WhyR? 2019 Conference
28.09.2019

Google PageSpeed with R

(or how to automate your web performance audits)

LESZEK SIEMIŃSKI
RINGIER AXEL SPRINGER POLSKA

About me

Data analyst in
Monetization Team of Video
Department

Former SEO specialist

Working in digital marketing
agencies since 2014

Most important tools:

- R
- SQL
- Google Analytics
- SEO tools & knowledge

ringier
axel springer

Top publisher in Poland

Owner of Onet, Newsweek,
Forbes, Fakt, VOD.pl

Responsibilites:

- Analysing website traffic
- Analysing monetization data
- Improving traffic & website health
- Improving internal data flow

SEO = Search Engine Optimization

Objective:

- increasing domain's visibility in Search Engine Results Pages

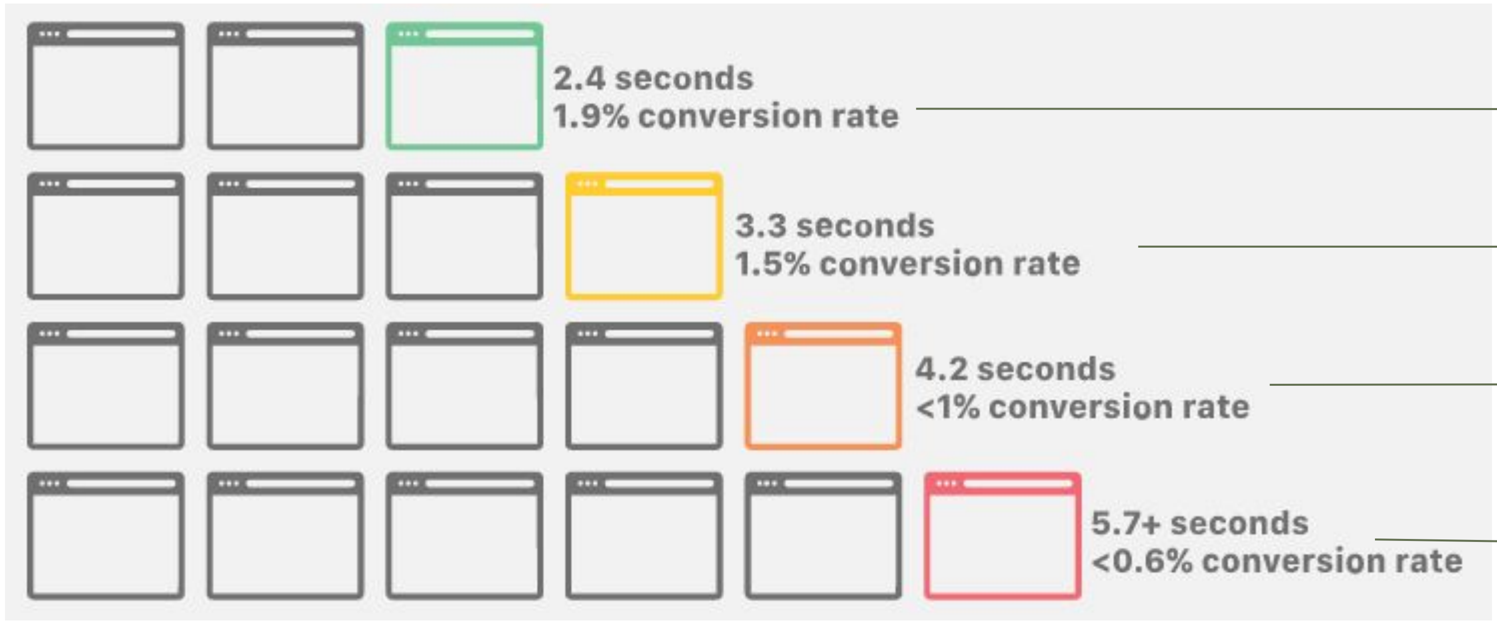
Most important activities:

- technical optimisation (i.e. **improving loading performance**)
- content creation & optimization
- optimization & acquisition of external backlinks

Why should we even care about improving page's loading time?

1. Better user experience
2. More sales
3. Lesser costs

Imagine that 1 000 users come to the shopping page...



→ 19 users will buy

→ 15 users will buy

→ less than 10 users will buy

→ less than 6 users will buy

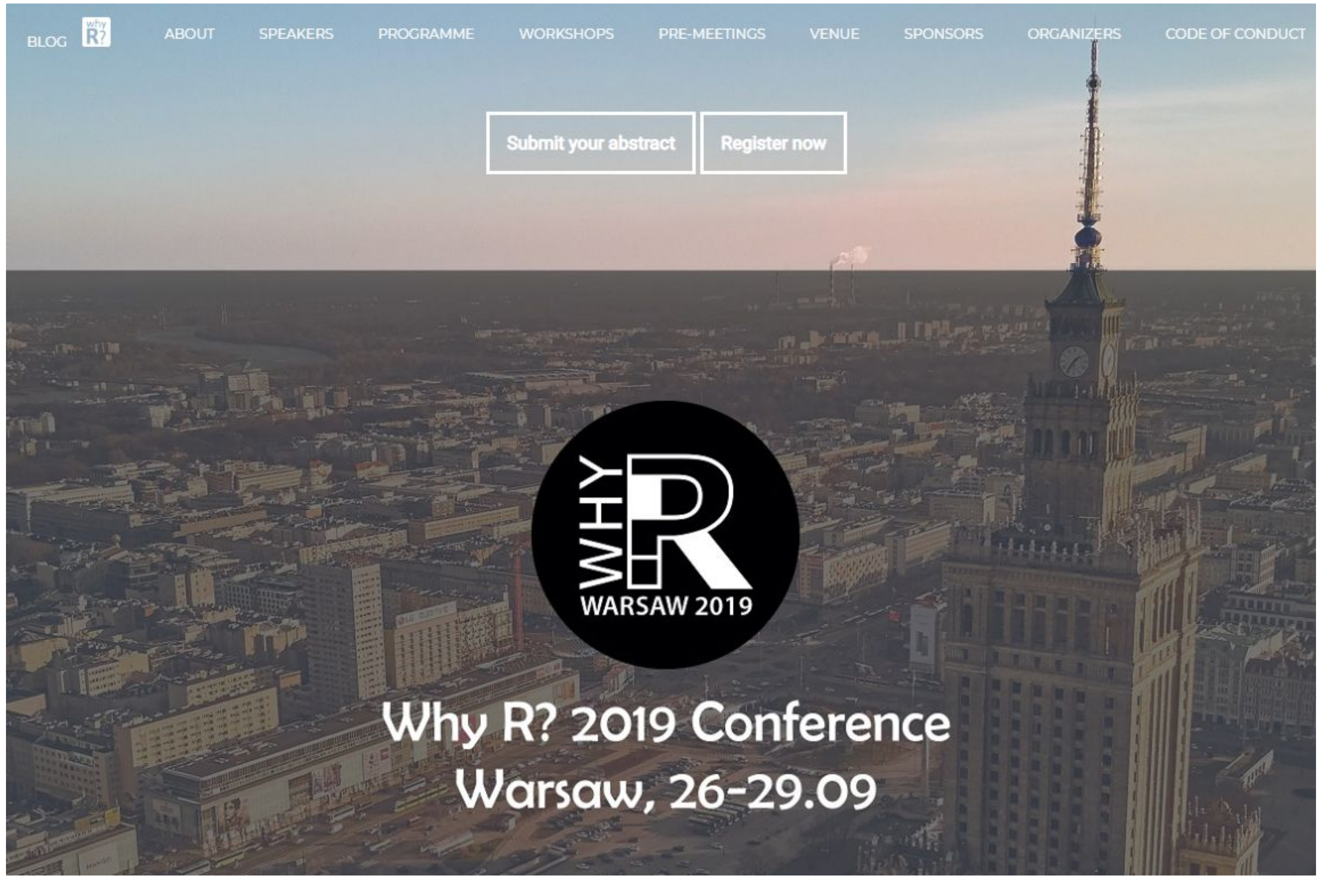
Source: https://www.cloudflare.com/learning/performance/more/website-performance-conversion-rates/

