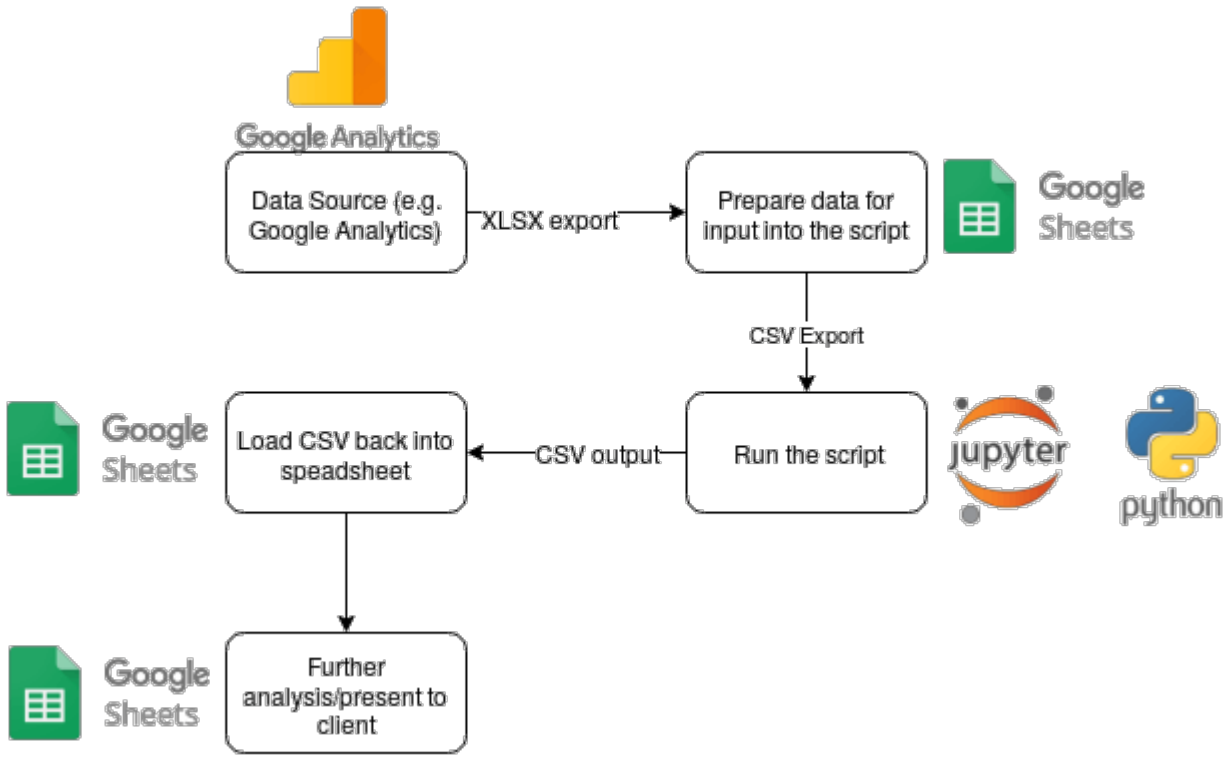
How to show Lighthouse Scores in Google Sheets with a custom function

Automation and machine learning have tremendous potential to help all of us in marketing. But at the moment a lot of these tools are inaccessible to people who can't code or who can code a bit but aren't really that comfortable with it.

What often happens is that there ends up being one or two people in the office who are comfortable with writing and editing code and then these people produce scripts and notebooks that everyone else runs. The workflow looks a bit like this:



I will show you a simple way to streamline this workflow to remove the steps where people need to run a script and format the output. Instead they can run the automation directly from within Google Sheets.

The example I will show you is for a Sheets custom function that returns the Lighthouse score for a URL like in this gif:



The method I will show you isn’t the only way of doing this, but it does illustrate a much more general technique that can be used for many things, including machine learning algorithms.

There are two parts:

1. A Google Cloud Run application that will do the complicated stuff (in this case run a Lighthouse test) and that will respond to HTTP requests.
2. An Appscript custom function that will make requests to the API you created in step 1 and return the results into the Google Sheet.

Cloud run applications

Cloud Run is a Google service that takes a docker image that you provide and makes it available over HTTP. You only pay when an HTTP request is made, so for a service like this that isn’t being used 24/7 it is very cheap. The actual cost will depend on how much you use it, but I would estimate less than $1 per month to run thousands of tests.

