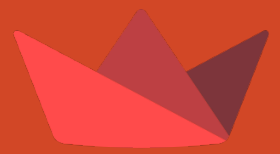
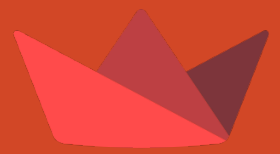
Streamlit Tutorial

Part II



Model 3

Progress Bar, Widget, Caching, Multipages, and Themes



Module 3

- Progress Bar Widget
- Caching
- Multipages
- Session & Callback

Progress Bar

- When adding long running computations to an app, you can use **st.progress()** to display status in real time.

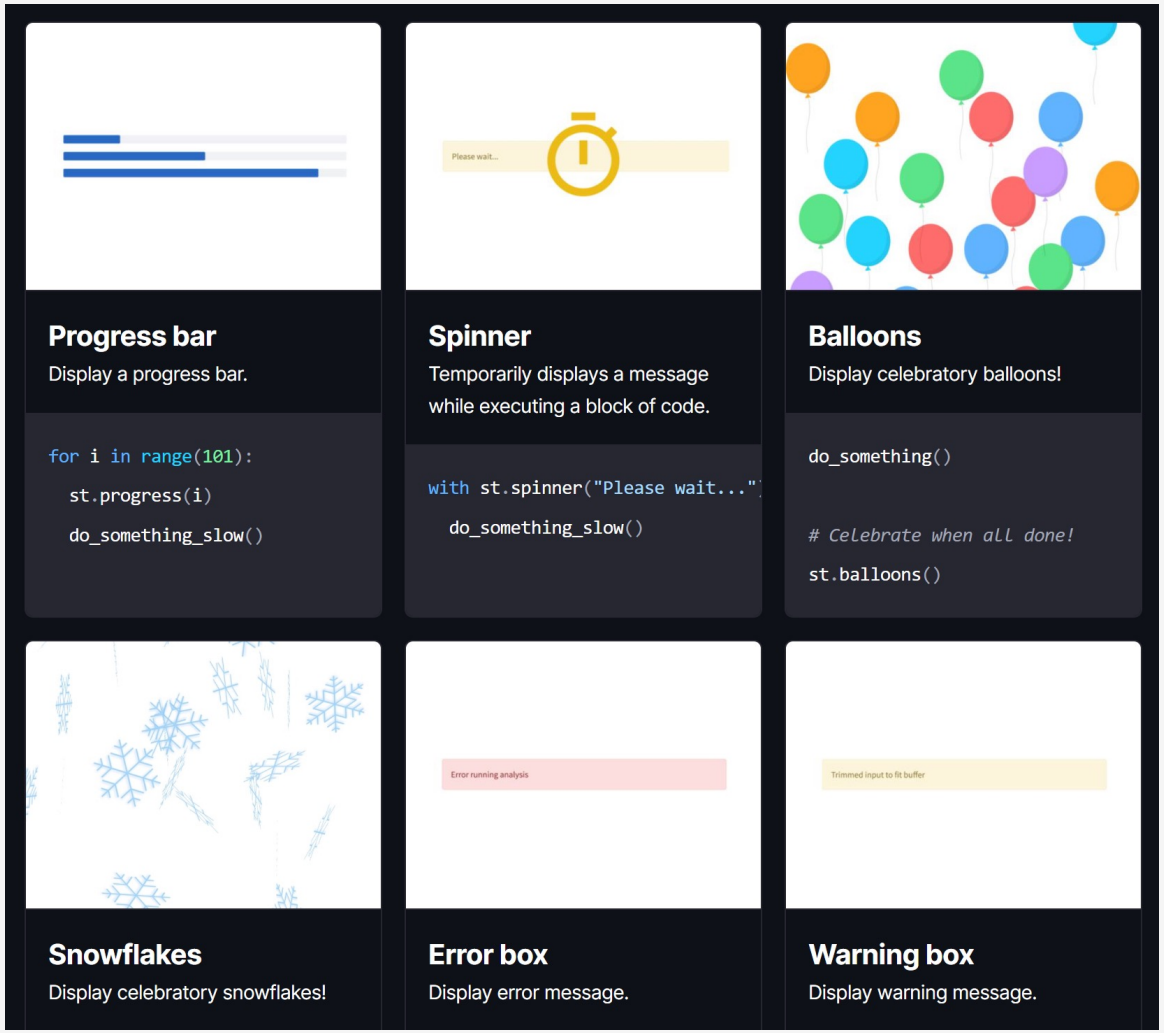
- First, let's import time. We're going to use the **time.sleep()** method to simulate a long running computation.