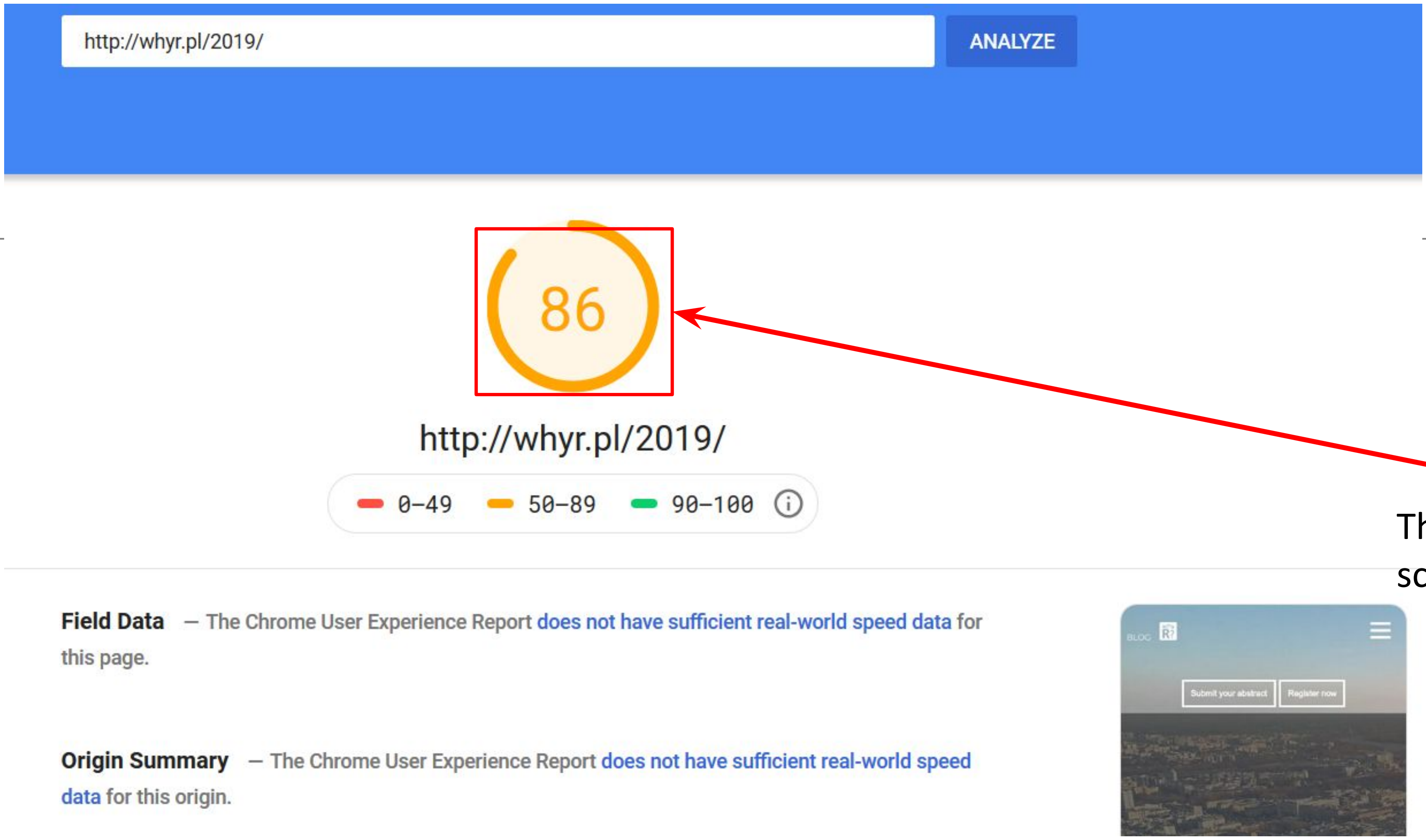
Working in optimization: Google tools

External tools for checking URL’s performance & load time:

- **Google PageSpeed Insights**
- Google Test My Site
- Google Lighthouse
- Google CrUx
- GTmetrix (not Google!)
- other

Let’s take a closer look for a report’s example:





http://whyr.pl/2019/

ANALYZE

86

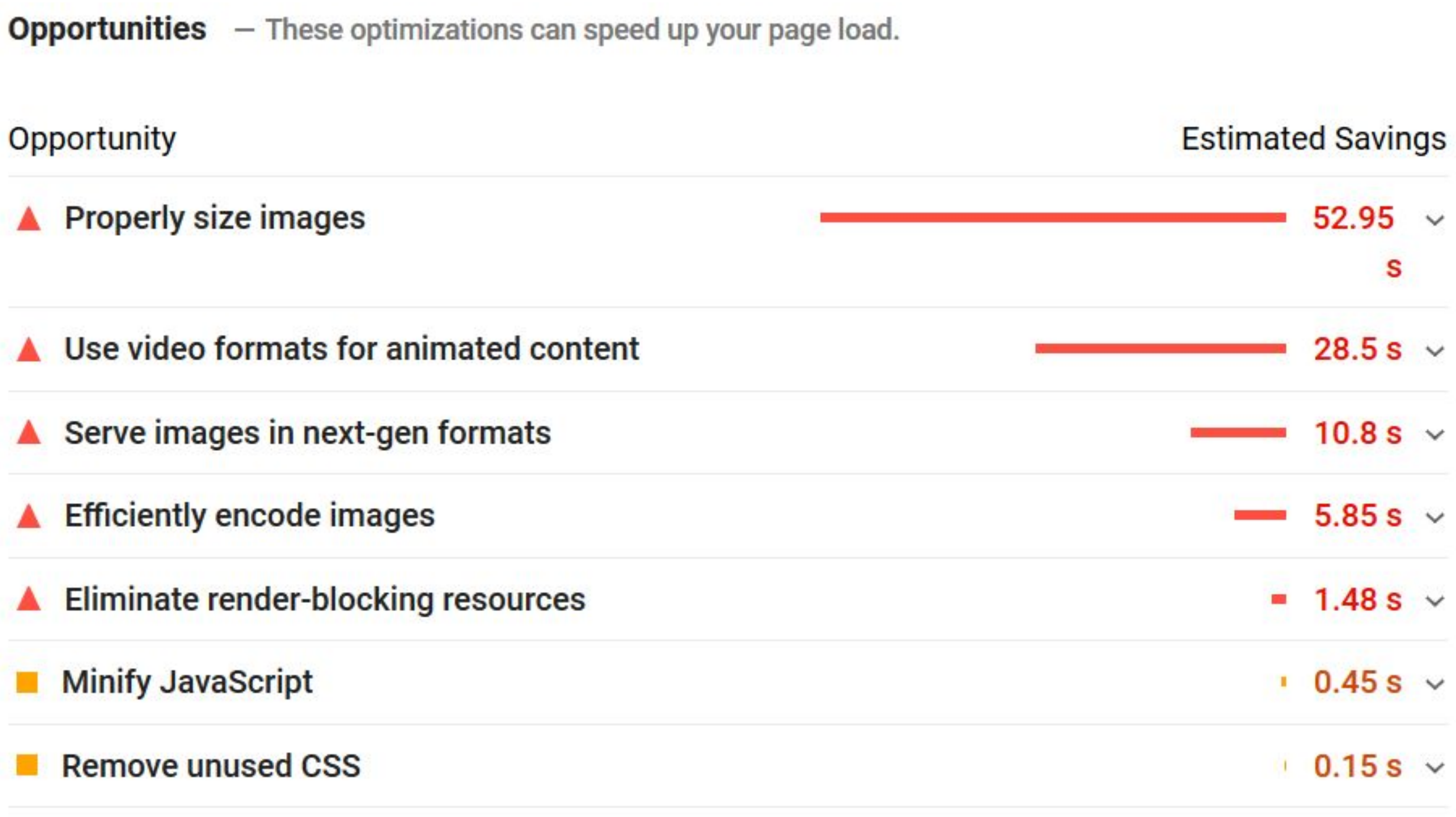
http://whyr.pl/2019/

🟥 0–49 🟧 50–89 🟩 90–100

Field Data — The Chrome User Experience Report does not have sufficient real-world speed data for this page.