The first thing we need to do is make a Docker image that will perform the Lighthouse analysis when we make an HTTP request to it. Luckily for us there is some documentation showing how to run a Lighthouse audit programatically on Github. The linked code saves the analysis to a file rather than returning the response over HTTP, but this is easy to fix by wrapping the whole thing in an Express app like this:

```
const express = require('express');
const app = express();
const lighthouse = require('lighthouse');
const chromeLauncher = require('chrome-launcher');
```

```
app.get('/', async (req, res) => { // Check that the url query parameter exists if(req.query && req.query.url) { // decode the url const url = decodeURIComponent(req.query.url) const chrome = await chromeLauncher.launch({chromeFlags: ['--headless', '--no-sandbox','--disable-gpu']}); const options = {logLevel: 'info', output: 'html', port: chrome.port}; const runnerResult = await lighthouse(url, options); await chrome.kill(); res.json(runnerResult.lhr) }
}); const port = process.env.PORT || 8080;
app.listen(port, () => { console.log(`Listening on port ${port}`);
});
```

Save this code as index.js.

Then you will also need a file called package.json which describes how to install the above application and a Dockerfile so we can wrap everything up in Docker. All the code files are available on Github.

package.json

```
{ "name": "lighthouse-sheets", "description": "Backend API for putting Lighthouse scores in Google sheets", "version": "1.0.0", "author": "Richard Fergie", "license": "MIT", "main": "index.js", "scripts": { "start": "node index.js" }, "dependencies": { "express": "^4.17.1", "lighthouse": "^6.3" }, "devDependencies": {}
}
```

Dockerfile

```
# Use the official lightweight Node.js 10 image.
# https://hub.docker.com/_/node
FROM node:12-slim # Our container needs to have chrome installed to
# run the lighthouse tests
RUN apt-get update && apt-get install -y \ apt-transport-https \ ca-certificates \ curl \ gnupg \ --no-install-recommends \ && curl -sSL https://dl.google.com/linux/linux_signing_key.pub | apt-key add - \ && echo "deb https://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/sources.list.d/google-chrome.list \ && apt-get update && apt-get install -y \ google-chrome-stable \ fontconfig \ fonts-ipafont-gothic \ fonts-wqy-zenhei \ fonts-thai-tlwg \
```

```
fonts-kacst \ fonts-symbola \ fonts-noto \ fonts-freefont-ttf \ --no-install-recommends \ && apt-get purge --auto-remove -y curl gnupg \ && rm -rf /var/lib/apt/lists/* # Create and change to the app directory.
WORKDIR /usr/src/app # Copy application dependency manifests to the container image.
# A wildcard is used to ensure copying both package.json AND package-lock.json (when available).
# Copying this first prevents re-running npm install on every code change.
COPY package*.json ./ # Install production dependencies.
# If you add a package-lock.json, speed your build by switching to 'npm ci'.
# RUN npm ci --only=production
RUN npm install --only=production # Copy local code to the container image.
COPY . ./ # Run the web service on container startup.
CMD [ "node", "--unhandled-rejections=strict","index.js" ]
```

Build the docker image and then you can test things locally on your own computer like this:

First start the image:

```
docker run -p 8080:8080 lighthouse-sheets
```

And then test to see if it works:

```
curl -v "localhost:8080?url=https%3A%2F%2Fwww.example.com"
```

Or visit localhost:8080?url=https%3A%2F%2Fwww.example.com in your browser. You should see a lot of JSON.

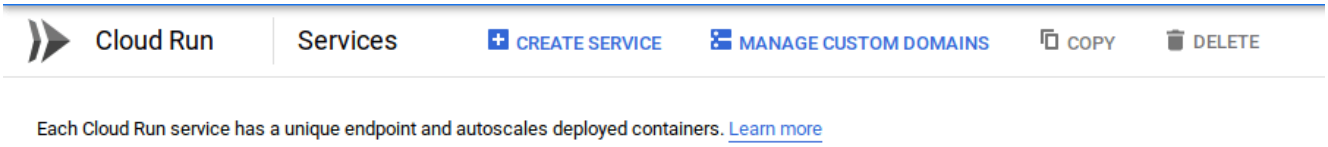
The next step is to push your image to the Google Container registry. For me, this is a simple command:

```
docker push gcr.io/MY_PROJECT_ID/lighthouse-sheets
```

