

# Modern GUI for Data Science plots using Python



Lukas Schaub · Follow

4 min read · Jan 2



Listen



Share

If you've ever worked with Tkinter, the built-in Python GUI framework, you know that it can be a bit cumbersome to customize the appearance of your widgets. You might have spent hours scouring the documentation, trying to figure out how to change the font, background color, or border of a button or label.

Enter **CustomTkinter**, a library that makes it easy to customize the appearance of your Tkinter widgets. In this blog post, we'll take a closer look at CustomTkinter and how it can help you create more visually appealing Python GUI applications.

I will show you how to create a simple modern-looking GUI to display a matplotlib scatterplot using a slider and custom input values.



0.34 s

SD

4.2K views

gfycat

## Installation

In your console type the following pip command to install customtkinter

```
pip install customtkinter
```

## Basics

First, a root element has to be created and its geometry has to be defined. Then it is already possible to display the resulting root element.

```
import customtkinter as ctk

root = ctk.CTk()
root.geometry("1200x800")
root.update()
"Here all other widgetts will be added"
root.mainloop()
```

The structure is always the same. Invoke a root element, set its geometry, and update the element. Finally, to run the GUI one needs to use the `mainloop()` method.

## Adding Widgets to the root

To add the different widgets to the root window the widgets are called according to their type. `CTkFrame` invokes a frame to the root element. Likewise, `CTkButton`, `CTkEntry`, and `CTKSlider` invoke their respective widgets.

```
"""
Example for a simple 1200x400 window with the following widgets:
Frame, Button, Silder and, Entryfield in darkmode.
"""
import customtkinter as ctk

ctk.set_appearance_mode("dark")

root = ctk.CTk()
root.geometry("1200x400")
root.update()

plotframe = ctk.CTkFrame(master=root,
                        height= root.winfo_height()*0.95,
                        width = root.winfo_width()*0.66,
```

```

        fg_color="darkblue")
    plotframe.place(relx=0.33, rely=0.025)

    Button = ctk.CTkButton(master = root,
                           width=300,
                           height=50)
    Button.place(relx=0.025, rely=0.025)

    TypedInput = ctk.CTkEntry(master=root,
                              width=300,
                              height=50,

```

[Open in app](#) ↗

[Sign up](#)
[Sign In](#)


```

        height=20)
    Slider.place(relx= 0.025, rely=0.90)

    root.mainloop()

```

Note that whenever a new widget is added it needs a master which basically provides the surface the widget is placed on. Next, the geometry is set using the width and height keywords. Finally, foreground colors can be set by adding the **fg\_color** keyword. Then, the element needs to be placed. This happens via the `place()` method and can be done as shown here with the relative x and y coordinates `relx` and `rely`.

### Add functionality via the “command” keyword

It is possible to link previously defined functions to a button or slider widget. The example below will print a random number between 0–100 in the console every time the button is pressed.

```

"""
Example for a 400x400 window in darkmode with a button that
generates a number between 0 and 100 everytime it is pressed.
"""
import customtkinter as ctk
import numpy as np

def random_number():
    print(np.random.randint(0,100))

ctk.set_appearance_mode("dark")

```

```

root = ctk.CTk()
root.geometry("400x400")
root.update()

Button = ctk.CTkButton(master = root,
                        width=50,
                        height=50,
                        text="Gen",
                        command=random_number)
Button.place(relx=0.425, rely=0.425)

root.mainloop()

```

Note that the command function needs to be previously defined otherwise the paradigm should be changed to work with an object-oriented programming approach.

### Object-oriented programming approach

Think of the window as your Object. The `__init__()` method is used to set up the window that is later presented to the user. By adding custom methods we can organize the structure of the window neatly and have a great overview of all of the functions which are used for the GUI.

