**How to write Web apps using simple Python for Data Scientists?**

**Convert your Data Science Projects into cool apps easily without knowing any web frameworks**

<https://towardsdatascience.com/how-to-write-web-apps-using-simple-python-for-data-scientists-a227a1a01582>

[[](https://towardsdatascience.com/@rahul_agarwal?source=post_page-----a227a1a01582----------------------)](https://towardsdatascience.com/@rahul_agarwal?source=post_page-----a227a1a01582----------------------)

[Rahul Agarwal](https://towardsdatascience.com/@rahul_agarwal?source=post_page-----a227a1a01582----------------------)

[Follow](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fhow-to-write-web-apps-using-simple-python-for-data-scientists-a227a1a01582&source=-e8cce06956c9-------------------------follow_byline-)

[Oct 13, 2019](https://towardsdatascience.com/how-to-write-web-apps-using-simple-python-for-data-scientists-a227a1a01582?source=post_page-----a227a1a01582----------------------) · 9 min read

A Machine Learning project is never really complete if we don’t have a good way to showcase it.

While in the past, a well-made visualization or a small PPT used to be enough for showcasing a data science project, with the advent of dashboarding tools like RShiny and Dash, a good data scientist needs to have a fair bit of knowledge of web frameworks to get along.

And Web frameworks are hard to learn. I still get confused in all that HTML, CSS, and Javascript with all the hit and trials, for something seemingly simple to do.

Not to mention the many ways to do the same thing, making it confusing for us data science folks for whom web development is a secondary skill.

***So, are we doomed to learn web frameworks? Or to call our developer friend for silly doubts in the middle of the night?***

This is where StreamLit comes in and delivers on its promise to create web apps just using Python.

Zen of Python: Simple is better than complex and Streamlit makes it dead simple to create apps.

***This post is about understanding how to create apps that support data science projects using Streamlit.***

To understand more about the architecture and the thought process that led to streamlit, have a look at this excellent [post](https://towardsdatascience.com/coding-ml-tools-like-you-code-ml-models-ddba3357eace) by one of the original developers/founder [Adrien Treuille](https://medium.com/u/39dfc90d7a34?source=post_page-----a227a1a01582----------------------).

If you want to deploy a streamlit app on Amazon ec2 check out my next [post](https://towardsdatascience.com/how-to-deploy-a-streamlit-app-using-an-amazon-free-ec2-instance-416a41f69dc3).

**Installation**

Installation is as simple as running the command:

pip install streamlit

To see if our installation is successful, we can just run:

streamlit hello

This should show you a screen that says:



You can go to the local URL: localhost:8501 in your browser to see a Streamlit app in action. The developers have provided some cool demos that you can play with. Do take your time and feel the power of the tool before coming back.



**Streamlit Hello World**

Streamlit aims to make app development easy using simple Python.

So let us write a simple app to see if it delivers on that promise.

Here I start with a simple app which we will call the Hello World of streamlit. Just paste the code given below in a file named helloworld.py

import streamlit as stx = st.slider('x')  
st.write(x, 'squared is', x \* x)

And, on the terminal run:

streamlit run helloworld.py

And voila, you should be able to see a simple app in action in your browser at localhost:8501 that allows you to move a slider and gives the result.



A Simple slider widget app

It was pretty easy. In the above app, we used two features from Streamlit:

* the st.slider widget that we can slide to change the output of the web app.
* and the versatile st.write command. I am amazed at how it can write anything from charts, dataframes, and simple text. More on this later.

***Important: Remember that every time we change the widget value, the whole app runs from top to bottom.***

**Streamlit Widgets**

Widgets provide us a way to control our app. The best place to read about the widgets is the [API reference](https://streamlit.io/docs/api.html) documentation itself but I will describe some most prominent ones that you might end up using.

**1. Slider**

**streamlit.slider(*label*, *min\_value=None*, *max\_value=None*, *value=None*, *step=None*, *format=None*)**

We already saw st.slider in action above. It can be used with min\_value,max\_value, and step for getting inputs in a range.

