None*, *preserve_ts=False*)

Window data into fixed length chunks. If no columnName is provided then the windowing will be performed on all the columns.

Parameters

- **windowDuration** (*int*) – duration of a window in seconds
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **slideDuration** (*int*) – slide duration of a window
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided
- **preserve_ts** (*bool*) – setting this to True will return timestamps of corresponding to each windowed value

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Note: This windowing method will use collect_list to return values for each window. collect_list is not optimized.

---

withColumn(*colName*, *col*)

Returns a new DataStream by adding a column or replacing the existing column that has the same name. The column expression must be an expression over this DataStream; attempting to add a column from some other datastream will raise an error. :param colName: name of the new column. :type colName: str :param col: a Column expression for the new column.

Examples

```
>>> ds.withColumn('col_name', ds.col_name + 2)
```

withColumnRenamed(*existing*, *new*)

Returns a new DataStream by renaming an existing column. This is a no-op if schema doesn't contain the given column name.

Parameters

- **existing** (*str*) – string, name of the existing column to rename.
- **new** (*str*) – string, new name of the column.

Examples

```
>>> ds.withColumnRenamed('col_name', 'new_col_name')
```

Returns DataStream object with new column name(s)

write()
Interface for saving the content of the non-streaming DataFrame out into external storage.

Returns DataFrameWriter

New in version 1.4.

writeStream()
Interface for saving the content of the streaming DataFrame out into external storage.

Note: Evolving.

Returns DataStreamWriter

New in version 2.0.

get_window(*x*)

windowing_udf(*x*)

Module contents

class DataStream(*data: object = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*)
Bases: pyspark.sql.dataframe.DataFrame

agg(**exprs*)
Aggregate on the entire DataStream without groups

Parameters ***exprs** –

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.agg({"age": "max"}).collect()
>>> # Below example shows how to use pyspark functions in add method
>>> from pyspark.sql import functions as F
>>> ds.agg(F.min(ds.age)).collect()
```

alias(*alias*)
Returns a new DataStream with an alias set.

Parameters **alias** – string, an alias name to be set for the datastream.

Returns DataStream object

Return type object

Examples

```
>>> df_as1 = df.alias("df_as1")
>>> df_as2 = df.alias("df_as2")
```

approxQuantile(*col*, *probabilities*, *relativeError*)

Calculates the approximate quantiles of numerical columns of a DataStream.

The result of this algorithm has the following deterministic bound: If the DataStream has N elements and if we request the quantile at probability p up to error err, then the algorithm will return a sample x from the DataStream so that the exact rank of x is close to (p * N). More precisely,

floor((p - err) * N) <= rank(x) <= ceil((p + err) * N).

This method implements a variation of the Greenwald-Khanna algorithm (with some speed optimizations). The algorithm was first present in [[http://dx.doi.org/10.1145/375663.375670 Space-efficient Online Computation of Quantile Summaries]] by Greenwald and Khanna.

Note that null values will be ignored in numerical columns before calculation. For columns only containing null values, an empty list is returned.

Parameters

- **col** (*str[list]*) – Can be a single column name, or a list of names for multiple columns.
- **probabilities** – a list of quantile probabilities Each number must belong to [0, 1]. For example 0 is the minimum, 0.5 is the median, 1 is the maximum.
- **relativeError** – The relative target precision to achieve (>= 0). If set to zero, the exact quantiles are computed, which could be very expensive. Note that values greater than 1 are accepted but give the same result as 1.

Returns the approximate quantiles at the given probabilities. If the input col is a string, the output is a list of floats. If the input col is a list or tuple of strings, the output is also a list, but each element in it is a list of floats, i.e., the output is a list of list of floats.

colRegex(*colName*)

Selects column based on the column name specified as a regex and returns it as Column.

Parameters **colName** (*str*) – column name specified as a regex.

Returns

Return type *DataStream*

Examples

```
>>> ds.colRegex("colName")
```

collect()

Collect all the data to master node and return list of rows

Returns rows of all the dataframe

Return type List

Examples

```
>>> ds.collect()
```

compute(*udfName*, *windowDuration: int = None*, *slideDuration: int = None*, *groupByColumnName: List[str] = []*, *startTime=None*)

Run an algorithm. This method supports running an udf method on windowed data

Parameters

- **udfName** – Name of the algorithm
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

corr(*col1*, *col2*, *method=None*)

Calculates the correlation of two columns of a DataStream as a double value. Currently only supports the Pearson Correlation Coefficient.

Parameters

- **col1** (*str*) – The name of the first column
- **col2** (*str*) – The name of the second column
- **method** (*str*) – The correlation method. Currently only supports "pearson"

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.corr("col1", "col2", "pearson").collect()
```

count()

Returns the number of rows in this DataStream.

Examples

```
>>> ds.count()
```

cov(*col1*, *col2*)

Calculate the sample covariance for the given columns, specified by their names, as a double value.

Parameters

- **col1** (*str*) – The name of the first column
- **col2** (*str*) – The name of the second column

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.cov("cal1", "col2", "pearson").collect()
```

create_windows(*window_length='hour'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

crossJoin(*other*)
Returns the cartesian product with another DataStream

Parameters **other** – Right side of the cartesian product.

Returns DataStream object with joined streams

Examples

```
>>> ds.crossJoin(ds2.select("col_name")).collect()
```

crosstab(*col1*, *col2*)
Computes a pair-wise frequency table of the given columns. Also known as a contingency table. The number of distinct values for each column should be less than 1e4. At most 1e6 non-zero pair frequencies will be returned. The first column of each row will be the distinct values of col1 and the column names will be the distinct values of col2. The name of the first column will be $col1_$col2. Pairs that have no occurrences will have zero as their counts.

Parameters

- **col1** (*str*) – The name of the first column. Distinct items will make the first item of each row.
- **col2** (*str*) – The name of the second column. Distinct items will make the column names of the DataStream.

Returns DataStream object

Examples

```
>>> ds.crosstab("col_1", "col_2")
```

data
get stream data

Returns (DataFrame):

describe(**cols*)
Computes basic statistics for numeric and string columns. This include count, mean, stddev, min, and max. If no columns are given, this function computes statistics for all numerical or string columns.

Parameters ***cols** –

Examples

```
>>> ds.describe(['col_name']).show()
>>> ds.describe().show()
```

distinct()
Returns a new DataStream containing the distinct rows in this DataStream.

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.distinct().count()
```

drop(**cols*)
Returns a new Datastream that drops the specified column. This is a no-op if schema doesn't contain the given column name(s).

Parameters ***cols** – a string name of the column to drop, or a Column to drop, or a list of string name of the columns to drop.

Returns

Return type Datastream

Examples

```
>>> ds.drop('col_name')
```

dropDuplicates(*subset=None*)
Return a new DataStream with duplicate rows removed, optionally only considering certain columns.

Parameters **subset** – optional list of column names to consider.

Returns

Return type Datastream

Examples

```
>>> ds.dropDuplicates().show()
>>> # Example on how to use it with params
>>> ds.dropDuplicates(['col_name1', 'col_name2']).show()
```

dropna(*how='any'*, *thresh=None*, *subset=None*)
Returns a new DataStream omitting rows with null values.

Parameters

- **how** – 'any' or 'all'. If 'any', drop a row if it contains any nulls. If 'all', drop a row only if all its values are null.
- **thresh** – int, default None If specified, drop rows that have less than thresh non-null values. This overwrites the how parameter.
- **subset** – optional list of column names to consider.