- Now, let's create a progress bar:

- Alternative to Progress Bar: https://docs.streamlit.io/library/api-reference/status

```
import streamlit as st
import time

'Starting a long computation...'

# Add a placeholder
latest_iteration = st.empty()
bar = st.progress(0)

for i in range(100):
  # Update the progress bar with each iteration.
  latest_iteration.text(f'Iteration {i+1}')
  bar.progress(i + 1)
  time.sleep(0.1)

'...and now we\'re done!'
```

Status Widget



Caching

- The Streamlit cache allows your app to execute quickly even when loading data from the web, manipulating large datasets, or performing expensive computations.

- To use the cache, wrap functions with the **@st.cache** decorator:

- Other options:

```
st.cache(func=None, persist=False,
allow_output_mutation=False,
show_spinner=True,
suppress_st_warning=False,
hash_funcs=None, max_entries=None,
ttl=None)
```

```python
import streamlit as st

@st.cache  # 👈 This function will be cached
def my_slow_function(arg1, arg2):
    # Do something really slow in here!
    return the_output

# Persist data on disk
@st.cache(persist=True)
def fetch_and_clean_data(url):
     # Fetch data from URL here, and then clean it up.
     return data
```

How The @st.cache Works?

When you mark a function with the **@st.cache decorator**, it tells Streamlit that whenever the function is called it needs to check a few things:

- The input parameters that you called the function with
- The value of any external variable used in the function
- The body of the function
- The body of any function used inside the cached function

If this is the first time Streamlit has seen these four components with these exact values and in this exact combination and order, it runs the function and stores the result in a local cache. Then, next time the cached function is called, if none of these components changed, Streamlit will skip executing the function altogether and, instead, return the output previously stored in the cache.

For more information about the Streamlit cache, its configuration parameters, and its limitations, see Caching.

Caching

```
@st.cache
 def fetch_and_clean_data(url):
     # Fetch data from URL here, and then clean it up.
     return data

d1 = fetch_and_clean_data(DATA_URL_1)
# Actually executes the function, since this is the first time it was
# encountered.

d2 = fetch_and_clean_data(DATA_URL_1)
# Does not execute the function. Instead, returns its previously computed
# value. This means that now the data in d1 is the same as in d2.

d3 = fetch_and_clean_data(DATA_URL_2)
# This is a different URL, so the function executes.
```

Multi Pages

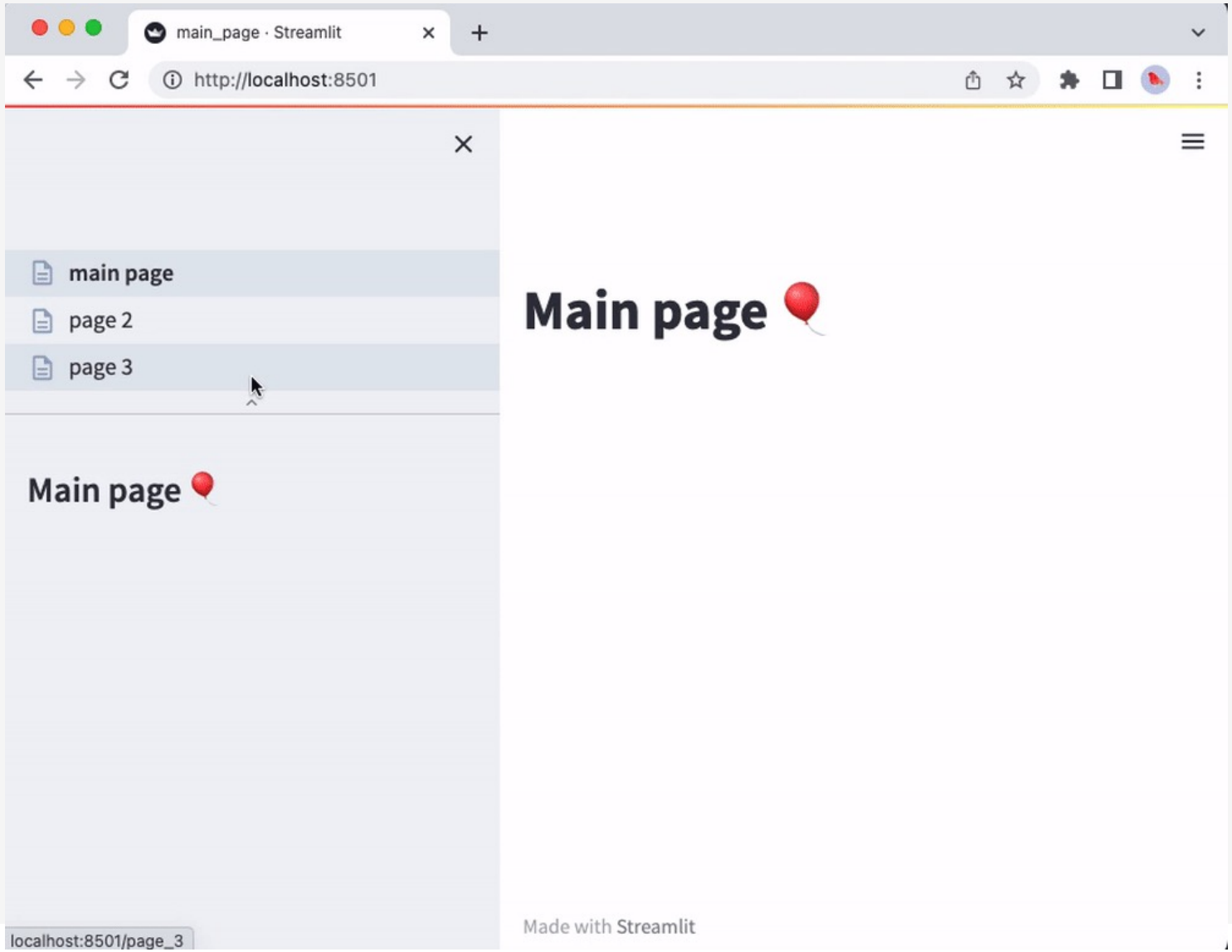
- Multipage app is as easy as building a single-page app! Just add more pages to an existing app as follows:
- In the folder containing your main script, create a new **pages** folder. Let's say your main script is named main_page.py.
- Add new .py files in the pages folder to add more pages to your app.
- Run streamlit run main_page.py as usual.

```
# main_page.py
import streamlit as st

st.markdown("# Main page 🎈")
st.sidebar.markdown("# Main page 🎈")
```

```
# pages/second_page.py

import streamlit as st

st.markdown("# Page 2 ❄️")
st.sidebar.markdown("# Page 2 ❄️")
```

```
# page/third_page.py
import streamlit as st

st.markdown("# Page 3 🎉")
st.sidebar.markdown("# Page 3 🎉")
```

