Choosing among web app frameworks for your project

Journey through a landscape of options

Dr Caterina Constantinescu

GlobalLogic®
A Hitachi Group Company

October 7, 2022

Outline

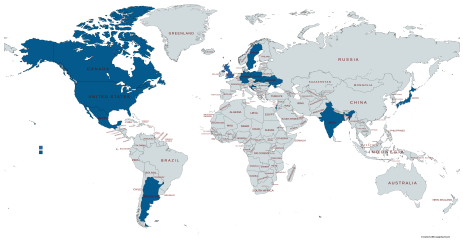
Outline

Hello!

Principal Consultant @ GlobalLogic
c.constantinescu@globallogic.com
@c__constantine
https://datapowered.io/
CaterinaC

GlobalLogic:

- Based in Silicon Valley & a Hitachi Group Company
- An AWS, MS and GC partner
- Industries: automotive, communications, financial services, healthcare, manufacturing, media and entertainment, semiconductor, and technology industries
- Yuliya Shtukaturova & Kateryna Hubaryeva selected among Top 25 Women Leaders in IT Services by Forbes.



Outline

Talk aims

GlobalLogic
A Hitachi Group Company

- Gentle introduction to basics of using various webapp tools in R & Python
- Giving you a broad-strokes intuition for what the coding experience is like across these options
- If you're familiar with just one of these tools and find yourself re-using it just for lack of time to explore further
- Supporting collaboration with multi-language teams

Outline

Project brief

- Energy sector data
- Let's imagine we have to forecast energy usage and understand how this varies along several parameters...
- Enter the National Energy Efficiency Data-Framework (NEED) dataset
- Much more information available:
  - NEED explorer
  - Consumption data tables

Project brief

GlobalLogic
A Hitachi Group Company

	A	B	C	D	E	AM	AN	AO
1	Attribute 1	Attribute 2	Elec Median 2019	Elec Mean 2019	Elec N 2019	Gas Median 2019	Gas Mean 2019	Gas N 2019
212	Pre 1919	North East	2500	3200	196020	12800	14200	165210
213	1919-44	North East	2600	3000	190190	13400	14300	178770
214	1945-64	North East	2600	3000	249500	12500	13200	233410
215	1965-82	North East	2500	3000	237720	12100	12800	208350
216	1983-92	North East	2600	3200	70830	12600	13500	59570
217	1993-99	North East	2900	3500	54280	12700	13400	46160
218	Post 1999	North East	2600	3200	137150	10500	11700	111780
219	Pre 1919	North West	2700	3500	649490	12500	14000	556930
220	1919-44	North West	3000	3500	540170	13500	14400	509420
221	1945-64	North West	2800	3400	501770	12100	13000	467320
222	1965-82	North West	2800	3300	558810	11800	12700	488420
223	1983-92	North West	2800	3500	180610	11300	12500	148690
224	1993-99	North West	3100	3700	159110	11500	12600	137100
225	Post 1999	North West	3000	3700	330840	9800	11000	232340
226	Pre 1919	Yorks & Humber	2700	3600	398310	13700	15300	347250
227	1919-44	Yorks & Humber	2800	3300	390470	13500	14500	382170
228	1945-64	Yorks & Humber	2700	3200	402230	12500	13300	380650
229	1965-82	Yorks & Humber	2600	3200	427520	12200	13200	382880
230	1983-92	Yorks & Humber	2700	3300	132390	12100	13200	113210
231	1993-99	Yorks & Humber	3000	3600	113720	12500	13600	101000
232	Post 1999	Yorks & Humber	2900	3700	274770	10600	11900	203430
233	Pre 1919	East Midlands	2900	3900	312890	13900	15200	249220
234	1919-44	East Midlands	3100	3700	264820	14000	15000	243310
235	1945-64	East Midlands	2900	3500	338200	12700	13500	305120
236	1965-82	East Midlands	2700	3400	406120	12000	12800	350560
237	1983-92	East Midlands	2900	3700	147290	11800	12700	124220
238	1993-99	East Midlands	3100	3900	116640	11900	12800	99470
239	Post 1999	East Midlands	2800	3500	282920	9900	11100	226120
240	Pre 1919	West Midlands	2900	3900	314640	14500	15900	245990
241	1919-44	West Midlands	3100	3700	407330	14200	15200	377830
242	1945-64	West Midlands	2900	3600	463540	12600	13400	416730
243	1965-82	West Midlands	2800	3500	489600	11600	12400	416430
244	1983-92	West Midlands	3000	3600	148050	12000	12800	121810
245	1993-99	West Midlands	3200	3900	111470	11600	12600	92360
246	Post 1999	West Midlands	2800	3600	273750	9600	10900	196910

We have 5 dimensions already:

- Building age
- Region
- Energy type
- Time (year)
- Metric type (mean/median)

We have 5 dimensions already:

- Building age
- Region
- Energy type
- Time (year)
- Metric type (mean/median)

So let's build something that lets us navigate these by:

- Adding some input menus to serve particular bits of data only
- Computing a forecast for those bits in particular (normally outside the app, but let's make various allowances here for demo purposes)
- Outputting the forecast (usually) as a visualisation

Outline

Shiny in R

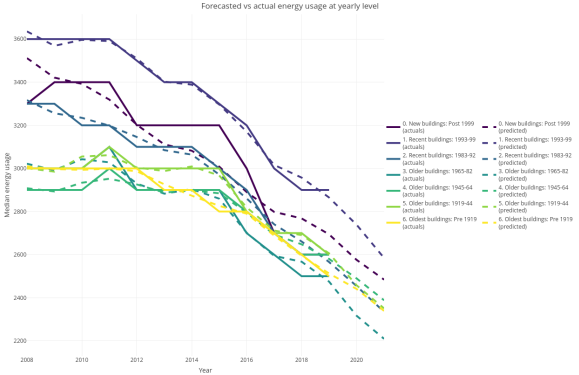
Test app

Energy type:

Electricity

Select region:

North East

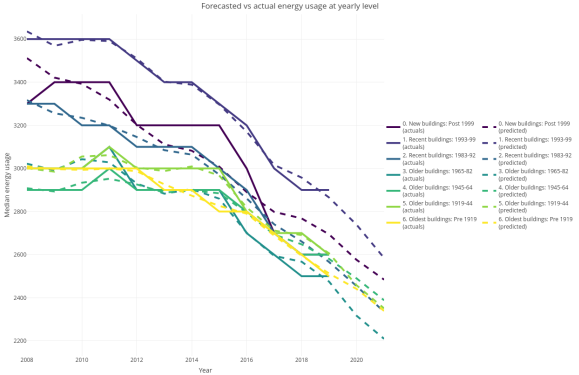


Test app

Energy type:

Electricity

Electricity
Gas



Test app

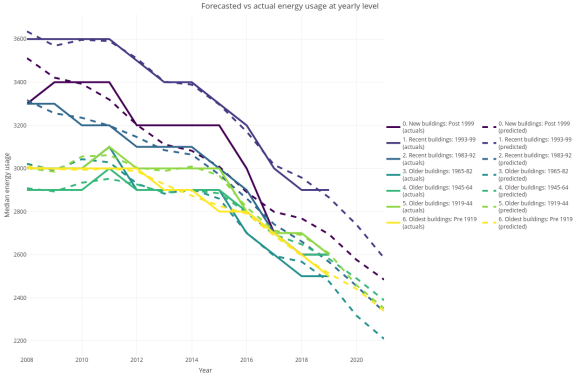
Energy type:

Electricity

Select region:

North East

North East
North West
Yorks & Humber
East Midlands
West Midlands
East of England
London
South East
South West
Wales



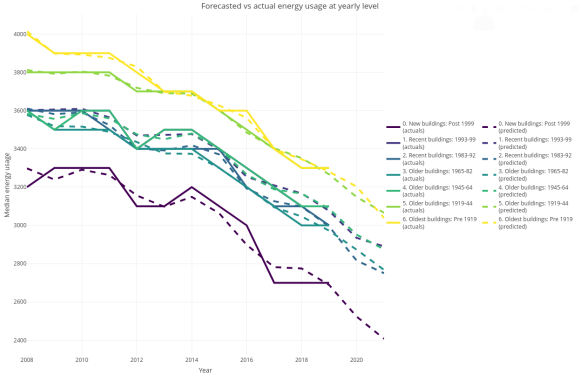
Test app

Energy type:

Electricity

Select region:

South West



- So how did we build that? Using some key libraries which exist in both R and Python:
  - Plotly
  - Prophet
- Let's take a look at some (pseudo) code
- Full code on GitHub here.

```r
# Load some libraries...dplyr, shiny, shinyWidgets, plotly, data.table, magrittr, prophet

dt <- fread("my/path/NEED_data_explorer_2021.csv",
            na.strings = "n/a")

REGIONS <- c("North East",
             "North West",
             "Yorks & Humber",
             "East Midlands",
             "West Midlands",
             "East of England",
             "London",
             "South East",
             "South West",
             "Wales")

PERIODS <- c("Pre 1919",
             "1919-44",
             "1945-64",
             "1965-82",
             "1983-92",
             "1993-99",
             "Post 1999")

GAS_DIMS <- grep("Gas Median", names(dt), value =  TRUE)
ELEC_DIMS <- grep("Elec Median", names(dt), value =  TRUE)
```

```
ui <- pageWithSidebar(
  headerPanel('Test app'),
  sidebarPanel(
    pickerInput(inputId = "energy_type",
                label = "Energy type:",
                choices = c("Electricity", "Gas"),
                multiple = FALSE,
                selected = "Electricity"),

    pickerInput(inputId = "Id008",
                label = "Select region:",
                choices = REGIONS,
                multiple = FALSE,
                selected = REGIONS[1])

  ),
  mainPanel(
    plotlyOutput('plot1', height = "800px")
  )
)
```

```
server <- function(input, output, session) {

  selectedData <- reactive({
    # Restrict subset to selected energy type and region, but leave in all building ages:
    if (input$energy_type == "Gas") {
      dt_chunk <- dt[ `Attribute 1` %in% input$Id008 & `Attribute 2` %in% PERIODS,
                      c("Attribute 1",
                        "Attribute 2",
                        GAS_DIMS),
                      with = FALSE]


    } else if (input$energy_type == "Electricity") {
      dt_chunk <- dt[ `Attribute 1` %in% input$Id008 & `Attribute 2` %in% PERIODS,
                      c("Attribute 1",
                        "Attribute 2",
                        ELEC_DIMS),
                      with = FALSE]
    }

    dt_chunk <- melt(dt_chunk, id.vars = c("Attribute 1", "Attribute 2"))
    dt_chunk[ , variable := gsub("Gas Median |Elec Median", "", variable) %>% as.numeric()]
    dt_chunk[ , value := as.numeric(value)]

    return(dt_chunk)
  })
```

```r
output$plot1 <- renderPlotly({

  s_data_copy <- selectedData()
  s_data_copy[ , date := as.POSIXct(as.Date(paste(variable, "01", "01", sep = "-")))]

  forecast_collector <- data.table()
  for (build_age in unique(s_data_copy$`Attribute 2`)) {

    s_data_copy_subset <- s_data_copy[ `Attribute 2` == build_age, ]
    history <- data.frame(ds = s_data_copy_subset$date,
                          y = s_data_copy_subset$value)
    m <- prophet(history)
    future <- make_future_dataframe(m, periods = 2, freq='year')
    forecast <- predict(m, future)
    # E.g., model for North East + Post 1999
    # Then keep accumulating more building ages for that region + energy type
    s_data_copy_subset <- merge(s_data_copy_subset,
                                forecast[ , c("ds", "yhat", "yhat_lower", "yhat_upper")],
                                by.x = "date",
                                by.y = "ds",
                                all = TRUE)

    s_data_copy_subset[ is.na(`Attribute 1`), "Attribute 1"] <- unique(s_data_copy_subset$`Attribute 1`) %>% na.exclude()
    s_data_copy_subset[ is.na(`Attribute 2`), "Attribute 2"] <- unique(s_data_copy_subset$`Attribute 2`) %>% na.exclude()

    forecast_collector <- rbind(forecast_collector, # Now add to full dataset
                                s_data_copy_subset)
  } # This now contains forecasts (+ obs data) for a region x energy type combo, and all building ages within that
```

```
forecast_collector <- melt(forecast_collector,
                           id.vars = c("date", "Attribute 1", "Attribute 2"),
                           measure.vars = c("value", "yhat"))

forecast_collector[ , `Attribute 2` := recode(`Attribute 2`,
                                              `Pre 1919` = "6. Oldest buildings: Pre 1919",
                                              `1919-44` = "5. Older buildings: 1919-44",
                                              `1945-64` = "4. Older buildings: 1945-64",
                                              `1965-82` = "3. Older buildings: 1965-82",
                                              `1983-92` = "2. Recent buildings: 1983-92",
                                              `1993-99` = "1. Recent buildings: 1993-99",
                                              `Post 1999` = "0. New buildings: Post 1999")]
forecast_collector[ , `Attribute 2` := factor(`Attribute 2`,
                                              levels = sort(unique(`Attribute 2`)),
                                              ordered = TRUE)]
forecast_collector[ , variable := recode(variable,
                                         `value` = "(actuals)",
                                         `yhat` = "(predicted)")]
```

```
    plot_ly(data = forecast_collector,
            x = ~date,
            y = ~value,
            legendgroup = ~variable,
            linetype = ~variable,
            color = ~`Attribute 2`) %>%
      add_lines(
        type = 'scatter',
        mode = "lines+markers",
        line = list(width = 4)
      ) %>%
      layout(title = 'Forecasted vs actual energy usage at yearly level',
             legend = list(orientation = "h",   # show entries horizontally
                           xanchor = "left",  # use center of legend as anchor
                           y = 0.7,
                           x = 1
             ),
             xaxis = list(title = 'Year'),
             yaxis = list(title = 'Median energy usage')
      )

  })

}

# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

GlobalLogic
A Hitachi Group Company

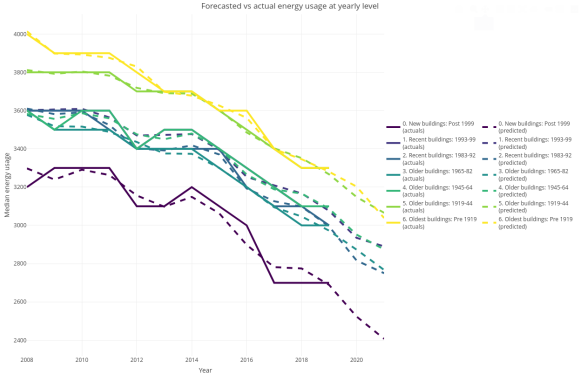
Test app

Energy type:

Electricity

Select region:

South West

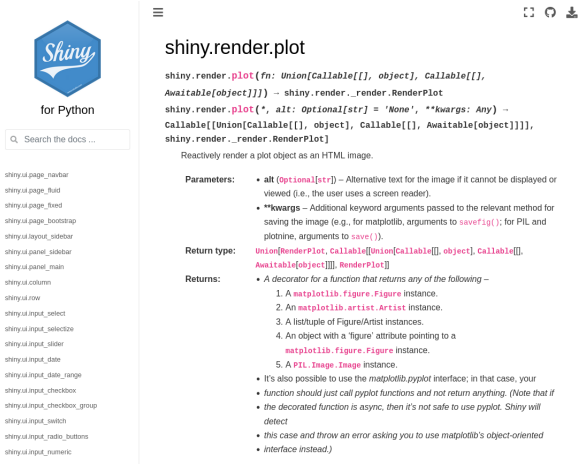


Shiny in Python

Shiny: Python

GlobalLogic
A Hitachi Group Company