Returns

Return type Datastream

Examples

```
>>> ds.dropna()
```

exceptAll(*other*)
Return a new DataStream containing rows in this DataStream but not in another DataStream while preserving duplicates.

Parameters **other** – other DataStream object

Returns

Return type Datastream

Examples

```
>>> ds1.exceptAll(ds2).show()
```

explain(*extended=False*)
Prints the (logical and physical) plans to the console for debugging purpose.

Parameters **extended** – boolean, default False. If False, prints only the physical plan.

Examples

```
>>> ds.explain()
```

fillna(*value*, *subset=None*)
Replace null values

Parameters

- **value** – int, long, float, string, bool or dict. Value to replace null values with. If the value is a dict, then subset is ignored and value must be a mapping from column name (string) to replacement value. The replacement value must be an int, long, float, boolean, or string.
- **subset** – optional list of column names to consider. Columns specified in subset that do not have matching data type are ignored. For example, if value is a string, and subset contains a non-string column, then the non-string column is simply ignored.

Returns

Return type Datastream

Examples

```
>>> ds.fill(50).show()
>>> ds.fill({'col1': 50, 'col2': 'unknown'}).show()
```

filter(*condition*)
Filters rows using the given condition

Parameters **condition** – a Column of types.BooleanType or a string of SQL expression.

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Examples

```
>>> ds.filter("age > 3")
>>> df.filter(df.age > 3)
```

filter_user(*user_ids: List*)
filter data to get only selective users' data

Parameters **user_ids** (*List[str]*) – list of users' UUIDs

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

filter_version(*version: List*)
filter data to get only selective users' data

Parameters **version** (*List[str]*) – list of stream versions

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Todo: Metadata version should be return with the data

---

first()
Returns the first row as a Row.

Returns First row of a DataStream

Examples

```
>>> ds.first()
```

foreach(*f*)
Applies the f function to all Row of DataStream. This is a shorthand for df.rdd.foreach()

Parameters **f** – function

Returns DataStream object

Examples

```
>>> def f(person):
...     print(person.name)
>>> ds.foreach(f)
```

freqItems(*cols*, *support=None*)
Finding frequent items for columns, possibly with false positives. Using the frequent element count algorithm described in "http://dx.doi.org/10.1145/762471.762473, proposed by Karp, Schenker, and Papadimitriou".

Returns

Return type *DataStream*

Examples

```
>>> ds.freqItems("col-name")
```

get_metadata() → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
get stream metadata

Returns single version of a stream

Return type *Metadata*

Raises Exception – if specified version is not available for the stream

groupby(**cols*)
Groups the DataFrame using the specified columns, so we can run aggregation on them. This method will return pyspark.sql.GroupedData object.

Parameters of columns to group by. Each element should be a column name (*list*) –

Returns:

head(*n=None*)
Returns the first n rows.

Parameters **n** (*int*) – default 1. Number of rows to return.

Returns If n is greater than 1, return a list of Row. If n is 1, return a single Row.

Notes

This method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver's memory.

Examples

```
>>> ds.head(5)
```

intersect(*other*)
Return a new DataFrame containing rows only in both this frame and another frame. This is equivalent to INTERSECT in SQL.

Parameters **other** (*int*) – DataStream object

Returns If n is greater than 1, return a list of Row. If n is 1, return a single Row.

Examples

```
>>> ds.intersect(other=ds2)
```

intersectAll(*other*)
Return a new DataFrame containing rows in both this dataframe and other dataframe while preserving duplicates.

Parameters **other** (*int*) – DataStream object

Returns If n is greater than 1, return a list of Row. If n is 1, return a single Row.

Examples

```
>>> ds.intersectAll(ds2).show()
```

join(*other*, *on=None*, *how=None*)
Joins with another DataStream, using the given join expression.

Parameters

- **other** (DataStream) – Right side of the join
- **– a string for the join column name, a list of column names, a join expression** (*on*) –
- **how** (*str*) – inner, cross, outer, full, full_outer, left, left_outer, right, right_outer, left_semi, and left_anti.

Examples

```
>>> ds.join(ds2, 'user', 'outer').show()
```

Returns DataStream object with joined streams

join_stress_streams(*dataStream*, *propagation='forward'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

limit(*num*)
Limits the result count to the number specified.

Parameters **num** –

Returns

Return type Datastream

map_stream(*window_ds*)
Map/join a stream to a windowed stream

Parameters **window_ds** (*Datastream*) – windowed datastream object

Returns joined/mapped stream

Return type Datastream

metadata
return stream metadata

Returns

Return type *Metadata*

orderBy(**cols*)
order by column name

Parameters ***cols** –

Returns

Return type Datastream

printSchema()
Prints out the schema in the tree format.

Examples

```
>>> ds.printSchema()
```

repartition(*numPartitions*, **cols*)
Returns a new DataStream partitioned by the given partitioning expressions. The resulting DataStream is hash partitioned.

numPartitions can be an int to specify the target number of partitions or a Column. If it is a Column, it will be used as the first partitioning column. If not specified, the default number of partitions is used.

Parameters

- **numPartitions** –
- ***cols** –

Returns:

replace(*to_replace*, *value*, *subset=None*)
Returns a new DataStream replacing a value with another value. Values to_replace and value must have the same type and can only be numerics, booleans, or strings. Value can have None. When replacing, the new value will be cast to the type of the existing column. For numeric replacements all values to be replaced should have unique floating point representation. In case of conflicts (for example with {42: -1, 42.0: 1}) and arbitrary replacement will be used.

Parameters

- **to_replace** – bool, int, long, float, string, list or dict. Value to be replaced. If the value is a dict, then value is ignored or can be omitted, and to_replace must be a mapping between a value and a replacement.

- **value** – bool, int, long, float, string, list or None. The replacement value must be a bool, int, long, float, string or None. If value is a list, value should be of the same length and type as to_replace. If value is a scalar and to_replace is a sequence, then value is used as a replacement for each item in to_replace.
- **subset** – optional list of column names to consider. Columns specified in subset that do not have matching data type are ignored. For example, if value is a string, and subset contains a non-string column, then the non-string column is simply ignored.

Returns

Return type Datastream

Examples

```
>>> ds.replace(10, 20).show()
>>> ds.replace('some-str', None).show()
>>> ds.replace(['old_val1', 'new_val1'], ['old_val2', 'new_val2'], 'col_name
↪').show()
```

select(**cols*)

Projects a set of expressions and returns a new DataStream :param cols: list of column names (string) or expressions (Column). If one of the column names is '*', that column is expanded to include all columns in the current DataStream :type cols: str

Returns this will return a new datastream object with selected columns

Return type *DataStream*

Examples

```
>>> ds.select('*')
>>> ds.select('name', 'age')
>>> ds.select(ds.name, (ds.age + 10).alias('age'))
```

selectExpr(**expr*)

This is a variant of select() that accepts SQL expressions. Projects a set of expressions and returns a new DataStream

Parameters **expr** (*str*) –

Returns this will return a new datastream object with selected columns

Return type *DataStream*

Examples

```
>>> ds.selectExpr("age * 2")
```

show(*n=20*, *truncate=True*, *vertical=False*)

Parameters

- **n** – Number of rows to show.
- **truncate** – If set to `True`, truncate strings longer than 20 chars by default.

