Goodtables.io Documentation

Open Knowledge Foundation

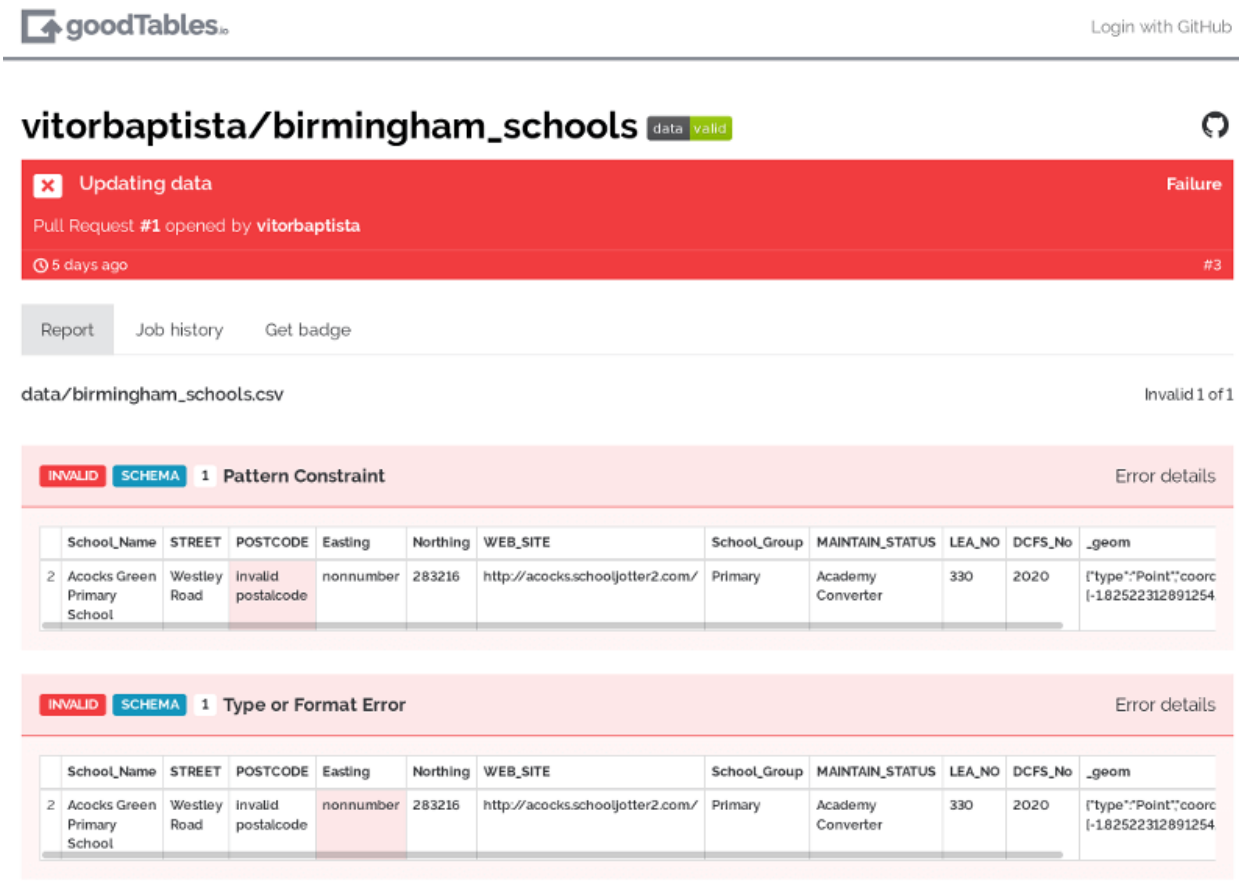
May 22, 2019

Contents

Goodtables increases your confidence in your data by performing a number of checks on it. It is able to validate things like:

- Is my CSV valid?
- Do all rows have the same number of columns?
- Are all dates valid?
- Is there any invalid e-mail?

