Presentation: Communicating data and code

Using the notebook format and app frameworks to communicate code and data

Thomas H. Simm

Content

- Notebooks
  - What are they?
  - Examples
  - Pros and cons
- Streamlit
  - Use in communicating code
- Websites and HTML
  - Converting notebooks to HTML and websites
- Presentations
  - Using notebooks for presentations
- Tabular Data
  - Comments on Excel
  - Thoughts on code alternatives

Overview Communicating code

- Communicating when code is a large element of what is being presented
  - Microsoft Word/ppt- type methods aren't set-up well to include code
  - Programming files (e.g. `.py`) aren't set-up well to share or understand what is going on in the code
  - Videoing code with outputs is an option, but doesn't translate to other formats (i.e. we may also need to do a written format of this)

  - Excel is a good way to present tabular data **but** if we are only viewing is slow and inefficient

- Better ways?
  - Programming notebooks (e.g. `.ipynb`) offer a good and easy to share code alongside written content
    - And can be converted to other formats easily such as html
  - Apps (e.g. streamlit) can be good
    - particularly when interactivity is required

Notebooks General

Jupyter Notebooks

From TalkPython: Awesome Jupyter Libraries and Extensions

> Jupyter is an amazing environment for exploring data and generating executable reports with Python. But there are many external tools, extensions, and libraries to make it so much better and make you more productive.

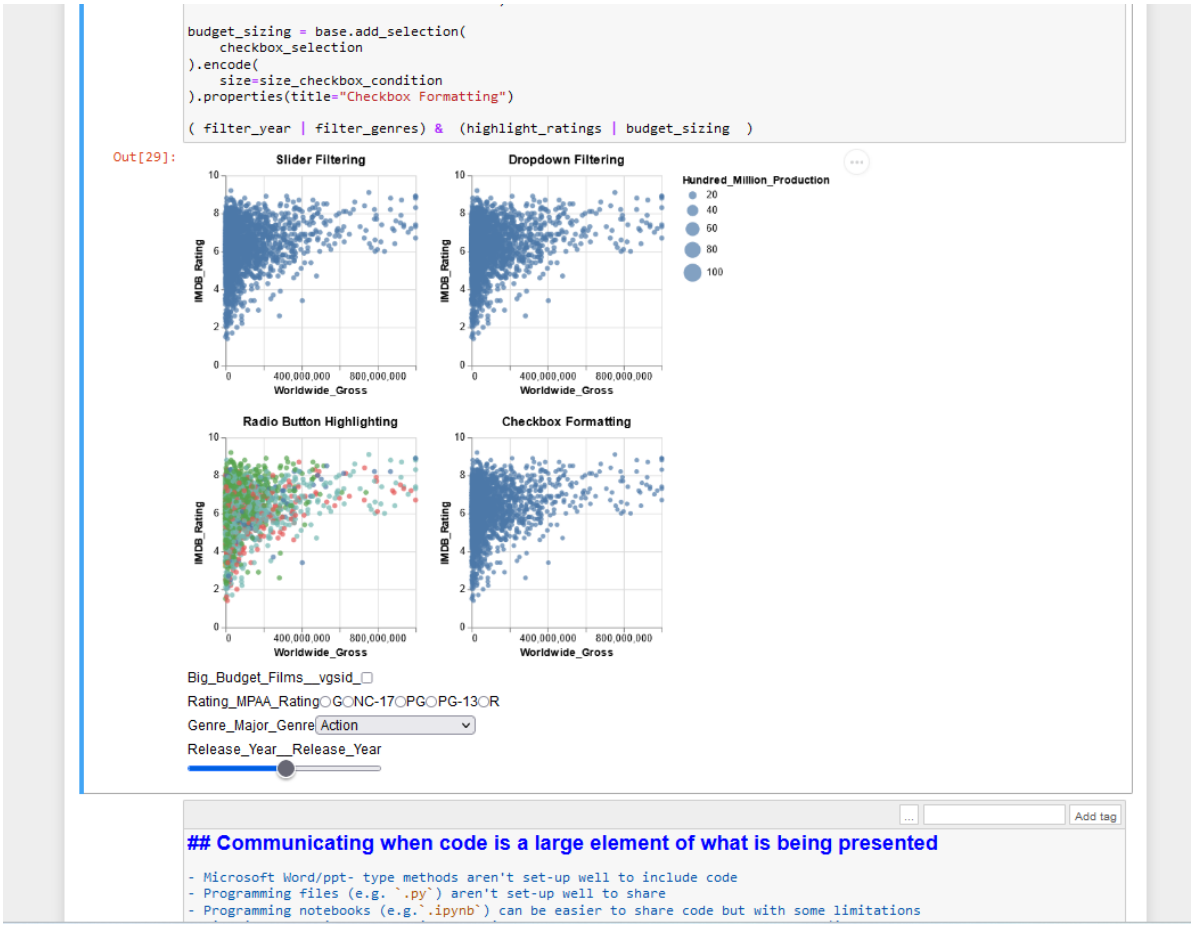
- A notebook consists of two parts
  - markdown part where we can:
    - write text, add images, links, html, LaTeX etc
  - code part which runs and displays output of code

