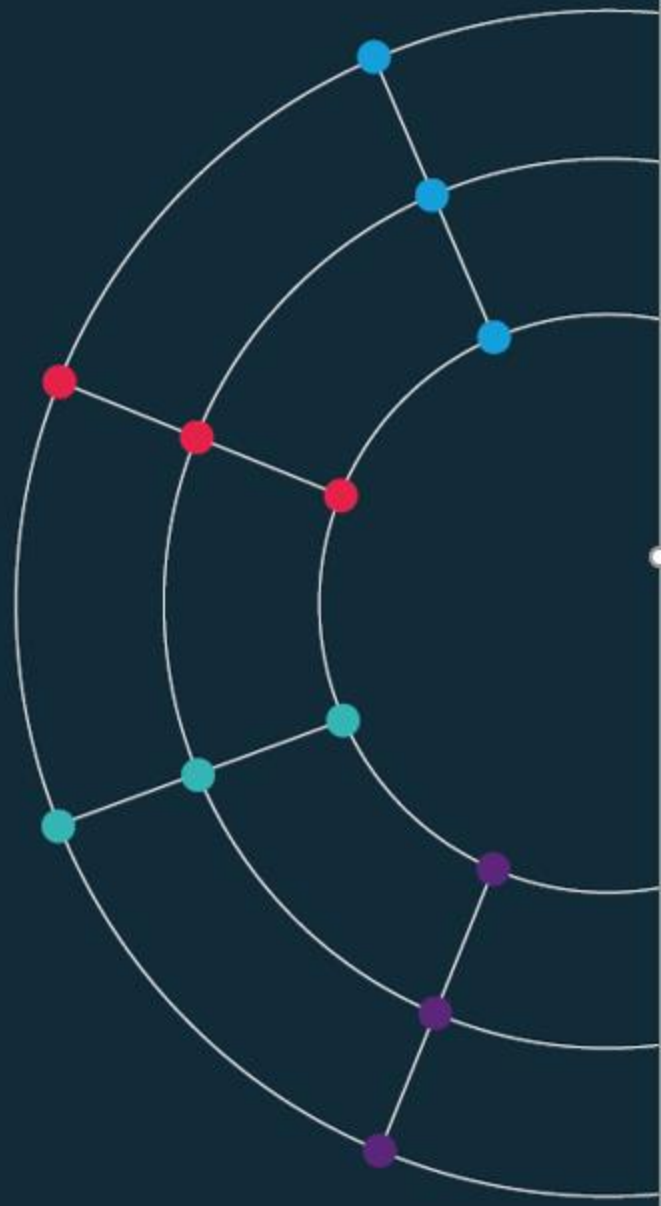
Session 3.

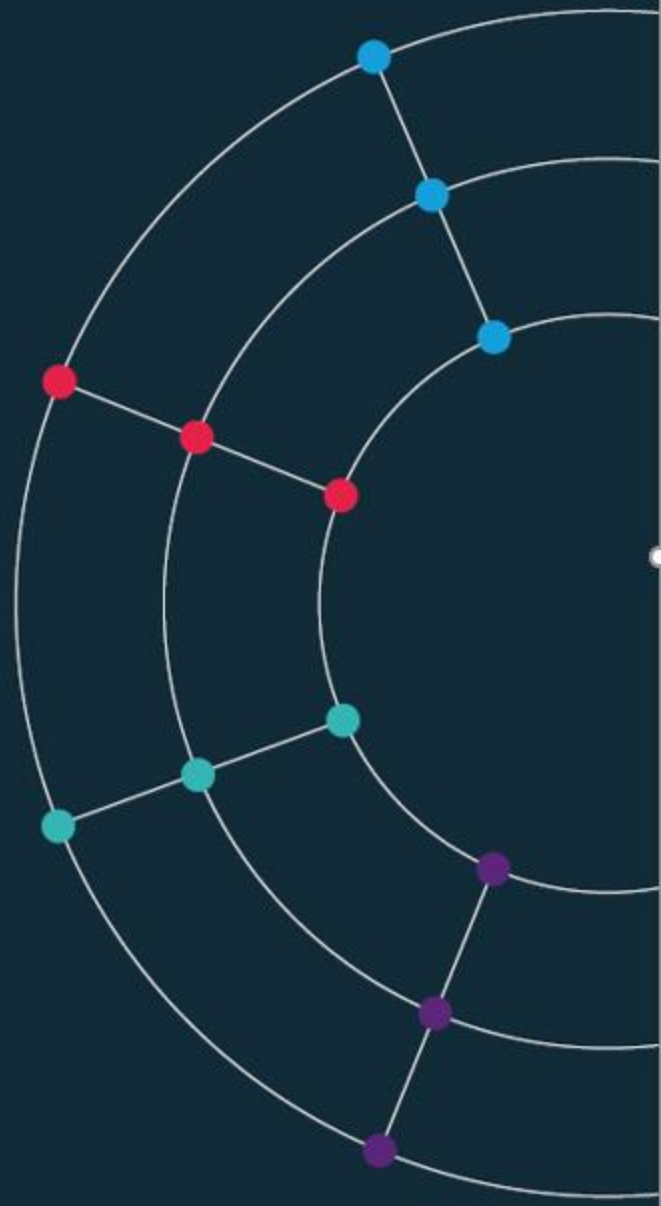
Accessing data programmatically



Session 3.

Accessing data programmatically

Introduction to control functions



The idea.

Control structures | Flow control | Control statements

We want our programs/analysis to take decisions for us. Not to continue doing the same thing again and again, but to be able to decide what to do next.

Without control structures (AKA ‘flow control’) programs don’t do much.

What might you want a program to do for you?

- Stop or start.

- Take a decision on what to do next. Do different things in different conditions:
  - Time of day, or days of the week;
  - If data has certain properties: (stock market alert).

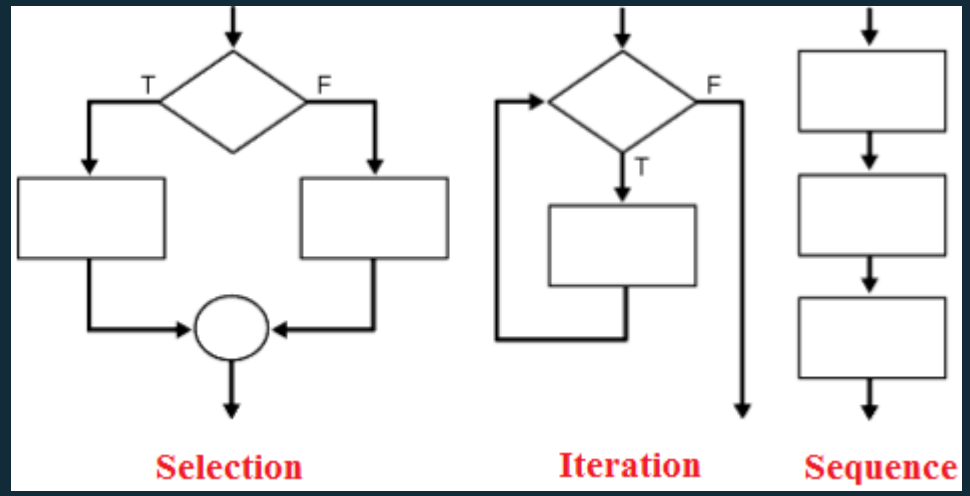
- Do something many times.
  - Dynamic programming / maximisation;
  - Batches of analysis: downloading, cleaning, charting.

The big three.

Sequence | Iteration | Conditionality/Selection

Programming languages typically feature three types of control:

- Sequence. Tells the program the order to do things in.
- Selection/conditionality. Makes a decision based on a testable condition.
- Iteration. Repeats a command over and over again, until a condition is met. Then stop and continue the code.



Control in data science.

If-else | Loops

In practical terms during a career in data you are going to make daily use two particular examples of these general ideas.

- If-else. Test some condition in your data. Based on the results of this test, take a number of different actions.

- Loops. Do the same thing to many pieces of data, many variables, many data sets. Or do the same thing on a number of different days.

These can be combined all possible ways.

- An if statement inside an if (AKA “nested”)
- A loop withing a loop.
- If inside a loop
- Loop inside an if.

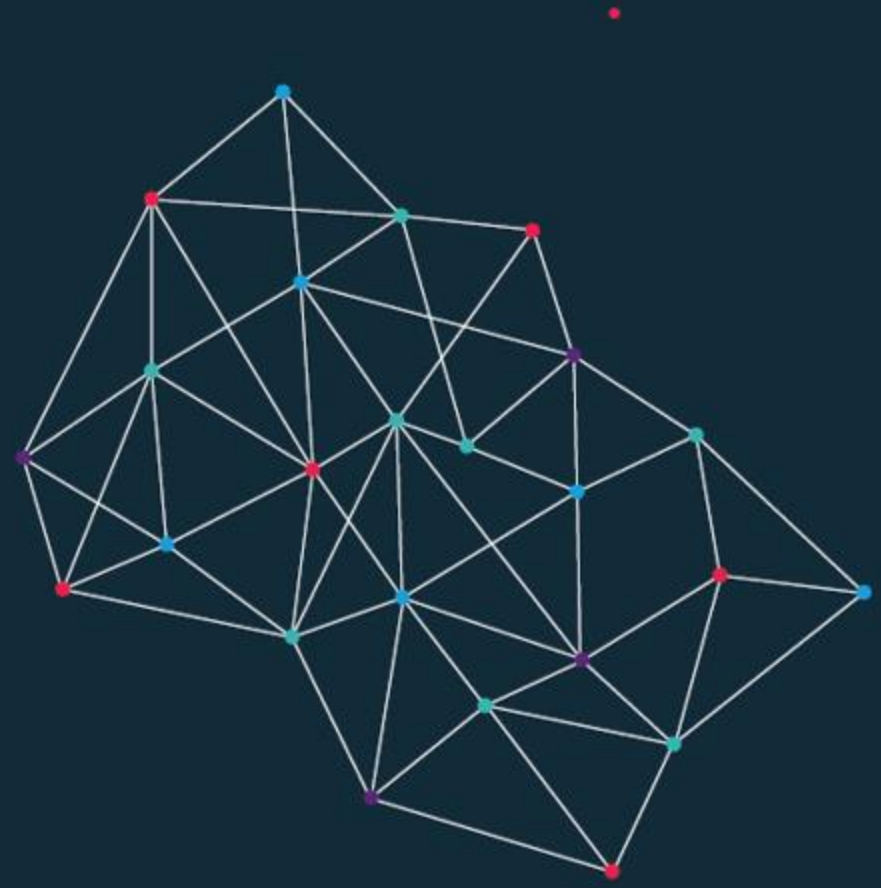
How many languages?

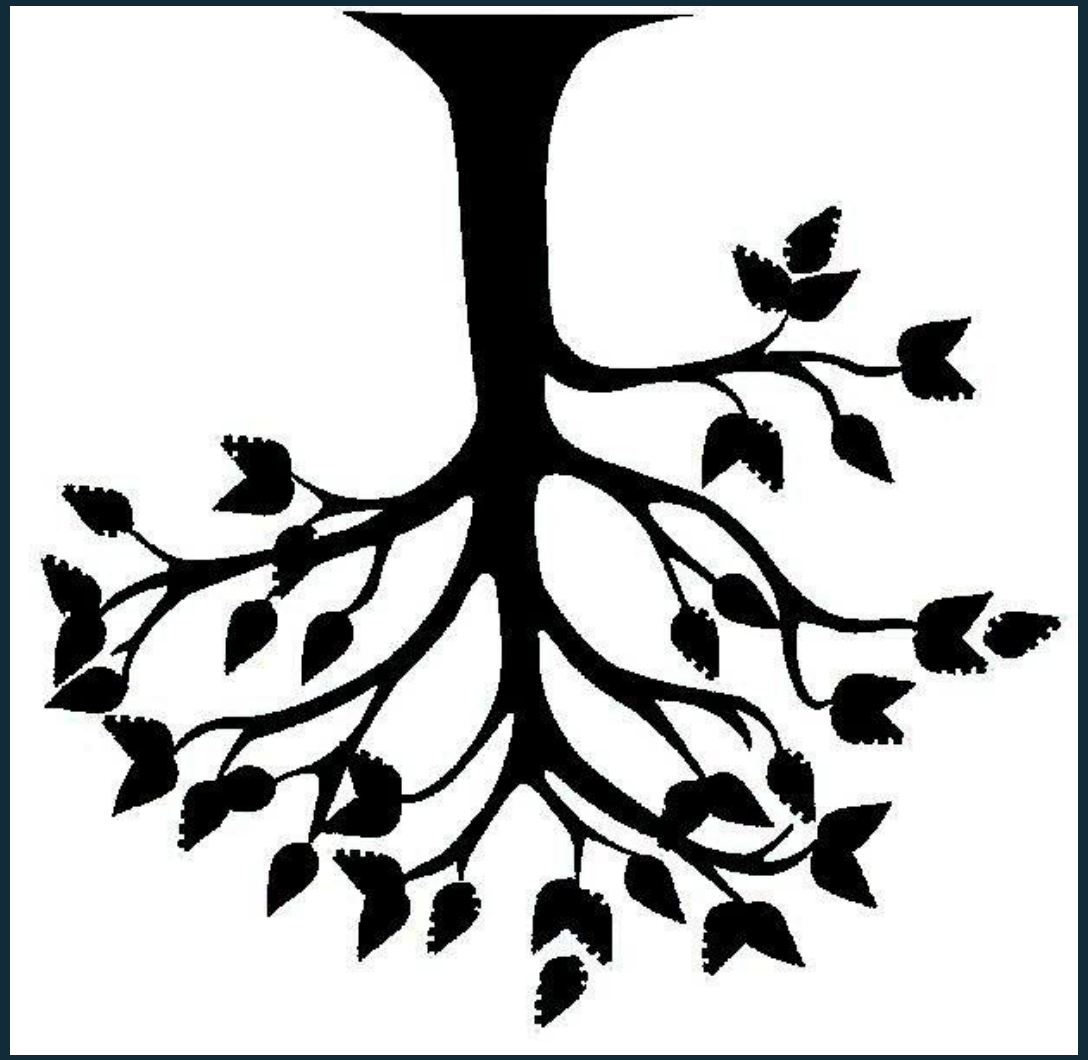
Five?

How many languages?

Three?

Conditionals.





Code as a tree |
Conditional statements influence which branch your code goes along.

If-else.

Excel | Python | JavaScript | CSS

Syntax:

IF({CONDITION}, {VALUE IF TRUE}, {VALUE IF FALSE})

Example:
=if(A1=“Richard”, “Yes”, “No”)

If-else.

Excel | Python | JavaScript | CSS

Practical usage:

Data cleaning

You have two data sets on countries that you want to match.
But the names are not consistent.
You want to create a column that provides a 1:1 correspondence between the data sets

=if(A1="Côte d'Ivoire", "Ivory Coast", A1)

Categorisation

You have data on firm size, measured by number of employees…
You want to analyse based on two types, large firms and SMEs..
You define SME as anything with up to 250 employees..

=if(A1>250, "Large", "SME")

If-else.

Excel | Python | JavaScript | CSS

```
# Example: how big are firms in our dataset?

firm1 = 14
firm2 = 250

# Comparing values
if firm1 > firm2:
  print("Firm 1 has more employees than Firm 2")
else:
  print("Firm 2 has more employees than Firm 1")

# Assessing size
if firm1 > 249:
  print("Firm 1 is large")
else:
  print("Firm 1 is an SME")
```

If-else.

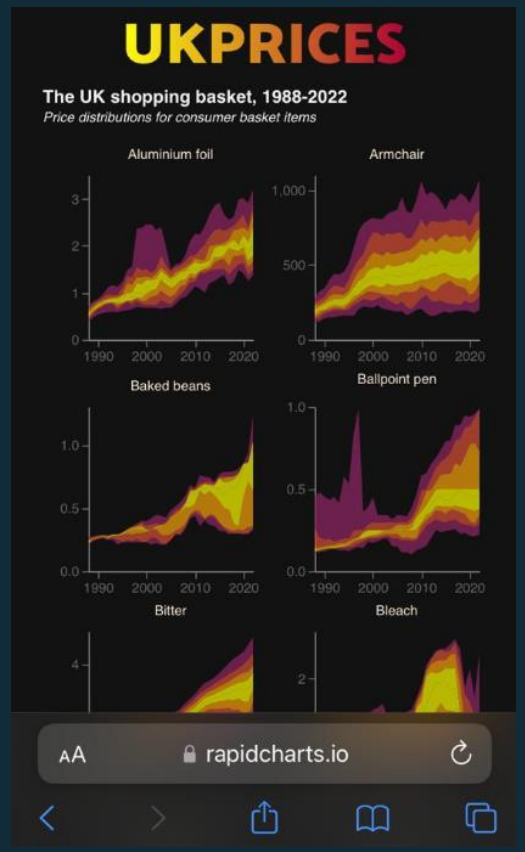
Excel | Python | JavaScript | CSS

```
//Simple if condition
if (condition) {
    //  Add code here--can be many lines, to run if the condition is TRUE.
    //  If the condition is FALSE then nothing happens
}

//If else
if (condition) {
    //  Code to run if condition TRUE.
  } else {
    //  Code to run if condition FALSE.
  }

//If-elif-else:
if (conditionA) {
    //  Code to run if conditionA TRUE.
  } else if (conditionB) {
    //  Code to run if conditionB TRUE.
  } else {
    //  Code to run if BOTH conditionA and conditionB are FALSE.
  }
```

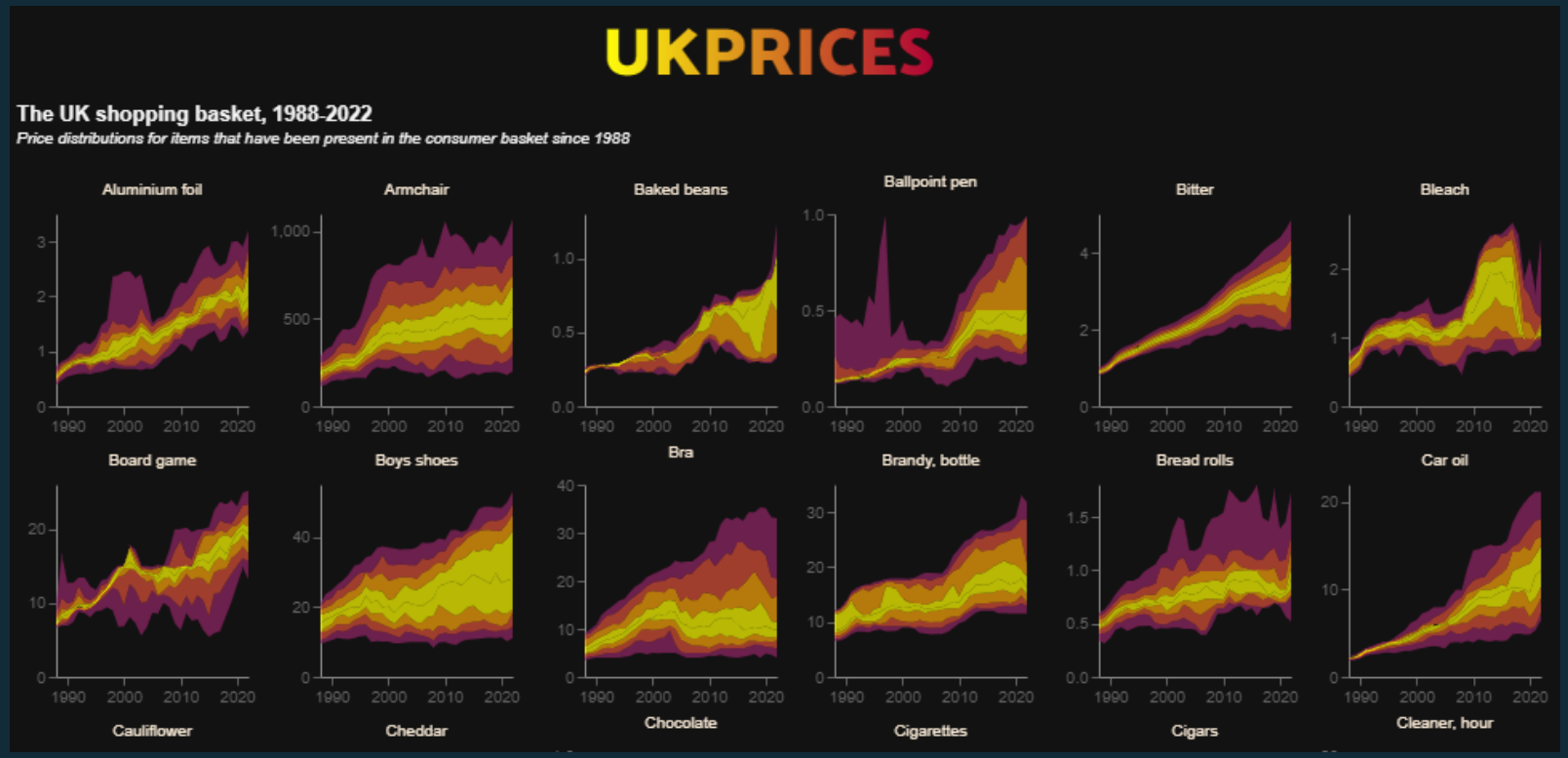
Example: screen sizes

Excel | Python | JavaScript | CSS

Phone



Desktop



Example: screen sizes

Excel | Python | JavaScript | CSS

```
<!-- CONDITIONAL SCRIPT TO EMBED BASED ON SCREEN WIDTH -->
<script>

// Find the current screen width:
let width = screen.width;

// Use an if function to pick the approprite visualisation:
if (width > 950) {
  prices1 = "charts/ONSinflation/distributionsPerrenials_DarkWide.json";
} else if (width > 450) {
  prices1 = "charts/ONSinflation/distributionsPerrenials_DarkMedium.json";
} else {
  prices1 = "charts/ONSinflation/distributionsPerrenials_DarkNarrow.json";
}

// Now embed the chart, which will vary based on screen width:
vegaEmbed('#chart1', prices1, {"actions": false});

</script>
<!-- END - CONDITIONAL SCRIPT TO EMBED BASED ON SCREEN WIDTH -->
```

If-else.

Excel | Python | JavaScript | CSS

CSS does not officially support conditionals.

But there is a way we can make CSS react in different ways in different situations?

Media Queries (@media) allow you to change the look based on screen size:

- With these you are saying:
- If the screen size is {CONDITION} then do this {ACTION}
- So CSS does allow conditionals.
- Understanding this is the way to make your site have two looks: one for mobile, one for laptop/desktop.

If-else.

Excel | Python | JavaScript | CSS

```css
/* Some CSS to alter the colour of my site */
body{
    background-color: white;
}

/* Screen size 1 */
@media screen and (max-width: 450px) {
    body {
       background-color: lightblue;
    }
  }

/* Screen size 2 */
@media screen and (max-width: 600px) {
    body {
       background-color: blue;
    }
  }

/* Screen size 3 */
@media screen and (max-width: 800px) {
    body {
       background-color: pink;
    }
  }
```

What colour on:

Smartphone?

Laptop?

iPad?

If-else.

Excel | Python | JavaScript | CSS

```
/* Some CSS to alter the colour of my site */
body{
    background-color: white;
}

/* Screen size 1 */
@media screen and (max-width: 450px) {
    body {
       background-color: lightblue;
    }
  }

/* Screen size 2 */
@media screen and (max-width: 600px) {
    body {
       background-color: blue;
    }
  }

/* Screen size 3 */
@media screen and (max-width: 800px) {
    body {
       background-color: pink;
    }
  }
```

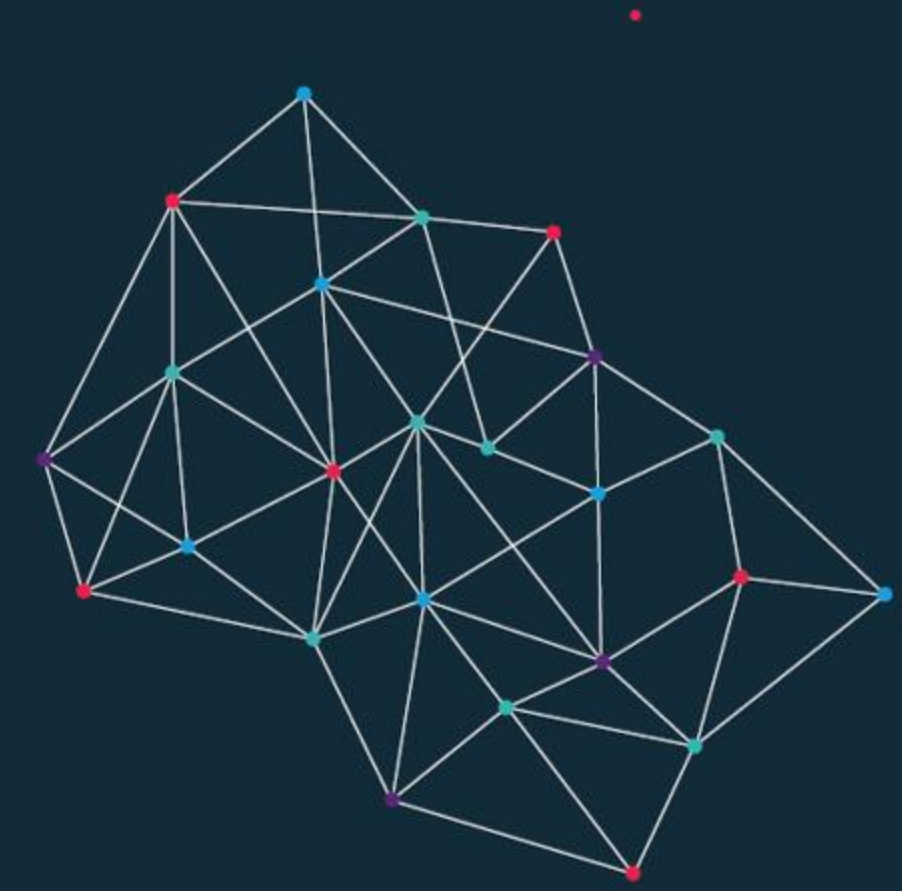
What colour on:

Smartphone? (400px wide)

Laptop? (~1350px)

iPad? (768x1024)

Loops.



Loops.

Excel | Python | JavaScript | CSS

The Excel GUI is not built for iteration.

You can write loops in its background language VBA

This is fiddly

Loops.

Excel | Python | JavaScript | CSS

```
for i in range (1, 100):
    print i

for i in range (1, 10, 100):
    print i

for i in range (1, 100, 10):
    print i


variables = ["debt", "deficit", "GDP", "inflation"]
for i in variables:
  print(i)
```

An example of subtle differences between languages / functions.

Loops.

Excel | Python | JavaScript | CSS

```
for (statement_1; statement_2; statement_3) {
    // Code block to run
}
```

What happens here:

- Statement_1 runs once, before the code block starts.
- Statament_2 defines a condition that must hold for the code block to run.
- Statement_3 runs each time the code block has been executed.

Loops.

Excel | Python | JavaScript | CSS

```
for (let i = 1; i < 101; i++) {
    console.log(i);
  }


for (let i = 1; i < 101; i+10) {
    console.log(i);
  }
```

Loops.

Excel | Python | JavaScript | CSS

```
// Set a list of variables:
variables = ["debt", "deficit", "GDP", "inflation"]

// We can index these:
console.log(variables)
console.log(variables[0])
console.log(variables[3])

// Work out how long this thing is:
len = variables.length

// Iterate though it, printing out each particular variable
for (let i=0; i<len; i++) {
    x = variables[i]
    console.log(x)
}
```

Loops.

Excel | Python | JavaScript | CSS

Again, CSS is not really built for looping

However, it can repeat commands. An example: @keyframes (an At-Rule).

```
@keyframes border-pulsate2 {
  0%   { border-color: #22e68b}
  10%  { border-color: rgba(230, 34, 75, 0)}
  20%  { border-color: rgba(230, 34, 75)}
  30%  { border-color: rgba(230, 34, 75, 0)}
  40%  { border-color: rgba(0, 47, 167)}
  50%  { border-color: rgba(230, 34, 75, 0)}
  60%  { border-color: rgba(250, 250, 0)}
  70%  { border-color: rgba(230, 34, 75, 0)}
  80% { border-color: #22e68b}
}

.readingWeek {
  background-color: #122b39;
  border-radius: 10px;
  border-style: dotted;
  border-width: 5px;
  height:100%;
  animation: border-pulsate2 2s infinite;
}
```

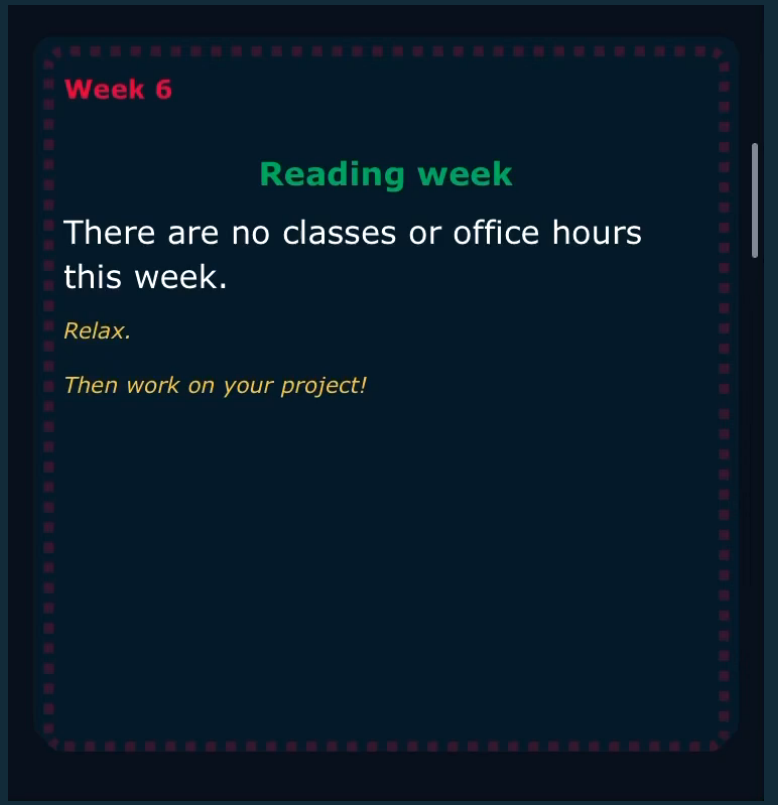
Loops.

Excel | Python | JavaScript | CSS

Again, CSS is not built for looping

However, it can iterate over certain commands for a given time. One example

```css
@keyframes border-pulsate2 {
  0%   { border-color: #22e68b}
  10%  { border-color: rgba(230, 34, 75, 0)}
  20%  { border-color: rgba(230, 34, 75)}
  30%  { border-color: rgba(230, 34, 75, 0)}
  40%  { border-color: rgba(0, 47, 167)}
  50%  { border-color: rgba(230, 34, 75, 0)}
  60%  { border-color: rgba(250, 250, 0)}
  70%  { border-color: rgba(230, 34, 75, 0)}
  80% { border-color: #22e68b}
}

.readingWeek {
  background-color: #122b39;
  border-radius: 10px;
  border-style: dotted;
  border-width: 5px;
  height:100%;
  animation: border-pulsate2 2s infinite;
}
```



Using loops

Using loops.

Batching analysis

Problem: We often have too much data
How can visualising help us understand the data in the early stage of a project?

Code intuition:
- Think of a chart that summarises the data in some useful way:

[Useful options: histogram, percentiles as a swathe.]

- Code this chart for one part (one firm, one year, one product).

- Use a loop: for each product, draw the chart

Example: UK CPI data, which has ~1300 products.

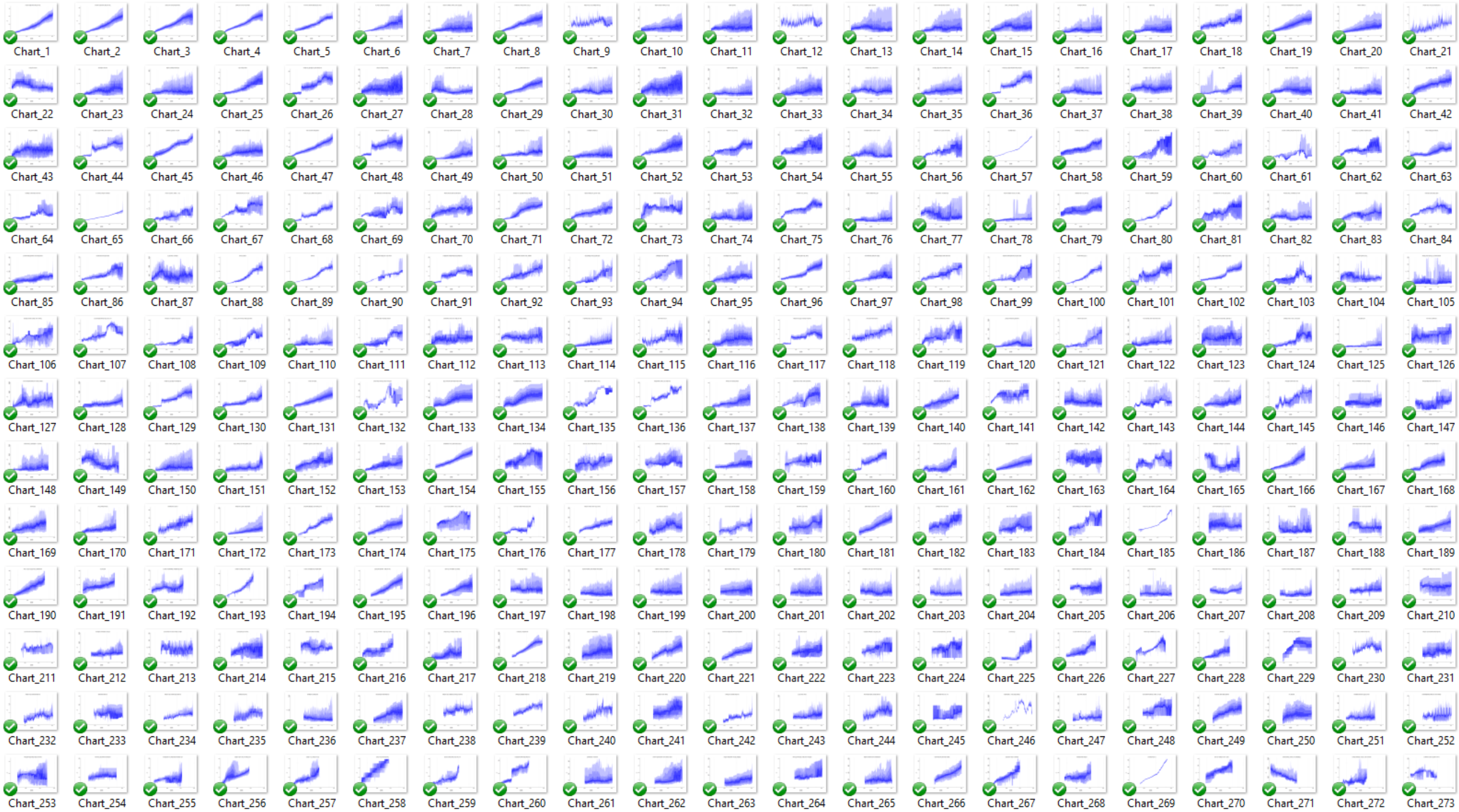


Chart_1 Chart_2 Chart_3 Chart_4 Chart_5 Chart_6 Chart_7 Chart_8 Chart_9 Chart_10 Chart_11 Chart_12 Chart_13 Chart_14 Chart_15 Chart_16 Chart_17 Chart_18 Chart_19 Chart_20 Chart_21
Chart_22 Chart_23 Chart_24 Chart_25 Chart_26 Chart_27 Chart_28 Chart_29 Chart_30 Chart_31 Chart_32 Chart_33 Chart_34 Chart_35 Chart_36 Chart_37 Chart_38 Chart_39 Chart_40 Chart_41 Chart_42
Chart_43 Chart_44 Chart_45 Chart_46 Chart_47 Chart_48 Chart_49 Chart_50 Chart_51 Chart_52 Chart_53 Chart_54 Chart_55 Chart_56 Chart_57 Chart_58 Chart_59 Chart_60 Chart_61 Chart_62 Chart_63
Chart_64 Chart_65 Chart_66 Chart_67 Chart_68 Chart_69 Chart_70 Chart_71 Chart_72 Chart_73 Chart_74 Chart_75 Chart_76 Chart_77 Chart_78 Chart_79 Chart_80 Chart_81 Chart_82 Chart_83 Chart_84
Chart_85 Chart_86 Chart_87 Chart_88 Chart_89 Chart_90 Chart_91 Chart_92 Chart_93 Chart_94 Chart_95 Chart_96 Chart_97 Chart_98 Chart_99 Chart_100 Chart_101 Chart_102 Chart_103 Chart_104 Chart_105
Chart_106 Chart_107 Chart_108 Chart_109 Chart_110 Chart_111 Chart_112 Chart_113 Chart_114 Chart_115 Chart_116 Chart_117 Chart_118 Chart_119 Chart_120 Chart_121 Chart_122 Chart_123 Chart_124 Chart_125 Chart_126
Chart_127 Chart_128 Chart_129 Chart_130 Chart_131 Chart_132 Chart_133 Chart_134 Chart_135 Chart_136 Chart_137 Chart_138 Chart_139 Chart_140 Chart_141 Chart_142 Chart_143 Chart_144 Chart_145 Chart_146 Chart_147
Chart_148 Chart_149 Chart_150 Chart_151 Chart_152 Chart_153 Chart_154 Chart_155 Chart_156 Chart_157 Chart_158 Chart_159 Chart_160 Chart_161 Chart_162 Chart_163 Chart_164 Chart_165 Chart_166 Chart_167 Chart_168
Chart_169 Chart_170 Chart_171 Chart_172 Chart_173 Chart_174 Chart_175 Chart_176 Chart_177 Chart_178 Chart_179 Chart_180 Chart_181 Chart_182 Chart_183 Chart_184 Chart_185 Chart_186 Chart_187 Chart_188 Chart_189
Chart_190 Chart_191 Chart_192 Chart_193 Chart_194 Chart_195 Chart_196 Chart_197 Chart_198 Chart_199 Chart_200 Chart_201 Chart_202 Chart_203 Chart_204 Chart_205 Chart_206 Chart_207 Chart_208 Chart_209 Chart_210
Chart_211 Chart_212 Chart_213 Chart_214 Chart_215 Chart_216 Chart_217 Chart_218 Chart_219 Chart_220 Chart_221 Chart_222 Chart_223 Chart_224 Chart_225 Chart_226 Chart_227 Chart_228 Chart_229 Chart_230 Chart_231
Chart_232 Chart_233 Chart_234 Chart_235 Chart_236 Chart_237 Chart_238 Chart_239 Chart_240 Chart_241 Chart_242 Chart_243 Chart_244 Chart_245 Chart_246 Chart_247 Chart_248 Chart_249 Chart_250 Chart_251 Chart_252
Chart_253 Chart_254 Chart_255 Chart_256 Chart_257 Chart_258 Chart_259 Chart_260 Chart_261 Chart_262 Chart_263 Chart_264 Chart_265 Chart_266 Chart_267 Chart_268 Chart_269 Chart_270 Chart_271 Chart_272 Chart_273

Using loops.

Why are they so helpful?

Tutorial: Short verion of FRED downloader - limited comments

In [1]:

```
# // FRED DOWNLOADER

# // Import things:
import requests
import json
from google.colab import files
import os

# // Base url, filename, and series. url and filename have {} and will vary with each iteration of the loop
url_base = "https://api.stlouisfed.org/fred/series/observations?series_id={}&api_key=22ee7a76e736e32f54f5d1
file_base = "data_FRED-{}.json"
fredSeries = ['PCEPI', 'CPIAUCSL', 'PAYEMS', 'DGS10', 'INDPRO', 'UNRATE', 'LES1252881600Q'] ## Add the seri

# // Downloader loop:
for i in fredSeries:
   URL = url_base.format(i) # Build the url
   data = requests.get(URL).json() # Request data from the url
   fileName = file_base.format(i) # Make a new filename
   with open(fileName, 'w', encoding='utf-8') as f:
     json.dump(data, f, ensure_ascii=False, indent=4) # Put the json data into this new file
   files.download(fileName) # Download the file.
```

What is an API?

- Application Programming Interface
- An API is a software intermediary that allows two applications to talk to each other. They are everywhere: each time you use an app like Facebook or Instagram, send an instant message, or check your weather app on your phone, you are using an API (example: Apple Watch)
- APIs are extremely useful to developers and data scientists because they provide a way to share/access abstracted data and display/use it in a chosen setting

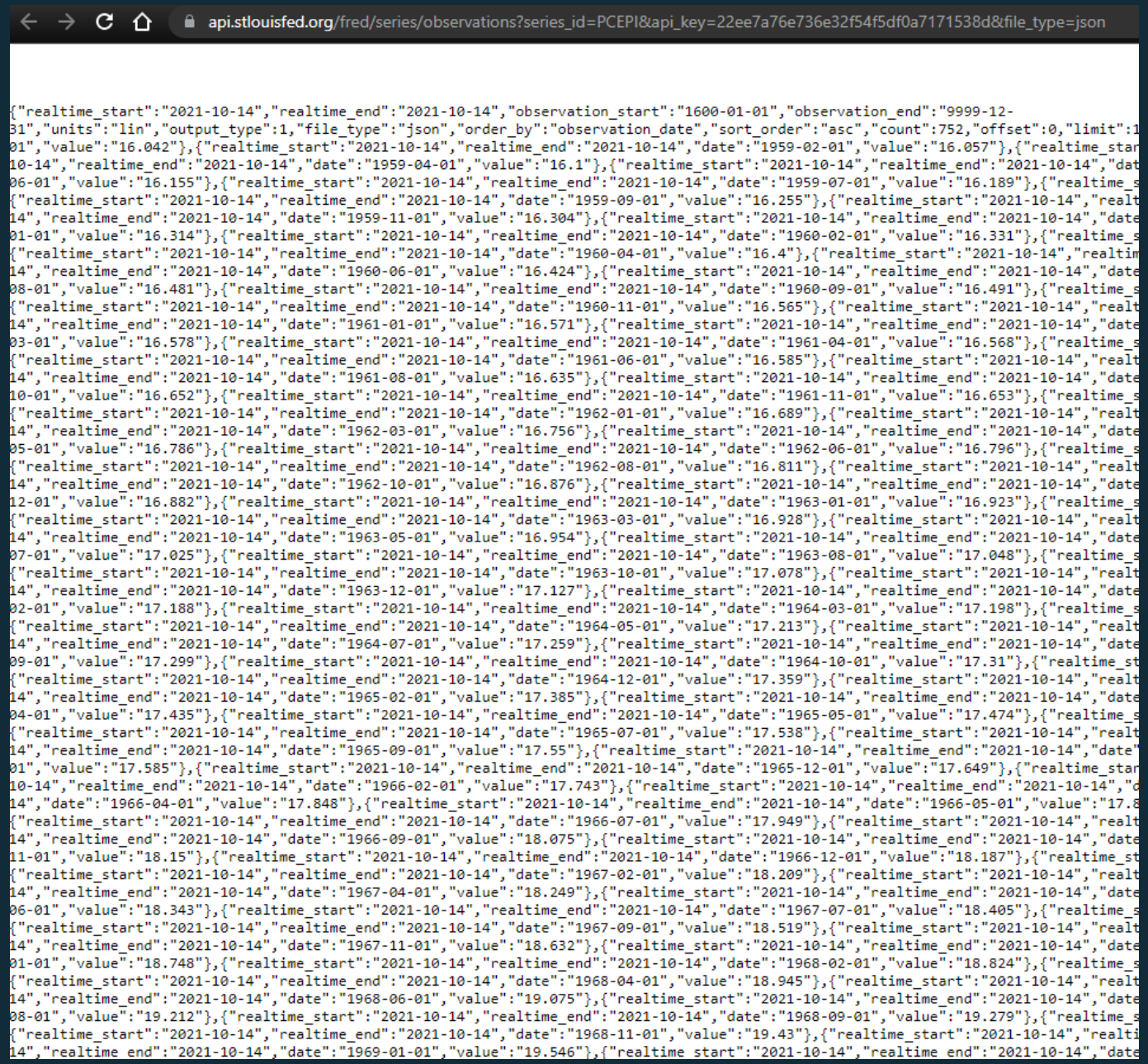
APIs and data science.

- APIs are useful for data scientists because they allow us to automate data collection. Rather than manually downloading new data and re-uploading it to a server (like GitHub), we can ask our computer to 'talk' to another computer
- Some examples:
  - Covid-19 UK official data
  - Office for National Statistics
  - Nomics (cryptocurrencies)
  - Emissions

API guidance.

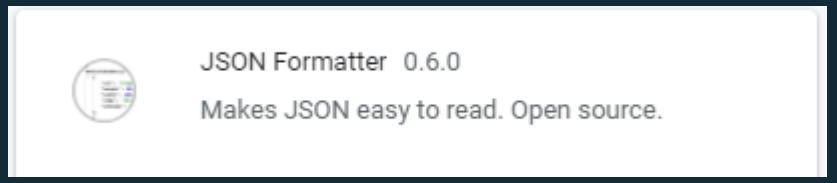
- They all look different but have a similar set up.
- A base url: e.g. https://api.stlouisfed.org/fred/series/observations?
- A series of options you can choose: series_id= file_type= time_start=
- Often a request for your API key: api_key=
- Often, when the API requires more information/choices from you, a series of & symbols. An example:

  https://api.stlouisfed.org/fred/series/observations?series_id=UNRATE&api_key=22ee7a76e736e32f54f5df0a7171538d&file_type=json



Raw JSON.

Download a JSON formatter plug in for your browser



api.stlouisfed.org/fred/series/observations?series_id=UNRATE&api_key=22ee7a76e736e32f54f5df0a7171538d&file_type=json


```
{
  "realtime_start": "2021-10-14",
  "realtime_end": "2021-10-14",
  "observation_start": "1600-01-01",
  "observation_end": "9999-12-31",
  "units": "lin",
  "output_type": 1,
  "file_type": "json",
  "order_by": "observation_date",
  "sort_order": "asc",
  "count": 885,
  "offset": 0,
  "limit": 100000,
  "observations": [
    {
      "realtime_start": "2021-10-14",
      "realtime_end": "2021-10-14",
      "date": "1948-01-01",
      "value": "3.4"
    },
    {
      "realtime_start": "2021-10-14",
      "realtime_end": "2021-10-14",
      "date": "1948-02-01",
      "value": "3.8"
    },
    {
      "realtime_start": "2021-10-14",
      "realtime_end": "2021-10-14",
      "date": "1948-03-01",
      "value": "4.0"
    },
    {
      "realtime_start": "2021-10-14",
      "realtime_end": "2021-10-14",
      "date": "1948-04-01",
      "value": "3.9"
    },
    {
      "realtime_start": "2021-10-14",
      "realtime_end": "2021-10-14",
      "date": "1948-05-01",
      "value": "3.5"
    },
    {
      "realtime_start": "2021-10-14",
      "realtime_end": "2021-10-14",
      "date": "1948-06-01",
      "value": "3.6"
```

Unemployment Rate (UNRATE)

DOWNLOAD

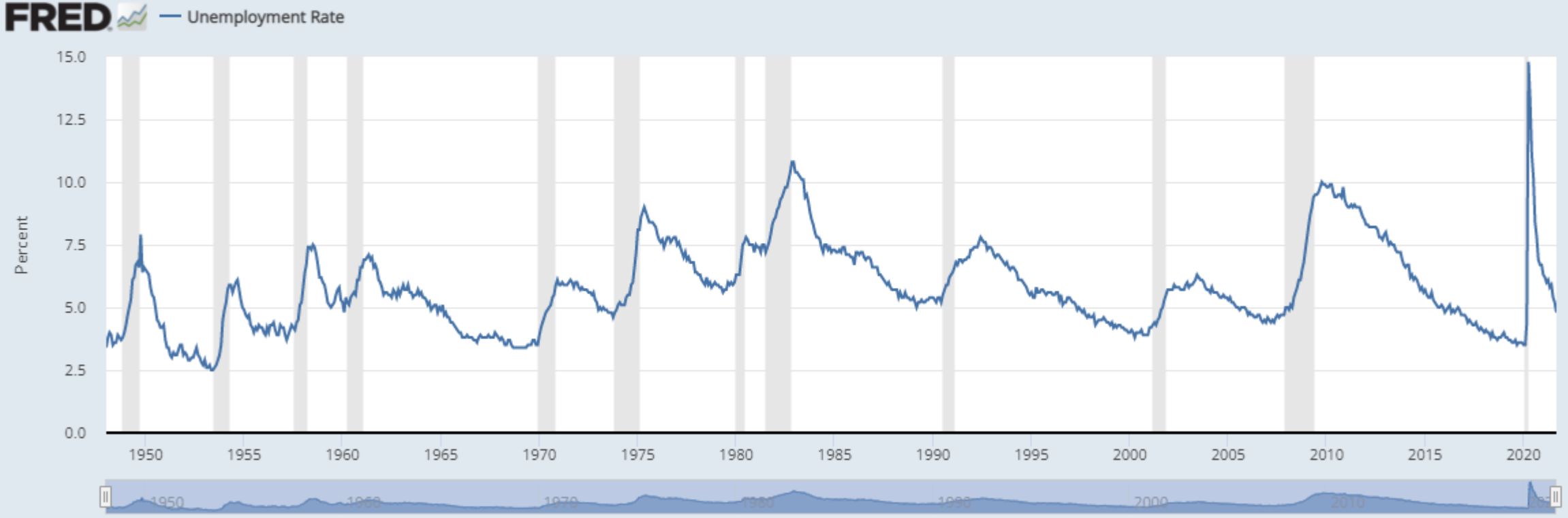
Observation: Sep 2021: 4.8 (+ more)
Updated: Oct 8, 2021

Units: Percent, Seasonally Adjusted

Frequency: Monthly

1Y | 5Y | 10Y | Max 1948-01-01 to 2021-09-01

EDIT GRAPH



Shaded areas indicate U.S. recessions.

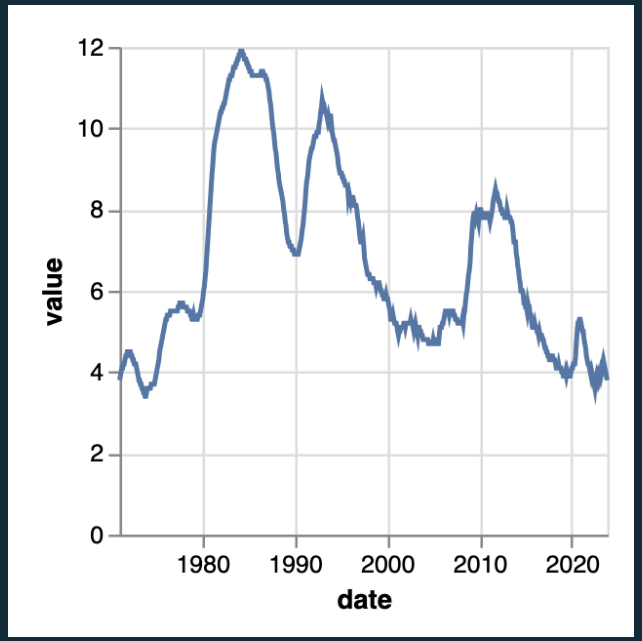
Source: U.S. Bureau of Labor Statistics

fred.stlouisfed.org

Worked example.