```python
from shiny import App, reactive, render, ui
import pandas as pd
from pathlib import Path
from prophet import Prophet # import plotly.express as px  # !!! Not available yet

dt = pd.read_csv(Path(__file__).parent / "NEED_data_explorer_2021.csv", encoding="latin-1", na_values="n/a")
REGIONS = [...]
PERIODS = [...]
GAS_DIMS = dt.filter(regex="Gas Median").columns.values.tolist()
ELEC_DIMS = dt.filter(regex="Elec Median").columns.values.tolist()

app_ui = ui.page_fluid(
    ui.h2("Test app: Py Shiny"),
    ui.layout_sidebar(
        ui.panel_sidebar(
            ui.input_selectize("energy_type",
                               "Energy type:",
                               ['Gas', 'Electricity'],
                               multiple=False),
            ui.input_selectize("region_name",
                               "Region:",
                               REGIONS,
                               multiple=False)
        ),
        ui.panel_main(
            ui.output_table("linechart")  # !!! Can't use plotly just now
        )
    )
)
```


for Python
Search the docs ...
shiny.ui.page_navbar
shiny.ui.page_fluid
shiny.ui.page_fixed
shiny.ui.page_bootstrap
shiny.ui.layout_sidebar
shiny.ui.panel_sidebar
shiny.ui.panel_main
shiny.ui.column
shiny.ui.row
shiny.ui.input_select
shiny.ui.input_selectize
shiny.ui.input_slider
shiny.ui.input_date
shiny.ui.input_date_range
shiny.ui.input_checkbox
shiny.ui.input_checkbox_group
shiny.ui.input_switch
shiny.ui.input_radio_buttons
shiny.ui.input_numeric
shiny.render.plot
shiny.render.plot(fn: Union[Callable[[], object], Callable[[], Awaitable[object]]]) → shiny.render._render.RenderPlot
shiny.render.plot(*, alt: Optional[str] = 'None', **kwargs: Any) → Callable[[Union[Callable[[], object], Callable[[], Awaitable[object]]]], shiny.render._render.RenderPlot]
Reactively render a plot object as an HTML image.
Parameters:
alt (Optional[str]) – Alternative text for the image if it cannot be displayed or viewed (i.e., the user uses a screen reader).
**kwargs – Additional keyword arguments passed to the relevant method for saving the image (e.g., for matplotlib, arguments to savefig(); for PIL and plotnine, arguments to save()).
Return type:
Union[RenderPlot, Callable[[Union[Callable[[], object], Callable[[], Awaitable[object]]]], RenderPlot]]
Returns:
A decorator for a function that returns any of the following –
1. A matplotlib.figure.Figure instance.
2. An matplotlib.artist.Artist instance.
3. A list/tuple of Figure/Artist instances.
4. An object with a 'figure' attribute pointing to a matplotlib.figure.Figure instance.
5. A PIL.Image.Image instance.
It's also possible to use the matplotlib.pyplot interface; in that case, your
function should just call pyplot functions and not return anything. (Note that if
the decorated function is async, then it's not safe to use pyplot. Shiny will detect
this case and throw an error asking you to use matplotlib's object-oriented
interface instead.)

Shiny: Python

```python
def server(input, output, session):

    @reactive.Calc
    def selectedData():

        dt_chunk = dt.loc[(dt["Attribute 1"] == input.region_name()) & (dt["Attribute 2"].isin(PERIODS)), ]

        if input.energy_type() == "Gas":
            dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + GAS_DIMS]
        else:
            dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + ELEC_DIMS]

        dt_chunk = pd.melt(dt_chunk, id_vars=["Attribute 1", "Attribute 2"])
        dt_chunk.variable = dt_chunk.variable.str.replace("Gas Median |Elec Median ", "", regex=True)
        dt_chunk.value = pd.to_numeric(dt_chunk.value)
        dt_chunk.variable = pd.to_numeric(dt_chunk.variable)
        return dt_chunk
```

Shiny: Python

```python
@output(id="linechart")
@render.table  # could also have been: # @render.plot
def linechart():
    dt_chunk = selectedData()

    dt_chunk["ds"] = [str(year) + "-01-01" for year in dt_chunk.variable]
    dt_chunk = dt_chunk.rename(columns={"value": "y"})

    forecast_collector = []
    for build_age in dt_chunk["Attribute 2"].unique().tolist():
        history = dt_chunk[dt_chunk["Attribute 2"] == build_age][["ds", "y"]]
        history["ds"] = pd.to_datetime(history["ds"])
        m = Prophet()
        m.fit(history)

        history = history.reset_index()
        future = pd.DataFrame([history.ds.max() + pd.offsets.DateOffset(years=2)] +
                              [history.ds.max() + pd.offsets.DateOffset(years=1)] +
                              history.ds.tolist(),
                              columns=["ds"]
                              )

        forecast = m.predict(future)
        forecast["Attribute 1"] = dt_chunk["Attribute 1"].unique()[0]
        forecast["Attribute 2"] = build_age
        forecast = forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper', 'Attribute 1', 'Attribute 2']]
        forecast_collector.append(forecast)
    forecast_collector = pd.concat(forecast_collector)
```

```
        # Merge forecast with OG data:
        forecast_collector['origin'] = "Predicted"
        dt_chunk["origin"] = "Observed"
        forecast_collector = forecast_collector.rename(columns={"yhat": "y"})  # not strictly correct, but helps with concat later

        output = pd.concat([dt_chunk, forecast_collector])
        output = output[['Attribute 1', 'Attribute 2', 'ds', 'y', 'origin']]

        return output  # Output...which happens to be tabular in this case
```

```
        # # Plotly no worky yet in py-shiny:
        # fig = px.line(output,
        #               x="ds",
        #               y="y",
        #               color_discrete_sequence=["#FDE725",
        #                                        "#8FD744",
        #                                        "#35B779",
        #                                        "#21908C",
        #                                        "#31688E",
        #                                        "#443A83",
        #                                        "#440154"],
        #               color='Attribute 2',
        #               line_dash='origin',
        #               title="Forecasted vs actual energy usage at yearly level",
        #               labels={
        #                   "ds": "Year",
        #                   "y": "Median energy usage",
        #                   "Attribute 2": "Building age",
        #                   "origin": "Source"
        #               }
        #               )
        # fig.update_traces(line=dict(width=4))
        # # fig.show()
        # # return fig

app = App(app_ui, server, debug=True)
```

Run command in a Terminal:

```
1 shiny run --port 5000 shiny-python.py
```

Full code on GitHub here.

GlobalLogic
A Hitachi Group Company

Test app: Py Shiny

Energy type:

Gas

Region:

North East

Attribute 1	Attribute 2	ds	y	origin
North East	Pre 1919	2019-01-01	12800.000000	Observed
North East	1919-44	2019-01-01	13400.000000	Observed
North East	1945-64	2019-01-01	12500.000000	Observed
North East	1965-82	2019-01-01	12100.000000	Observed
North East	1983-92	2019-01-01	12600.000000	Observed
North East	1993-99	2019-01-01	12700.000000	Observed
North East	Post 1999	2019-01-01	10500.000000	Observed
North East	Pre 1919	2018-01-01	12700.000000	Observed
North East	1919-44	2018-01-01	13200.000000	Observed
North East	1945-64	2018-01-01	12400.000000	Observed
North East	1965-82	2018-01-01	11900.000000	Observed
North East	1983-92	2018-01-01	12500.000000	Observed
North East	1993-99	2018-01-01	12600.000000	Observed
North East	Post 1999	2018-01-01	10600.000000	Observed
North East	Pre 1919	2017-01-01	13000.000000	Observed
North East	1919-44	2017-01-01	13600.000000	Observed
North East	1945-64	2017-01-01	12700.000000	Observed
North East	1965-82	2017-01-01	12300.000000	Observed
North East	1983-92	2017-01-01	12800.000000	Observed
North East	1993-99	2017-01-01	13000.000000	Observed
North East	Post 1999	2017-01-01	11000.000000	Observed

GlobalLogic
A Hitachi Group Company

Test app: Py Shiny

Energy type:

Gas

Region:

Yorks & Humber

Attribute 1	Attribute 2	ds	y	origin
Yorks & Humber	Pre 1919	2019-01-01	13700.000000	Observed
Yorks & Humber	1919-44	2019-01-01	13500.000000	Observed
Yorks & Humber	1945-64	2019-01-01	12500.000000	Observed
Yorks & Humber	1965-82	2019-01-01	12200.000000	Observed
Yorks & Humber	1983-92	2019-01-01	12100.000000	Observed
Yorks & Humber	1993-99	2019-01-01	12500.000000	Observed
Yorks & Humber	Post 1999	2019-01-01	10600.000000	Observed
Yorks & Humber	Pre 1919	2018-01-01	13600.000000	Observed
Yorks & Humber	1919-44	2018-01-01	13300.000000	Observed
Yorks & Humber	1945-64	2018-01-01	12400.000000	Observed
Yorks & Humber	1965-82	2018-01-01	12100.000000	Observed
Yorks & Humber	1983-92	2018-01-01	12000.000000	Observed
Yorks & Humber	1993-99	2018-01-01	12400.000000	Observed
Yorks & Humber	Post 1999	2018-01-01	10700.000000	Observed
Yorks & Humber	Pre 1919	2017-01-01	13600.000000	Observed
Yorks & Humber	1919-44	2017-01-01	13500.000000	Observed
Yorks & Humber	1945-64	2017-01-01	12600.000000	Observed
Yorks & Humber	1965-82	2017-01-01	12300.000000	Observed
Yorks & Humber	1983-92	2017-01-01	12200.000000	Observed
Yorks & Humber	1993-99	2017-01-01	12600.000000	Observed
Yorks & Humber	Post 1999	2017-01-01	11100.000000	Observed

Streamlit

- Streamlit's architecture lets you write apps the same way you write plain Python scripts

- Streamlit's architecture lets you write apps the same way you write plain Python scripts
- `st.write()` is Streamlit's "Swiss Army knife". You can pass almost anything to `st.write()`
- You can also write your app without calling st.write()/any Streamlit methods at all: "magic commands" are supported

- Streamlit's architecture lets you write apps the same way you write plain Python scripts
- `st.write()` is Streamlit's "Swiss Army knife". You can pass almost anything to `st.write()`
- You can also write your app without calling st.write()/any Streamlit methods at all: "magic commands" are supported
- The power of `@st.cache()`

- Streamlit's architecture lets you write apps the same way you write plain Python scripts
- `st.write()` is Streamlit's "Swiss Army knife". You can pass almost anything to `st.write()`
- You can also write your app without calling st.write()/any Streamlit methods at all: "magic commands" are supported
- The power of `@st.cache()`
- Allows using phone camera pictures as direct input + other media inputs (audio/video): `st.image()`, `st.audio()`, `st.video()`

- Streamlit's architecture lets you write apps the same way you write plain Python scripts
- `st.write()` is Streamlit's "Swiss Army knife". You can pass almost anything to `st.write()`
- You can also write your app without calling st.write()/any Streamlit methods at all: "magic commands" are supported
- The power of `@st.cache()`
- Allows using phone camera pictures as direct input + other media inputs (audio/video): `st.image()`, `st.audio()`, `st.video()`
- Automatically detects changes in source code / input menus and reruns app

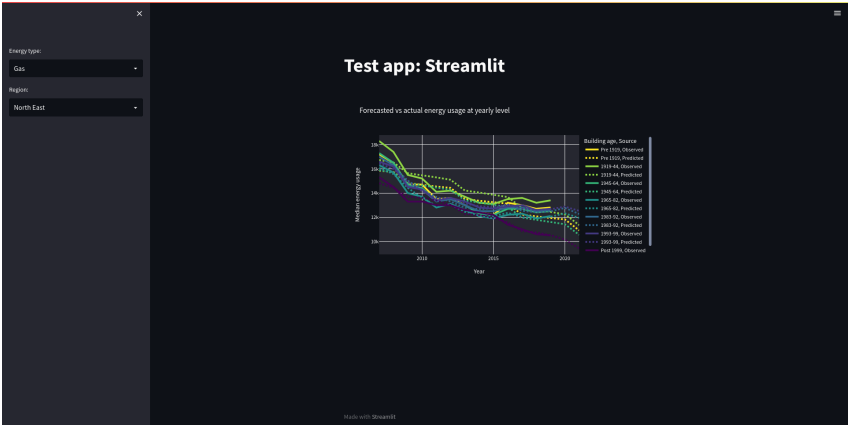
```python
import streamlit as st
import pandas as pd
from pathlib import Path
import plotly.express as px  # Works here
from prophet import Prophet