Some links:

- Jupyter Book
- A curated list of awesome Jupyter projects
- Code Documentation QA of Code
- FastAI guide for better blogs

Example of a notebook

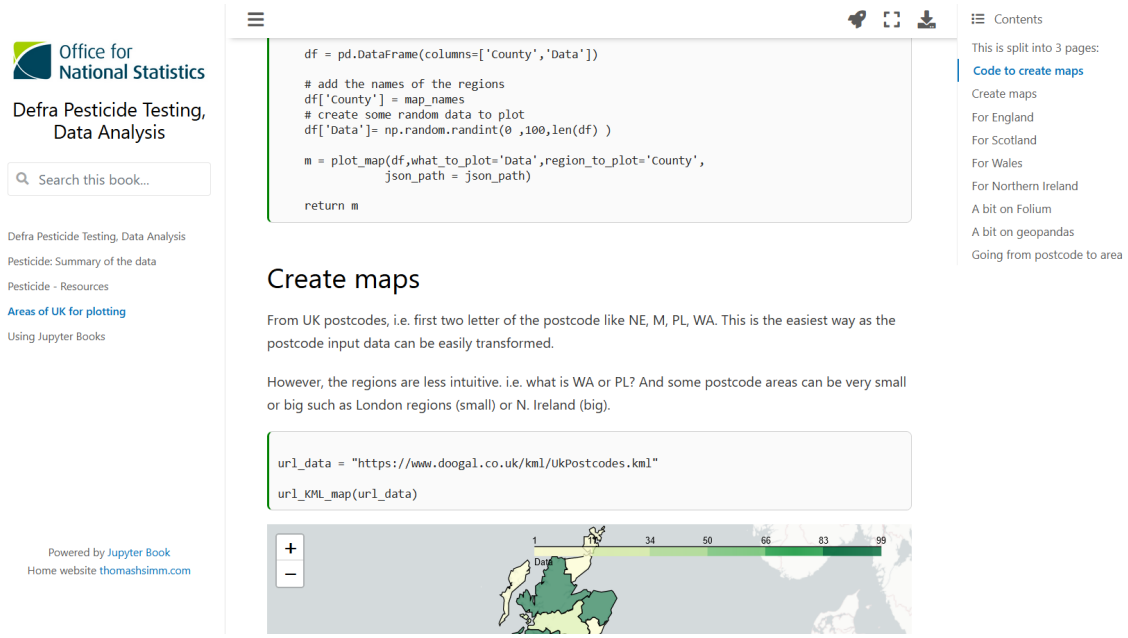
An example notebook



Markdown in a notebook 1

Some useful commands:

1. `# Notebooks Markdown and Code` and `## Markdown in a notebook`
2. `![](ghtop_images/pest.png)` looks like this



Markdown in a notebook 2

3. And the same with a mp4 file `![](ghtop_images/revealjs.mp4)`

Markdown in a notebook 3

4. `> If we want text like this`

   If we want text like this

5. Or if we want code use ‘a = b + c‘

or:

“‘

a = b

a = a + c

“‘

```
a = b

a = a + c
```

Markdown in a notebook 4

6. HTML works too

<img src="ghtop_images/pest.png"></img>

Code in a notebook

Example interactive format using altair:

```
import altair as alt
from vega_datasets import data

movies = alt.UrlData(
    data.movies.url,
    format=alt.DataFormat(parse={"Release_Date":"date"})
)
ratings = ['G', 'NC-17', 'PG', 'PG-13', 'R']
genres = ['Action', 'Adventure', 'Black Comedy', 'Comedy',
       'Concert/Performance', 'Documentary', 'Drama', 'Horror', 'Musical',
       'Romantic Comedy', 'Thriller/Suspense', 'Western']

base = alt.Chart(movies, width=200, height=200).mark_point(filled=True).transform_calculat
    Rounded_IMDB_Rating = "floor(datum.IMDB_Rating)",
    Hundred_Million_Production =  "datum.Production_Budget > 100000000.0 ? 100 : 10",
    Release_Year = "year(datum.Release_Date)"
).transform_filter(
    alt.datum.IMDB_Rating > 0
).transform_filter(
    alt.FieldOneOfPredicate(field='MPAA_Rating', oneOf=ratings)
).encode(
    x=alt.X('Worldwide_Gross:Q', scale=alt.Scale(domain=(100000,10**9), clamp=True)),
    y='IMDB_Rating:Q',
    tooltip="Title:N"
)

# A slider filter
year_slider = alt.binding_range(min=1969, max=2018, step=1)
slider_selection = alt.selection_single(bind=year_slider, fields=['Release_Year'], name="R
```

```
filter_year = base.add_selection(
    slider_selection
).transform_filter(
    slider_selection
).properties(title="Slider Filtering")

# A dropdown filter
genre_dropdown = alt.binding_select(options=genres)
genre_select = alt.selection_single(fields=['Major_Genre'], bind=genre_dropdown, name="Gen

filter_genres = base.add_selection(
    genre_select
).transform_filter(
    genre_select
).properties(title="Dropdown Filtering")

#color changing marks
rating_radio = alt.binding_radio(options=ratings)

rating_select = alt.selection_single(fields=['MPAA_Rating'], bind=rating_radio, name="Rati
rating_color_condition = alt.condition(rating_select,
                      alt.Color('MPAA_Rating:N', legend=None),
                      alt.value('lightgray'))

highlight_ratings = base.add_selection(
    rating_select
).encode(
    color=rating_color_condition
).properties(title="Radio Button Highlighting")

# Boolean selection for format changes
input_checkbox = alt.binding_checkbox()
checkbox_selection = alt.selection_single(bind=input_checkbox, name="Big Budget Films")

size_checkbox_condition = alt.condition(checkbox_selection,
                                        alt.SizeValue(25),
                                        alt.Size('Hundred_Million_Production:Q')
                                       )

budget_sizing = base.add_selection(
    checkbox_selection
```

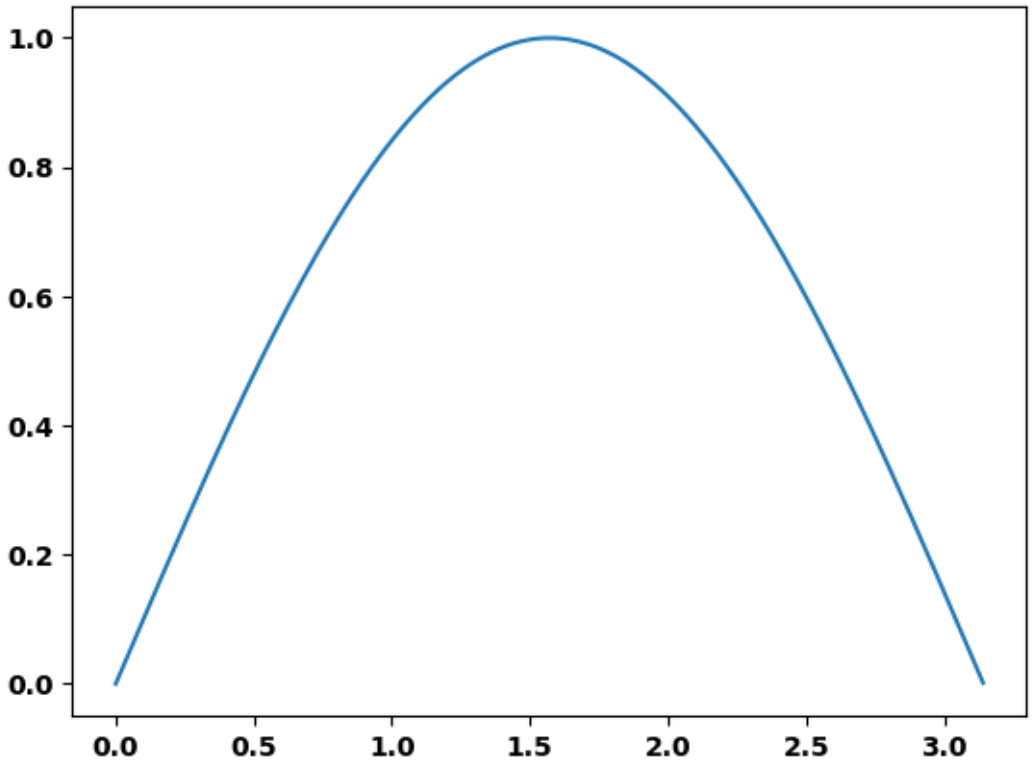
```
  ).encode(
      size=size_checkbox_condition
  ).properties(title="Checkbox Formatting")

  ( filter_year | filter_genres) &  (highlight_ratings | budget_sizing  )
```

```
alt.VConcatChart(...)
```