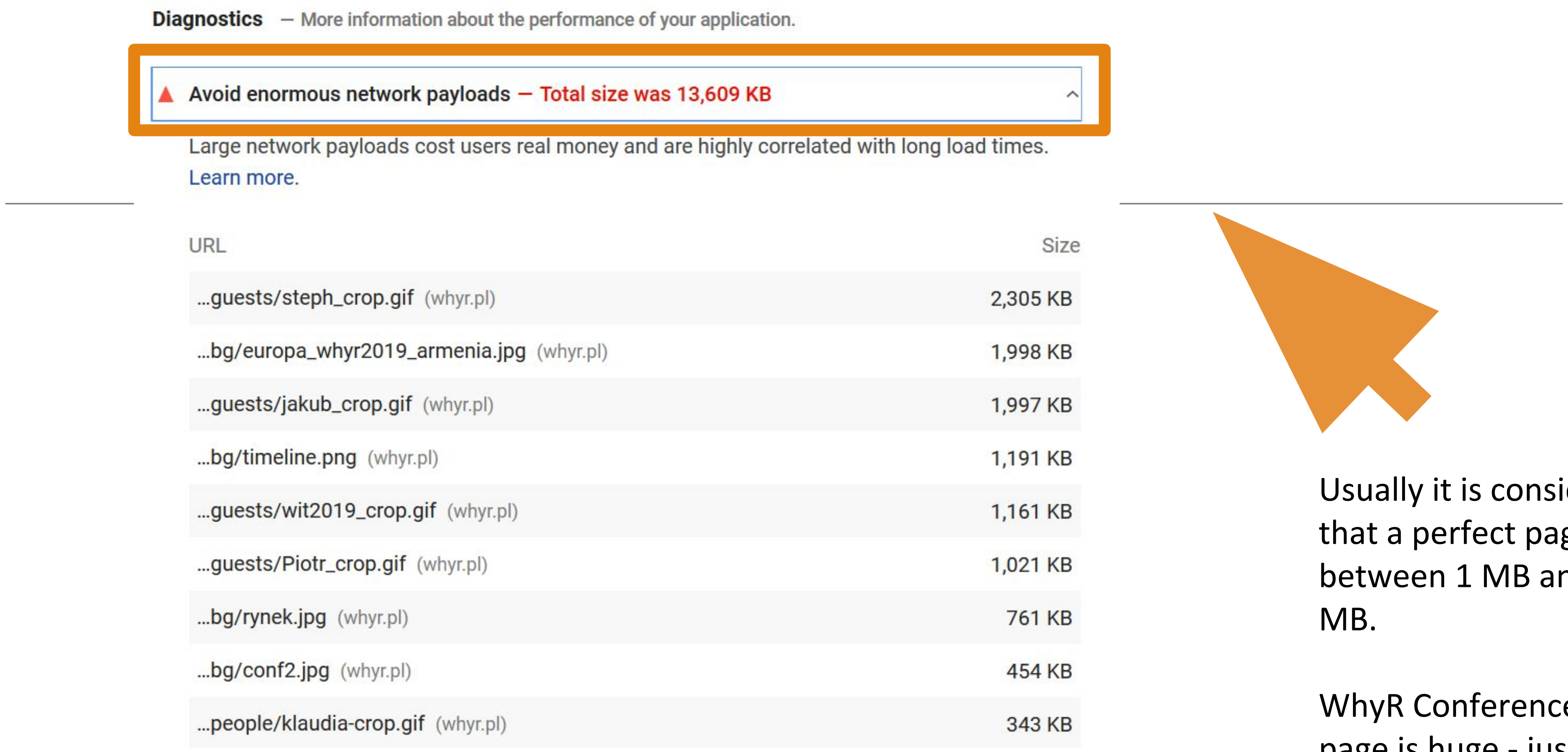
Origin Summary — The Chrome User Experience Report does not have sufficient real-world speed data for this origin.

That's a good score!


Opportunities — These optimizations can speed up your page load.

Opportunity	Estimated Savings
▲ Properly size images	52.95 s
▲ Use video formats for animated content	28.5 s
▲ Serve images in next-gen formats	10.8 s
▲ Efficiently encode images	5.85 s
▲ Eliminate render-blocking resources	1.48 s
■ Minify JavaScript	0.45 s
■ Remove unused CSS	0.15 s

… but the loading time of images is terribly long and may decrease user experience (especially on mobile)


Diagnostics — More information about the performance of your application.

▲ Avoid enormous network payloads — Total size was 13,609 KB

Large network payloads cost users real money and are highly correlated with long load times. Learn more.

URL	Size
...guests/steph_crop.gif (whyr.pl)	2,305 KB
...bg/europa_whyr2019_armenia.jpg (whyr.pl)	1,998 KB
...guests/jakub_crop.gif (whyr.pl)	1,997 KB
...bg/timeline.png (whyr.pl)	1,191 KB
...guests/wit2019_crop.gif (whyr.pl)	1,161 KB
...guests/Piotr_crop.gif (whyr.pl)	1,021 KB
...bg/rynek.jpg (whyr.pl)	761 KB
...bg/conf2.jpg (whyr.pl)	454 KB
...people/klaudia-crop.gif (whyr.pl)	343 KB

Usually it is considered that a perfect page is between 1 MB and 2 MB.

WhyR Conference home page is huge - just the images are ~13 MB (!)

Problem: How can we increase efficiency of web performance testing?

- Problems with traditional use of PageSpeed Insights (paste the URL, wait for the output, collect the report):
  - slow
  - impossible to automate this process for a non-technical specialist

- Solutions
  - slow -> we can’t really speed it up, so let’s at least schedule regular tests earlier
  - impossible to automate this process for a non-technical specialist -> let’s request the reports throught the **API***

* “Web APIs are the **defined interfaces** through which interactions happen between an enterprise and applications that use its assets, which also is a Service Level Agreement (SLA) to specify the functional provider and expose the service path or URL for its API users.” *"API-fication" (PDF). www.hcltech.com. 08.2014.*

There are some existing solutions in R...

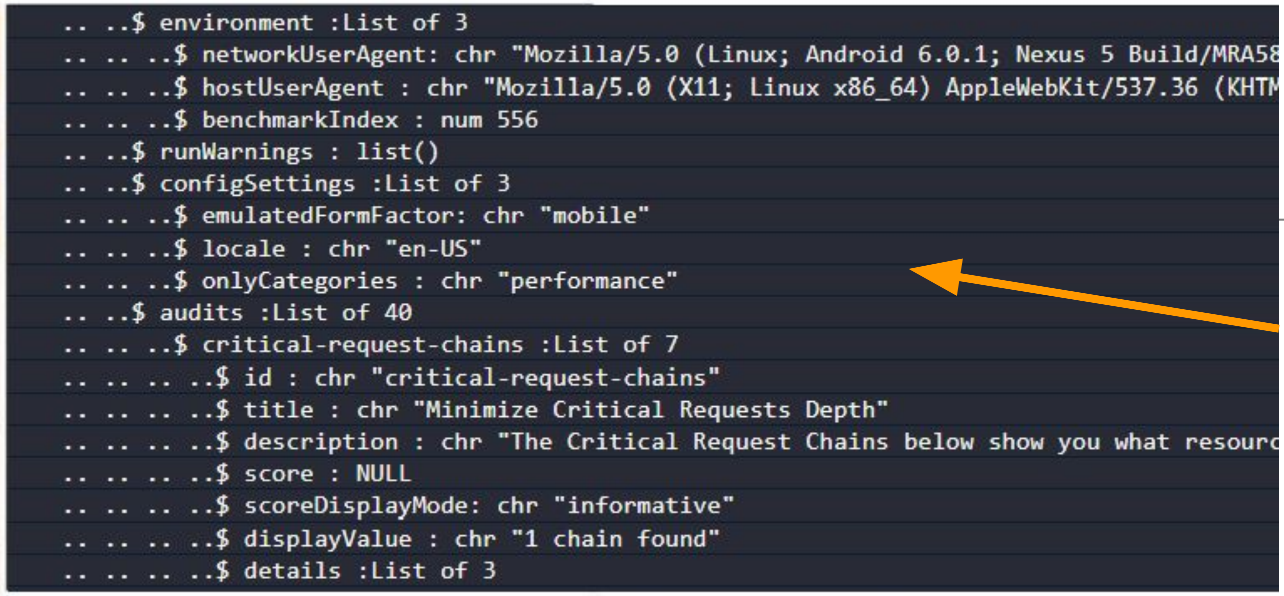
- “googlePageSpeedR” - https://github.com/Phippsy/googlePageSpeedR
- “gpagespeed” - https://github.com/simitpatel/gpagespeed
- “pagespeed” - https://github.com/mhairi/pagespeed

