Lecture 07 Outline: Data Visualization

1. Data Visualization with pandas, matplotlib, and seaborn
2. Design Principles for Effective Visualization
3. Advanced Visualization

- Use `import` statements to include external libraries.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

    - *Think of libraries as the special moves in your coding arsenal—gotta import them to unleash their power!*

- **Working with DataFrames**
    - DataFrames are table-like data structures provided by pandas.

```
df = pd.read_csv('data.csv')
```

    - *Like Excel spreadsheets, but way cooler and code-friendly!*

- **Basic Plotting**
    - Call plotting functions from libraries to create visualizations.

```
plt.plot(x, y)
```

    - *Because a picture is worth a thousand rows of data!*

- **Displaying Plots**

Introduction to Data Visualization Tools

- **pandas**
  - High-level data manipulation library.
  - Built-in plotting methods that simplify creating visualizations from DataFrames.
  - *Pandas: Not just adorable bears, but also your data's best friend!*
- **Matplotlib**
  - Foundation library for data visualization in Python.
  - Offers detailed control over plots.
  - *Think of it as the "Swiss Army Knife" of plotting—versatile and handy!*
- **Seaborn**
  - Statistical data visualization library.
  - Provides attractive default styles and color palettes.
  - *Because your data deserves to look as fabulous as it feels!*

Matplotlib: The Foundation

Introduction

- **Matplotlib** is the most widely used library for plotting in Python.
  - *It's the OG of Python plotting—setting the stage since the early days!*
- Think of it as the core building block for most other Python visualization libraries.
  - *Other libraries stand on the shoulders of this giant!*

Key Concepts in Matplotlib

- **Figures and Axes**
  - **Figure**: The overall container for all plot elements.
    - *The canvas where your masterpiece comes to life!*
  - **Axes**: The area where data is plotted (can be thought of as individual plots).
    - *The stage where your data takes center spotlight!*

Important Methods in Matplotlib

1. `plot()`: Basic Line Plot

- **Explanation**: Creates a simple line plot connecting data points.
- **Structure**:

```
plt.plot(x, y, marker, linestyle, color, label)
```

  - **Required Arguments**:
    - `x`: Data for the x-axis.
    - `y`: Data for the y-axis.
  - **Optional Arguments**:
    - `marker`: Style of the data point markers (e.g., `'o'` for circles).
    - `linestyle`: Style of the line connecting data points (e.g., `'-'` for solid line).
    - `color`: Color of the line.
    - `label`: Label for the legend.

Code Example: Line Plot

```
import matplotlib.pyplot as plt

# Data
years = [2010, 2012, 2014, 2016, 2018, 2020]
patients = [150, 180, 200, 230, 260, 300]

# Create line plot
plt.plot(years, patients, marker='o', linestyle='-', color='b', label='Patients')

# Add labels and title
plt.xlabel('Year')
plt.ylabel('Number of Patients')
plt.title('Number of Patients Over Years')

# Add legend
plt.legend()

# Show plot
plt.show()
```

- **Defining Data:**

```
years = [2010, 2012, 2014, 2016, 2018, 2020]
patients = [150, 180, 200, 230, 260, 300]
```

  - *Years and patient counts—the dynamic duo of our plot!*

- **Creating the Plot:**

```
plt.plot(years, patients, marker='o', linestyle='-', color='b', label='Patients')
```

  - *Connecting the dots like a constellation in the data sky!*

- **Adding Labels and Title:**

```
plt.xlabel('Year')
plt.ylabel('Number of Patients')
plt.title('Number of Patients Over Years')
```

  - *Because every hero (plot) needs an introduction!*

- **Adding a Legend:**

```
plt.legend()
```

Output Example

An image showing a line plot of patients over years with labeled axes and title.

#FIXME-{{Insert the actual line plot image showing the trend of patient numbers over the years}}

Common Issues and Troubleshooting

- **No Plot Displayed**:
  - Ensure `plt.show()` is called after plotting commands.
    - *Even the best actors need the curtain to rise—don't forget `plt.show()`!*
- **Data Length Mismatch**:
  - Verify that `x` and `y` are of equal length.
    - *Mismatch is great in a rom-com, but not in data plotting!*
- **Import Errors**:
  - Install Matplotlib using `pip install matplotlib` if it's not installed.
    - *Because missing imports are the coding equivalent of forgetting your keys!*

2. `scatter()`: Scatter Plot

- **Explanation**: Creates a scatter plot of x vs. y, useful for showing relationships between variables.
- **Structure**:

```
plt.scatter(x, y, s, c, alpha)
```

  - **Required Arguments**:
    - `x`: Data for the x-axis.
    - `y`: Data for the y-axis.
  - **Optional Arguments**:
    - `s`: Size of markers.
    - `c`: Color of markers.
    - `alpha`: Transparency level of markers.

Code Example: Scatter Plot

```
import matplotlib.pyplot as plt

# Data
age = [25, 35, 45, 20, 30, 40, 50, 60]
blood_pressure = [120, 130, 125, 115, 135, 140, 150, 145]

# Create scatter plot
plt.scatter(age, blood_pressure, c='red', alpha=0.7)

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Blood Pressure')
plt.title('Blood Pressure vs Age')

# Show plot
plt.show()
```

Code Explanation

- **Defining Data**:
  - `age` : List of ages.
  - `blood_pressure` : Corresponding blood pressure readings.
  - *Matching ages with blood pressures like pairing fine wine with cheese!*
- **Creating the Plot**:

```
plt.scatter(age, blood_pressure, c='red', alpha=0.7)
```

  - *Plotting points like throwing darts at the board—aiming for insights!*
- **Adding Labels and Title**:
  - Labels the axes and sets the title.
  - *Guiding the audience through your data journey!*
- **Displaying the Plot**:
  - Uses `plt.show()` to render the plot.

Output Example

An image showing a scatter plot of blood pressure vs age.

#FIXME-{{Include the actual scatter plot image of blood pressure versus age}}

Additional Plot Types in Matplotlib

- **Bar Plot**:

```
plt.bar(categories, values)
```

  - *Great for categorical data—because sometimes you just need to raise the bar!*

- **Histogram**:

```
plt.hist(data, bins=10)
```

  - *Unveiling the distribution secrets of your data—think of it as data's fingerprint!*

- **Box Plot**:

```
plt.boxplot(data)
```

  - *Spotting outliers faster than you can say "Houston, we have a problem!"*

- **Pie Chart**:

```
plt.pie(sizes, labels=labels)
```

  - *When you want to show proportions and make your data slice-of-life interesting!*

pandas: Built-in Plotting

Introduction

- pandas provides convenient plotting methods directly on DataFrames and Series.
- Simplifies the creation of plots without explicitly using Matplotlib commands.
- *It's like having a fast-pass to plotting—skip the lines and get straight to the fun!*

Important Methods in pandas

1. `plot()`: Line Plot

- **Explanation**: Plots DataFrame or Series data as lines.
- **Structure**:

```
df.plot(x='column1', y='column2', kind='line', marker, title)
```

    - *Who says data can't be classy? Let pandas do the heavy lifting!*

Code Example: pandas Line Plot

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv('hospital_admissions.csv')

# Inspect data
print(df.head())
```

Sample Output:

```
   Year  Admissions
0  2016         500
1  2017         550
2  2018         600
3  2019         650
4  2020         700
```

Continuing the Code Example

```
# Plot data
df.plot(x='Year', y='Admissions', kind='line', marker='o', title='Yearly Hospital Admissions')

# Add labels
plt.xlabel('Year')
plt.ylabel('Number of Admissions')

# Show plot
plt.show()
```

Code Explanation

- **Loading Data:**

```
df = pd.read_csv('hospital_admissions.csv')
```

  - *Because data isn't going to read itself—time to bring it into the game!*

- **Inspecting Data:**

```
print(df.head())
```

  - *Always good to peek under the hood before hitting the road!*

- **Creating the Plot:**

```
df.plot(x='Year', y='Admissions', kind='line', marker='o', title='Yearly Hospital Admissions')
```

  - *One line of code to plot—all the cool kids are doing it!*

- **Adding Labels:**
  - Sets x and y-axis labels.
- **Displaying the Plot:**
  - Shows the plot using `plt.show()`.

Output Example

An image showing a line plot of yearly hospital admissions.

#FIXME-{{Include the actual line plot image of hospital admissions over the years}}

Common Issues and Troubleshooting

- **Column Not Found**:
  - Ensure column names in `x` and `y` match exactly with the DataFrame.
    - *Check your spelling—autocorrect won't save you here!*
- **No Plot Displayed**:
  - If plots don't display in Jupyter notebooks, use `%matplotlib inline` at the beginning.
    - *A little magic command to make plots appear—abracadabra!*

2. `hist()`: Histogram

- **Explanation**: Plots a histogram of a single column or series.
- *Unleash your inner statistician and dive into the distribution!*

Code Example: pandas Histogram

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv('patient_ages.csv')

# Plot histogram
df['Age'].hist(bins=10, color='skyblue', alpha=0.7)

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Number of Patients')
plt.title('Age Distribution of Patients')

# Show plot
plt.show()
```

Code Explanation

- **Loading Data**:
  - Reads patient age data into a DataFrame.
- **Creating the Histogram**:

```
df['Age'].hist(bins=10, color='skyblue', alpha=0.7)
```

  - *Plotting that data like a boss—with colors and style!*
- **Adding Labels and Title**:
  - Provides context to your masterpiece.
- **Displaying the Plot**:
  - Renders the histogram.

Output Example

An image showing a histogram of patient age distribution.

#FIXME-{{Include the actual histogram image of patient age distribution}}

Seaborn: Statistical Data Visualization

Introduction

- **Seaborn** enhances Matplotlib's functionality by providing high-level interfaces.
- Ideal for statistical plots and works well with pandas DataFrames.
- *It's like upgrading from a bicycle to a sports car—same principles, more speed and style!*

Important Methods in Seaborn

1. `scatterplot()`: Scatter Plot

- **Explanation**: Creates enhanced scatter plots with additional functionalities.
- **Structure**:

```
sns.scatterplot(x='x_col', y='y_col', data=df, hue, size, style)
```

    - *Bring your scatter plots to life with colors and styles—because who wants a boring plot?*

Code Example: Seaborn Scatter Plot

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load dataset
df = pd.read_csv('health_data.csv')

# Create scatter plot
sns.scatterplot(x='BMI', y='BloodPressure', hue='AgeGroup', data=df)

# Add title
plt.title('Blood Pressure vs BMI by Age Group')

# Show plot
plt.show()
```

Code Explanation

- **Importing Libraries**:
  - Imports seaborn, matplotlib, and pandas.
- **Loading Data**:

```
df = pd.read_csv('health_data.csv')
```

  - *Data in, insights out—that's the way we roll!*
- **Creating the Plot**:

```
sns.scatterplot(x='BMI', y='BloodPressure', hue='AgeGroup', data=df)
```

  - *Adding a splash of color—because life is too short for monochrome plots!*
- **Adding Title**:
  - Sets the title of the plot.
- **Displaying the Plot**:
  - Renders the scatter plot.

Output Example

An image showing a scatter plot of Blood Pressure vs BMI colored by Class.

#FIXME-{{Include the actual Seaborn scatter plot image with hue based on Age Group}}

Interpreting the Plot

- **Color Coding**:
  - Different colors represent different age groups.
    - *Like sorting candies by flavor—visually satisfying and informative!*
- **Trend Analysis**:
  - Helps identify how BMI relates to Blood Pressure across age groups.
    - *Finding patterns faster than a detective on a TV show!*

Additional Plot Types in Seaborn

- **Histogram and KDE Plot:**

```
sns.histplot(data=df, x='BMI', kde=True)
```

  - *Because sometimes you need more curves than a roller coaster!*

- **Box Plot:**

```
sns.boxplot(x='AgeGroup', y='Cholesterol', data=df)
```

  - *Unboxing the secrets of your data—no subscription required!*

- **Heatmap:**

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

  - *Visualizing correlations hotter than the latest trends!*

- **FacetGrid:**

