

All your favorite parts of Medium are now in one sidebar for easy access.

the best of Medium for less than \$1/week. [Become a member](#)



Okay, got it

[Analytics Weekly](#) · [Follow publication](#)

Welcome to the Global Power Plants Dashboard

Explore distribution of power plants by country

Select Countries

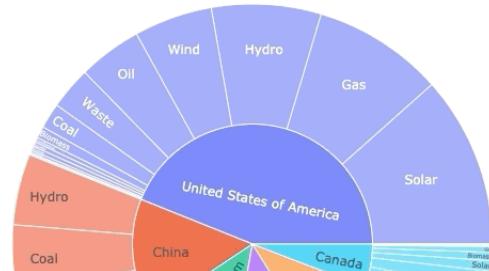
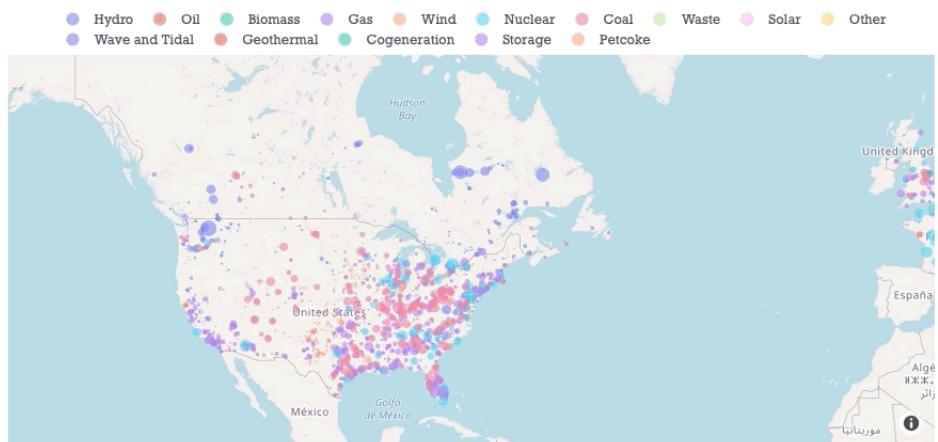
- United States of America China
 - United Kingdom Brazil France
 - Canada
-
- Afghanistan
 - Albania
 - Algeria
 - Angola
 - Antarctica
 - Argentina

Reported Capacity, 2017.0

- Source for Plant info: [Natural Resources Canada](#)

Note: nan means the information is not in the database

Source of Global Power Plants Data



[Global Power Plants dashboard created using Dash](#)

End-to-End Guide: Creating a Web Application using Dash

8 min read · Oct 17, 2020



Hanan Ather

Follow

Listen

Share

More

By the end of this guide you will know how to **create** and **deploy** your own dashboard on the web.

Why Dash?

Why use Dash to build an application over other great tools? I became interested in

All your favorite parts of Medium are now in one sidebar for easy access.

:
h other big data frameworks such as Spark for building a datasets

- it can create highly customizable interactive Dashboards for unstructured datasets using R or Python
- it requires no knowledge of HTML or Javascript

Why not just use BI tools?

I want to preface by saying Tableau and Power BI are great tools. If your goal is to make a great-looking interactive dashboard using a structured dataset, they are a great option.

However, Tableau and other BI tools don't offer the same level of flexibility as Python and R when it comes to working with unstructured data. It can be more efficient to use Python or R if the data requires a lot of pre-processing and transformation before analytics.

What I find most appealing about Dash is its integrability with big data and parallel computing frameworks. Dash apps can serve as the front-end to Spark clusters. If your file is too large to fit on your memory, and you don't want to work with cloud computing frameworks, Dash is also compatible with Vaex. Vaex is a python library for lazy “Out-of-Core” DataFrames (similar to Pandas) which can be as large as the size of your hard drive! The founders of Vaex wrote an excellent blog post, which I highly recommend if you're interested in working with large datasets.

The Global Power Plant Dataset

The Global Power Plant data displayed on the dashboard comes from the World Resources Institute. The Global Power Plant dataset is comprehensive and contains information about approximately 30,000 power plants across 164 different countries. The dataset contains detailed features about power plants such as geolocation, plant owner, primary fuel and secondary fuel types, etc.

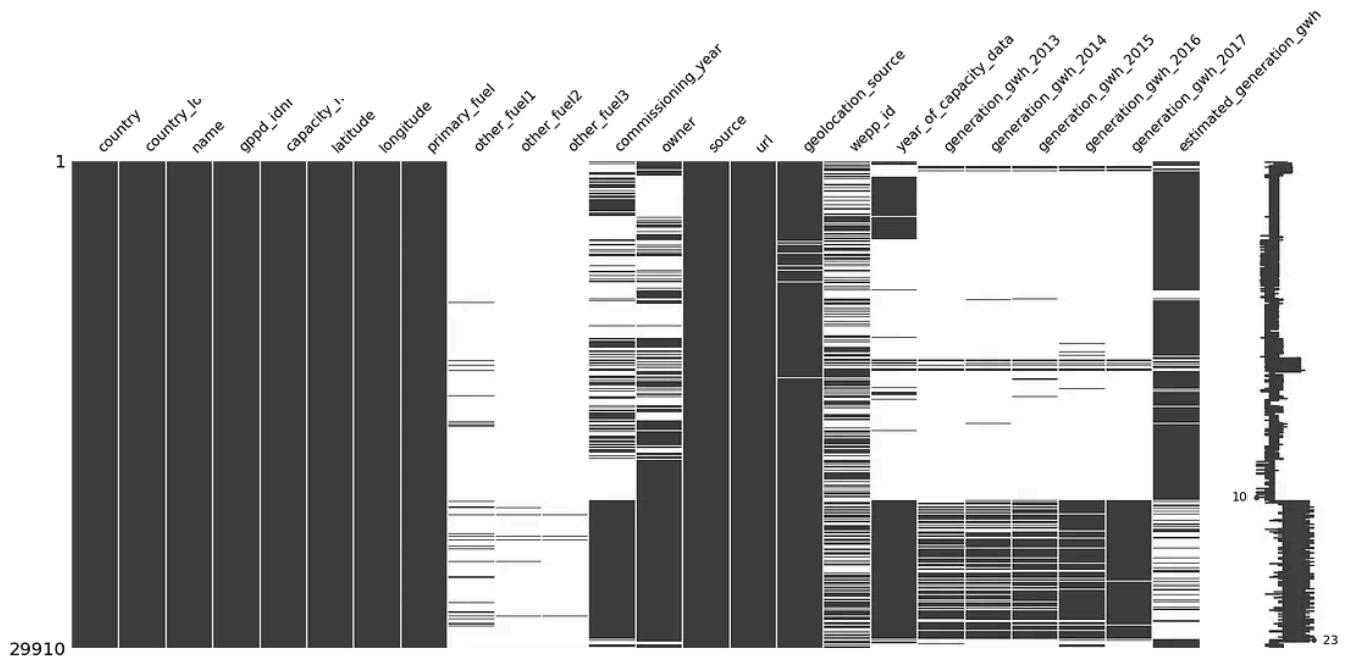
Getting to know the data

The first step towards building the dashboard was gaining familiarity with the dataset and figuring out what relevant information I wanted to convey. After

narrowing down a list of features, the quality of the data was accessed.

The `missingno` library to visualize the distribution of missing values in the

All your favorite parts of
Medium are now in one
sidebar for easy access.



This is a crucial step because it helped refine the list of import features. I decided it was not worth including values from fields such as `other_fuel2` & `other_fuel3` since most of the values are missing.

Designing the Dashboard

I decided that an interactive map would be the best way to capture the dataset. I started by plotting all power plant's latitude and longitude with Plotly's

```
px.scattermap .
```

```

1 # filtering data
2 dff = df[df.country_long.isin(countries)]
3

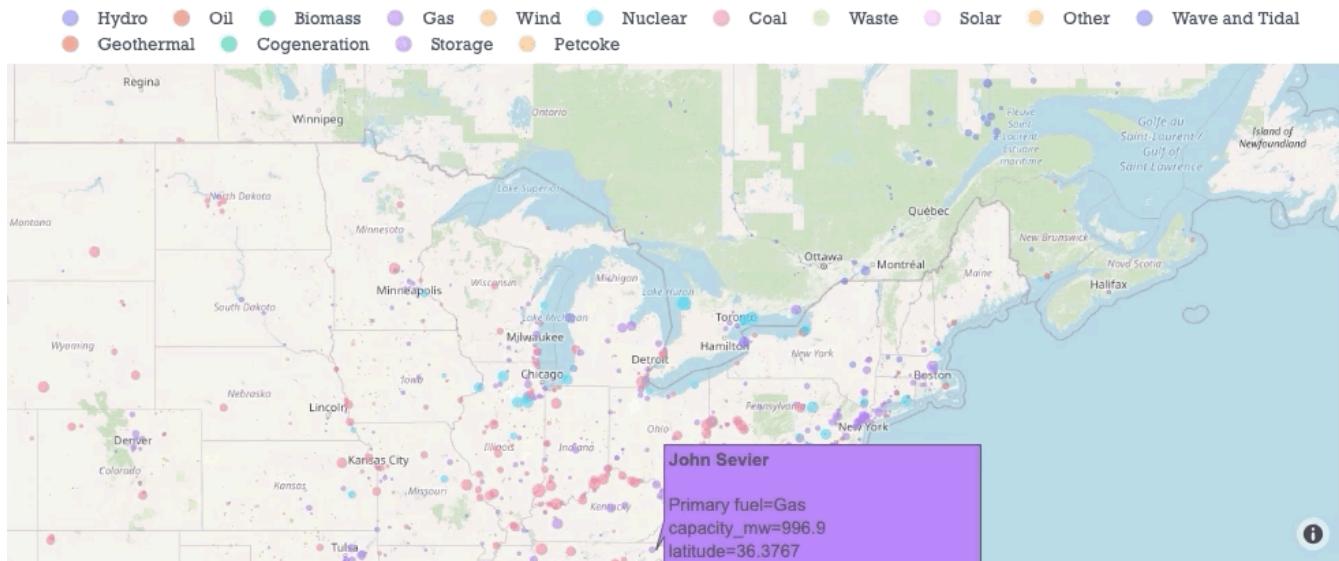
All your favorite parts of
Medium are now in one
sidebar for easy access.

9
10
11
12
13 # customize font and legend orientation & position
14 fig.update_layout(
15     font_family="Rockwell",
16     autosize=True,
17     margin=go.layout.Margin(l=0, r=0, t=0, b=35),
18     legend=dict(title= None, orientation="h", y=1, yanchor="bottom", x=0.5,
19                 xanchor="center"))
20

```

scatter-map.py hosted with ❤ by GitHub

[view raw](#)



It also has additional functionality that allows the user to filter power plants by fuel types by clicking on the legend.

One of the principal quantities I was interested in comparing was the distribution of different categories of power plants worldwide. I used the Sunburst plots because they can be an effective way to visualize hierarchical data distributions.

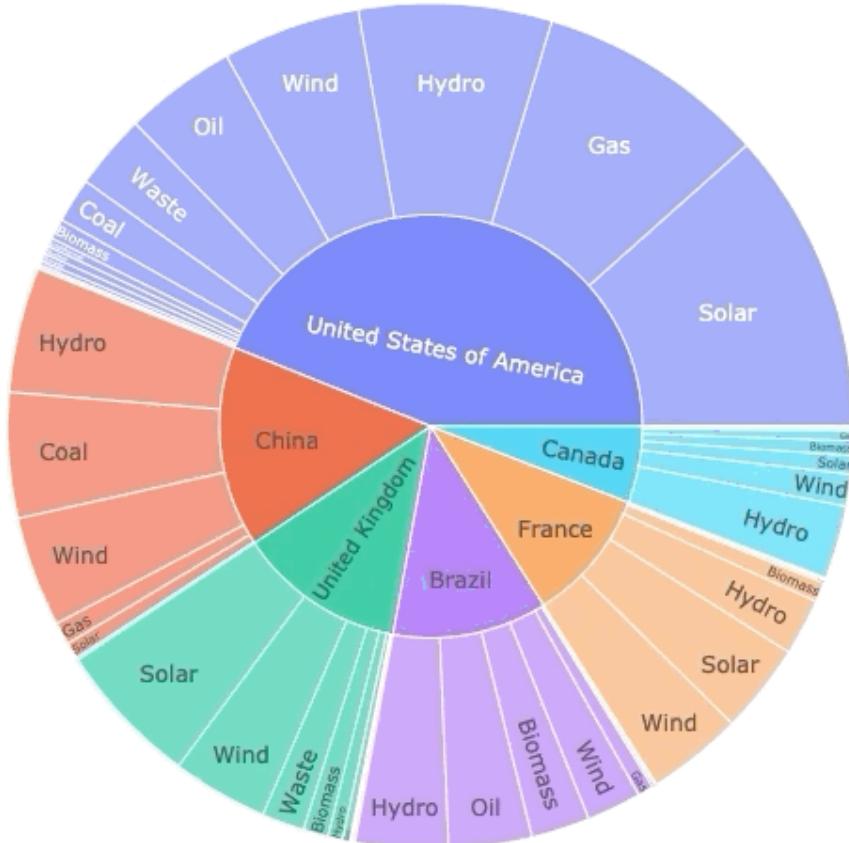
```
1 Grouped = dff.groupby(['country_long','primary_fuel'])['primary_fuel'].count().rename_ax  
2 .reset_index(name='Count')  
3
```

All your favorite parts of Medium are now in one sidebar for easy access.

```
rst(Grouped, path=['Country', 'Fuel'], values='Count')  
out(margin=go.layout.Margin(l=0, r=0, t=0, b=0))
```

by GitHub

[view raw](#)



This plot displays the distribution of power plants top five countries with the most power plants.