… but

- they all use old API version (4) instead of the newest (5)
  - the 4th version is “classic” PageSpeed, but 5th contains more data from the Lighthouse!
- they offer little data and no errors descriptions/examples
- … and the loading time is long
  - in fact we download ALL the data but we can access only small portion of it
- they are not resistant for 404 errors in input
  - if your input has 10k URLs and your output has 3k URLs, you don’t know which URLs failed
- not really „vectorized” - if you want it in a loop, you must do it yourself
- the only one package that is “vectorized” loses errors
- no documentation

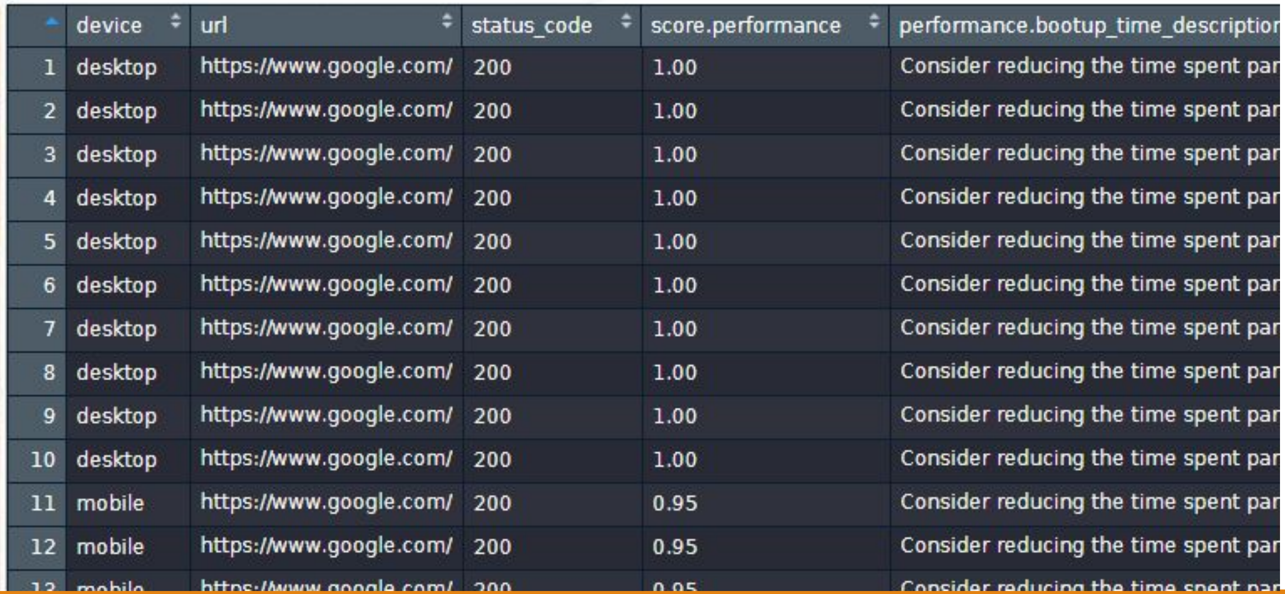
And this is not really their creators fault!

What is the reason for all those issues?



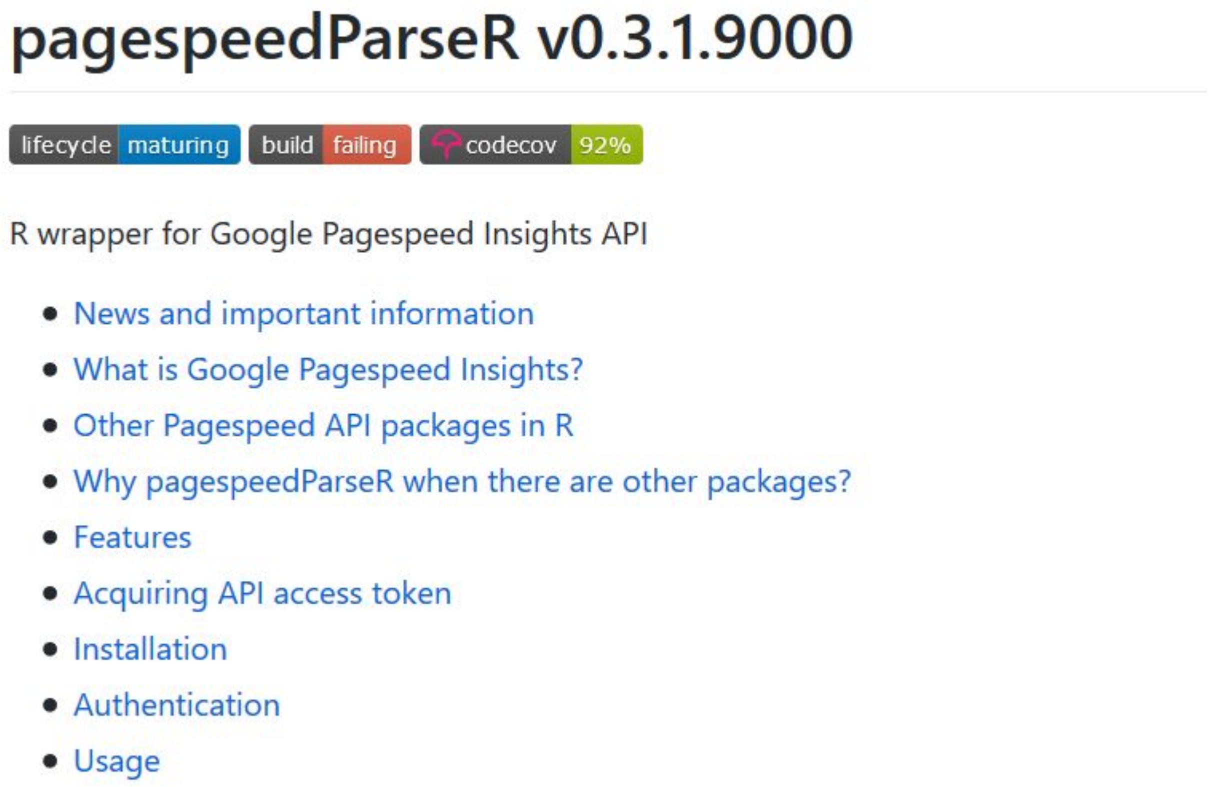
```
.. ..$ environment :List of 3
.. .. ..$ networkUserAgent: chr "Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58
.. .. ..$ hostUserAgent : chr "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTM
.. .. ..$ benchmarkIndex : num 556
.. ..$ runWarnings : list()
.. ..$ configSettings :List of 3
.. .. ..$ emulatedFormFactor: chr "mobile"
.. .. ..$ locale : chr "en-US"
.. .. ..$ onlyCategories : chr "performance"
.. ..$ audits :List of 40
.. .. ..$ critical-request-chains :List of 7
.. .. .. ..$ id : chr "critical-request-chains"
.. .. .. ..$ title : chr "Minimize Critical Requests Depth"
.. .. .. ..$ description : chr "The Critical Request Chains below show you what resourc
.. .. .. ..$ score : NULL
.. .. .. ..$ scoreDisplayMode: chr "informative"
.. .. .. ..$ displayValue : chr "1 chain found"
.. .. .. ..$ details :List of 3
```

API returns nested lists...



	device	url	status_code	score.performance	performance.bootup_time_descriptior
1	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
2	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
3	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
4	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
5	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
6	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
7	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
8	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
9	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
10	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent par
11	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent par
12	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent par
13	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent par

Solution: well, do it yourself. So I did!