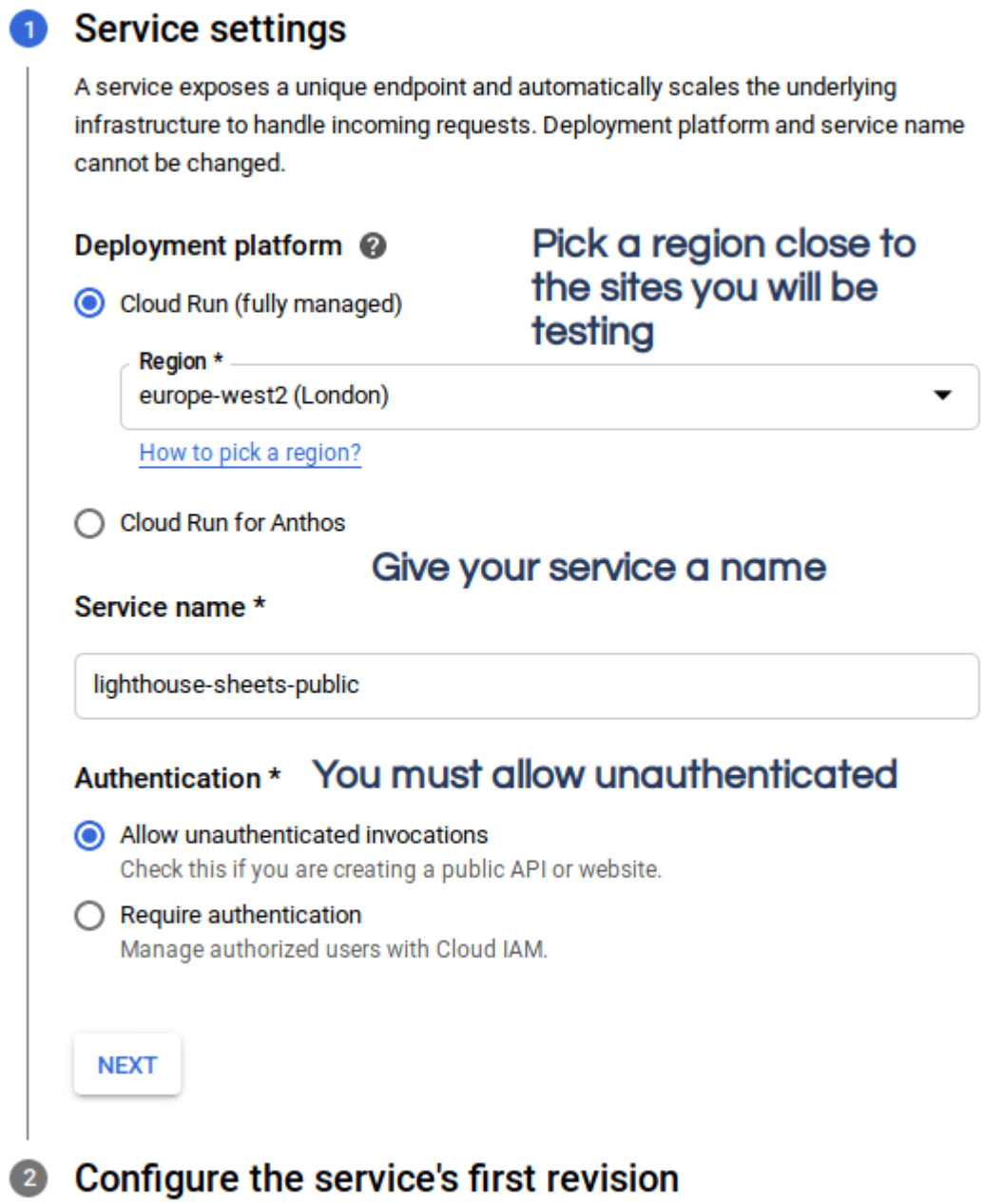
But you might have to setup the docker authentication first before you can do this. An alternative method is the use Google Cloud Build to make the image; this might work better for you if you can’t get the authentication working.

Next you need to create a Cloud Run service with this docker image.