I used Dash's [Multi-Value Dropdown](#) to filter the countries displayed on the sunburst plot and the map.

```
1 df = pd.read_csv("global_power_plant_database.csv")  
2  
3 countries = df['country_long'].unique()  
4 dcc.Dropdown(id='country',  
5 placeholder='Select a Country',  
6 options=[{'label': i, 'value': i} for i in countries],  
7 value=['United States of America', 'China',  
8 'United Kingdom', 'Brazil', 'France', 'Canada'],  
9 multi=True  
10 )
```

multi-dropdown.py hosted with ❤ by GitHub

[view raw](#)

All your favorite parts of Medium are now in one sidebar for easy access.



After importing the dataset as a pandas DataFrame, the names of all the countries in the dataset are extracted into a pandas series. **The multi-value dropdown takes the Pandas series containing the countries as an iterable to create a dictionary that the dash-component requires.**

Connecting the components

Up until now, we have successfully created individual components, but we still haven't connected them.

To connect the different components, we have to use Dash Callbacks functions.

What are Dash callback functions?

Dash callback functions are python functions that are automatically called by Dash whenever an input component's property changes.

Whenever the user selects a different set of countries in the multi-value dropdown, a python function, referred to as a **Dash Callback**, will be called. This function's input parameter will be the new countries that have been selected by the user, and it will return an updated map and a sunburst plot.

The code looks something approximate to this:

```

1 @app.callback([
2     Output('map', 'figure'),
3     Output('graph', 'figure')])
4         [, 'value')])
5 All your favorite parts of
6 Medium are now in one
7 sidebar for easy access.
8     a
9     ntry_long.isin(countries)]
10
11     # code for map
12     fig = px.scatter_mapbox(dff, lat="latitude", lon="longitude",
13                             hover_name="name", hover_data=["country_long",
14                             "primary_fuel"],
15                             color="primary_fuel", zoom=2,
16                             center = {'lat': 45.4215, 'lon' :-75.6972}, opacity = 0.6,
17                             mapbox_style='open-street-map',
18                             labels = {'primary_fuel': 'Primary fuel'},
19                             size = 'capacity_mw')
20
21     # customize font and legend orientation & position
22     fig.update_layout(
23         font_family="Rockwell",
24         autosize=True,
25         margin=go.layout.Margin(l=0, r=0, t=0, b=35),
26         legend=dict(title= None, orientation="h", y=1, yanchor="bottom", x=0.5,
27                     xanchor="center"))
28
29     )
30
31     # code for sunburst chart
32     Grouped = dff.groupby(['country_long','primary_fuel'])['primary_fuel'] \
33                 .count().rename_axis(['Country','Fuel']) \
34                 .reset_index(name='Count')
35
36     graph = px.sunburst(Grouped, path=['Country', 'Fuel'], values='Count')
37     graph.update_layout(margin=go.layout.Margin(l=0, r=0, t=0, b=0))
38
39
40     return fig, graph

```

callback.py hosted with ❤ by GitHub

[view raw](#)

NOTE: If you try to run the code, it won't compile. This is just an intermediate step. You can scroll below to find a version that will compile and run for you.

The “Inputs” (a set of countries) and the “Outputs” (updated map and sunburst plot) are described **declaratively** as the arguments of the `@app.callback` decorator.

If you don't know what decorator is, essentially, declaring the `@app.callback` decorator tells Dash to call the function below it whenever the “Input” value changes.

Therefore, whenever the selected countries are changed by the user, the `def update_figures` function is called. The input parameter of this function takes the

All your favorite parts of Medium are now in one sidebar for easy access.

the `@app.callback` decorator. For instance, if we had two or, the `def update_figures` function would have to have two actor has two “Outputs”: the map and the graph. Hence, `def` two objects: `scatter_map` and `graph`.

Similarly, I used another `@app.callback` decorator for updating the summary when the user clicks on a particular power plant.

```
1  @app.callback(dash.dependencies.Output('plant_summary', 'children'),
2                  [dash.dependencies.Input('map', 'clickData')])
3  def update_summary(click_Data):
4      plant_name = click_Data['points'][0]['hovertext']
5      owner = df[df['name'] == plant_name]['owner'].iloc[0]
6      gwh = round(df[df['name'] == plant_name]['estimated_generation_gwh'].iloc[0],2)
7      commissioning_year = df[df['name'] == plant_name]['commissioning_year'].iloc[0]
8      capacity_mw = df[df['name'] == plant_name]['capacity_mw'].iloc[0]
9      estimated_generation_gwh = df[df['name'] == plant_name]['estimated_generation_gwh']
10     year_capacity = df[df['name'] == plant_name]['year_of_capacity_data'].iloc[0]
11     source = df[df['name'] == plant_name]['source'].iloc[0]
12     url = df[df['name'] == plant_name]['url'].iloc[0]
13     update = f'''
14             **Summary of *{plant_name}*:**\n
15             - Plant owner: {owner}\n
16             - Estimated Yearly Generation (GWH):\n
17                 {gwh}\n
18             - First Year the Plant Generated Energy:\n
19                 {commissioning_year}\n
20             - installed electrical capacity (MW):\n
21                 {capacity_mw}\n
22             - Year of Reported Capacity: {year_capacity}\n
23             - Source for Plant info: [{source}]( {url})\n
24             - Annual generation in gigawatt hours (GWs):\n
25                 {estimated_generation_gwh}\n
26\n
27             ...
28\n
29
30     return update
```

Connecting-components-2.py hosted with ❤ by GitHub

[view raw](#)

Note that in the decorator, the “Input” is now the `dcc.Graph` component (the scatter map), and the "Output" is a Dash markdown-component.

The function `update_summary` takes a parameter called `click_Data`. This comes from the `clickData` attribute of `dcc.Graph` components. `clickData` is one of four user-

All your favorite parts of Medium are now in one sidebar for easy access.

of `dcc.Graph` components. The other attributes are `hoverData`, `rayData`. More information about them can be found [here](#).