```
g = sns.FacetGrid(df, col='Gender')
g.map(plt.hist, 'BMI')
```

Common Issues and Troubleshooting

- **Seaborn Version Compatibility**:
  - Ensure you have the latest version installed using `pip install seaborn --upgrade`.
    - *Because outdated libraries are so last season!*
- **Attribute Errors**:
  - Verify function names and parameters, as they may differ between versions.
    - *Double-check before you wreck—your code, that is!*

Tips for Effective Visualization

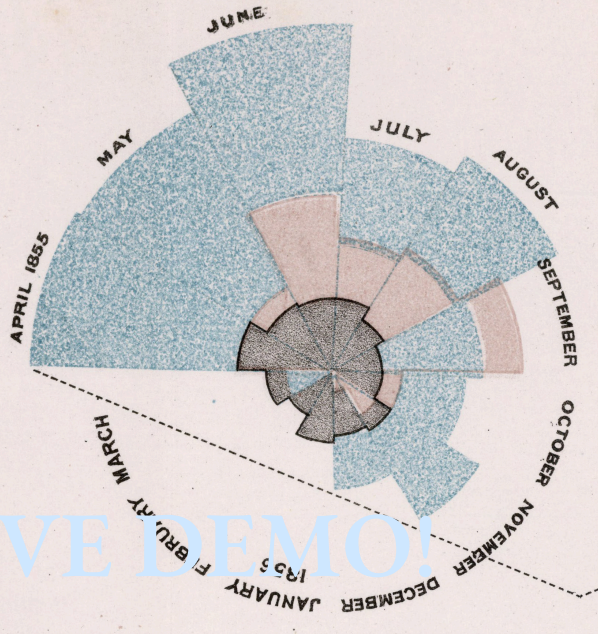
- **Consistency**:
  - Use consistent styles and color schemes throughout your visualizations.
    - *Because even data likes to coordinate its outfit!*
- **Clarity**:
  - Label axes, include units of measurement, and provide legends when necessary.
    - *Don't make your audience play "Guess Who" with your plots!*
- **Simplicity**:
  - Avoid unnecessary decorations that don't add informational value.
    - *Remember, sometimes less is more—unless we're talking about tacos!*
- **Interpretation**:
  - Always consider how the audience will interpret your plots.
    - *Aim for "Aha!" moments, not head-scratching confusion!*

Resources for Further Learning

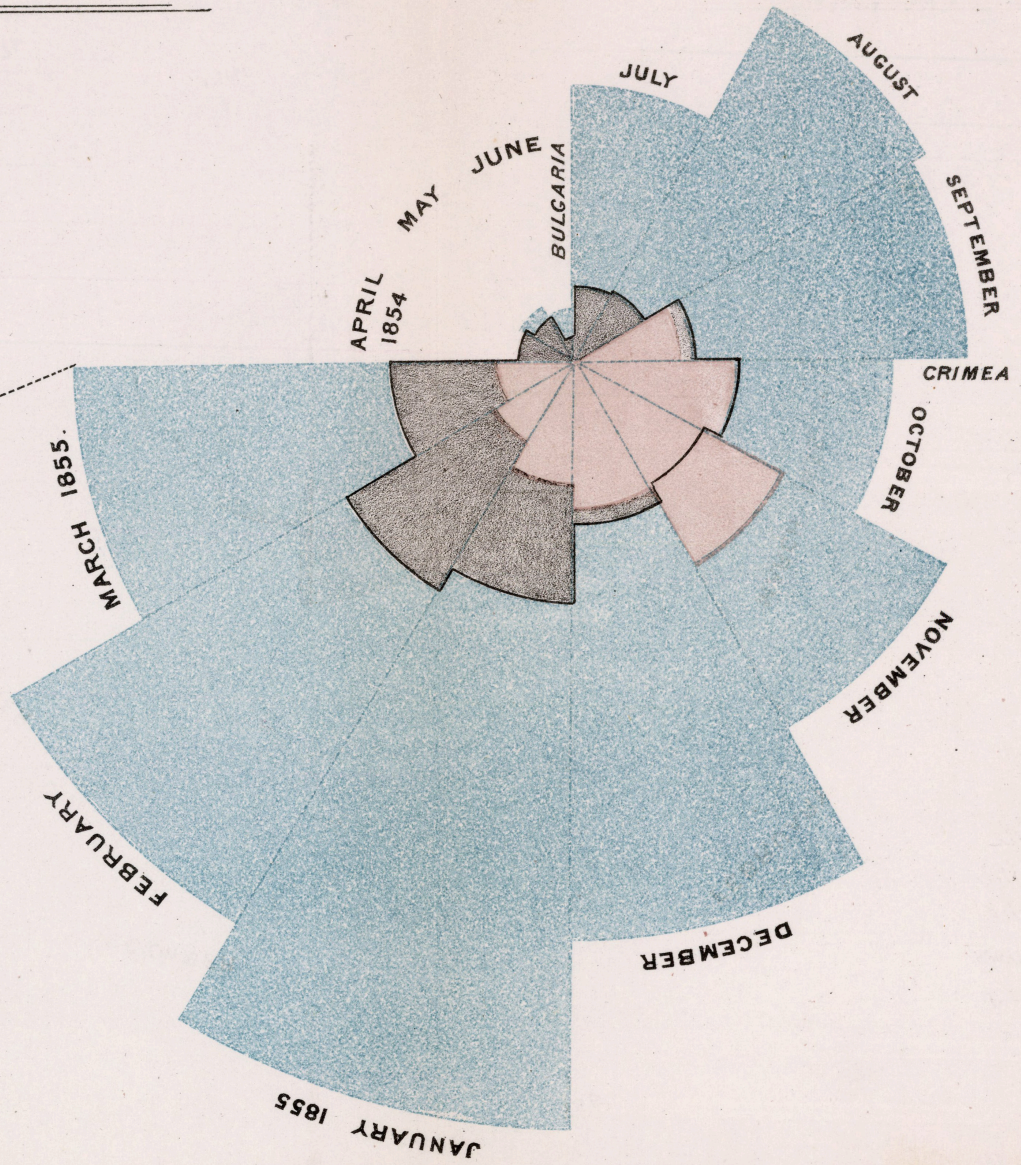
- **Matplotlib Documentation**: matplotlib.org
- **pandas Visualization Guide**: pandas.pydata.org
- **Seaborn Tutorials**: seaborn.pydata.org
- **Troubleshooting Tips**: Search for errors on Stack Overflow or consult the official documentation.

DIAGRAM OF THE CAUSES OF MORTALITY IN THE ARMY IN THE EAST.

2.
APRIL 1855 TO MARCH 1856.



1.
APRIL 1854 TO MARCH 1855.



The Areas of the blue, red, & black wedges are each measured from the centre as the common vertex.

The blue wedges measured from the centre of the circle represent area for area the deaths from Preventible or Mitigable Zymotic diseases; the red wedges measured from the centre the deaths from wounds; & the black wedges measured from the centre the deaths from all other causes.

The black line across the red triangle in Nov.r 1854 marks the boundary of the deaths from all other causes during the month.

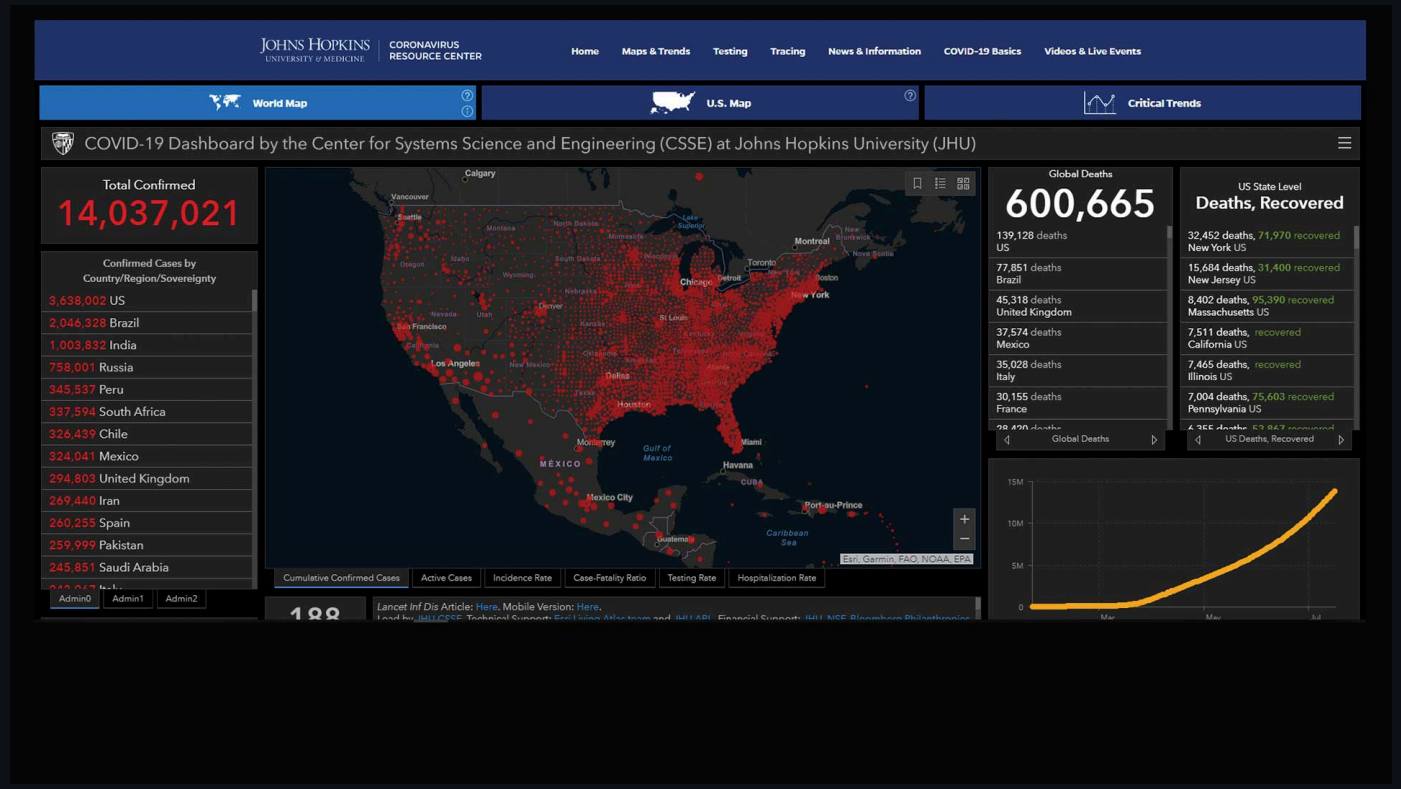
In October 1854, & April 1855, the black area coincides with the red; in January & February 1855, the blue coincides with the black.

The entire areas may be compared by following the blue, the red & the black lines enclosing them.

Harrison & Sons, St. Martin's Lane.

Florence Nightingale: The Rose Diagram

- **Coxcomb Chart (Rose Diagram)**
  - Visualized causes of mortality during the Crimean War.
  - Highlighted the impact of poor sanitary conditions.
  - *Proving that data visualization isn't just for the birds—even when it's called a "coxcomb"!*
- **Impact**
  - Influenced medical reform and sanitary practices.
  - Early use of data visualization to drive policy change.
  - *Nightingale: Saving lives with stats before it was cool.*



deaths.

- *When the world pressed pause, this dashboard played on.*

- **Effective Use of Maps and Time Series**
  - Interactive and continuously updated.
  - *Bringing pandemic data to the masses faster than toilet paper flew off shelves!*

- **Global Impact**
  - Became a crucial

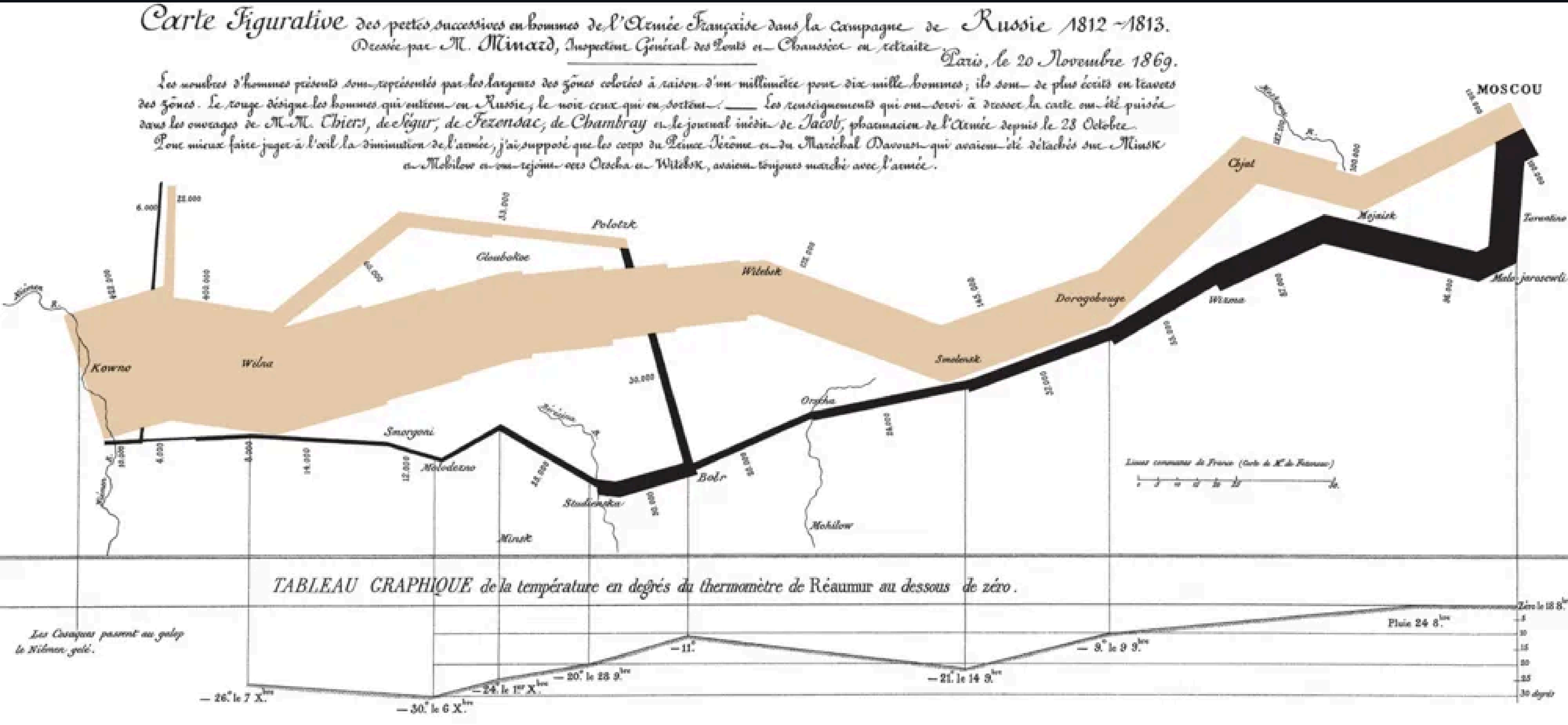
Introduction to Design Principles

- **Importance of Good Design**
  - Enhances comprehension and retention
  - Communicates data accurately and ethically
  - *Because nobody wants their data presentation to be the next plot twist nobody asked for!*
- **Based on Works of Edward Tufte and Claus O. Wilke**
  - Focus on clarity, precision, and efficiency in data presentation
  - *Think of them as the Jedi Masters of data visualization!*

Key Concepts

Simplify and Focus

- **Eliminate Non-Essential Elements**
  - Remove unnecessary gridlines, backgrounds, and decorations
  - *Less is more—channel your inner minimalist!*
- **Highlight Key Data**
  - Use visual emphasis (bolding, color) to draw attention to important information
  - *Make your data the Beyoncé of the visualization—let it steal the spotlight!*


Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite
Paris, le 20 Novembre 1869.
Les nombres d'hommes présents sont représentés par les largeurs des zônes colorées à raison d'un millimètre pour dix mille hommes; ils sont de plus écrits en travers des zônes. Le rouge désigne les hommes qui entrent en Russie, le noir ceux qui en sortent. Les renseignements qui ont servi à dresser la carte ont été puisés dans les ouvrages de M.M. Thiers, de Ségur, de Fezensac, de Chambray et le journal inédit de Jacob, pharmacien de l'Armée depuis le 28 Octobre.
Pour mieux faire juger à l'œil la diminution de l'armée, j'ai supposé que les corps du Prince Jérôme et du Maréchal Davoust qui avaient été détachés sur Minsk et Mohilow et ont rejoint vers Orscha et Witebsk, avaient toujours marché avec l'armée.
MOSCOU
Chjat
Mojaisk
Tarantino
Malo-jarosewli
Wizma
Dorogobouge
Smolensk
Orscha
Witebsk
Polotzk
Gloubokoe
Wilna
Kowno
Smorgoni
Molodezno
Studienska
Bobr
Minsk
Mohilow
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.
Les Cosaques passent au galop le Niémen gelé.
Pluie 24 8.bre
Zéro le 18 8.bre
— 9.° le 9 9.bre
— 11.°
— 21.° le 14 9.bre
— 20.° le 28 9.bre
— 24.° le 1.er X.bre
— 30.° le 6 X.bre
— 26.° le 7 X.bre

- **Data-Ink Ratio**
  - Maximizing the proportion of ink used to present data.
  - *Less is more—think of it as the "Marie Kondo" approach to data!*
- **Chartjunk**
  - Eliminating unnecessary decorative elements.
  - *Say no to chartjunk—because nobody needs a pie chart with 50 shades of gray!*
- **Clarity and Precision**
  - Advocated for minimalist design focused on the data.
  - *Data used to be shy; Tufte made it the star of the show!*
- **Notable Works**
  - *The Visual Display of Quantitative Information*
  - *Envisioning Information*

Definitions

Data-Ink Ratio

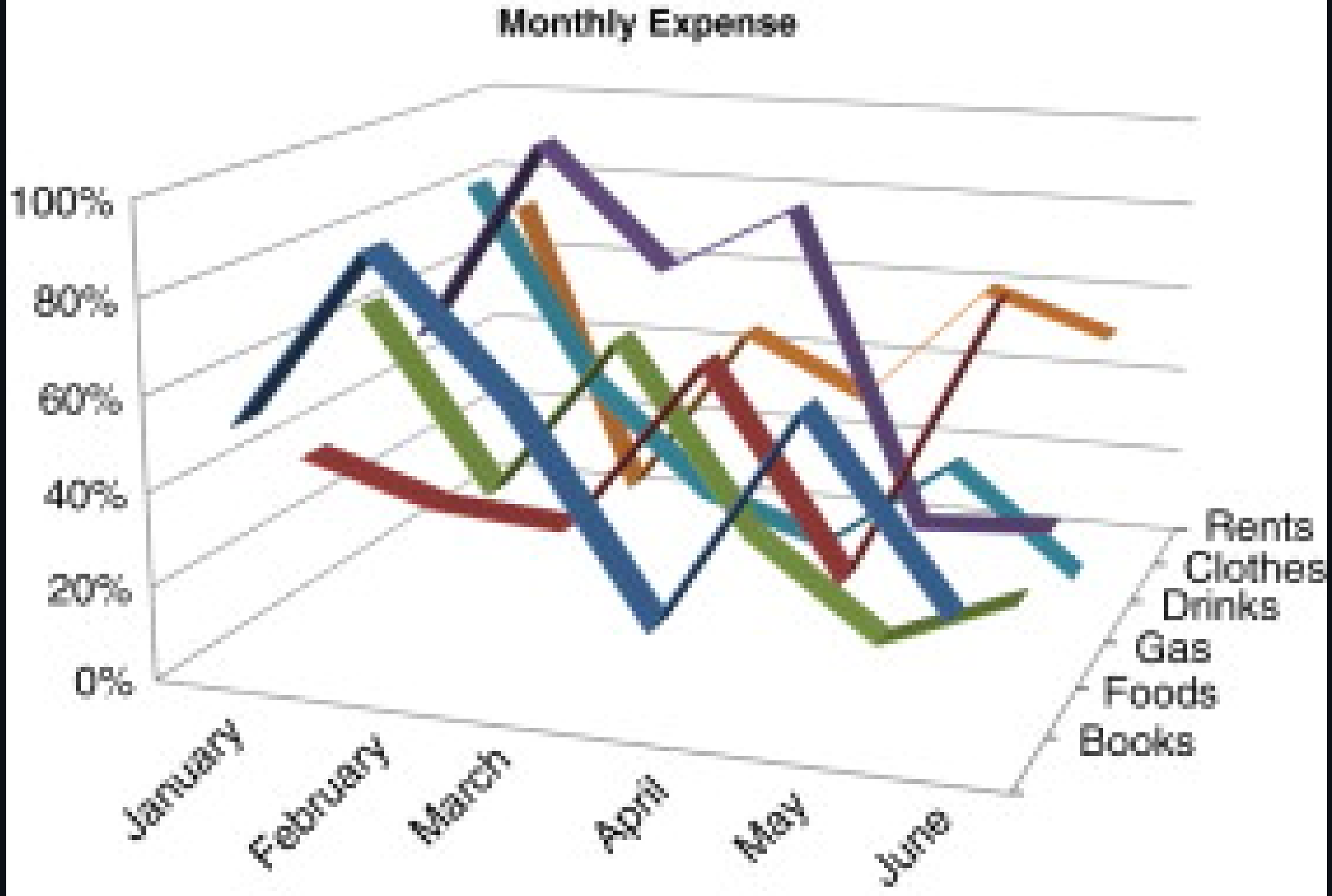
- **Definition**: The proportion of ink used to present actual data compared to the total ink used in the graphic
- **Goal**: Maximize data-ink ratio by reducing non-essential elements
- *In other words, don't let your chart wear a heavy coat in summer—keep it light!*

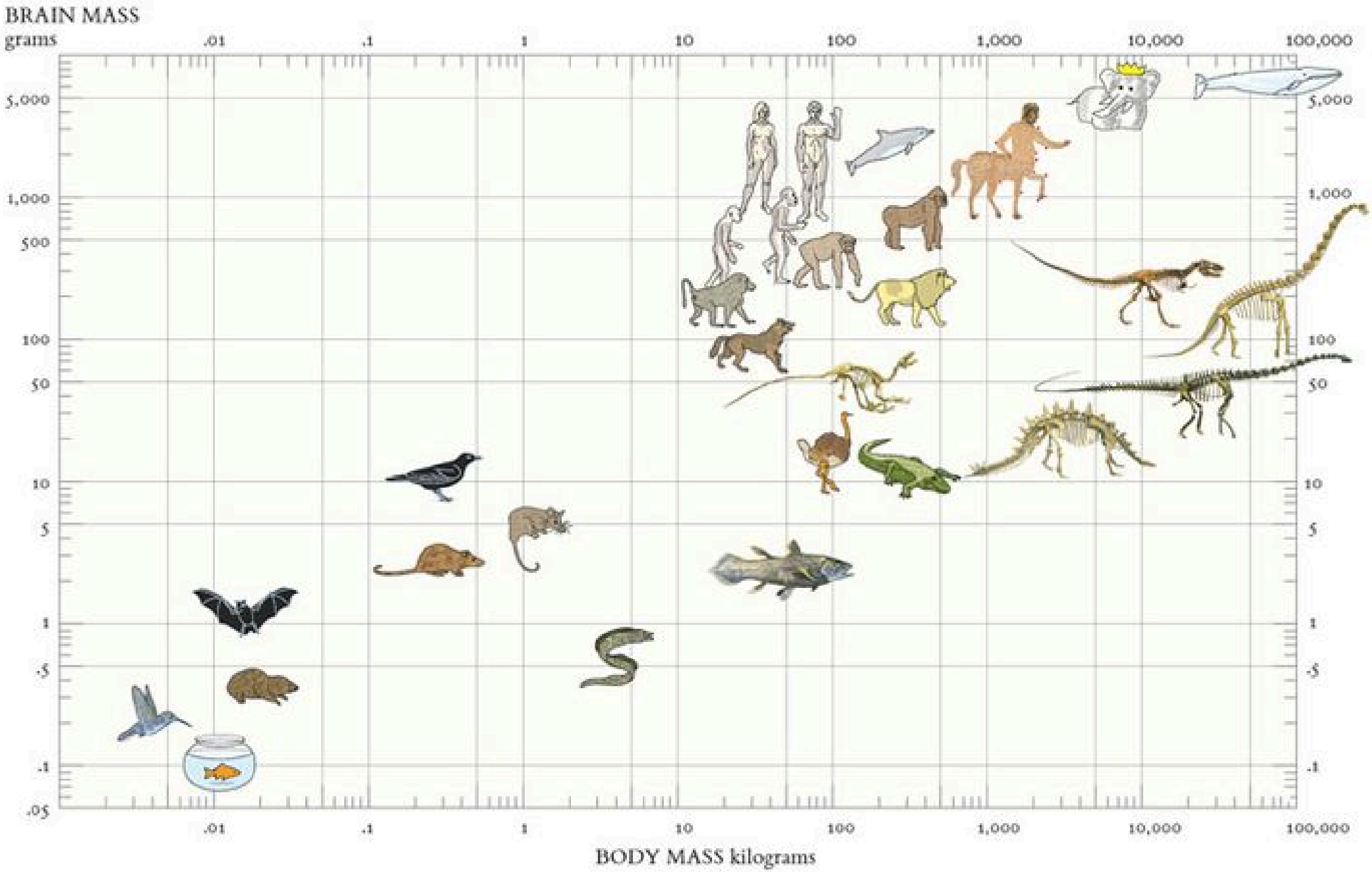
Chartjunk

- **Definition**: Unnecessary or distracting decorations in data visualizations that do not improve the viewer's understanding
- **Includes**:
  - Excessive colors or patterns
  - 3D effects that distort data
  - Decorative images or clip art
- *Chartjunk isn't adding pineapple to pizza, it's like adding poptarts—it might seem like a good idea to some, but it often just complicates things!*

Bad Example

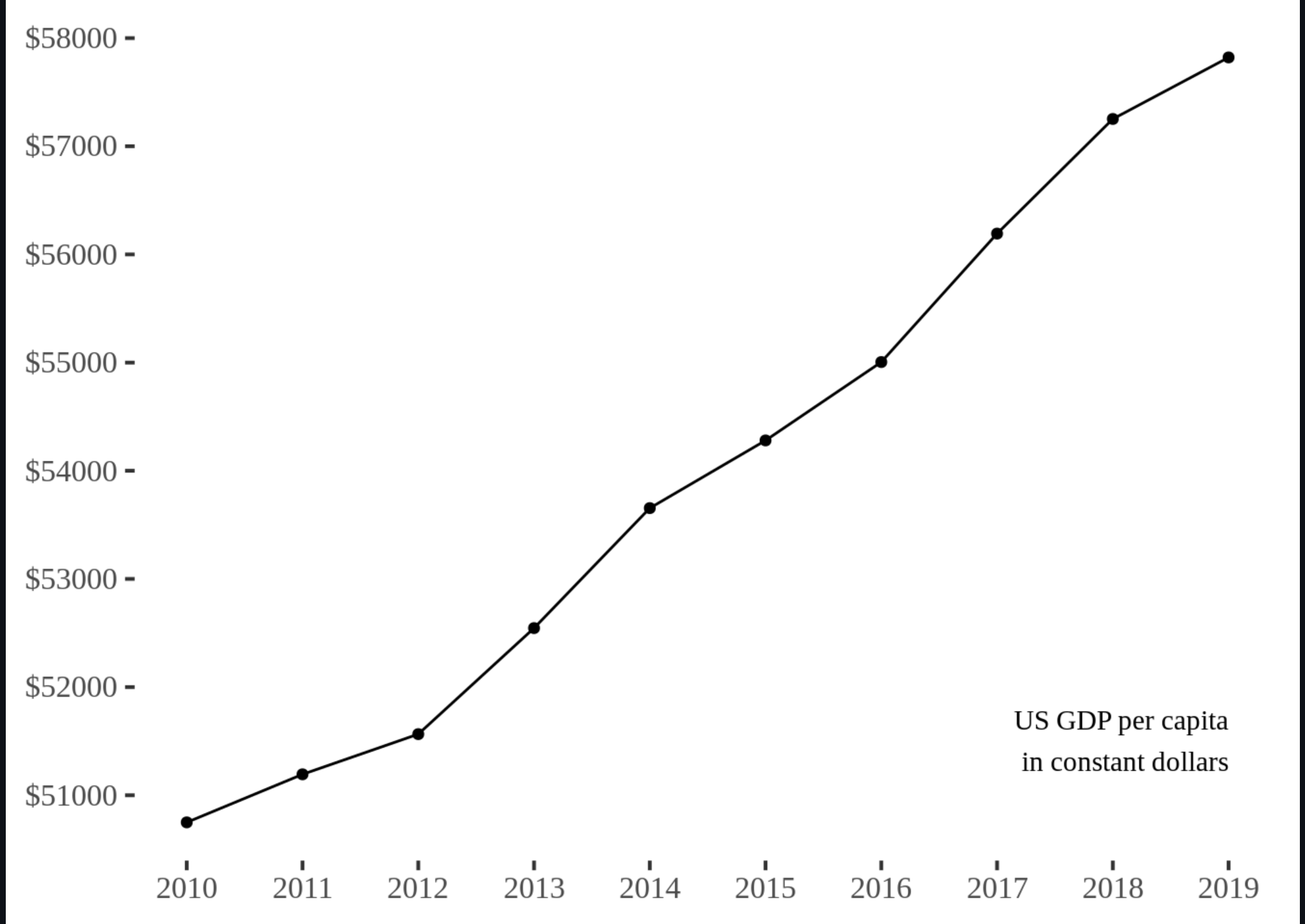
- **Issues:**
  - Distracting colors
  - Misleading scales
  - Unnecessary 3D effects