Multipages


main_page · Streamlit
http://localhost:8501
main page
page 2
page 3
Main page
Main page
Made with Streamlit
localhost:8501/page_3

Session & Callback

Session State is a way to share variables between reruns, for each user session. In addition to the ability to store and persist state, Streamlit also exposes the ability to manipulate state using Callbacks.

```
# Initialization, it’s just a dictionary
if 'key' not in st.session_state:
    st.session_state['key'] = 'value'

# Session State also supports attribute based syntax
if 'key' not in st.session_state:
    st.session_state.key = 'value'
```

```
# Updates the state


st.session_state.key = 'value2'     # Attribute API
st.session_state['key'] = 'value2'  # Dictionary like API
```

```
# Delete a single key-value pair
del st.session_state[key]

# Delete all the items in Session state
for key in st.session_state.keys():
    del st.session_state[key]
```

Callback

A callback is a python function which gets called when an input widget changes.

Order of execution: When updating Session state in response to **events**, a callback function gets executed first, and then the app is executed from top to bottom.

Callbacks can be used with widgets using the parameters on_change (or on_click), **args**, and **kwargs**:

Widgets which support the `on_click` event:

- `st.button`
- `st.download_button`
- `st.form_submit_button`

```
#without any arguments (args,kwargs)
def form_callback():
    st.write(st.session_state.my_slider)
    st.write(st.session_state.my_checkbox)

with st.form(key='my_form'):
    slider_input = st.slider('My slider', 0, 10, 5, key='my_slider')
    checkbox_input = st.checkbox('Yes or No', key='my_checkbox')
    submit_button = st.form_submit_button(label='Submit', on_click=form_callback)
```

Callback

Widgets which support the `on_change` event:

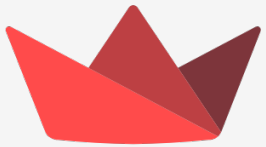
- `st.checkbox`
- `st.color_picker`
- `st.date_input`
- `st.multiselect`
- `st.number_input`
- `st.radio`
- `st.select_slider`
- `st.selectbox`
- `st.slider`
- `st.text_area`
- `st.text_input`
- `st.time_input`
- `st.file_uploader`

Widgets which support the `on_click` event:

- `st.button`
- `st.download_button`
- `st.form_submit_button`

More...

- API reference
  - Write and magic
  - Text elements
  - Data display elements
  - Chart elements
  - Input widgets
  - Media elements
  - Layouts and containers
  - Status elements
  - Control flow
  - Utilities
  - Mutate charts
  - State management
  - Performance
  - Personalization