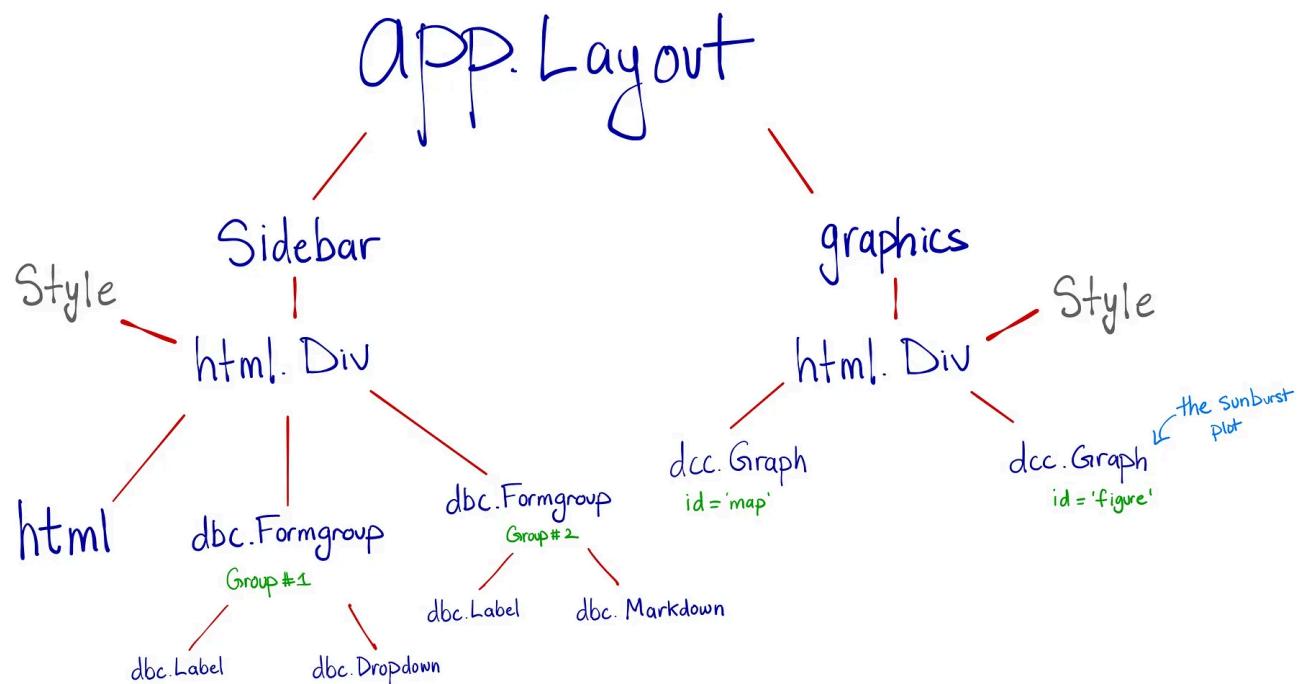
to be the best choice for the Global Power Plants dashboard. I am out to see which one works best for your Dashboard.

Structure of App Layout

There are a few different ways to configure the layout of a Dash application. I used the dash-bootstrap-components library for styling my app. If you're just getting started, I highly recommend [this](#) video by Charming Data.

Using Bootstrap I separated the app into two main components:

- the Sidebar
- graphics



- The `dash_html_components` (HTML)library contains components for all HTML tags.
- The `dash_core_components` (dcc) describes higher-level interactive components such as the map and the sunburst plot

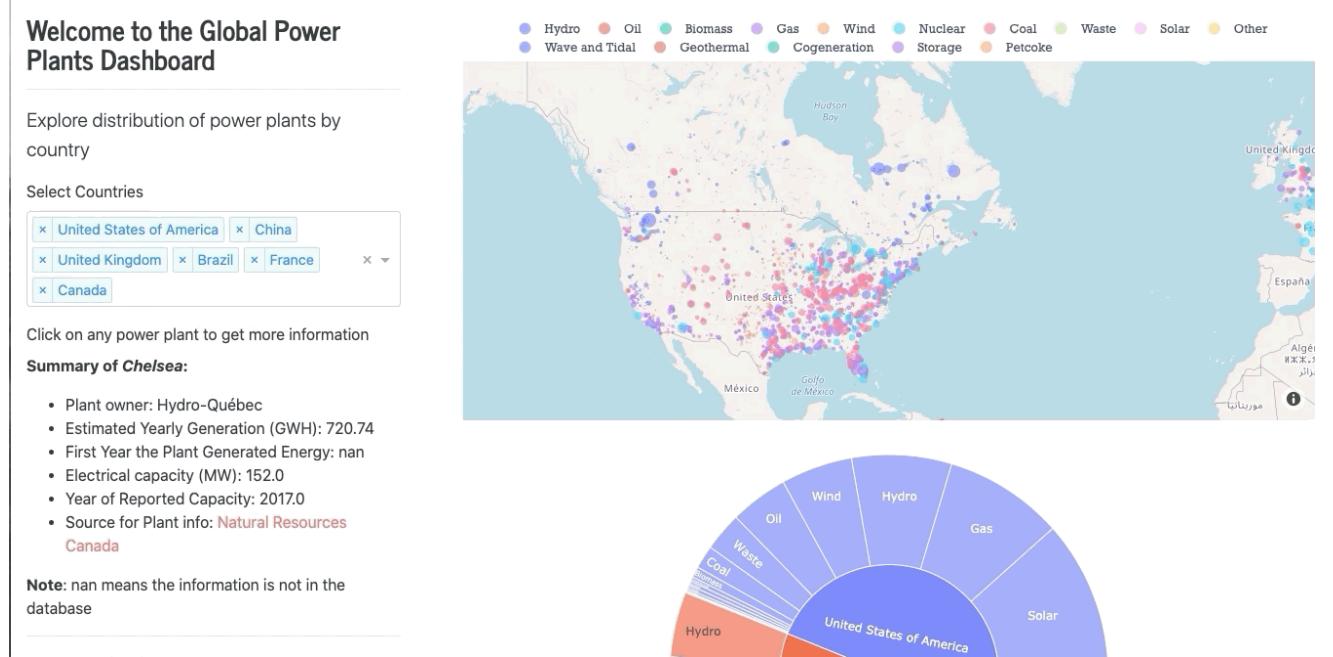
- the `dash_bootstrap_components` (`dbc`) library contains Bootstrap components for Dash

All your favorite parts of Medium are now in one sidebar for easy access.

In fact, the layout of Dash apps is structured like a tree. This structure provides a lot of flexibility in terms of adding or removing components. It isn't necessary to use Bootstrap for styling your app, but it does provide a lot more flexibility. There are many free Bootstrap stylesheets available and the theme of the application can be changed with a single line:

```
df = pd.read_csv("data/globalpowerplantdata/global_power_plant_database.csv")
countries = df['country_long'].unique()

app = dash.Dash(external_stylesheets=[dbc.themes.BOOTSTRAP])
```



The above example demonstrates the ease by which the theme of an app can be modified.