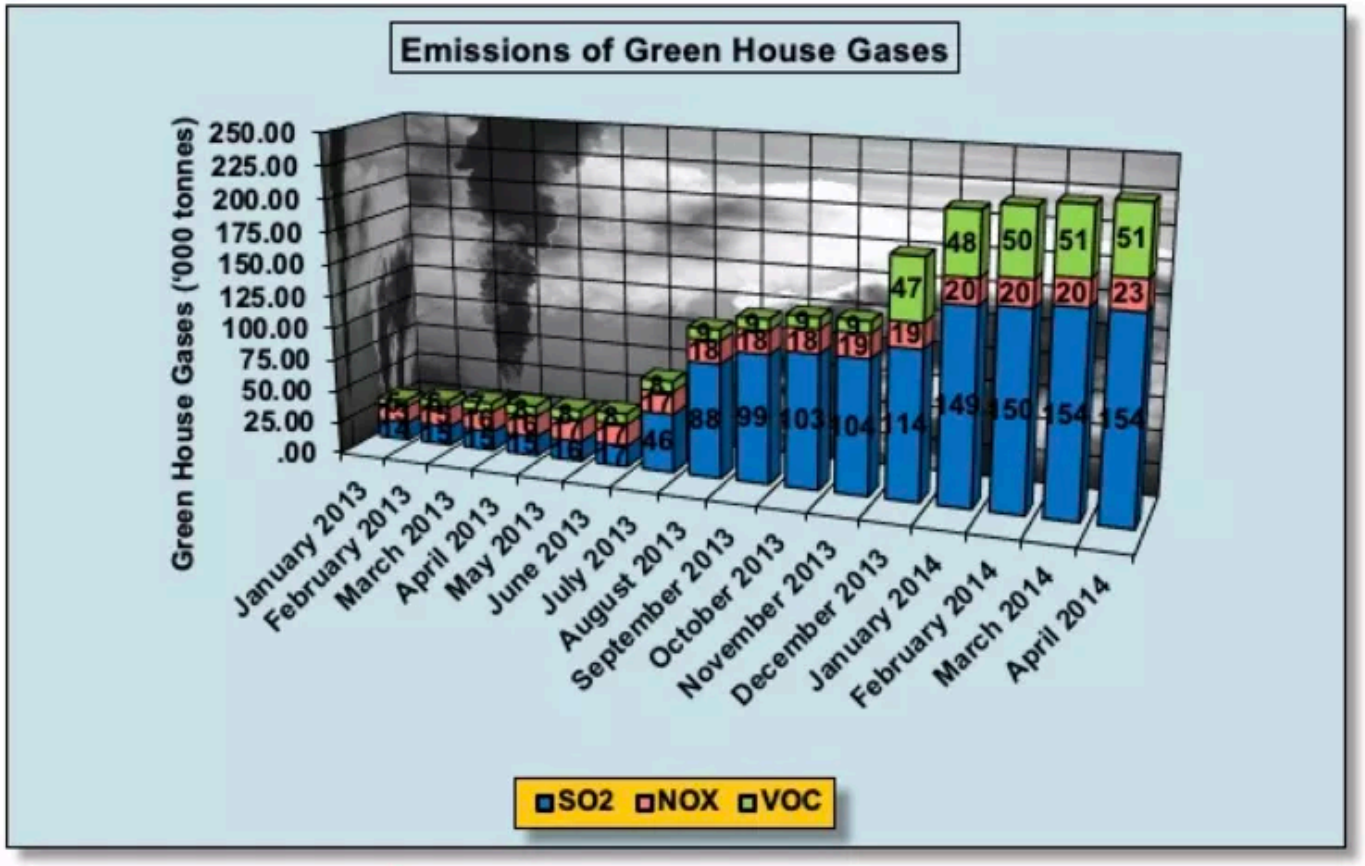

Monthly Expense
100%
80%
60%
40%
20%
0%
January
February
March
April
May
June
Rents
Clothes
Drinks
Gas
Foods
Books

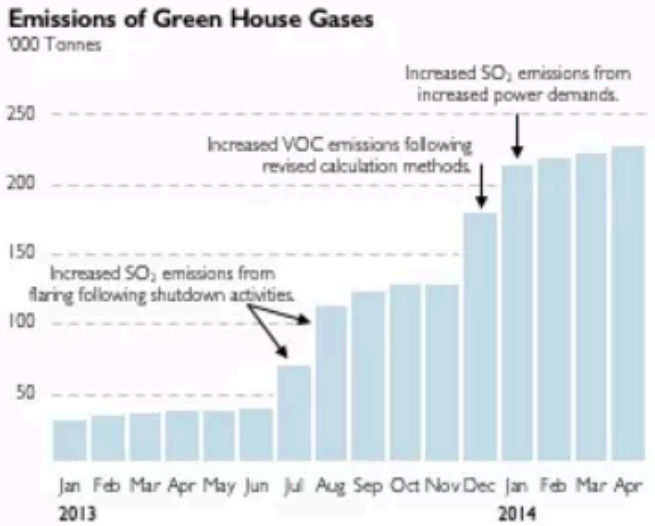

BRAIN MASS
grams
.01
.1
1
10
100
1,000
10,000
100,000
5,000
1,000
500
100
50
10
5
1
.5
.1
.05
BODY MASS kilograms

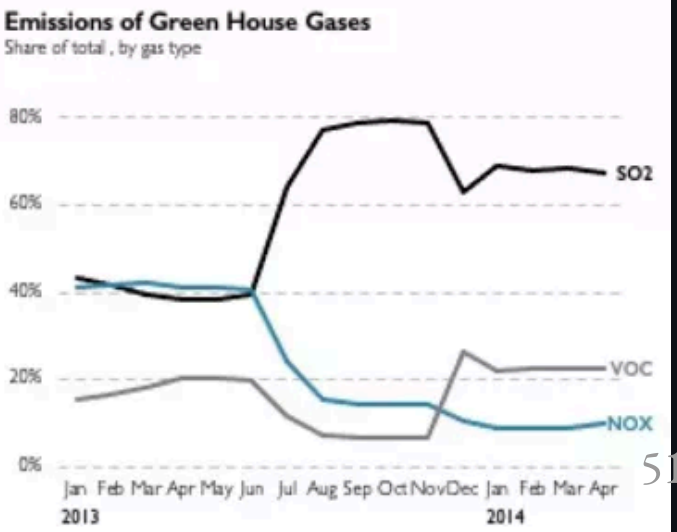
Good Example

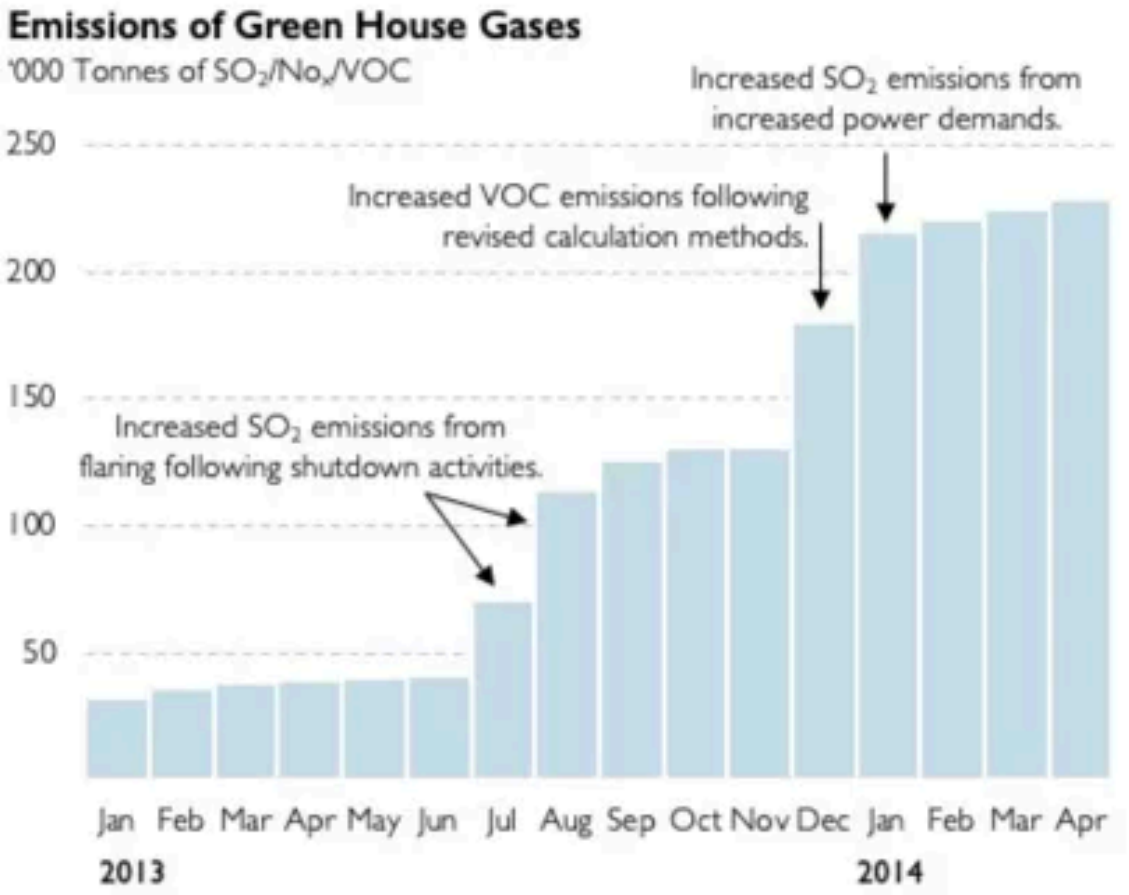
- **Features:**
  - Clear labels and titles
  - Minimalist design
  - Accurate representation of data

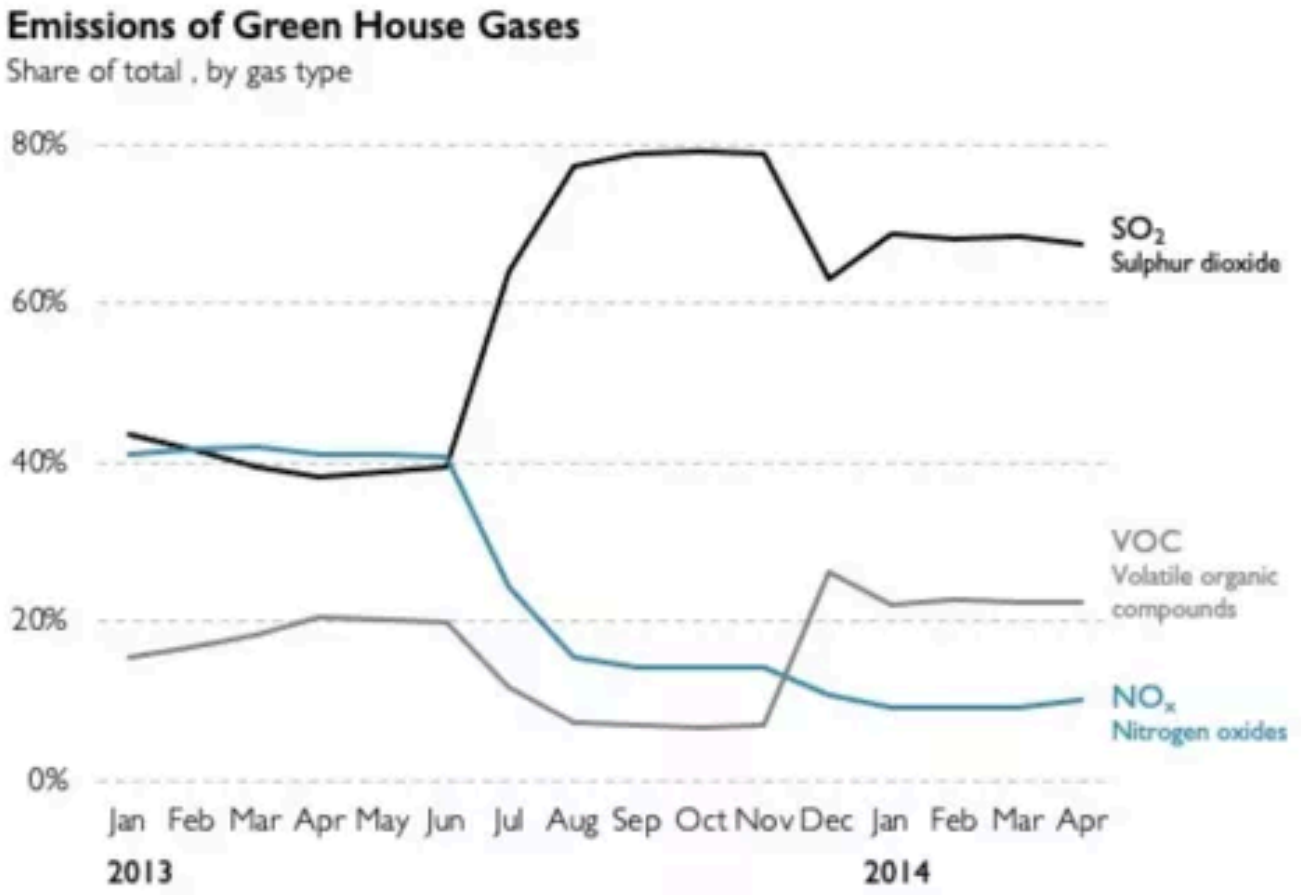

$58000
$57000
$56000
$55000
$54000
$53000
$52000
$51000
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
US GDP per capita
in constant dollars

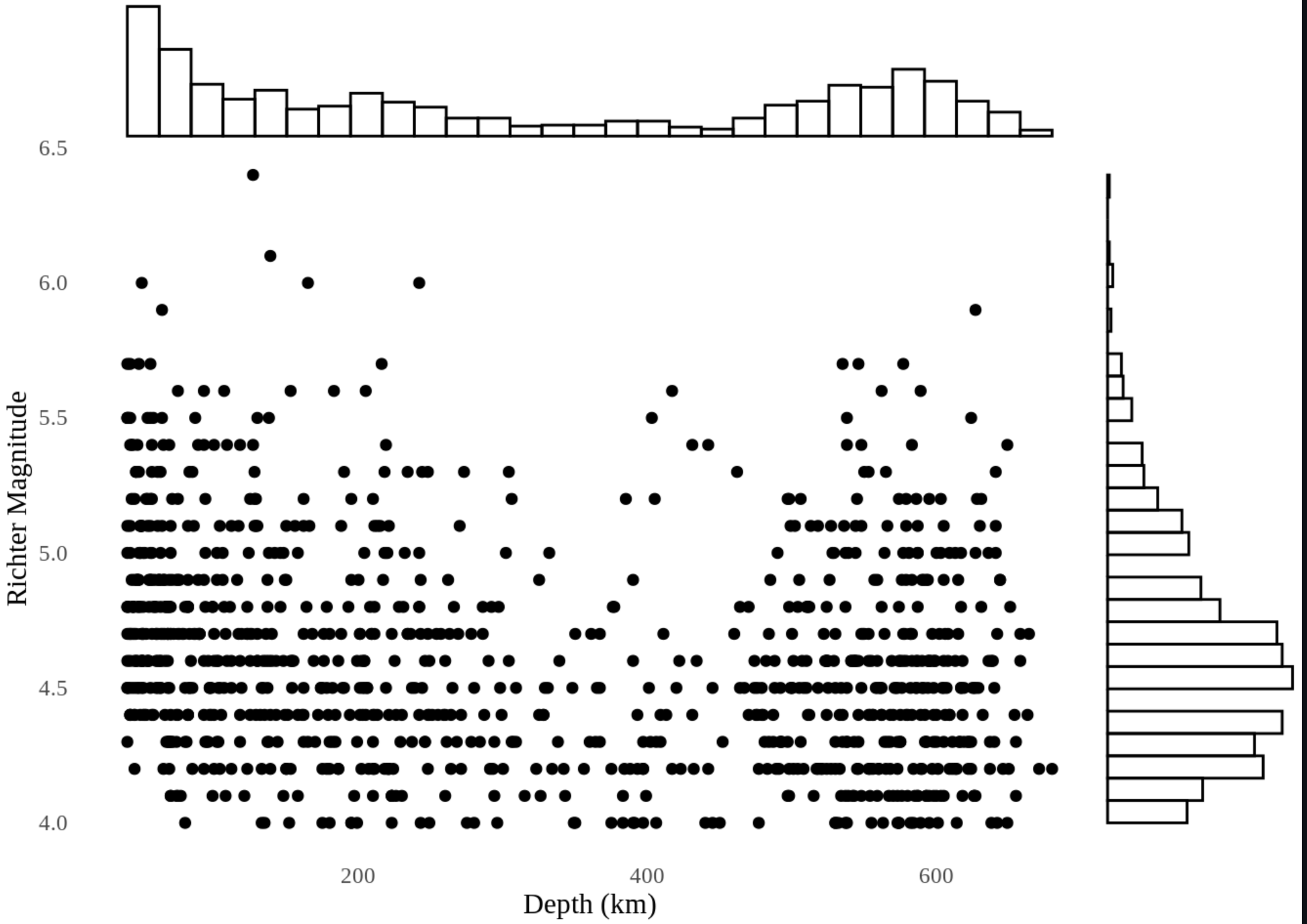

Emissions of Green House Gases
Green House Gases ('000 tonnes)
250.00
225.00
200.00
175.00
150.00
125.00
100.00
75.00
50.00
25.00
.00
January 2013
February 2013
March 2013
April 2013
May 2013
June 2013
July 2013
August 2013
September 2013
October 2013
November 2013
December 2013
January 2014
February 2014
March 2014
April 2014
48
50
51
51
47
20
20
20
23
19
18
18
19
18
18
46
88
99
103
104
114
149
150
154
154
SO2
NOX
VOC

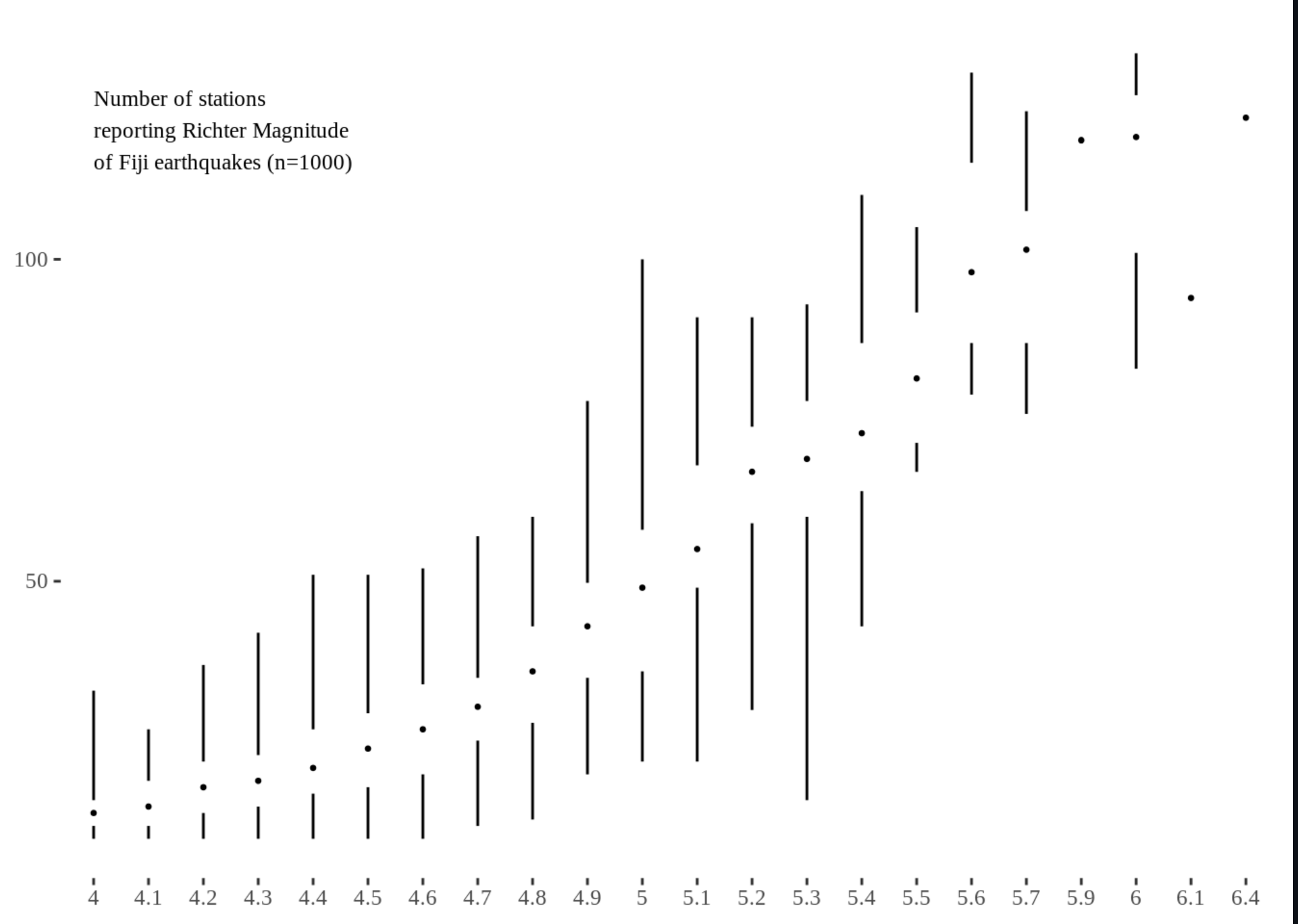

Emissions of Green House Gases
'000 Tonnes
Increased SO_2 emissions from increased power demands.
Increased VOC emissions following revised calculation methods.
Increased SO_2 emissions from flaring following shutdown activities.
250
200
150
100
50
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr
2013
2014


Emissions of Green House Gases
Share of total , by gas type
80%
60%
40%
20%
0%
SO2
VOC
NOX
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr
2013
2014


Emissions of Green House Gases
'000 Tonnes of SO_2/No$_x$/VOC
Increased SO_2 emissions from increased power demands.
Increased VOC emissions following revised calculation methods.
Increased SO_2 emissions from flaring following shutdown activities.
250
200
150
100
50
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr
2013
2014


Emissions of Green House Gases
Share of total , by gas type
80%
60%
40%
20%
0%
SO_2
Sulphur dioxide
VOC
Volatile organic compounds
NO_x
Nitrogen oxides
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr
2013
2014


6.5
6.0
5.5
5.0
4.5
4.0
Richter Magnitude
200
400
600
Depth (km)


Number of stations
reporting Richter Magnitude
of Fiji earthquakes (n=1000)
100
50
4
4.1
4.2
4.3
4.4
4.5
4.6
4.7
4.8
4.9
5
5.1
5.2
5.3
5.4
5.5
5.6
5.7
5.9
6
6.1
6.4

Ethical Representation of Data (chartcrime)

- **Avoid Misleading Visuals**
  - Start axes at zero when appropriate to prevent exaggeration
    - *Unless you're trying to create the next big conspiracy theory, keep it real!*
  - Use consistent scales across related visuals
    - *Inconsistency is only cool in plot twists, not in plots!*
- **Accurate Data Representation**
  - Do not manipulate visuals to mislead or bias the audience
    - *No one likes a data manipulator—trust is key!*
  - Clearly indicate any data exclusions or manipulations
    - *Transparency isn't just for windows!*

Use of Color

- **Functional Use**
  - Differentiate data categories meaningfully
    - *Think Power Rangers—each color represents a different hero!*
  - Use color to highlight important data points
- **Accessibility**
  - Use colorblind-friendly palettes (e.g., Viridis, Cividis)
