kpireport

Release 0.1.8

KPI Reporter LLC

Oct 06, 2022

GETTING STARTED

KPI Reporter is a **dev-friendly**, **on-premises** tool for for crafting automated reports.

Support for a variety of reporting sources is built-in, including, e.g., *MySQL databases*, *Prometheus metrics*, and the *Jenkins API*. Reports can be sent via email (with plugins for *SMTP* and *SendGrid*), *Slack*, or simply rendered as *HTML*.

Note: **Author note**: I created this tool after being surprised that there was a distinct lack of simple developer-friendly reporting tools for communicating business and/or operational KPIs. I hope you find it useful as well. For more information, see *Motivation*.

CHAPTER

ONE

INSTALLATION

KPI Reporter is a Python module that is installable via `pip`:

```
pip install kpireport
```

1.1 Docker

A Docker image is available on DockerHub with all dependencies required by all available plugins.

1.2 Usage

Invoking the installed bin script without any arguments will default to generating a report over a window ending at the current date and starting at one week ago. To specify different windows, use the `--start-date` and `--end-date` options.

```
# Generate report over last 7 days (default)
kpireport --config-file my-report.yaml

# Generate report from last week
kpireport --config-file my-report.yaml \
  --start-date $(date +%Y-%m-%d -d'-2 week') \
  --end-date $(date +%Y-%m-%d -d'-1 week')
```

If you do not specify a `--config-file` option, the tool will attempt to find a configuration in the following locations (in order):

1. `./config.yaml`
2. `/etc/kpireporter/config.yaml`

If using the *Docker image*, the configuration file can be mounted in to one of these locations:

```
docker run --rm -v my-config.yaml:/etc/kpireporter/config.yaml \
  kpireporter/kpireporter:edge
```

1.2.1 Installing licenses

Important: Your license file should be kept secret! If you post your license file online or in a source code repository, anyone could steal your license. If you would like to request a new license in case of compromise, you can send an email here.

By default, KPI Reporter looks for a license files (ending in `.pem` or `.key`) in `/etc/kpireporter`. The *last file found* is used. This allows you to name your license files by date if you want.

```
mv path/to/license.pem /etc/kpireporter/
```

If using the *Docker image*, you can mount the license file inside the container:

```
docker run --rm -v license.pem:/etc/kpireporter/license.pem:ro \
  kpireporter/kpireporter:
```

You can also use the `--license-file` flag to load the license from a different location.

```
kpireport --license-file path/to/license.pem [...args]
```

1.3 Plugins

If you are not using the distributed Docker image, and are installing KPI Reporter via pip, you will have to install some small set of additional plugins to get started. Two simple plugins you may want are the *Plot* and *Static file* ones.

Plugins provided as part of KPI Reporter project are prefixed `kpireport-`, and so are installed like the following:

```
# Install KPI reporter with MySQL, Prometheus and SendGrid plugins
pip install \
  kpireport \
  kpireport-sql \
  kpireport-prometheus \
  kpireport-sendgrid
```

Note: It is possible to install all available plugins via the `all` extra:

```
pip install kpireport[all]
```

In practice due to how pip handles (or doesn't handle) cross-dependencies this can be tricky. It may be better to install some "core" plugins first before attempting:

```
pip install kpireport kpireport-static && pip install kpireport[all]
```

CHAPTER

TWO

CONFIGURATION FILE

A report is declared entirely within a YAML file consisting of a few main sections: `datasources`, `views`, and `outputs`. In each section, you can declare as many plugin instances as you wish (to e.g., declare multiple database *Datasources* or multiple *Plot* visualizations in your report). As dictated by the YAML spec, duplicate keys (IDs) are not allowed; ensure that each plugin instance has its own ID unique to its section.

Note: It is possible to specify multiple configuration files when generating a report. In this case, the configurations are merged together, with the last file taking priority:

```
kpireport -c base-config.yaml -c extra-config.yaml
```

2.1 A full example

For more examples see *Examples*.

```yaml
---
title: My report

datasources:
  db:
    plugin: mysql
    args:
      host: 127.0.0.1
  jenkins:
    plugin: jenkins
    args:
      host: 127.0.0.1:8000

views:
  line_plot:
    title: A line plot
    plugin: plot
    args:
      # Note that this refers to the "db" datasource registered above.
      datasource: db
      query: |
        select time, value from my_db.my_table
        where time > {from} and time < {to}
```

(continues on next page)

(continued from previous page)

```
  results_table:
    title: A table of values
    plugin: table
    args:
      # Note that this refers to the "db" datasource registered above.
      datasource: db
      query: |
        select count(value), value from my_db.my_table
        where time > {from} and time < {to}
        group by value

outputs:
  mail:
    plugin: smtp
    args:
      smtp_host: 127.0.0.1
      smtp_port: 1025
```

2.2 Schema

2.2.1 Configuration file

https://kpi-reporter.readthedocs.io/page/configuration.schema.json		
type	*object*	
properties		
• title	Title of the report.	
	type	*string*
• interval_days	The report interval in days. This value is only used if start_date is not explicitly defined.	
	type	*integer*
	default	7
• start_date	Beginning of reporting period.	
	type	*string*
	default	End date minus the number of dates in interval_days.
	format	date
• end_date	End of reporting period.	
	type	*string*
	default	Current date.
	format	date
• theme	*theme*	
• datasources	Set of named datasource instances; the keys define the IDs.	
	type	*object*
	additionalProperties	*datasource*
• views	Set of named view instances; the keys define the IDs.	
	type	*object*
	additionalProperties	*view*
• outputs	Set of named output driver instances; the keys define the IDs.	
	type	*object*
	additionalProperties	*output*
additionalProperties	False	

theme

type	*object*	
properties		
• num_columns	Number of columns in view grid layout.	
	type	*integer*
	default	6
• column_width	Width of single column in view grid layout, in pixels.	
	type	*integer*
	default	86
• theme_dir	Path to directory with additional theme assets.	
	type	*string*
additionalProperties	False	

datasource

type	*object*	
properties		
• **plugin**	Name of datasource plugin to use.	
	type	*string*
• args	type	*object*
	additionalProperties	True
additionalProperties	False	

view

type	*object*	
properties		
• **plugin**	Name of view plugin to use.	
	type	*string*
• args	type	*object*
	additionalProperties	True
• title	Optional headline title for the view.	
	type	*string*
• description	Optional description for the view.	
	type	*string*
• cols	Number of grid columns to take up.	
	type	*integer*
	default	Total number of columns defined in theme (full bleed).
additionalProperties	False	

output

type	*object*	
properties		
• **plugin**	Name of output driver plugin to use.	
	type	*string*
• args	type	*object*
	additionalProperties	True
additionalProperties	False	

2.3 View instances

The View instances defined in the `views` section define what is rendered in the final report. Each view is placed into a layout in the order in which they are defined, i.e., the first declared View will show at the top, and the last will show at the bottom.

The report layout follows a simple grid system with 6 columns. By default, Views will each take the full width. However, you can change this with the `cols` configuration option. For example, consider this configuration:

```yaml
views:
  view_a:
  view_b:
    cols: 2
  view_c:
    cols: 4
  view_d:
  view_e:
    cols: 3
  view_f:
    cols: 3
```

The views would be rendered in the report like this:

View A	
View B	View C
View D	
View E	View F

CHAPTER

THREE

EXAMPLES

3.1 Top-of-funnel report

View HTML

This example utilizes a MySQL Datasource and multiple Plot visualizations to show a high-level overview of the number of new signups over the last week (both as an running total and as a count of new signups per day).

```
---
title: Top-of-funnel report

datasources:
  db:
    plugin: mysql
    args:
      host: mysql
      user: kpireport
      passwd: kpireport_pass

views:
  # Show a time series of user signups over the report period.
  signups_over_time:
    plugin: plot
    title: Total sign-ups
    description: |
      The running total of user accounts.
    args:
      datasource: db
      query: |
        select created_at,
        count(*) over (order by created_at)
          + (select count(*) from kpireport.signups where created_at < {from})
          as total_signups
        from kpireport.signups
        where created_at >= {from} and created_at < {to}
      query_args:
        parse_dates: ['created_at']
  new_signups:
    plugin: plot
    title: Daily new sign-ups
    description: |
```

(continues on next page)

(continued from previous page)

```
    How many new users signed up on each day.
  cols: 4
  args:
    kind: bar
    datasource: db
    query: |
      select date_format(created_at, '%%Y-%%m-%%d') as day, count(*) as daily_total
      from kpireport.signups
      where created_at >= {from} and created_at < {to}
      group by day(created_at)
    query_args:
      # Because we're roughly grouping by day, and not a full date time,
      # automatic parsing of dates doesn't quite work, so we need to give
      # a bit of help to say which columns should be treated as dates.
      parse_dates:
        day: '%Y-%m-%d'
    time_column: day
# Additionally, show a stat indicating how much the total signups have
# increased (or decreased) in comparison to the previous report period.
signups_change:
  plugin: single_stat
  title: Week total
  cols: 2
  args:
    datasource: db
    query: |
      select count(*) from kpireport.signups
      where created_at >= {from} and created_at < {to}
    comparison_query: |
      select count(*) from kpireport.signups
      where created_at >= date_sub({from}, {interval})
        and created_at < {from}
    comparison_type: percent
    # comparison_type: raw
new_signups_table:
  plugin: table
  args:
    datasource: db
    query: |
      select date_format(created_at, '%%Y-%%m-%%d') as Day, count(*) as 'Daily total'
      from kpireport.signups
      where created_at >= {from} and created_at < {to}
      group by day(created_at)
```

3.2 CI report

View HTML

This example uses both a View and Datasource provided by the Jenkins plugin to show an overview of build jobs and their success/failure statuses.

```
---
title: CI report

datasources:
  jenkins:
    plugin: jenkins
    args:
      host: jenkins:8080
      user: jenkins_user
      api_token: jenkins_token

views:
  app_build_summary:
    plugin: jenkins.build_summary
    title: Problem application builds
    args:
      filters:
        name: -app
  other_builds:
    plugin: jenkins.build_summary
    title: Other builds
    args:
      filters:
        invert: True
        name: -app
```

3.3 Ops report

View HTML

This example uses both a View and Datasource provided by the Prometheus plugin to show a visualization of some time series data representing server load, as well as a summary of alerts fired by the Prometheus server over the report window.

```
---
title: Ops report

datasources:
  prom:
    plugin: prometheus
    args:
      host: prometheus:9090

views:
  server_load:
```

(continues on next page)

(continued from previous page)

```
    plugin: plot
    title: Load
    args:
      datasource: prom
      query: |
        100 - (avg by(hostname) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100
      groupby: hostname
      plot_rc:
        lines.linewidth: 1
  critical_alerts:
    plugin: prometheus.alert_summary
    title: Critical alerts
    args:
      datasource: prom
      labels:
        severity: critical
  warning_alerts:
    plugin: prometheus.alert_summary
    title: Warnings
    args:
      datasource: prom
      show_timeline: False
      labels:
        severity: warning
```

CHAPTER

FOUR

ABOUT THIS TOOL

4.1 Motivation

Visualizing metrics is an incredibly common and valuable task. Virtually every department in a business likes to leverage data when making decisions. Time and time again I've seen developers implement one-off solutions for automatic reporting, sometimes quick and dirty, sometimes highly polished. For example:

- A weekly email to the entire company showing the main engagement KPIs for the product.
- A weekly Slack message to a team channel showing how many alerts the on-call team member had to respond to in the previous week.
- A daily summary of sales numbers and targets.

Thanks largely to Grafana, teams are customizing real-time dashboards to always have an up-to-date view on the health of a system or the health of the business. However, there is often value in distilling a continuum of real-time metrics into a short digestible report. KPI Reporter attempts to make it easier to build such reports.

A few guiding principles that shape this project:

1. **It should be possible to run on-premises.** It is far easier to run a reporting tool within an infrastructure due to the amount of data sinks that must be accessible. Security teams should rightly raise eyebrows when databases are exposed externally just so a reporting tool can reach in.
2. **It should be highly customizable.** There should not be many assumptions about either layout or appearance. The shape and type of data ultimately will drive this.
3. **It should be possible to extend.** The space of distinct user needs is massive. While the tool should aim to provide a lot of useful functionality out of the box, it will always be the case that custom extensions will be required to achieve a particular implementation. The tool should embrace this reality.

4.2 Pricing

KPI Reporter is **free for personal use and by noncommercial organizations**. All commercial users are required to *obtain an annual license* after 30 days. I believe that if you find enough value in the tool, this is a fair trade. You are free to implement your own third-party plugins at any time and distribute them under any license and/or pricing you wish.

Personal use is defined as:

> *Personal use for research, experiment, and testing for the benefit of public knowledge, personal study, private entertainment, hobby projects, amateur pursuits, or religious observance, without any anticipated commercial application.*

A noncommercial organization is defined as:

Any charitable organization, educational institution, public research organization, public safety or health organization, environmental protection organization, or government institution, regardless of the source of funding or obligations resulting from the funding.

```
# The Prosperity Public License 3.0.0

Contributor: KPI Reporter LLC

Source Code: https://github.com/kpireporter/kpireporter

## Purpose

This license allows you to use and share this software for noncommercial
purposes for free and to try this software for commercial purposes for thirty
days.

## Agreement

In order to receive this license, you have to agree to its rules.  Those rules
are both obligations under that agreement and conditions to your license.  Don't
do anything with this software that triggers a rule you can't or won't follow.

## Notices

Make sure everyone who gets a copy of any part of this software from you, with
or without changes, also gets the text of this license and the contributor and
source code lines above.

## Commercial Trial

Limit your use of this software for commercial purposes to a thirty-day trial
period.  If you use this software for work, your company gets one trial period
for all personnel, not one trial per person.

## Contributions Back

Developing feedback, changes, or additions that you contribute back to the
contributor on the terms of a standardized public software license such as [the
Blue Oak Model License 1.0.0](https://blueoakcouncil.org/license/1.0.0), [the
Apache License 2.0](https://www.apache.org/licenses/LICENSE-2.0.html), [the MIT
license](https://spdx.org/licenses/MIT.html), or [the two-clause BSD
license](https://spdx.org/licenses/BSD-2-Clause.html) doesn't count as use for a
commercial purpose.

## Personal Uses

Personal use for research, experiment, and testing for the benefit of public
knowledge, personal study, private entertainment, hobby projects, amateur
pursuits, or religious observance, without any anticipated commercial
application, doesn't count as use for a commercial purpose.

## Noncommercial Organizations

```

(continues on next page)

(continued from previous page)

```
Use by any charitable organization, educational institution, public research
organization, public safety or health organization, environmental protection
organization, or government institution doesn't count as use for a commercial
purpose regardless of the source of funding or obligations resulting from the
funding.

## Defense

Don't make any legal claim against anyone accusing this software, with or
without changes, alone or with other technology, of infringing any patent.

## Copyright

The contributor licenses you to do everything with this software that would
otherwise infringe their copyright in it.

## Patent

The contributor licenses you to do everything with this software that would
otherwise infringe any patents they can license or become able to license.

## Reliability

The contributor can't revoke this license.

## Excuse

You're excused for unknowingly breaking [Notices](#notices) if you take all
practical steps to comply within thirty days of learning you broke the rule.

## No Liability

***As far as the law allows, this software comes as is, without any warranty or
condition, and the contributor won't be liable to anyone for any damages related
to this software or this license, under any kind of legal claim.***
```

4.2.1 Obtaining a license

You can purchase a one-year license for $99/year (for individuals/small businesses) or $499/year (for larger businesses.) You can use discretion about which tier you belong to.

4.3 Comparisons

There are several existing products and applications that do some of what KPI Reporter does. Many of them do a far better job depending on your specific use-case. Here is a brief summary of the ones I know of.

4.3.1 Cloud SaaS

All of these services are rather sophisticated and focus on delivering user-friendly interfaces aimed at a wider range of skill-sets. They are also more expensive for that reason. Most support some overlapping (and typically wider) set of data sources and reporting/visualization capabilities as KPI Reporter.

- Chartio: subscriptions start at $40/month per user after a 14 day trial.
- Databox: subscriptions start at $49/month for 10 users and 10 data sources.
- Daily Metrics: a subscription is $10/month
- Grafana Enterprise Reporting: pricing is only available on request, putting it safely above any of the other products.
- Grow.com: pricing available on request, similar to Grafana Enterprise.
- Klipfolio: a subscription is $70/month after a 14-day trial.
- Sunrise KPI: a subscription is $15/month after a 14-day trial.

4.3.2 On-premises

- Zoho Analytics: while principally another cloud SaaS product, Zoho Analytics is the only offering I could find that supports an on-premises deployment at time of writing. The on-premises version is $30/month per seat, with a minimum of 5 seats, so $150/month. The online version is $45/month for the same number of users.

CHAPTER

FIVE

GOOGLE ANALYTICS

```
pip install kpireport-googleanalytics
```

5.1 API

class kpireport_googleanalytics.datasource.**GoogleAnalyticsDatasource**(*report:* Report, ***kwargs*)

Bases: *Datasource*

A Datasource that provides data from the Google Analytics APIs.

This Datasource supports a whitelist of query types:

report: Get a Report from the V4 Reporting API.
See *query_report()* for additional options/arguments.

To use in a View, the type of query is specified as the first argument. Any additional keyword arguments are interpreted as options specific to that query type.

```
# From within a View member function...
df = self.datasources.query("ga", "report", account_like="MyAccount")
```

key_file

The Google service account key file (must be in JSON format.) Refer to the Google Cloud documentation for more information on how to set up this authentication credential. Default `/etc/kpireporter/google_oauth2_key.json`.

Type
str

query(*input: str*, ***kwargs*) → DataFrame

Query the Google Analytics API.

Parameters

- **input** (*str*) – The name of the query command to invoke. Currently supports only "report".
- ****kwargs** – keyword arguments to pass through to invoked report command.

Returns
A DataFrame with the query results.

query_report(*account_like=None*, *property_like=None*, *view_like=None*, *dimensions=None*, *metrics=None*, *filters_expression=None*, *order_bys=None*) → DataFrame

Request a report from the GA v4 Analytics API.

Parameters

- **account_like** (*str*) – the GA account name or ID to search for the view. If not defined, the first account found is used.
- **property_like** (*str*) – the GA property name or ID to search for the view. If not defined, the first property found is used.
- **view_like** (*str*) – the GA view name or ID. If not defined, the first view found is used.

  Note: If you have multiple accounts or properties available from your credentials, ensure you set `account_like` and `property_like` if you are using this field, as the default functionality for both of those options is to naively take the first account/property found, which may not have the view you're looking for.

- **dimensions** (*List[str]*) – a list of dimensions. These can be just dimension names, or the full object syntax. If one of these dimensions is a "date-like" dimension (e.g., "ga:date.*"), the output DataFrame will have this dimension treated as a DateTimeIndex, making it effectively return something that looks like a time series.
- **metrics** (*List[str]*) – a list of metrics. These can be just metric expressions, or the full object syntax.
- **filters_expression** (*str*) – an optional filter expression.
- **order_bys** (*List[dict]*) – a list of orderings.

Returns

a `pd.DataFrame` with dimensions and metrics added.
The dimensions will be the first columns in the resulting table, and each metric returned will be in a subsequent column.

Return type
pd.DataFrame

5.2 Changelog

5.2.1 0.0.2

Bug Fixes

- Fixes support for Python 3.7

5.2.2 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

SIX

JENKINS

```
pip install kpireport-jenkins
```

The Jenkins plugin provides both a Datasource for querying the Jenkins API, as well as a View for displaying a summary of job/build statuses. The list of jobs can be filtered to target jobs that are of interest in your reporting.

6.1 Datasource

```
---
title: CI report

datasources:
  jenkins:
    plugin: jenkins
    args:
      host: jenkins:8080
      user: jenkins_user
      api_token: jenkins_token

views:
  app_build_summary:
    plugin: jenkins.build_summary
    title: Problem application builds
    args:
      filters:
        name: -app
  other_builds:
    plugin: jenkins.build_summary
    title: Other builds
    args:
      filters:
        invert: True
        name: -app
```

6.2 Build summary

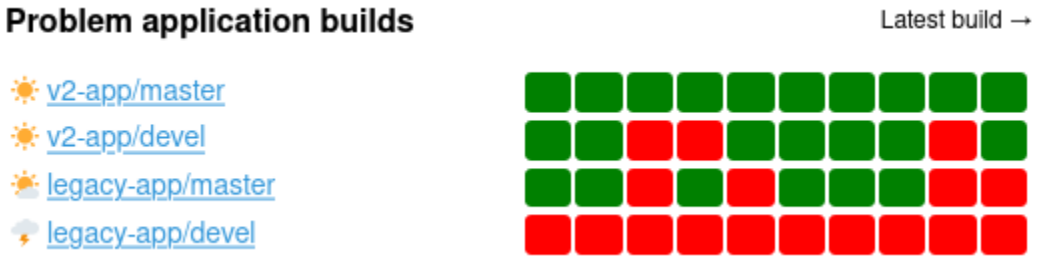

Fig. 1: An example showing jobs matching a certain name pattern "*-app"

6.3 API

class kpireport_jenkins.datasource.**JenkinsDatasource**(*report:* Report, ***kwargs*)

Bases: *Datasource*

Provides accessors for listing all jobs and builds from a Jenkins host.

The Jenkins Datasource exposes an RPC-like interface to fetch all jobs, as well as job details for each job. The plugin will call the Jenkins API at the host specified using a username and API token provided as plugin arguments.

host

Jenkins host, e.g. https://jenkins.example.com.

Type
str

user

Jenkins user to authenticate as.

Type
str

api_token

Jenkins user API token to authenticate with.

Type
str

get_all_jobs()

List all jobs on the Jenkins server.

Returns

a DataFrame with columns:

fullname
the full job name (will include folder path components)

url
a URL that resolves to the job on the Jenkins server

Return type
pandas.DataFrame

`get_job_info`(*job_name*)

Get a list of builds for a given job, including their statuses.

Parameters
job_name (*str*) – Full name of the job.

Returns

a DataFrame with columns:

status
the build status, e.g. "SUCCESS" or "FAILURE"

Return type
pandas.DataFrame

`query`(*fn_name*, **args*, ***kwargs*)

Query the Datsource for job or build data.

Calls a supported accessor function by name and passthrough any positional and keyword arguments.

Examples:

```
# Get a list of all jobs in the Jenkins server
datasources.query("jenkins", "get_all_jobs")
# Get detailed information about 'some-job'
datasources.query("jenkins", "get_job_info", "some-job")
```

Parameters
fn_name (*str*) – the RPC operation to invoke.

Raises
ValueError – if an invalid RPC operation is requested.

`class kpireport_jenkins.build_summary.JenkinsBuildSummary`(*report: Report*, *datasources: DatasourceManager*, ***kwargs*)

Bases: *View*

Display a list of jobs with their latest build statuses, and health.

Formats
html, md

Parameters

- **datasource** (*str*) – the Datasource ID to query for Jenkins data
- **filters** (*dict*) – optional filters to limit which jobs are rendered in the view. These filters are directly passed to *JenkinsBuildFilter*.

`class kpireport_jenkins.build_summary.JenkinsBuildFilter`(*name=None*, *invert=False*)

Filters a list of Jenkins jobs/builds by a general criteria

Currently only filtering by name is supported, but this class can be extended in the future to filter on other attributes, such as build status or health.

Parameters

- **name** (*Union[str, List[str]]*) – the list of name filter patterns. These will be compiled as regular expressions. In the case of a single filter, a string can be provided instead of a list.
- **invert** (*bool*) – whether to invert the filter result

filter_job(*job*)

Checks a job against the current filters

Parameters
job (*dict*) – the Jenkins job

Return type
bool

Returns
whether the job passes the filters

6.4 Changelog

6.4.1 0.0.1

Prelude

Initial release.

New Features

- Initial release.

CHAPTER

SEVEN

MYSQL

```
pip install kpireport-sql
```

The SQL plugin provides a Datasource that enables execution of SQL queries against an existing MySQL or SQLite database.

```
---
title: Top-of-funnel report

datasources:
  db:
    plugin: mysql
    args:
      host: mysql
      user: kpireport
      passwd: kpireport_pass

views:
  # Show a time series of user signups over the report period.
  signups_over_time:
    plugin: plot
    title: Total sign-ups
    description: |
      The running total of user accounts.
    args:
      datasource: db
      query: |
        select created_at,
        count(*) over (order by created_at)
          + (select count(*) from kpireport.signups where created_at < {from})
          as total_signups
        from kpireport.signups
        where created_at >= {from} and created_at < {to}
      query_args:
        parse_dates: ['created_at']
  new_signups:
    plugin: plot
    title: Daily new sign-ups
    description: |
      How many new users signed up on each day.
```

(continues on next page)

(continued from previous page)

```
    cols: 4
    args:
      kind: bar
      datasource: db
      query: |
        select date_format(created_at, '%%Y-%%m-%%d') as day, count(*) as daily_total
        from kpireport.signups
        where created_at >= {from} and created_at < {to}
        group by day(created_at)
      query_args:
        # Because we're roughly grouping by day, and not a full date time,
        # automatic parsing of dates doesn't quite work, so we need to give
        # a bit of help to say which columns should be treated as dates.
        parse_dates:
          day: '%Y-%m-%d'
      time_column: day
  # Additionally, show a stat indicating how much the total signups have
  # increased (or decreased) in comparison to the previous report period.
  signups_change:
    plugin: single_stat
    title: Week total
    cols: 2
    args:
      datasource: db
      query: |
        select count(*) from kpireport.signups
        where created_at >= {from} and created_at < {to}
      comparison_query: |
        select count(*) from kpireport.signups
        where created_at >= date_sub({from}, {interval})
          and created_at < {from}
      comparison_type: percent
      # comparison_type: raw
  new_signups_table:
    plugin: table
    args:
      datasource: db
      query: |
        select date_format(created_at, '%%Y-%%m-%%d') as Day, count(*) as 'Daily total'
        from kpireport.signups
        where created_at >= {from} and created_at < {to}
        group by day(created_at)
```

7.1 API

class kpireport_sql.datasource.**SQLDatasource**(*report:* Report, ***kwargs*)

Bases: *Datasource*

Provides an interface for running queries agains a SQL database.

driver

which DB driver to use. Possible values are "mysql" and "sqlite".

