PCATR: powerful Python call center analysis toolkit

**Date**: Oct 28, 2019 **Version**: 0.0.2

**Useful links**: Source repository is private and cannot be shared to personnel other than authorized officials.

**PCATR** is a closed source, non-licensed Python library providing efficient, easy-to-use algorithms and data analysis tools for the prediction of call arrival times and rates.

See the Package overview for more detail about what’s in the library.

* [What’s new in 0.0.2 (Oct 30, 2019)](#new)
* [Installation](#installation)
* [Getting started](#start)
* [API reference](#api)
* [Release Notes](#notes)

What’s new in 0.0.2

Following are the additions to PCATR v0.0.2

**Bug fixes**

* Fixed TimeSeriesForecast.predict in PCATR.CallTimePredictor.CTPAlgorithm.time\_series\_forecast
* Fixed LstmForecast.fit in PCATR.CallTimePredictor.CTPAlgorithm.lstm\_forecast
* Fixed SeasonalForecast.fit in PCATR.CallTimePredictor.CTPAlgorithm.seasonal\_forecast
* Fixed SeasonalForecast.predict in PCATR.CallTimePredictor.CTPAlgorithm.seasonal\_forecast

**Modules**

* CallTimePredictor
* DataTank
* ValidationMetric
* Logger

**Contributors**

* Emad Bin Abid
* Ateeb Ahmed
* Syed Bilal Hoda

Installation

The easiest way to install pandas is to install it as part of the pip package manager. This is the recommended installation method for most users.

**Python version support**

* Officially Python 3.6.8 and above.

**Installing PCATR**

* pip install PCATR-0.0.2-py3-none-any.whl

**Dependencies**

|  |  |
| --- | --- |
| pandas | 0.24.2 |
| NumPy | 1.16.4 |
| Matplotlib | 3.1.1 |
| sci-kit learn | 0.18.1 |
| seaborn | 0.7.1 |
| statsmodels | 0.10.1 |
| torch | 1.1.0 |

Getting started

**Package overview**

* Development team
* Emad Bin Abid, Ateeb Ahmed, Syed Bilal Hoda
* Syeda Saleha Raza, Zeehasham Rasheed
* Data structures
* pandas’ Series
* pandas’ DataFrame
* Getting support
* [emad.bin.abid@gmail.com](mailto:emad.bin.abid@gmail.com)
* [sh02918@st.habib.edu.pk](mailto:sh02918@st.habib.edu.pk)
* [aa02751@st.habib.edu.pk](mailto:aa02751@st.habib.edu.pk)
* Institutional partners
* Habib University
* Afiniti Software Solutions (Pvt.) Ltd.

API reference

|  |  |
| --- | --- |
| CallTimePredictor   * [CTPAlgorithm](#ctpalgorithm) * [CTPDataAnalysis](#ctpdataanalysis) | This module implements algorithms to predict the time (in seconds) after which the next call can be expected. |
| [DataTank](#datatank) | This module implements algorithms to manipulate data which is used by other modules for prediction purposes. |
| [ValidationMetric](#validationmetric) | This module implements algorithms and metrics to validate results of trained models on test samples. |
| [Logger](#logger) | This module implements functions to deal with efficient logging mechanisms in PCATR where necessary. |

PCATR.CallTimePredictor.CTPAlgorithm

Following is the list of prediction algorithms which the module implements.

|  |  |
| --- | --- |
| [simple\_average\_forecast](#simple_average_forecast) | Computes the average interval difference of the complete dataset and uses it to predict the time left until the next immediate call. |
| [interday\_average\_forecast](#interday_average_forecast) | Computes a set of average values and each value is associated with a specific day. As input, the model takes a time and day, determines its associated average value and predicts the time until the next call. |
| [halfday\_interval\_average\_forecast](#halfday_interval_average_forecast) | Computes a set of average values where each value is associated with a specific day and the first or second half of the day. It distinguishes between the two halves of the day and determines the time until the next call. |
| [hourly\_interval\_average\_forecast](#hourly_interval_average_forecast) | Computes a set of average values and each value is associated with a specific day and a specific hour gap. It distinguishes between hours and determines associated average value and predicts the time until the next call. |
| [smoothing\_forecast](#smoothing_forecast) | Predicts future values using weighted averages, where the weights decrease exponentially as observations come from further in the past – the smallest weights are associated with the oldest observations. |
| [time\_series\_forecast](#time_series_forecast) | Computes future values using ARIMA modeling. Past time points of time series data can impact current and future time points. ARIMA models take this concept into account when forecasting current and future values. ARIMA uses a number of lagged observations of time series to forecast observations. |
| [double\_smoothing\_forecast](#double_smoothing_forecast) | Computes/predicts forecasts upon calculating weighted averages of past individual observations and a weighted average of the estimated trend at a respective time. |
| [seasonal\_forecast](#seasonal_forecast) | Computes/predicts forecasts upon calculating weighted averages of past individual observations and a weighted average of the estimated trend at a respective time by additionally capturing seasonality components. |
| [lstm\_forecast](#lstm_forecast) | Computes the prediction results using a deep learning technique. It uses LSTM network (special kind of recurrent neural network) with sequence length of over 100. |
| [poisson\_forecast](#poisson_forecast) |  |