The `dash_bootstrap_components.themes` module contains the Content Delivery Network (CDN) links for Bootstrap and Bootswatch themes. It is possible to modify or compile your own theme. It can be served locally by replacing `dbc.themes.BOOTSTRAP` with the URL of the stylesheet. For more information, you can read the [documentation](#).

After tweaking and styling everything, our final python script for creating the dashboard looks like this:

```
1 @author: hananather
2 """
3 import dash
All your favorite parts of
Medium are now in one
sidebar for easy access.
        components as dcc
        components as html
        cies import Input, Output
        d
        ess as px
9 import dash_bootstrap_components as dbc
10 from plotly import graph_objs as go
11
12
13 app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])
14 server = app.server
15
16 df = pd.read_csv("data/global_power_plant_database.csv")
17 countries = df['country_long'].unique()
18
19 # the style arguments for the sidebar. We use position:fixed and a fixed width
20 SIDEBAR_STYLE = {
21     "position": "fixed",
22     "top": 0,
23     "left": 0,
24     "bottom": 0,
25     "width": "26rem",
26     "padding": "2rem 1rem",
27     "background-color": "#f8f9fa",
28 }
29
30 # the styles for the main content position it to the right of the sidebar and
31 # add some padding.
32 CONTENT_STYLE = {
33     "margin-left": "28rem",
34     "margin-right": "2rem",
35     "padding": "2rem 1rem",
36 }
37
38 sidebar = html.Div(
39     [
40         html.H3("Welcome to the Global Power Plants Dashboard"),
41         html.Hr(),
42         html.P(
43             "Explore distribution of power plants by country", className="lead"
44         ),
45
46         dbc.FormGroup( # GROUP 1
47             [
48                 dbc.Label("Select Countries")
```

```
48         dbc.Label('Select Countries'),
49         dcc.Dropdown(id='country',
50                         placeholder='Select a Country',
51                         options=[{'label': i, 'value': i} for i in countries],
52                         value=['United States of America', 'China',
53                                'United Kingdom', 'Brazil', 'France', 'Canada'],
54                         multi=True
55                     ),
56
57     ),
58     dbc.FormGroup( # GROUP 2
59     [
60         dbc.Label("Click on any power plant to get more information"),
61         dcc.Markdown(id='plant_summary')
62     ]
63 ),
64
65     html.Hr(),
66     html.A("Source of Global Power Plants Data", href='https://datasets.wri.or
67
68 ],
69     style=SIDEBAR_STYLE,
70 )
71
72 content =html.Div(children= [dcc.Graph(id='map',clickData={'points': [{''hovertext': 'C
73                                     dcc.Graph(id='graph')}],style=CONTENT_STYLE)
74
75
76
77
78 app.layout = html.Div([sidebar, content])
79
80 @app.callback([
81     Output('map', 'figure'),
82     Output('graph', 'figure')],
83     [Input('country', 'value')])
84 def update_map(countries):
85     df = df[df.country_long.isin(countries)]
86
87     fig = px.scatter_mapbox(df, lat="latitude", lon="longitude",
88                             hover_name="name", hover_data=["country_long", "primary_fuel"]
89                             color="primary_fuel",zoom=2,
90                             center = {'lat': 45.4215, 'lon' :-75.6972},opacity = 0.6,
91                             mapbox_style='open-street-map',
92                             labels = {'primary_fuel': 'Primary fuel'},
93                             size = 'capacity_mw')
94     fig.update_layout( # customize font and legend orientation & position
95
```

```

96     # font and legend
97     font_family="Rockwell",
98     autosize=True,
99
100    All your favorite parts of
101    Medium are now in one
102    sidebar for easy access.
103
104
105    Grouped = dff.groupby(['country_long','primary_fuel'])['primary_fuel'].count().ren
106    graph = px.sunburst(Grouped, path=['Country', 'Fuel'], values='Count')
107
108    graph.update_layout(margin=go.layout.Margin(l=0, r=0, t=0, b=35),
109                          )
110
111    return fig, graph
112
113 @app.callback(dash.dependencies.Output('plant_summary', 'children'),

```

[Open in app ↗](#)

Medium



```

118    gwh = round(df[df['name'] == plant_name]['estimated_generation_gwh'].iloc[0],2)
119    commissioning_year = df[df['name'] == plant_name]['commissioning_year'].iloc[0]
120    capacity_mw = df[df['name'] == plant_name]['capacity_mw'].iloc[0]
121    year_capacity = df[df['name'] == plant_name]['year_of_capacity_data'].iloc[0]
122    source = df[df['name'] == plant_name]['source'].iloc[0]
123    url = df[df['name'] == plant_name]['url'].iloc[0]
124
125
126    update = f'''
127                **Summary of *{plant_name}*:**\n
128                - Plant owner: {owner}\n
129                - Estimated Yearly Generation (GWH): {gwh}\n
130                - First Year the Plant Generated Energy: {commissioning_year}\n
131                - Electrical capacity (MW): {capacity_mw}\n
132                - Year of Reported Capacity: {year_capacity}\n
133                - Source for Plant info: [{source}]({url})\n
134
135
136
137                **Note**: nan means the information is not in the database\n
138                ...
139
140
141    return update
142
143 if __name__ == '__main__':

```

Deploying the application on the Web

All your favorite parts of Medium are now in one sidebar for easy access.

hare the dashboard, we can start the deployment process. o deploy your application on the Web. The method shown thod that I've come across for quickly deploying your web involve creating a virtual environment or using the command

Step 1: Initialize and create a GitHub repository. To deploy your application on the Web, we need the following five files in our repository:

- app.py
- CSV file containing your dataset
- README.md
- requirement.txt
- Procfile

The first three files are self-explanatory. The app.py is what we have been working on thus far, and the README.md is a markdown file that contains information regarding the project.

The requirement.txt looks like this:

All your favorite parts of Medium are now in one sidebar for easy access.

This file contains all the packages and their respective versions that our application uses.

It is important to add the correct version of the packages you are using in your environment. If you don't know the versions of your packages don't worry! The `print(name_of_package.__version__)` command will print the version of the package, and you can directly copy-paste them in the `requirement.txt` file using the format I showed above. **The gunicorn is an essential package for deployment!** It is a Python WSGI HTTP Server for UNIX. If you don't have this package simply use the `$ pip install gunicorn` command in the terminal.

The `Procfile` is an essential file for deploying our application, and it looks like this :

All your favorite parts of
Medium are now in one
sidebar for easy access.