**2. Text Input**

The simplest way to get user input be it some URL input or some text input for sentiment analysis. It just needs a single label for naming the textbox.

import streamlit as sturl = st.text\_input('Enter URL')  
st.write('The Entered URL is', url)

This is how the app looks:



A Simple text\_input widget app

**Tip:**You can just change the file helloworld.py and refresh the browser. The way I work is to open and changehelloworld.py in sublime text and see the changes in the browser side by side.

**3. Checkbox**

One use case for checkboxes is to hide or show/hide a specific section in an app. Another could be setting up a boolean value in the parameters for a function.[st.checkbox()](https://streamlit.io/docs/api.html#streamlit.checkbox) takes a single argument, which is the widget label. In this app, the checkbox is used to toggle a conditional statement.

import streamlit as st  
import pandas as pd  
import numpy as npdf = pd.read\_csv("football\_data.csv")  
if st.checkbox('Show dataframe'):  
 st.write(df)



A Simple checkbox widget app

**4. SelectBox**

We can use [st.selectbox](https://streamlit.io/docs/api.html" \l "streamlit.selectbox" \t "_blank) to choose from a series or a list. Normally a use case is to use it as a simple dropdown to select values from a list.

import streamlit as st  
import pandas as pd  
import numpy as npdf = pd.read\_csv("football\_data.csv")option = st.selectbox(  
 'Which Club do you like best?',  
 df['Club'].unique())'You selected: ', option



A Simple dropdown/selectbox widget app

**5. MultiSelect**

We can also use multiple values from a dropdown. Here we use st.multiselect to get multiple values as a list in the variable options

import streamlit as st  
import pandas as pd  
import numpy as npdf = pd.read\_csv("football\_data.csv")options = st.multiselect(  
 'What are your favorite clubs?', df['Club'].unique())st.write('You selected:', options)



A Simple multiselect widget app

**Creating Our Simple App Step by Step**

So much for understanding the important widgets. Now, we are going to create a simple app using multiple widgets at once.

To start simple, we will try to visualize our football data using streamlit. It is pretty much simple to do this with the help of the above widgets.

import streamlit as st  
import pandas as pd  
import numpy as npdf = pd.read\_csv("football\_data.csv")clubs = st.multiselect('Show Player for clubs?', df['Club'].unique())nationalities = st.multiselect('Show Player from Nationalities?', df['Nationality'].unique())# Filter dataframe  
new\_df = df[(df['Club'].isin(clubs)) & (df['Nationality'].isin(nationalities))]# write dataframe to screen  
st.write(new\_df)

Our simple app looks like:



Using multiple widgets in conjunction

That was easy. But it seems pretty basic right now. Can we add some charts?

Streamlit currently supports many libraries for plotting.***Plotly, Bokeh, Matplotlib, Altair, and Vega charts*** being some of them. ***Plotly Express*** also works, although they didn’t specify it in the docs. It also has some inbuilt chart types that are “native” to Streamlit, like st.line\_chart and st.area\_chart.

We will work with plotly\_express here. Here is the code for our simple app. We just used four calls to streamlit. Rest is all simple python.

import streamlit as st  
import pandas as pd  
import numpy as np  
import plotly\_express as pxdf = pd.read\_csv("football\_data.csv")**clubs = st.multiselect('Show Player for clubs?', df['Club'].unique())  
nationalities = st.multiselect('Show Player from Nationalities?', df['Nationality'].unique())**new\_df = df[(df['Club'].isin(clubs)) & (df['Nationality'].isin(nationalities))]  
**st.write(new\_df)**# create figure using plotly express  
fig = px.scatter(new\_df, x ='Overall',y='Age',color='Name')# Plot!  
**st.plotly\_chart(fig)**



Adding charts

**Improvements**

In the start we said that each time we change any widget, the whole app runs from start to end. This is not feasible when we create apps that will serve deep learning models or complicated machine learning models. Streamlit covers us in this aspect by introducing ***Caching***.