    - *Because everyone should be able to appreciate your data masterpiece!*
  - Ensure sufficient contrast between colors

Privacy and Confidentiality

- **Aggregate Data**
  - Use aggregated or de-identified data to protect patient privacy
    - *Doctor's code: First, do no harm—even in data!*
- **Geographic Detail**
  - Be cautious with maps; avoid pinpointing exact locations if not necessary
    - *We're not playing 'Where's Waldo' with patient data!*

Ethical Representation

- **Sensitive Topics**
  - Present data on sensitive health issues with care and respect
    - *Handle with care—like it's hot coffee!*
- **Cultural Sensitivity**

Applying Principles in Practice

- **Critical Evaluation**
  - Assess visualizations for simplicity and clarity
    - *Is your chart more complicated than the plot of 'Inception'? Simplify!*
- **Iterative Refinement**
  - Make incremental improvements based on feedback
    - *Remember, even Tony Stark upgraded his suit—improve iteratively!*
- **Audience Consideration**
  - Tailor visuals to the needs and expectations of the audience
    - *Know your audience—don't serve sushi at a pizza party!*

Tell a Story

- **Narrative Flow**
  - Guide the viewer through the data logically
    - *Be the Bob Ross of data—paint a happy little story, one insight at a time!*
- **Annotations and Highlights**
  - Use text and markers to emphasize key points
    - *Because sometimes even data needs a little spotlight!*
- **Provide Context**
  - Background information aids interpretation
    - *Set the scene before the action begins!*

Takeaways

- **Simplicity is Key**
  - Strive for clarity and avoid unnecessary complexity
    - *Don't turn your chart into a 'Where's Waldo' page!*
- **Ethics Matter**
  - Represent data honestly and responsibly
    - *With great data comes great responsibility!*
- **Design with Purpose**
  - Every element in your visualization should serve a function
    - *No more, no less—just like a perfectly balanced equation!*

Additional Resources

- **"The Visual Display of Quantitative Information"** by Edward Tufte
- **"Fundamentals of Data Visualization"** by Claus O. Wilke
- **"Tufte's Principles of Data-Ink"** Liu & Zhuang
- **Color Brewer 2**: colorbrewer2.org for choosing colorblind-friendly palettes
- *Because lifelong learning is the real infinite game!*

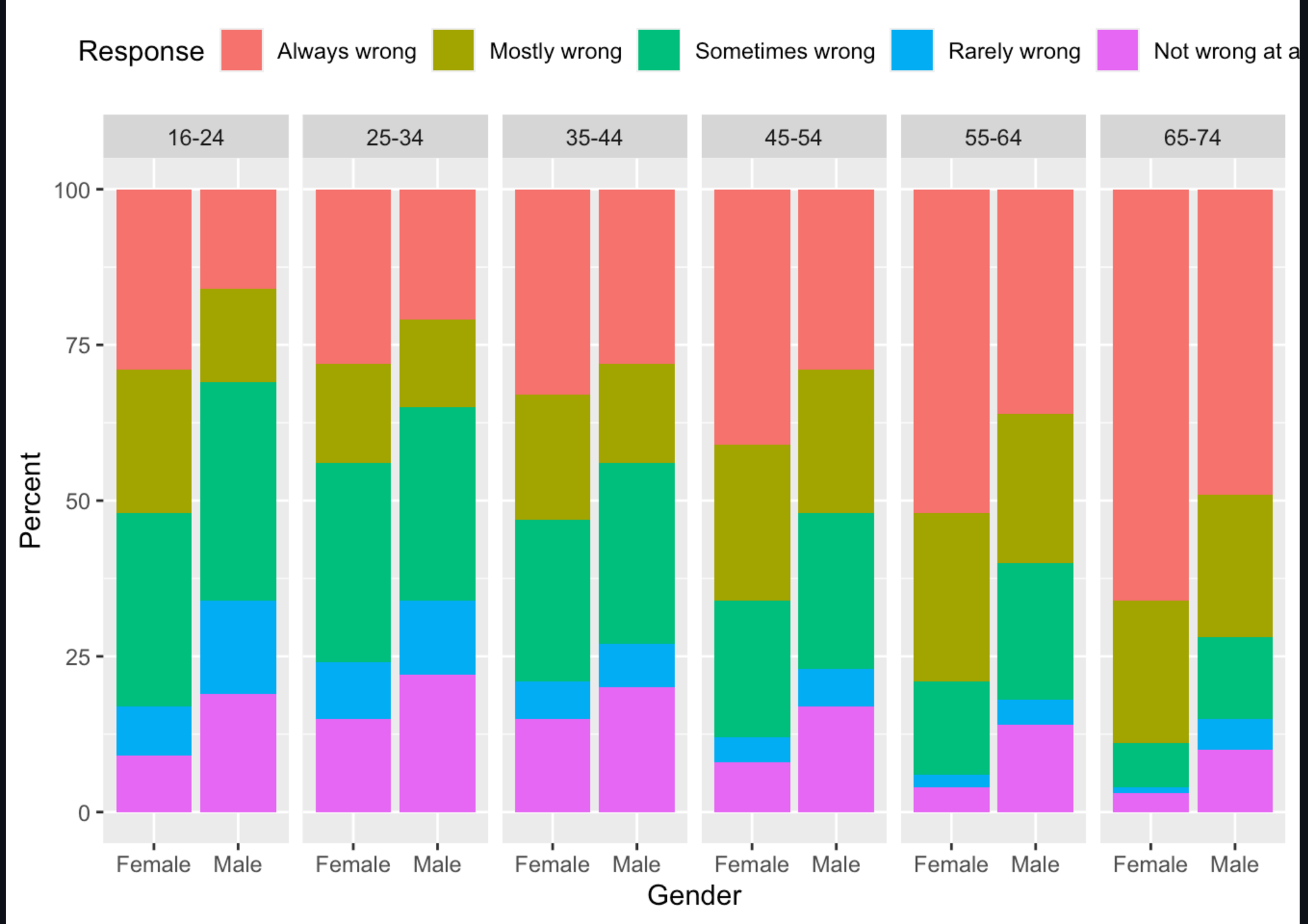
LIVE DEMO!

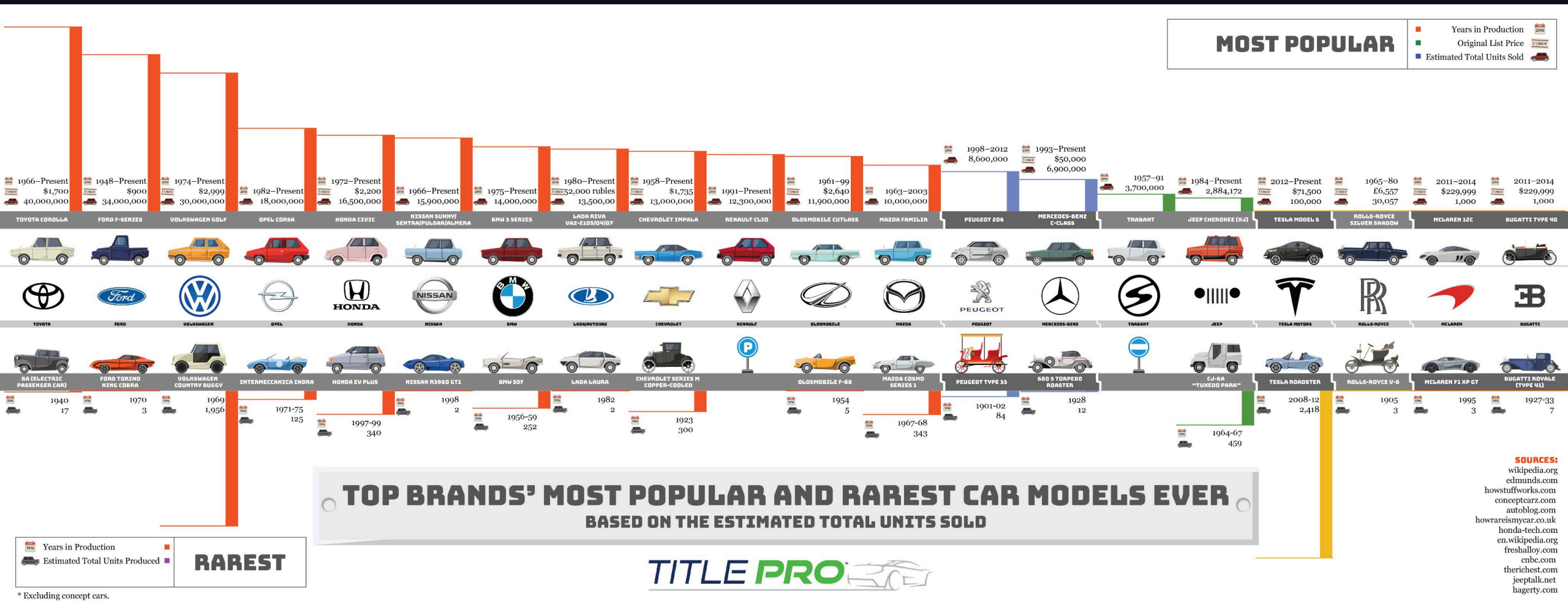
(sort of)

Interactive Activity

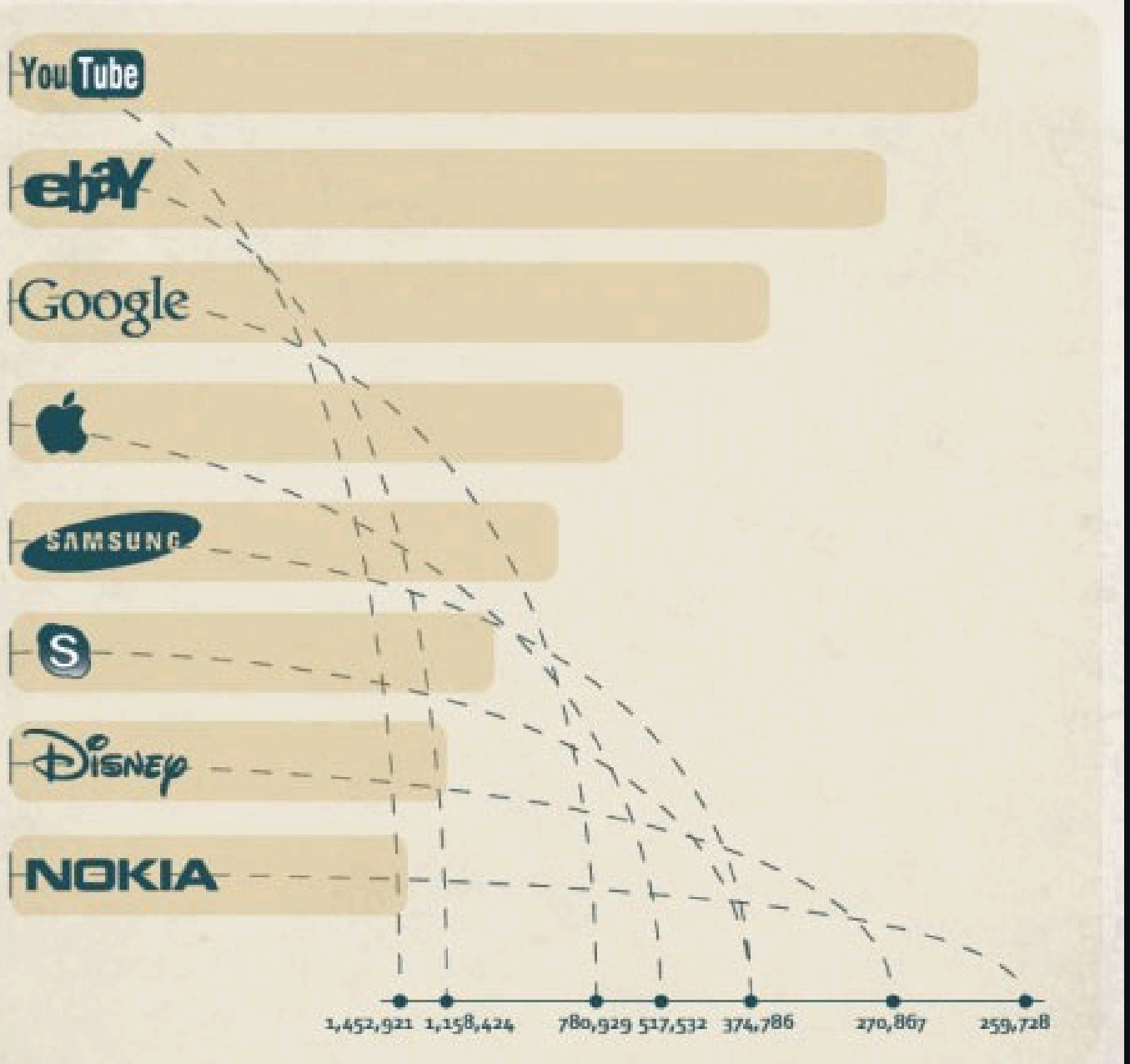
Critiquing a Visualization

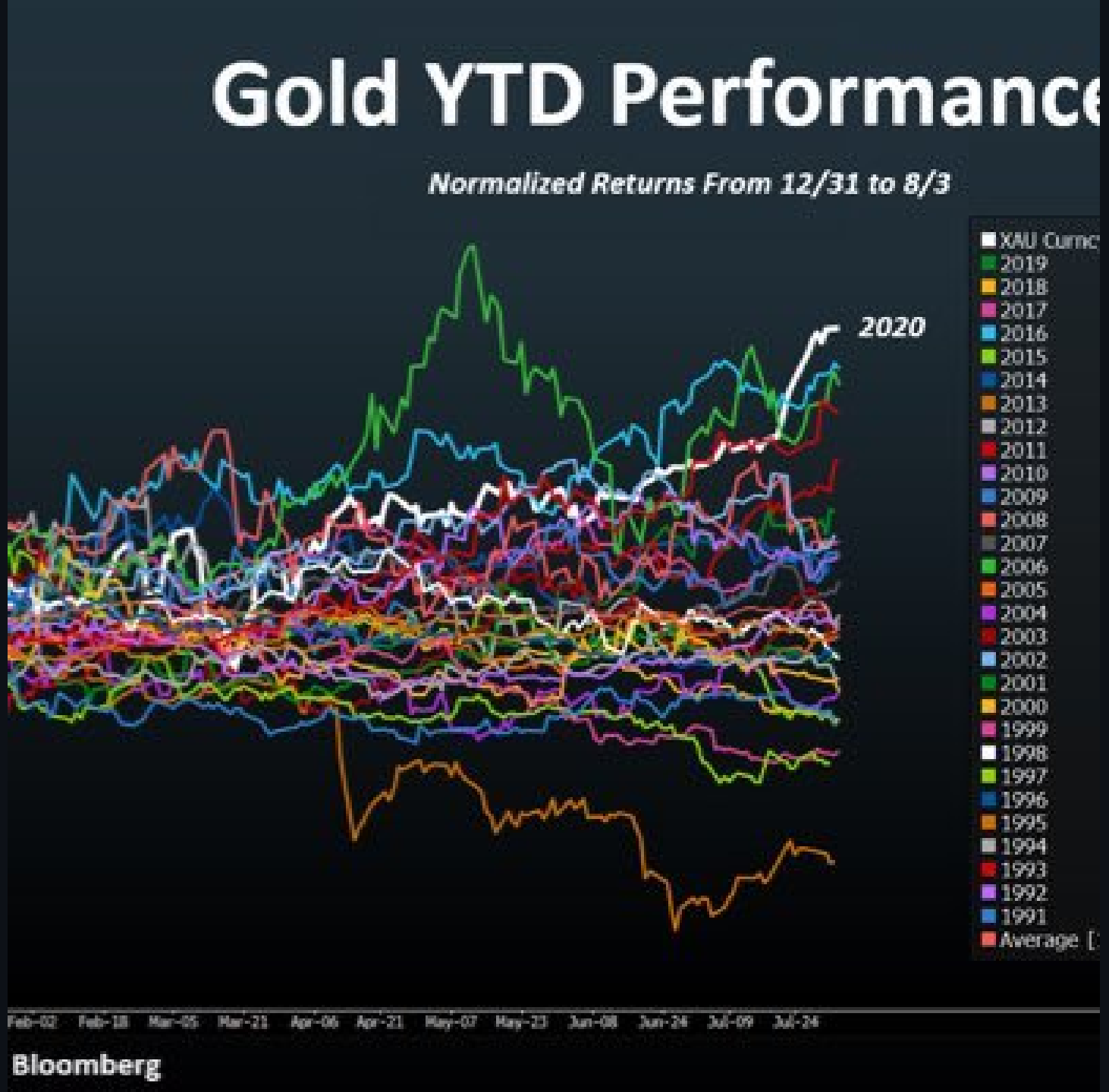
- **Exercise**:
  - Examine the following chart and identify areas for improvement
- **Consider**:
  - Clarity of labels and titles
  - Use of color and chart elements
  - Ethical representation of data
- *Channel your inner Simon Cowell!*


Response
Always wrong
Mostly wrong
Sometimes wrong
Rarely wrong
Not wrong at a
16-24
25-34
35-44
45-54
55-64
65-74
100
75
50
25
0
Percent
Female
Male
Gender


MOST POPULAR
Years in Production
Original List Price
Estimated Total Units Sold
1966–Present
$1,700
40,000,000
TOYOTA COROLLA
1948–Present
$900
34,000,000
FORD F-SERIES
1974–Present
$2,999
30,000,000
VOLKSWAGEN GOLF
1982–Present
18,000,000
OPEL CORSA
1972–Present
$2,200
16,500,000
HONDA CIVIC
1966–Present
15,900,000
NISSAN SUNNY/ SENTRA/PULSAR/ALMERA
1975–Present
14,000,000
BMW 3 SERIES