Type
str

kwargs

any keyword arguments are passed through to `pymysql.connect()` (in the case of the MySQL driver) or `sqlite3.connect()` (for the SQLite driver.)

query(*sql: str*, ***kwargs*) → DataFrame

Execute a query SQL string.

Some special tokens can be included in the SQL query. They will be replaced securely and escaped with the built-in parameter substition capabilities of the MySQL client.

- `{from}`: the start date of the Report
- `{to}`: the end date of the Report
- `{interval}`: an interval string set to the Report interval, i.e., how many days is the Report window. This is useful when doing date substitution, e.g.

```
; Also include previous interval
WHERE time > DATE_SUB({from}, {interval})
```

Note: By default, no automatic date parsing will occur. To ensure that your timeseries data is properly parsed as a date, use the `parse_dates` kwarg supported by `pandas.read_sql()`, e.g.,

```
self.datasources.query('my_db', 'select time, value from table',
  parse_dates=['time'])
```

Parameters

- **sql** (*str*) – the SQL query to execute
- **kwargs** – keyword arguments passed to `pandas.read_sql()`

Returns

a table with any rows returned by the query.

Columns selected in the query will be columns in the output table.

Return type
pandas.DataFrame

7.2 Changelog

7.2.1 0.1.0

New Features

- Adds support for SQLite in addition to MySQL. This is controlled via a new `driver` kwargs, which defaults to "mysql".

7.2.2 0.0.2

Prelude

Initial commit.

New Features

- Initial commit.

Bug Fixes

- The MySQL connector library is now PyMySQL, which is a pure Python implementation and does not require additional libraries on the host.

CHAPTER

EIGHT

PLOT

```
pip install kpireport-plot
```

8.1 Plot

The Plot is a simple workhorse View for displaying a variety of timeseries data. It is designed to be compatible with several Datasources and handle most KPI graphs, which tend to plot only a single metric or perhaps a set of related metrics. It utilizes `matplotlib` under the hood.

```
views:
   new_signups:
      plugin: plot
      title: New signups
      args:
         datasource: users_db
         kind: bar
         query: |
            select
               date_format(created_at, '%Y-%m-%d') as time,
               count(*) as daily_total
            from signups
            where created_at >= {from} and created_at < {to}
            group by day(created_at)
         query_args:
            parse_dates:
               time: '%Y-%m-%d'
```

8.2 Single stat

Sometimes it is useful to only show one number, and a fine-grained trend of the number is less important; in this case, you can use a "single stat" view, which is included as part of the Plot plugin for convenience.

```
views:
   new_signups:
      plugin: single_stat
```

(continues on next page)

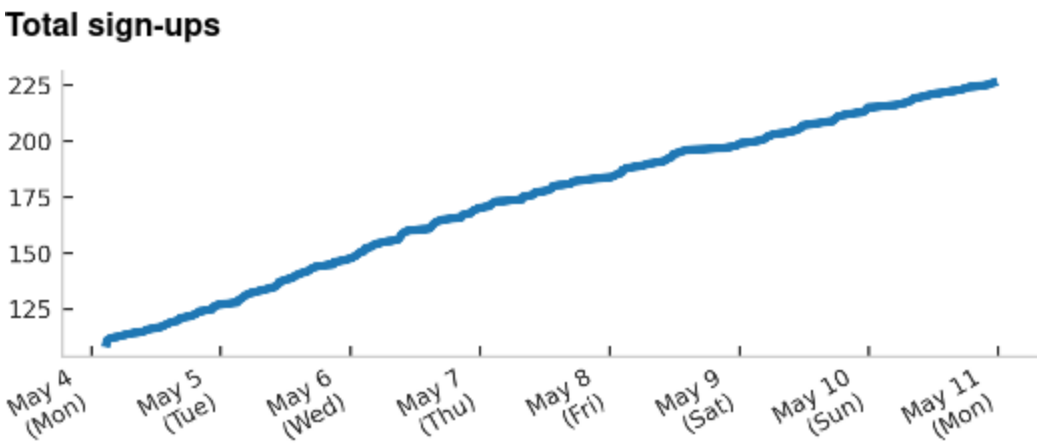

Fig. 1: A simple line plot from MySQL data

(continued from previous page)

```
    title: New signups
    args:
      datasource: users_db
      query: |
        select count(*)
        from signups
        where created_at >= {from} and created_at < {to}
      comparison_query: |
        select count(*)
        from signups
        where created_at >= date_sub({from}, {interval})
          and created_at < {from}
      comparison_type: percent
```

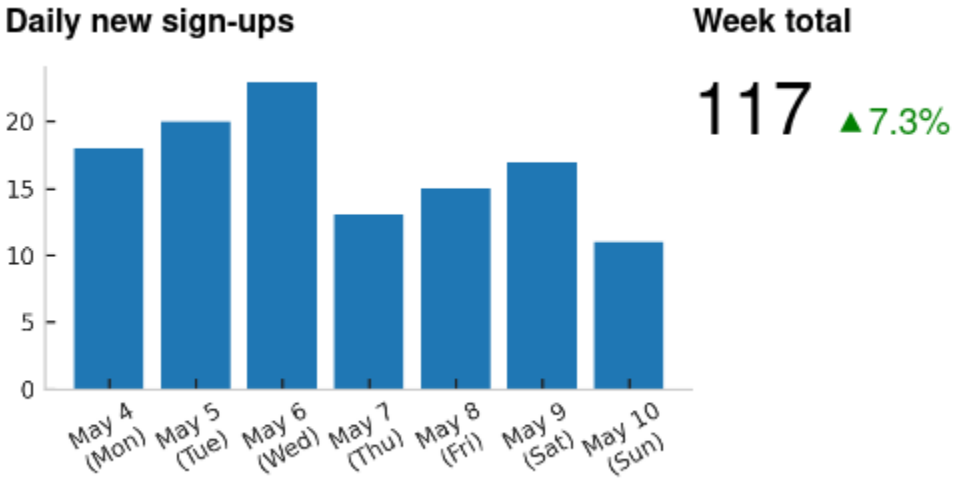

Fig. 2: An example of a plot view combined with a single stat view

8.3 API

class kpireport_plot.**Plot**(*report:* Report, *datasources:* DatasourceManager, ***kwargs*)

Bases: *View*

Render a graph as a PNG file inline.

The `matplotlib` module handles rendering the plot. The Plot view sets a few default styles that make sense for plotting timeseries data, such as hiding the x-axis label and setting up date formatting for the x major labels.

Expected data formats

The Plot plugin can work for many different types of queries and Datasources, as long as a few properties hold for the response:

- The returned table should ideally only have two columns: one for the time, and one for the value of the metric at that time.
- If the table has more than two columns, it is assumed that each column is a separate series and will be displayed as such, unless `groupby` is used.

Example of valid two-column result table:

time	value
2020-01-01	1.0
2020-01-02	1.2
2020-01-03	2.1

Example of valid three-column result table:

time	value_a	value_b
2020-01-01	1.0	0.4
2020-01-01	3.2	0.2
2020-01-02	1.2	0.2
2020-01-02	2.7	0.7

Example of valid three-column result table, where `groupby: country` would cause the data to be segmented according to the `country` column:

time	value	country
2020-01-01	1.0	USA
2020-01-01	3.2	Germany
2020-01-02	1.2	USA
2020-01-02	2.7	Germany

datasource

ID of Datasource to fetch from.

Type
str

query

the query to execute against the Datasource.

Type
str

`query_args`

additional arguments to pass to the query function. Some Datasources may support additional parameters.

Type
dict

`time_column`

the name of the column in the query result table that contains timeseries data. (Default `"time"`)

Type
str

`kind`

the kind of plot to draw. Currently only "line", "bar", and "scatter" are supported. (Default `"line"`)

Type
str

`stacked`

whether to display the line/bar graph types as a stacked plot, where each series is stacked atop the last. This does not have any effect when rendering scatter plots. (Default `False`)

Type
bool

`groupby`

the name of the column in the query result table that should be used to group the data into separate series. (Default `None`)

Type
str

`bar_labels`

whether to label each bar with its value (only relevant when kind is "bar".) (Default `False`)

Type
bool

`xtick_rotation`

how much to rotate the X labels by when displaying.