@st.cache
def get_data():
    dat = pd.read_csv(Path(__file__).parent / "NEED_data_explorer_2021.csv", encoding="latin-1", na_values="n/a")
    return dat

dt = get_data()
REGIONS = [...]
PERIODS = [...]
GAS_DIMS = ...
ELEC_DIMS = ...

st.title('Test app: Streamlit')
# Add a selectbox to the sidebar:
energy_type = st.sidebar.selectbox(
    'Energy type:',
    ('Gas', 'Electricity')
)

region_name = st.sidebar.selectbox(
    'Region:',
    REGIONS
)
```

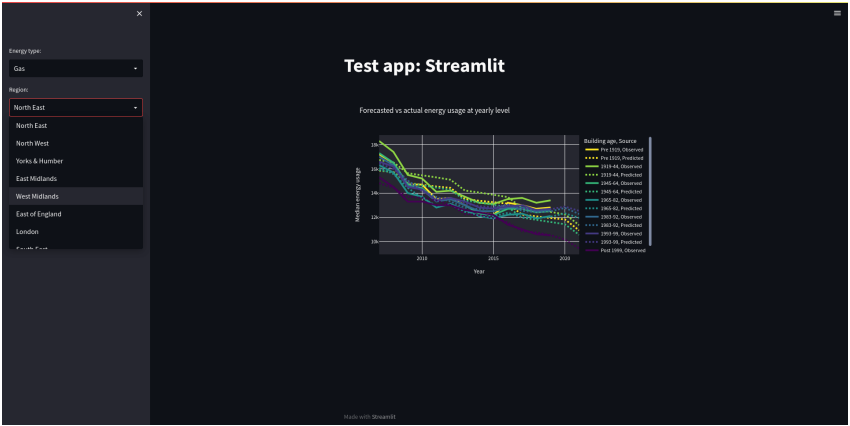
```
# Start subsetting data based on menus:
dt_chunk = dt.loc[(dt["Attribute 1"] == region_name) & (dt["Attribute 2"].isin(PERIODS)), ]

if energy_type == "Gas":
    dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + GAS_DIMS]
else:
    dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + ELEC_DIMS]

dt_chunk = pd.melt(dt_chunk, id_vars=["Attribute 1", "Attribute 2"])
dt_chunk.variable = dt_chunk.variable.str.replace("Gas Median |Elec Median ", "", regex=True)
dt_chunk.value = pd.to_numeric(dt_chunk.value)
dt_chunk.variable = pd.to_numeric(dt_chunk.variable)

# st.write(dt_chunk)

dt_chunk["ds"] = [str(year) + "-01-01" for year in dt_chunk.variable]
dt_chunk = dt_chunk.rename(columns={"value": "y"})
```

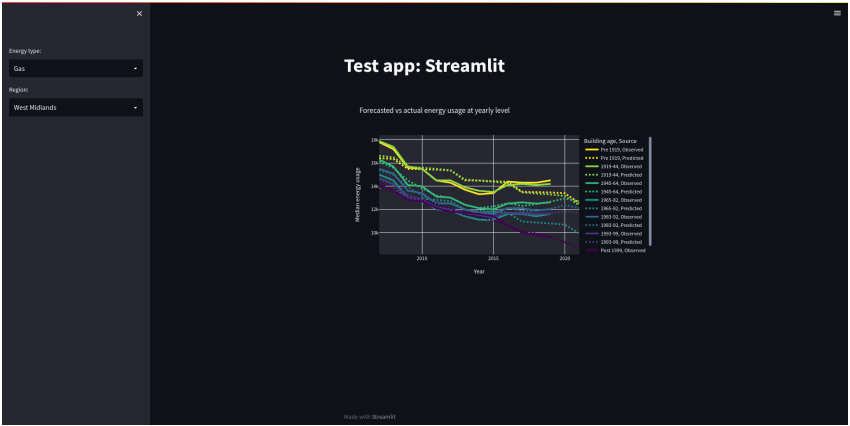
```python
@st.cache
def create_forecasts(ts):
    forecast_collector = []
    for build_age in ts["Attribute 2"].unique().tolist():
        ...
        forecast_collector.append(forecast)
    forecast_collector = pd.concat(forecast_collector)
    ...
    output = pd.concat([dt_chunk, forecast_collector])
    ...
    return output

model_output = create_forecasts(dt_chunk)
# st.write(model_output)  # Figures out for you it's a table and outputs accordingly

fig = px.line(...)
st.plotly_chart(fig, use_container_width=True)
```

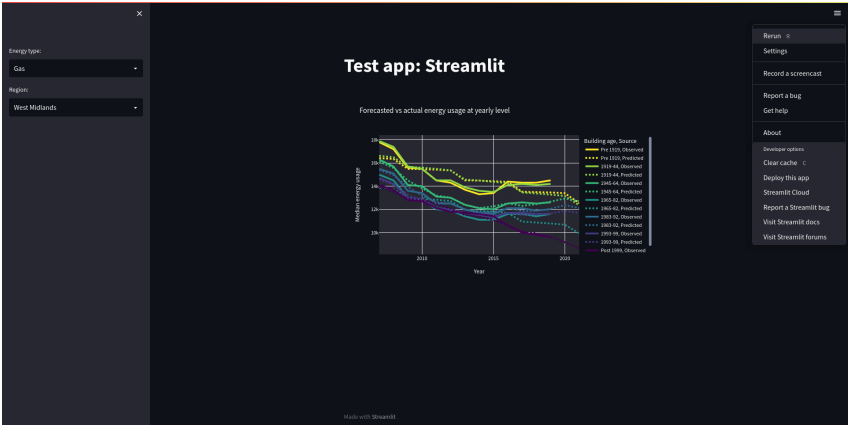
Run command in a Terminal:

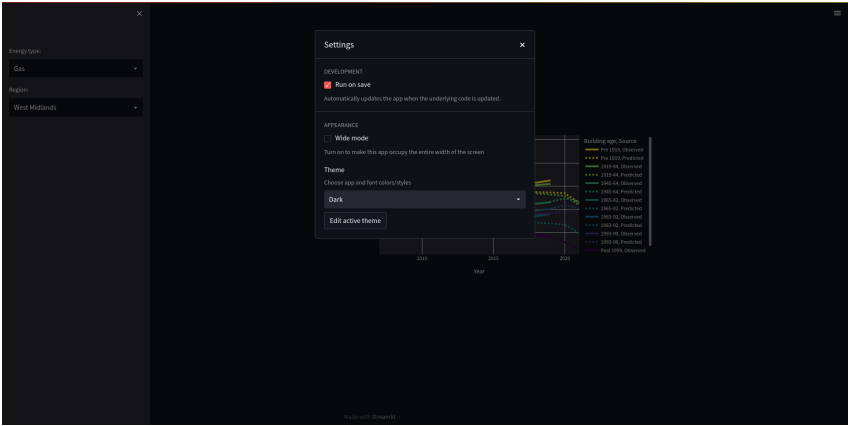
```
1 streamlit run streamlit-app.py
```


Energy type:
Gas
Region:
North East
Test app: Streamlit
Forecasted vs actual energy usage at yearly level
Median energy usage
20k
18k
16k
12k
10k
2010
2015
2020
Year
Building age, Source
Pre 1919, Observed
Pre 1919, Predicted
1919-44, Observed
1919-44, Predicted
1945-64, Observed
1945-64, Predicted
1965-82, Observed
1965-82, Predicted
1983-92, Observed
1983-92, Predicted
1993-99, Observed
1993-99, Predicted
Post 1999, Observed
Made with Streamlit


Energy type:
Gas
Region:
North East
North East
North West
Yorks & Humber
East Midlands
West Midlands
East of England
London
Test app: Streamlit
Forecasted vs actual energy usage at yearly level
Median energy usage
19k
18k
16k
12k
10k
2010
2015
2020
Year
Building age, Source
Pre 1919, Observed
Pre 1919, Predicted
1919-44, Observed
1919-44, Predicted
1945-64, Observed
1945-64, Predicted
1965-82, Observed
1965-82, Predicted
1983-92, Observed
1983-92, Predicted
1993-99, Observed
1993-99, Predicted
Post 1999, Observed
Made with Streamlit


Energy type:
Gas
Region:
West Midlands
Test app: Streamlit
Forecasted vs actual energy usage at yearly level
Median energy usage
Year
Building age, Source
Pre 1919, Observed
Pre 1919, Predicted
2019-44, Observed
2019-44, Predicted
1945-64, Observed
1945-64, Predicted
1965-82, Observed
1965-82, Predicted
1983-92, Observed
1983-92, Predicted
1993-99, Observed
1993-99, Predicted
Post 1999, Observed
Made with Streamlit


Energy type:
Gas
Region:
West Midlands
Test app: Streamlit
Forecasted vs actual energy usage at yearly level
Median energy usage
19k
18k
16k
12k
10k
2010
2015
2020
Year
Building age, Source
Pre 1919, Observed
Pre 1919, Predicted
2019-44, Observed
2019-44, Predicted
1945-64, Observed
1945-64, Predicted
1965-82, Observed
1965-82, Predicted
1983-92, Observed
1983-92, Predicted
1993-99, Observed
1993-99, Predicted
Post 1999, Observed
Rerun R
Settings
Record a screencast
Report a bug
Get help
About
Developer options
Clear cache C
Deploy this app
Streamlit Cloud
Report a Streamlit bug
Visit Streamlit docs
Visit Streamlit forums
Made with Streamlit


Energy type:
Gas
Region:
West Midlands
Settings
DEVELOPMENT
Run on save
Automatically updates the app when the underlying code is updated.
APPEARANCE
Wide mode
Turn on to make this app occupy the entire width of the screen
Theme
Choose app and font colors/styles
Dark
Edit active theme
Building age, Source
Pre 1919, Observed
Pre 1919, Predicted
2019-44, Observed
2019-44, Predicted
1945-64, Observed
1945-64, Predicted
1965-82, Observed
1965-82, Predicted
1983-92, Observed
1983-92, Predicted
1993-99, Observed
1993-99, Predicted
Post 1999, Observed
2010
2015
2020
Year
Made with Streamlit

Dash

Three key building blocks:

- Two which go inside the Dash layout:
  - Dash core components & HTML components (dropdown, date picker, Div etc)
  - Plotly graphs / Data tables
- The third: the callback (which connects the former two to create the interactive functionality)

Dash also includes 'hot-reloading', similarly to Streamlit.

```python
import dash
import dash_bootstrap_components as dbc
from dash import Input, Output, dcc, html, Dash, dash_table
import pandas as pd
from pathlib import Path
import plotly.express as px
from prophet import Prophet

dt = pd.read_csv(Path(__file__).parent / "NEED_data_explorer_2021.csv",
                 encoding="latin-1",
                 na_values="n/a")

REGIONS = [...]
PERIODS = [...]
GAS_DIMS = ...
ELEC_DIMS = ...

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

```
controls = dbc.Card(
    [
        html.Div(
            [
                dbc.Label("Energy type:"),
                dcc.Dropdown(
                    id="energy_type",
                    options=[
                        {"label": "Gas", "value": "Gas"},
                        {"label": "Electricity", "value": "Electricity"}
                    ],
                    multi=False,
                    value="Gas",
                    style={'width': "100%"}
                ),
            ]
        ),
        html.Div(
            [
                dbc.Label("Select region:"),
                dcc.Dropdown(
                    REGIONS,
                    REGIONS[0],
                    id="region_name",
                ),
            ]
        ),
    ],
)
```

```
app.layout = dbc.Container(
    [
        html.H1("Dash test app"),
        html.Hr(),
        dbc.Row(
            [
                dbc.Col(controls, md=2),
                dbc.Col(dcc.Graph(id="ts-graph"), md=6),
                dbc.Col(html.Div(id="ts-table"), md=4)
            ],
            align="center",
        ),
    ],
    fluid=True,
)
```

```python
@app.callback(
        Output("ts-graph", component_property="figure"),
        Output("ts-table", component_property="children"),
        Input("energy_type", component_property="value"),
        Input("region_name", component_property="value")
)
def make_graph(energy_type, region_name):

    dt_chunk = dt.copy()
    dt_chunk = dt_chunk.loc[(dt_chunk["Attribute 1"] == region_name) & (dt_chunk["Attribute 2"].isin(PERIODS)), ]

    if energy_type == "Gas":
        dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + GAS_DIMS]
    else:
        dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + ELEC_DIMS]

    ...  # extra data reshaping etc

    forecast_collector = []
    for build_age in dt_chunk["Attribute 2"].unique().tolist():
        history = dt_chunk[dt_chunk["Attribute 2"] == build_age][["ds", "y"]]
        history["ds"] = pd.to_datetime(history["ds"])
        m = Prophet()
        m.fit(history)
        ...
        forecast_collector.append(forecast)
    forecast_collector = pd.concat(forecast_collector)
    ...
```

```
    output = pd.concat([dt_chunk, forecast_collector])
    output = output[['Attribute 1', 'Attribute 2', 'ds', 'y', 'origin']]

    fig = px.line(...)
    fig.update_traces(line=dict(width=4))

    tabular_otpt = dash_table.DataTable(data=output.to_dict('records'),
                                        columns=[{"name": i, "id": i} for i in output.columns],
                                        page_size=10)
    return fig, tabular_otpt


if __name__ == "__main__":
    app.run_server(debug=True, port=8888)
```