### Linking matplotlib plots to customtkinter frames

This can be done using “`from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg`”. Generating a figure via the subplots method and then modifying it to a canvas. The canvas can then be placed on top of the other frame.

```

import customtkinter as ctk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# generate root
root = ctk.CTk()
root.geometry("800x400")

# generate random numbers for the plot
x,y,s,c = np.random.rand(4,100)

# generate the figure and plot object which will be linked to the root element
fig, ax = plt.subplots()
fig.set_size_inches(8,4)
ax.scatter(x,y,s*50,c)

```

```

ax.axis("off")
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, wspace=0, hspace=0)
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.draw()
canvas.get_tk_widget().place(relx=0.15, rely=0.15)

# initiate the window
root.mainloop()

```

## Example: Customtkinter with dynamic matplotlib scatterplot in modern look

Combining everything learned the following code is used to generate what was shown before. The `__init__()` method sets up the GUI where the methods `update_window()` and `update_surface()` change the provided matplotlib scatterplot. Using global variables for `x`, `y`, `s`, and `c` it is possible to update them dynamically.

```

import customtkinter as ctk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

class ctkApp:

    def __init__(self):
        ctk.set_appearance_mode("dark")
        self.root = ctk.CTk()
        self.root.geometry("1200x400+200x200")
        self.root.title("Dynamic Scatterplot")
        self.root.update()
        self.frame = ctk.CTkFrame(master=self.root,
                                   height= self.root.winfo_height()*0.95,
                                   width = self.root.winfo_width()*0.66,
                                   fg_color="darkblue")
        self.frame.place(relx=0.33, rely=0.025)
        self.input = ctk.CTkEntry(master=self.root,
                                   placeholder_text=100,
                                   justify='center',
                                   width=300,
                                   height=50,
                                   fg_color="darkblue")
        self.input.insert(0, 100)
        self.input.place(relx=0.025, rely=0.5)
        self.slider = ctk.CTkSlider(master=self.root,
                                     width=300,

```

```

        height=20,
        from_=1,
        to=1000,
        number_of_steps=999,
        command=self.update_surface)

self.slider.place(relx= 0.025, rely=0.75)
self.button = ctk.CTkButton(master = self.root,
                             text="Update Graph",
                             width=300,
                             height=50,
                             command=self.update_window)

self.button.place(relx=0.025, rely=0.25)
self.root.mainloop()

def update_window(self):
    fig, ax = plt.subplots()
    fig.set_size_inches(11,5.3)
    global x,y,s,c
    x,y,s,c = np.random.rand(4,int(self.input.get()))
    ax.scatter(x,y,s*self.slider.get(),c)
    ax.axis("off")
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, wspace=0, hspace=0)
    canvas = FigureCanvasTkAgg(fig, master=self.root)
    canvas.draw()
    canvas.get_tk_widget().place(relx=0.33, rely=0.025)
    self.root.update()

def update_surface(self, other):
    fig, ax = plt.subplots()
    fig.set_size_inches(11,5.3)
    ax.scatter(x,y,s*self.slider.get(),c)
    ax.axis("off")
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, wspace=0, hspace=0)
    canvas = FigureCanvasTkAgg(fig, master=self.root)
    canvas.draw()
    canvas.get_tk_widget().place(relx=0.33, rely=0.025)
    self.root.update()

if __name__ == "__main__":
    CTK_Window = ctkApp()

```

The code then yields the following GUI:



Dynamic Scatterplot that can generate new graphs on the fly

## Final Thoughts

It is not as hard to get a good-looking GUI for a quick project that needs to be visualized. Make sure you use customtkinter and have this article bookmarked so you can check it out again if you need guidance. If you are planning on building more complex GUIs it would be helpful to work with OOP and make sure to check out the [customtkinter documentation](#).