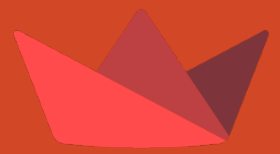


Streamlit

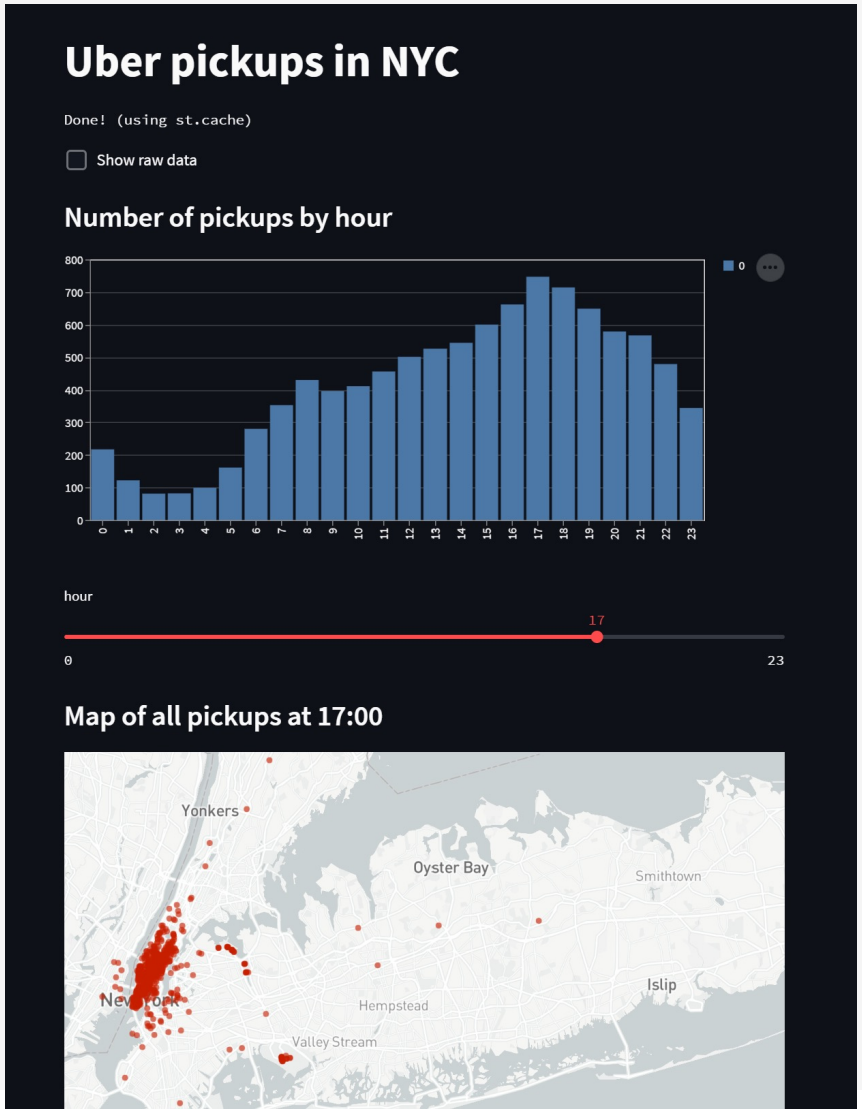
https://docs.streamlit.io/

Module 4

App Model and Development

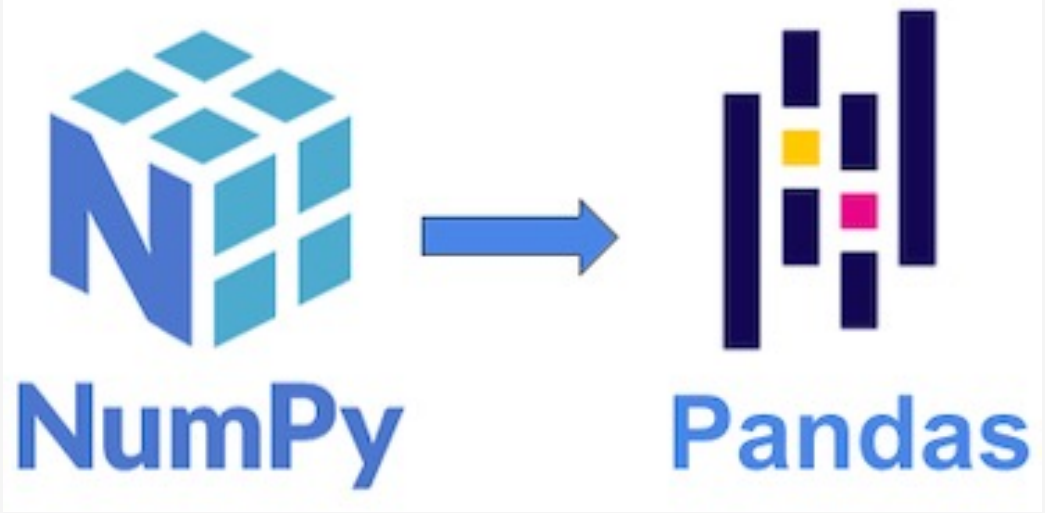


Creating An Apps



Numpy and Pandas

- We will use these libraries for the project
- Numpy is a library for processing multidimensional array
- While Pandas is a data exploration library especially for tabular data



Let’s do an exploration on uber data

Createa notebook **.ipynb** file, and add this to cells:

- Import the packages

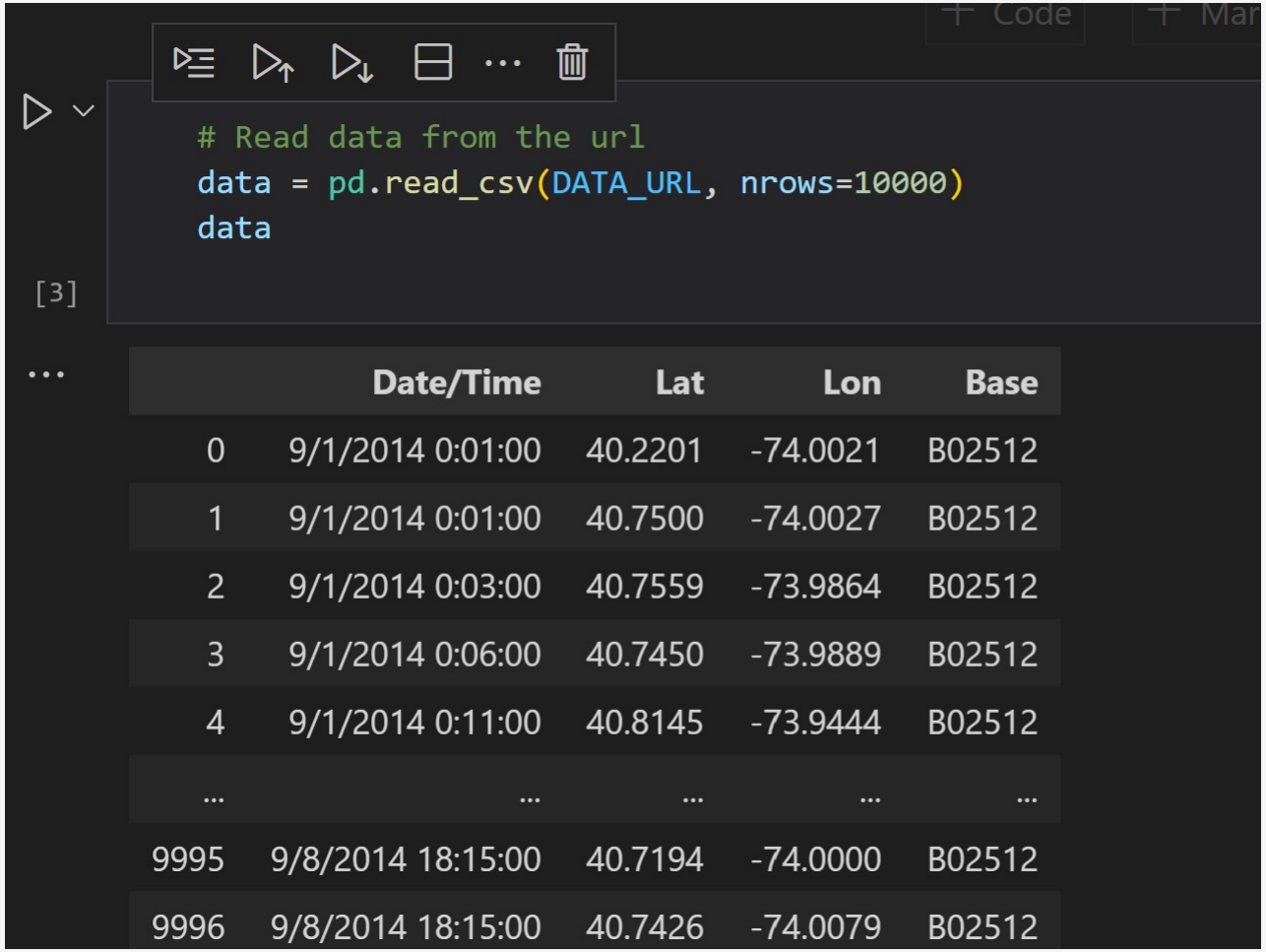
- Define an URL and date_column (we will use it later)

```
# import the packages/libraries
import pandas as pd
import numpy as np
```

```
# Define a constant
DATE_COLUMN = 'date/time'
DATA_URL = 'https://s3-us-west-2.amazonaws.com/streamlit-demo-data/uber-raw-data-sep14.csv.gz'
```

Read and shows the data

From the data there are 4 columns, but the first 3 columns will be used in our apps.



	Date/Time	Lat	Lon	Base
0	9/1/2014 0:01:00	40.2201	-74.0021	B02512
1	9/1/2014 0:01:00	40.7500	-74.0027	B02512
2	9/1/2014 0:03:00	40.7559	-73.9864	B02512
3	9/1/2014 0:06:00	40.7450	-73.9889	B02512
4	9/1/2014 0:11:00	40.8145	-73.9444	B02512
...	...	...	...	...
9995	9/8/2014 18:15:00	40.7194	-74.0000	B02512
9996	9/8/2014 18:15:00	40.7426	-74.0079	B02512

Transform the columns

Rename the columns to lower case and convert the date/time to datetime format, we will use the date/time later as the slicer

```
# Transform the column names to the lower case
lowercase = lambda x: str(x).lower()
data.rename(lowercase, axis='columns', inplace=True)

# Transform the date column to datetime format
data[DATE_COLUMN] = pd.to_datetime(data[DATE_COLUMN])
data
```

[4]

	date/time	lat	lon	base
0	2014-09-01 00:01:00	40.2201	-74.0021	B02512
1	2014-09-01 00:01:00	40.7500	-74.0027	B02512
2	2014-09-01 00:03:00	40.7559	-73.9864	B02512
3	2014-09-01 00:06:00	40.7450	-73.9889	B02512
4	2014-09-01 00:11:00	40.8145	-73.9444	B02512
...	...	...	...	...