Simpler code output

```
x = np.arange(0,np.pi,.01)
y = np.sin(x)
plt.plot(x,y)
```

```
  import matplotlib.pyplot as plt
  import numpy as np
  x = np.arange(0,np.pi,.01)
  y = np.sin(x)
  plt.plot(x,y)
```



But Not everyone loves notebooks :(

Notebooks have their *validish* detractors I don't like notebooks.- Joel Grus Youtube

https://www.youtube.com/embed/7jiPeIFXb6U

Notebooks Opinion

Although notebooks have their *validish* detractors I don't like notebooks.- Joel Grus Youtube I think if you approach them in the right way they are a super powerful tool.

The negatives seem to be:

- encourage bad practice in code (a genuine problem)
- issues around order of what cell is run (easily got around with good practice)
- issues around lack of auto complete (I don't see the issue, use in visual studio autocomplete is there)
- no grammar/spelling correction
- issues with using git and version control

  - there are ways around this though

Notebook Benefits

- Notebooks are **intuitive**
  - You have the code then the result of the code
  - Plus can add details of how code works
  - And it's linear
- Can get things up and working **quickly**
- Aid with communicating code
- Encourages **Writing**
  - and writing things down aids thinking in the now and understanding what you did and why in the future
  - FastAI guide for better blogs
- Can use shell commands e.g. `!pip install pandas`
- Can use magic commands e.g. `%%time` to time a cell
- Easy to convert code to a pipeline

With many companies moving towards Python/R from Excel and a varied level of skills.

- The first of these is particularly important to aid communicating code

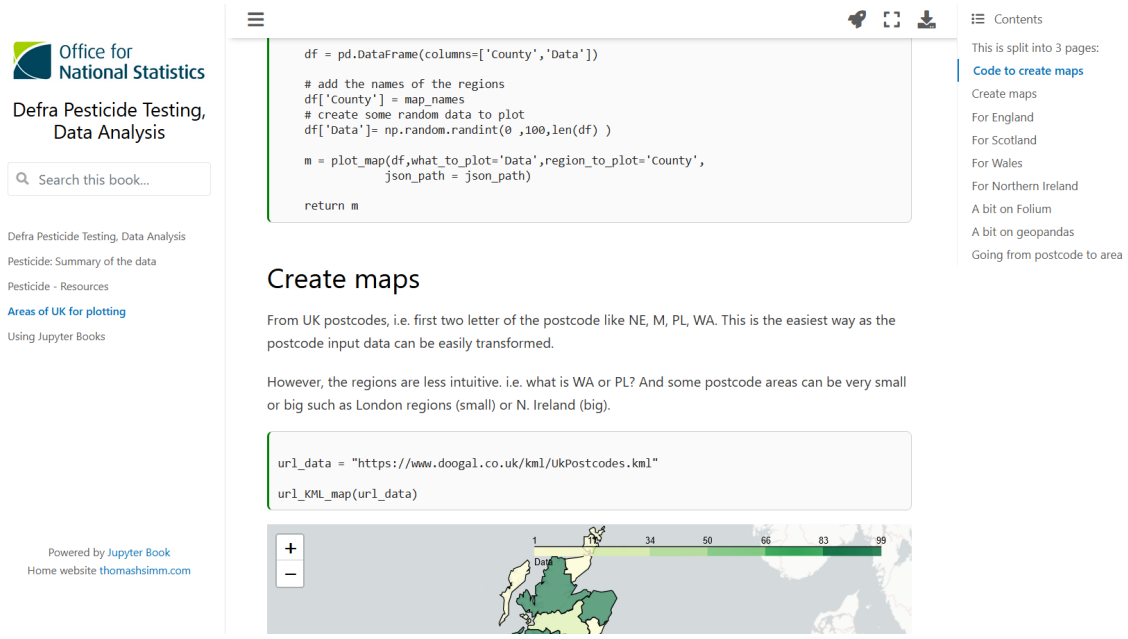
Example Useage

Example: Documenting Code

- Here is my website for my research project on pesticides in UK food.
- This is not the same as documentation for a package but there are parallels

This does a few things:

- Documents the analysis steps I have taken including the code and outputs
  - Useful for data transparency, useability of the code if needs modifiying/adapting, and why I did XYZ
- Provides a way to present the data
  - There is a streamlit app, but sometimes I like to be able to see the code



Example: Tool to aid learning

A big area I have been using Jupyter Notebooks for is to aid learning

- If you want to understand something it helps to write it down
- Having the code next to it is a big advantage
- And if stored on github you can access it anywhere

Tensoflow cheat sheet

Example: Debugging Code

- Since starting at ONS I have been working with understanding an existing project and latterly adding code to it

- The project consists of multiple python files across several folders
  - My Python was good but lots of the functions and their useage weren't immediately obvious to me
- **break-points** in VS Studio is really good to step through the code and work out what happens in the code.
  - I had not used before with Python (but had lots with MATLAB), and it's really useful
- But it can be limited what you can do
  - difficult to probe code if want to write more than 1 line of code
  - the experience/knowledge exists as you go through it but no documentation to refer to later, e.g. function X does this when I give it Y etc

Example: Debugging Code 2

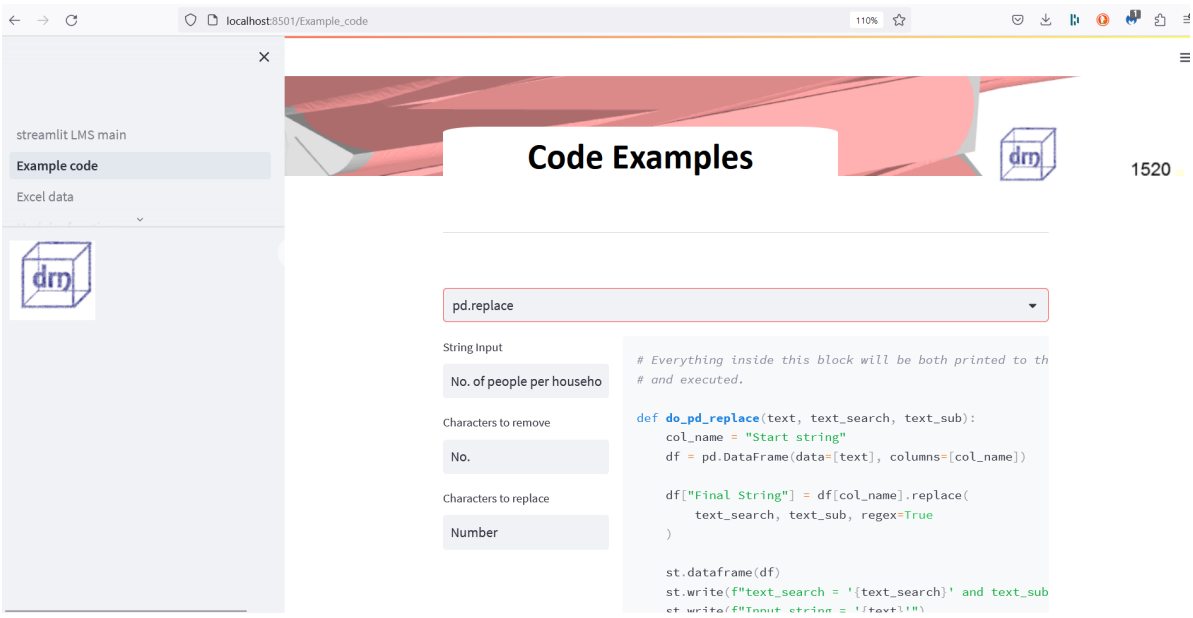
- By copying and pasting code into Jupyter cells I could see and document how they worked (e.g. changing inputs)
  - This (copying and pasting) would get around code changes too (which would be an issue if modules were just imported)
  - because this was all done in Jupyter notebook I can have a ipynb code file and a html file showing how the code works
  - I could even save a pickle file of the variables at a particularly point to understand how the code would work from this point

Streamlit

Streamlit Overview

> Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps. So let's get started!

