



practicum
building ai knowledge

Python Crash Course!



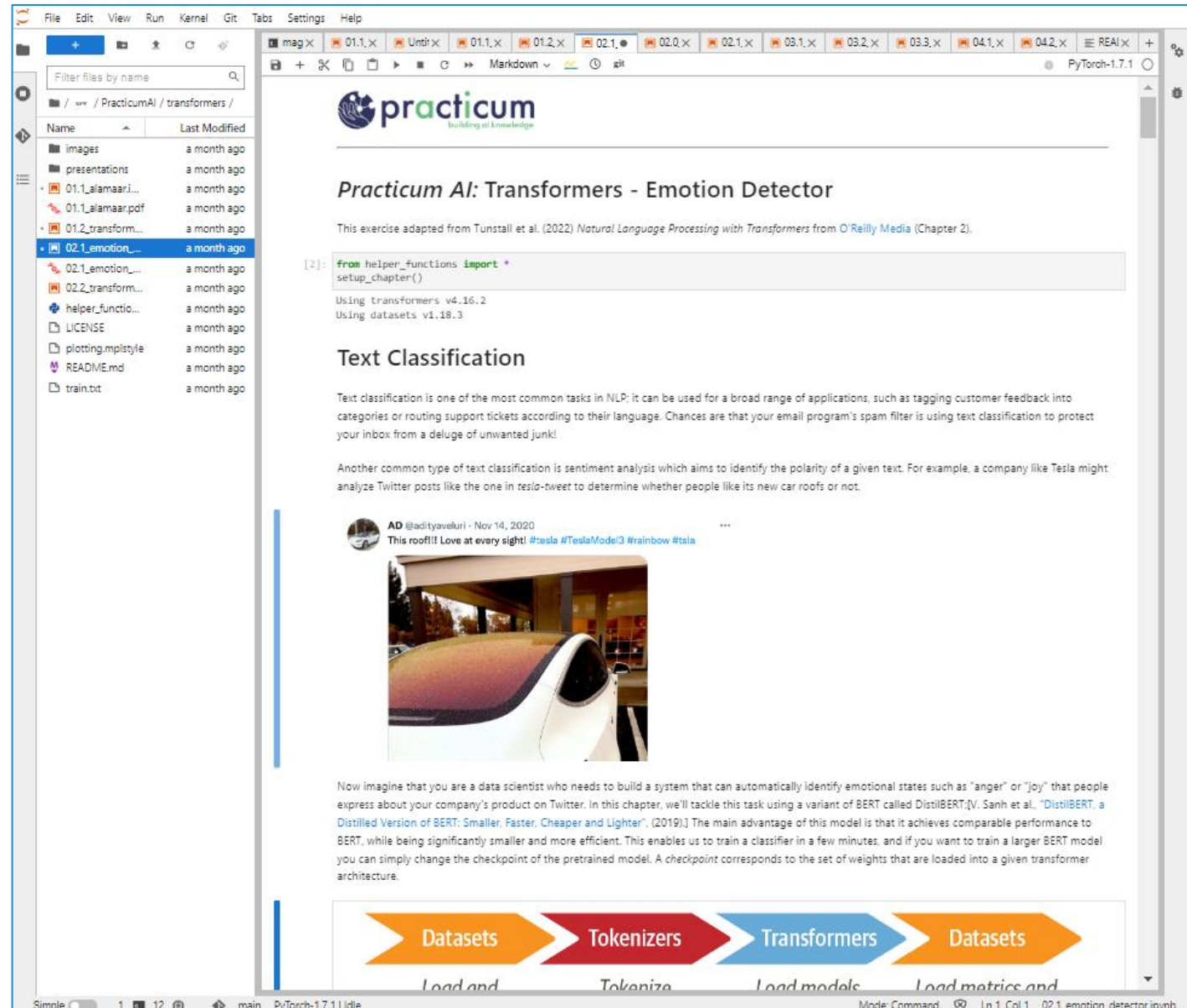
Time	Topic	Minutes
1:00	Get logged into Atlas, Welcome	10
1:10	Jupyter Notebook Basics walkthrough	20
1:30	5 min Break	5
1:35	Intro to Python	10
1:45	Python NB 01 Self-Paced	25
2:10	Python NB 01 Review	20
2:30	5 min Break	5
2:35	Pandas and Data Wrangling	10
2:45	Python NB 02 Self-Paced	25
3:10	Python NB 02 Review	20
3:30	5 min Break	5
3:35	Python NB 03 Self-Paced	15
3:50	Python NB 03 Review	10
4:00	Plotnine and Data Visualization	10
4:10	Python NB 04 Self-Paced	25
4:35	Python NB 04 Review	20
4:55	Exit Survey	5
5:00	End	

Hold on to your keyboards!





JupyterLab

A screenshot of the JupyterLab web interface. The left sidebar shows a file explorer with a list of files and folders, including 'images', 'presentations', and several notebooks like '01.1_alamaar...', '01.2_transform...', '02.1_emotion...', and '02.2_transform...'. The main area displays a notebook titled 'Practicum AI: Transformers - Emotion Detector'. The notebook content includes a title, a subtitle 'This exercise adapted from Tunstall et al. (2022) Natural Language Processing with Transformers from O'Reilly Media (Chapter 2).', a code cell with a Python import statement, and a section titled 'Text Classification' with explanatory text and a tweet image of a Tesla car. At the bottom, there is a flow diagram showing the process: Datasets (Load and) -> Tokenizers (Tokenize) -> Transformers (Load models) -> Datasets (Load metrics and). The status bar at the bottom indicates 'PyTorch-1.7.1 | Idle' and 'Mode: Command'.

Time	Topic	Minutes
1:00	Get logged into Atlas, Welcome	10
1:10	Jupyter Notebook Basics walkthrough	20
1:30	5 min Break	5
1:35	Intro to Python	10
1:45	Python NB 01 Self-Paced	25
2:10	Python NB 01 Review	20
2:30	5 min Break	5
2:35	Pandas and Data Wrangling	10
2:45	Python NB 02 Self-Paced	25
3:10	Python NB 02 Review	20
3:30	5 min Break	5
3:35	Python NB 03 Self-Paced	15
3:50	Python NB 03 Review	10
4:00	Plotnine and Data Visualization	10
4:10	Python NB 04 Self-Paced	25
4:35	Python NB 04 Review	20
4:55	Exit Survey	5
5:00	End	



5 Minute Break!

Go look at a tree!





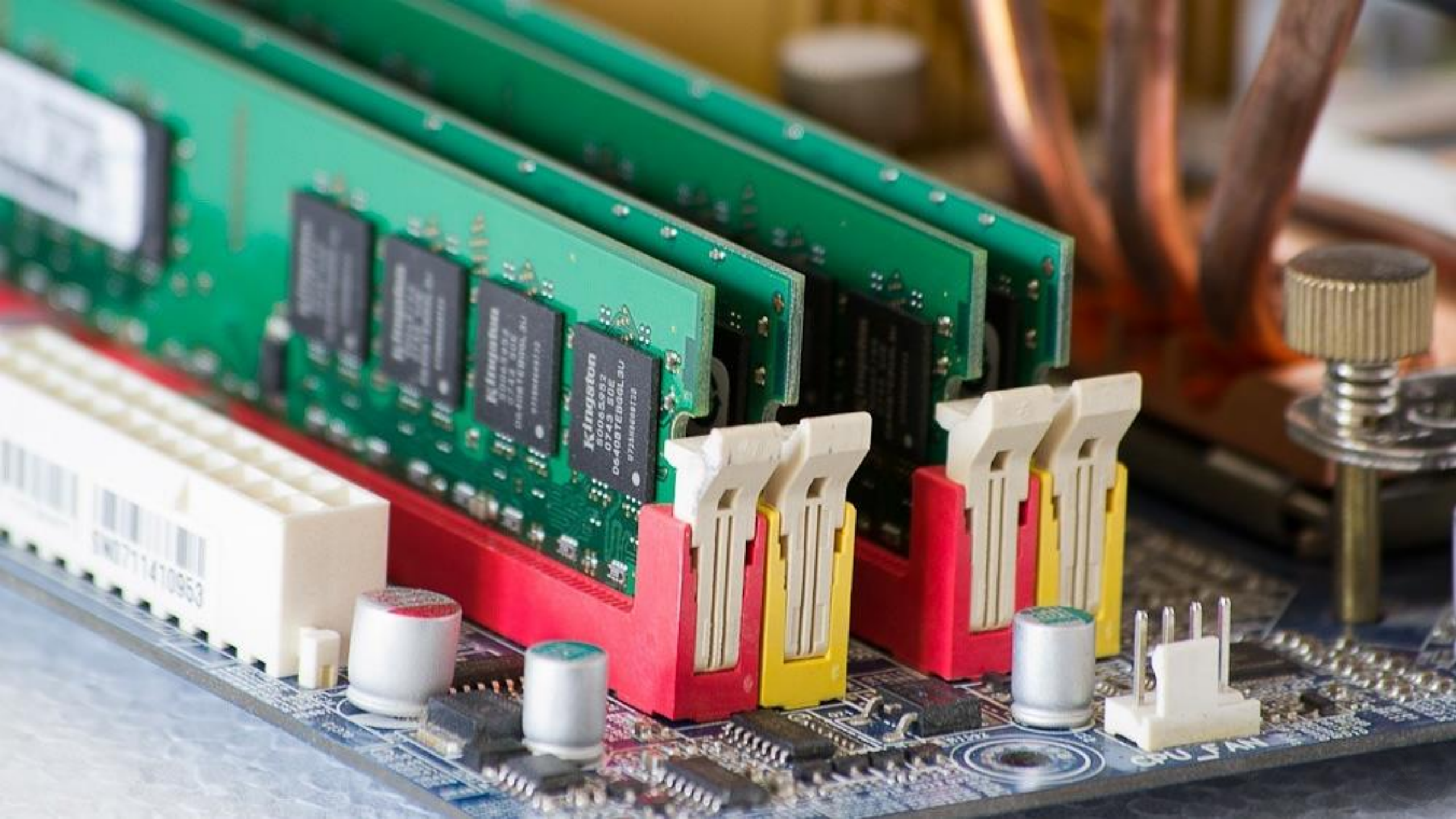
practicum
building ai knowledge

Variables



Variables





Python Variables

- Python variables are **dynamically typed**
- The data type is inferred by the data that is assigned

Pros

- Quick
- Usually correct

Cons

- May get 'interesting' results
- Slower

Variable	Type	Value
a = 123	Int	123
b = 3.1415	Float	3.1415
c = 4	int	4
d = c / 2	float	2.0
e = 'Hello!'	str	Hello



Variable Name Rules

Variable names (in Python) are case sensitive and **MUST**

- Begin with a **letter** (or an **underscore**)
- **Not** contain a space or a period (.)
- **Not** be a reserved keyword
- Be **unique** (can't be the same as another variable or function name)
- Not more than **79** Characters (if this is a problem, we need to talk!)



Reserved words

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield



Variable Naming Recommendations

Naming conventions are suggested, not required

- Make your names **intuitive, readable, concise, and consistent!**
- Begin variable names with a lower case letter (“a” not “A”)
- Use “**camel casing**” – capitalize each new “word” (**myVar, sumDollars**)

OR

- Use underscores to separate each “word” (**my_var, sum_dollars**)

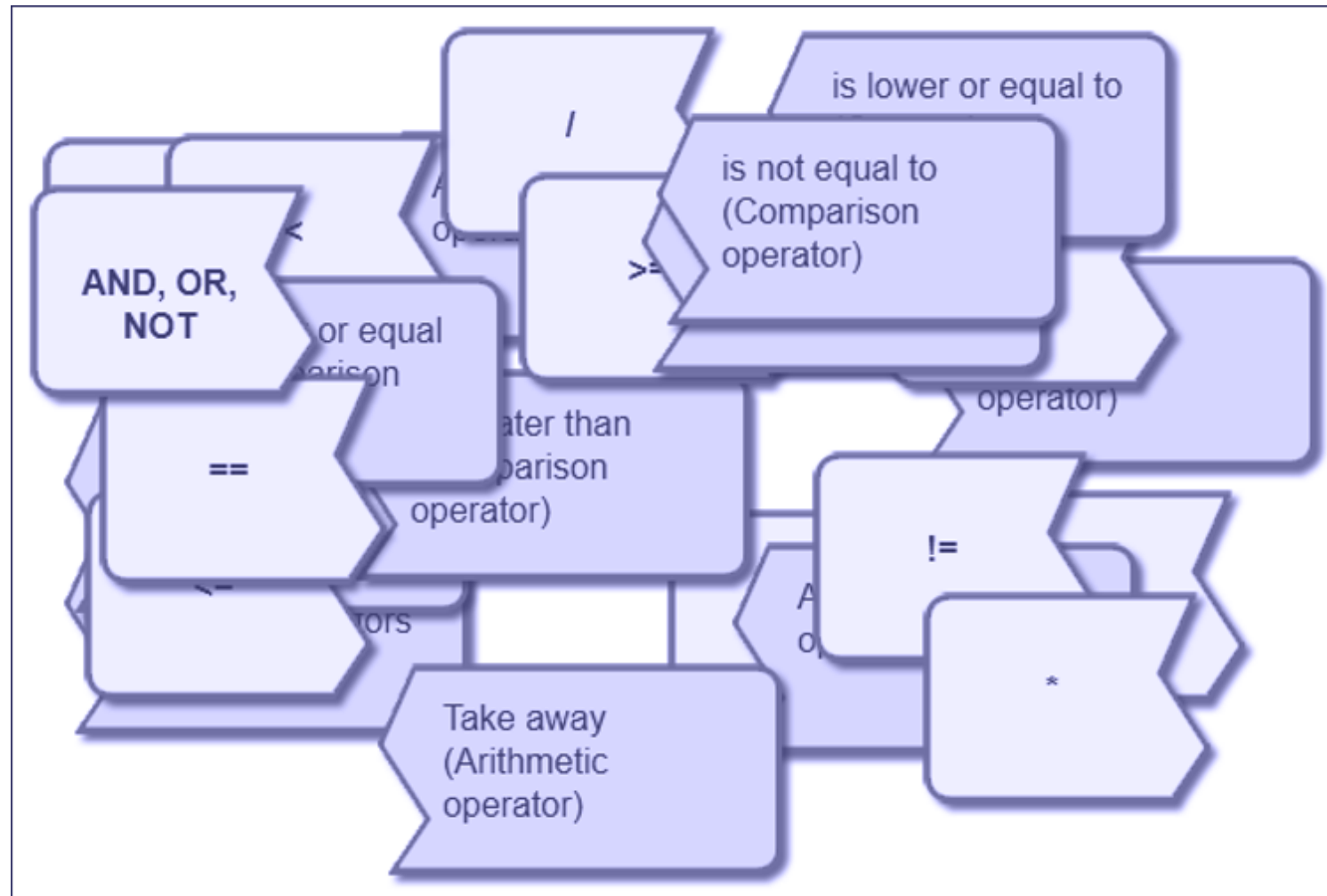




practicum
building ai knowledge

Operators





Python Arithmetic Operators

- Addition $+$
- Subtraction $-$
- Multiplication $*$
- Division $/$
- Exponent (power) $**$
($3**2$)
- Grouping $()$
- Integer (floor) Division $//$
- Modulus $%$
 - Remainder i.e. $5 \% 2 = 1$



Python Comparison Operators

- Comparison Operators compare 2 numbers or strings
- They always return a **Boolean** value (**True** or **False**)

Equal to	<code>==</code>	(note that this is different than <code>'='</code>)
----------	-----------------	--

Less Than	<code><</code>
-----------	-------------------

Greater Than	<code>></code>
--------------	-------------------

Less than or equal to	<code><=</code>
-----------------------	--------------------

Greater than or equal to	<code>>=</code>
--------------------------	--------------------

Not Equal to	<code>!=</code>	(<code>!</code> is often called "bang" and stands for "not")
--------------	-----------------	---



Other Operators

- Logical Operators

- `and` - evaluates to True if left and right sides are **BOTH** True `(5 > 2) and (90 < 100)`

- `or` - evaluates to True **EITHER** left or right sides are True `(5 < 2) or (90 < 100)`

- `not` - reverses True to False and vice versa

- Identity Operators

- `is` - returns True if both variables are the same object

- `is not` - returns True if both variables are **NOT** the same object

- Membership Operators

- `in` - returns True if specified sequence is present in the object `(x in y)`

- `not in` - returns True if specified sequence is NOT in the object `(x not in y)`



Python Assignment Operators

- Python has several augmented operators that allow assignment and operations to be done simultaneously (but no unary increment or decrement, ++ or --)

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5





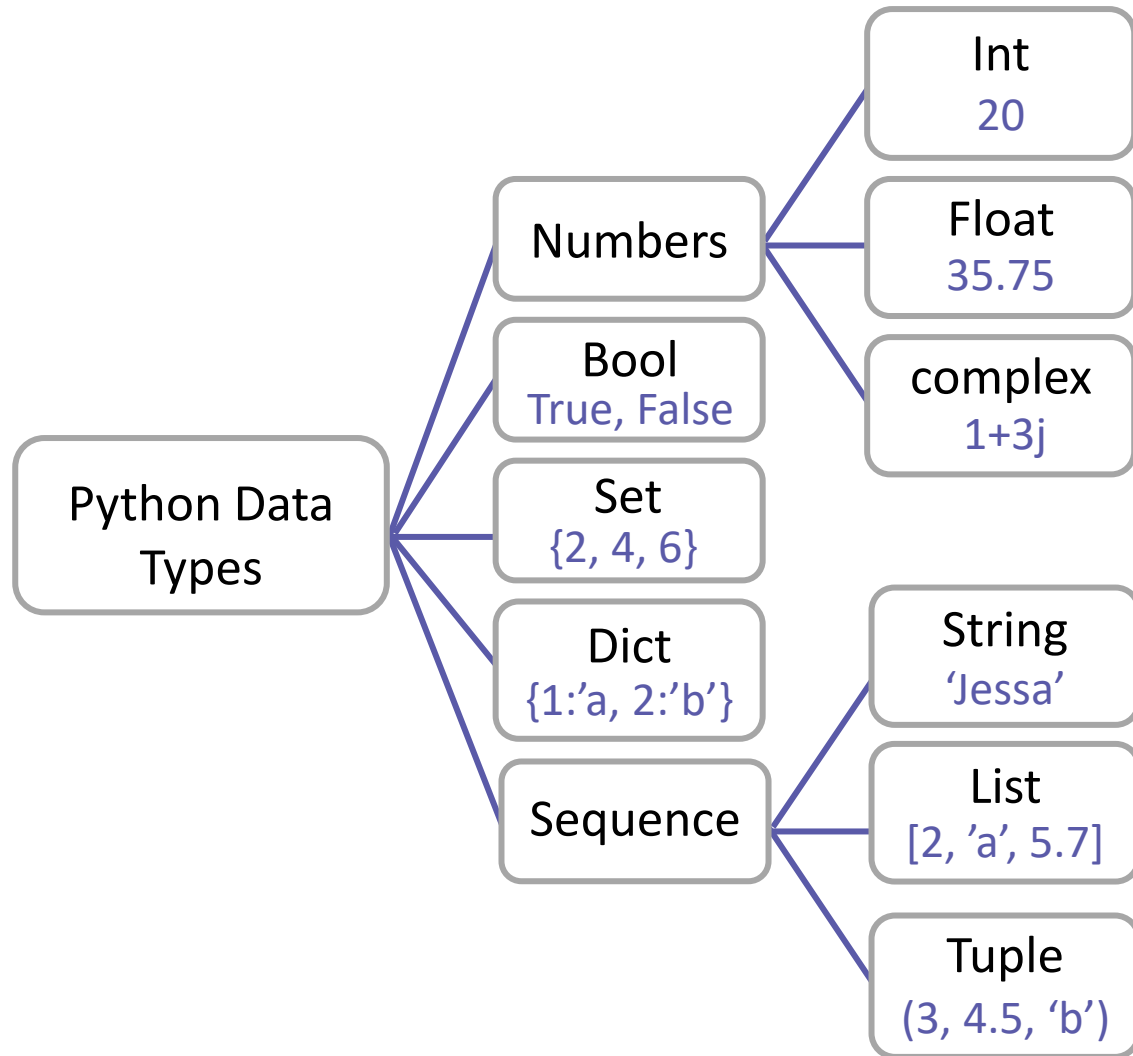
practicum
building ai knowledge

DataTypes

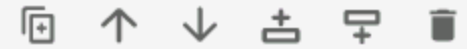


Data Types





```
[ ]: string_number = '44'  
type(string_number)
```



```
[ ]: real_integer = int(string_number)  
type(real_integer)
```

```
[ ]: real_float = float(string_number)  
type(real_float)
```

```
[ ]: print(real_integer, real_float, string_number)
```



int()	string, floating point to Integer
float()	string, integer to floating point number
str()	integer, float, list, tuple, dictionary to string
list()	string, tuple, dictionary to list
tuple()	string, list to tuple





Coding Style and Documentation





Cabernet Sauvignon



Grüner Veltliner



Pinot Noir



Riesling



Python Comments

Inline comments

- Precede with #

Use comments liberally!

- Comments are for the future you!
- They are also good for debugging, by activating or deactivating parts of your code





practicum
building ai knowledge

Functions



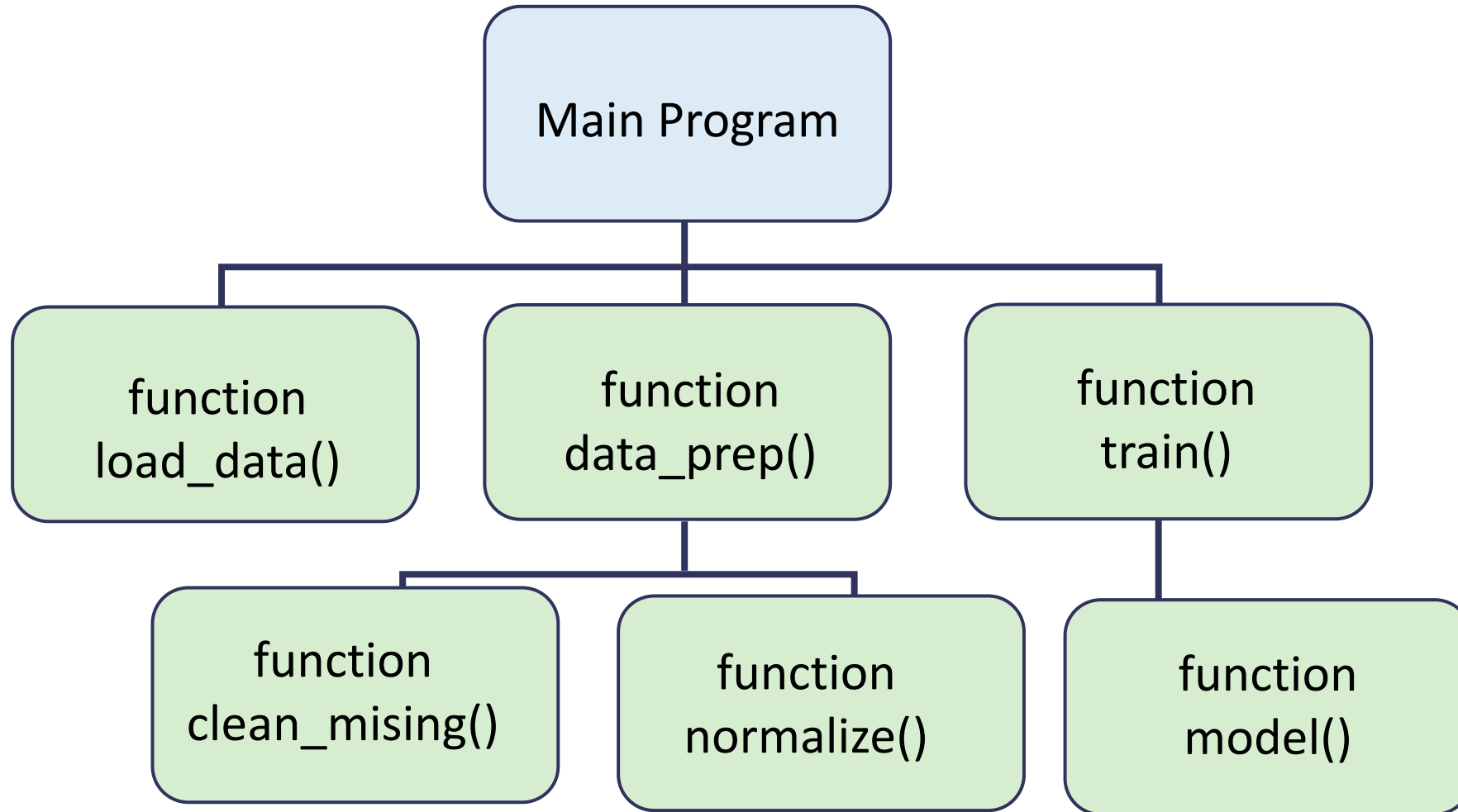
“You can think of a function as a small program inside a program. The basic idea of a function is that we write a sequence of statements and give that sequence a name. The instructions can then be executed at any point in the program by referring to the function name...”

*When a function is subsequently used in a program, we say that the definition is **called** or **invoked**.”*

John Zelle



Why use Functions?



A function is a self-contained block of code...

- It may accept inputs (**parameters**)
- It will output (return) one or more results. Sometimes, the returned value is nothing! (**None**)
- Can contain any statements found in a normal program
- We've already encountered functions: i.e., **int()**, **float()**, **range()**, and many more – these are "**built-in functions**". The functions we write ourselves are "**user-defined**" functions



Advantages of Functions

Code Reuse

Simplicity

Flexibility

Ease of Testing

Improved Collaboration



Thou shalt not repeat thyself
Thou shalt not repeat thyself
Thou shalt not repeat thyself
Thou shalt not repeat thyself



Syntax for Creating a Function

```
def <name>():  
    <statements>  
    return <values>
```

```
def hello():  
    print('Hello!')
```



Syntax for Calling a Function

<function_name>()

```
def hello():  
    print('Hello!')
```

```
hello()
```

Hello!



Function Parameters and Arguments

```
def <function_name>(param1, param2, . . .)
```



Parameters vs Arguments

Parameter



```
def hello(name):  
    print(f'Hello {name}!')
```

```
myName = input('Name? ')  
hello(myName)
```



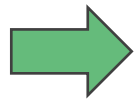
Argument



Returning Values

`return <value1>,<value2>, . . .`

```
def hello(name):  
    print(f'Hello {name}!')  
    return True
```



```
myName = input('Name? ')  
status = hello(myName)  
print(status)
```

Name? Randy
Hello Randy!
True



Call by Position or Keyword

```
def hello(name):  
    print(f'Hello {name}!')
```

```
myName = input('Name? ')
```

```
# Call by position  
hello(myName)
```

```
# Call by keyword  
hello(name = myName)
```



Time	Topic	Minutes
1:00	Get logged into Atlas, Welcome	10
1:10	Jupyter Notebook Basics walkthrough	20
1:30	5 min Break	5
1:35	Intro to Python	10
1:45	Python NB 01 Self-Paced	25
2:10	Python NB 01 Review	20
2:30	5 min Break	5
2:35	Pandas and Data Wrangling	10
2:45	Python NB 02 Self-Paced	25
3:10	Python NB 02 Review	20
3:30	5 min Break	5
3:35	Python NB 03 Self-Paced	15
3:50	Python NB 03 Review	10
4:00	Plotnine and Data Visualization	10
4:10	Python NB 04 Self-Paced	25
4:35	Python NB 04 Review	20
4:55	Exit Survey	5
5:00	End	



5 Minute Break!

Stay Hydrated!





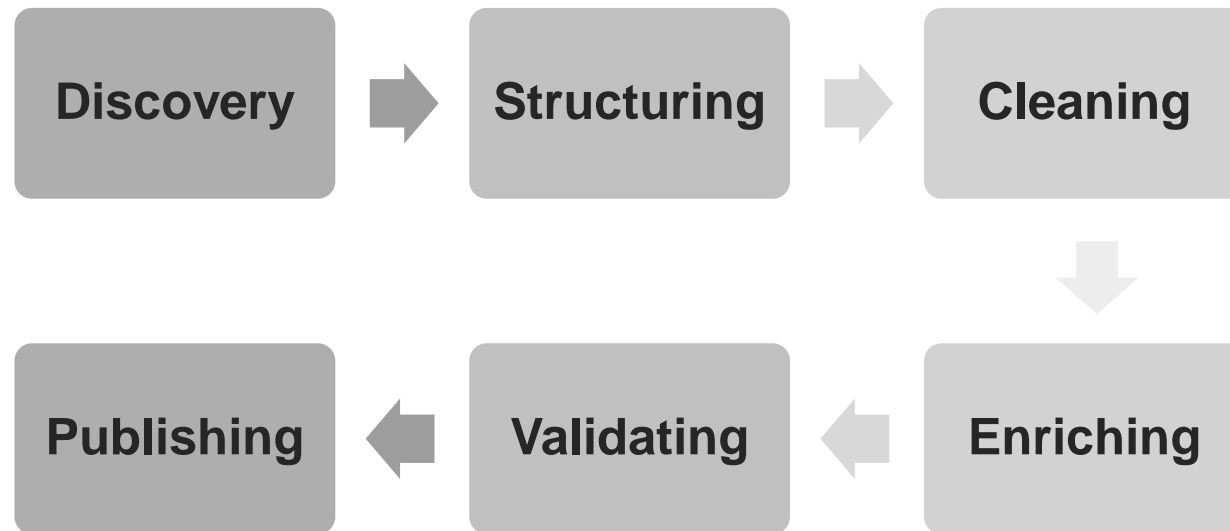
practicum
building ai knowledge

Data Wrangling



What is Data Wrangling?

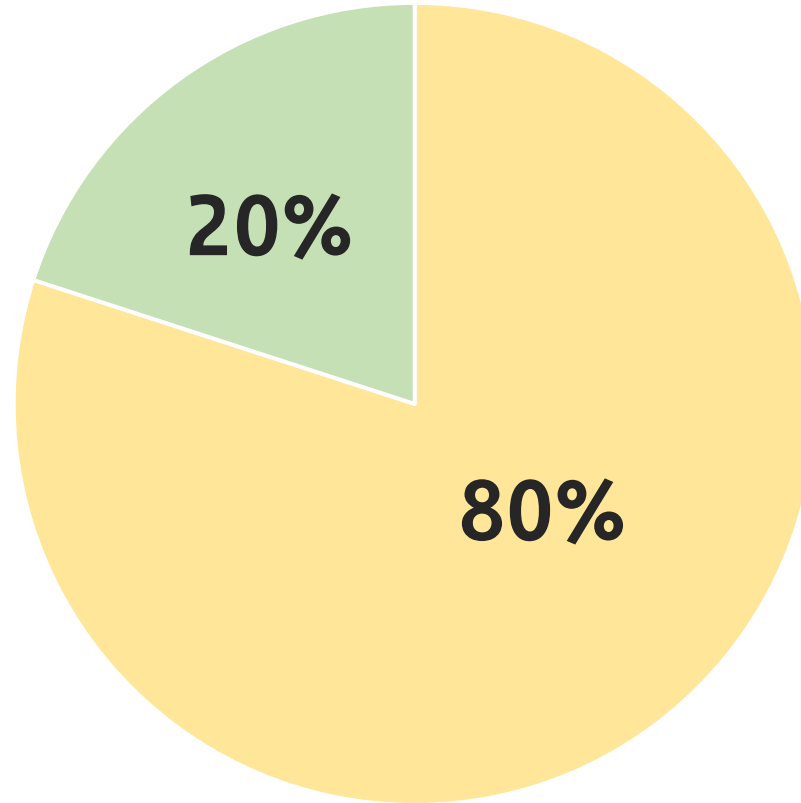
Data Wrangling is the process of gathering, collecting, and transforming raw data into another format for better understanding, decision-making, accessing, and analysis in less time.



Steps in data wrangling



Importance of Data Wrangling



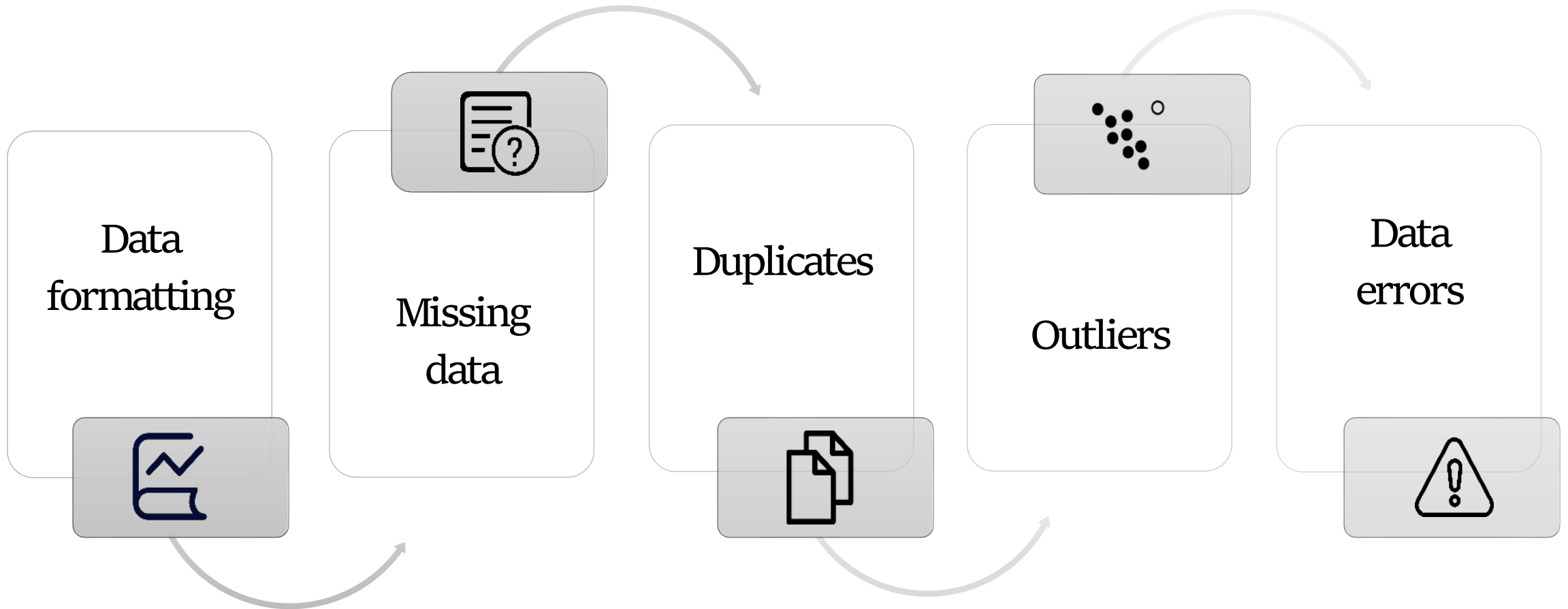
Data Exploration and Modeling



Data Wrangling



Common Data Wrangling Challenges



Data Wrangling Techniques

Cleaning

Removing or correcting data that is incorrect, incomplete, or irrelevant.

Parsing

Breaking down complex data structures into simpler, more manageable forms.

Filtering

Selecting a subset of the data based on specific criteria, such as data ranges or numerical values.

Joining

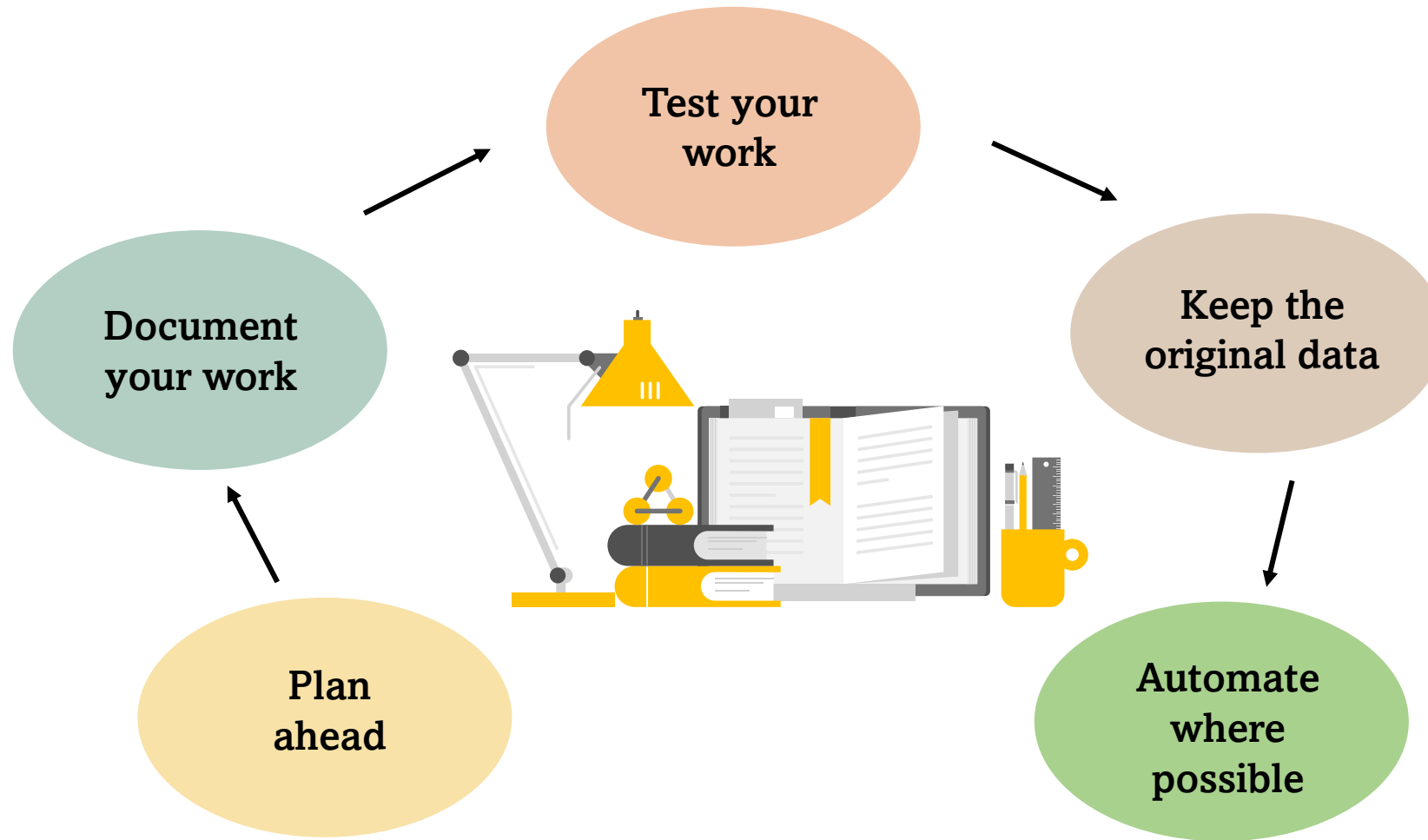
Combining data from multiple sources into a single, unified dataset.

Aggregating

Summarizing data into groups or categories, such as calculating averages or counts.



Data Wrangling Best Practices



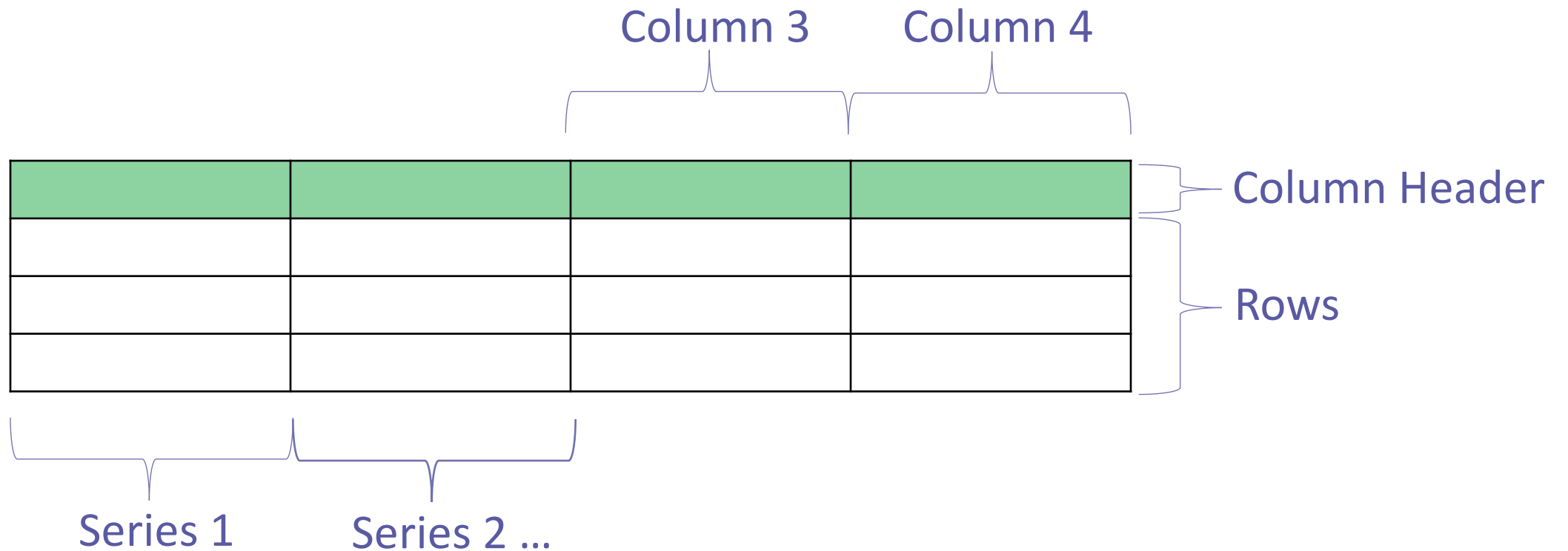


Pandas





Anatomy of the Pandas DataFrame



Creating a DataFrame from a File

```
gapminder = pd.read_csv("gapminder.csv")
```

Continent	Country	gdpPerCap_2002	gdpPerCap_2007
Americas	Argentina	8797	12779
Asia	South Korea	19233	23348
Asia	Japan	28604	31656



Adding a New Column

```
gapminder['gdpChange'] = gapminder['gdpPerCap_2007'] - gapminder['gdpPerCap_2002']
```

Continent	Country	gdpPerCap_2002	gdpPerCap_2007	gdpChange
Americas	Argentina	8797	12779	3982
Asia	South Korea	19233	23348	4115
Asia	Japan	28604	31656	3052



Dropping a Column

```
gapminder = gapminder.drop('gdpChange', axis = 1)
```

Continent	Country	gdpPerCap_2002	gdpPerCap_2007	gdpChange
Americas	Argentina	8797	12779	3982
Asia	South Korea	19233	23348	4115
Asia	Japan	28604	31656	3052



Time	Topic	Minutes
1:00	Get logged into Atlas, Welcome	10
1:10	Jupyter Notebook Basics walkthrough	20
1:30	5 min Break	5
1:35	Intro to Python	10
1:45	Python NB 01 Self-Paced	25
2:10	Python NB 01 Review	20
2:30	5 min Break	5
2:35	Pandas and Data Wrangling	10
2:45	Python NB 02 Self-Paced	25
3:10	Python NB 02 Review	20
3:30	5 min Break	5
3:35	Python NB 03 Self-Paced	15
3:50	Python NB 03 Review	10
4:00	Plotnine and Data Visualization	10
4:10	Python NB 04 Self-Paced	25
4:35	Python NB 04 Review	20
4:55	Exit Survey	5
5:00	End	



5 Minute Break!

Get up and stretch!





Basic Data Stats & Visualization



Basic Dataframe Statistics

	People	Age	Weight	Height
0	Ann	21	55	160
1	Brandon	12	35	135
2	Chen	32	77	170
3	David	45	68	165
4	Emily	37	70	173
5	Farook	18	60	168
6	Gagen	28	72	175
7	Hamish	52	69	159
8	Imran	5	18	105
9	Joseph	40	65	171
10	Katherine	48	82	155
11	Lily	15	48	158

```
df['Age'].count()
```

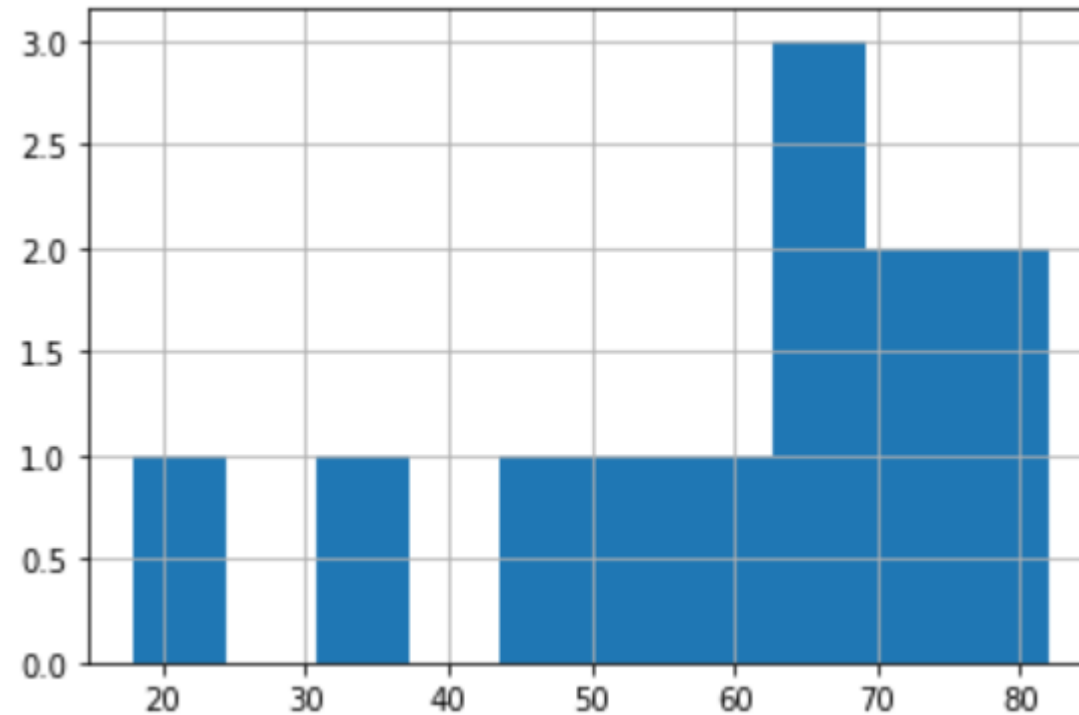
```
df['Height'].mean()
```

```
df.describe()
```