Run command in a Terminal:

```
1 python dash-app.py
```

Then visit http://127.0.0.1:8888/ in your web browser.

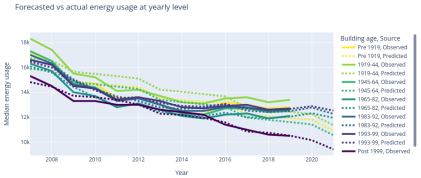
Dash test app

Energy type:

Gas

Select region:

North East



Attribute 1	Attribute 2	ds	y	origin
North East	Pre 1919	2019-01-01	12800	Observed
North East	1919-44	2019-01-01	13400	Observed
North East	1945-64	2019-01-01	12500	Observed
North East	1965-82	2019-01-01	12100	Observed
North East	1983-92	2019-01-01	12600	Observed
North East	1993-99	2019-01-01	12700	Observed
North East	Post 1999	2019-01-01	10500	Observed
North East	Pre 1919	2018-01-01	12700	Observed
North East	1919-44	2018-01-01	13200	Observed
North East	1945-64	2018-01-01	12400	Observed

« ‹ 1 / 20 › »

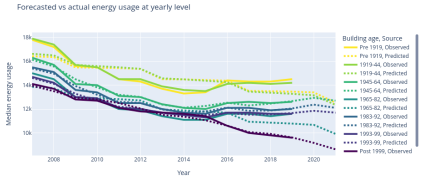
Dash test app

Energy type:

Gas

Select region:

West Midlands



Attribute 1	Attribute 2	ds	y	origin
West Midlands	Pre 1919	2019-01-01	14500	Observed
West Midlands	1919-44	2019-01-01	14200	Observed
West Midlands	1945-64	2019-01-01	12600	Observed
West Midlands	1965-82	2019-01-01	11600	Observed
West Midlands	1983-92	2019-01-01	12000	Observed
West Midlands	1993-99	2019-01-01	11600	Observed
West Midlands	Post 1999	2019-01-01	9600	Observed
West Midlands	Pre 1919	2018-01-01	14300	Observed
West Midlands	1919-44	2018-01-01	14100	Observed
West Midlands	1945-64	2018-01-01	12500	Observed

« ‹ 1 / 20 › »

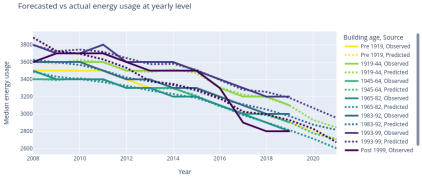
Dash test app

Energy type:

Electricity

Select region:

West Midlands



Attribute 1	Attribute 2	ds	y	origin
West Midlands	Pre 1919	2019-01-01	2900	Observed
West Midlands	1919-44	2019-01-01	3100	Observed
West Midlands	1945-64	2019-01-01	2900	Observed
West Midlands	1965-82	2019-01-01	2800	Observed
West Midlands	1983-92	2019-01-01	3000	Observed
West Midlands	1993-99	2019-01-01	3200	Observed
West Midlands	Post 1999	2019-01-01	2800	Observed
West Midlands	Pre 1919	2018-01-01	3000	Observed
West Midlands	1919-44	2018-01-01	3200	Observed
West Midlands	1945-64	2018-01-01	3000	Observed

« ‹ 1 / 19 › »

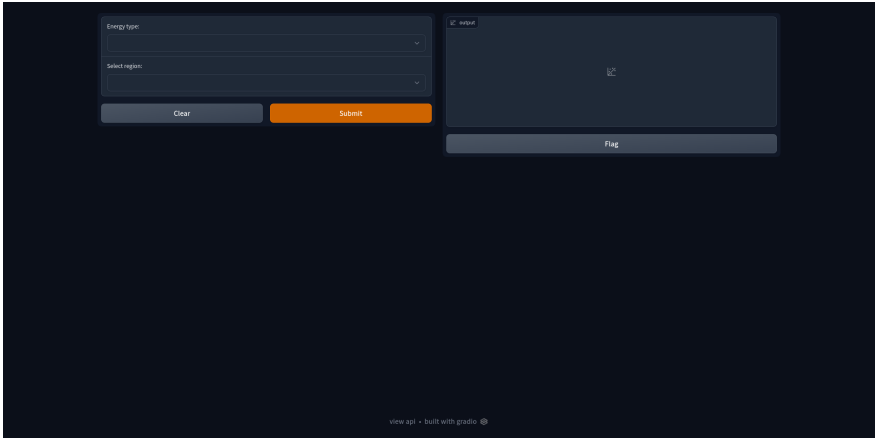
Gradio

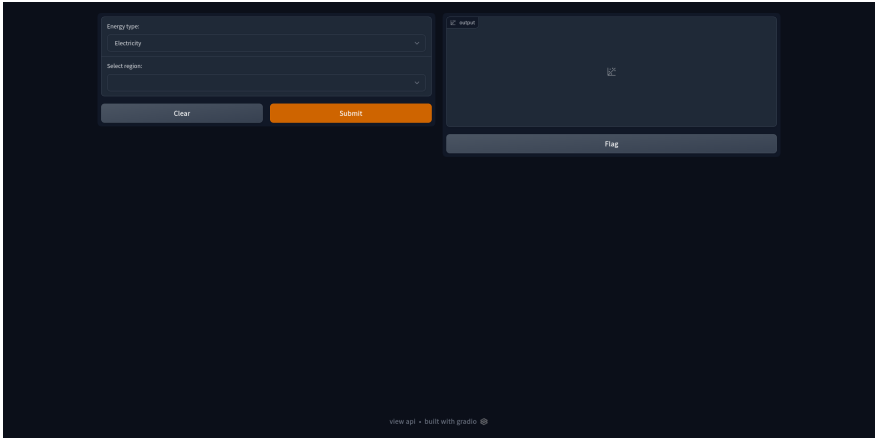
- "Gradio is the fastest way to demo your machine learning model with a friendly web interface" - only needs a few lines of code
- You can host your demo on Hugging Face & get a link you can share.
- You must specify three parameters to create your Interface:
  - The function to create the GUI
  - The desired input components
  - The desired output components