pagespeedParseR v0.3.1.9000

R wrapper for Google Pagespeed Insights API

- News and important information
- What is Google Pagespeed Insights?
- Other Pagespeed API packages in R
- Why pagespeedParseR when there are other packages?
- Features
- Acquiring API access token
- Installation
- Authentication
- Usage

“pagespeedParseR” package

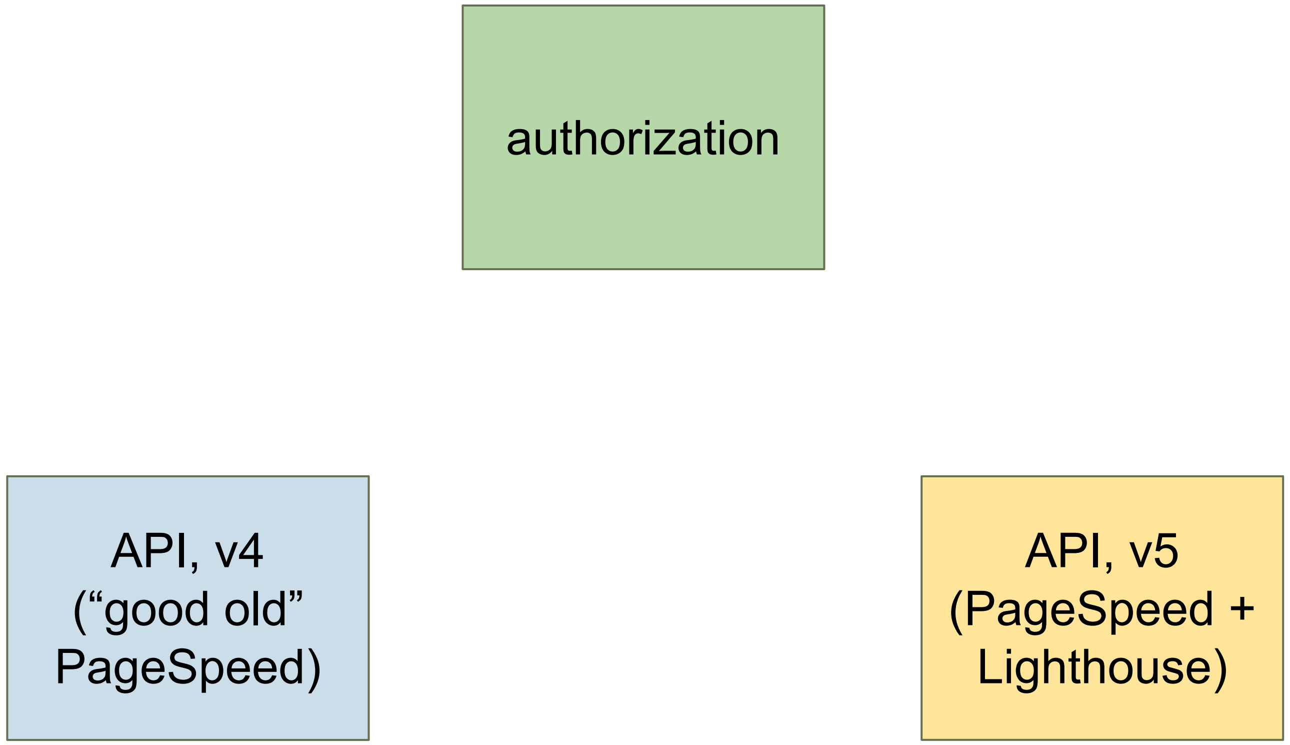
Underlying assumptions:

- I wanted it to let me download everything as a nested list or to download most of the data as a data frame (parsing to data frame on-the-fly)
- I wanted it to let me choose API version (4th or 5th)
- I wanted it to have built-in loops (because I’m lazy)
- I wanted it to keep the information which URL returned an error (non 200 HTTP status) and which error was that
- I wanted it to let me check desktop and mobile in a single call (yes, I know I’m lazy)
- I wanted it to have a simple mechanism that I can use to enforce waiting some time between the calls to keep the API request limits happy ;) (requests/minute)
- I wanted it to have at least acceptable documentation that helps the user

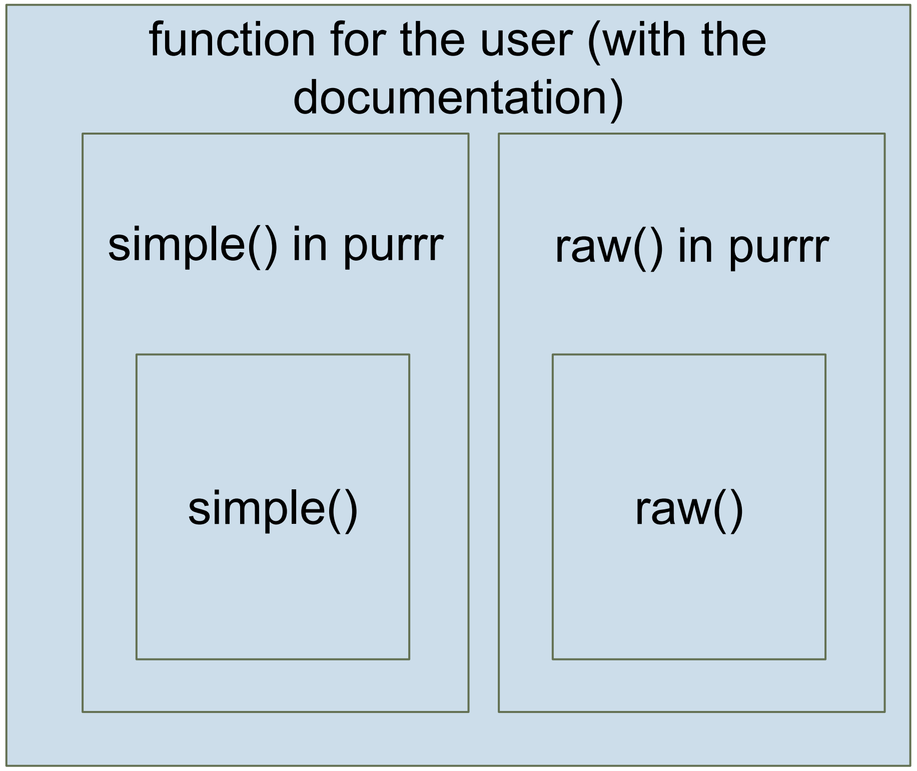
Some problems with PageSpeed output nested lists:

- not stable – no performance error found creates other nested lists that some errors found
- often doesn’t have stable length (number of subelements) – for example redirection errors audit can have up to 20 subelements or more
- some audits generate empty lists when passed, other don’t
- scores mean different things - “score = 1” may mean 100% (perfect grade) in some audits and 1% in the other ones (bad) - the scale is changing and depend on the type of the audit
- Google loves its toys and continually adds more audits or changes exisiting ones