Type
Union[int, str]

`plot_rc`

properties to set as `matplotlib.RcParams`. This can be used to customize the display of the output chart beyond the defaults provided by the Theme.

Type
dict

`post_plot`(*ax*, *df=None*, *index_data=None*, *series_data=None*)

A post-render hook that can be used to process the plot before outputting.

Subclasses of the Plot class can override this to add, e.g., annotations or otherwise tweak the final plot output.

`class kpireport_plot.SingleStat`(*report:* Report, *datasources:* DatasourceManager, ***kwargs*)

Bases: *View*

Display a single stat and optionally a delta.

If the input query returns multiple rows, the rows are summed. If the query result is a DataFrame with multiple columns, only the first column is summed.

`datasource`

the ID of the Datasource to query.

Type
str

`query`

the query to execute agains the Datasource.

Type
str

`query_args`

additional arguments to pass to the query function. Some Datasources may support additional parameters.

Type
dict

`label`

a templated label that can be used to change how the stat is rendered. A `{stat}` template variable will be filled in with the stat value. (Default `"{stat}"`)

This can be used to create arbitrary other rendered output, e.g.:

```
# Add a separate link element
label: |
  {stat} <a href="https://example.com">More</a>
```

Type
str

`link_url`

a hyperlink URL to open if the viewer clicks on the rendered output. The link wraps the entire display. (Default `None`)

Type
str

`comparison_query`

an optional query to use as a comparison value. If defined, the current stat will be displayed, and the delta between the stat obtained via the comparison query will be shown next to it. (Default `None`)

Type
str

`comparison_query_args`

additional arguments to pass to the query function. Some Datasources may support additional parameters.

Type
dict

`comparison_type`

how to show the delta; possible values are “raw”, meaning the raw difference between the two values is displayed, or “percent”, meaning the percentage increase/decrease is displayed. (Default `"raw"`)

Type
str

`precision`

The floating point precision to use on the resulting stat. Set to the number of significant digits you want displayed. If 0, the stat is rounded to the nearest integer (Default `0`)

Type
int

8.4 Changelog

8.4.1 0.3.1

Bug Fixes

- Fixes issue creating single stat view from single-column (i.e. index-only) DataFrame from a datasource query.

8.4.2 0.3.0

New Features

- A new `precision` argument is available to allow you to define the displayed precision of the stat. Precision defaults to 0, meaning the stat is rounded to the nearest integer.
- Query results that return multiple rows are columns now have some attempted support. If there are columns, the first column is summed and displayed. If there is only an index, but it contains multiple rows, the rows are summed.

8.4.3 0.2.2

Bug Fixes

- Fixes support for Python 3.7

8.4.4 0.2.1

Bug Fixes

- Plots will now be clipped on the x-axis to the report window.

8.4.5 0.2.0

New Features

- Plots now support a `label_map` option, which maps series names (column names) to some text value. This can be used to make nicer, more readable legends.
- The Plot class now defines a `post_plot` function. This is called directly after running most of the plot logic, but before rendering the final output. It is a good place to modify legends or add additional presentation to the plot output. The intention is that a plugin could subclass Plot and provide some additional hooks here.

Bug Fixes

- When trying to plot DataFrames with multiple non-numeric columns, the plotter will now log a warning and drop the non-numeric columns. Non-numeric columns should be grouped over, not plotted.

8.4.6 0.1.2

Bug Fixes

- Fixes issue where stacked line graphs would not render properly in some cases.

8.4.7 0.1.1

Bug Fixes

- Fixes an issue where non-grouped data with multiple columns would only have the first series plotted, and no legend would be auto-applied.

8.4.8 0.1.0

New Features

- Adds a new `groupby` configuration parameter, allowing grouping of the queried data on some arbitrary column or columns. This replaces the old "auto-grouping" mechanism, which was a bit buggy and not well-suited to all shapes of data.

8.4.9 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

NINE

PROMETHEUS

```
pip install kpireport-prometheus
```

The Prometheus plugin provides both a Datasource capable of returning PromQL query results and a View that summarizes alerts fired by the Prometheus server over the report interval.

9.1 Datasource

```
datasources:
   prom:
      plugin: prometheus
      args:
         host: prometheus:9090
views:
   # Using Plot plugin to graph data
   server_load:
      plugin: plot
      title: Load
      args:
         datasource: prom
         query: |
         100 - (avg by(hostname) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100
```

9.2 Alert summary

```
critical_alerts:
   plugin: prometheus.alert_summary
   title: Critical alerts
   args:
      datasource: prom
      labels:
         severity: critical
warning_alerts:
   plugin: prometheus.alert_summary
   title: Warnings
```

(continues on next page)

(continued from previous page)

```
  args:
    datasource: prom
    show_timeline: False
    labels:
      severity: warning
```

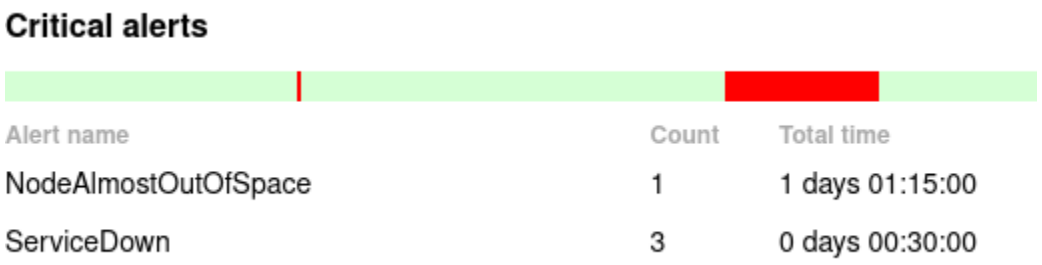

Fig. 1: An example rendered alert summary. The timeline at the top displays the points in time when any alert was firing over the report window. Individual alert labels are not shown; the view's purpose is to highlight trends or patterns that can be looked at in more detail at the source.

9.3 API

class kpireport_prometheus.**PrometheusDatasource**(*report:* Report, ***kwargs*)

Bases: *Datasource*

Datasource that executes PromQL queries against a Prometheus server.

host

the hostname of the Prometheus server (may include port), e.g., `https://prometheus.example.com:9090`. If no protocol is given, "http://" is assumed.

Type
str

basic_auth

HTTP Basic Auth credentials to use when authenticating to the server. Must be a dictionary with `username` and `password` keys.

Type
dict

query(*query: str*, *step='1h'*) → DataFrame

Execute a PromQL query against the Prometheus server.

Parameters

- **query** (*str*) – the PromQL query
- **step** (*str*) – the step size for the range query. The Datasource will execute a range query over the report window and capture all time series data within the report boundaries. The step size indicates the query resolution. A lower value provides more granularity but at the cost of a more expensive query and more data points to analyze. If your report window is significantly short, it may make sense to reduce this.

Returns

a table of time series results.

The timeseries value will be in a `time` column; any labels associated with the metric will be added as additional columns.

Return type
pandas.DataFrame

class kpireport_prometheus.**PrometheusAlertSummary**(*report:* Report, *datasources:* DatasourceManager, ***kwargs*)

Bases: *View*

Display a list of alerts that fired recently.

Supported output formats: `html`, `md`, `slack`

datasource

the ID of the Prometheus Datasource to query

Type
str

resolution

the size of the time window used to group alerts–the window is used to define buckets. A higher resolution is a lower time window (e.g., "5m" versus "1h"–"5m" is the higher resolution). Higher resolutions mean the timeline and time estimates for outage length will be more accurate, but may decrease performance when the report interval is large, as it requires pulling more data from Prometheus. (default `"15m"`)

Type
str

hide_labels

a set of labels to hide from the output display. Alerts containing these labels will still be listed, but the label values will not be printed. (default `["instance", "job"]`)

Type
List[str]

labels

a set of labels that the alert must contain in order to be displayed (default `None`)

Type
Dict[str,str]

ignore_labels

a set of labels that the alert must _not_ contain in order to be displayed (default `None`)

Type
Dict[str,str]

show_timeline

whether to show a visual timeline of when alerts were firing (default `True`)

Type
bool

timeline_height

rendered height of the timeline in pixels (default `15`)

Type
int

9.4 Changelog

9.4.1 0.0.2

Bug Fixes

- Fixes an issue where the alert summary timeline view would not render if datetimes included timezone information.