PCATR.CallTimePredictor.CTPAlgorithm.simple\_average\_forecast

Computes the average interval difference of the complete dataset and uses it to predict the time left until the next immediate call.

Using the simple average forecast, the future forecasts are all equal to the average of all of the interval difference in the complete dataset. It grants equal importance to all observations and gives them equal weights when generating forecasts.

**Benefits:**

* Summarize the large amount of data into a single value
* Allows us to check on the variability within the data using the mean as a reference

**Limitations:**

The dataset contains value that are randomly spread and interval differences, sometimes being random, can also be quite high or low which can mess up the average and provide an inaccurate average values.

*class* PCATR.CallTimePredictor.CTPAlgorithm.simple\_average\_forecast SimpleAverageForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit simple average forecasting model |
| predict (self, testData) | Predict using simple averaging forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.interday\_average\_forecast

Computes a set of average values and each value is associated with a specific day. As input, the model takes a time and day, determines its associated average value and predicts the time until the next call.

The sample dataset is categorized under the months and further, classified across the days.

The average value Interval difference is computed with respect to each day. For example:

Set of Average Values = {‘**Monday’**:20.05, ‘**Tuesday**’:30.98,’**Wednesday**’:15.22 …}

The value associated with Monday is the average of all interval differences spanning each Monday throughout the 3 months in the sample dataset.

Similar to the mathematical formula for the simple averaging model, the mathematical formula for this model is:

**Benefits:**

* By obtaining values under a specified formulation of the dataset, interday averaging models allows us to perform more detailed exploratory data analysis including the possibility of finding a trend across the days.

**Limitations:**

There is no guarantee that Interval differences across each Monday will be similar to other Monday. Since, individual interval differences can vary greatly, then the averages across each day will also have great variability.

*class* PCATR.CallTimePredictor.CTPAlgorithm.interday\_average\_forecast. InterdayAverageForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit interday average forecasting model |
| predict (self, testData) | Predict using interday averaging forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.halfday\_interval\_average\_forecast

Computes a set of average values where each value is associated with a specific day and the first or second half of the day. It distinguishes between the two halves of the day and determines the time until the next call.

The sample dataset is categorized under months, days, and then further processed to establish one more category which categorizes times as belonging to the first half or the second half of the day.

The average interval difference is computed for each day and the respective half of day. For example:

Set of Average Values = {‘**Monday**’: [[0, 20.05], [1, 15.11]] …}

From the above illustration, we can understand that the average interval difference for the first half of the Monday, spanning across all Mondays in the dataset, is 20.05 seconds whereas for the second half of the day is 15.11 seconds.

Similar to the mathematical formula for the Simple Averaging Model, the mathematical formula for this model is:

*class* PCATR.CallTimePredictor.CTPAlgorithm.halfday\_interval\_average\_forecast. HalfdayIntervalAverageForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit halfday interval average forecasting model |
| predict (self, testData) | Predict using halfday interval averaging forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.hourly\_interval\_average\_forecast

Computes a set of average values and each value is associated with a specific day and a specific hour gap. It distinguishes between hours and determines associated average value and predicts the time until the next call.

The sample dataset is categorized under months, days, and then hours. The average value interval difference is computed with respect to each day and each hour.

Hence, we can say that the average interval difference on Friday between 7 and 8, computed across every Friday is specific, **say 21.505**.

