`timeline` Documentation

Manuel Haid Thomas Mauerhofer Matthias Wölbitsch

June 12, 2016

Contents

1 Introduction

2 Requirements

- >= Python 3.5

3 Dependencies

This section contains a description of all used external dependencies (e.g. libraries) and why they were used in this project. The python dependencies can be installed using `pip`, the Python package manager, using the `requirements.txt` file. It is recommended to use a virtual environment when developing for this project to avoid conflicts with other system-wide installed versions of the same dependencies.

3.1 Back-end (Python)

Flask Flask is a microframework (simple but extendable) for web development. It allows RESTful request dispatching and is well documented. In this project Flask is used for the rendering of the front-end using its template engine and providing an API for the supported visualizations.

Pandas Pandas is a widely used data analysis library. It provides data structures to efficiently store and query in-memory data. All temporal data (i.e. all time series) in this project is stored in Pandas DataFrame or Series objects.

NumPy NumPy is a powerful package for scientific computing. It provides efficient data structures and algorithms to operate on them. However, in this project this library is only used to sample random data for the example data sets and is therefore not very important.

3.2 Front-end (HTML5)

jQuery jQuery is the feature rich JavaScript library to simplify client-side scripting. In this project it is mainly used for AJAX calls.

MetricsGraphics.js MetricsGraphics.js is used to render the time series visualizations. It is based on the well-known D3.js graphic library and provides all necessary primitives and features to plot the required figures.

Bootstrap This CSS framework is used to for its beautiful design templates to provide a familiar and responsive user interface. Furthermore, some bootstrap add-ons were used for additional widgets (e.g. the bootstrap datetime picker).

4 Modules

4.1 APP

This module simply creates the Flask object and resisters the routes for the front-end and for the API. The Flask object must be used to start timeline using the `run` method of the app object. The source code is located in the `app.py` file.

4.2 API

The task of this module is to take requests from the front-end in form of AJAX calls and return the visualization primitives for the MetricsGraphics.js library as JSON objects. The source code is located in the `api.py` file.

Time Series Plot: `api/time_plot/{time_series_id}`

Method: GET

Description: This route create the visualization primitive for a simple time plot (i.e. a time plot of only one time series). It takes the identifier of the time series as part of the URL (e.g. `api/time_plot/42`) and can take additional parameter. It may return a HTTP status code 404 if no time series with the given identifier exists and a HTTP status code 400 for invalid values for the optional GET parameter.

Additional Parameter: All of the following parameter are optional. It is possible to only retrieve a slice of the time series using the `start_date` and/or `end_date` parameter.

Each of this parameter must be an integer, which represents to UNIX time stamp of the start time of the slice and the end time of the slice, respectively. Additional it is possible to add the graphs of the rolling mean and/or the rolling standard deviation to the plot. The `rolling_mean_window` and `rolling_std_window` parameter must be integer that represent the number of samples that is used to calculate the running mean and standard deviation, respectively.

Time Series Plots: `api/time_plots`

Method: GET

Parameter: list of time series ids

Description:

Live Time Series Plot: `api/live_plot/{live_time_series_id}`

Method: GET

Parameter: last received

Description:

ACF Plot: `api/acf_plot/{time_series_id}`

Method: GET

Parameter: max lag, scale

Description:

Forecasting Plot: `api/forecasting_plot/{forecast_id}`

Method: GET

Parameter: none

Description: