

LECTURE 8

# Visualization I

Visualizing distributions and KDEs

**Data Science, Fall 2023 @ Knowledge Stream**

Sana Jabbar

# Goals for this Lecture

---

## Lecture 9

Reviewing and concluding regex

Understand the theories behind effective visualizations and start to generate plots of our own

- The necessary "pre-thinking" before creating a plot
- Python libraries for visualizing data

# Agenda

---

## Lecture 9

- Regex
  - Regex review and regex functions
- Visualization
  - Goals of visualization
  - Visualizing distributions
  - Kernel density estimation


# Regex Review and regex Functions

---

## Lecture 9

- **Regex**
  - **Regex review and regex functions**
- Visualization
  - Goals of visualization
  - Visualizing distributions
  - Kernel density estimation

| Operation                                   | Order | Example   | Matches        | Doesn't match      |
|---|-------|-----------|----------------|--------------------|
| <b>concatenation</b><br>(consecutive chars) | 3     | AABAAB    | AABAAB         | every other string |
| <b>or,  </b>                                | 4     | AA BAAB   | AA<br>BAAB     | every other string |
| <b>*</b><br>(zero or more)                  | 2     | AB*A      | AA<br>ABBBBBBA | AB<br>ABABA        |
| <b>group</b><br>(parenthesis)               | 1     | A(A B)AAB | AAAAB<br>ABAAB | every other string |
|   |       | (AB)*A    | A<br>ABABABABA | AA<br>ABBA         |

 The regex order of operations. Grouping is evaluated first.

| Operation                                | Example        | Matches             | Doesn't match          |
|--|----------------|---------------------|------------------------|
| <b>any character</b><br>(except newline) | .U.U.U.        | CUMULUS<br>JUGULUM  | SUCCUBUS<br>TUMULTUOUS |
| <b>character class</b>                   | [A-Za-z][a-z]* | word<br>Capitalized | camelCase<br>4illegal  |
| <b>repeated exactly a times: {a}</b>     | j[aeiou]{3}hn  | jaoehn<br>jooohn    | jhn<br>jaeiouhn        |
| <b>repeated from a to b times: {a,b}</b> | j[ou]{1,2}hn   | john<br>juohn       | jhn<br>jooohn          |
| <b>at least one</b>                      | jo+hn          | john<br>joooooooohn | jhn<br>jjohn           |

| Operation         | Example               | Matches              | Doesn't match |
|-------------------|-----------------------|----------------------|---------------|
| beginning of line | <code>^ark</code>     | ark two<br>ark o ark | dark          |
| end of line       | <code>ark\$</code>    | dark<br>ark o ark    | ark two       |
| escape character  | <code>cow\.com</code> | cow.com              | cowscom       |

## Extraction


```
re.findall(pattern, text)
```

Return a list of all matches to **pattern**.

```
text = "My social security number is 123-45-  
6789 bro, or actually maybe it's 321-45-  
6789.";
```

```
pattern = r"[0-9]{3}-[0-9]{2}-[0-9]{4}"  
re.findall(pattern, text)
```

```
['123-45-6789', '321-45-6789']
```



A **match** is a substring that matches the provided regex.



## Extraction

```
re.findall(pattern, text)
```

Return a list of all matches to **pattern**.

```
text = "My social security number is 123-45-6789 bro, or actually maybe it's 321-45-6789.";  
pattern = r"[0-9]{3}-[0-9]{2}-[0-9]{4}"  
re.findall(pattern, text)
```

```
['123-45-6789', '321-45-6789']
```

A **match** is a substring that matches the provided regex.

```
ser.str.findall(pattern)
```

Returns a Series of lists

```
df["SSN"].str.findall(pattern)
```

|   | SSN                            |
|---|--------------------------------|
| 0 | 987-65-4321                    |
| 1 | forty                          |
| 2 | 123-45-6789 bro or 321-45-6789 |
| 3 | 999-99-9999                    |

```
0          [987-65-4321]  
1                  []  
2  [123-45-6789, 321-45-6789]  
3          [999-99-9999]  
Name: SSN, dtype: object
```

## Extraction with Capture Groups

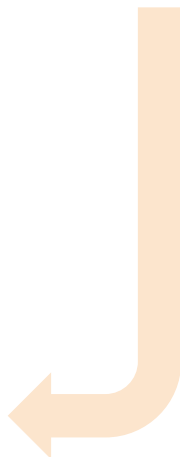
`ser.str.extract(pattern)`

Returns a DataFrame of each capture group's **first** match in the string

```
pattern_cg = r"([0-9]{3})-([0-9]{2})-([0-9]{4})"  
df["SSN"].str.extract(pattern_cg)
```

|   | SSN                            |
|---|--------------------------------|
| 0 | 987-65-4321                    |
| 1 | forty                          |
| 2 | 123-45-6789 bro or 321-45-6789 |
| 3 | 999-99-9999                    |

|   | 0   | 1   | 2    |
|---|-----|-----|------|
| 0 | 987 | 65  | 4321 |
| 1 | NaN | NaN | NaN  |
| 2 | 123 | 45  | 6789 |
| 3 | 999 | 99  | 9999 |



# Extraction with Capture Groups

## ser.str.extract(pattern)

Returns a DataFrame of each capture group's **first** match in the string

```
pattern_cg = r"([0-9]{3})-([0-9]{2})-([0-9]{4})"
df["SSN"].str.extract(pattern_cg)
```

|   | SSN                            |
|---|--------------------------------|
| 0 | 987-65-4321                    |
| 1 | forty                          |
| 2 | 123-45-6789 bro or 321-45-6789 |
| 3 | 999-99-9999                    |

|   | 0   | 1   | 2    |
|---|-----|-----|------|
| 0 | 987 | 65  | 4321 |
| 1 | NaN | NaN | NaN  |
| 2 | 123 | 45  | 6789 |
| 3 | 999 | 99  | 9999 |



## ser.str.extractall(pattern)

Returns a multi-indexed DataFrame of **all** matches for each capture group

```
df["SSN"].str.extractall(pattern_cg)
```

|   | SSN                            |
|---|--------------------------------|
| 0 | 987-65-4321                    |
| 1 | forty                          |
| 2 | 123-45-6789 bro or 321-45-6789 |
| 3 | 999-99-9999                    |

|       |   | 0   | 1  | 2    |
|-------|---|-----|----|------|
| match |   |     |    |      |
| 0     | 0 | 987 | 65 | 4321 |
| 2     | 0 | 123 | 45 | 6789 |
|       | 1 | 321 | 45 | 6789 |
| 3     | 0 | 999 | 99 | 9999 |



## Substitution

---

`re.sub(pattern, repl, text)`

Returns text with all instances of **pattern** replaced by **repl**.

```
text = '<div><td valign="top">Moo</td></div>'
pattern = r"<[^>]+>"
re.sub(pattern, '', text) # returns Moo
```

Moo



How it works:

- **pattern** matches HTML tags
- Then, sub/replace HTML tags with **repl=''** (i.e., empty string)

## Substitution

```
re.sub(pattern, repl, text)
```

Returns text with all instances of **pattern** replaced by **repl**.

```
text = '<div><td  
valign="top">Moo</td></div>'  
pattern = r"<[^>]+>"  
re.sub(pattern, '', text) # returns Moo
```

Moo



How it works:

- **pattern** matches HTML tags
- Then, sub/replace HTML tags with **repl=''** (i.e., empty string)

```
ser.str.replace(pattern, repl,
```

```
regex=True )
```

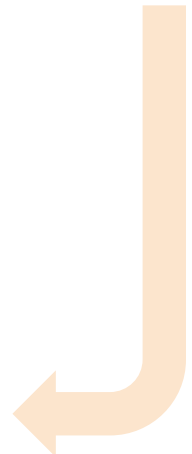
Returns Series with all instances of the **pattern** in Series **ser** replaced by **repl**.

```
df["Html"].str.replace(pattern, '')
```

Html

```
0 <div><td valign="top">Moo</td></div>  
1 <a href="http://ds100.org">Link</a>  
2 <b>Bold text</b>
```

```
0 Moo  
1 Link  
2 Bold text  
Name: Html, dtype: object
```



## String Function Summary

| Base Python                                      | re                           | pandas str   |
|--|------------------------------|--|
| <code>s.lower()</code><br><code>s.upper()</code> |                              | <code>ser.str.lower()</code><br><code>ser.str.upper()</code>   |
| <code>s.replace(...)</code>                      | <code>re.sub(...)</code>     | <code>ser.str.replace(...)</code>  |
| <code>s.split(...)</code>                        | <code>re.split(...)</code>   | <code>ser.str.split(...)</code>  |
| <code>s[1:4]</code>                              |                              | <code>ser.str[1:4]</code>  |
|  | <code>re.findall(...)</code> | <code>ser.str.findall(...)</code><br><code>ser.str.extractall(...)</code><br><code>ser.str.extract(...)</code> |
| <code>'ab' in s</code>                           | <code>re.search(...)</code>  | <code>ser.str.contains(...)</code>   |
| <code>len(s)</code>                              |                              | <code>ser.str.len()</code>   |
| <code>s.strip()</code>                           |                              | <code>ser.str.strip()</code>   |



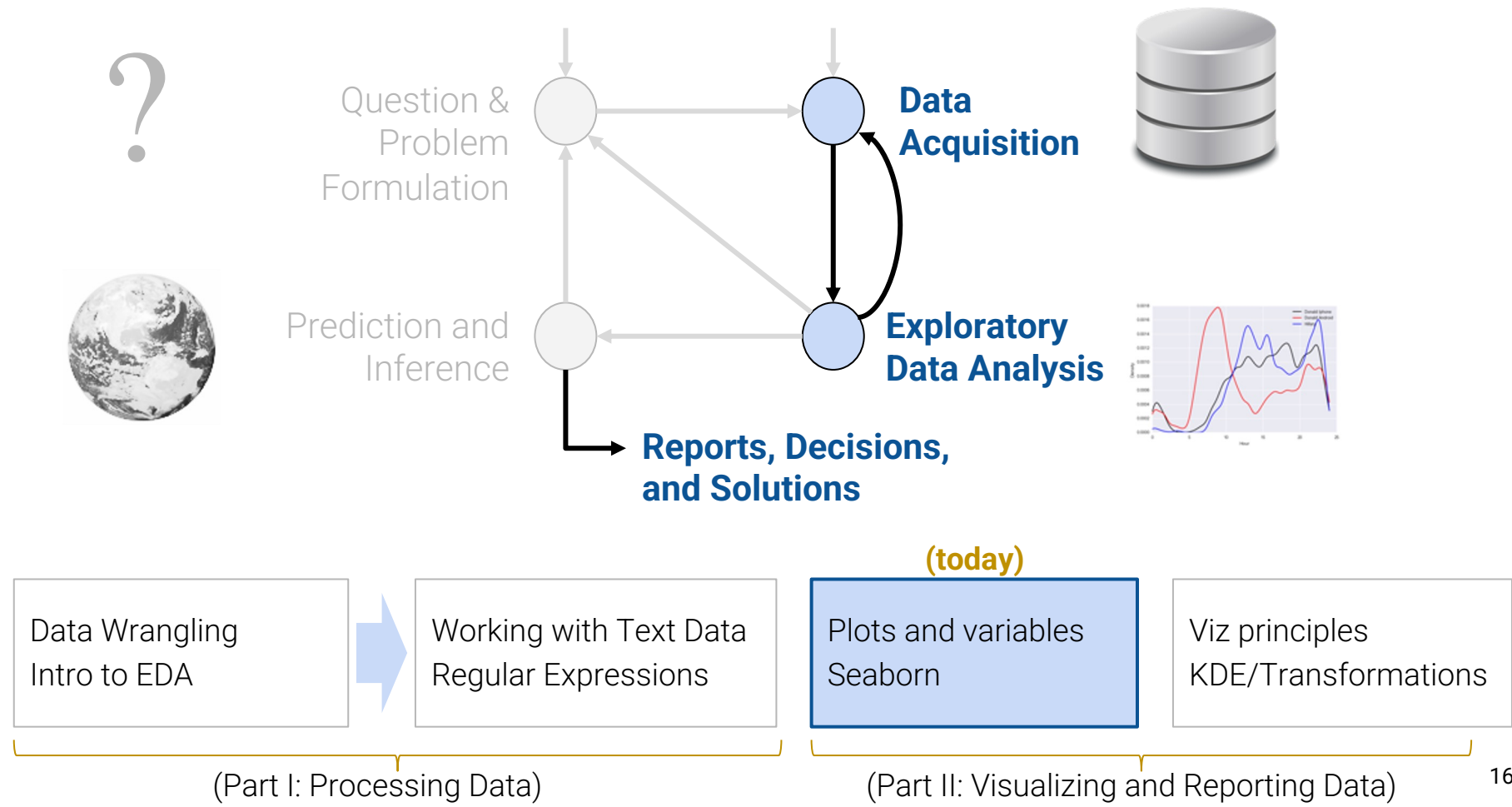
# Goals of Visualization

---

Lecture 8, Spring 2024

- Regex
  - Regex review and regex functions
- **Visualization**
  - **Goals of visualization**
  - Visualizing distributions
  - Kernel density estimation

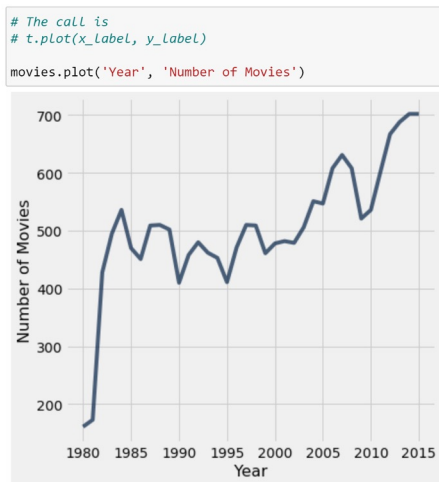
# Where are we?



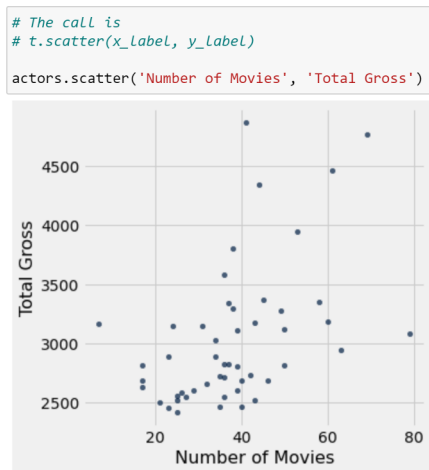


# Visualizations in BS (and in Data Science, so far)

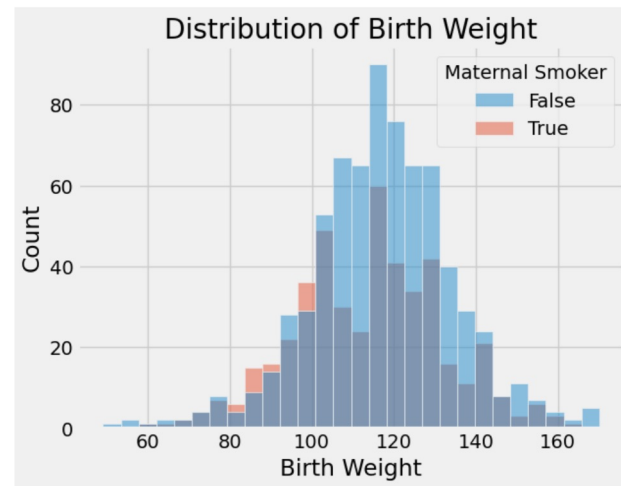
You worked with many types of visualizations throughout.



Line plot



Scatter plot



Histogram

What did these achieve?

- Provide a high-level overview of a complex dataset.
- Communicated trends to viewers.

# Goals of Data Visualization

---

Goal 1: To **help your own understanding** of your data/results.

- Key part of exploratory data analysis.
- Summarize trends visually before in-depth analysis.
- Lightweight, iterative and flexible.

Goal 2: To **communicate results/conclusions to others**.

- Highly editorial and selective.
- Be thoughtful and careful!
- Fine-tuned to achieve a communications goal.
- Considerations: clarity, accessibility, and necessary context.

What do these goals imply?

Visualizations aren't a matter of making "pretty" pictures.

We need to do a lot of thinking about what stylistic choices communicate ideas most effectively.

# Goals of Data Visualization

---

What do these goals imply?

Visualizations aren't a matter of making "pretty" pictures.

We need to do a lot of thinking about what stylistic choices communicate ideas most effectively.



First half of visualization topics in Data Science:  
Choosing the "right" plot for

- Introducing plots for different variable types
- Generating these plots through code

Second half of visualization topics in Data Science:  
Stylizing plots appropriately

- Smoothing and transforming visual data
- Providing context through labeling and color

# Visualizing Distributions

---

## Lecture 9

- Regex
  - Regex review and regex functions
- **Visualization**
  - Goals of visualization
  - **Visualizing distributions**
  - Kernel density estimation

A distribution describes...

- The set of values that a variable can possibly take.
- The frequency with which each value occurs for a **single** variable

Example: Distribution of students across discussion sections in Data Science.

- The list of discussion sections (09-12 pm, 02-05 pm, etc.)
- The number of students enrolled in each section

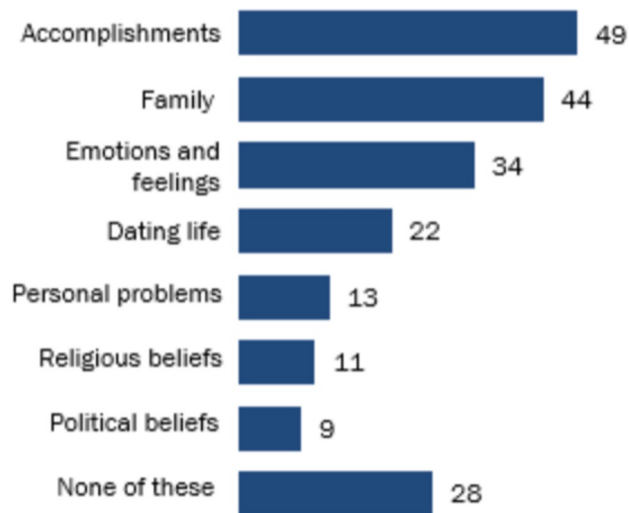
In other words: How is the variable distributed across all of its possible values?

This means that percentages **should sum to 100%** (if using proportions) and counts should **sum to the total number of datapoints** (if using raw counts).

Let's see some examples.

## While about half of teens post their accomplishments on social media, few discuss their religious or political beliefs

*% of U.S. teens who say they ever post about their \_\_\_ on social media*



Note: Respondents were allowed to select multiple options.

Respondents who did not give an answer are not shown.

Source: Survey conducted March 7–April 10, 2018.

"Teens' Social Media Habits and Experiences"

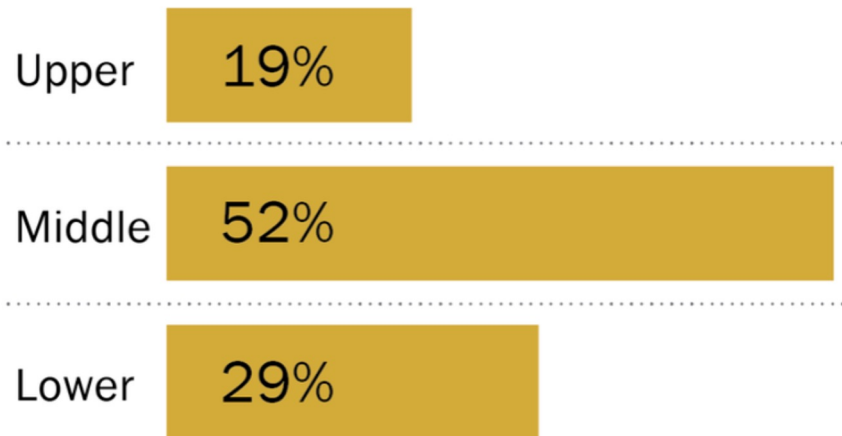
PEW RESEARCH CENTER

## Does this chart show a distribution?

**No.**

- The chart does show percents of individuals in different categories!
- But, this is not a distribution because individuals can be in more than one category (see the fine print).

SHARE OF AMERICAN ADULTS  
IN EACH INCOME TIER



**Does this chart show a distribution?**

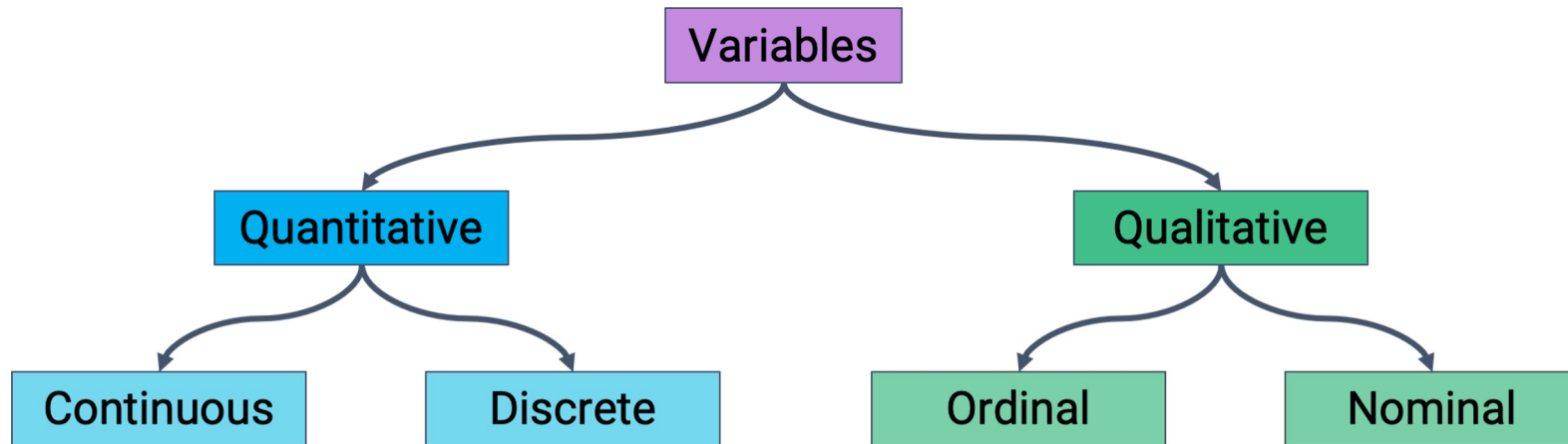
**Yes!**

- This chart shows the distribution of the qualitative ordinal variable "income tier."
- Each individual is in exactly one category.
- The values we see are the proportions of individuals in that category.
- Everyone is represented, as the total percentage is 100%.

## Variable Types Should Inform Plot Choice

---

Different plots are more or less suited for displaying particular types of variables.



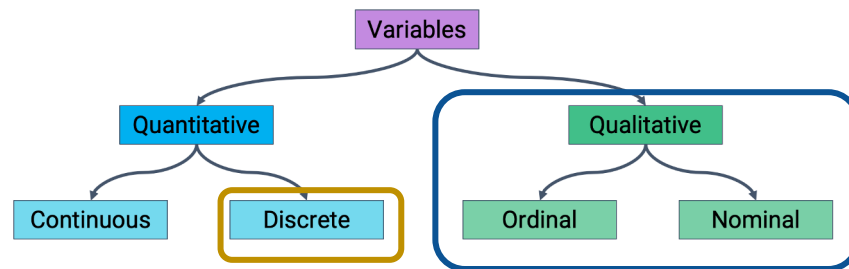
First step of visualization: Identify the variables being visualized. Then, select a plot type accordingly.



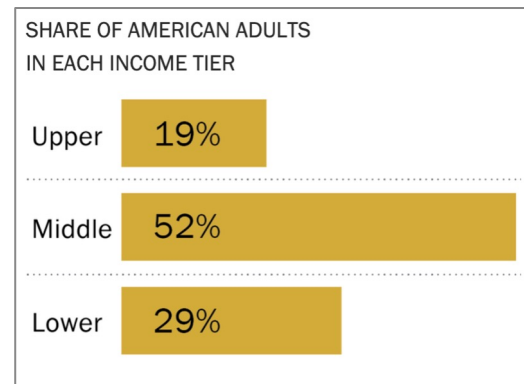
# Bar Plots: Distributions of Qualitative Variables

Bar plots are the most common way of displaying the **distribution** of a **qualitative** variable.

\*Sometimes quantitative discrete data too, if there are few unique values.



- For example, the proportion of adults in the upper, middle, and lower classes.
- Lengths encode values.
  - *Widths* encode *nothing*!
  - *Color* could indicate a sub-category (but not necessarily).



We will be using the **wb** dataset about world countries for most of our work today.

|   | Continent | Country  | Primary completion rate: Male: % of relevant age group: 2015 | Primary completion rate: Female: % of relevant age group: 2015 | Lower secondary completion rate: Male: % of relevant age group: 2015 | Lower secondary completion rate: Female: % of relevant age group: 2015 | Youth literacy rate: Male: % of ages 15-24: 2005-14 | Youth literacy rate: Female: % of ages 15-24: 2005-14 | Adult literacy rate: Male: % ages 15 and older: 2005-14 | Adult literacy rate: Female: % ages 15 and older: 2005-14 |
|---|-----------|----------|--|--|--|--|---|---|---|---|
|   |           |          |  |  |  |  |   |   |   |   |
| 0 | Africa    | Algeria  | 106.0  | 105.0  | 68.0   | 85.0   | 96.0  | 92.0  | 83.0  | 68.0  |
| 1 | Africa    | Angola   | NaN  | NaN  | NaN  | NaN  | 79.0  | 67.0  | 82.0  | 60.0  |
| 2 | Africa    | Benin    | 83.0   | 73.0   | 50.0   | 37.0   | 55.0  | 31.0  | 41.0  | 18.0  |
| 3 | Africa    | Botswana | 98.0   | 101.0  | 86.0   | 87.0   | 96.0  | 99.0  | 87.0  | 89.0  |
| 5 | Africa    | Burundi  | 58.0   | 66.0   | 35.0   | 30.0   | 90.0  | 88.0  | 89.0  | 85.0  |

## Generating Bar Plots: Matplotlib


---

We will mainly use two libraries for generating plots: [Matplotlib](#) and [Seaborn](#).

Most Matplotlib plotting functions follow the same structure: We pass in a sequence (**list**, **array**, or **Series**) of values to be plotted on the x-axis, and a second sequence of values to be plotted on the y-axis.

```
import matplotlib.pyplot as plt  
plt.plotting_function(x_values, y_values)
```

Matplotlib is typically  
given the alias `plt`



To add labels and a title:

```
plt.xlabel("x axis label")  
plt.ylabel("y axis label")  
plt.title("Title of the plot");
```

## Generating Bar Plots: Matplotlib

To create a bar plot in Matplotlib: `plt.bar( )`

```
continents = wb["Continent"].value_counts()
```

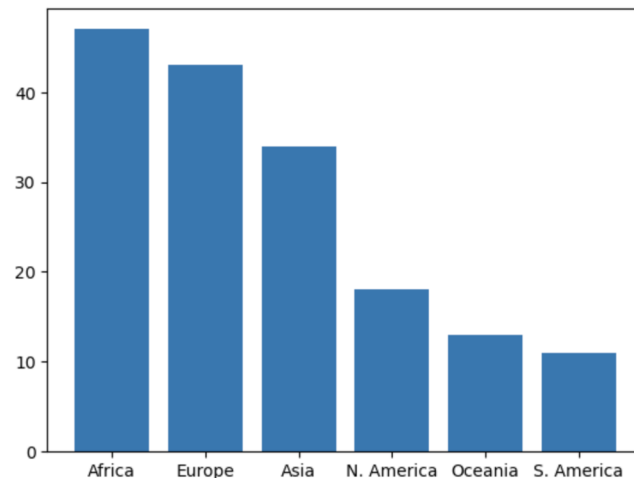
|            |    |
|------------|----|
| Africa     | 47 |
| Europe     | 43 |
| Asia       | 34 |
| N. America | 18 |
| Oceania    | 13 |
| S. America | 11 |

Name: Continent, dtype: int64

```
plt.bar(continents.index, continents.values);
```

x values

y values

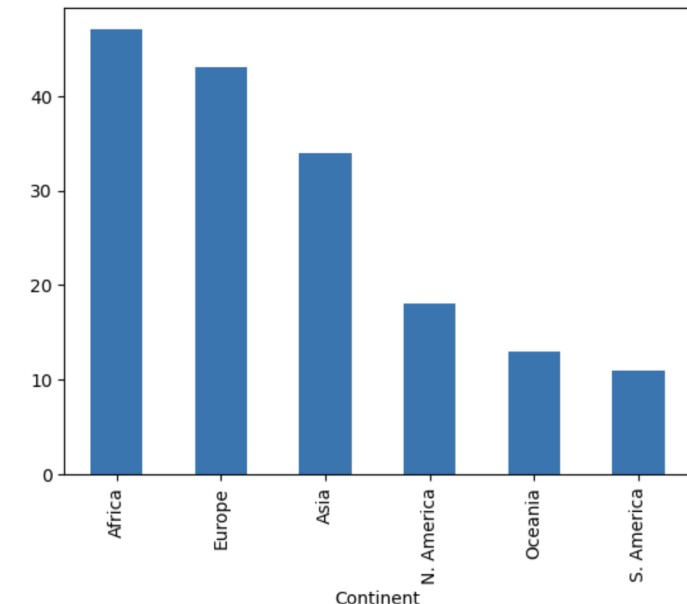


## Generating Bar Plots: pandas Native Plotting

To create a bar plot in native pandas: `.plot(kind='bar')`

```
Africa      47  
Europe      43  
Asia        34  
N. America  18  
Oceania     13  
S. America  11  
Name: Continent, dtype: int64
```

```
wb["Continent"].value_counts().plot(kind='bar')
```




## Generating Bar Plots: Seaborn

---

Seaborn plotting functions use a different structure: Pass in an entire **DataFrame**, then specify what column(s) to plot.

```
import seaborn as sns  
sns.plotting_function(data=df, x="x_col", y="y_col")
```

Seaborn is typically given the alias `sns`

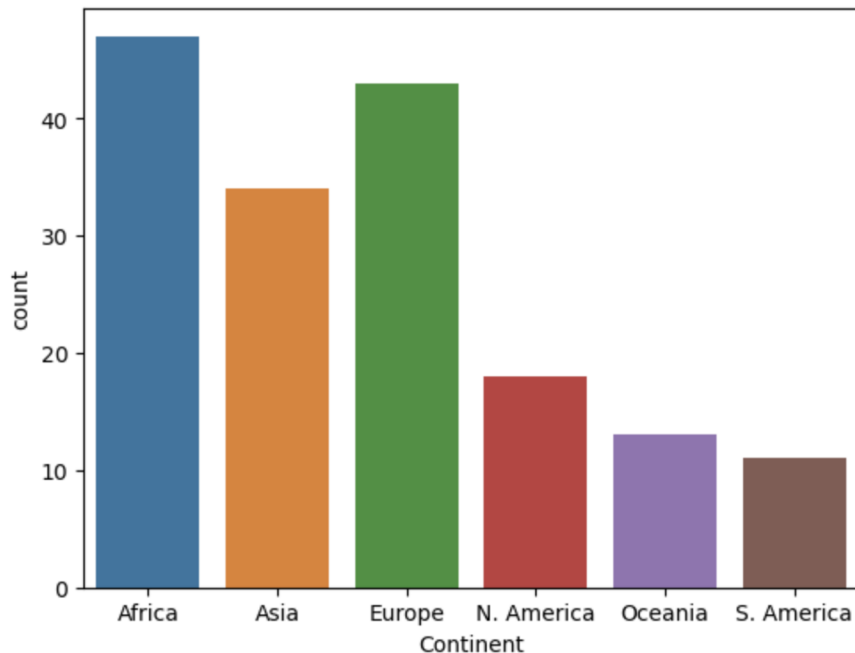


To add labels and a title, use the same syntax as before:

```
plt.xlabel("x axis label")  
plt.ylabel("y axis label")  
plt.title("Title of the plot");
```

## Generating Bar Plots: Seaborn

To create a bar plot in Seaborn: `sns.countplot( )`



`countplot` operates at a higher level of abstraction!

You give it the entire **DataFrame** and it does the counting for you.

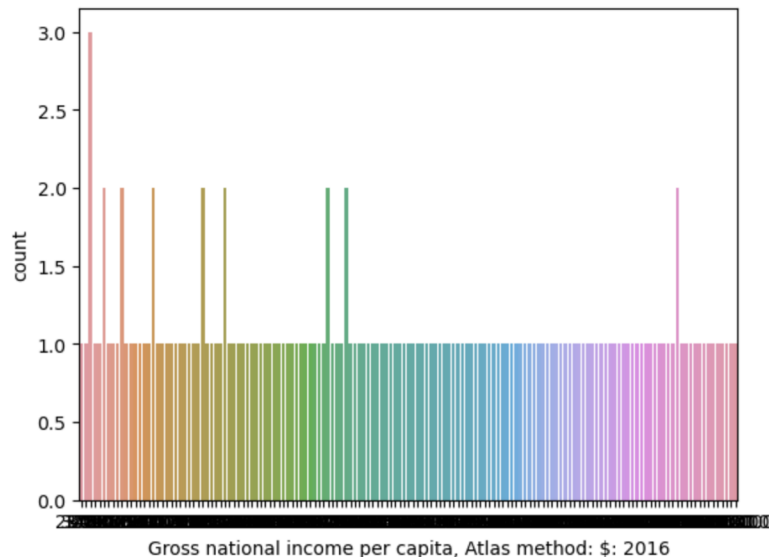
```
import seaborn as sns  
  
sns.countplot(data=wb, x="Continent");
```

## Distributions of Quantitative Variables

Earlier, we said that bar plots are appropriate for distributions of qualitative variables.

Why only qualitative? Why not quantitative as well?

- For example: The distribution of gross national income per capita.

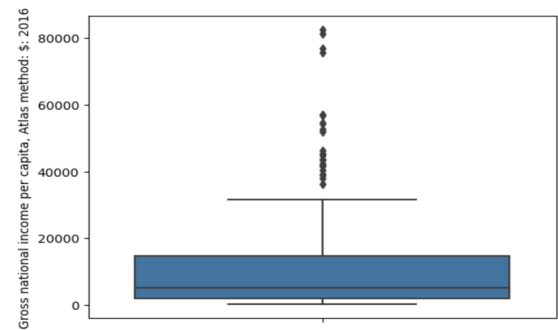


A bar plot will create a separate bar for each unique value. This leads to too many bars for continuous data!

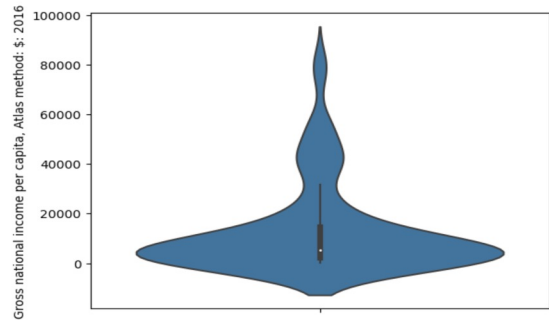


# Distributions of Quantitative Variables

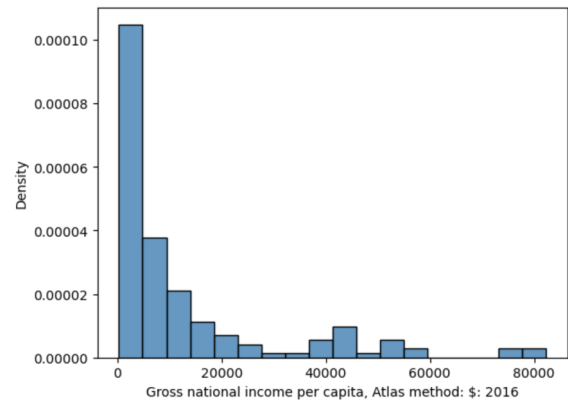
To visualize the distribution of a continuous quantitative variable:



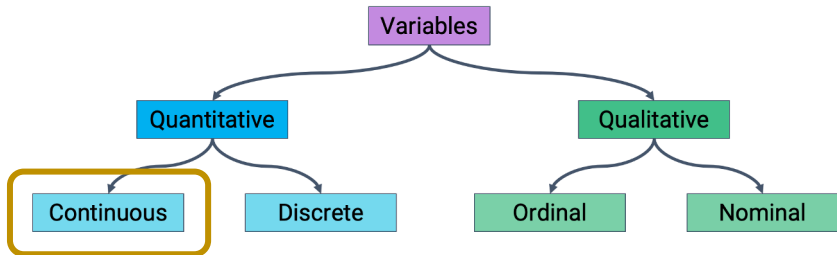
Box plot



Violin plot



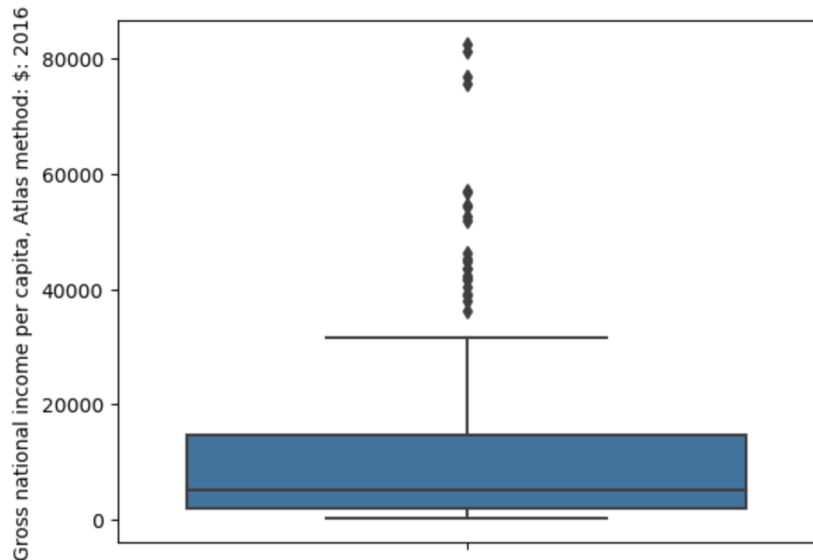
Histogram



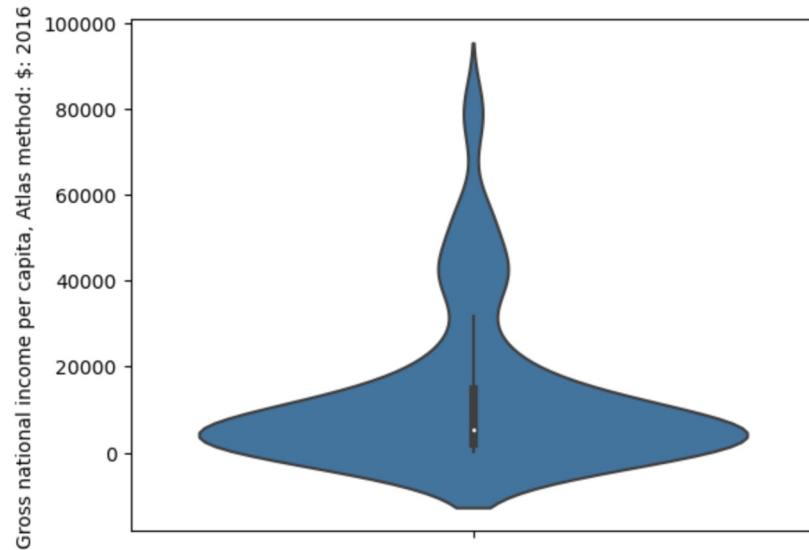
## Box plots and Violin Plots

Box plots and violin plots display distributions using information about **quartiles**.

- In a box plot, the width of the box encodes no meaning.
- In a violin plot, the width of the "violin" indicates the density of datapoints at each value.

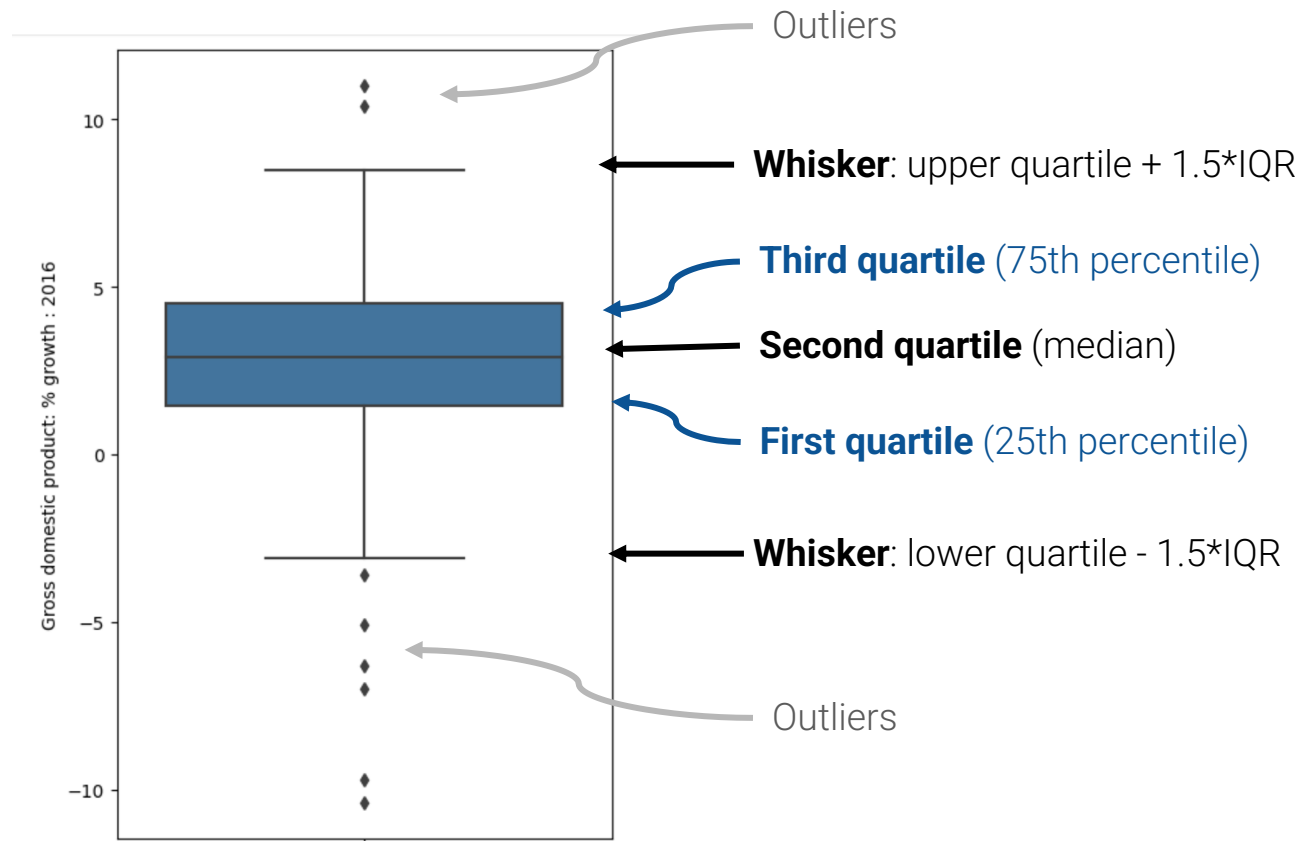


```
sns.boxplot(data=df, y="y_variable");
```