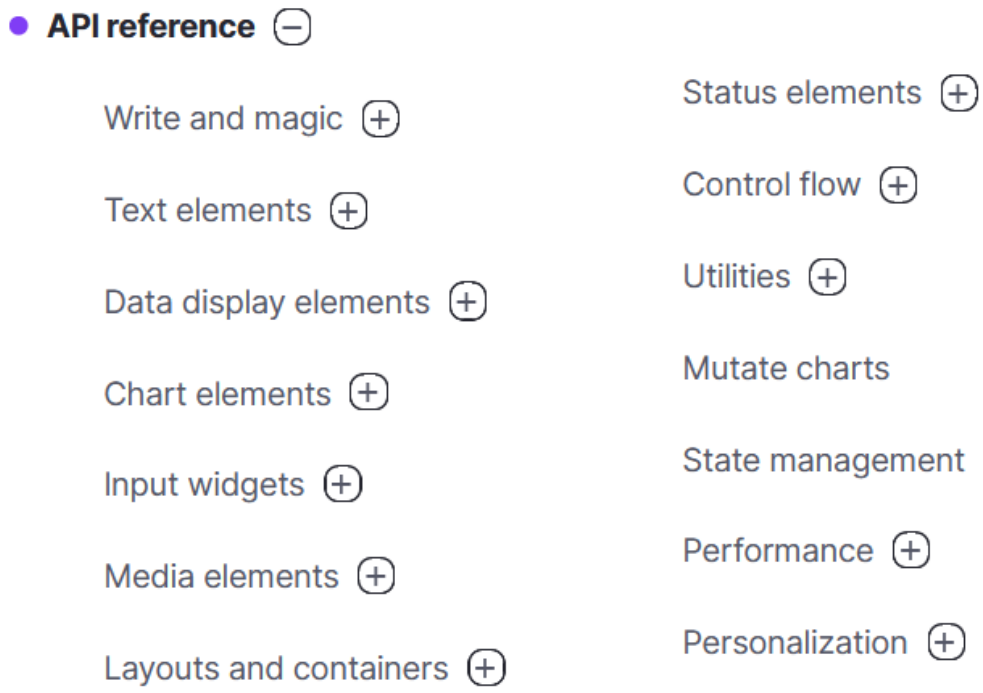
Principally used to create apps, but some of the functionality works well for code/data presentations



Streamlit Functionality: overview

Streamlit allows various functionality:

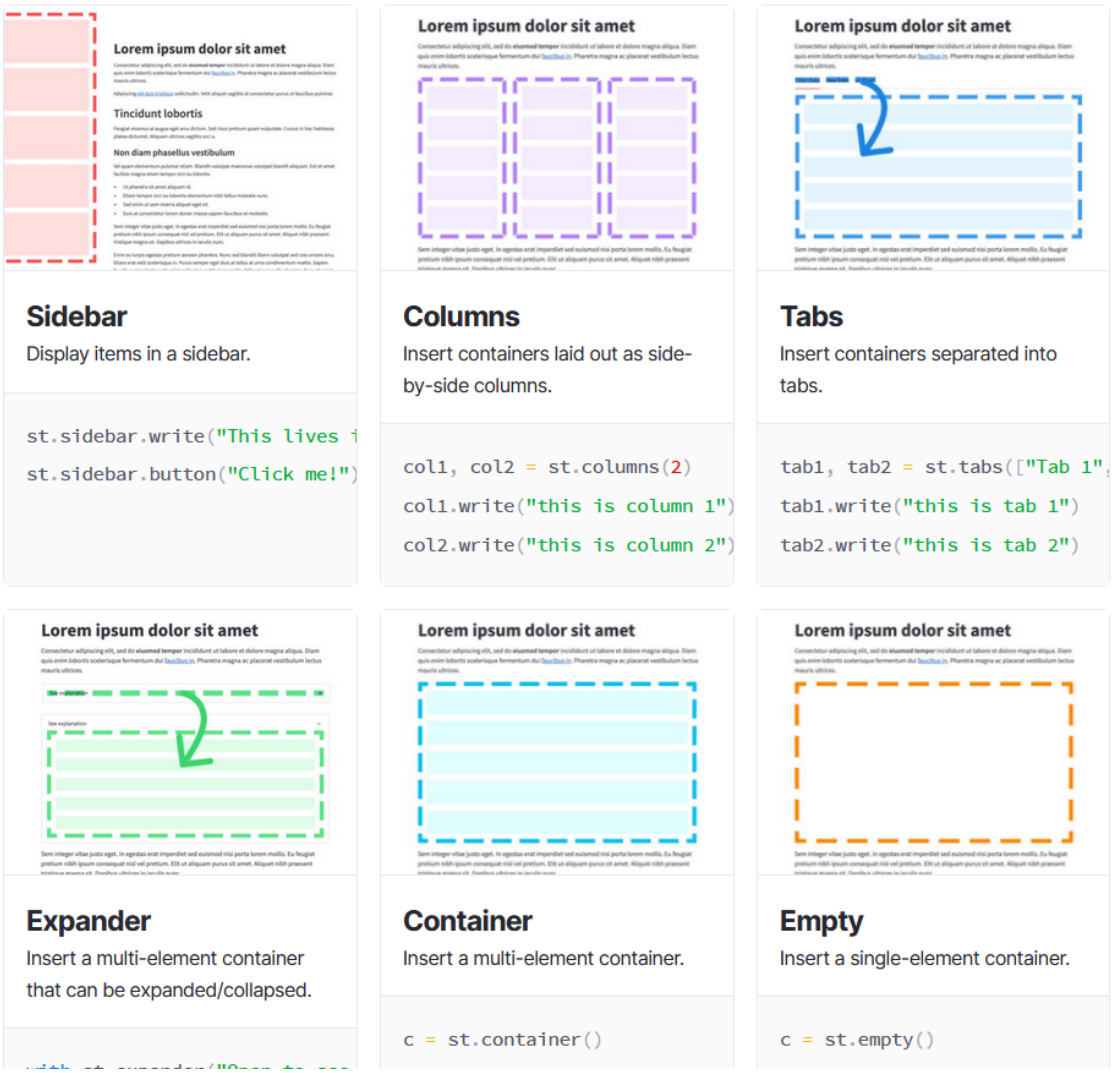
- textbox
- images/videos
- charts/tables
- menus/buttons
- etc