Create a histogram

```
np.histogram(data[DATE_COLUMN].dt.hour, bins=24, range=(0,24))
```

[6] ✓ 0.4s

```
(array([217, 122,  81,  82,  99, 161, 280, 353, 430, 396, 411, 456, 501,
        526, 544, 600, 662, 747, 714, 649, 579, 567, 479, 344], dtype=int64),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
        13., 14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24.]))
```

```
# create the histogram
hist_values = np.histogram(data[DATE_COLUMN].dt.hour, bins=24, range=(0,24))[0]
hist_values
```

[5] ✓ 0.4s

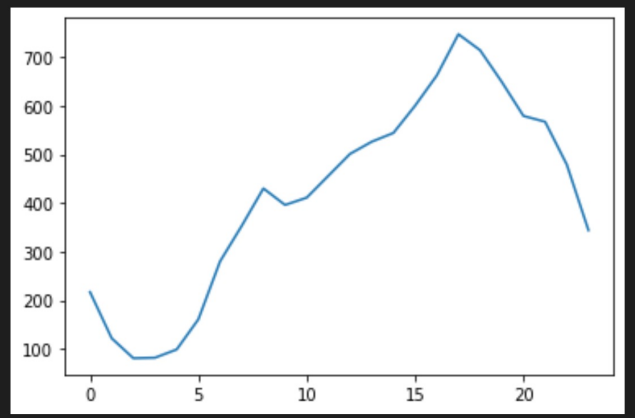
```
array([217, 122,  81,  82,  99, 161, 280, 353, 430, 396, 411, 456, 501,
       526, 544, 600, 662, 747, 714, 649, 579, 567, 479, 344], dtype=int64)
```

Let’s plot the histogram

```
# let's plot it
import matplotlib.pyplot as plt

plt.plot(hist_values)
```

[6]

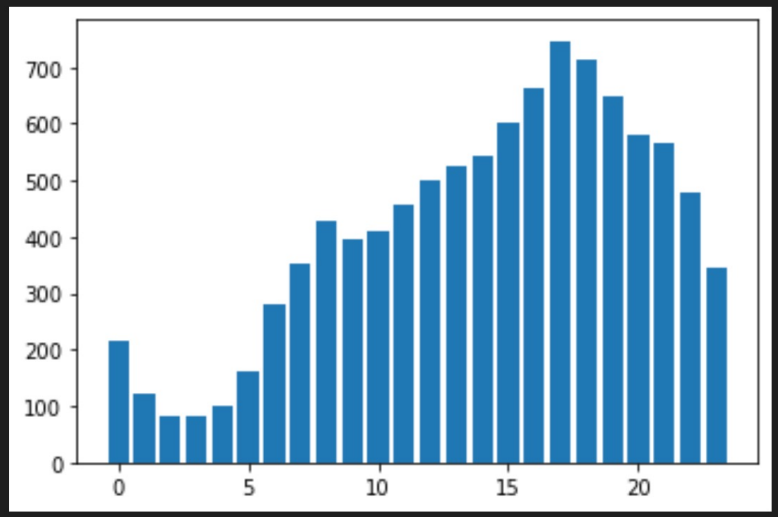
```
[<matplotlib.lines.Line2D at 0x1b65ef6adf0>]
```




```
# make a bar chart
plt.bar(x=[i for i in range(24)],height=hist_values)
```

[10]

```
<BarContainer object of 24 artists>
```



Create and apps

Streamlit is more than just a way to make data apps, it's also a community of creators that share their apps and ideas and help each other make their work better. Please come join us on the community forum. We love to hear your questions, ideas, and help you work through your bugs — stop by today!

The first step is to create a new Python script. Let's call it uber_pickups.py.

Basically, we just copy the script from our notebook previously.