9.4.2 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

TEN

S3

```
pip install kpireport-s3
```

The S3 plugin provides an output driver that can upload the final report contents to an S3 bucket. Each file in the report output structure is uploaded as a separate object. Each report is outputted with its report ID, which contains the report interval. Additionally, a special report with the "latest" designation is overridden with the last generated report.

Note: Currently only the HTML format is supported.

10.1 API

class kpireport_s3.**S3OutputDriver**(*report:* Report, ***kwargs*)

Bases: *StaticOutputDriver*

bucket

the S3 bucket to upload to.

Type
str

prefix

the key prefix.

Type
str

kwargs

any additional keyword arguments are passed in to the `boto3.client` constructor.

10.2 Changelog

10.2.1 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

ELEVEN

SCP (SECURE COPY)

```
pip install kpireport-scp
```

The `scp` plugin writes the report contents to local disk and then copies them to a remote host via `scp`. This can be used to copy the report to a server's webroot or simply keep a backup around.

Note: Currently only the HTML format is supported.

Note: This plugin utilizes `fabric` for executing commands over SSH, which in turn utilizes `paramiko`, which is licensed under LGPL 2.1. A copy of this license is included in `LGPL-2.1.md` in the plugin source.

11.1 API

class kpireport_scp.**SCPOutputDriver**(*report:* Report, ***kwargs*)

Bases: *StaticOutputDriver*

11.2 Changelog

11.2.1 0.0.3

Bug Fixes

- Fixes the entry point declaration to be valid so the plugin is discoverable.

11.2.2 0.0.2

Bug Fixes

- The destination path on the remote host will be lazily created if not existing.

11.2.3 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

TWELVE

SENDGRID

```
pip install kpireport-sendgrid
```

Send the report as an email using SendGrid. This plugin utilizes SendGrid's API, and you must generate an API key that has the "Mail Send" permission to authenticate and send the report.

Images or other attachments are by default embedded within the email, but you can optionally link them in as remote assets instead. Remote linking requires that the report additionally be placed somewhere accessible over the Internet, via, e.g., the *Static file*, *S3*, or *SCP (secure copy)* plugins.

Note: This plugin utilizes `premailer` for CSS inlining, which in turn utilizes `cssutils`, which is licensed under LGPL 3.0. A copy of this license is included in `LGPL-3.0.md` in the plugin source.

12.1 API

class kpireport_sendgrid.**SendGridOutputDriver**(*report:* Report, ***kwargs*)

Bases: *OutputDriver*

Email a report via SendGrid.

Note: When testing, you can set a `SENDGRID_SANDBOX_ENABLED=1` environment variable, which will only verify the mail payload on SendGrid, but will not actually send it.

email_from

the sender email address.

Type
str

email_to

a list of target email addresses.

Type
List[str]

api_key

a SendGrid API key authorized to send mail on behalf of the sending address.

Type
str

12.2 Changelog

12.2.1 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

THIRTEEN

SLACK

```
pip install kpireport-slack
```

Output a report to one or more Slack channel(s). Because Slack has its own flavor of Markdown, which does not support many of the "standard" Markdown features, it has its own output format. All view plugins provided by KPI Reporter have support for outputting to Slack, but third-party plugins may not.

13.1 API

class kpireport_slack.**SlackOutputDriver**(*report:* Report, ***kwargs*)

Bases: *OutputDriver*

Send a report to one or more Slack channel(s).

api_token

a Slack API token with authorization to publish a message to the target channel.

Type
str

channels

a list of Slack channels to publish the report to.

Type
List[str]

image_remote_base_url

a base URL where blob assets (images etc.) are served. It is highly recommended to use another plugin, such as the *S3* or *SCP (secure copy)* plugin, in order to place the assets in the expected folder structure.

Type
str

13.2 Changelog

13.2.1 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

FOURTEEN

SMTP

```
pip install kpireport-smtp
```

Note: This plugin utilizes `premailer` for CSS inlining, which in turn utilizes `cssutils`, which is licensed under LGPL 3.0. A copy of this license is included in `LGPL-3.0.md` in the plugin source.

14.1 API

class kpireport_smtp.**SMTPOutputDriver**(*report:* Report, ***kwargs*)

Bases: *OutputDriver*

Email a report's contents via SMTP to one or more recipients.

email_from

From email address.

Type
str

email_to

Email address(es) to send to.

Type
Union[str,List[str]]

smtp_host

SMTP server to relay mail through. Defaults to "localhost".

Type
str

smtp_port

SMTP port to use. Defaults to 25.

Type
int

image_strategy

Strategy to use for including images in the mail contents. Two options are available:

- `embed`: embed the image directly in the mail using Content-ID (RFC2392) linked resources. These should be compatible with most modern desktop and web mail clients.
- `remote`: link the image to a remote resource. For this strategy to work, the image assets must exist on a server reachable via the public Internet (and not require authentication). Consider using the SMTP plugin in conjunction with e.g., the *S3* or *SCP* plugins to accomplish this entirely within KPI reporter.

Note: No tracking information is included when rendering remote image URLs; if for some reason you need to track open rates, consider using the *SendGrid* plugin to send the report instead.

Type
str

`image_remote_base_url`

When using the "remote" image strategy, the base URL for the image assets. Image blobs generated by Views are placed in folders named after the View ID; this base URL should point to the root path for all of these folders.

Type
str

14.2 Changelog

14.2.1 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

FIFTEEN

STATIC FILE

```
pip install kpireport-static
```

15.1 API

class kpireport_static.**StaticOutputDriver**(*report:* Report, ***kwargs*)

Bases: *OutputDriver*

Export a report's contents to disk.

output_dir

the directory to output the report contents to. (Default "./_build")

Type
str

output_format

The output format, which can be one of "html" or "png". (Default "html".) Depending on the format, the output will have a few different forms:

Html
The HTML report will be outputted in a new directory in the output path, named after the report. A "latest" directory will also be outputted/updated with the contents of this report.

Png
The report will be rendered as a single PNG image named after the report in the output path, and a "latest" PNG will also be outputted.

Note: `wkhtmltopdf` is required if using PNG output. You will probably also need to install `Xvfb` if using a Docker container that doesn't already have an X server packaged.

Type
str

15.2 Changelog

15.2.1 0.1.1

Bug Fixes

- Fixes an issue where relative font sizes would not be rendered accurately when using the PNG output format.
- Fixes issue with PNG output when no X server is installed; imgkit will fail because wkhtmltoimage requires an X server to capture the output. Xvfb is already supported as a workaround; if this is installed on the host, configure imgkit to use it.

15.2.2 0.1.0

New Features

- A new `output_path` option is now available and can be used to instruct the plugin to output a rendered PNG file instead of the default HTML contents. PNG output requires the `wkhtmltopdf` binary to be installed on the host.

15.2.3 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

SIXTEEN

TABLE

```
pip install kpireport-table
```

16.1 API

class kpireport_table.**Table**(*report:* Report, *datasources:* DatasourceManager, ***kwargs*)

Bases: *View*

Render data fetched from a datasource as a table.

A table can be rendered as HTML, Markdown, or for display in *Slack*. When displayed for Slack, the table is rendered as an image, as Slack does not have support for rendering tables as part of a message block.

datasource

the datasource ID to fetch from.

Type
str

query

the query to execute against the datasource.

Type
str

query_args

any additional keyword arguments to the datasource query operation.

Type
dict

max_rows

maximum number of rows to display. If the output table has more rows, they are ignored. (Default 10)

Type
int

16.2 Changelog

16.2.1 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

SEVENTEEN

TWITTER

```
pip install kpireport-twitter
```

17.1 API

class kpireport_twitter.**TwitterDatasource**(*report:* Report, ***kwargs*)

Bases: *Datasource*

A datasource that can fetch metrics from Twitter's V2 API.

Currently the following queries are supported:

- `tweets`: request a list of the user's latest Tweets via the Timeline API. If requesting the authenticated user's own timeline, "non-public" metrics such as impression counts are included in the output table result. Otherwise, only public metrics such as like, reply, and retweet counts are retrieved. The text and ID of the Tweet are also included in the output table.

consumer_key

The Twitter application consumer public key.

Type
str

consumer_secret

The Twitter application consumer secret.

Type
str

access_token_key

The user-scoped access token public key.

Type
str

access_token_secret

The user-scoped access token secret.

Type
str

pagination_delay_s

The number of seconds to wait before fetching the next page of results from Twitter's API. It is recommended to set this to at least 1 to avoid rate limits or downstream errors from the API. (Default 5)

Type
int

query(*query: str*, ***kwargs*) → DataFrame

Query the datasource.

Parameters
input (*str*) – The query string.

Returns
The query result.

