DATA VIZ WITH PYTHON

LEARNING OUTCOMES

LO1: apply the **data analysis workflow** to solve a problem
LO2: create **visuals** using **Python** programming
LO3: **communicate** your findings using **data viz**

AGENDA

1. Python 101
2. Source your data
3. Explore your data
4. Prepare your data
5. Visualize with Plotly

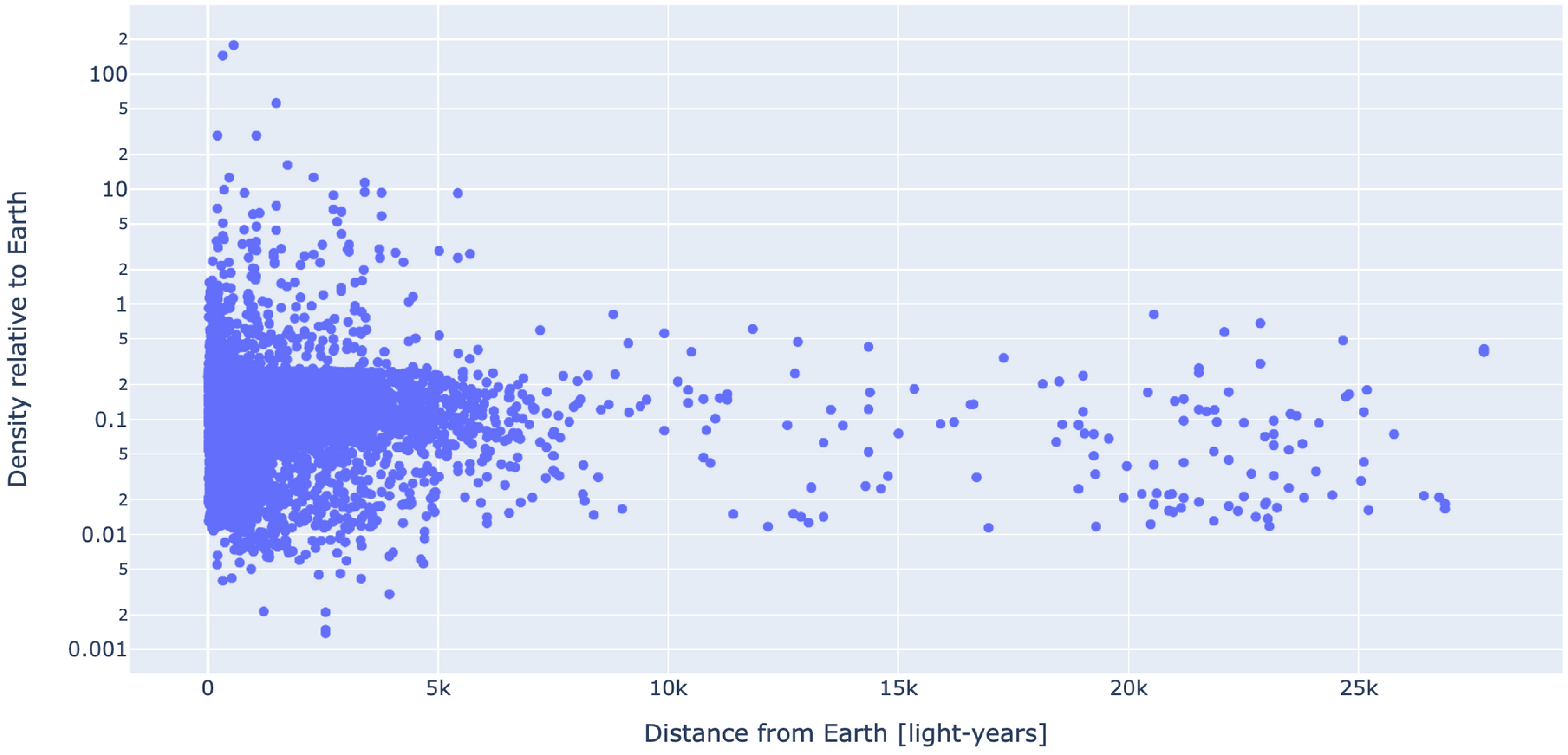
why would we want to **visualize** data?

e.g. “Is there a relationship between an exoplanet’s density and its distance from Earth?”

Could try to answer this question looking at the raw data...