Streamlit Functionality: streamlit_layout

But unlike some apps (am thinking MATLAB GUIs) you can't create the look and functionality separately. So if you want something in a certain position it can be tricky. HTML can be used with `st.markdown` to give more control but it isn't recommended to use by streamlit.

Instead, to create the layout as you would like they have the following features:



Streamlit Functionality: columns and sidebar

The most useable are the first two: columns and sidebar

Columns allows us to split the app vertically. The code is fairly simple:

Either `colL, colM, colR = st.columns(3)` for 3 equal columns or to split columns with different sizes:

```
colL, _, colR = st.columns((10, 5, 20))
with colL:
    st.write('On the left')
with colR:
    st.write('On the right twice as big as left')
```

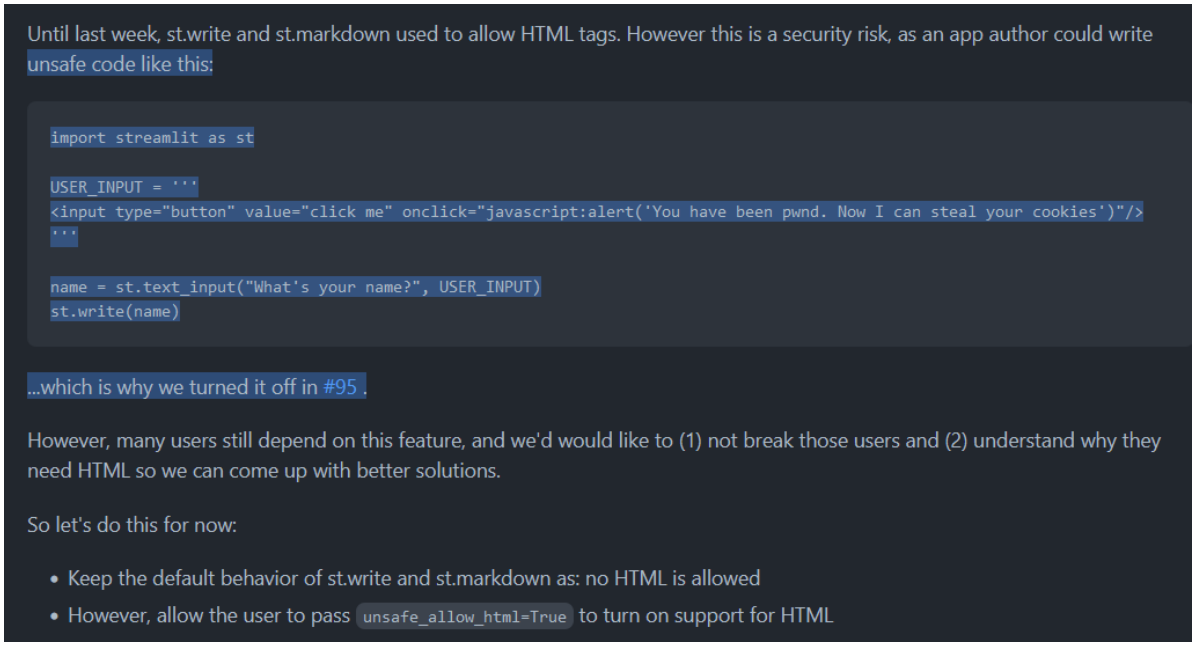
`st.sidebar` just adds a sidebar to the app that can be hidden or shown.