You can copy-paste the line above into your repository. When finished, our repository for the application looks like this.

The screenshot shows a GitHub repository page for 'global-power-plants'. The top navigation bar includes links for Projects, Wiki, Security, Insights, and Settings. On the left, a sidebar lists favorite parts of Medium. The main content area shows a list of commits:

File	Time
README.md	16 hours ago
app.py	16 hours ago
requirements.txt	16 hours ago
Initial commit	23 days ago
Create Procfile	16 hours ago
Add files via upload	16 hours ago

Below the commits, there's a file viewer for README.md, which contains the text 'global-power-plants' and a description 'A dash board app of the global power plants database'. To the right, sections for About, Releases, Packages, and Environments are displayed.

Step 2: Create a new App on Heroku. After registering for [Heroku](#), we create a new app and choose a name for our application.

The screenshot shows the Heroku dashboard. A purple banner at the top says 'Welcome to Heroku' and 'Now that your account has been set up, here's how to get started.' It includes a 'Show next steps' button and a search bar for 'Filter apps and pipelines'. Below the banner, a list of apps is shown, with 'global-power-plants' selected. The details for this app are visible, including its language (Python), stack (heroku-18), and region (United States). There are also tabs for Pipelines, Spaces, and Settings.

Step 3: Connecting our GitHub repository to Heroku app. Once we have created our Heroku app, we can see several deployment methods. In my opinion, the simplest option is connecting with GitHub since it requires no knowledge of Command Line Interface (CLI) or Virtual environments. After selecting GitHub as our deployment method, we enter the name of the GitHub repository we created in **Step 1**.

All your favorite parts of Medium are now in one sidebar for easy access.



Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Close this window

This is how it looks when the Heroku app successfully connects to our GitHub repository.

Step 4: Manual Deployment. Finally, we deploy our app by the “Deploy Branch” command. This step takes a few minutes.

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. Deployes happen automatically: be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#).

Choose a branch to deploy

master

Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

master

Deploy Branch

Receive code from GitHub



Build master 14f4955e



It takes a few minutes to install all the required packages and deploy the GitHub repository to a website.

Step 5: View App! Once the deployment step is complete we can view the webpage of our application in the browser! The Global Power Plants Dashboard can be viewed [here](#)!

All your favorite parts of Medium are now in one sidebar for easy access.

Choose a branch to deploy

 master Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Deploy a GitHub branch

Deploy the current state of a branch to this app.

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

 master**Deploy Branch**

Receive code from GitHub

Build master `b700b042`

Release phase



Deploy to Heroku



Your app was successfully deployed.

 [View](#)**Data Science****Data Visualization****Dashboard****Dash****Plotly****Follow**

Published in Analytics Vidhya

75K followers · Last published 2 days ago

Analytics Vidhya is a community of Generative AI and Data Science professionals. We are building the next-gen data science ecosystem <https://www.analyticsvidhya.com>

**Follow**

Written by Hanan Ather

13 followers · 15 following

<http://hananather.github.io/>

Responses (1)



All your favorite parts of Medium are now in one sidebar for easy access.



Matt Emrac

Apr 24

...

Fantastic work However do you mind sharing a link to your source code in case someone wanted to try it out please?



[Reply](#)

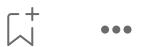
More from Hanan Ather and Analytics Vidhya



In Analytics Vidhya by Harikrishnan N B

Confusion Matrix, Accuracy, Precision, Recall, F1 Score

Binary Classification Metric



All your favorite parts of Medium are now in one sidebar for easy access.



In Analytics Vidhya by Leland Roberts

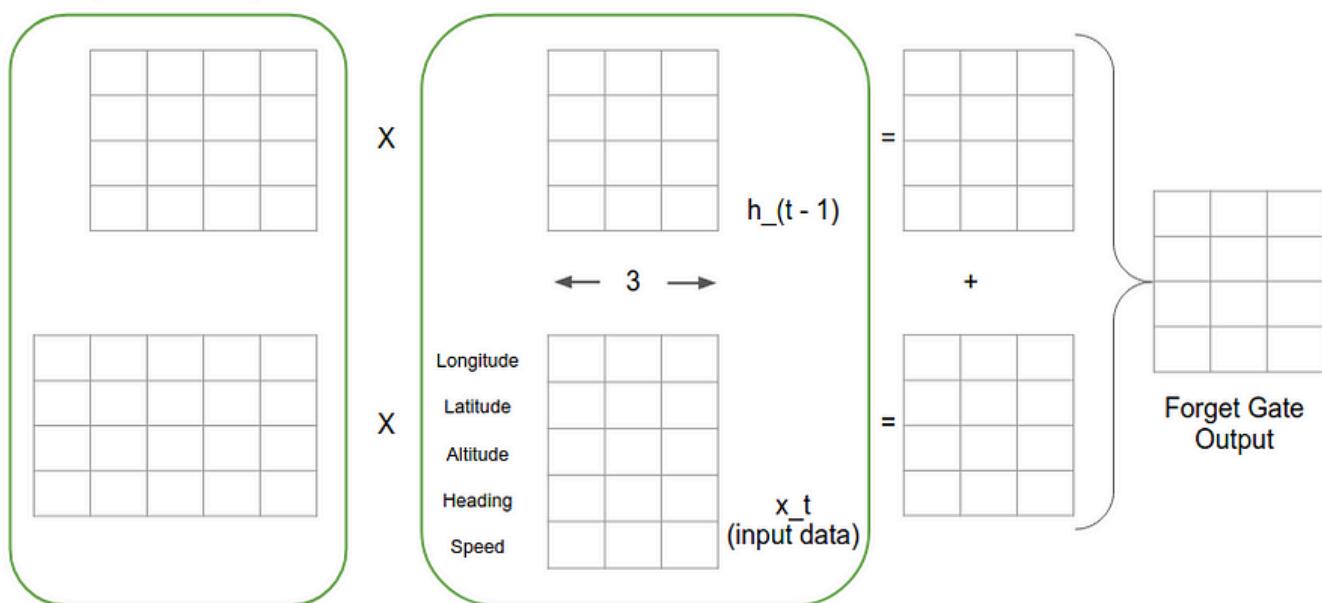
Understanding the Mel Spectrogram (and Other Topics in Signal Processing)

Mar 6, 2020 2.3K 27



Forget Gate Weights
(in LSTM Cell)

Forget Gate Inputs



In Analytics Vidhya by Ryan T. J. J.

LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras

All your favorite parts of Medium are now in one sidebar for easy access.

er guide on LSTMs / RNNs / Keras / whatever. There are SO many them full of false...



...



In Analytics Vidhya by Kia Eisinga

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open source project online? In this blog you will learn...