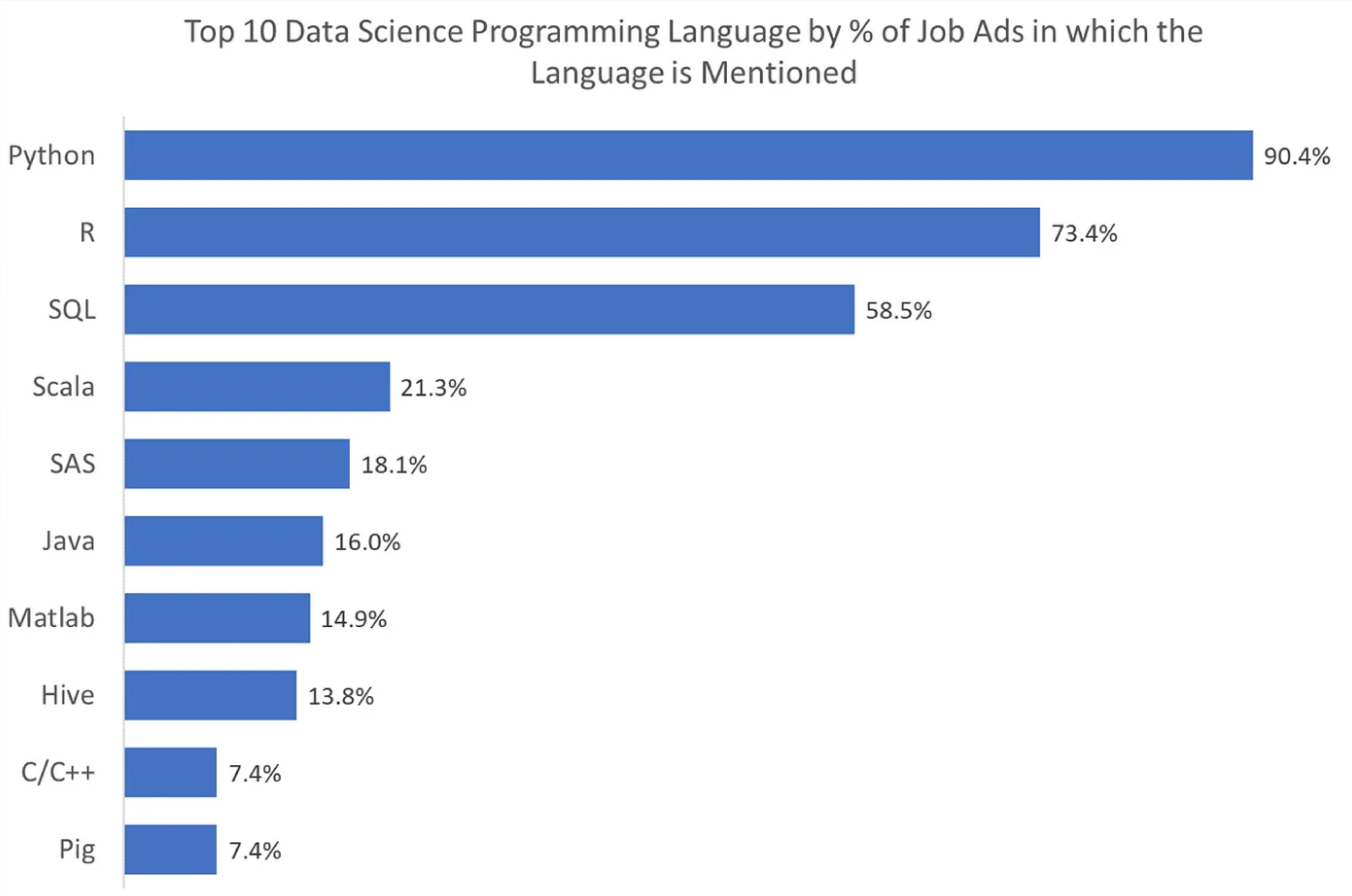
	name	distance	stellar_magnitude	planet_type	discovery_year	mass_multiplier	mass_wrt
0	11 Comae Berenices b	304.0	4.72307	Gas Giant	2007	19.40000	Jupiter
1	11 Ursae Minoris b	409.0	5.01300	Gas Giant	2009	14.74000	Jupiter
2	14 Andromedae b	246.0	5.23133	Gas Giant	2008	4.80000	Jupiter
3	14 Herculis b	58.0	6.61935	Gas Giant	2002	8.13881	Jupiter
4	16 Cygni B b	69.0	6.21500	Gas Giant	1996	1.78000	Jupiter
...	...	...	...	...	...	...	...
5245	XO-7 b	764.0	10.52100	Gas Giant	2019	0.70900	Jupiter
5246	YSES 2 b	357.0	10.88500	Gas Giant	2021	6.30000	Jupiter
5247	YZ Ceti b	12.0	12.07400	Terrestrial	2017	0.70000	Earth
5248	YZ Ceti c	12.0	12.07400	Super Earth	2017	1.14000	Earth
5249	YZ Ceti d	12.0	12.07400	Super Earth	2017	1.09000	Earth

Or you could answer the question using a simple graph



Why using Python to create graphs?