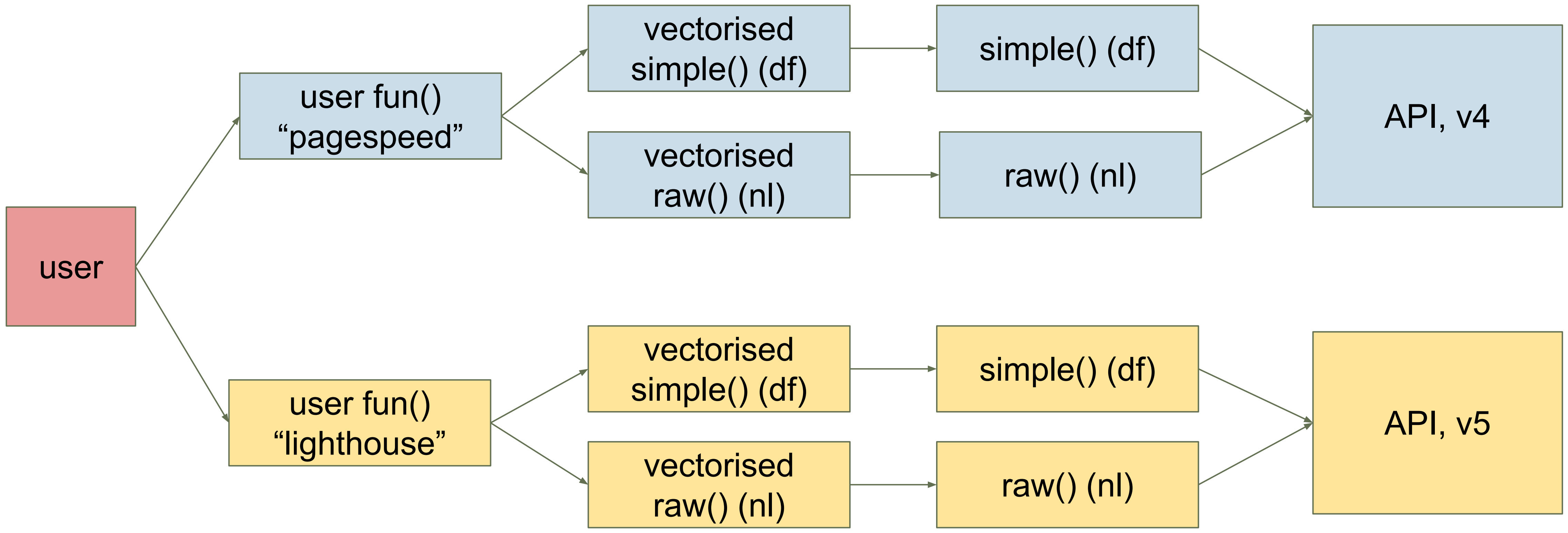
Package architecture: 3 pillars



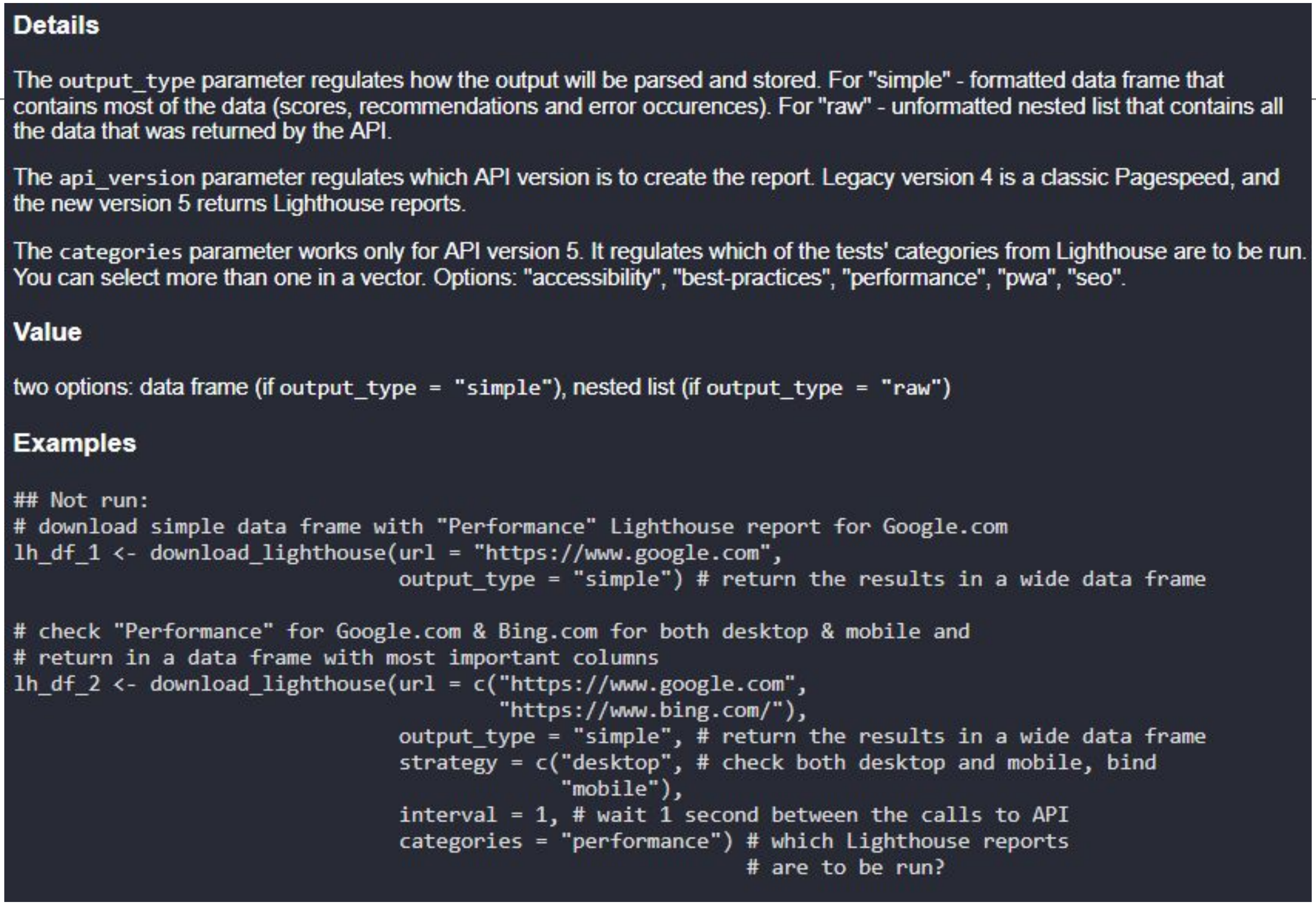
Package architecture: internal functions



Package architecture: communication



Top level functions: documentation ;)



Details

The `output_type` parameter regulates how the output will be parsed and stored. For "simple" - formatted data frame that contains most of the data (scores, recommendations and error occurences). For "raw" - unformatted nested list that contains all the data that was returned by the API.

The `api_version` parameter regulates which API version is to create the report. Legacy version 4 is a classic Pagespeed, and the new version 5 returns Lighthouse reports.

The `categories` parameter works only for API version 5. It regulates which of the tests' categories from Lighthouse are to be run. You can select more than one in a vector. Options: "accessibility", "best-practices", "performance", "pwa", "seo".

Value

two options: data frame (if `output_type = "simple"`), nested list (if `output_type = "raw"`)