- **set to a number greater than one, truncates long strings to length truncate** (*If*) –
- **align cells right.** (*and*) –
- **vertical** – If set to `True`, print output rows vertically (one line
- **column value)** (*per*) –

Returns:

sort (**cols*, ***kwargs*)

Returns a new DataStream sorted by the specified column(s).

Parameters

- **cols** – list of Column or column names to sort by.
- **ascending** – boolean or list of boolean (default True). Sort ascending vs. descending. Specify list for multiple sort orders. If a list is specified, length of the list must equal length of the cols.

Returns DataStream object

Return type object

Examples

```
>>> ds.sort("col_name", ascending=False)
```

summary (**statistics*)

Computes specified statistics for numeric and string columns. Available statistics are: - count - mean - stddev - min - max - arbitrary approximate percentiles specified as a percentage (eg, 75%) If no statistics are given, this function computes count, mean, stddev, min, approximate quartiles (percentiles at 25%, 50%, and 75%), and max.

Parameters ***statistics** –

Examples

```
>>> ds.summary().show()
>>> ds.summary("count", "min", "25%", "75%", "max").show()
>>> # To do a summary for specific columns first select them:
>>> ds.select("col1", "col2").summary("count").show()
```

take (*num*)

Returns the first num rows as a list of Row.

Returns row(s) of a DataStream

Return type Row(list)

Examples

```
>>> ds.take()
```

toPandas ()

This method converts pyspark dataframe into pandas dataframe.

Notes

This method will collect all the data on master node to convert pyspark dataframe into pandas dataframe. After converting to pandas dataframe datastream objects helper methods will not be accessible.

Returns this will return a new datastream object with blank metadata

Return type Datastream (*Metadata*, pandas.DataFrame)

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = CC.get_stream("STREAM-NAME")
>>> new_ds = ds.toPandas()
>>> new_ds.data.head()
```

union(*other*)

Return a new Datastream containing union of rows in this and another frame.

This is equivalent to UNION ALL in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by distinct().

Also as standard in SQL, this function resolves columns by position (not by name).

Parameters **other** (DataStream) –

Returns

Return type Datastream

Examples

```
>>> ds.union(ds2).collect()
```

unionByName(*other*)

Returns a new Datastream containing union of rows in this and another frame.

This is different from both UNION ALL and UNION DISTINCT in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by distinct().

The difference between this function and union() is that this function resolves columns by name (not by position):

Parameters **other** (DataStream) –

Returns

Return type Datastream

Examples

```
>>> ds.unionByName(ds2).show()
```

where(*condition*)

where() is an alias for filter().

Parameters **condition** –

Returns

Return type Datastream

Examples

```
>>> ds.filter("age > 3").collect()
```

window(*windowDuration: int = None*, *groupByColumnName: List[str] = []*, *slideDuration: int = None*, *startTime=None*, *preserve_ts=False*)

Window data into fixed length chunks. If no columnName is provided then the windowing will be performed on all the columns.

Parameters

- **windowDuration** (*int*) – duration of a window in seconds
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **slideDuration** (*int*) – slide duration of a window
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided
- **preserve_ts** (*bool*) – setting this to True will return timestamps of corresponding to each windowed value

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Note: This windowing method will use collect_list to return values for each window. collect_list is not optimized.

---

withColumn(*colName*, *col*)

Returns a new DataStream by adding a column or replacing the existing column that has the same name. The column expression must be an expression over this DataStream; attempting to add a column from some other datastream will raise an error. :param colName: name of the new column. :type colName: str :param col: a Column expression for the new column.

Examples

```
>>> ds.withColumn('col_name', ds.col_name + 2)
```

withColumnRenamed(*existing*, *new*)

Returns a new DataStream by renaming an existing column. This is a no-op if schema doesn't contain the given column name.

Parameters

- **existing** (*str*) – string, name of the existing column to rename.
- **new** (*str*) – string, new name of the column.

Examples