Data Science

Data Visualization

GUI

Matplotlib

Visualization



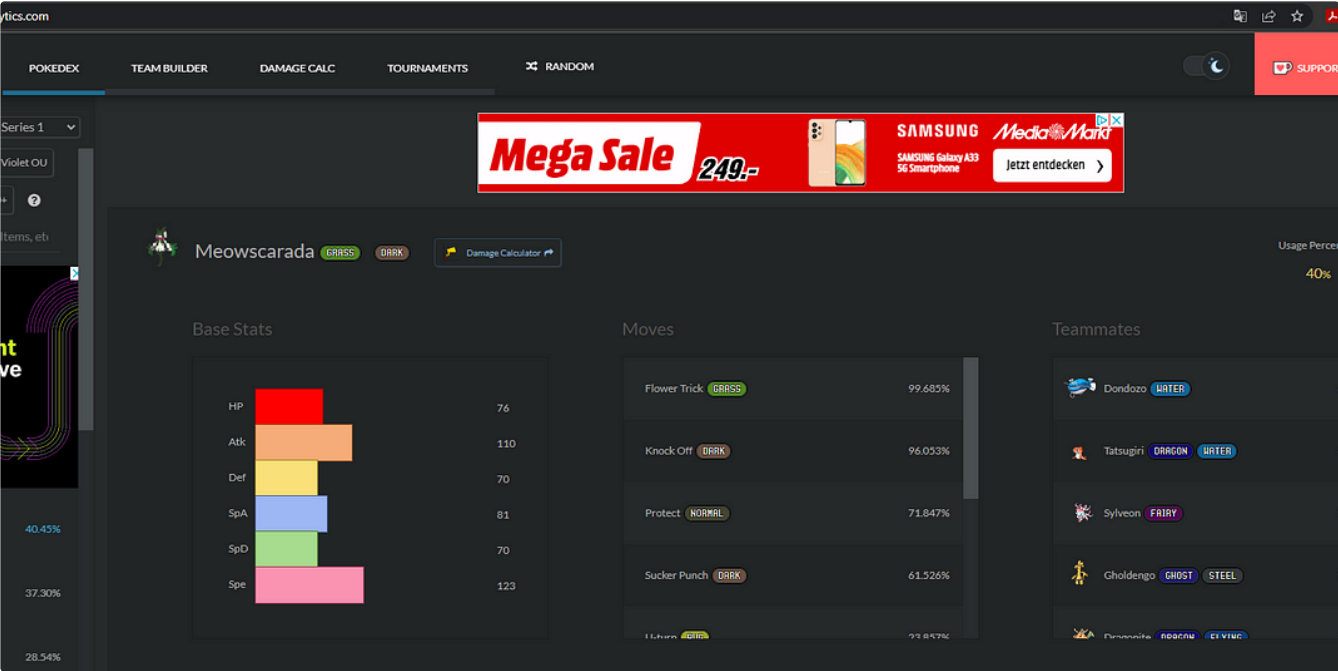
Follow


## Written by Lukas Schaub

33 Followers

Programming and Data addict. With many other interests including languages and cooking.

### More from Lukas Schaub



 Lukas Schaub

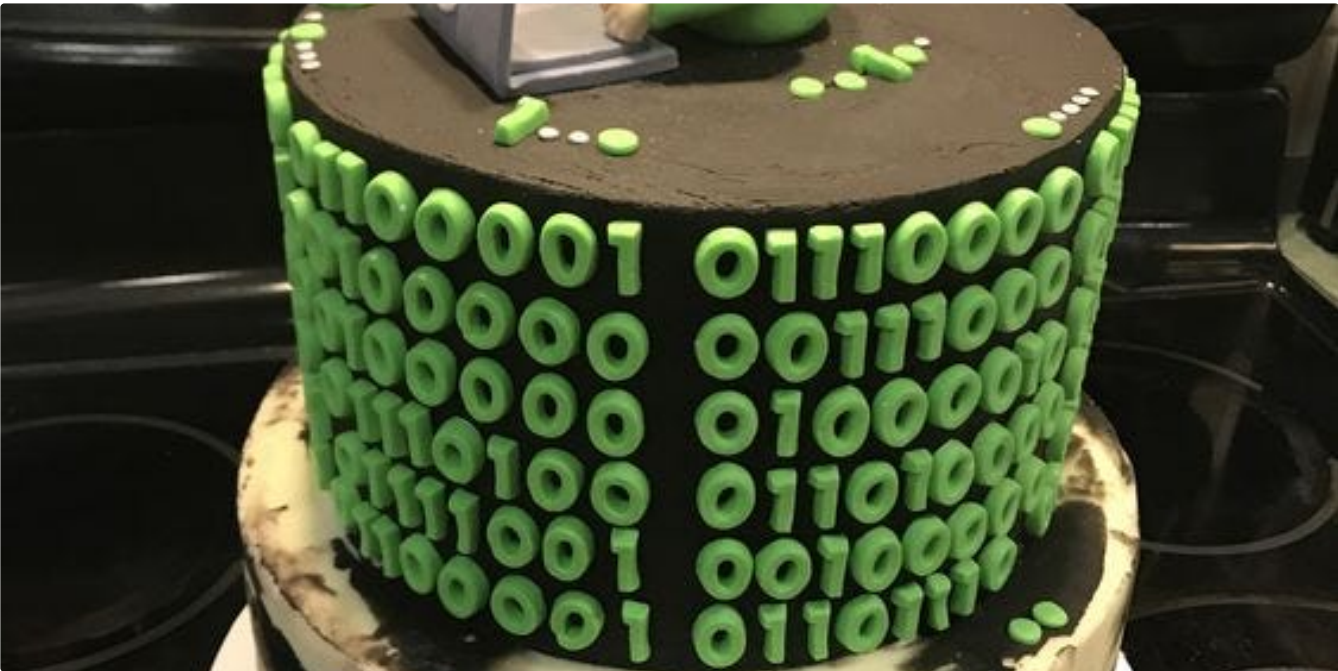
# Play Pokemon like a Data Scientist — Part 4: Probabilistic Team Building


