Data Wrangling

What is Data Wrangling?

1. Cleaning (Preprocessing - e.g. Removing Null values)
2. Structuring (Changing to required table format- Wide/narrow etc.)
3. Enriching raw data into desired format (Keeping relevant columns/ rows (selection and slicing), aggregating etc.)

Applications:

1. Data visualization
2. Analytics
3. Machine Learning and Forecasting

Observable

What is observable?

- A place where you can -
  - Sketch with live data
  - Prototype visualizations
  - Share and reuse code
  - Publish your work for the world to see
- Supports D3 and Vega-Lite - Widely used frameworks for Web-based data visualizations
- Supports different text formats
- Please check out this tutorial!

What kind of text formats does Observable support?

- An observable notebook is made up cells
- Each cell can be filled with:
  - JavaScript
  - Markdown / Markdown summary
  - HTML
- Javascript:
  - JavaScript cells are run one line at a time, so if you want multiline expressions within a single cell, you enclose the statements in curly braces.

Markdown summary

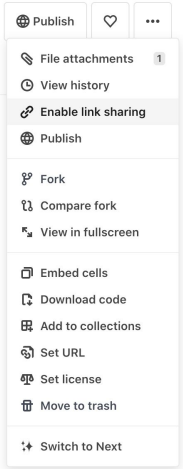
Desired style	Use the following Markdown annotation	Produces the following sample HTML
Heading 1	# Title	Title
Heading 2	## Title	Title
Heading 3	### Title	Title
Heading 4	#### Title	Title
Heading 5	##### Title	Title
Heading 6	###### Title	Title
Paragraph	Just start typing	Just start typing
Bold	**Text**	Text
Italic	*Text*	Text
~~Strike~~	~~Text~~	~~Text~~
Quoted (indent)	> Text	<blockquote> Text </blockquote>
Code (inline)	``Statement``	<code>Statement</code>

Code (fenced)	```
	Statement 1
	Statement 2
	Statement 3
	``
		<pre><code>Statement 1
		Statement 2Statement 3
		</code></pre>

List (unordered)	* List item 1 * List item 2 * List item 3	- List item 1 - List item 2 - List item 3
List (ordered)	1. List item 1 1. List item 2 1. List item 3	- List item 1 - List item 2 - List item 3
Images	![Alternate text for image](path/image.jpg)	<img src="path/image.jpg" alt="Alternative text for image>
Hyperlinks	[Link text](https://www.perthobservatory.com.au/)	Link text

Keyboard shortcuts/ Getting used to observable...

- **Running cells**
  - Use **Shift-Enter**, or the **play button in the top right corner** of each cell to run each cell.
  - The output will appear at the **top** of the cell !!!
- **Viewing or hiding the source code**
  - Click the left-hand margin
- **Pin/Unpin the source code**
  - Hover over the pin icon in the left hand margin
  - Ctrl+Shift+P
- **Forking**
- **Sharing**
  - Private - Enable link-sharing
  - Public - Publish
  - If you make changes to your notebook after the link has been shared, you will have to Reshare the link in order for others to see the changes.
- **Note: Changes you make to the notebook will not be saved unless you fork the document (A copy will be saved under your account and any changes made to the forked copy will be saved)**



Time to practice!!! 

1. Open the notebook
2. 👉**Practice**

   Add a new cell and try adding some markdown, HTML, or JavaScript

3. Use Markdown summary for reference!

Cell Values

- If you give the value in a cell a name, you can refer to it in other cells
- This will recalculate their values whenever the **named** cell is updated

```
currentWeek = 1
```

```
currentWeek = 1
```

```
"There are 9 weeks left in the quarter"
```

```
"There are " + (10 - currentWeek) + " weeks left in the quarter"
```

Time to Practice!!! 

👉Practice

Fix the RuntimeError below by adding a named cell called `exponent` with whatever value you would like.

Data formats

There are 3 common data formats

1. wide (tidy)
2. narrow (long)
3. pivoted

Wide or Tidy format

- Each variable is a separate **column**
- Each observation is a **row**
- The **values** are the **cells of the table**
- The attributes / variables are provided in the **header row**

Advantages:

- Makes it easy to look up the value for a particular attribute and observation
- Easy to extract the values using column names

Person	Age	Weight	Height
Bob	32	168	180
Alice	24	150	175
Steve	64	144	165

Narrow (or long) data format

- Attributes are included in one column and the values are included in another.
- Harder to work with because we now have to traverse the entire column to identify the values for a particular attribute (such as the age).
- How would you filter the table to remove people with height less than 170?
- This is a lot less straightforward !!!

Person	Variable	Value
Bob	Age	32
Bob	Weight	168
Bob	Height	180
Alice	Age	24
Alice	Weight	150
Alice	Height	175
Steve	Age	64
Steve	Weight	144
Steve	Height	165

Pivoted/ cross tabulation

- Attributes are included in both rows and columns
- Cell value is **dependent** on both the row and column headers
- Each column is not explicitly defined. In the example below columns are just the values of the attribute 'Year'
- The cell values can be aggregated (e.g. population) ⇒ Transforming the pivoted data back into the tidy format is **pain**
- How do you know that the values depict population? -- You need extra information !!!

Age	1850	1860	1870
0	2934165	4213008	5517185
5	2770735	3583239	4822149
10	2476213	3152990	4770011

Data Transformations

Tools!