Open Cloud Run and click “Create service”



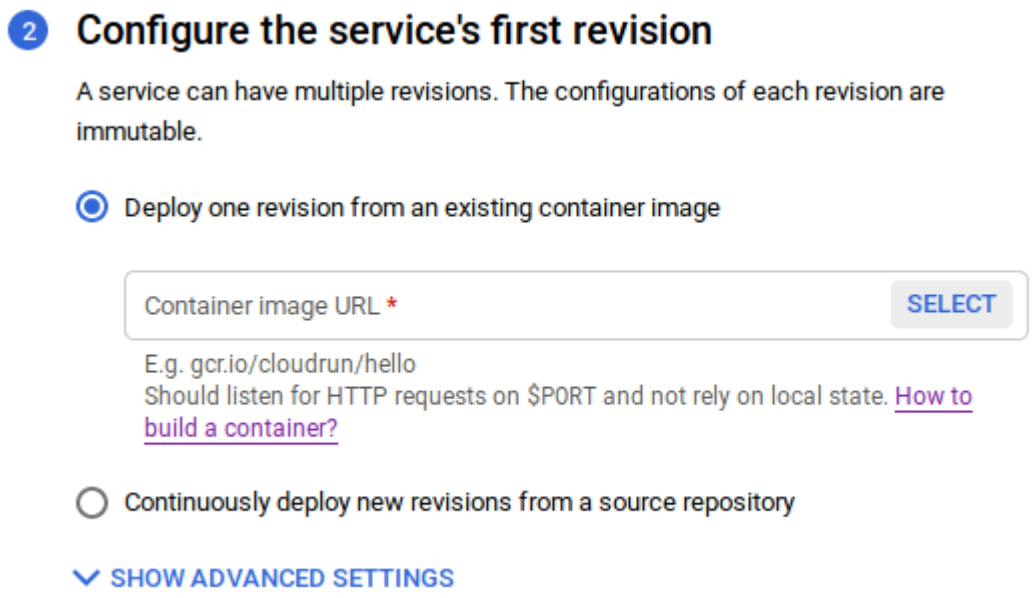
Name and adjust settings. You must give your service a name and configure a few other settings:



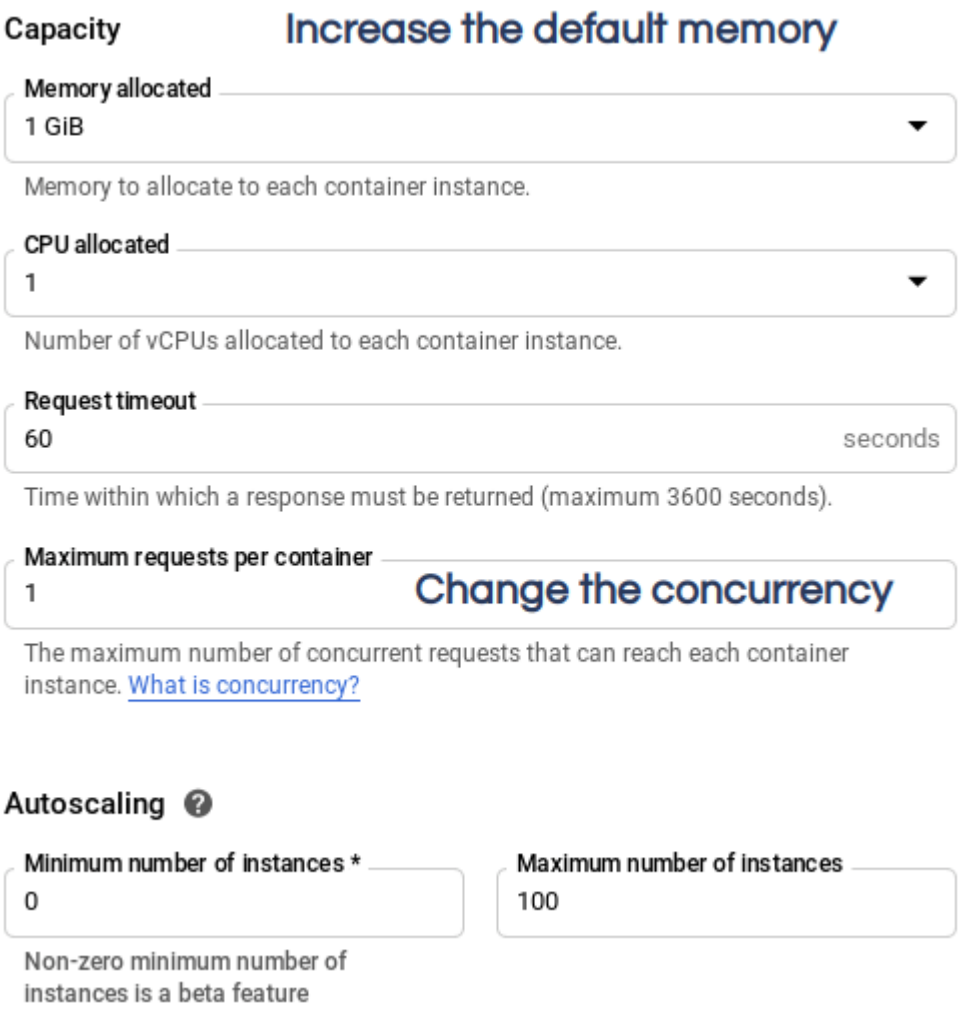
It is best to pick a region that is close to where most of the audience for your sites live. Checking the site speed for a UK site from Tokyo won't give you the same results as what your audience get.