This chart pulls data from the Economics Observatory API:

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",

  "data": {"url": "https://api.economicsobservatory.com/gbr/unem?vega"},

  "mark": "line",

  "encoding": {

    "x": {"field": "date", "type": "temporal"},

    "y": {"field": "value", "type": "quantitative"}
  }
}
```



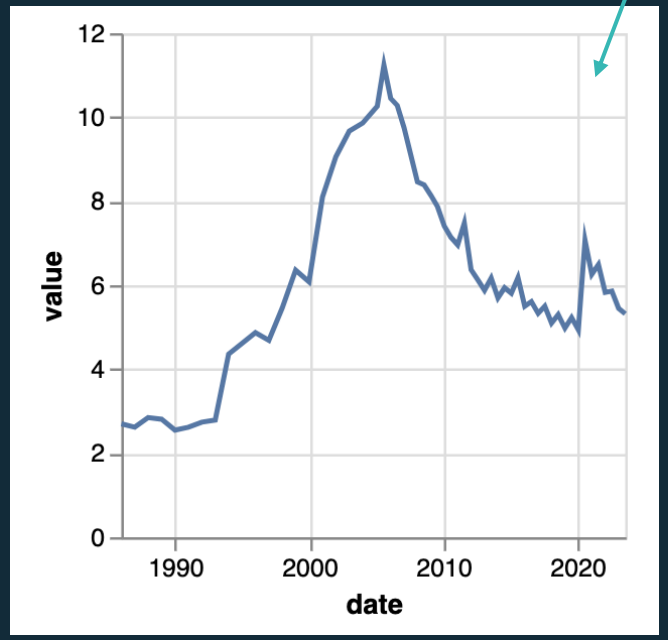
Worked example.

Edit the URL to draw data from a different country:

API URL tweaked

Chart now shows Indonesian data

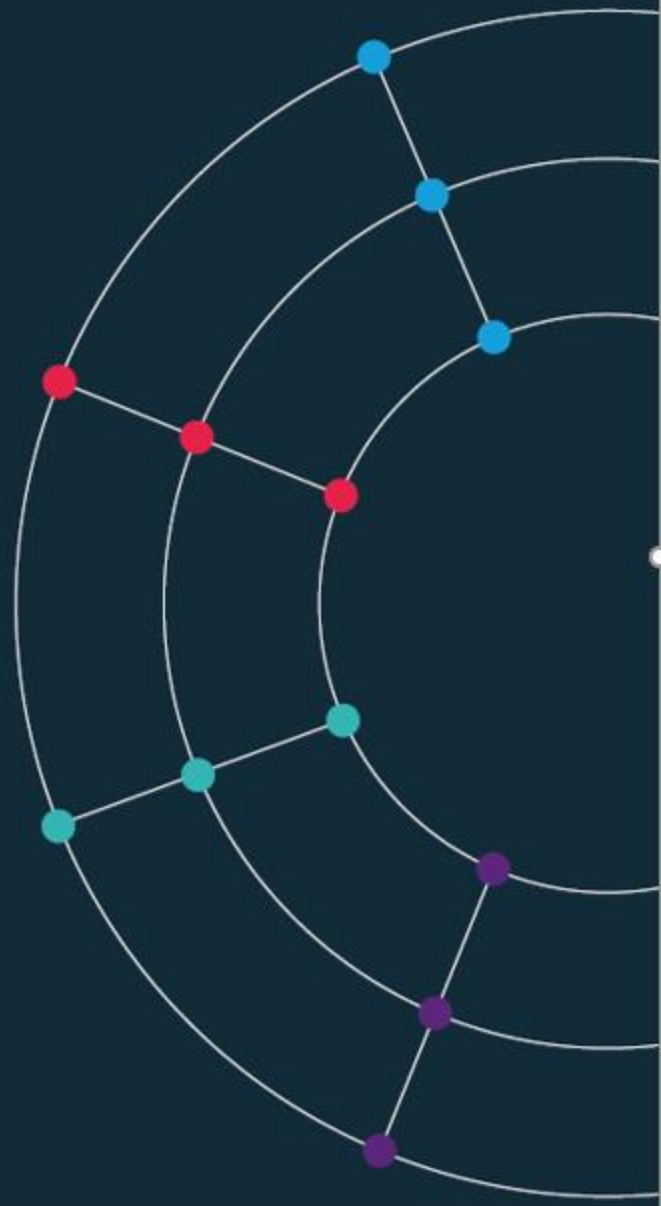
```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",

  "data": {"url": "https://api.economicsobservatory.com/idn/unem?vega"},

  "mark": "line",

  "encoding": {

    "x": {"field": "date", "type": "temporal"},

    "y": {"field": "value", "type": "quantitative"}
  }
}
```



Session 3.

Accessing data programmatically

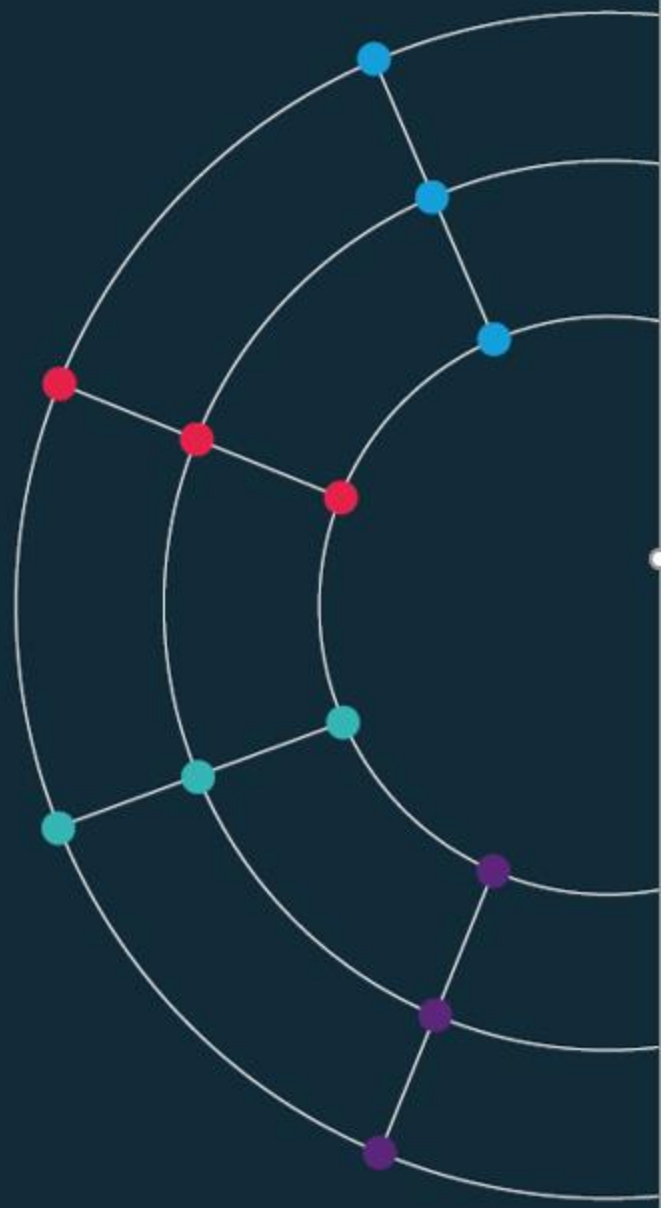
Code-along and automated data access



Session 3.

Accessing data programmatically

https://github.com/EconomicsObservatory/courses/blob/main/README.md



Code-along.

In this third practical session, we will be using Google Colab to explore Python, and then edit your website using VS Code and GitHub.

1. Quick introduction to APIs
2. Use Python to batch download data
3. Embed a chart using data accessed via batch download