**1. Caching**

In our simple app. We read the pandas dataframe again and again whenever a value changes. While it works for the small data we have, it will not work for big data or when we have to do a lot of processing on the data. Let us use caching using the st.cache decorator function in streamlit like below.

import streamlit as st  
import pandas as pd  
import numpy as np  
import plotly\_express as px**df = st.cache(pd.read\_csv)("football\_data.csv")**

Or for more complex and time taking functions that need to run only once(think loading big Deep Learning models), using:

@st.cache  
def complex\_func(a,b):  
 DO SOMETHING COMPLEX# Won't run again and again.  
complex\_func(a,b)

When we mark a function with Streamlit’s cache decorator, whenever the function is called streamlit checks the input parameters that you called the function with.

***If this is the first time Streamlit has seen these params, it runs the function and stores the result in a local cache.***

When the function is called the next time, if those params have not changed, Streamlit knows it can skip executing the function altogether. It just uses the results from the cache.

**2. Sidebar**

For a cleaner look based on your preference, you might want to move your widgets into a sidebar, something like Rshiny dashboards. ***This is pretty simple. Just add st.sidebar in your widget’s code.***

import streamlit as st  
import pandas as pd  
import numpy as np  
import plotly\_express as pxdf = st.cache(pd.read\_csv)("football\_data.csv")clubs = **st.sidebar.multiselect**('Show Player for clubs?', df['Club'].unique())  
nationalities = **st.sidebar.multiselect**('Show Player from Nationalities?', df['Nationality'].unique())new\_df = df[(df['Club'].isin(clubs)) & (df['Nationality'].isin(nationalities))]  
st.write(new\_df)# Create distplot with custom bin\_size  
fig = px.scatter(new\_df, x ='Overall',y='Age',color='Name')# Plot!  
st.plotly\_chart(fig)



Move widgets to the sidebar

**3. Markdown?**

I love writing in Markdown. I find it less verbose than HTML and much more suited for data science work. So, can we use Markdown with the streamlit app?

Yes, we can. There are a couple of ways to do this. In my view, the best one is to use [Magic commands](https://streamlit.io/docs/api.html#id1). Magic commands allow you to write markdown as easily as comments. You could also have used the command st.markdown

import streamlit as st  
import pandas as pd  
import numpy as np  
import plotly\_express as px'''  
# Club and Nationality AppThis very simple webapp allows you to select and visualize players from certain clubs and certain nationalities.  
'''  
df = st.cache(pd.read\_csv)("football\_data.csv")clubs = st.sidebar.multiselect('Show Player for clubs?', df['Club'].unique())  
nationalities = st.sidebar.multiselect('Show Player from Nationalities?', df['Nationality'].unique())new\_df = df[(df['Club'].isin(clubs)) & (df['Nationality'].isin(nationalities))]  
st.write(new\_df)# Create distplot with custom bin\_size  
fig = px.scatter(new\_df, x ='Overall',y='Age',color='Name')'''  
### Here is a simple chart between player age and overall  
'''st.plotly\_chart(fig)



Our final App Demo

**Conclusion**

Streamlit has democratized the whole process to create apps, and I couldn’t recommend it more.

In this post, we created a simple web app. But the possibilities are endless. To give an example here is [face GAN](https://research.nvidia.com/publication/2017-10_Progressive-Growing-of) from the streamlit site. And it works by just using the same guiding ideas of widgets and caching.



I love the default colors and styles that the developers have used, and I found it much more comfortable than using Dash, which I was using until now for my demos. You can also include [audio](https://streamlit.io/docs/api.html#display-interactive-widgets) and video in your streamlit apps.

**On top of that, Streamlit is a free and open-source rather than a proprietary web app that just works out of the box.**

In the past, I had to reach out to my developer friends for any single change in a demo or presentation; now it is relatively trivial to do that.

I aim to use it more in my workflow from now on, and considering the capabilities it provides without all the hard work, I think you should too.

I don’t have an idea if it will perform well in a production environment yet, but its a boon for the small proof of concept projects and demos. I aim to use it more in my workflow from now on, and considering the capabilities it provides without all the hard work, I think you should too.

You can find the full code for the final app [here](https://github.com/MLWhiz/data_science_blogs/tree/master/streamlit_football_demo). If you want to deploy this app on Amazon ec2 check out my next [post](https://towardsdatascience.com/how-to-deploy-a-streamlit-app-using-an-amazon-free-ec2-instance-416a41f69dc3).