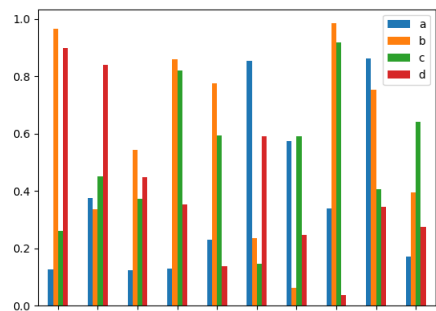


Histogram

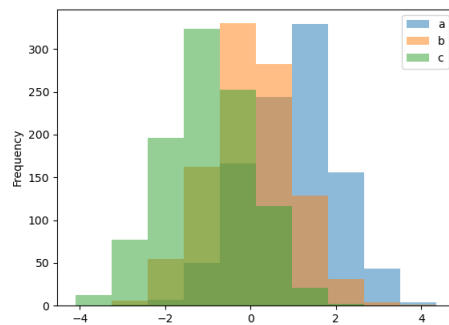
```
df['Weight'].hist()
```



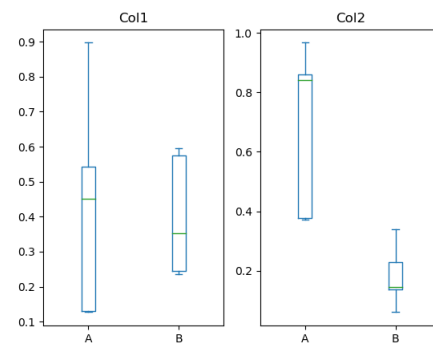
Pandas Chart Types



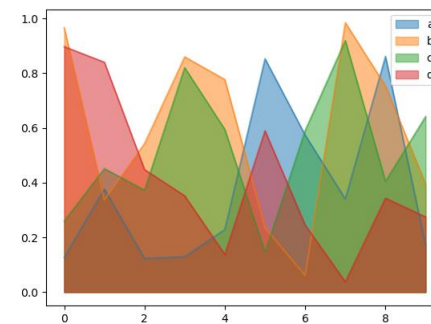
bar



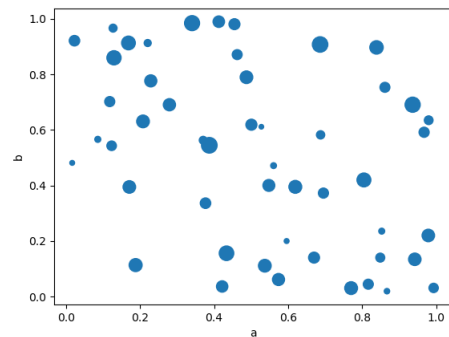
hist



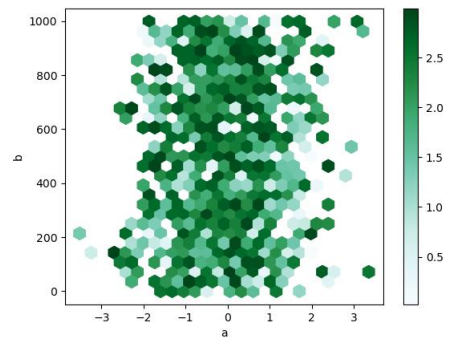
box



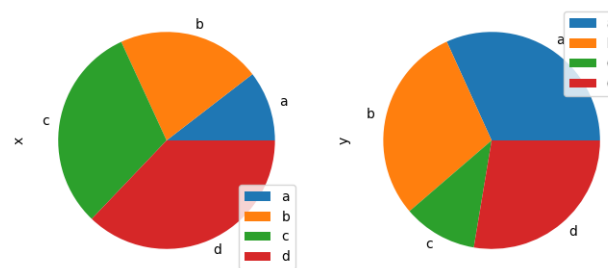
area



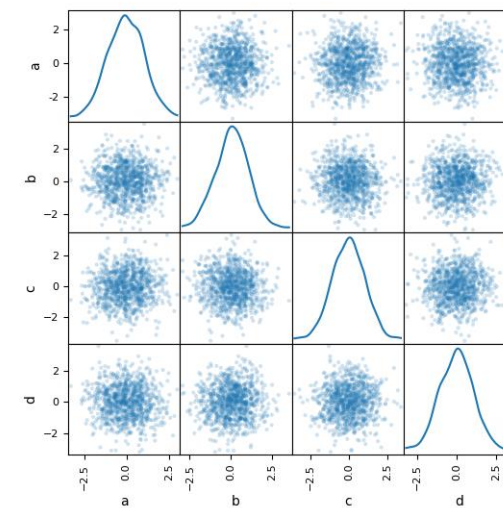
scatter



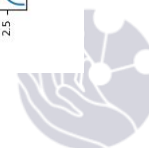
hexbin



pie

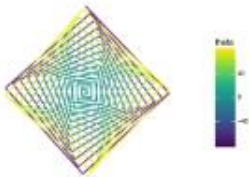


scatter_matrix

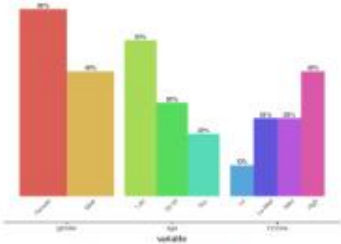




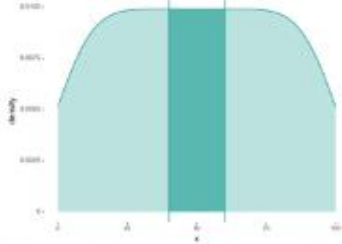
Gallery



Spiral Animation



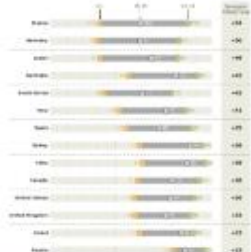
Two Variable Bar Plot



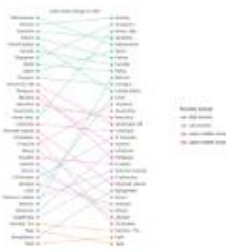
Shading a Region under a Density Curve



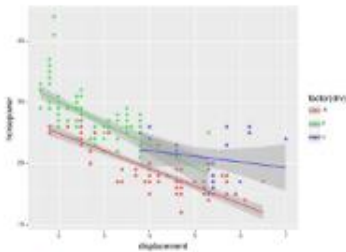
The Political Territories of Westeros



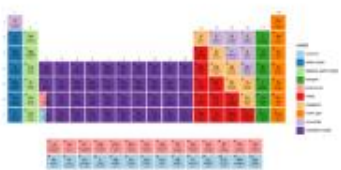
Ranges of Similar Variables



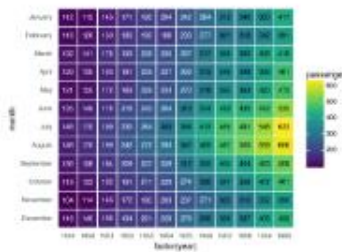
Change in Rank



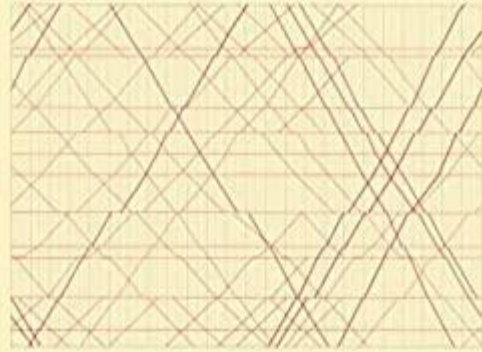
Smoothed conditional means



Periodic Table of Elements



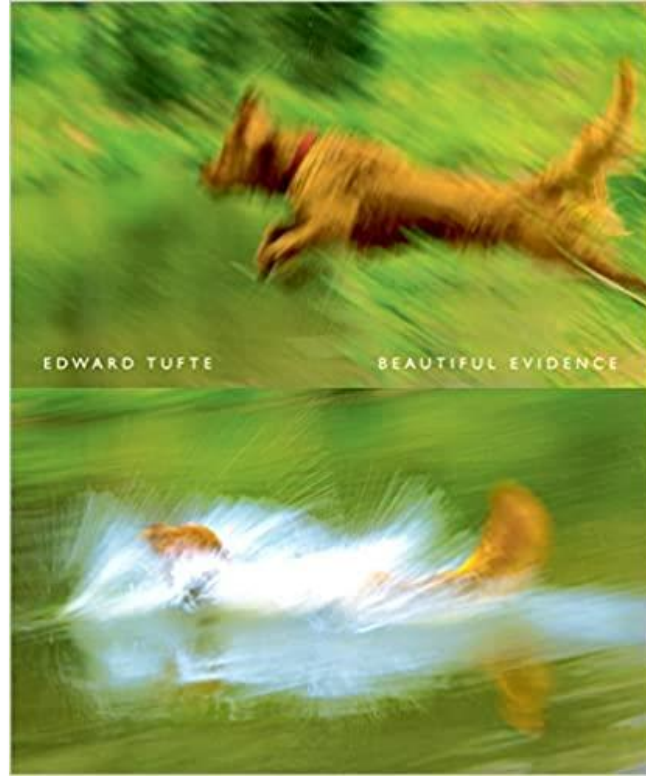
Annotated Heatmap



SECOND EDITION

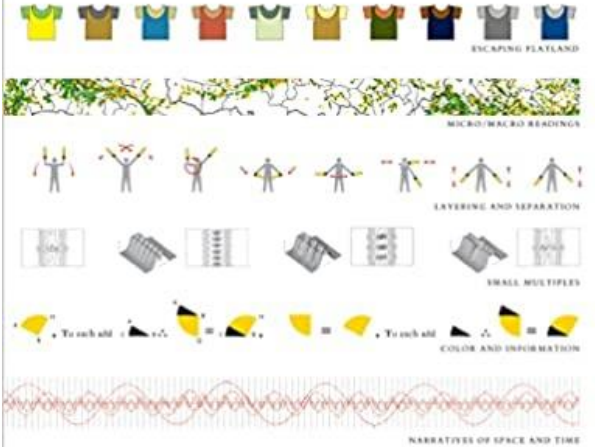
The Visual Display of Quantitative Information

EDWARD R. TUFTE



Edward R. Tufte

Envisioning Information





You Made It!!!

Please help us improve our courses by taking this short survey:

https://ufl.qualtrics.com/jfe/form/SV_78QzYsgDWoC7ksS