Open uber_pickups.py in your favorite IDE or text editor, then add these lines:

```
import streamlit as st
import pandas as pd
import numpy as np
```

Set the title

- Set the title and add (copy) the constant

```
# 1 SET TITLE --> RUN
st.title('Uber pickups in NYC')


# 2 SET CONST
DATE_COLUMN = 'date/time'
DATA_URL = ('https://s3-us-west-2.amazonaws.com/'
            'streamlit-demo-data/uber-raw-data-sep14.csv.gz')
```

Create data loader function

- Use the cache decorator so that we do not require to download everytime we run the program
- The function relatively the same as we done in the notebook, i.e., load data from url, transform the columns.

```
# 2.1 Create Func. for loading the data
@st.cache
def load_data(nrows):
    data = pd.read_csv(DATA_URL, nrows=nrows)
    lowercase = lambda x: str(x).lower()
    data.rename(lowercase, axis='columns', inplace=True)
    data[DATE_COLUMN] = pd.to_datetime(data[DATE_COLUMN])
    return data
```

Show the data to the streamlit

- Use the checkbox to show and hide the data

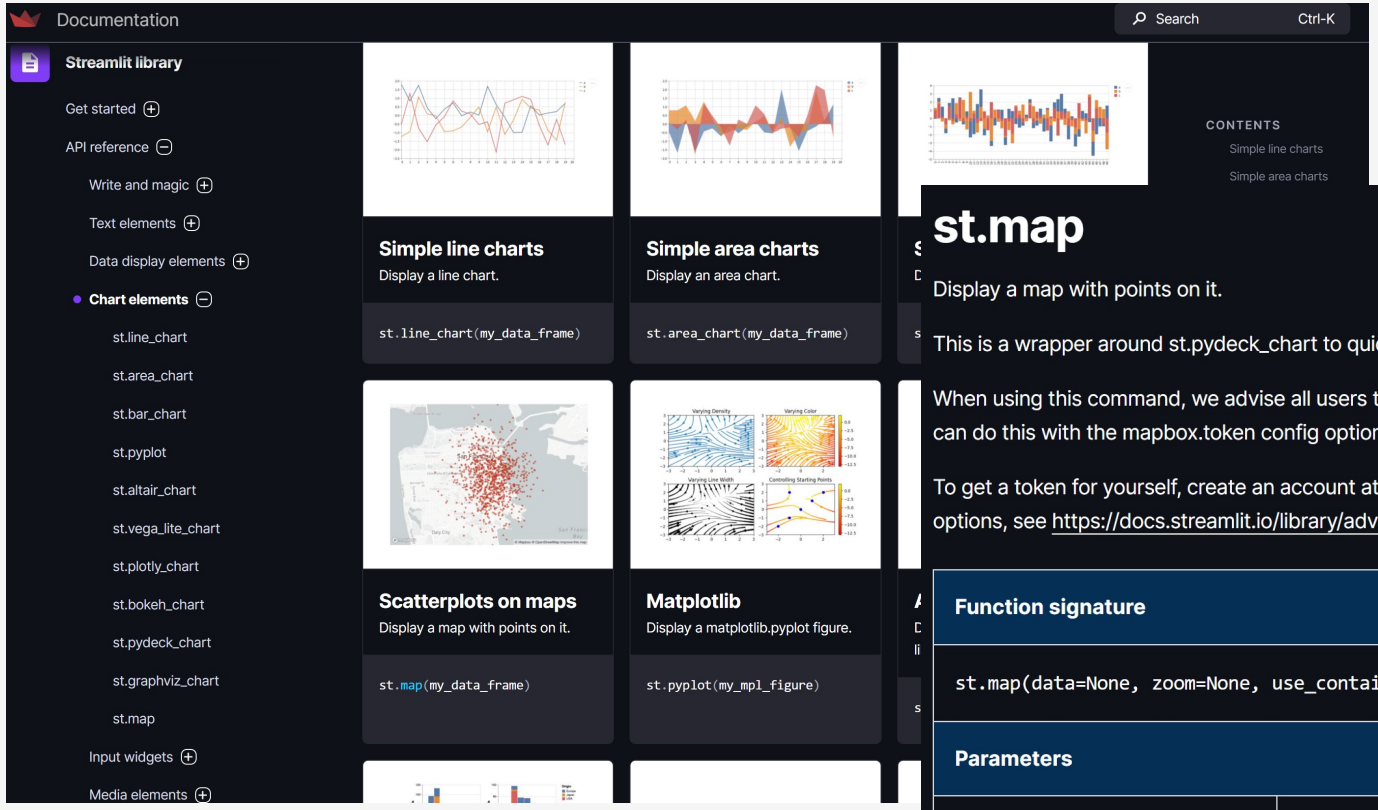
```
# 2.2 Get data --> RUN
data_load_state = st.text('Loading data...')
data = load_data(10000)
data_load_state.text("Done! (using st.cache)")


# 3 Show and Hide DataFrame --> RUN
if st.checkbox('Show raw data'):
    st.subheader('Raw data')
    st.write(data)
```

Add the histogram

- The histogram is used to show the distribution of the number of pickups by hour we can use the bar_chart function in streamlit to plot the chart.

```
# 4 Adding Histogram --> RUN
st.subheader('Number of pickups by hour')
hist_values = np.histogram(data[DATE_COLUMN].dt.hour, bins=24, range=(0,24))[0]
st.bar_chart(hist_values)
```

Use the map API



st.map

v1.10.0

Display a map with points on it.