Return type
pandas.DataFrame

class kpireport_twitter.**TwitterEngagement**(*report:* Report, *datasources:* DatasourceManager, ***kwargs*)

Bases: *Plot*

Display a summary of engagement with an account's own Tweets.

****kwargs**

keyword arguments passed to the parent *Plot* plugin.

17.2 Changelog

17.2.1 0.1.1

Bug Fixes

- Tweets are now displayed in the timezone configured in the report.
- Always use OAuth2 when authenticating, even when using consumer key/secret application-only authentication. OAuth1.0 will throw an error from the TwitterAPI module.

17.2.2 0.1.0

New Features

- A new View `twitter.engagement` is available. The view will show a scatter plot of recent tweets, displaying their like/reply/retweet counts. The highest- liked tweet is called out visually as well.

17.2.3 0.0.1

Prelude

Initial commit.

New Features

- A new `twitter` Datasource is available from this plugin, which allows you to query for Tweets of a given user.

CHAPTER

EIGHTEEN

ARCHITECTURE

18.1 Plugins

KPI Reporter features an extensible and composable plugin architecture that allows you to customize the tool to suit your specific needs. There are three types of plugins:

Datasource plugins
: allow you to interface with specific backend databases and services. A Datasource abstracts the backend behind a simple query interface and handles transforming data into a standard `pandas.DataFrame` object containing the results for interoperability with various Views. *Read more about Datasources*.

View plugins
: allow you to implement visualizations of data, but can also be used to render static information. Typically Views will fetch data from Datasources and then somehow transform or summarize the data visually. Views are expected to output text, but can generate binary "blobs" that can later be linked to or rendered inline. *Read more about Views*.

Output driver plugins
: allow you to change how a report is published, e.g., written to a local disk, uploaded to a cloud storage system, or directly sent as mail. *Read more about Output drivers*.

Plugins are Python modules, and can be installed however you prefer, e.g., with `pip`. Plugins must be installed into the same Python path as KPI Reporter, as that is how they are automatically discovered at runtime.

18.2 Why Dataframes?

The data interchange format between the View and Datasource layers is the `pandas.DataFrame`. There are a few reasons for this:

- The abstraction is already widely used by data scientists and maps well to other concepts developers are exposed to (e.g., arrays, matrices, databases.)
- A wide variety of shapes of data can be expressed.
- The abstraction maps easily to most database storage abstractions (e.g., row-based or column-based tabular data.)
- Documentation is plentiful and kept up-to-date.
- Visualization libraries (e.g., `matplotlib`) have good support for rendering DataFrames built-in.

That said, there are some downsides:

- DataFrames are harder to transform than plan JSON data structures.

- DataFrames don't readily support mixing list and dict-like structures; some normalization is typically needed (see the source for the *Jenkins* for an example.)

The architecture intends to allow for wide interoperability between the View and Datasource layers, so having a well-formed, if limiting, data interface is important enough to override these concerns.

CHAPTER

NINETEEN

PLUGINS

Note: Plugin development guide coming soon.

CHAPTER

TWENTY

KPI REPORTER API

Release
0.1.8

Date
Oct 06, 2022

KPI Reporter API Reference:

20.1 Report

20.1.1 Module: kpireport.report

class kpireport.report.**Content**(*j2: Environment*, *report:* Report)

The rendered report, in a variety of formats.

j2

a Jinja2 context to use for loading and rendering templates.

Type
Jinja2

report

the Report object for the current report.

Type
Report

formats

the list of all formats available for the report. Any formats added via *add_format()* will be reflected here.

Type
List[str]

add_format(*fmt: str*, *blocks: List[*Block*]*)

Render the specified format and add to the output contents.

If a layout file is found for this format, it will be used to render the raw output. If a layout file is not found, there will be no raw output, however, the list of Views will still be stored for the output format. This can be important for output drivers that may not be able to display/send a final rendered report in text, but could still render each view separately. The *Slack* output driver is a good example of this.

The layout file is expected to exist at `./templates/layout/default.{fmt}`, e.g., `./template/layout/default.html` for the HTML format.

Parameters

- **fmt** (*str*) – the output format, e.g., "md" or "html".
- **blocks** (*List[Block]*) – the list of rendered view Blocks

get_blocks(*fmt: str*) → List[*Block*]

Get the rendered views for the given format.

Parameters
fmt (*str*) – the desired output format.

Returns
the list of View Blocks rendered under that format.

Return type
List[*Block*]

get_format(*fmt: str*) → Optional[str]

Get the rendered string for the given format.

Parameters
fmt (*str*) – the desired output format.

Returns
the rendered content for the given format, if any.

Return type
Optional[str]

class kpireport.report.**Report**(*title=None*, *interval_days=None*, *start_date: Optional[datetime] = None*, *end_date: Optional[datetime] = None*, *timezone=None*, *theme=None*)

The report object.

Note: This class is not meant to be instantiated directly; instead, use the *ReportFactory* class to generate an instance.

---

title

the report title.

Type
str

interval_days

number of days.

Type
int

start_date

the start date.

Type
dateobj

end_date

the end date.

Type
dateobj

timezone

the timezone name. Defaults to the system timezone.

Type
str

theme

the report Theme.

Type
Theme

class kpireport.report.**ReportFactory**(*config*)

A factory class for building and executing an entire report.

Once you have a report parsed from its YAML *Configuration file*, you can execute the report like so:

```
ReportFactory(conf).create()
```

config

the (parsed) configuration YAML file.

Type
dict

supported_formats

the output formats that any report can target.

Type
List[str]

create()

Render all Views in the report and output using the output driver.

Important: This will send the report using all configured output drivers! Disable any output drivers you don't wish to send to during testing.

class kpireport.report.**Theme**(*num_columns=6*, *column_width=86*, *padding_width=20*, *theme_dir=None*, *ui_colors=None*, *error_colors=None*, *success_colors=None*, *series_colors=None*, *heading_font=None*)

The report theme options.

num_columns

the number of columns in the report grid. (Default 6)

Type
int

column_width

the width of each column, in pixels. (Default 86)

Type
int

padding_width

the width of the horizontal padding at the report edges (Default 20)

Type
int

`theme_dir`

a directory where additional templates can be found. These templates will override the default templates of the same name, and can be used to alter the overall report appearance.

Type
str

`ui_colors`

a list of user interface colors. This is expected to be a 5-tuple of (text color, lighter text color, dark bg color, bg accent, bg color)

Type
List[str]

`error_colors`

a list of error colors used when views render an error vs. success state. This is expected to be a 2-tuple of (dark, light).

Type
List[str]

`success_colors`

a list of success colors used when views render an error vs. success state. This is expected to be a 2-tuple of (dark, light).

Type
List[str]

`series_colors`

a list of series colors. There can be as many or as few series colors in the theme; you just want to ensure you can handle whatever needs you have for plotting or displaying data in charts or graphs such that series can be identified clearly.

Type
List[str]

`heading_font`

A CSS font-family declaration, which will define how the headings are styled. (Default "Helvetica, Arial, sans-serif"). Note that due to Jinja escaping rules, this does not like embedded quotes. Quotes are *not* required even when a typeface has a space in the name, so they are safe to simply omit.

Type
str

20.2 Datasource

A Datasource is responsible for taking a query input string and returning its result in the form of a `pandas.DataFrame` instance. One instance of a Datasource may be used by multiple `View` instances.

Datasources are only initialized when they are expiclitly declared within the report configuration. When the Datasource is initialized, any arguments included in the report configuration are passed as keyword arguments to its *`Datasource.init()`* function (**Note**: this is *not* the same as Python's built-in `__init__()`.) Each Datasource is required to provide both *`Datasource.init()`*, which is responsible for setting up the Datasource with any additional state, and

Datasource.query(), which provides the mechanism to execute a given query and return a pandas.DataFrame with the results back to the caller.

To create your own Datasource, it is simplest to extend the *Datasource* class, though this is not required. In is required to return a pandas.DataFrame instance, as this is the API contract with the View layer; it allows Views to use a variety of Datasources as seamlessly as possible.