Anything in the sidebar is just prefixed by `st.sidebar` so:

```
st.sidebar.write('I am in the sidebar')
st.write('I am in the main app')
st.sidebar.write('I am back in the sidebar')
```

Streamlit Functionality: html

It is possible to add various additional personalisations using html. BUT it does come with security risks and so is [not recommended]](https://github.com/streamlit/streamlit/issues/152)



But it does allow much more control over the layout of the app that can be useful for a presentation: - Can add a background image - Can add background color to a textbox - Control over positioning of widgets - lots more

HTML is implementated using `st.markdown` with `unsafe_allow_html=True` inside the former

Streamlit Functionality: html examples

add background to a text box

```
text = "Code Examples"
        st.markdown(f'<center><p style=font-family:"Calibri";background-color:#FFFFFF;color:#
```

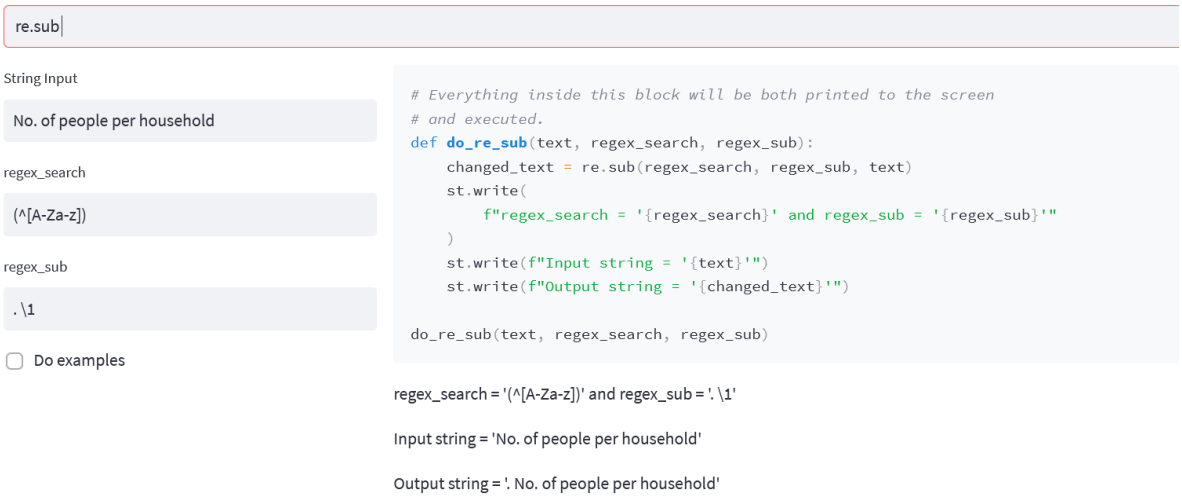
Or to add a background image

```
import streamlit as st
import base64

@st.cache(allow_output_mutation=True)
def get_base64_of_bin_file(bin_file):
    with open(bin_file, 'rb') as f:
        data = f.read()
    return base64.b64encode(data).decode()

def set_png_as_page_bg(png_file):
    bin_str = get_base64_of_bin_file(png_file)
    page_bg_img = '''
    <style>
    .stApp {
    background-image: url("%s");
    background-size: contain;
    background-repeat: no-repeat;
    background-attachment: scroll; # doesn't work
    }
    </style>
    ''' % bin_str
    st.markdown(page_bg_img, unsafe_allow_html=True)
    return
```

Streamlit Functionality: echo

> Sometimes you want your Streamlit app to contain both your usual Streamlit graphic elements and the code that generated those elements. That's where st.echo() comes in

Easier to display this by an example:



In the example above the right of the image is given below (st.columns is used, where the input for the function is found from the left column).

- `st.echo` is used with the `with` statement.
- everything within the `with` is printed to the screen and executed

```
with st.echo():
    # Everything inside this block will be both printed to the screen
    # and executed.

    def do_pd_replace(text, text_search, text_sub):
        col_name = "Start string"
        df = pd.DataFrame(data=[text], columns=[col_name])

        df["Final String"] = df[col_name].replace(
            text_search, text_sub, regex=True
        )

        st.dataframe(df)
        st.write(f"text_search = '{text_search}' and text_sub = '{text_sub}'")
        st.write(f"Input string = '{text}'")
        st.write(f"Output string = '{df['Final String'].values[0]}'")

    do_pd_replace(text, text_search, text_sub)
```