1980–Present
52,000 rubles
13,500,00
LADA RIVA VAZ-2105/04/07
1958–Present
$1,735
13,000,000
CHEVROLET IMPALA
1991–Present
12,300,000
RENAULT CLIO
1961–99
$2,640
11,900,000
OLDSMOBILE CUTLASS
1963–2003
10,000,000
MAZDA FAMILIA
1998–2012
8,600,000
PEUGEOT 206
1993–Present
$50,000
6,900,000
MERCEDES-BENZ C-CLASS
1957–91
3,700,000
TRABANT
1984–Present
2,884,172
JEEP CHEROKEE (XJ)
2012–Present
$71,500
100,000
TESLA MODEL S
1965–80
£6,557
30,057
ROLLS-ROYCE SILVER SHADOW
2011–2014
$229,999
1,000
MCLAREN 12C
2011–2014
$229,999
1,000
BUGATTI TYPE 40
TOYOTA
FORD
VOLKSWAGEN
OPEL
HONDA
NISSAN
BMW
LADA/AUTOVAZ
CHEVROLET
RENAULT
OLDSMOBILE
MAZDA
PEUGEOT
MERCEDES-BENZ
TRABANT
JEEP
TESLA MOTORS
ROLLS-ROYCE
MCLAREN
BUGATTI
BA (ELECTRIC PASSENGER CAR)
1940
17
FORD TORINO KING COBRA
1970
3
VOLKSWAGEN COUNTRY BUGGY
1969
1,956
INTERMECCANICA INDRA
1971-75
125
HONDA EV PLUS
1997-99
340
NISSAN R390 GT1
1998
2
BMW 507
1956-59
252
LADA LAURA
1982
2
CHEVROLET SERIES M COPPER-COOLED
1923
300
OLDSMOBILE F-88
1954
5
MAZDA COSMO SERIES 1
1967-68
343
PEUGEOT TYPE 33
1901-02
84
680 S TORPEDO ROADSTER
1928
12
CJ-6A "TUXEDO PARK"
1964-67
459
TESLA ROADSTER
2008-12
2,418
ROLLS-ROYCE V-8
1905
3
MCLAREN F1 XP GT
1995
3
BUGATTI ROYALE (TYPE 41)
1927-33
7
TOP BRANDS' MOST POPULAR AND RAREST CAR MODELS EVER
BASED ON THE ESTIMATED TOTAL UNITS SOLD
Years in Production
Estimated Total Units Produced
RAREST
* Excluding concept cars.
TITLE PRO
SOURCES:
wikipedia.org
edmunds.com
howstuffworks.com
conceptcarz.com
autoblog.com
howrareismycar.co.uk
honda-tech.com
en.wikipedia.org
freshalloy.com
cnbc.com
therichest.com
jeeptalk.net
hagerty.com

Top Brands by Social Media Presence




Gold YTD Performance
Normalized Returns From 12/31 to 8/3
2020
XAU Curncy
2019
2018
2017
2016
2015
2014
2013
2012
2011
2010
2009
2008
2007
2006
2005
2004
2003
2002
2001
2000
1999
1998
1997
1996
1995
1994
1993
1992
1991
Average
Feb-02
Feb-18
Mar-05
Mar-21
Apr-06
Apr-21
May-07
May-23
Jun-08
Jun-24
Jul-09
Jul-24
Bloomberg

Advanced Visualization Techniques in Health Data Science

- **Plotnine**
  - Python implementation of the **Grammar of Graphics**
  - Inspired by R's **ggplot2**
- **Command-Line Visualization**
  - Tools for visualizing data directly from the command line
  - Examples: **Mermaid.js**, **spark**
- **Interactive and BI Visualizations**
  - Tools for building interactive dashboards and applications
  - Examples: **Plotly Dash**, **Streamlit**

Plotnine: Grammar of Graphics in Python

Understanding the Grammar of Graphics

- **Theory**: A structured approach to data visualization that breaks down graphs into semantic components
  - **Data**: The dataset being visualized
  - **Aesthetics (aes)**: Mappings between data and visual properties (e.g., x, y, color)
  - **Geometries (geoms)**: Visual elements that represent data (e.g., points, lines)
  - **Facets**: Subsets of data shown in multiple plots
  - **Statistical Transformations (stats)**: Summarizing data (e.g., binning, smoothing)
  - **Scales**: Control mapping from data space to aesthetic space
  - **Coordinate Systems (coords)**: The space in which the data is represented (e.g., Cartesian, polar)

Plotnine vs. ggplot2

- **Similar Syntax and Concepts**
  - Plotnine mirrors ggplot2's structure and functions
  - Beneficial for students familiar with R
- **Example Comparison**
  - **ggplot2 (R)**:

```
ggplot(data, aes(x, y)) + geom_point()
```

  - **Plotnine (Python)**:

```
(ggplot(data, aes('x', 'y')) + geom_point())
```

Important Components in Plotnine

1. `ggplot()`: Initialize a Plot

- **Explanation**: Creates a new plot object with data and aesthetic mappings
- **Structure**:

```
ggplot(data=DataFrame, mapping=aes('x_var', 'y_var'))
```

  - **Required Arguments**:
    - `data`: DataFrame containing the data
    - `mapping`: Aesthetic mappings created with `aes()`

Code Example: Basic Scatter Plot

```
from plotnine import ggplot, aes, geom_point, ggtitle
import pandas as pd

# Sample data
df = pd.DataFrame({
    'BMI': [22, 25, 28, 24, 27],
    'BloodPressure': [120, 130, 125, 118, 135]
})

# Create plot
plot = (ggplot(df, aes(x='BMI', y='BloodPressure'))
        + geom_point(color='blue')
        + ggtitle('Blood Pressure vs BMI'))

print(plot)
```

Code Explanation

- **Importing Libraries:**

```
from plotnine import ggplot, aes, geom_point, ggtitle
import pandas as pd
```

  - Imports necessary components from Plotnine and pandas.

- **Defining Data:**
  - Creates a DataFrame `df` with 'BMI' and 'BloodPressure' columns.

- **Creating the Plot:**

```
plot = (ggplot(df, aes(x='BMI', y='BloodPressure'))
        + geom_point(color='blue')
        + ggtitle('Blood Pressure vs BMI'))
```

  - Initializes the plot with data and aesthetic mappings.
  - Uses `geom_point()` to add scatter plot points.
  - Adds a title with `ggtitle()`.

- **Displaying the Plot:**

Output Description

#FIXME

- A scatter plot showing Blood Pressure versus BMI.
- Each point represents a data entry from the DataFrame.
- The plot includes axis labels and a title.

2. Layering Geometries and Aesthetics

- **Adding Layers**: Use the `+` operator to add layers.
- **Example**: Adding a regression line

```
from plotnine import geom_smooth

plot = (ggplot(df, aes(x='BMI', y='BloodPressure'))
        + geom_point(color='blue')
        + geom_smooth(method='lm')
        + ggtitle('Blood Pressure vs BMI with Regression Line'))
```

Code Explanation

- **Adding `geom_smooth()`**:
  - Adds a smooth line (here, a linear regression line) to the plot.
  - `method='lm'` specifies a linear model.
- **Enhanced Visualization**:
  - Helps in identifying trends or relationships in the data.

Output Description

#FIXME

- The scatter plot now includes a regression line.
- Visualizes the trend of Blood Pressure increasing with BMI.

3. Facetting: Creating Multiple Plots

- **Explanation**: Splits the data into subsets according to a variable and creates multiple plots.
- **Structure**:

```
+ facet_wrap('~ variable')
```

- **Example**:

```
from plotnine import facet_wrap

plot = (ggplot(df, aes(x='BMI', y='BloodPressure'))
        + geom_point(color='blue')
        + facet_wrap('~ AgeGroup')
        + ggtitle('Blood Pressure vs BMI by Age Group'))
```

Code Explanation

- **Assuming `df` Includes 'AgeGroup'**:
  - The DataFrame has a categorical variable 'AgeGroup'.
- **Using `facet_wrap()`**:
  - Creates separate plots for each age group.
- **Purpose**:
  - Allows comparison across different subsets of the data.

Output Description

#FIXME

- Multiple scatter plots, each corresponding to an age group.
- Facilitates analysis of patterns within subgroups.

Command-Line Visualization

Mermaid.js

Introduction

- **Mermaid.js**: A JavaScript-based tool for generating diagrams and flowcharts from text definitions.
- **Advantages**:
  - Quick creation of diagrams without graphic design tools.
  - Integration with markdown documents and presentations.

Use Cases

- **Documentation**: Embed diagrams in technical documents or code repositories.
- **Workflow Visualization**: Illustrate processes, algorithms, or decision trees.
- **Education**: Visual aid for teaching concepts.

Creating Diagrams with Mermaid.js

- **Structure:**