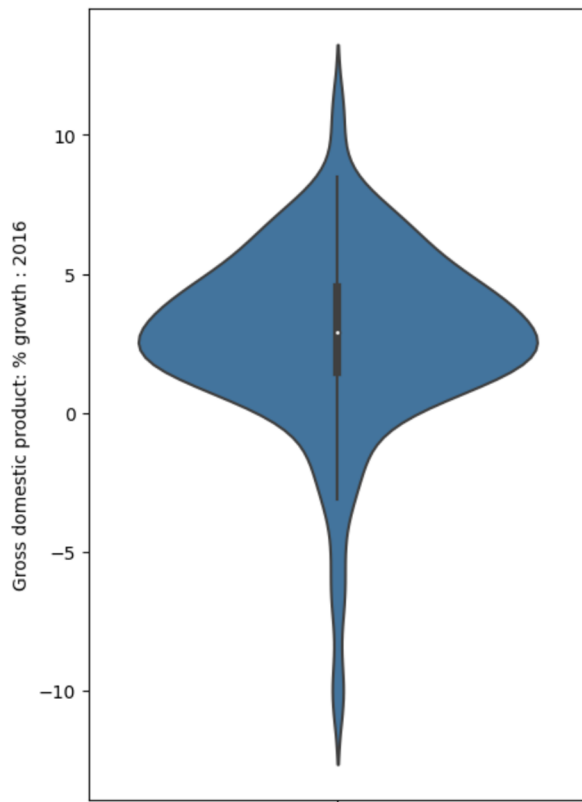
```
sns.violinplot(data=df, y="y_variable");
```

# Box Plots



```
sns.boxplot(data=wb, y="Gross domestic product: % growth : 2016")
```

## Violin Plots



Violin plots are similar to box plots, but also show smoothed density curves.

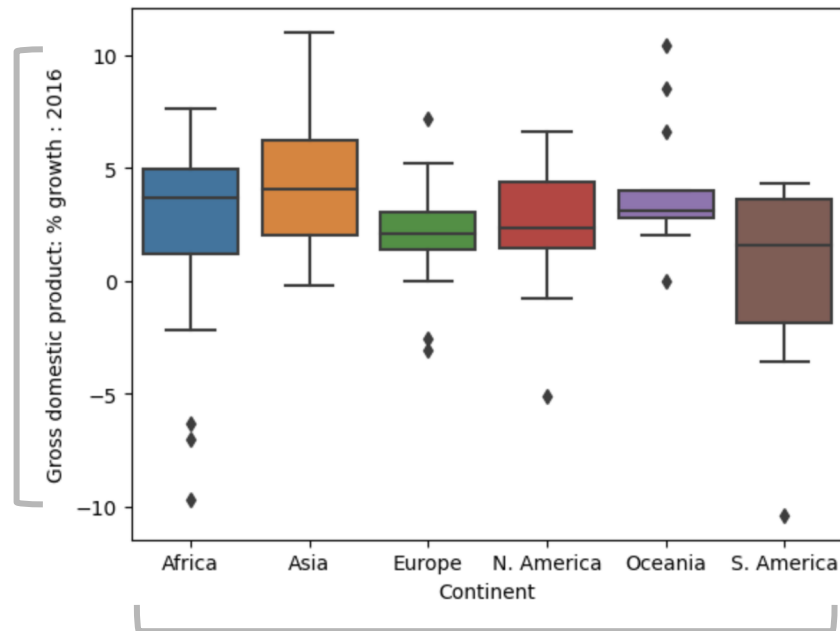
- The "width" of our "box" now has meaning!
- The three quartiles and "whiskers" are still present – look closely.

```
sns.violinplot(data=wb, y="Gross domestic product: % growth : 2016")
```

## Side-by-side Box and Violin Plots

What if we wanted to incorporate a *qualitative* variable as well? For example, compare the distribution of a quantitative continuous variable *across* different qualitative categories.

```
sns.boxplot(data=wb, x="Continent", y="Gross domestic product: % growth : 2016");
```



GDP growth:  
quantitative continuous

Continent: qualitative nominal

# Quartiles

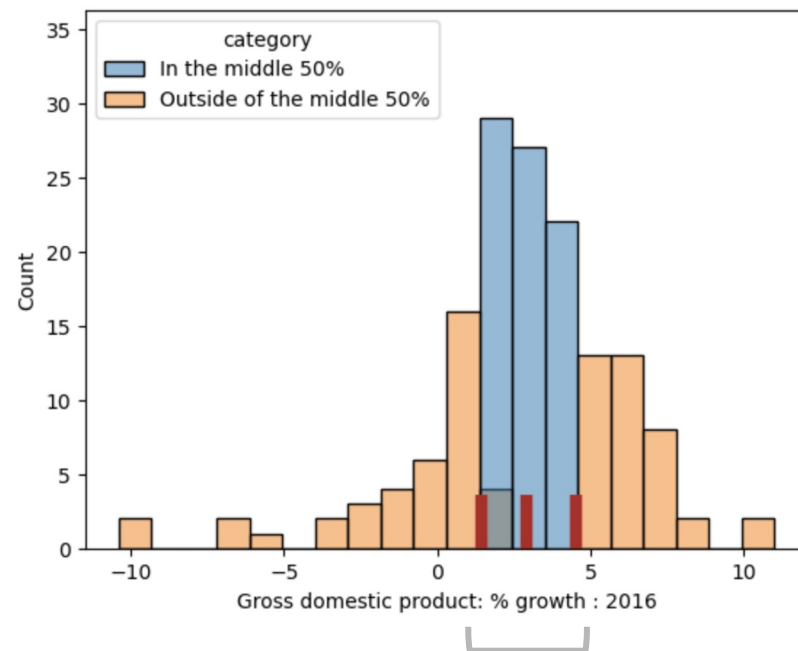
For a quantitative variable:

- First or lower quartile: 25th percentile.
- Second quartile: 50th percentile (median).
- Third or upper quartile: 75th percentile.

The interval [first quartile, third quartile] contains the "middle 50%" of the data.

**Interquartile range (IQR)** measures spread.

- $IQR = \text{third quartile} - \text{first quartile}$ .



The length of this region is the IQR

## LECTURE 8

# Visualization

Start Working on Notebooks