Building the best Pokemon Team using math is possible. Today I want to show you how this can be done using pikalytics, BeautifulSoup, and...

5 min read · Jan 15







 Lukas Schaub

## Decorators for Python Data Science



## The Icing on the Cake

4 min read · Apr 13



10



write a 4min blogpost for medium. The topic should be "an introduction to numpy". It should show its main advantages. It has to describe the installation process of the package. Introduce the most important functions of numpy and give an good example where to use it.



Numpy is a powerful open-source numerical computation library for Python. It is widely used in scientific computing and data analysis, and has become a go-to tool for many data scientists and researchers.



One of the main advantages of numpy is its ability to perform efficient operations on large arrays and matrices of data. This makes it much faster and more efficient than working with large datasets using Python's built-in list and tuple data types.



Lukas Schaub

## An introduction to numpy ... according to chatGPT

I want to write more blogposts and I know I want to share some more knowledge about common libraries which are used for data science and...

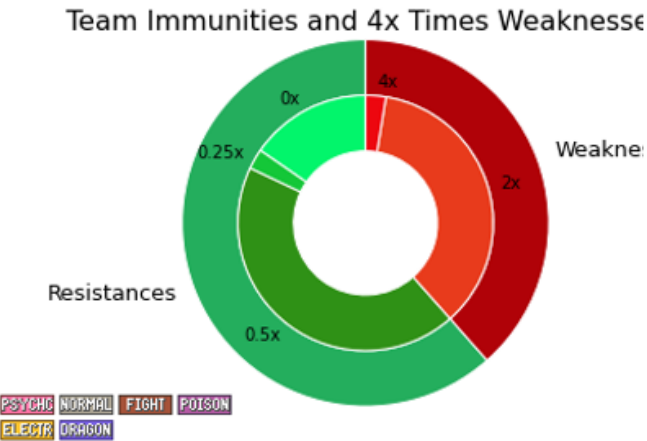
4 min read · Jan 6




4



```
1: Greninja = Pokemon("Greninja")
2: Greninja.get_weakness_and_resistance()
3:
4: ['Fire', 'Water', 'Ice', 'Ghost', 'Dark', 'Steel'],
5: ['Electric', 'Grass', 'Fighting', 'Bug', 'Fairy'],
6: ['Psychic']}
```



 Lukas Schaub

## Play Pokemon like a Data Scientist - Part 2: Team Strengths and Weaknesses

This is Part 2 where we elaborate on the previously written code, which will allow us to check our Pokemon Teams Strengths and Weaknesses...