```
```mermaid
[Diagram Definition]
```

- **Example:**

```
graph LR
  A[Start] --> B{Is the patient symptomatic?}
  B -->|Yes| C[Conduct Tests]
  B -->|No| D[Monitor Patient]
  C --> E[Treatment Plan]
  D --> E
  E --> F[Follow-up]
```
```

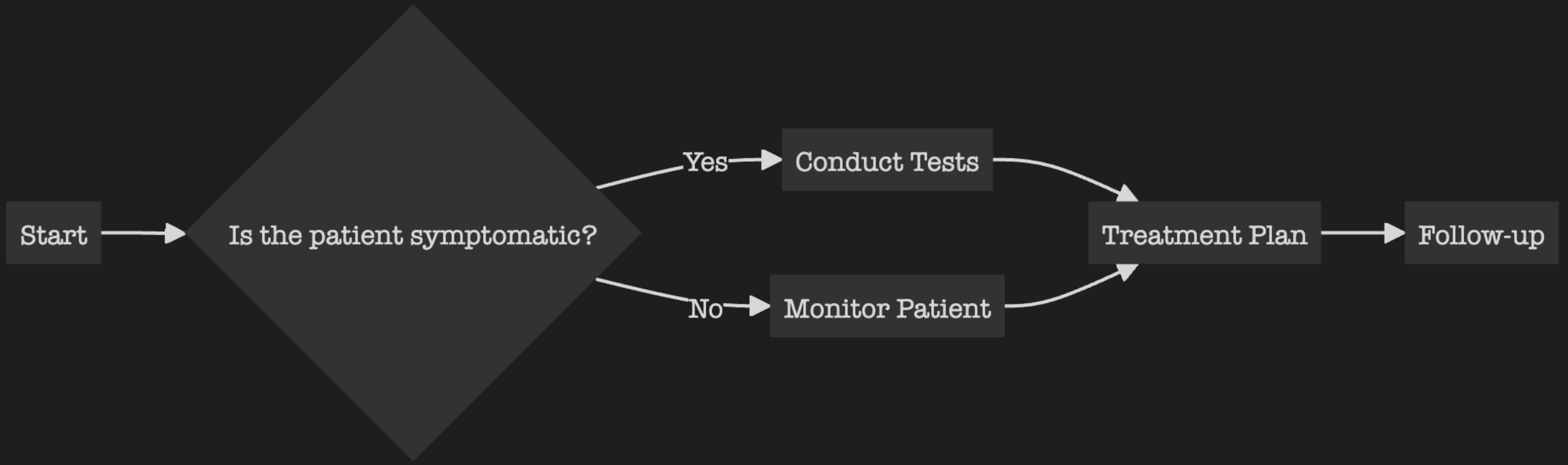

Start
Is the patient symptomatic?
Yes
Conduct Tests
No
Monitor Patient
Treatment Plan
Follow-up

Diagram Explanation

- **Nodes**:
  - `A`, `B`, `C`, `D`, `E`, `F` represent steps or decisions.
- **Edges**:
  - Arrows define the flow between nodes.
  - Labels like `|Yes|` and `|No|` represent decision outcomes.

Integrating Mermaid.js into Documents

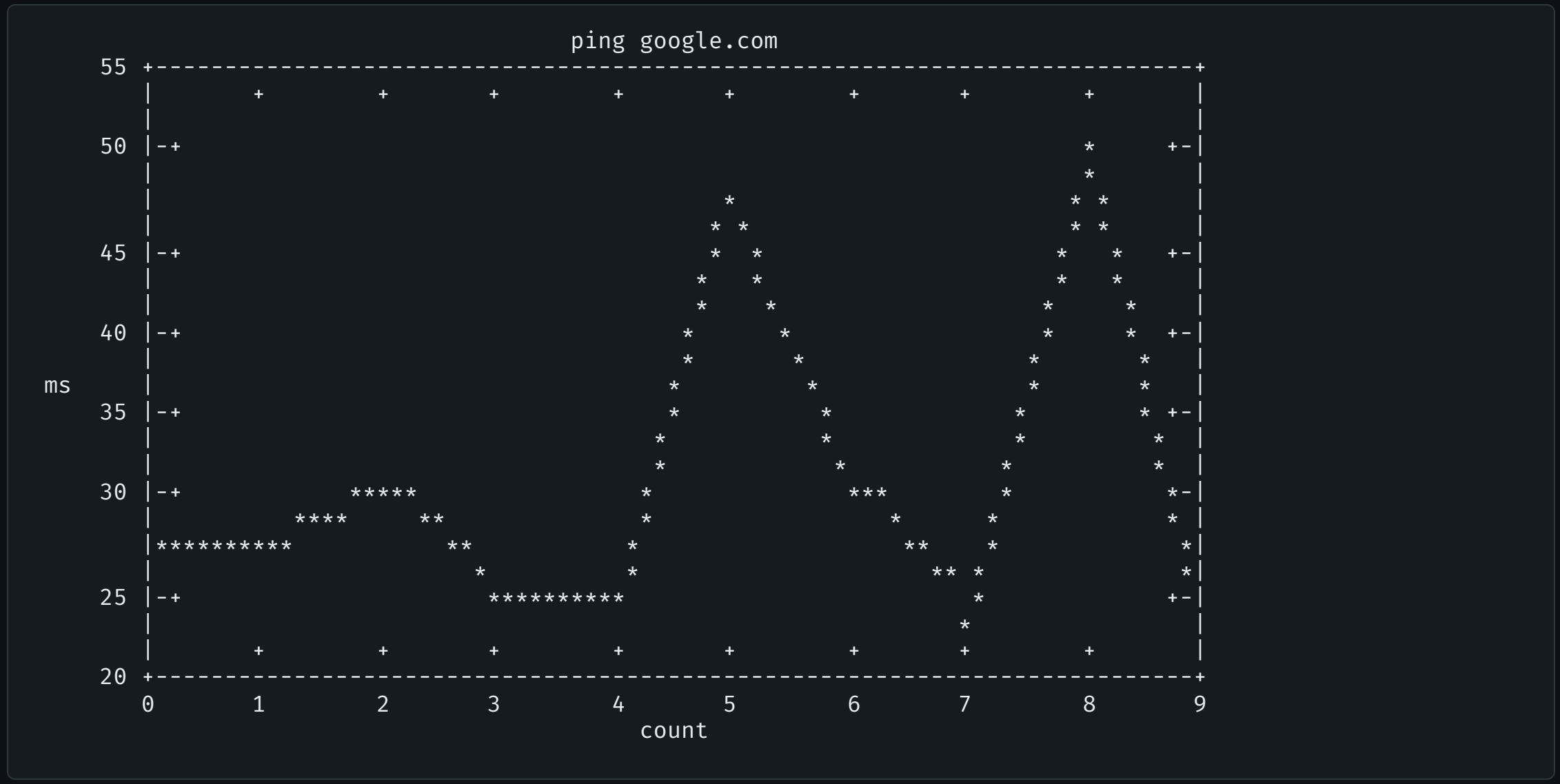
- **Markdown Files**:
  - Supported in many markdown editors and viewers (Notion, VSCode, Obsidian, ...)
- **Presentations**:
  - Tools like **Marp** allow embedding Mermaid diagrams in slides.
- **Version Control**:
  - Diagrams are text-based, facilitating collaboration and version tracking.
- **Command Line**:
  - `mermaid-cli` tool for rendering diagrams from the command line.

```
npm install -g @mermaid-js/mermaid-cli
mmdc -i input.mmd -o output.svg
```

Gnuplot

Viewing graphs from the command line (you will almost never do this, but it's cool!)

```
ping -c 10 google.com -i 0.2 | awk '/time=/{ print $(NF-1) }' | cut -d= -f2 | \
  gnuplot -e \
  "set terminal dumb size 90, 30; set autoscale; set title 'ping google.com';
   set ylabel 'ms'; set xlabel 'count'; plot '-'  with lines notitle";
```


ping google.com
55
50
45
40
35
30
25
20
ms
0
1
2
3
4
5
6
7
8
9
count

Sparkline

```
curl -s https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_day.csv | \
  sed '1d' | \
  cut -d, -f5 | \
  spark
```

Interactive and BI Visualizations

Overview

- **Plotly Dash**:
  - Focused on creating complex, customizable applications.
  - Uses Flask and React.js under the hood.
- **Streamlit**:
  - Designed for rapid development and simplicity.
  - Emphasizes minimal code to produce apps.
- **Tableau/Superset/Looker/PowerBI**:
  - Popular BI tools for creating interactive dashboards. ($$$)

Comparison

Feature	Plotly Dash	Streamlit
Ease of Use	Moderate (requires understanding callbacks)	Easy (simple script-based apps)
Customization	High (extensive layout and styling options)	Moderate
Best For	Complex dashboards, enterprise apps	Quick prototypes, data exploration

Plotly Dash: Building an App

Code Example: Simple Dash App

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

# Load data
df = pd.read_csv('health_data.csv')

# Initialize the app
app = dash.Dash(__name__)

# Define the layout
app.layout = html.Div([
    html.H1('Health Data Dashboard'),
    dcc.Dropdown(
        id='age-dropdown',
        options=[{'label': age, 'value': age} for age in df['AgeGroup'].unique()],
        value=df['AgeGroup'].unique()[0]
    ),
    dcc.Graph(id='bmi-bloodpressure-scatter')
])

# Define the callback
@app.callback(
    Output('bmi-bloodpressure-scatter', 'figure'),
    [Input('age-dropdown', 'value')]
)
def update_graph(selected_age):
    filtered_df = df[df['AgeGroup'] == selected_age]
    fig = px.scatter(
        filtered_df, x='BMI', y='BloodPressure',
        title=f'BMI vs Blood Pressure for Age Group {selected_age}'
    )
    return fig

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

Code Explanation

- **Imports**:
  - `dash`, `dash_core_components` (`dcc`), `dash_html_components` (`html`), `dash.dependencies` for interactivity.
  - `plotly.express` for plotting.
- **Data Loading**:
  - Reads health data into a DataFrame.
- **App Initialization**:
  - Creates a Dash app instance.
- **Layout Definition**:
  - Contains a header, dropdown menu, and a graph component.
- **Callback Function**:
  - Updates the graph based on the selected age group from the dropdown.

Output Description

- An interactive dashboard with:
    - A dropdown to select the age group.
    - A scatter plot of BMI vs. Blood Pressure that updates dynamically.

Streamlit: Building an App

Code Example: Simple Streamlit App

```
import streamlit as st
import pandas as pd
import plotly.express as px

# Load data
df = pd.read_csv('health_data.csv')

# App title
st.title('Health Data Explorer')

# Sidebar filters
age_group = st.sidebar.selectbox('Select Age Group', df['AgeGroup'].unique())

# Filtered data
filtered_df = df[df['AgeGroup'] == age_group]

# Display data
st.write(f'Data for Age Group: {age_group}')
st.write(filtered_df)

# Plot
fig = px.scatter(filtered_df, x='BMI', y='BloodPressure',
                 title=f'BMI vs Blood Pressure for Age Group {age_group}')
st.plotly_chart(fig)
```

Code Explanation

- **Imports**:
  - `streamlit` for the app interface.
  - `pandas` and `plotly.express` for data handling and plotting.
- **Data Loading**:
  - Reads the health data into a DataFrame.
- **App Components**:
  - `st.title()` sets the title.
  - `st.sidebar.selectbox()` creates a dropdown in the sidebar.
  - `st.write()` displays text and data.
  - `st.plotly_chart()` renders the plot.
- **Interactivity**:
  - Selecting an age group filters the data and updates the display and plot.

Output Description

- A simple web app with:
    - A sidebar for selecting the age group.
    - Display of filtered data.
    - An interactive scatter plot.

Deployment Considerations

- **Local Deployment**:
  - Run the app on your local machine for testing and development.
- **Sharing Apps**:
  - **Plotly Dash**:
    - Deploy on platforms like Heroku or Dash Enterprise.
  - **Streamlit**:
    - Use Streamlit Sharing or deploy on a cloud platform.
- **Requirements**:
  - Package dependencies specified in `requirements.txt`.

Applications in Health Data Science

- **Interactive Data Exploration**:
  - Enable users to explore datasets dynamically.
- **Patient Data Dashboards**:
  - Visualize patient metrics for clinical decision support.
- **Educational Tools**:
  - Create apps to teach concepts using real data.

Summary

- **Advanced Visualization Tools** offer powerful ways to represent and interact with health data.
- **Plotnine** brings the grammar of graphics to Python, allowing for elegant and complex static visualizations.
- **Command-Line Tools** like **Mermaid.js** enable quick diagram creation within documentation.
- **Interactive Frameworks** like **Plotly Dash** and **Streamlit** facilitate the development of data apps for deeper insights.

Further Resources

- **Plotnine Documentation**: plotnine.readthedocs.io
- **Mermaid.js Documentation**: mermaid-js.github.io
- **Plotly Dash User Guide**: dash.plotly.com
- **Streamlit Documentation**: docs.streamlit.io
- **Deployment Guides**:
  - Dash: Deployment
  - Streamlit: Sharing Apps