In order for you to call this service from Google Sheets it must allow unauthenticated invocations. If you're worried about locking down and securing the service to prevent other people from using it you will have to do this by (for example) checking from an API secret in the HTTP request or something like that.

Next you must select the container you made earlier. You can type in the name if you remember it or click "Select" and choose it from the menu.



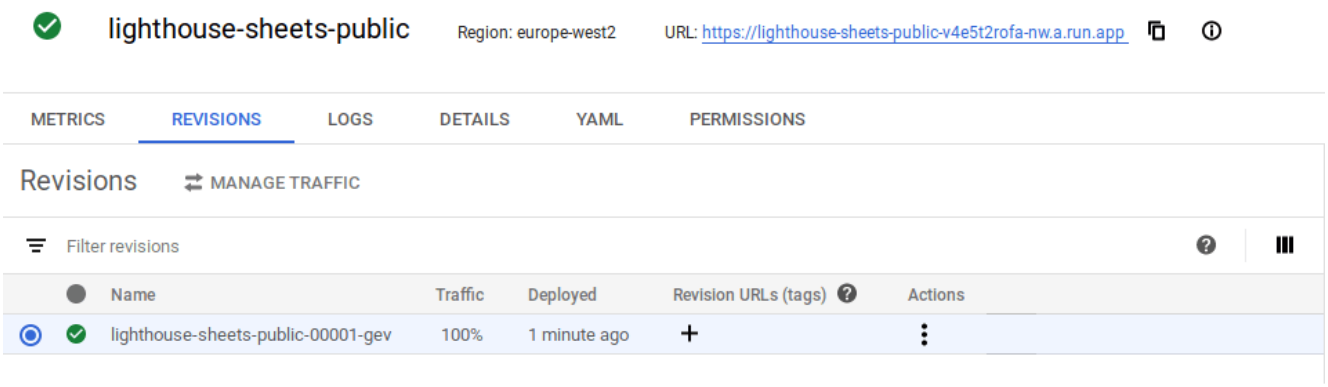
Then click "Show Advanced Settings" because there is further configuration to do.



You need to increase the memory allocation because Lighthouse tests need more than 256Mb to run. I have chosen 1GiB here but you might need the maximum allowance of 2GiB for some sites.

I have found that reducing the concurrency to 1 improves the reliability of the service. This means Google will automatically start a new container for each HTTP request. The downside is that this costs slightly more money.

Click “Create” and your Cloud Run service will be ready shortly.