- Datasets -
  - Both the datasets are from 'vega datasets' (Collection of datasets used in Vega and Vega-Lite examples)
  - Both of them are in JSON format
  - Cars dataset
    - Wide format
    - Columns - Name, Miles Per Gallon, Cylinders, Displacement etc.
    - 8 columns, 406 rows
  - Population dataset
    - Wide format
    - Columns - Year, Age, Sex, People
- Libraries
  - Vega Lite API
  - Arquero
  - Printable

```
import {vl} from @vega/vega-lite-api

import {aq, op} from @uwdata/arquero

import {printTable} from @uwdata/data-utilities
```

Tools!

- Vega Lite: A language for building interactive data visualizations
- Arquero:
  - A library for transforming array-backed data into tables/ data frames
  - Allows you to filter, sort, query, and derive new columns on the dataframe
- printTable: A handy function to view our data tables in Observable

Playing with the datasets

- Cars:
  - Viewing the data (JSON format)
  - Printing the top 10 rows
- Population:
  - Viewing the data (JSON format)
  - Viewing the tabular form and assigning the table value

```
cars = (await require('vega-datasets@1'))['cars.json']()
```

```
printTable(cars.slice(0,10))
```

```
population = (await require('vega-datasets@1'))['population.json']()
```

```
// viewof shows the table view, but assigns the table value
viewof population_table = aq.from(population).view()
```

Time to practice!!! 

- Think about the questions to ask (given the two datasets)...

👉Practice

> Based on your observations, how would we transform the cars dataset vs. the population dataset?

What questions can you ask?

- Cars dataset - examples
  - Which car is the heaviest?
  - Which USA-made car gives you the most miles per gallon?
  - Solution: Transform the data by **filtering and aggregating the data along the Origin column and the Miles_Per_Gallon column**
- Population dataset - examples
  - Total Female / Male population in 1850?
  - Population of kids (0-15) in 1860?
  - If the dataset is in the pivoted format, how will the transformation process change?

year	age	sex	people
1850	0	1	1483789
1850	0	2	1450376
1850	5	1	1411067
1850	5	2	1359668
1850	10	1	1260099
1850	10	2	1216114
1850	15	1	1077133
1850	15	2	1110619
1850	20	1	1017281
1850	20	2	1003841

Age	1850	1860	1870
0	2934165	4213008	5517185
5	2770735	3583239	4822149
10	2476213	3152990	4770011

Transformation- Examples

Which USA-made cars give you the most miles per gallon?

- Strategy:
  - Keep the rows with origin = USA (Note :we also want to allow 'usa' or 'Usa' or 'USA') - Use filter function!
    - create Arquero friendly table -
    - filter!
    - What does this mean?
    - For every row in the cars_table check if the uppercase value of the column 'Origin' includes 'USA' . If yes, show that row in the output.

```
cars_table = aq.from(cars)
```

```
cars_table
  .filter(d => op.includes(op.upper(d.Origin), 'USA'))
  .view()
```

  - Order by Miles per gallon in the descending order
  - Choose the top rows - 5? 10?

```
cars_table
  .filter(d => op.includes(op.upper(d.Origin), 'USA'))
  .orderby(aq.desc("Miles_per_Gallon"))
  .view(5)
```

Time to practice!!!

👉Practice

Now it is your turn to wrangle the population dataset. How will you transform the population dataset to answer your question? Are the transformation operations different from how we transformed the cars dataset?

Sample question

Males of age 5 in 1850?

year	age	sex	people
1850	5	1	1411067

```
// Add your transformations here!
population_table
.filter(d => d.year == 1850)
.filter(d => d.age == 5)
.filter(d => d.sex == 1)
.view()
```

Converting pivoted population table into tidy format using 'fold' in arquero

1. The first part of the command is the set of columns to look at (columns 1-15 correspond to the years 1850-2000)

2. Second command specifies the naming convention for the resulting columns.

3. We turn each matrix cell into a new row where the "key" (column name) is saved in the "year" column and the "value" (cell) is saved in the "population" column.

age	1850	1860	1870	1880	1900	1910	1920	1930	1
0	2934165	4213008	5517185	6955259	9208740	10584300	11629036	11537780	10419
5	2770735	3583239	4822149	6476645	8856266	9857942	11482968	12672153	10827
10	2476213	3152990	4770011	5725474	8059418	9122634	10694213	12051349	11828
15	2187752	2934093	4035638	4985552	7576589	9158423	9504301	11479039	12358
20	2021122	2721583	3714685	5060583	7445099	9085315	9292043	10871374	11578
25	1662029	2333988	3083344	4093634	6625336	8203908	9094448	9980082	11102
30	1370274	1979229	2527163	3345879	5584138	6953345	8093197	8879621	10211
35	1093499	1604046	2322826	3024489	4981620	6516508	7795353	9170307	9516
40	904096	1316260	1913262	2476782	4266057	5331496	6381680	7829161	8831
45	725465	1014143	1600588	2069684	3516438	4478731	5812980	7029710	8240
50	607923	863481	1374188	1827496	2983019	3900108	4746526	5937299	7274

age	year	population
0	1850	2934165
0	1860	4213008
0	1870	5517185
0	1880	6955259
0	1900	9208740
0	1910	10584300
0	1920	11629036
0	1930	11537780
0	1940	10419281
0	1950	16074073

```
{
  let popPivot = aq.fromCSV(await FileAttachment('population.csv').text())
  return popPivot
    .fold(aq.range(1,15), {as:['year', 'population']})
    .view()
}
```