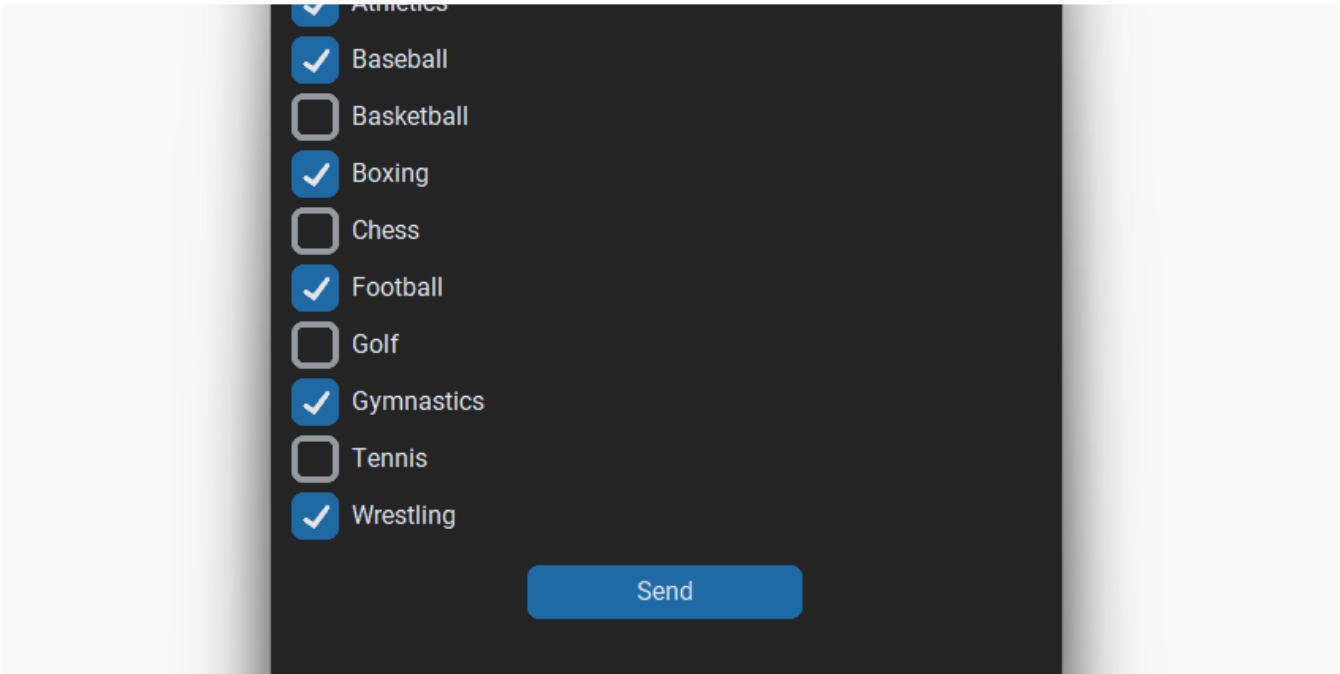
5 min read · Nov 7, 2022

 12 



See all from Lukas Schaub

Recommended from Medium



 Better Everything

## Dynamically adding Checkboxes—Stylish GUIs with Python CustomTkinter #2

Add multiple checkboxes to a Custom Tkinter GUI by looping over a list of strings in Python.

★ · 4 min read · Feb 22

 56 



 Olha Bahaieva in UX Designers Club

# 15 Amazing Dashboard Designs for Your Work Inspiration

The dashboard designs of the week.

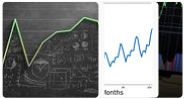
★ · 3 min read · Jan 28



2



## Lists



### Predictive Modeling w/ Python

18 stories · 154 saves



### New\_Reading\_List

174 stories · 33 saves



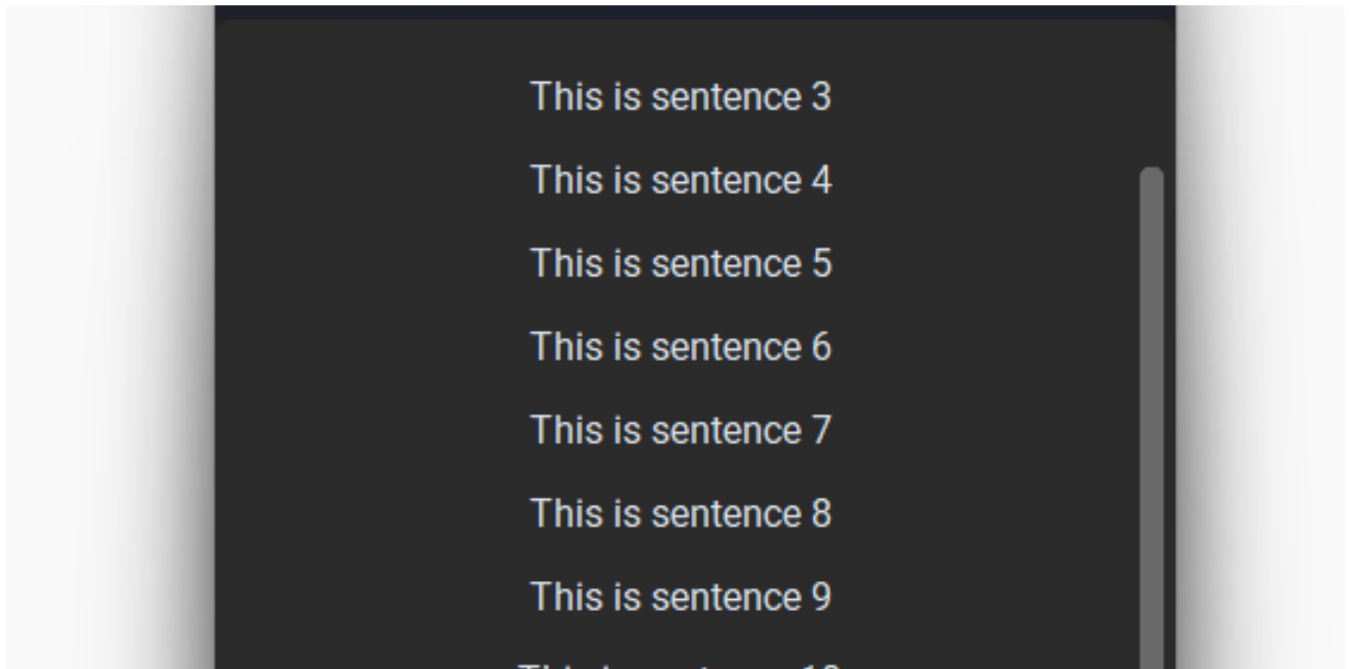
### Practical Guides to Machine Learning

