Section 1

Data Wrangling

CSE 412: Data Visualization
Today we will learn about...

- What is Data Wrangling?
- Observable
  - What is observable?
  - What kind of text formats it supports
  - Keyboard shortcuts and getting used to observable
  - Cell Values
- Data formats
  - Wide/ Tidy
  - Narrow/ Long
  - Cross tabulation
- Data Transformations + Transformation examples
  - Filtering
  - Slicing
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 of
  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.
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.

```plaintext
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

<table>
<thead>
<tr>
<th>Person</th>
<th>Age</th>
<th>Weight</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>32</td>
<td>168</td>
<td>180</td>
</tr>
<tr>
<td>Alice</td>
<td>24</td>
<td>150</td>
<td>175</td>
</tr>
<tr>
<td>Steve</td>
<td>64</td>
<td>144</td>
<td>165</td>
</tr>
</tbody>
</table>
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!!!
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 !!!

<table>
<thead>
<tr>
<th>Age</th>
<th>1850</th>
<th>1860</th>
<th>1870</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2934165</td>
<td>4213008</td>
<td>5517185</td>
</tr>
<tr>
<td>5</td>
<td>2770735</td>
<td>3583239</td>
<td>4822149</td>
</tr>
<tr>
<td>10</td>
<td>2476213</td>
<td>3152990</td>
<td>4770011</td>
</tr>
</tbody>
</table>
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
    ```javascript
    import {vl} from @vega/vega-lite-api
    ```
  - Arquero
    ```javascript
    import {aq, op} from @uwdata/arquero
    ```
  - Printable
    ```javascript
    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

```javascript
const cars = (await require('vega-datasets@1'))['cars.json']()
printTable(cars.slice(0,10))
```

- **Population:**
  - Viewing the data (JSON format)
  - Viewing the tabular form and assigning the table value

```javascript
const population = (await require('vega-datasets@1'))['population.json']()
// viewof shows the table view, but assigns the table value
viewof population_table = ag.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?
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.

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

<table>
<thead>
<tr>
<th>year</th>
<th>age</th>
<th>sex</th>
<th>people</th>
</tr>
</thead>
<tbody>
<tr>
<td>1850</td>
<td>5</td>
<td>1</td>
<td>1411067</td>
</tr>
</tbody>
</table>

// 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.

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

1. Post on Ed!
2. Come to the office hours!
3. Email the course staff!
References:

2. https://observablehq.com/d/d69e24aa78dabd68