Similar to the mathematical formula for the simple averaging model, the mathematical formula for this model is:

*class* PCATR.CallTimePredictor.CTPAlgorithm.hourly\_interval\_average\_forecast. HourlyIntervalAverageForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit hourly interval average forecasting model |
| predict (self, testData) | Predict using hourly interval averaging forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.smoothing\_forecast

Predicts future values using weighted averages, where the weights decrease exponentially as observations come from further in the past – the smallest weights are associated with the oldest observations.

The naïve method forecasts for the future are all equal to the last observed values in the series whereas in the average method all future forecasts are equal to a simple of the averaged data. **Simple Exponential Smoothing** lies in between of the two. It attaches larger weights to more recent observations than to observations in the distant part.

**Where**  is the smoothing parameter. Therefore, a forecast is a weighted average of all the observations in the series. The rate at which the weights decrease is controlled by the parameter .

**Benefits:**

* It gives more weight to recent observations
* It allows exploration of stationary series

**Limitations:**

* Stationary series may be accompanied by trend or seasonality. SES does not take account either of those

*class* PCATR.CallTimePredictor.CTPAlgorithm.smoothing\_forecast. SmoothingForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit smoothing forecasting model |
| predict (self, testData) | Predict using smoothing forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.time\_series\_forecast

Computes future values using ARIMA modeling. Past time points of time series data can impact current and future time points. ARIMA models take this concept into account when forecasting current and future values. ARIMA uses a number of lagged observations of time series to forecast observations.

**Benefits:**

* ARIMA model has a fixed structure and is specifically built for time series (sequential) data. If the data is generated by a process similar to ARIMA assumptions then it works well. I have observed it to work well some cases but not so well in forecasting financial markets.

*class* PCATR.CallTimePredictor.CTPAlgorithm.time\_series\_forecast. TimeSeriesForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData, testData) | Fit time series forecasting model |
| predict (self, testData) | Predict using time series forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe   **testData**: *DataFrame*  Test data / Samples – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.double\_smoothing\_forecast

Computes forecasts upon calculating weighted averages of past individual observations and a weighted average of the estimated trend at a respective time.

This uses Holt’s Linear Trend method internally which itself extends simple exponential smoothing to allow the forecasting of data with trend. This method involves a forecast equation and two smoothing equation as shown below:

**Forecast Equation:**

**Level Equation:**

**Trend Equation:**

Where denotes an estimate of the level series at time t, denotes an estimate of the trend (slope) of the series at time t,  α is the smoothing parameter for the level, 0≤α≤1 and β∗ is the smoothing parameter for the trend, 0≤β∗≤1.

**Benefits:**

* Holt’s Linear trend takes into account the linear trend in series

*class* PCATR.CallTimePredictor.CTPAlgorithm.double\_smoothing\_forecast. DoubleSmoothingForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit double smoothing forecasting model |
| predict (self, testData) | Predict using double smoothing forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.seasonal\_forecast

Computes forecasts upon calculating weighted averages of past individual observations and a weighted average of the estimated trend at a respective time by additionally capturing seasonality components.

The method uses Holt-Winters method and combines a total of 4 equations, one being the typical forecast equation while the other three are smoothening equations where one equation is for the level, one for the trend and the last one is for the seasonal component. The three corresponding smoothening parameters are α, β and and **m** is used to denote the seasonality, that is, the number of seasons in a year.

**Benefits:**

* Captures both the trend and the seasonality, thereby allowing for the prediction of more accurate values based on the respective conditions.

*class* PCATR.CallTimePredictor.CTPAlgorithm.seasonal\_forecast. SeasonalForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit seasonal forecasting model |
| predict (self, testData) | Predict using seasonal forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.lstm\_forecast

Computes the prediction results using a deep learning technique. It uses LSTM network (special kind of recurrent neural network) with sequence length of over 100.

**Benefits:**

* LSTM captures both long term and short term effects
* Network hyper parameters can be changed to observe significant changes in loss value

**Limitations:**

* Requires significant amount of GPU memory for training

*class* PCATR.CallTimePredictor.CTPAlgorithm.lstm\_forecast. LstmForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit LSTM forecasting model |
| predict (self, testData) | Predict using LSTM forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPAlgorithm.poisson\_forecast

*class* PCATR.CallTimePredictor.CTPAlgorithm.poisson\_forecast. PoissonForecast ()