```
>>> ds.withColumnRenamed('col_name', 'new_col_name')
```

Returns DataStream object with new column name(s)

write()
Interface for saving the content of the non-streaming DataFrame out into external storage.

Returns DataFrameWriter

New in version 1.4.

writeStream()
Interface for saving the content of the streaming DataFrame out into external storage.

Note: Evolving.

Returns DataStreamWriter

New in version 2.0.

cerebralcortex.core.log_manager package

Submodules

cerebralcortex.core.log_manager.log_handler module

class LogHandler
Bases: object

log(*error_message=''*, *error_type=(1,)*)

class LogTypes
Bases: object

CRITICAL = (2,)

DEBUG = 6

ERROR = (3,)

EXCEPTION = (1,)

MISSING_DATA = (5,)

WARNING = (4,)

cerebralcortex.core.log_manager.logging module

class CCLogging(*CC*)
Bases: *cerebralcortex.core.log_manager.log_handler.LogHandler*

Module contents

cerebralcortex.core.metadata_manager package

Subpackages

cerebralcortex.core.metadata_manager.stream package

Submodules

cerebralcortex.core.metadata_manager.stream.data_descriptor module

class DataDescriptor

Bases: object

from_json(*obj*)

Cast DataDescriptor class object into json

Parameters **obj** (DataDescriptor) – object of a data descriptor class

Returns

Return type self

set_attribute(*key*, *value*)

Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises ValueError – if key/value are missing

set_name(*value*)

Name of data descriptor

Parameters **value** (*str*) – name

Returns

Return type self

set_type(*value: str*)

Type of a data descriptor

Parameters **value** (*str*) – type

Returns

Return type self

cerebralcortex.core.metadata_manager.stream.metadata module

class Metadata

Bases: object

add_annotation(*annotation: str*)

Add annotation stream name

Parameters **annotation** (*str*) – name of annotation or list of strings

Returns self

add_dataDescriptor(*dd: cerebralcortex.core.metadata_manager.stream.data_descriptor.DataDescriptor*)

Add data description of a stream

Parameters **dd** (DataDescriptor) – data descriptor

Returns self

add_input_stream(*input_stream: str*)

Add input streams that were used to derive a new stream

Parameters **input_stream** (*str*) – name of input stream OR list of input_stream names

Returns self

add_module(*mod: cerebralcortex.core.metadata_manager.stream.module_info.ModuleMetadata*)

Add module metadata

Parameters **mod** (ModuleMetadata) – module metadata

Returns self

from_json_file(*metadata: dict*) → List

Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

from_json_sql(*metadata_json: dict*) → List

Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

get_dataDescriptor(*name*)

get data descriptor by name

Parameters **name** (*str*) –

Returns DataDescriptor object

get_hash() → str

Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Returns hash id of metadata

Return type str

get_hash_by_json(*metadata: dict = None*) → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Parameters **metadata** – only pass this if this method is used on a dict object outside of Metadata class

Returns hash id of metadata

Return type str

get_name()
Returns: name of a stream

is_valid() → bool
check whether all required fields are set

Returns True if fields are set or throws an exception in case of missing values

Return type bool

Exception: ValueError: if metadata fields are not set

set_description(*stream_description: str*)
Add stream description

Parameters **stream_description** (*str*) – textual description of a stream

Returns self

set_name(*value: str*)
set name of a stream

Parameters **value** (*str*) – name of a stream

Returns self

set_study_name(*value: str*)
set study name

Parameters **value** (*str*) – study name

Returns self

to_json() → dict
Convert MetaData object into a dict (json) object

Returns dict form of MetaData object

Return type dict

cerebralcortex.core.metadata_manager.stream.module_info module

class ModuleMetadata
Bases: object

from_json(*obj*)
Cast ModuleMetadata class object into json

Parameters **obj** (ModuleMetadata) – object of a ModuleMetadata class

Returns

Return type self

set_attribute(*key: str*, *value: str*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises ValueError – if key/value are missing

set_author(*key*, *value*)
set author key/value pair. For example, key=name, value=md2k

Parameters

- **key** (*str*) – author metadata key
- **value** (*str*) – author metadata value

Returns

Return type self

set_authors(*authors*)
set author key/value pair. For example, key=name, value=md2k

Parameters **authors** (*list[dict]*) – List of authors names and emails ids in dict. For example, authors = [{"ali":"ali@gmail.com"}, {"nasir":"nasir@gmail.com"}]

Returns

Return type self

set_name(*value*)
name of the module

Parameters **value** (*str*) – name

Returns

Return type self

set_version(*value*)
version of the module

Parameters **value** (*str*) – version

Returns

Return type self

Module contents

class Metadata
Bases: object

add_annotation(*annotation: str*)
Add annotation stream name

Parameters **annotation** (*str*) – name of annotation or list of strings

Returns self

add_dataDescriptor(*dd: cerebralcortex.core.metadata_manager.stream.data_descriptor.DataDescriptor*)
Add data description of a stream

Parameters **dd** (DataDescriptor) – data descriptor

Returns self

add_input_stream(*input_stream: str*)
Add input streams that were used to derive a new stream

Parameters **input_stream** (*str*) – name of input stream OR list of input_stream names

Returns self

add_module(*mod: cerebralcortex.core.metadata_manager.stream.module_info.ModuleMetadata*)
Add module metadata

Parameters **mod** (ModuleMetadata) – module metadata

Returns self

from_json_file(*metadata: dict*) → List
Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

from_json_sql(*metadata_json: dict*) → List
Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

get_dataDescriptor(*name*)
get data descriptor by name

Parameters **name** (*str*) –

Returns DataDescriptor object

get_hash() → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Returns hash id of metadata

Return type str

get_hash_by_json(*metadata: dict = None*) → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Parameters **metadata** – only pass this if this method is used on a dict object outside of Metadata class

Returns hash id of metadata

Return type str

get_name()
Returns: name of a stream

is_valid() → bool
check whether all required fields are set

Returns True if fields are set or throws an exception in case of missing values

Return type bool

Exception: ValueError: if metadata fields are not set

set_description(*stream_description: str*)
Add stream description

Parameters **stream_description** (*str*) – textual description of a stream

Returns self

set_name(*value: str*)
set name of a stream

Parameters **value** (*str*) – name of a stream

Returns self

set_study_name(*value: str*)
set study name

Parameters **value** (*str*) – study name

Returns self

to_json() → dict
Convert MetaData object into a dict (json) object

Returns dict form of MetaData object

Return type dict

class DataDescriptor
Bases: object

from_json(*obj*)
Cast DataDescriptor class object into json

Parameters **obj** (DataDescriptor) – object of a data descriptor class

Returns

Return type self

set_attribute(*key*, *value*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises ValueError – if key/value are missing

set_name (*value*)
Name of data descriptor

Parameters **value** (*str*) – name

Returns

Return type self

set_type (*value: str*)
Type of a data descriptor

Parameters **value** (*str*) – type

Returns

Return type self

class ModuleMetadata
Bases: object

from_json (*obj*)
Cast ModuleMetadata class object into json

Parameters **obj** (ModuleMetadata) – object of a ModuleMetadata class

Returns

Return type self

set_attribute (*key: str*, *value: str*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises ValueError – if key/value are missing

set_author (*key*, *value*)
set author key/value pair. For example, key=name, value=md2k

Parameters

- **key** (*str*) – author metadata key
- **value** (*str*) – author metadata value

Returns

Return type self

set_authors (*authors*)
set author key/value pair. For example, key=name, value=md2k

Parameters **authors** (*list[dict]*) – List of authors names and emails ids in dict. For example, authors = [{"ali":"ali@gmail.com"}, {"nasir":"nasir@gmail.com"}]

Returns

Return type self

set_name(*value*)
name of the module

Parameters **value** (*str*) – name

Returns

Return type self

set_version(*value*)
version of the module

Parameters **value** (*str*) – version

Returns

Return type self

cerebralcortex.core.metadata_manager.user package

Submodules

cerebralcortex.core.metadata_manager.user.user module

class User(*user_id: uuid.UUID*, *username: str*, *password: str*, *token: str = None*, *token_issued_at: datetime.datetime = None*, *token_expiry: datetime.datetime = None*, *user_role: datetime.datetime = None*, *user_metadata: dict = None*, *active: bool = 1*)
Bases: object

isactive
user status

Type Returns (int)

password
encrypted password

Type Returns

Type (str)

token
auth token

Type Returns

Type (str)

token_expiry
date and time when token will expire

Type Returns

Type (datetime)

token_issued_at
date and time when token was issues

Type Returns

Type (datetime)

user_id
user id

Type Returns

Type (str)

user_metadata
metadata of a user

Type Returns (dict)

user_role
role

Type Returns (str)

username
user name

Type Returns

Type (str)

Module contents

Module contents

cerebralcortex.core.util package

Submodules

cerebralcortex.core.util.data_formats module

msgpack_to_pandas(*input_data: object*) → pandas.core.frame.DataFrame
Convert msgpack binary file into pandas dataframe

Parameters **input_data** (*msgpack*) – msgpack data file

Returns pandas dataframe

Return type dataframe

pandas_to_msgpack(*df: pandas.core.frame.DataFrame*, *file_name*) → object
Convert pandas dataframe to msgpack format

Parameters **df** (*pd.DataFrame*) – pandas dataframe

cerebralcortex.core.util.datetime_helper_methods module

get_timezone(*tz_offset: float*, *common_only: bool = False*)
Returns a timezone for a given offset in milliseconds

Parameters

- **tz_offset** (*float*) – in milliseconds
- **common_only** (*bool*) –

Returns timezone of an offset

Return type str

cerebralcortex.core.util.spark_helper module

get_or_create_sc(*type='sparkContext'*, *name='CerebralCortex-Kernal'*, *enable_spark_ui=False*)
get or create spark context

Parameters

- **type** (*str*) – type (sparkContext, SparkSessionBuilder, sparkSession, sqlContext). (default="sparkContext")
- **name** (*str*) – spark app name (default="CerebralCortex-Kernal")

Returns:

Module contents

Module contents

cerebralcortex.examples package

Submodules

cerebralcortex.examples.brushing_detection module

generate_candidates(*CC*, *user_id*, *accel_stream_name*, *gyro_stream_name*, *output_stream_name*)

generate_features(*CC*, *user_id*, *candidate_stream_name*, *output_stream_name*)

predict_brushing(*CC*, *user_id*, *features_stream_name*)

cerebralcortex.examples.mprov_get module

cerebralcortex.examples.mprov_gps_example module

cerebralcortex.examples.stress_from_ecg module

Module contents

cerebralcortex.markers package

Subpackages

cerebralcortex.markers.brushing package

Submodules

cerebralcortex.markers.brushing.features module