10 stories · 176 saves



### Coding & Development

11 stories · 68 saves



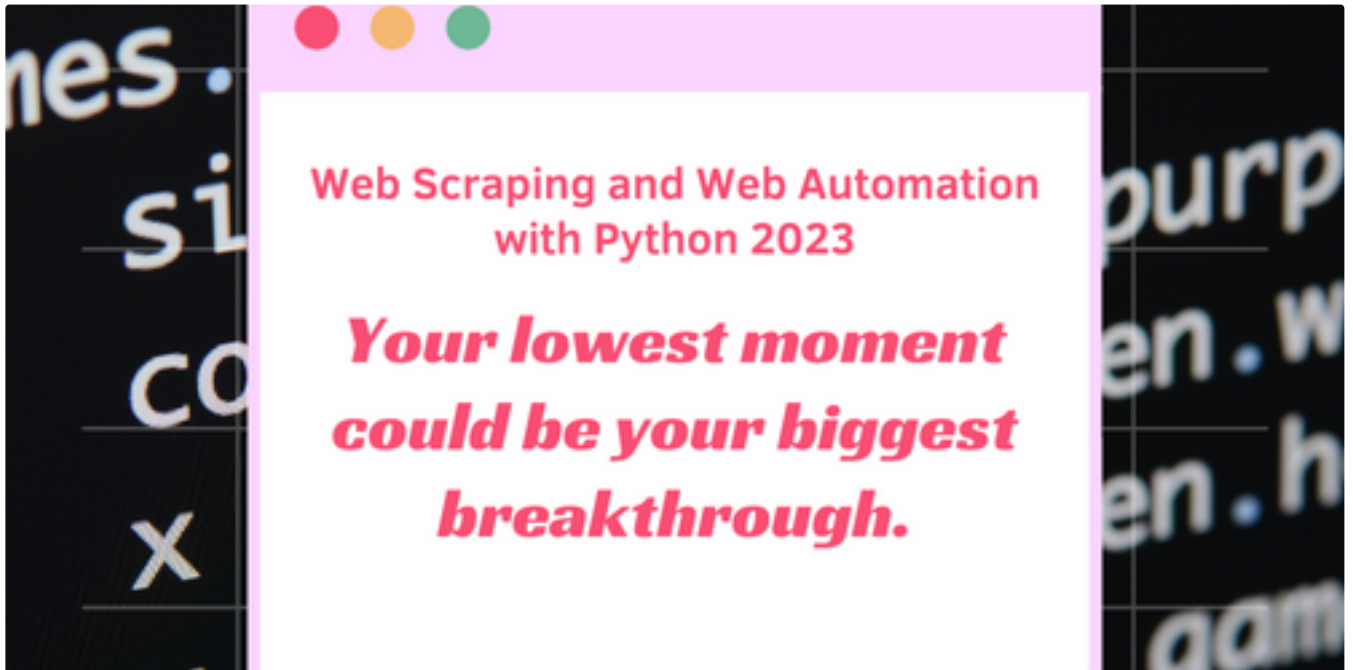
Better Everything

## Fullscreen GUIs with Scrollbar - Stylish Python CustomTkinter GUIs #4

Part 4 of the Stylish Python CustomTkinter GUIs series: making a fullscreen, maximized GUI with a scrollbar.