Jan 26, 2020

2.9K

33



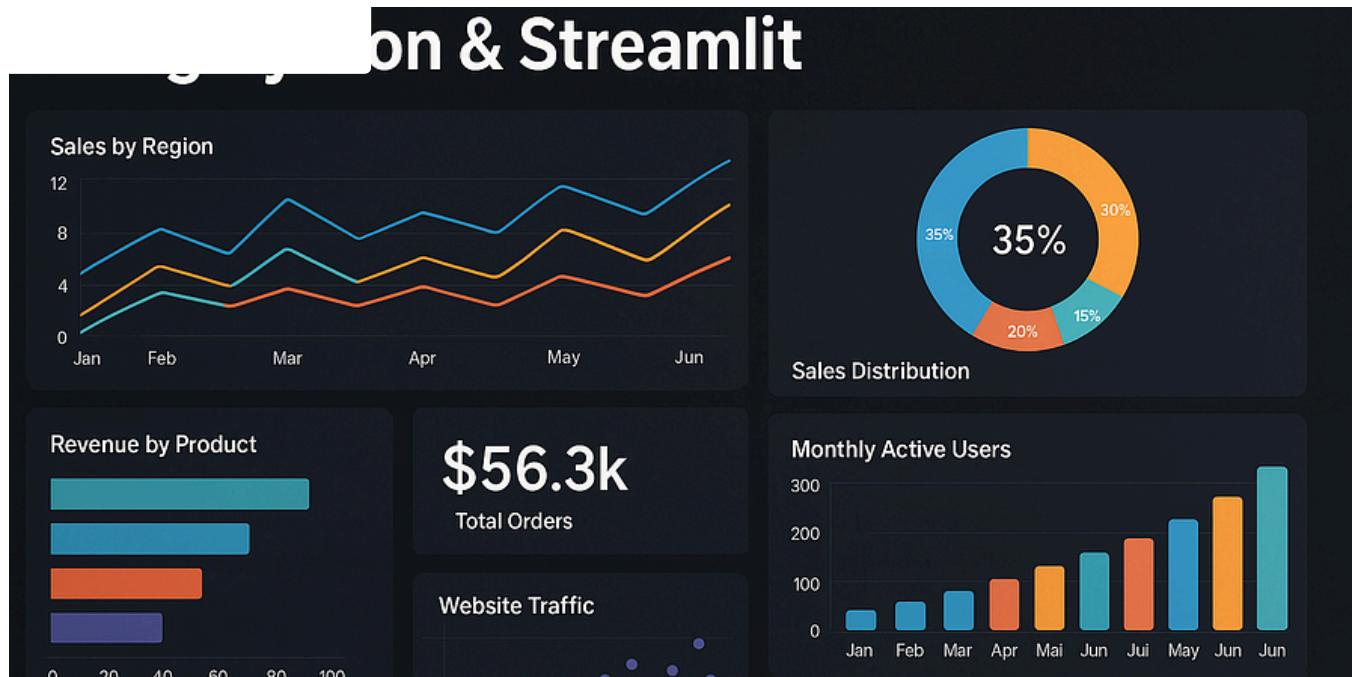
...

See all from Hanan Ather

See all from Analytics Vidhya

All your favorite parts of Medium are now in one sidebar for easy access.

| Medium



Nikulsinh Rajput

The Dashboard I Built in 20 Minutes Using Python & Streamlit

How Streamlit turned my messy CSV files into a polished, interactive data app—with almost no web dev.

Jun 18 46 3



\$46,231

All your favorite parts of Medium are now in one sidebar for easy access.

pr Jun

2020 2021

Sales

Tasks

- Website Traffic
- Launch new website
- Update user profile page
- Create Sales report

Website Traffic

Month	Website Traffic
Jan	100
Feb	150
Apr	200
Jun	300
Jun	400

Bounce Rate

0,40% ▲ 4%

1,00% ▲ 1%

32,1% ▼ 2%



Dashboard Design Lessons I Gained from Exploring 100+ Impressive Dashboard Examples

Patterns, principles, and practical tips for better dashboard design



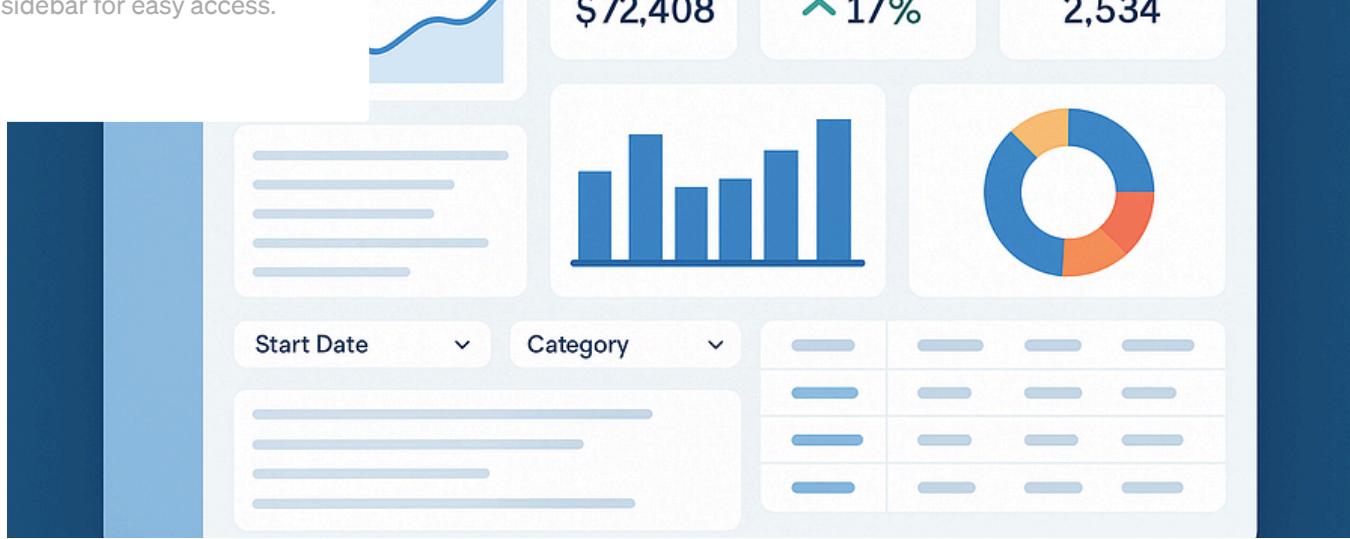
Streamlit + Matplotlib: Visualization Dashboards

Hello,

Apr 21



All your favorite parts of Medium are now in one sidebar for easy access.