If any of the checks fail, goodtables generates a report telling you which and where the errors occurred.



Example report with failed validation

By adding goodtables to your data publishing workflow, you'll make sure your data is free from these types of errors.

CHAPTER 1

Features

- **Structural checks**: Ensure that there are no empty rows, no blank headers, etc.
- **Content checks**: Ensure that the values have the correct types ("string", "number", "date", etc.), that their format is valid ("string must be an e-mail"), and that they respect the constraints ("age must be a number greater than 18").
- **Support for multiple tabular formats**: CSV, Excel, LibreOffice, Data Package, etc.
- **Automatically validate data on every update**: Support for data on GitHub and Amazon S3.

CHAPTER 2

Table of Contents

2.1 Getting started

2.1.1 Validating data on GitHub

This is a very short tutorial on using goodtables.io to continuously validate data hosted on GitHub.

Pre-requisites

- Tabular data on a GitHub repository

Instructions

1. Login on goodtables.io using your GitHub account and accept the permissions confirmation.
2. Once we've synchronized your repository list, go to the Manage Sources page and enable the repository with the data you want to validate.
    - If you can't find the repository, try clicking on the Refresh button on the Manage Sources page

Goodtables will then validate all tabular data files (CSV, XLS, XLSX, ODS) and data packages in the repository. These validations will be executed on every change, including pull requests.

Next steps

- Add a badge to your README to display your data validation status. The instructions are on the "Get badge" tab in the data report page.
- Write a table schema to validate the contents of your data
- Configure which files are validated and how

2.1.2 Validating data on Amazon S3

This is a very short tutorial on using goodtables.io to continuously validate data hosted on Amazon S3.

Pre-requisites

- A GitHub login
- An Amazon S3 login

Instructions

Setting up Amazon S3 bucket and read-only user

1. Create a bucket on S3 to hold your data
   - Create the bucket on the `us-west-2` region. It's a current limitation of goodtables.io that we're working to fix.
2. Create a new IAM user. This user will be used by goodtables.io to read your bucket.
   - Make sure you take note of the AWS Access Key ID, AWS Secret Access Key, and the User ARN.
3. Go to your bucket's overview page, click on the `Permissions` tab, and find the `Bucket Policy` link. We need the permissions:
   - *s3:ListBucket*: To list the bucket's contents
   - *s3:GetObject*: To read the bucket's files
   - *s3:GetBucketPolicy*, *s3:PutBucketPolicy*, *s3:GetBucketLocation*, and *s3:PutBucketNotification*: To set up the AWS Lambda functions that notifies goodtables.io when a new file is added

The final bucket policy should look like:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM_USER_ARN"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketPolicy",
                "s3:PutBucketPolicy",
                "s3:PutBucketNotification"
            ],
            "Resource": "arn:aws:s3:::BUCKET_NAME"
        },
        {
            "Sid": "statement2",
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM_USER_ARN"
```

(continues on next page)

(continued from previous page)

```
            },
            "Action": ["s3:GetObject"],
            "Resource": "arn:aws:s3:::BUCKET_NAME/*"
        }
    ]
}
```

With your IAM User ARN and Bucket Name substituting the `IAM_USER_ARN` and `BUCKET_NAME`.

Setting up goodtables.io

1. Login on goodtables.io using your GitHub account.
2. Go to the Manage Sources page, click on the `Amazon` tab, and on the plus sign on the right of the Filter input.
3. Fill in the `Access Key Id`, `Secret Access Key` and `Bucket Name` with the IAM User and bucket you just created in the previous section.

We're all set. Goodtables will automatically validate whenever a file is added or modified in the bucket. You can now upload data to your bucket and goodtables will automatically validate any tabular files (CSV, XLS, ODS, ...) and tabular data packages.

Next steps

- Write a table schema to validate the contents of your data
- Configure which files are validated and how

2.1.3 Describing your data schema

Without knowledge of the data structure, goodtables is only able to check if the structure of the data is valid. For example, that all rows have the same number of columns, that there are no blank headers, etc. To validate the actual contents, you need to describe the data schema.

The data schema describes what each column should have (strings, numbers, dates), their formats (this string should be an e-mail), and constraints (numbers on age column should be bigger than 18). You can think of it as a kind of data dictionary. The best way to describe the data schema is by writing a data package.

Instructions

On the root folder of your data, create a `datapackage.json` file with the contents:

```
{
  "name": "my-dataset",
  "title": "My dataset",
  "resources": [
    {
      "name": "my-data",
      "path": "data/data.csv"
    }
  ]
}
```

This is the simplest tabular data package we can create. Let's see how our data looks like so we can write the table schema for it.

date	from	to	amount
2017-01-01	Jane	John	1000
2017-01-15	Jane	Paul	500
2017-02-03	John	Jane	2000

A table schema has three parts: the data type ("string", "number", "date"), the data format ("e-mail", "URI", "ISO date"), and the constraints ("number must be above 18"). Not all columns will have all three parts.

In our data, we have the following columns:

column	type	format	constraints
date	date	YYYY-MM-DD	
from	string		
to	string		
amount	numeric		Greater or equal to 0

Writing this as a table schema in our data package, we have:

```
{
  "name": "my-dataset",
  "title": "My dataset",
  "resources": [
    {
      "name": "my-data",
      "path": "data/data.csv",
      "schema": {
        "fields": [
          {
            "name": "date",
            "type": "date",
            "description": "The transaction date"
          },
          {
            "name": "from",
            "type": "string",
            "description": "Payer"
          },
          {
            "name": "to",
            "type": "string",
            "description": "Payee"
          },
          {
            "name": "amount",
            "type": "number",
            "description": "Transaction value in Euros",
            "constraints": {
              "minimum": 0
            }
          }
        ]
      }
    }
```

(continues on next page)

(continued from previous page)

```
  ]
}
```

Note that we didn't have to define the date format explicitly, as the default format is YYYY-MM-DD.

You can find all supported data types, formats and constraints in the Table Schema specification.

2.2 Configuration

Goodtables.io is configured via a `goodtables.yml` file in the root directory. For example, you can define:

- Which files goodtables should validate
- Which spreadsheet page should be validated
- What delimiter your CSV file uses (e.g. `;`)
- Which validation checks should be executed

The rest of this page is divided in sections on common things you want to change. For the full reference, check the goodtables.yml file reference.

2.2.1 Defining the files to validate

By default goodtables validates all files with extension CSV, ODS, XLS, or XLSX, and all files named `datapackage.json`.

You can overwrite the default files in `goodtables.yml`:

```
files:
  - source: data1.csv
    schema: schema1.json
  - source: data2.xls
    schema: schema2.json
```

Alternatively, you can define a pattern like:

```
files: '*.csv'
```

You can also configure how the file is loaded using the options:

Option	Description
format	The file format (csv, xls, ...)
encoding	The file encoding (utf-8, ...)
skip_rows	Either the number of rows to skip, or an array of strings (e.g. `#`, `//`, ...). Rows that begin with any of the strings will be ignored.

2.2.2 Validating data packages

By default goodtables validates all files named `datapackage.json`.

You can overwrite this default in `goodtables.yml`:

```
datapackages:
  - report1/datapackage.json
  - report2/datapackage.json
```

2.2.3 Validating CSV files with custom dialects

You can configure how the CSV file is loaded by adding one of the following options on `goodtables.yml`:

```
files:
  - source: data.csv
    delimiter: ;
    doublequote: True
    escapechar: \
    lineterminator: \r\n
    quotechar: "
```

The entire list of options can be found on the Python CSV formatting reference.

2.2.4 Defining the spreadsheet page to validate

By default goodtables validates the first sheet of a spreadsheet.

You can overwrite the default sheet in `goodtables.yml`:

```
files:
  - source: data.xlsx
    sheet: 3
```

2.2.5 Changing the limit of rows to validate

By default goodtables validates at most 1,000 rows. You can change it in `goodtables.yml`:

```
settings:
  row_limit: 2000
```

2.2.6 Defining which validation checks are executed

By default goodtables runs all validation checks. You can customize which checks are executed in `goodtables.yml`:

```
settings:
  checks:
    # You can pass check types
    - structure
    - schema
    # ... or individual checks
    - blank-header
    - duplicate-row
    - missing-value
  skip_checks:
    # You can also skip individual checks
    - minimum-constraint
```

Note that if you use the `checks` setting, you have to define all checks you want to be used. Because of this, we recommend using `skip_checks` instead.

The list of validation checks can be found on the goodtables-py documentation.

2.2.7 Automatically inferring the schema

By default goodtables does not infer the data schema. You can enable inferring in `goodtables.yml`:

```
settings:
  infer_schema: True
  infer_fields: True
```

Goodtables will infer the schema of all files and columns that don't have an explicit schema.

2.3 The goodtables.yml

```
files:
  - source: data.csv
    schema: schema.json
    format: csv
    encoding: utf-8
    skip_rows: 3
    delimiter: ';'

  - source: data.xls
    format: xls
    sheet: 4

datapackages:
  - 'datapackage.json'

settings:
  checks:
    # You can pass check types
    - structure
    - schema
    # ...or individual checks
    - no-headers
    - blank-headers
  skip_checks:
    - duplicate-lines
  error_limit: 1
  table_limit: 1
  row_limit: 5000
  infer_schema: True
  infer_fields: True
  order_fields: True
```

2.4 API

Goodtables.io provides a HTTP API that allows you to do programatically everything that can be done via the web interface. All the endpoints documented below live in https://goodtables.io/api/, and require that you send your autho-

rization token via the `Authorization` header.

2.4.1 Endpoints

`GET /source`
Returns list of user's sources

Status Codes

- 200 OK – Success

`POST /source`
Creates a new source. By default, the *integration_name* is "api"

Query Parameters

- **`name`** (*`string`*) – The source name
- **`private`** (*`boolean`*) – Controls if the created source is private or not

Status Codes

- 200 OK – Success

`GET /source/{source_id}`
Gets the source ID

Parameters

- **`source_id`** (*`string`*) – The source id

Status Codes

- 200 OK – Success
- 404 Not Found – Source not found
- 403 Forbidden – You don't have permissions to access source

`GET /source/{source_id}/job`
Returns list of source's jobs

Parameters

- **`source_id`** (*`string`*) – The job source's id

Status Codes

- 200 OK – Success
- 404 Not Found – Source not found
- 403 Forbidden – You don't have permissions to access source

`POST /source/{source_id}/job`

Query Parameters

- **`data`** (*`string`*) –

Status Codes

- 200 OK – Success
- 404 Not Found – Source not found
- 403 Forbidden – You don't have permissions to access source, or the source integration_name isn't API, or there are too many files.

- 400 Bad Request – Malformed, invalid, or missing job configuration

GET /source/{source_id}/job/{job_id}
Returns a specific source's job

Parameters

- **source_id** (*string*) – The job source's id
- **job_id** (*string*) – The job id

Status Codes

- 200 OK – Success
- 404 Not Found – Job not found
- 403 Forbidden – You don't have permissions to access source

GET /token
Return user's API token list

Status Codes

- 200 OK – Success

POST /token
Create new API token

Query Parameters

- **description** (*string*) – Token description

Status Codes

- 200 OK – Success

DELETE /token/{token_id}
Delete API Token

Parameters

- **token_id** (*string*) – The Token id

Status Codes

- 200 OK – Success
- 404 Not Found – Token not found

HTTP Routing Table

/source

/token