compute_corr_mse_accel_gyro(*self, exclude_col_names: list = [], accel_column_names: list = ['accelerometer_x', 'accelerometer_y', 'accelerometer_z'], gyro_column_names: list = ['gyroscope_y', 'gyroscope_x', 'gyroscope_z'], windowDuration: int = None, slideDuration: int = None, groupByColumnName: List[str] = [], startTime=None*)

Compute correlation and mean standard error of accel and gyro sensors

Parameters

- **list** (*gyro_column_names*) – name of the columns on which features should not be computed
- **list** – name of accel data column
- **list** – name of gyro data column
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

Returns DataStream object with all the existing data columns and FFT features

compute_fourier_features(*self, exclude_col_names: list = [], feature_names=['fft_centroid', 'fft_spread', 'spectral_entropy', 'spectral_entropy_old', 'fft_flux', 'spectral_falloff'], windowDuration: int = None, slideDuration: int = None, groupByColumnName: List[str] = [], startTime=None*)

Transforms data from time domain to frequency domain.

Parameters

- **list** (*feature_names*) – name of the columns on which features should not be computed
- **list** – names of the features. Supported features are fft_centroid, fft_spread, spectral_entropy, spectral_entropy_old, fft_flux, spectral_falloff
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** (*groupByColumnName*) – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

Returns DataStream object with all the existing data columns and FFT features

cerebralcortex.markers.brushing.main module

generate_candidates(*CC*, *user_id*, *accel_stream_name*, *gyro_stream_name*, *output_stream_name*)

generate_features(*CC*, *user_id*, *candidate_stream_name*, *output_stream_name*)

predict_brushing(*CC*, *user_id*, *features_stream_name*)

cerebralcortex.markers.brushing.util module

classify_brushing(*X: pandas.core.frame.DataFrame*, *model_file_name: str*)

filter_candidates(*ds*)

get_candidates(*ds*, *uper_limit: float = 0.1*, *threshold: float = 0.5*)
Get brushing candidates. Data is windowed into potential brushing candidate :param ds: :type ds: DataStream :param uper_limit: threashold for accel. This is used to know how high the hand is :type uper_limit: float :param threshold: :type threshold: float

Returns:

get_max_features(*ds*)
This method will compute what are the max values for accel and gyro statistical/FFT features :param ds: :type ds: DataStream

Returns DataStream

get_orientation_data(*ds*, *wrist*, *ori=1*, *is_new_device=False*, *accelerometer_x='accelerometer_x'*, *accelerometer_y='accelerometer_y'*, *accelerometer_z='accelerometer_z'*, *gyroscope_x='gyroscope_x'*, *gyroscope_y='gyroscope_y'*, *gyroscope_z='gyroscope_z'*)
Get the orientation of hand using accel and gyro data. :param ds: DataStream object :param wrist: name of the wrist smart watch was worn :param ori: :param is_new_device: this param is for motionsense smart watch version :param accelerometer_x: :type accelerometer_x: float :param accelerometer_y: :type accelerometer_y: float :param accelerometer_z: :type accelerometer_z: float :param gyroscope_x: :type gyroscope_x: float :param gyroscope_y: :type gyroscope_y: float :param gyroscope_z: :type gyroscope_z: float

Returns DataStream object

reorder_columns(*ds*)

Module contents

cerebralcortex.markers.ecg_stress package

Submodules

cerebralcortex.markers.ecg_stress.stress_from_ecg module

stress_from_ecg(*ecg_data: cerebralcortex.core.datatypes.datastream.DataStream*, *sensor_name: str = 'autosense'*, *Fs: int = 64*, *model_path='./model/stress_ecg_final.p'*)
Compute stress episodes from ecg timeseries data

Parameters

- **ecg_data** (DataStream) – ecg data

- **sensor_name** (*str*) – name of the sensor used to collect ecg data. Currently supports 'autosense' only
- **Fs** (*int*) – frequency of sensor data

Returns stress episodes

Return type *DataStream*

Module contents

cerebralcortex.markers.mcontain package

Submodules

cerebralcortex.markers.mcontain.assign_covid_user module

assign_covid_user(*data*, *covid_users*)

make_CC_object(*config_dir='/home/jupyter/cc3_conf/'*, *study_name='mcontain'*)

save_data(*CC*, *data_result*, *centroid_present=True*, *metadata=None*)

cerebralcortex.markers.mcontain.daily_encounter_stats module

assign_covid_user(*data*, *covid_users*)

drop_centroid_columns(*data_result*, *centroid_present=True*)

generate_metadata_dailystats()

generate_metadata_encounter_daily()

generate_metadata_notif()

generate_metadata_notification_daily()

generate_metadata_user_encounter_count()

generate_metadata_visualization_daily()

get_notifications(*encounter_final_data_with_gps*, *day*, *multiplier=10*, *column_name='total_encounters'*, *metric_threshold=1*)

get_time_columns(*encounter_final_data*, *start_time*, *end_time*, *utc_offset*)

get_utcoffset()

remove_duplicate_encounters_day(*data*)

cerebralcortex.markers.mcontain.hourly_encounters module

combine_base_encounters(*base_encounters*, *time_threshold=600*)

compute_encounters(*data_all_v3*, *data_all_v4*, *data_map_stream*, *data_key_stream*, *start_time*, *end_time*, *ltime=True*)

compute_encounters_only_v4(*data_all_v4*, *data_key_stream*, *start_time*, *end_time*, *ltime=True*)

drop_centroid_columns(*data_result*, *centroid_present=True*)

generate_metadata_encounter()

generate_metadata_hourly()

generate_visualization_hourly(*data_all_v3*, *data_all_v4*, *data_map_stream*, *data_key_stream*, *start_time*, *end_time*, *ltime=True*)

get_key_stream(*data_key_stream*, *start_time*, *end_time*, *datetime_format='%Y-%m-%d %H:%m'*)

get_utcoffset()

groupby_final(*data_key_stream*)

match_keys(*base_encounters*, *data_key_stream*)

transform_beacon_data_columns(*data_all*)

Module contents

Module contents

cerebralcortex.plotting package

Subpackages

cerebralcortex.plotting.basic package

Submodules

cerebralcortex.plotting.basic.plots module

plot_hist(*ds*, *user_id: str*, *x_axis_column=None*)
histogram plot of timeseries data

Parameters

- **ds** (DataStream) –
- **user_id** (*str*) – uuid of a user
- **x_axis_column** (*str*) – x axis column of the plot

plot_timeseries(*ds: cerebralcortex.core.datatypes.datastream.DataStream*, *user_id: str*, *y_axis_column: str = None*)
line plot of timeseries data

Parameters

- **ds** (DataStream) –
- **user_id** (*str*) – uuid of a user
- **y_axis_column** (*str*) – x axis column is hard coded as timestamp column. only y-axis can be passed as a param

Module contents

cerebralcortex.plotting.gps package

Submodules

cerebralcortex.plotting.gps.plots module

plot_gps_clusters(*ds*, *user_id: str*, *zoom=5*)
Plots GPS coordinates

Parameters

- **ds** (DataStream) – datastream object
- **user_id** (*str*) – uuid of a user
- **zoom** – min 0 and max 100, zoom map

Module contents

cerebralcortex.plotting.stress package

Submodules

cerebralcortex.plotting.stress.plots module

plot_bar(*ds*, *x_axis_column='stresser_main'*)

Parameters

- **ds** –
- **user_id** –
- **x_axis_column** –

plot_comparison(*ds*, *x_axis_column='stresser_main'*, *usr_id=None*, *compare_with='all'*)

Parameters

- **ds** –
- **x_axis_column** –
- **usr_id** –
- **compare_with** –

plot_gantt(*ds*, *user_id*)

Parameters

- **ds** –
- **user_id** –

plot_pie(*ds*, *user_id*, *group_by_column='stresser_main'*)

Parameters

- **ds** –
- **user_id** –
- **group_by_column** –

plot_sankey(*ds, user_id, cat_cols=['stresser_main', 'stresser_sub'], value_cols='density', title="Stressers' Sankey Diagram"*)

Parameters

- **ds** –
- **user_id** –
- **cat_cols** –
- **value_cols** –
- **title** –

Module contents

Submodules

cerebralcortex.plotting.util module

ds_to_pdf(*ds*, *user_id=None*) → pandas.core.frame.DataFrame
converts DataStream object into pandas dataframe :param ds: :type ds: DataStream

Returns pandas.DataFrame

Module contents

cerebralcortex.test_suite package

Subpackages

cerebralcortex.test_suite.algorithms package

Subpackages

cerebralcortex.test_suite.algorithms.glucose package

Module contents

Module contents

cerebralcortex.test_suite.util package

Submodules

cerebralcortex.test_suite.util.data_helper module

gen_location_datastream(*user_id*, *stream_name*) → object
Create pyspark dataframe with some sample gps data (Memphis, TN, lat, long, alt coordinates)

Parameters

- **user_id** (*str*) – id of a user
- **stream_name** (*str*) – sample gps stream name

Returns datastream object of gps location stream with its metadata

Return type *DataStream*

gen_phone_battery_data() → object
Create pyspark dataframe with some sample phone battery data

Returns pyspark dataframe object with columns: ["timestamp", "offset", "battery_level", "ver", "user"]

Return type DataFrame

gen_phone_battery_data2() → object
Create pyspark dataframe with some sample phone battery data

Returns pyspark dataframe object with columns: ["timestamp", "offset", "battery_level", "ver", "user"]

Return type DataFrame

gen_phone_battery_metadata() → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
Create Metadata object with some sample metadata of phone battery data

Returns metadata of phone battery stream

Return type *Metadata*

Module contents

Submodules

cerebralcortex.test_suite.join_spark module

cerebralcortex.test_suite.test_glucose_metrics module

class TestDataframeUDF(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_00()

cerebralcortex.test_suite.test_gps_cluster_udf module

class TestDataframeUDF(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_01_udf_on_gps()
Window datastream and perform a gps clustering udf on top of it

cerebralcortex.test_suite.test_import_data module

cerebralcortex.test_suite.test_main module

class TestCerebralCortex(*methodName='runTest'*)
Bases: unittest.case.TestCase, *cerebralcortex.test_suite.test_nosql_storage.NoSqlStorageTest*, *cerebralcortex.test_suite.test_sql_storage.SqlStorageTest*

setUp()
Setup test params to being testing with.

Notes

DO NOT CHANGE PARAMS DEFINED UNDER TEST-PARAMS! OTHERWISE TESTS WILL FAIL. These values are hardcoded in util/data_helper file as well.

test_00()
This test will create required entries in sql database.

cerebralcortex.test_suite.test_nosql_storage module

class NoSqlStorageTest
Bases: object

test_01_save_stream()
Test functionality related to save a stream

test_02_stream()

test_03_get_stream()
Test functionality related to get a stream

test_04_get_storage_path()

test_05_path_exist()

test_06_ls_dir()

test_07_create_dir()

test_08_write_pandas_to_parquet_file()

test_09_is_study()

test_10_is_stream()

test_11_get_stream_versions()

test_12_list_streams()

test_14_search_stream()

cerebralcortex.test_suite.test_rest_api_server module

cerebralcortex.test_suite.test_sql_storage module

class SqlStorageTest
Bases: object

test_00_save_stream_metadata()

test_01_get_stream_metadata_by_name()

test_02_list_streams()

test_03_search_stream()

test_04_get_stream_versions()

test_05_get_stream_metadata_hash()

test_06_get_stream_name()

test_07_get_stream_metadata_by_hash()

test_08_is_stream()

test_09_is_metadata_changed()

test_create_user()

test_get_user_id()

test_get_user_metadata()

test_get_user_settings()

test_get_username()

test_is_user()

test_list_users()

test_login_user()

cerebralcortex.test_suite.tt module

Module contents

cerebralcortex.util package

Submodules

cerebralcortex.util.helper_methods module

get_study_names(*configs_dir_path: str*) → List[str]
CerebralCortex constructor

Parameters **configs_dir_path** (*str*) – Directory path of cerebralcortex configurations.

Returns list of study names available

Return type list(str)

Raises ValueError – If configuration_filepath is None or empty.

Examples