This is a wrapper around st.pydeck_chart to quickly create scatterplot charts on top of a map, with auto-centering and auto-zoom.

When using this command, we advise all users to use a personal Mapbox token. This ensures the map tiles used in this chart are more robust. You can do this with the mapbox.token config option.

To get a token for yourself, create an account at https://mapbox.com. It's free! (for moderate usage levels). For more info on how to set config options, see https://docs.streamlit.io/library/advanced-features/configuration#set-configuration-options

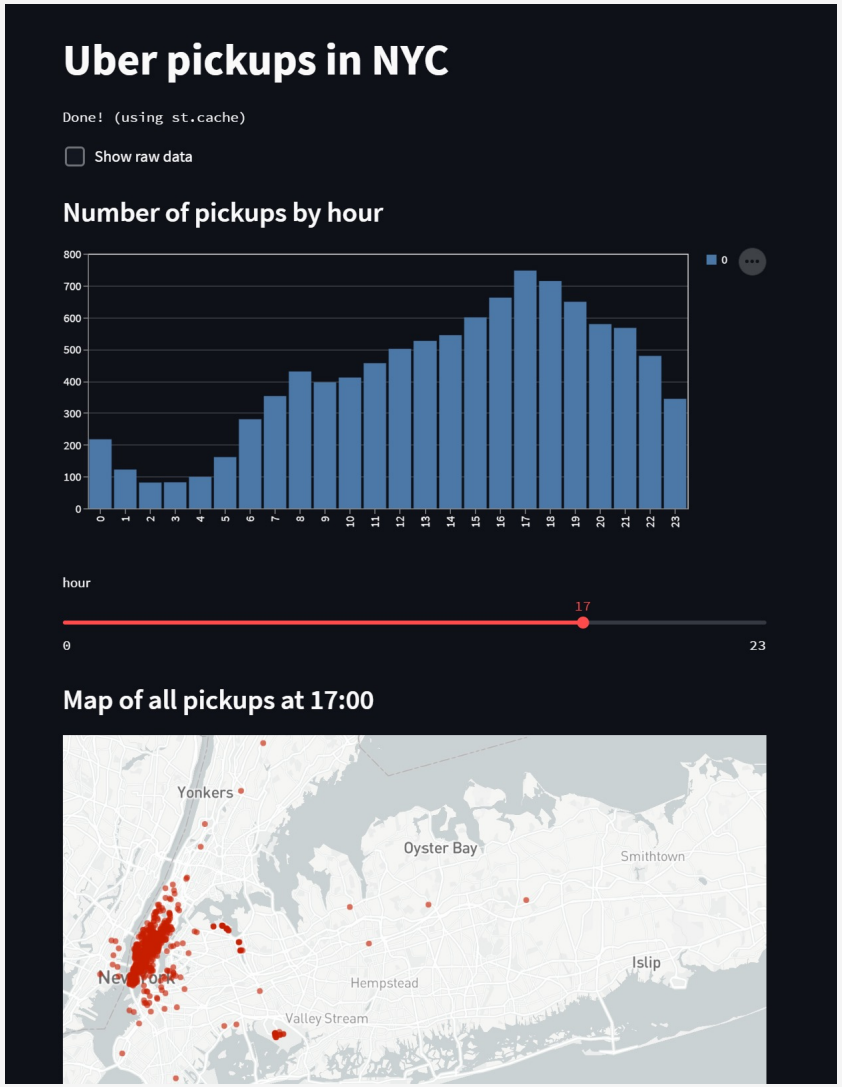
Function signature	
`st.map(data=None, zoom=None, use_container_width=True)`	
Parameters	
data *(pandas.DataFrame, pandas.Styler, numpy.ndarray, Iterable, dict,)*	or None The data to be plotted. Must have columns called 'lat', 'lon', 'latitude', or 'longitude'.
zoom *(int)*	Zoom level as specified in https://wiki.openstreetmap.org/wiki/Zoom_levels

Adding the maps

- As our data has lat and lon columns then we can use directly the st.map function.
- We would like to filter the data hourly using slider.
- The filtered data is shown in the map.

```
#5.1 Map filters based on the hour filter
# Some number in the range 0-23
hour_to_filter = st.slider('hour', 0, 23, 17)
filtered_data = data[data[DATE_COLUMN].dt.hour ==
hour_to_filter]

#5.2 Show the map --> RUN
st.subheader('Map of all pickups at %s:00' % hour_to_filter)
st.map(filtered_data)
```

Final apps


Uber pickups in NYC
Done! (using st.cache)
Show raw data
Number of pickups by hour
800
700
600
500
400
300
200
100
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
hour
17
0
23
Map of all pickups at 17:00
Yonkers
Oyster Bay
Smithtown
Islip
Hempstead
Valley Stream

Thank you