**Methods:**

|  |  |
| --- | --- |
| fit (self, trainData) | Fit Poisson forecasting model |
| predict (self, testData) | Predict using Poisson forecasting model |
| showPlot (self) | Displays the plot of train data, test data and predicted results |

fit (self, trainData)

|  |  |
| --- | --- |
| **Parameters:** | **trainData**: *DataFrame*   * Training data – full dataframe |
| **Returns:** | **self**: returns an instance of self |

predict (self, testData)

|  |  |
| --- | --- |
| **Parameters:** | **testData**: *DataFrame*   * Test data / Samples – full dataframe |
| **Returns:** | **DataFrame**: returns a dataframe of predicted values |

showPlot (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **None** |

PCATR.CallTimePredictor.CTPDataAnalysis

Following is the list of data analysis algorithms which the modules implements.

|  |  |
| --- | --- |
| [eda](#eda) | Extracts sufficient information from the dataset and computes the necessary exploratory data analysis features and results for meaningful insights. |

PCATR.CallTimePredictor.CTPDataAnalysis.eda

This module extracts sufficient information from the dataset and computes the necessary exploratory data analysis features and results for meaningful insights.

*class* PCATR.CallTimePredictor.CTPDataAnalysis.eda. EDA (dataframe)

**Methods:**

|  |  |
| --- | --- |
| callDays (self) | Gives the names of days at which the full dataset is spread |
| callTimeAndCallDifferenceInterval (self) | Gives the call arrival time and call difference interval for each individual entry in full dataset |
| eachDayCallCount (self, showPlot=False) | Gives the count of calls for each individual day in the dataset |
| eachDayCallCountDescription (self) | Gives different summary results on the count of calls for each individual day in the dataset |
| eachDayIntervalsCallCount (self, showPlot=False) | Gives the count of calls for *morning, afternoon, evening, night* intervals for each individual day in the dataset |
| interdayCallCount (self) | Gives the count of calls for each weekday grouped over full dataset |
| maxCallDifferenceInterval (self) | Gives the maximum of call difference interval from full dataset |
| maxCallTime (self) | Gives the maximum call arrival time from full dataset |
| meanCallCount (self) | Gives the mean of call count for on full dataset |
| minCallDifferenceInterval (self) | Gives the minimum of call difference interval from full dataset |
| minCallTime (self) | Gives the maximum call arrival time from full dataset |

callDays (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **array<string>**: returns an array containing unique days in the dataset |

callTimeAndCallDifferenceInterval (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** |  |

eachDayCallCount (self, showPlot=False)

|  |  |
| --- | --- |
| **Parameters:** | **showPlot**: *Boolean* |
| **Returns:** | **DataFrame**: returns a DataFrame object consisting the count of calls for each individual day in the dataset |

eachDayCallCountDescription (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **DataFrame**: returns a DataFrame object consisting of description of the count of calls for each individual day in the dataset |

eachDayIntervalsCallCount (self, showPlot=False)

|  |  |
| --- | --- |
| **Parameters:** | **showPlot**: *Boolean* |
| **Returns:** | **DataFrame**: returns a DataFrame object with ‘CallArrivalDate’, ‘IntervalOfDay’ and column of count |

interdayCallCount (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **DataFrame**: returns DataFrame object with ‘DayOfWeek’ and column of count |

maxCallDifferenceInterval (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** |  |

maxCallTime (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **DataFrame**: returns a DataFrame object with ‘CallArrivalDate’ and max call arrival time |

meanCallCount (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **int**: returns mean of call count on full dataset |

minCallDifferenceInterval (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** |  |

minCallTime (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **DataFrame**: returns a DataFrame object with ‘CallArrivalDate’ and min call arrival time |

PCATR.DataTank.data\_tank

Following is the list of data manipulation functionalities which the module implements.

|  |  |
| --- | --- |
| [DataTank](#datatank_datatank) | This module implements algorithms to manipulate data which is used by other modules for prediction purposes. |

PCATR.DataTank.data\_tank.DataTank

This module implements algorithms to manipulate data which is used by other modules for prediction purposes.