```
>>> get_study_names("/directory/path/of/configs/")
```

Module contents

11.1.2 Submodules

11.1.3 cerebralcortex.kernel module

class Kernel(*configs_dir_path: str = ''*, *cc_configs: dict = None*, *study_name: str = 'default'*, *new_study: bool = False*, *enable_spark: bool = True*, *enable_spark_ui=False*)

Bases: `object`

connect(*username: str*, *password: str*, *encrypt_password: bool = False*) → dict

Authenticate a user based on username and password and return an auth token

Parameters

- **username** (*str*) – username of a user
- **password** (*str*) – password of a user
- **encrypt_password** (*str*) – is password encrypted or not. mCerebrum sends encrypted passwords

Raises `ValueError` – User name and password cannot be empty/None.

Returns return eturn {"status":bool, "auth_token": str, "msg": str}

Return type dict

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.connect("nasir_ali",
↪"2ksdfhoi2r2ljndf823hlkf8234hohwef0234hlkjwer98u234", True)
>>> True
```

create_user(*username: str*, *user_password: str*, *user_role: str*, *user_metadata: dict*, *user_settings: dict*, *encrypt_password: bool = False*) → bool

Create a user in SQL storage if it doesn't exist

Parameters

- **username** (*str*) – Only alphanumeric usernames are allowed with the max length of 25 chars.
- **user_password** (*str*) – no size limit on password
- **user_role** (*str*) – role of a user
- **user_metadata** (*dict*) – metadata of a user
- **user_settings** (*dict*) – user settings, mCerebrum configurations of a user
- **encrypt_password** (*bool*) – encrypt password if set to true

Returns True if user is successfully registered or throws any error in case of failure

Return type bool

Raises

- `ValueError` – if selected username is not available
- `Exception` – if sql query fails

encrypt_user_password(*user_password: str*) → str
Encrypt password

Parameters **user_password** (*str*) – unencrypted password

Raises `ValueError` – password cannot be None or empty.