★ · 5 min read · Mar 8

👏 26 💬



 Builescu Daniel in Towards Dev

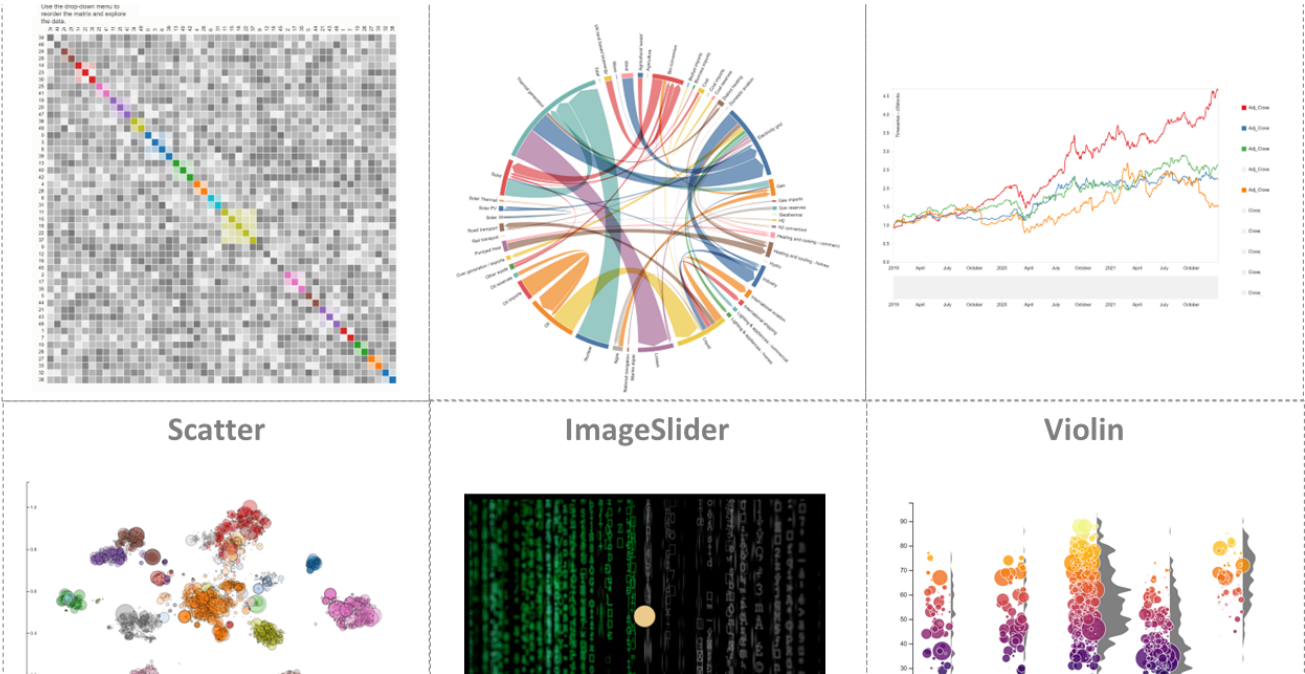
## Web Scraping and Web Automation with Python 2023

Introduction to Web Scraping and Web Automation with Python 2023

★ · 10 min read · Apr 14

👏 51 💬





Scatter

ImageSlider

Violin



Erdogan Taskesen in Towards Data Science

# D3Blocks: The Python Library to Create Interactive and Standalone D3js Charts.

Create interactive, and stand-alone charts that are built on the graphics of d3 javascript (d3js) but configurable with Python.

🌟 · 11 min read · Sep 22, 2022



835



6



Rhett Allain

## How to Use Python as Your Physics Calculator

Remember when humans used to calculate all their stuff on paper? Or maybe they were super-tech savvy and instead used a SLIDE-RULE? OK...

★ · 9 min read · Jan 27



52



2



See more recommendations