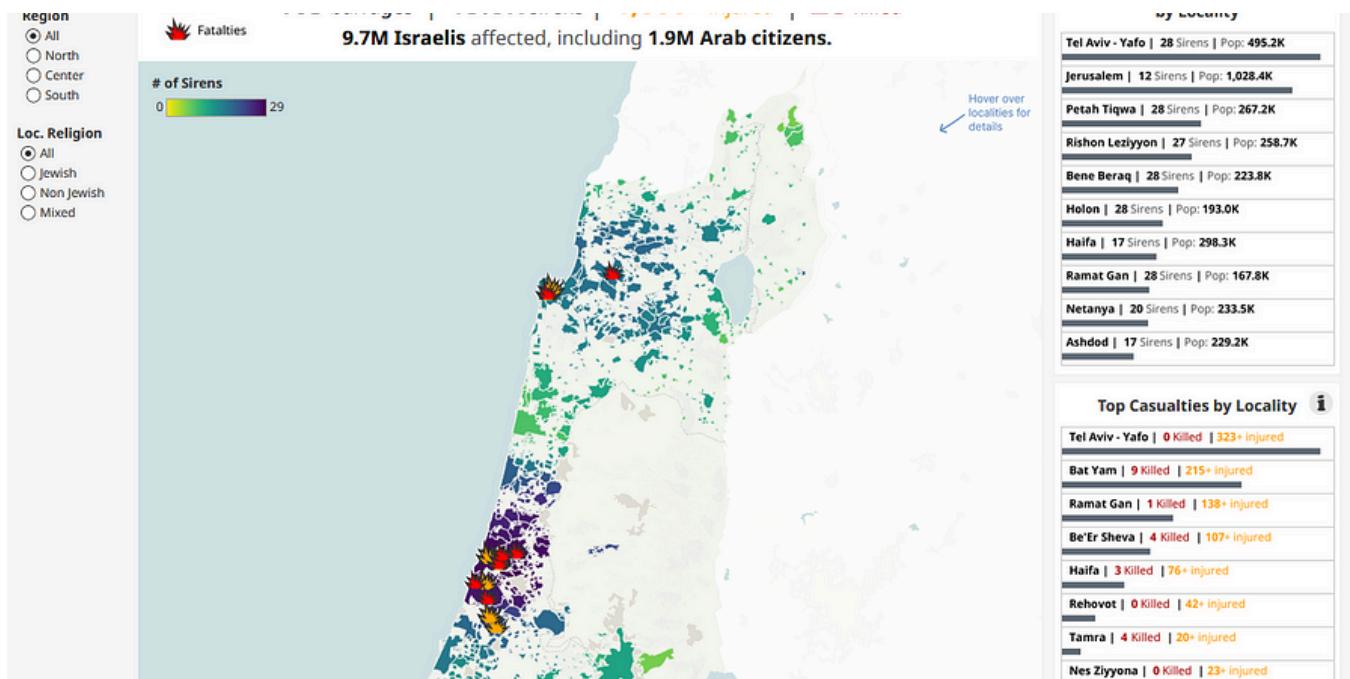


In Level Up Coding by Thomas Reid

Building a modern data dashboard

Using Python and the Gradio library

Jul 7 88 2

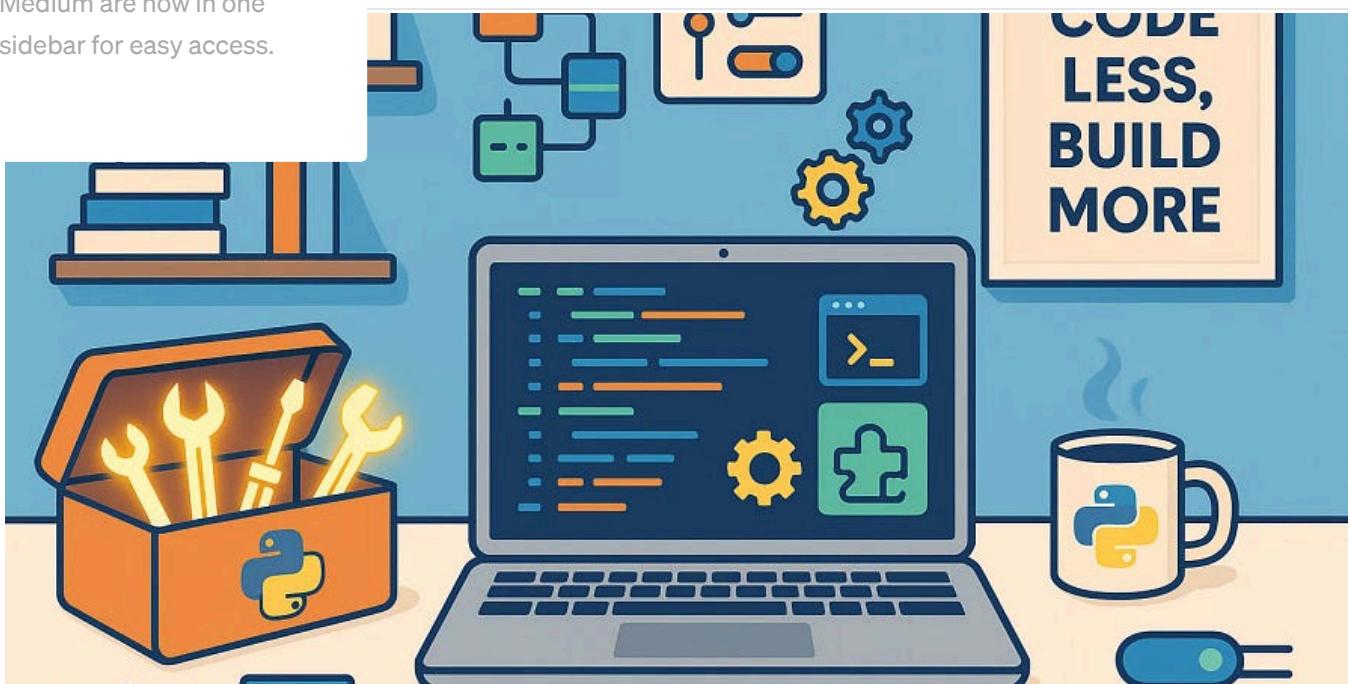


Nir Smilga

The Data behind “Iranian Projectiles” viz

In the days following the Iranian missile attacks, I wanted to create a clear, data-driven view of their impact on Israel's home front. The...

All your favorite parts of Medium are now in one sidebar for easy access.

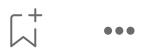


In Python in Plain English by Abdur Rahman

8 Python Libraries So Good, I Stopped Writing My Own Scripts

These will save you time, bugs, and brainpower

Jun 29 2.1K 26



[See more recommendations](#)