Returns encrypted password

Return type str

gen_random_pass(*string_type: str = 'varchar'*, *size: int = 8*) → str
Generate a random password

Parameters

- **string_type** – Accepted parameters are "varchar" and "char". (Default="varchar")
- **size** – password length (default=8)

Returns random password

Return type str

get_stream(*stream_name: str*, *version: str = 'latest'*, *user_id: str = None*, *data_type=<DataSet.COMPLETE: (1,)>*) → cerebralcortex.core.datatypes.datastream.DataStream
Retrieve a data-stream with it's metadata.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")
- **data_type** (`DataSet`) – DataSet.COMPLETE returns both Data and Metadata. DataSet.ONLY_DATA returns only Data. DataSet.ONLY_METADATA returns only metadata of a stream. (Default=DataSet.COMPLETE)

Returns contains Data and/or metadata

Return type *DataStream*

Raises `ValueError` – if stream name is empty or None

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> ds = CC.get_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV-
↪-RIGHT_WRIST")
>>> ds.data # an object of a dataframe
>>> ds.metadata # an object of MetaData class
>>> ds.get_metadata(version=1) # get the specific version metadata of a stream
```

get_stream_metadata_by_hash(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns [stream_name, metadata]

Return type List

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_metadata_by_hash("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ["name" .....] # stream metadata and other information
```

get_stream_metadata_by_name(*stream_name: str*, *version: str = 1*) → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]

Get a list of metadata for all versions available for a stream.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")

Returns Returns an empty list if no metadata is available for a stream_name or a list of metadata otherwise.

Return type *Metadata*

Raises ValueError – stream_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_metadata_by_name("ACCELEROMETER--org.md2k.motionsense--
↪MOTION_SENSE_HRV--RIGHT_WRIST", version=1)
>>> Metadata # list of MetaData class objects
```

get_stream_metadata_hash(*stream_name: str*) → list

Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes with name and versions

Return type list

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_
↪SENSE_HRV--RIGHT_WRIST")
>>> [["stream_name", "version", "metadata_hash"]]
```

get_stream_name(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions(*stream_name: str*) → list

Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises ValueError – if stream_name is empty or None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

get_user_id(*user_name: str*) → str

Get the user id linked to user_name.

Parameters **user_name** (*str*) – username of a user

Returns user id associated to user_name

Return type str

Raises ValueError – User name is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_user_id("nasir_ali")
>>> '76cc444c-4fb8-776e-2872-9472b4e66b16'
```

get_user_metadata(*user_id: str = None*, *username: str = None*) → dict
Get user metadata by user_id or by username

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns user metadata

Return type dict

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_user_metadata(username="nasir_ali")
>>> {"study_name":"mperf"........}
```

get_user_name(*user_id: str*) → str
Get the user name linked to a user id.

Parameters **user_name** (*str*) – username of a user

Returns user_id associated to username

Return type bool

Raises `ValueError` – User ID is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_username("76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> 'nasir_ali'
```

get_user_settings(*username: str = None*, *auth_token: str = None*) → dict
Get user settings by auth-token or by username. These are user's mCerebrum settings

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – auth-token

Returns List of dictionaries of user metadata

Return type list[dict]

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_user_settings(username="nasir_ali")
>>> [{"mcerebrum":"some-conf"........}]
```

is_auth_token_valid(*username: str*, *auth_token: str*, *checktime: bool = False*) → bool

Validate whether a token is valid or expired based on the token expiry datetime stored in SQL

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – token generated by API-Server
- **checktime** (*bool*) – setting this to False will only check if the token is available in system. Setting this to true will check if the token is expired based on the token expiry date.

Raises `ValueError` – Auth token and auth-token expiry time cannot be null/empty.

Returns returns True if token is valid or False otherwise.

Return type bool

is_stream(*stream_name: str*) → bool

Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

is_user(*user_id: str = None*, *user_name: str = None*) → bool

Checks whether a user exists in the system. One of both parameters could be set to verify whether user exist.

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns True if a user exists in the system or False otherwise.

Return type bool

Raises `ValueError` – Both user_id and user_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.is_user(user_id="76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> True
```

list_streams() → List[str]
Get all the available stream names with metadata

Returns list of available streams metadata

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.list_streams()
```

list_users() → List[dict]
Get a list of all users part of a study.

Parameters **study_name** (*str*) – name of a study. If no study_name is provided then all users' list will be returned

Raises `ValueError` – Study name is a requied field.

Returns Returns empty list if there is no user associated to the study_name and/or study_name does not exist.

Return type list[dict]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.list_users()
>>> [{"76cc444c-4fb8-776e-2872-9472b4e66b16": "nasir_ali"}] # [{user_id, user_
↪name}]
```

read_csv(*file_path*, *stream_name: str*, *header: bool = False*, *delimiter: str = ', '*, *column_names: list = []*, *timestamp_column_index: int = 0*, *timein: str = 'milliseconds'*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*) → cerebralcortex.core.datatypes.datastream.DataStream
Reads a csv file (compressed or uncompressed), parse it, convert it into CC DataStream object format and returns it

Parameters

- **file_path** (*str*) – path of the file
- **stream_name** (*str*) – name of the stream
- **header** (*bool*) – set it to True if csv contains header column
- **delimiter** (*str*) – seprator used in csv file. Default is comma
- **column_names** (*list[str]*) – list of column names

- **timestamp_column_index** (*int*) – index of the timestamp column name
- **timein** (*str*) – if timestamp is epoch time, provide whether it is in milliseconds or seconds
- **metadata** (Metadata) – metadata object for the csv file

Returns DataStream object

save_stream(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*, *overwrite=False*) → bool

Saves datastream raw data in selected NoSQL storage and metadata in MySQL.

Parameters

- **datastream** (DataStream) – a DataStream object
- **overwrite** (*bool*) – if set to true, whole existing datastream data will be overwritten by new data

Returns True if stream is successfully stored or throws an exception

Return type bool

Raises Exception – log or throws exception if stream is not stored

Todo: Add functionality to store data in influxdb.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_stream(ds)
```

search_stream(*stream_name*)

Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

update_auth_token(*username: str*, *auth_token: str*, *auth_token_issued_time: datetime.datetime*, *auth_token_expiry_time: datetime.datetime*) → bool

Update an auth token in SQL database to keep user stay logged in. Auth token valid duration can be changed in configuration files.

Notes

This method is used by API-server to store newly created auth-token

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – issued new auth token
- **auth_token_issued_time** (*datetime*) – datetime when the old auth token was issue
- **auth_token_expiry_time** (*datetime*) – datetime when the token will get expired

Raises ValueError – Auth token and auth-token issue/expiry time cannot be None/empty.

Returns Returns True if the new auth token is set or False otherwise.

Return type bool

11.1.4 Module contents

class Kernel(*configs_dir_path: str = ''*, *cc_configs: dict = None*, *study_name: str = 'default'*, *new_study: bool = False*, *enable_spark: bool = True*, *enable_spark_ui=False*)

Bases: object

connect(*username: str*, *password: str*, *encrypt_password: bool = False*) → dict

Authenticate a user based on username and password and return an auth token

Parameters

- **username** (*str*) – username of a user
- **password** (*str*) – password of a user
- **encrypt_password** (*str*) – is password encrypted or not. mCerebrum sends encrypted passwords

Raises ValueError – User name and password cannot be empty/None.