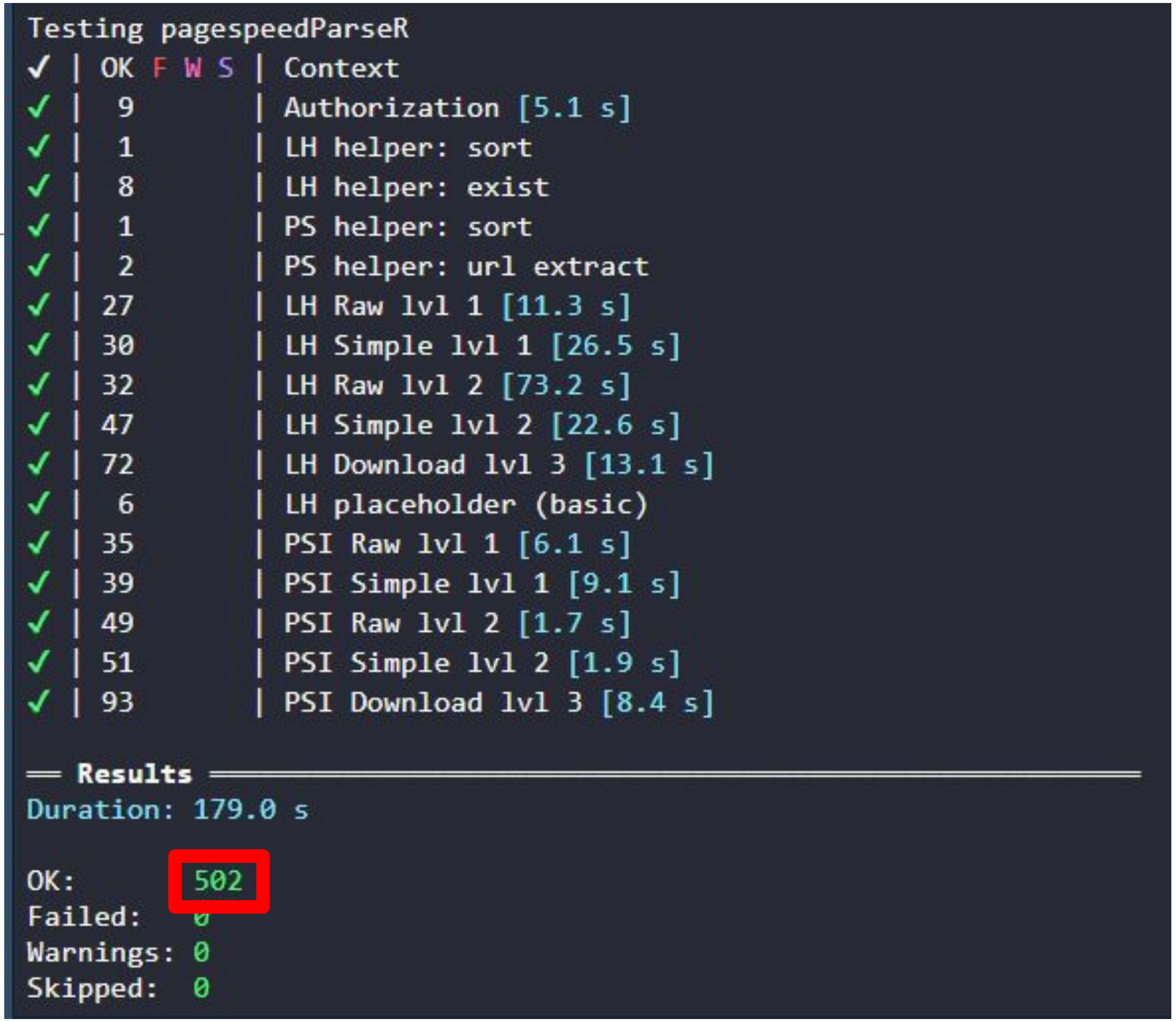
Examples

```
## Not run:
# download simple data frame with "Performance" Lighthouse report for Google.com
lh_df_1 <- download_lighthouse(url = "https://www.google.com",
                               output_type = "simple") # return the results in a wide data frame

# check "Performance" for Google.com & Bing.com for both desktop & mobile and
# return in a data frame with most important columns
lh_df_2 <- download_lighthouse(url = c("https://www.google.com",
                                       "https://www.bing.com/"),
                               output_type = "simple", # return the results in a wide data frame
                               strategy = c("desktop", # check both desktop and mobile, bind
                                            "mobile"),
                               interval = 1, # wait 1 second between the calls to API
                               categories = "performance") # which Lighthouse reports
                                                            # are to be run?
```

Unit tests

- boring…
- … you will hate them...
- … but they can save your life! ;)
- they let you find the bugs and/or changes in API much faster



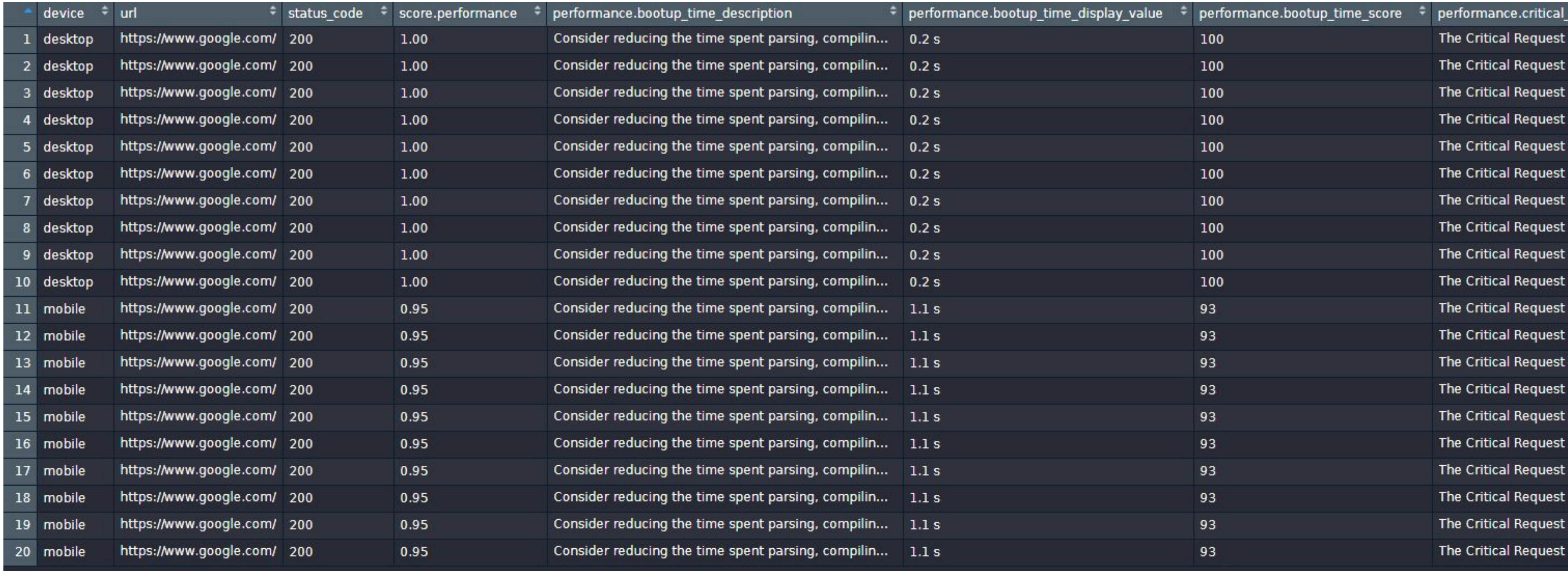
```
Testing pagespeedParseR
✓ | OK F W S | Context
✓ |  9       | Authorization [5.1 s]
✓ |  1       | LH helper: sort
✓ |  8       | LH helper: exist
✓ |  1       | PS helper: sort
✓ |  2       | PS helper: url extract
✓ | 27       | LH Raw lvl 1 [11.3 s]
✓ | 30       | LH Simple lvl 1 [26.5 s]
✓ | 32       | LH Raw lvl 2 [73.2 s]
✓ | 47       | LH Simple lvl 2 [22.6 s]
✓ | 72       | LH Download lvl 3 [13.1 s]
✓ |  6       | LH placeholder (basic)
✓ | 35       | PSI Raw lvl 1 [6.1 s]
✓ | 39       | PSI Simple lvl 1 [9.1 s]
✓ | 49       | PSI Raw lvl 2 [1.7 s]
✓ | 51       | PSI Simple lvl 2 [1.9 s]
✓ | 93       | PSI Download lvl 3 [8.4 s]

══ Results ═══════════════════════════════════════════════════════
Duration: 179.0 s

OK:       502
Failed:   0
Warnings: 0
Skipped:  0
```

Example (data frame)

- data frame can have 124 to 400 columns (Update: in newer version of the package it can be even 2k-7k columns. Pretty sure your Excel is now trembling in horror...)
- easy to compare the URLs
- you can possibly insert it into a database



	device	url	status_code	score.performance	performance.bootup_time_description	performance.bootup_time_display_value	performance.bootup_time_score	performance.critical_
1	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
2	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
3	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
4	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
5	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
6	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
7	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
8	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
9	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
10	desktop	https://www.google.com/	200	1.00	Consider reducing the time spent parsing, compilin...	0.2 s	100	The Critical Request
11	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
12	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
13	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
14	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
15	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
16	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
17	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
18	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
19	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request
20	mobile	https://www.google.com/	200	0.95	Consider reducing the time spent parsing, compilin...	1.1 s	93	The Critical Request

In development

- added support for “medium data” - reports saved as physical files that can support more 1k URLs checked at once (works for 5k-10k right now)
- changed the paradigm of working with the reports:
    - user will be able to download only nested lists and can work with them later on
    - parsing the data is done on the ready object (not on-the-fly as it slows down everything) and is done via separate functions
    - adding an option of leaving the data in JSON (nice to have for some databases’ users)
- increased the amount of data extracted into the data frame

Plans for the future:

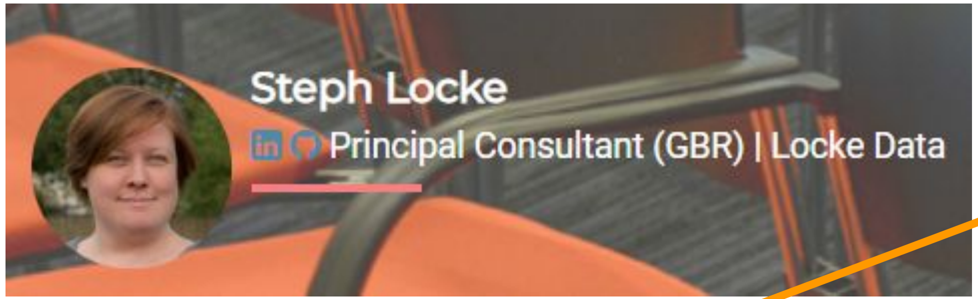
- improved stability
- speeding up the code:
  - lapply instead of for
  - less allocations
  - data.table
  - parallelisation (but it might be hard)
- CRAN release

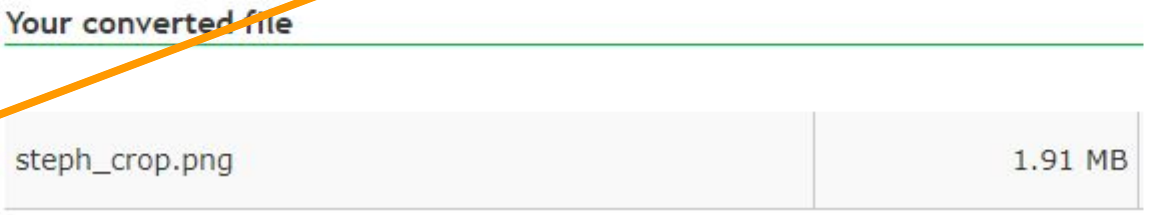
Example of use

We have found out that WhyR Conference page has too big images. Let’s improve them - faster page = more potential useRs!

.gif - 2.25 MB

.gif to .png - 1.91 MB

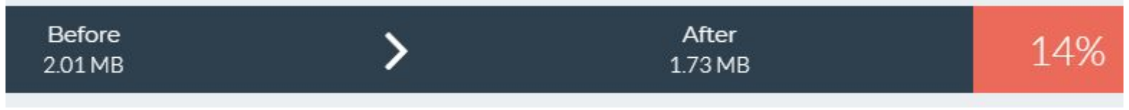



Your converted file

steph_crop.png 1.91 MB

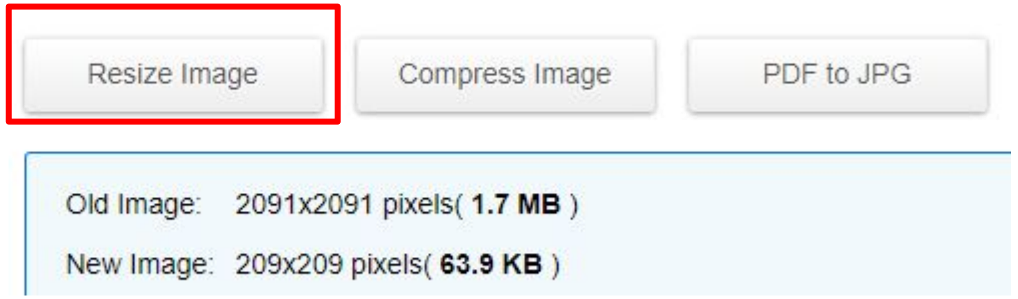
cropping .png - 64 KB

compressing .png - 1.73 MB


Before 2.01 MB

After 1.73 MB

14%



Old Image: 2091x2091 pixels(**1.7 MB**)

New Image: 209x209 pixels(**63.9 KB**)

Example of use

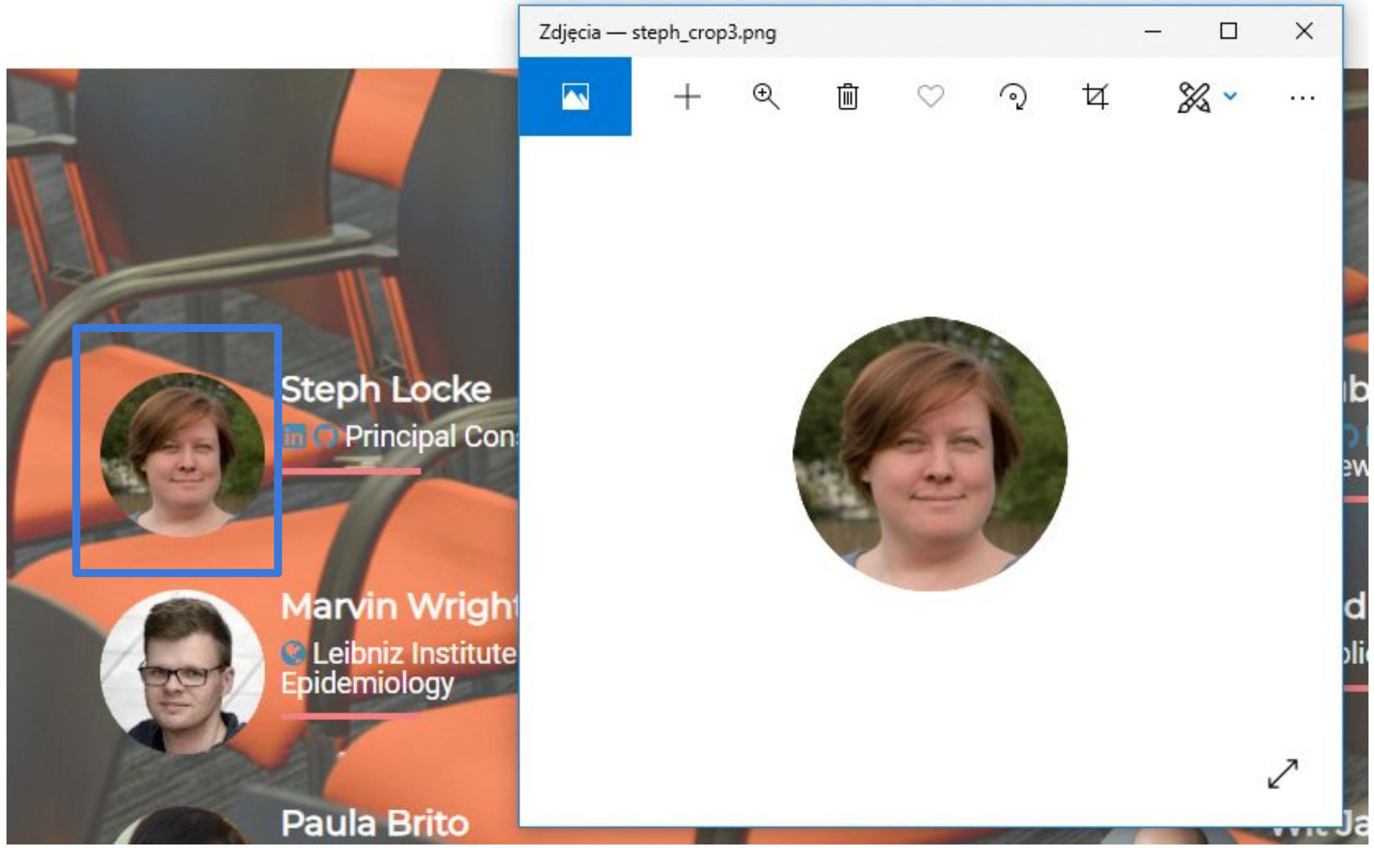
Image on the left: 2.25 MB

Image on the right: 63.9 KB

No visible difference

We have decreased the weight of the image by 97.2%

And maybe we can do it even further if we need to



Questions?

Thanks for your time! :)

Link to the package:

- https://github.com/Leszek-Sieminski/pagespeedParseR

My other R packages:

- https://github.com/Leszek-Sieminski/RAhrefs [CRAN]
- https://github.com/Leszek-Sieminski/screamingFrogR

Contact me:

- LinkedIn: https://pl.linkedin.com/in/leszek-sieminski