*class* PCATR.DataTank.data\_tank. DataTank ()

**Methods:**

|  |  |
| --- | --- |
| loadData (self, filename) | Loads the data in CSV file using Python’s pandas module. |
| getProcessedData (self) | Processes the data and adds a new column ‘CallDifferenceInterval’ to DataFrame object. |
| trainTestSplit (self, splitRatio=0.66) | Splits the data in DataFrame object into train and test. |

loadData (self, filename)

|  |  |
| --- | --- |
| **Parameters:** | **filename**: *string*   * Name of CSV file |
| **Returns:** | **DataFrame:** returns loaded data wrapped in pandas’ DataFrame object |

getProcessedData (self)

|  |  |
| --- | --- |
| **Parameters:** |  |
| **Returns:** | **DataFrame:** returns processed data wrapped in pandas’ DataFrame object |

trainTestSplit (self, splitRatio=0.66)

|  |  |
| --- | --- |
| **Parameters:** | **splitRatio**: *integer*   * Ratio of the train-test split |
| **Returns:** | **DataFrame:** returns DataFrame objects of train and test splits |

PCATR.ValidationMetric.validation\_metric

Following is the list of validation functionalities which the module implements.

|  |  |
| --- | --- |
| ValidationMetric | This module implements algorithms to validate results which are generated by trained models. |

PCATR.ValidationMetric.validation\_metric.ValidationMetric

This module implements algorithms to **validate results** **which are generated by trained models.**

*class* PCATR.ValidationMetric.validation\_metric. ValidationMetric ()

**Methods:**

|  |  |
| --- | --- |
| meanSquaredError (self, actual, predicted) | Finds the mean squared error between actual and predicted values |

meanSquaredError (self, actual, predicted)

|  |  |
| --- | --- |
| **Parameters:** | **actual**: *Dataframe* column   * Actual values   **predicted**: *Dataframe* column   * Predicted values |
| **Returns:** | **int:** returns MSE score |

PCATR.Logger.logger

Following is the list of logging functionalities which the module implements.

|  |  |
| --- | --- |
| [Logger](#logger_logger) | This module implements functions to deal with efficient logging mechanisms in PCATR where necessary. |

PCATR.Logger.logger.Logger

This module implements functions to deal with efficient logging mechanisms in PCATR where necessary.

*class* PCATR.Logger.logger. Logger ()

**Methods:**

|  |  |
| --- | --- |
| LOGERROR (filename, functionName, errorSummary, errorObject=None) | Logs the error occurrences for PCATR. |
| LOGDEBUG (filename, functionName, debugSummary) | Logs the success/debug/analytical occurrences for PCATR. |
| LOGINFO (filename, functionName, infoSummary) | Logs the general information occurrences for PCATR. |

LOGERROR (filename, functionName, errorSummary, errorObject=None)

|  |  |
| --- | --- |
| **Parameters:** | **filename**: *string*   * Name of file in which the function is called   **functionName**: *string*   * Name of function in which the function is called   **errorSummary**: *string*   * A short error description   **errorObject**: *Exception*   * Full error with stack trace |
| **Returns:** | **None** |

LOGDEBUG (filename, functionName, debugSummary)

|  |  |
| --- | --- |
| **Parameters:** | **filename**: *string*   * Name of file in which the function is called   **functionName**: *string*   * Name of function in which the function is called   **debugSummary**: *string*   * A short debug description |
| **Returns:** | **None** |

LOGINFO (filename, functionName, infoSummary)

|  |  |
| --- | --- |
| **Parameters:** | **filename**: *string*   * Name of file in which the function is called   **functionName**: *string*   * Name of function in which the function is called   **infoSummary**: *string*   * A short info description |
| **Returns:** | **None** |

Release Notes

This is the list of changes to PCATR between each release. For install and upgrade instructions, see [Installation](#installation).

Version 0.0.2

* [What’s new in 0.0.2 (October 30, 2019)](#new)
* [Bug fixes](#bug_fixes)
* [Future directions](#future_directions)

Bug fixes

* Fixed TimeSeriesForecast.predict in PCATR.CallTimePredictor.CTPAlgorithm.time\_series\_forecast
* Fixed LstmForecast.fit in PCATR.CallTimePredictor.CTPAlgorithm.lstm\_forecast
* Fixed SeasonalForecast.fit in PCATR.CallTimePredictor.CTPAlgorithm.seasonal\_forecast
* Fixed SeasonalForecast.predict in PCATR.CallTimePredictor.CTPAlgorithm.seasonal\_forecast

Future directions

Following is the list of directions and features which are planned for future releases.

* Further exploration and development of new algorithms of Poisson method.
* Further exploration and development of new algorithms of recurrent neural networks.
* Further exploration of data analysis including EDA.
* Further exploration of validation metrics.
* Further exploration of benefits and limitations of existing methods and approaches.
* Bug fixes (if any).
* Better visualization mechanisms.