`timeline` Documentation

Manuel Haid Thomas Mauerhofer Matthias Wölbitsch

June 17, 2016

Contents

1 Introduction

In this section we want to explain how to start a example that uses the `timeline` platform using vagrant.

1. Start vagrant using the `vagrant up` command.
2. Get a cup of coffee, because the first startup requires the installation of some libraries from source and this may take a while.
3. After that you can access the example from outside the vagrant box using port 5000 (i.e., open `http://localhost:5000/`).

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 identifier
>
> **Description:** This route creates the visualization primitive for a multi time series plot (i.e. a time plot of multiple time series). It may return a HTTP status code 404 if one or more time series are not found in the data storage.

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

> **Method:** GET
>
> **Parameter:** last received
>
> **Description:** This route create the visualization primitive for a simple live time plot. The first request to this route must not contain the `last_received` parameter, which is a UNIX time stamp. All other requests on the same URL should contain this parameter to only obtain a update on the time series data. It may return a HTTP status code 404 if no time series with the given identifier exists.

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

> **Method:** GET
>
> **Parameter:** max lag, scale
>
> **Description:** This route creates the visualization primitive for a autocorrelation function plot. 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. The `max_lag` parameter is required. It determines the number of autocorrelations offsets that should be included into the plot. The parameter must be an integer. The second parameter `scale` is optional, if it is null, the y-axis of the plot will be scaled automatically.

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

> **Method:** GET
>
> **Parameter:** none
>
> **Description:** This route create the visualization primitive for a simple forecasting plot. It may return a HTTP status code 404 if no time series with the given identifier exists.

4.3 DATA SETS

The task of this module is, to define the classes that describe the data sets for the time series, live data and forecast data. It also has functions to add and get the data sets from the from the storage. The source code is located in the `data_sets.py` file.

Class TimeSeries:

> **Description:** The member variables from the class TimeSeries are the auto-generated id, a name, a description, the data and optional the legend. It also has functions to get information like the number of samples, the start date, the end date and the period.

Class LiveTimeSeries:

> **Description:** These class is derived from the class TimeSeries. It only has additional functions to get and to update the data.

Class TimeSeriesForecast:

> **Description:** These class is also derived from the class TimeSeries. It only has additional member variables like the training data, the forecasted data, the test data and the validation split.

Register Data Sets:

> **Description:** In this section, we have three functions to add the data sets to the `time_series`, `live_time_series` or `forecast` dictionaries for storage.

Get Data Sets:

> **Description:** In this section, we have different functions to get the data sets (`time_series`, `live_time_series` or `forecast`). There are functions to get one data set (with an id) or you can get all data sets.

4.4 FRONT END

The task of this module is to render the different HTML templates of the user interface (for example, the info.html, index.html, multi_time_plots.html, live_info.html, etc).

4.5 VISUALIZATION

The task of this module is to build the visualization objects so that the visualization library in the front end can render it. The source code is located in the `visualizations.py` file.

Time Series Plot:

> **Description:** This function generates the visualization object for a time series plots, with the optional parameter start and end date. An optional feature is, that if the data set is a list, multiple data sets will be used, otherwise only one data set will be used. Afterwards the title will be set and we return a object with a title, the x accessor, the date, the y accessor, the value, the data and the legend.

Add Rolling Mean:

> **Description:** This function adds a rolling mean line to the existing time series plot. We calculate the rolling mean and add the data object with the data and the legend to the existing data set.

Add Rolling Std:

> **Description:** This function adds a rolling standard deviation to the existing time series plot. We calculate the rolling standard deviation and add the data object with the data and the legend to the existing data set.

Auto Correlation Plot:

> **Description:** This function generates an auto correlation visualization primitive for an existing data set. The `max_lag` parameter describes the max number of correlation steps.

Forecasting Eval Plot:

> **Description:** This function generates the visualization primitive for the forecast evaluation plot.

Build Data Object:

> **Description:** This function converts the time series data to the right format, so that the visualization library can use it.