Returns return eturn {"status":bool, "auth_token": str, "msg": str}

Return type dict

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.connect("nasir_ali",
↪"2ksdfhoi2r2ljndf823hlkf8234hohwef0234hlkjwer98u234", True)
>>> True
```

create_user(*username: str*, *user_password: str*, *user_role: str*, *user_metadata: dict*, *user_settings: dict*, *encrypt_password: bool = False*) → bool

Create a user in SQL storage if it doesn't exist

Parameters

- **username** (*str*) – Only alphanumeric usernames are allowed with the max length of 25 chars.
- **user_password** (*str*) – no size limit on password

- **user_role** (*str*) – role of a user
- **user_metadata** (*dict*) – metadata of a user
- **user_settings** (*dict*) – user settings, mCerebrum configurations of a user
- **encrypt_password** (*bool*) – encrypt password if set to true

Returns True if user is successfully registered or throws any error in case of failure

Return type bool

Raises

- `ValueError` – if selected username is not available
- `Exception` – if sql query fails

encrypt_user_password(*user_password: str*) → str

Encrypt password

Parameters **user_password** (*str*) – unencrypted password

Raises `ValueError` – password cannot be None or empty.

Returns encrypted password

Return type str

gen_random_pass(*string_type: str = 'varchar'*, *size: int = 8*) → str

Generate a random password

Parameters

- **string_type** – Accepted parameters are "varchar" and "char". (Default="varchar")
- **size** – password length (default=8)

Returns random password

Return type str

get_stream(*stream_name: str*, *version: str = 'latest'*, *user_id: str = None*, *data_type=<DataSet.COMPLETE: (1,)>*) → cerebralcortex.core.datatypes.datastream.DataStream

Retrieve a data-stream with it's metadata.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")
- **data_type** (`DataSet`) – DataSet.COMPLETE returns both Data and Metadata. DataSet.ONLY_DATA returns only Data. DataSet.ONLY_METADATA returns only metadata of a stream. (Default=DataSet.COMPLETE)

Returns contains Data and/or metadata

Return type *DataStream*

Raises `ValueError` – if stream name is empty or None

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> ds = CC.get_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV-
↪-RIGHT_WRIST")
>>> ds.data # an object of a dataframe
>>> ds.metadata # an object of MetaData class
>>> ds.get_metadata(version=1) # get the specific version metadata of a stream
```

get_stream_metadata_by_hash(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns [stream_name, metadata]

Return type List

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_metadata_by_hash("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ["name" .....] # stream metadata and other information
```

get_stream_metadata_by_name(*stream_name: str*, *version: str = 1*) → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]

Get a list of metadata for all versions available for a stream.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")

Returns Returns an empty list if no metadata is available for a stream_name or a list of metadata otherwise.

Return type *Metadata*

Raises ValueError – stream_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_metadata_by_name("ACCELEROMETER--org.md2k.motionsense--
↪MOTION_SENSE_HRV--RIGHT_WRIST", version=1)
>>> Metadata # list of MetaData class objects
```

get_stream_metadata_hash(*stream_name: str*) → list

Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes with name and versions

Return type list

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_
↪SENSE_HRV--RIGHT_WRIST")
>>> [["stream_name", "version", "metadata_hash"]]
```

get_stream_name(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions(*stream_name: str*) → list

Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises ValueError – if stream_name is empty or None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

get_user_id(*user_name: str*) → str

Get the user id linked to user_name.

Parameters **user_name** (*str*) – username of a user

Returns user id associated to user_name

Return type str

Raises ValueError – User name is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_user_id("nasir_ali")
>>> '76cc444c-4fb8-776e-2872-9472b4e66b16'
```

get_user_metadata(*user_id: str = None*, *username: str = None*) → dict
Get user metadata by user_id or by username

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns user metadata

Return type dict

Todo: Return list of User class object

Raises ValueError – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_user_metadata(username="nasir_ali")
>>> {"study_name":"mperf"........}
```

get_user_name(*user_id: str*) → str
Get the user name linked to a user id.

Parameters **user_name** (*str*) – username of a user

Returns user_id associated to username

Return type bool

Raises ValueError – User ID is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_username("76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> 'nasir_ali'
```

get_user_settings(*username: str = None*, *auth_token: str = None*) → dict
Get user settings by auth-token or by username. These are user's mCerebrum settings

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – auth-token

Returns List of dictionaries of user metadata

Return type list[dict]

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.get_user_settings(username="nasir_ali")
>>> [{"mcerebrum":"some-conf"........}]
```

is_auth_token_valid(*username: str*, *auth_token: str*, *checktime: bool = False*) → bool

Validate whether a token is valid or expired based on the token expiry datetime stored in SQL

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – token generated by API-Server
- **checktime** (*bool*) – setting this to False will only check if the token is available in system. Setting this to true will check if the token is expired based on the token expiry date.

Raises `ValueError` – Auth token and auth-token expiry time cannot be null/empty.

Returns returns True if token is valid or False otherwise.

Return type bool

is_stream(*stream_name: str*) → bool

Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

is_user(*user_id: str = None*, *user_name: str = None*) → bool

Checks whether a user exists in the system. One of both parameters could be set to verify whether user exist.

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns True if a user exists in the system or False otherwise.

Return type bool

Raises `ValueError` – Both user_id and user_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.is_user(user_id="76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> True
```

list_streams() → List[str]
Get all the available stream names with metadata

Returns list of available streams metadata

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.list_streams()
```

list_users() → List[dict]
Get a list of all users part of a study.

Parameters **study_name** (*str*) – name of a study. If no study_name is provided then all users' list will be returned

Raises `ValueError` – Study name is a requied field.

Returns Returns empty list if there is no user associated to the study_name and/or study_name does not exist.

Return type list[dict]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.list_users()
>>> [{"76cc444c-4fb8-776e-2872-9472b4e66b16": "nasir_ali"}] # [{user_id, user_
↪name}]
```

read_csv(*file_path*, *stream_name: str*, *header: bool = False*, *delimiter: str = ', '*, *column_names: list = []*, *timestamp_column_index: int = 0*, *timein: str = 'milliseconds'*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*) → cerebralcortex.core.datatypes.datastream.DataStream
Reads a csv file (compressed or uncompressed), parse it, convert it into CC DataStream object format and returns it

Parameters

- **file_path** (*str*) – path of the file
- **stream_name** (*str*) – name of the stream
- **header** (*bool*) – set it to True if csv contains header column
- **delimiter** (*str*) – seprator used in csv file. Default is comma
- **column_names** (*list[str]*) – list of column names

- **timestamp_column_index** (*int*) – index of the timestamp column name
- **timein** (*str*) – if timestamp is epoch time, provide whether it is in milliseconds or seconds
- **metadata** (Metadata) – metadata object for the csv file

Returns DataStream object

save_stream(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*, *overwrite=False*) → bool

Saves datastream raw data in selected NoSQL storage and metadata in MySQL.

Parameters

- **datastream** (DataStream) – a DataStream object
- **overwrite** (*bool*) – if set to true, whole existing datastream data will be overwritten by new data

Returns True if stream is successfully stored or throws an exception

Return type bool

Raises Exception – log or throws exception if stream is not stored

---

Todo: Add functionality to store data in influxdb.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_stream(ds)
```

search_stream(*stream_name*)

Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/", study_name="default")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

update_auth_token(*username: str*, *auth_token: str*, *auth_token_issued_time: datetime.datetime*, *auth_token_expiry_time: datetime.datetime*) → bool

Update an auth token in SQL database to keep user stay logged in. Auth token valid duration can be changed in configuration files.

Notes

This method is used by API-server to store newly created auth-token

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – issued new auth token
- **auth_token_issued_time** (*datetime*) – datetime when the old auth token was issue
- **auth_token_expiry_time** (*datetime*) – datetime when the token will get expired

Raises ValueError – Auth token and auth-token issue/expiry time cannot be None/empty.

Returns Returns True if the new auth token is set or False otherwise.

Return type bool

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

D

E

F

G

N

O

P

R

S

T

U

V

W