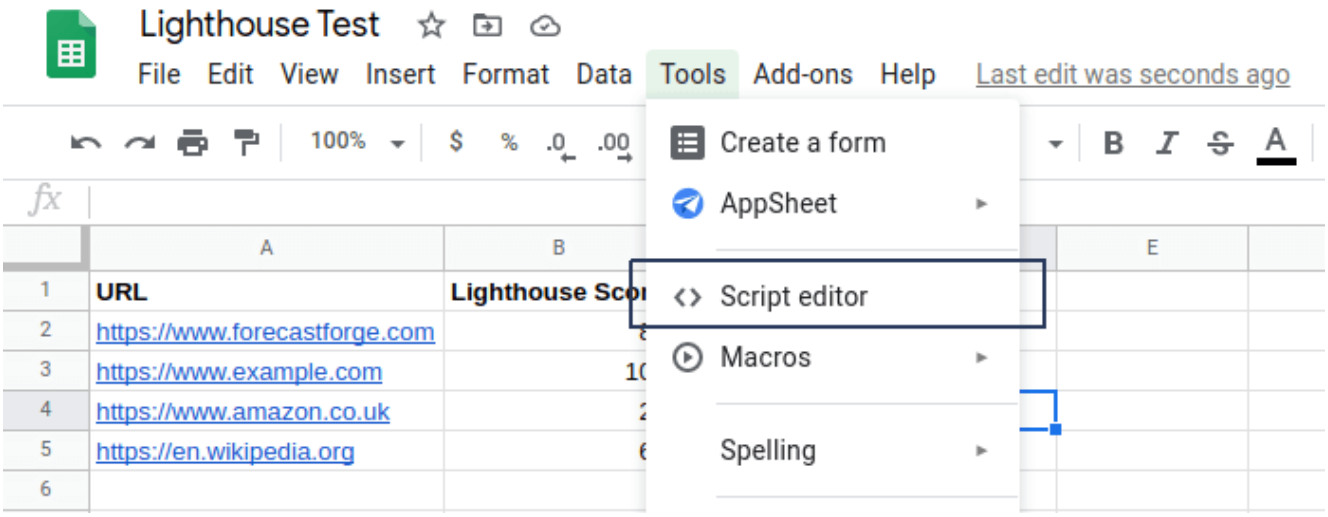
You can give it a quick test using the URL. For example:

```
curl -v "https://lighthouse-sheets-public-v4e5t2rofa-nw.a.run.app?url=https%3A%2F%2Fwww.example.com"
```

Or visit https://lighthouse-sheets-public-v4e5t2rofa-nw.a.run.app?url=https%3A%2F%2Fwww.example.com in your browser.

The next step is to write some Appscript so you can use your new API from within Google Sheets.

Open a new Google Sheet and the open up the Appscript editor.



This will open a new tab where you can code your Google Sheets custom function.

The key idea here is to use the Appscript UrlFetchApp function to perform the HTTP request to your API. Some basic code to do

this looks like this:

```
function LIGHTHOUSE(url) { const BASE_URL = "https://lighthouse-sheets-public-v4e5t2rofa-nw.a.run.app" var request_url = BASE_URL+"?url="+encodeURIComponent(url) var response = UrlFetchApp.fetch(request_url) var result = JSON.parse(response.getContentText()) return(result.categories.performance.score * 100) }
```

The last line returns the overall performance score into the sheet. You could edit it to return something else. For example to get the SEO score use result.categories.seo.score instead.

Or you can return multiple columns of results by returning a list like this:

```
[result.categories.performance.score, result.categoryies.seo.score]
```

Save the file and then you will have a custom function available in your Google Sheet called LIGHTHOUSE.

The easiest way to get started with this is to copy my example Google Sheet and then update the code yourself to point at your own API and to return the Lighthouse results you are most interested in.

Enhance your spreadsheet know-how

The great thing about this method is that it can work for anything that can be wrapped in a Docker container and return a result within 30 seconds. Unfortunately Google Sheets custom functions have a timeout so you won’t have long enough to train some massive deep learning algorithm, but that still leaves a lot that you can do.

I use a very similar process for my Google Sheets addon Forecast Forge, but instead of returning a Lighthouse score it returns a machine learning powered forecast for whatever

numbers you put into it.

The possibilities for this kind of thing are *really* exciting because in Search Marketing we have a lot of people who are very good with spreadsheets. I want to see what they can do when they can use all their spreadsheet knowledge and enhance it with machine learning.

Opinions expressed in this article are those of the guest author and not necessarily Search Engine Land. Staff authors are listed here.

About The Author



Richard Fergie is the founder of Forecast Forge which brings Machine Learning powered forecasts into Google Sheets so that anyone can use them to produce quick and accurate estimates for what will happen in the future. Richard consults on projects at the intersection of digital marketing, web analytics and data science where he has over 10 years of experience.
http://feeds.searchengineland.com/~r/searchengineland/~3/uFey-q-Li-s/how-to-show-lighthouse-scores-in-google-sheets-with-a-custom-function-343464