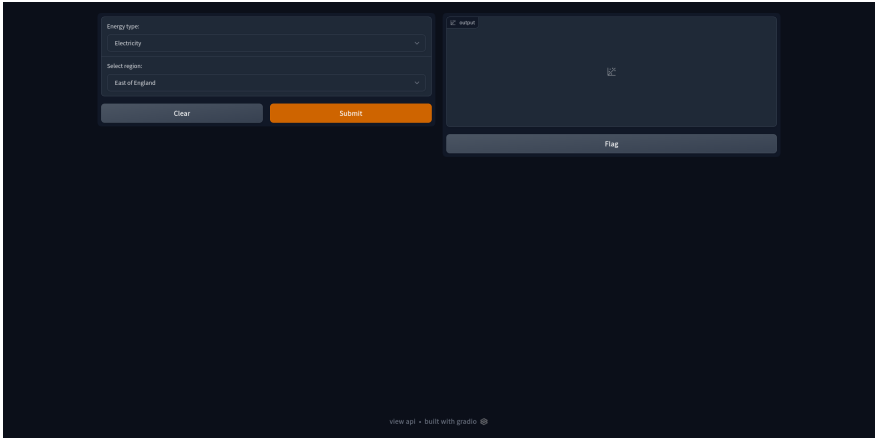
```python
import gradio as gr
import pandas as pd
from pathlib import Path
import plotly.express as px
from prophet import Prophet
pd.options.plotting.backend = "plotly"

dt = pd.read_csv(Path(__file__).parent / "NEED_data_explorer_2021.csv",
                 encoding="latin-1",
                 na_values="n/a")

REGIONS = [...]
PERIODS = [...]
GAS_DIMS = ...
ELEC_DIMS = ...
```

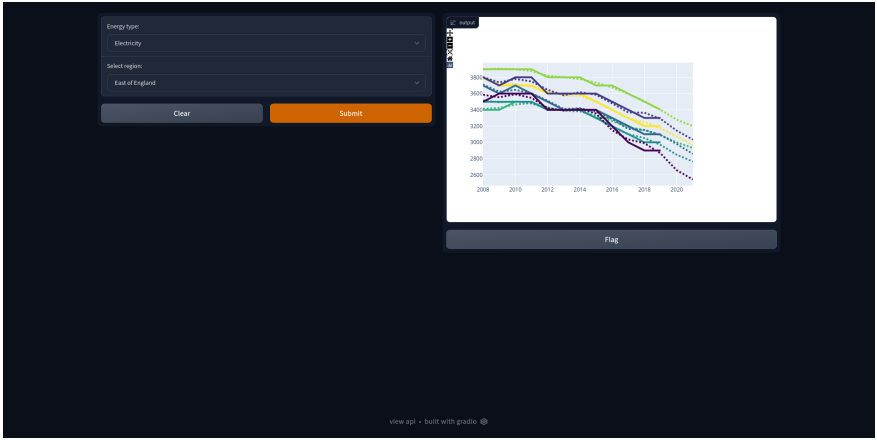
```python
def make_graph(energy_type, region_name):
    dt_chunk = dt.copy()
    dt_chunk = dt_chunk.loc[(dt_chunk["Attribute 1"] == region_name) & (dt_chunk["Attribute 2"].isin(PERIODS)),]

    if energy_type == "Gas":
        dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + GAS_DIMS]
    else:
        dt_chunk = dt_chunk[["Attribute 1", "Attribute 2"] + ELEC_DIMS]

    ...  # more data transformations

    forecast_collector = []
    for build_age in dt_chunk["Attribute 2"].unique().tolist():
        history = dt_chunk[dt_chunk["Attribute 2"] == build_age][["ds", "y"]]
        history["ds"] = pd.to_datetime(history["ds"])
        m = Prophet()
        m.fit(history)
        ...
        forecast_collector.append(forecast)
    forecast_collector = pd.concat(forecast_collector)

    ...  # more data transformations
    output = pd.concat([dt_chunk, forecast_collector])
    ...

    fig = px.line(...)
    return fig
```

```
demo = gr.Interface(fn=make_graph,
                    inputs=[gr.components.Dropdown(label="Energy type:", choices=["Gas", "Electricity"]),
                            gr.components.Dropdown(label="Select region:", choices=REGIONS)
                            ],
                    outputs=gr.Plot()  #,
                    # live=True  # Reactive behaviour happens when this is enabled
                    )

demo.launch()
```


Energy type:
Select region:
Clear
Submit
output
Flag
view api • built with gradio


Energy type:
Electricity
Select region:
Clear
Submit
output
Flag
view api • built with gradio


Energy type:
Electricity
Select region:
East of England
Clear
Submit
output
Flag
view api • built with gradio


Energy type:
Electricity
Select region:
East of England
Clear
Submit
output
3800
3600
3400
3200
3000
2800
2600
2008
2010
2012
2014
2016
2018
2020
Flag
view api • built with gradio

Extra options

Extra options

GlobalLogic
A Hitachi Group Company

- Anvil
- Flask
- Django
- Voila/Panel + (PyScript +) Jupyter Notebooks

Outline

You can use whatever you're comfortable with!

You can use whatever you're comfortable with!

But...

Here are some suggested criteria / decision points:

Conclusions I

GlobalLogic
A Hitachi Group Company

1. By computational complexity & based on the rest of the ecosystem:
    - For complex 'statsy' problems: Shiny in R to leverage all the existing R packages which have been tested more thoroughly (e.g., LMMs, GAMMs)
    - For complex 'ML' problems: Python alternatives - probs Gradio or even Streamlit / Dash
    - For less complex analytics/BI problems: Shiny/Dash (in either language)

2. By project maturity:

   - If POC/demo: Streamlit/Gradio
   - If full-blown web project containing a lot more than DS/analytics, maybe Django, or at least Dash/Shiny

- Other differences/classifications will exist probably, e.g., performance.

- Other differences/classifications will exist probably, e.g., performance.
- But.... avoid bikeshedding.
- Might also be able to switch from one to another as a project evolves, e.g.: from Dash to Django via `django-plotly-dash` library

Outline

Want to know more?

GlobalLogic
A Hitachi Group Company

To keep this short, here is a good place to start:

- https://www.datarevenue.com/en-blog/data-dashboarding-streamlit-vs-dash-vs-shiny-vs-voila
- Much more is out there!


GlobalLogic®
A Hitachi Group Company
Thanks for listening!
Any questions?
@c__constantine