20.2.1 Example: a custom Datasource

This Datasource will interact with a HTTP/JSON API as its backing data store. The input query string is passed to the API as a `query` parameter. The JSON result is parsed into a pandas.DataFrame via pandas.DataFrame.from_records().

```
import pandas as pd
from kpireport.datasource import Datasource
import requests

class HTTPDatasource(Datasource):
    def init(self, api_host=None):
        if not api_host:
            raise ValueError("'api_host' is required")
        self.api_host = api_host

    def query(self, input):
        res = requests.get(self.api_host, params=dict(query=input))
        return pd.DataFrame.from_records(res.json())
```

As with all plugins, your plugin should register itself as an entry_point under the namespace `kpireport.datasource`. We name it `http` in this example. When your plugin is installed alongside the kpireport package, it should be automatically loaded for use when the report runs.

```
[options:entry_points]
kpireport.datasource =
    http = custom_module:HTTPDatasource
```

You can configure your Datasource in a report by declaring it in the `datasources` section. The `plugin` name must match the entry_point name (here, `http`.) Any arguments passed in the `args` key are passed to *Datasource.init()* on instantiation as keyword arguments.

```
datasources:
  custom_datasource:
    plugin: http
    args:
      api_host: https://api.example.com/v1/search
```

You can name the Datasource as you wish (here, `custom_datasource`); Views can invoke your Datasource via this ID.

20.2.2 Example: an RPC Datasource

Instead of passing the query input to another service/database, it is possible to implement your own Datasource that provides an RPC-like interface. This can allow you to custom-tailor your parsing and transformation of the result returned by the backing service, as well as create more complex "queries" that compose multiple calls to the backing service. You could even front multiple backing services as a single Datasource interface. Here is a fully-formed example:

```
from datetime import datetime, timedelta
import pandas as pd
from kpireport.datasource import Datasource
import requests

class RPCDatasource(Datasource):
    def init(self, users_api_host=None, activity_api_host=None):
        if not (users_api_host and activity_api_host):
            raise ValueError((
              "Both 'users_api_host' and 'activity_api_host' "
              "are required."))
        self.users_api_host = users_api_host
        self.activity_api_host = activity_api_host

    def query(self, input):
        """
        Treats the "input" parameter as the name of a separate function
        on the Datasource to invoke.
        """
        fn = getattr(self, input)
        if not (fn and iscallable(fn)):
            raise ValueError(f"Query '{input}' is not supported")
        return fn()

    def get_all_users(self):
        """
        Return all users, regardless of their activity status.
        """
        users = requests.get(f"{self.users_api_host}/users")
        return pd.DataFrame.from_records(users)

    def get_active_users(self):
        """
        Return only the users active within the last month.
        """
        users = requests.get(f"{self.users_api_host}/users")
        user_ids = [u["id"] for u in users.json()]
        last_active_at = requests.get(
          f"{self.activity_api_host}/last_activity",
          params=dict(
            user_ids=user_ids.join(",")
          )
        )
        one_month_ago = datetime.now() - timedelta(months=1)
        active_in_last_month = [
```

(continues on next page)

(continued from previous page)

```
          a["user_id"]
          for a in last_active_at.json()
          if datetime.fromisoformat(a["last_active_at"]) > one_month_ago
        ]
        return pd.DataFrame.from_records([
          u for u in users if u["id"] in active_in_last_month])

    def total_active_users(self):
        """
        Return just the total number of active users, not a full set
        of columnar data.
        """
        return self.get_active_users().count()
```

From within a View, you could then invoke your Datasource like this (we have given the Datasource the ID `users` here.)

```
def render_html(self, j2):
    active_users = self.datasources.query("users", "get_active_users")
    # Do something with active users
```

Or, using an existing plugin, like `kpireport.plugins.plot.SingleStat`, which can be configured with just the report configuration:

```
views:
  active_users:
    title: Users active in last month
    plugin: single_stat
    args:
      datasource: users
      query: total_active_users
```

20.2.3 Module: `kpireport.datasource`

class kpireport.datasource.**Datasource**(*report:* Report, ***kwargs*)

Parameters

- **report** (*kpireport.report.Report*) – the Report object.
- **id** (*str*) – the Datasource ID declared in the report configuration.
- ****kwargs** – Additional datasource parameters, declared as `args` in the report configuration.

abstract init(***kwargs*)

Initialize the datasource from the report configuration.

Parameters

****kwargs** – Arbitrary keyword arguments, declared as `args` in the report configuration.

abstract query(*input: str*) → DataFrame

Query the datasource.

Parameters

input (*str*) – The query string.

Returns
The query result.

Return type
pandas.DataFrame

exception kpireport.datasource.**DatasourceError**

A base class for errors originating from the datasource.

class kpireport.datasource.**DatasourceManager**(*report: Report*, *config: Dict*, *extension_manager=None*)

exc_class

alias of *DatasourceError*

20.3 View

20.3.1 Module: kpireport.view

class kpireport.view.**Blob**(*id: str*, *content: str*, *mime_type: 'Optional[str]'*, *title: 'Optional[str]'*)

class kpireport.view.**Block**(*id: str*, *title: str*, *description: str*, *cols: int*, *blobs: 'List[Blob]'*, *output: str*, *tags: 'List[str]'*)

class kpireport.view.**View**(*report: Report*, *datasources: DatasourceManager*, ***kwargs*)

The view

exception kpireport.view.**ViewException**

class kpireport.view.**ViewManager**(*datasource_manager*, *report*, *config*, *extension_manager=None*)

exc_class

alias of *ViewException*

kpireport.view.**make_render_env**(*env: Environment*, *view: View*, *output_driver: OutputDriver*, *fmt: str*)

Create a Jinja environment for rendering the view.

The environment is specific to the view, output driver, and output format, and should only be used for that combination of entities. As such, it is best to create this environment just before rendering the view.

20.4 Output driver

20.4.1 Module: kpireport.output

class kpireport.output.**OutputDriver**(*report: Report*, ***kwargs*)

report

the current report.

Type
Report

id

the Output driver ID declared in the report configuration.

Type
str

supported_formats

a list of output formats supported by this driver. Defaults to `["md", "html"]`.

Type
List[str]

can_render(*fmt: str*) → bool

Determine if this driver supports a given output format.

Parameters
fmt (*str*) – the desired output format.

Returns
whether the output format can be rendered.

Return type
bool

abstract init(***kwargs*)

Initialize the output driver from the report configuration.

Parameters
****kwargs** – Arbitrary keyword arguments, declared as `args` in the report configuration.

render_blob_inline(*blob:* Blob, *fmt=None*)

Render a blob file inline in the report output.

Blobs are typically binary image files; many output formats afford some way of displaying them directly, e.g., in HTML via an <img> tag. Each output driver can define how to render a blob inline. An email output driver may implement some way of attaching the image and referencing it in the mail message, while a HTML file output driver may use an <img> tag and link to the file, or perhaps use a data-uri.

This function is used when invoking the `blob` template filter.

Parameters

- **blob** (*~kpireport.view.Blob*) – the Blob to render.
- **fmt** (*str*) – the output format.

Returns
the rendered Blob output.

Return type
str

abstract render_output(*content:* Content, *blobs*)

Render the report content for the target delivery mechanism.

Parameters

- **content** (*Content*) – the report contents.
- **blobs** (*List[Blob]*) – the blobs generated as part of the report.

exception kpireport.output.**OutputDriverError**

class kpireport.output.**OutputDriverManager**(*report:* Report, *config: Dict*, *extension_manager=None*)

exc_class

alias of *OutputDriverError*

20.5 Changelog

20.5.1 0.1.8

Bug Fixes

- If the "templates" folder inside a view plugin is missing, an error will no longer be thrown when initializing the PackageLoader for the plugin.

20.5.2 0.1.7

Bug Fixes

- Plugin template resolution now supports walking up class hierachies. If a concrete class extends an existing plugin, templates will be first searched for in the concrete class' module, then its parent class, and so on. Before, only the concrete class and the base View module were included in the lookup paths, so parent class templates would be ignored.

20.5.3 0.1.6

Bug Fixes

- The default values for `start_date` and `end_date` will now be timezone-aware and derive their timezones from the local timezone on the system running the report process. A `timezone` report argument is also now available, which can be a timezone strong, and will override the timezone for automatically-generated values for the start and end date.

20.5.4 0.1.5

Bug Fixes

- The font-family of the heading font(s) can now be specified via the `heading_font` theme argument. It defaults to a font-stack that renders something that looks like Helvetica and falls back to any sans-serif font installed.

20.5.5 0.1.4

Bug Fixes

- The wrapper margin is now customizable on the Theme via the `padding_width` attribute. It defaults to 20, which was the old value in the built-in theme.

20.5.6 0.1.3

Bug Fixes

- Include report templates in the published PyPI package.

20.5.7 0.1.2

Bug Fixes

- Fixes an issue with rendering the license text when the license was valid or expired.

20.5.8 0.1.1

Bug Fixes

- Fixes error due to missing import when loading configuration files automatically.

20.5.9 0.1.0

New Features

- If no configuration file is provided via the –config-file flag, a `config.yaml` file will be searched for, first in the working directory and then in `/etc/kpireporter`.

20.5.10 0.0.1

Prelude

Initial commit.

New Features

- Initial commit.

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX

R

S

T

U

V

X