Python is especially good at handling (very) large amounts of **data**


Top 10 Data Science Programming Language by % of Job Ads in which the Language is Mentioned
Python
90.4%
R
73.4%
SQL
58.5%
Scala
21.3%
SAS
18.1%
Java
16.0%
Matlab
14.9%
Hive
13.8%
C/C++
7.4%
Pig
7.4%

PYTHON 101

(quick recap)

strings are for literal text

strings are for literal text

"I am 29 years old"

"5684297"

strings can be added together

```
"Hello " + "world!"
```

strings can be added together

```
"Hello " + "world!"
```

```
"Hello world!"
```

integers are for whole numbers

integers are for whole numbers

15

-200

0

any math computation can be performed on **integers**

26 + 58

84

floats are for decimal numbers

floats are for decimal numbers

`3.14`

`-156.52628`

`0.000`

any math computation can be performed on **floats**

```
26.5 / 3.0
```

```
8.83333
```

booleans are for **T**rue or **F**alse statements

booleans are for **T**rue or **F**alse statements

```
1 > 5
```

booleans are for **T**rue or **F**alse statements

```
1 > 5
```

Is 1 greater than 5?

booleans are for **T**rue or **F**alse statements

```
1 > 5
```

```
False
```

booleans are for **T**rue or **F**alse statements

```
36 == 36
```

booleans are for **T**rue or **F**alse statements

```
36 == 36
```

Is 36 equal to 36?

booleans are for **T**rue or **F**alse statements

```
36 == 36
```

```
True
```

all types of data can be stored in memory using

variables

```
my_name = "Julie"
```

variable name
(box label)

```
my_name = "Julie"
```

content

you access **variables** by calling their names

```
my_name = "Julie"
```

```
"My name is " + my_name
```

```
my_name = "Julie"
"My name is " + my_name
"My name is Julie"
```

you can also **update** the content of your variables

```
my_age = 29
my_age = my_age + 2
print(my_age)
```

```
my_age = 29
my_age = my_age + 2
print(my_age)
31
```

methods are pieces of code that have **already been written** by someone else

```
my_name = "Julie"
```

```
my_name = "Julie"
```

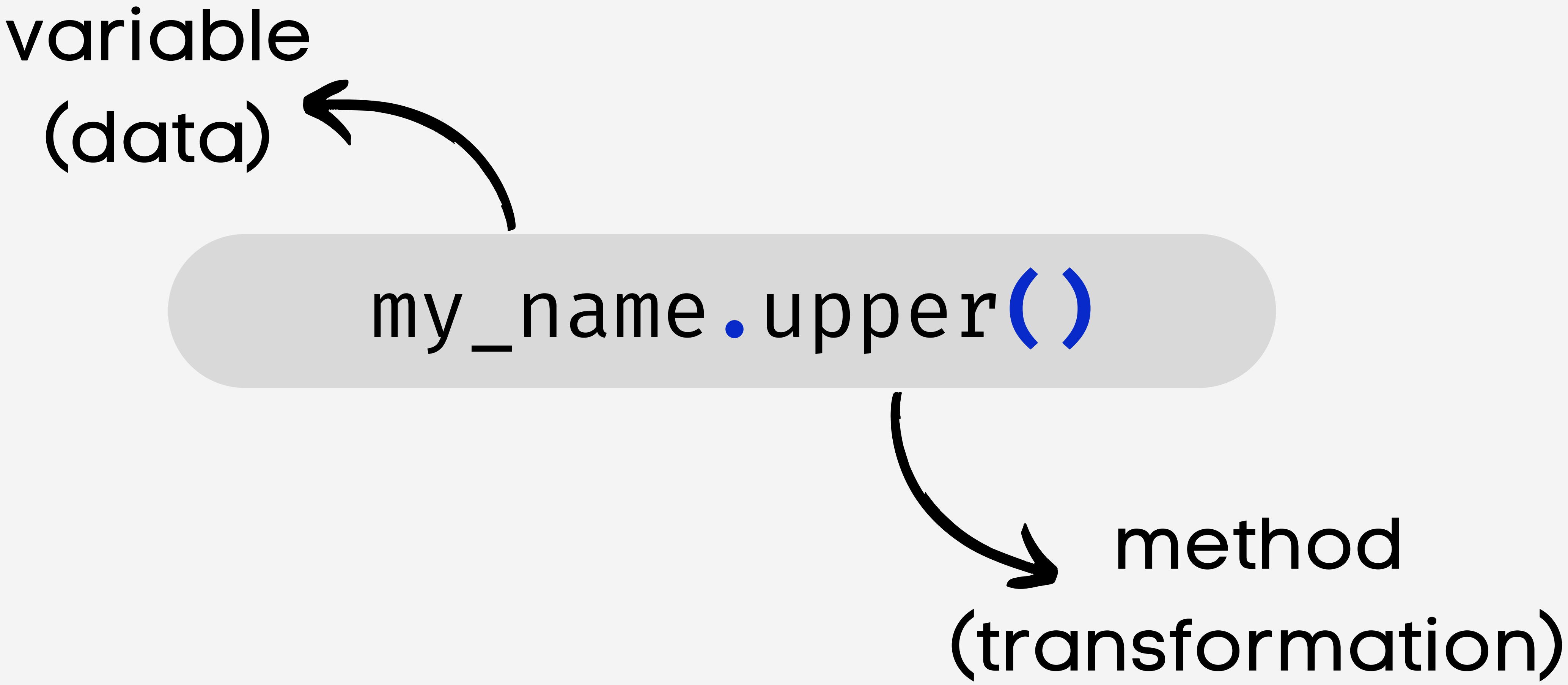
how can I write "Julie" in all uppercase?

Option 1: update manually

```
my_name = "JULIE"
```

Option 2: use a method!

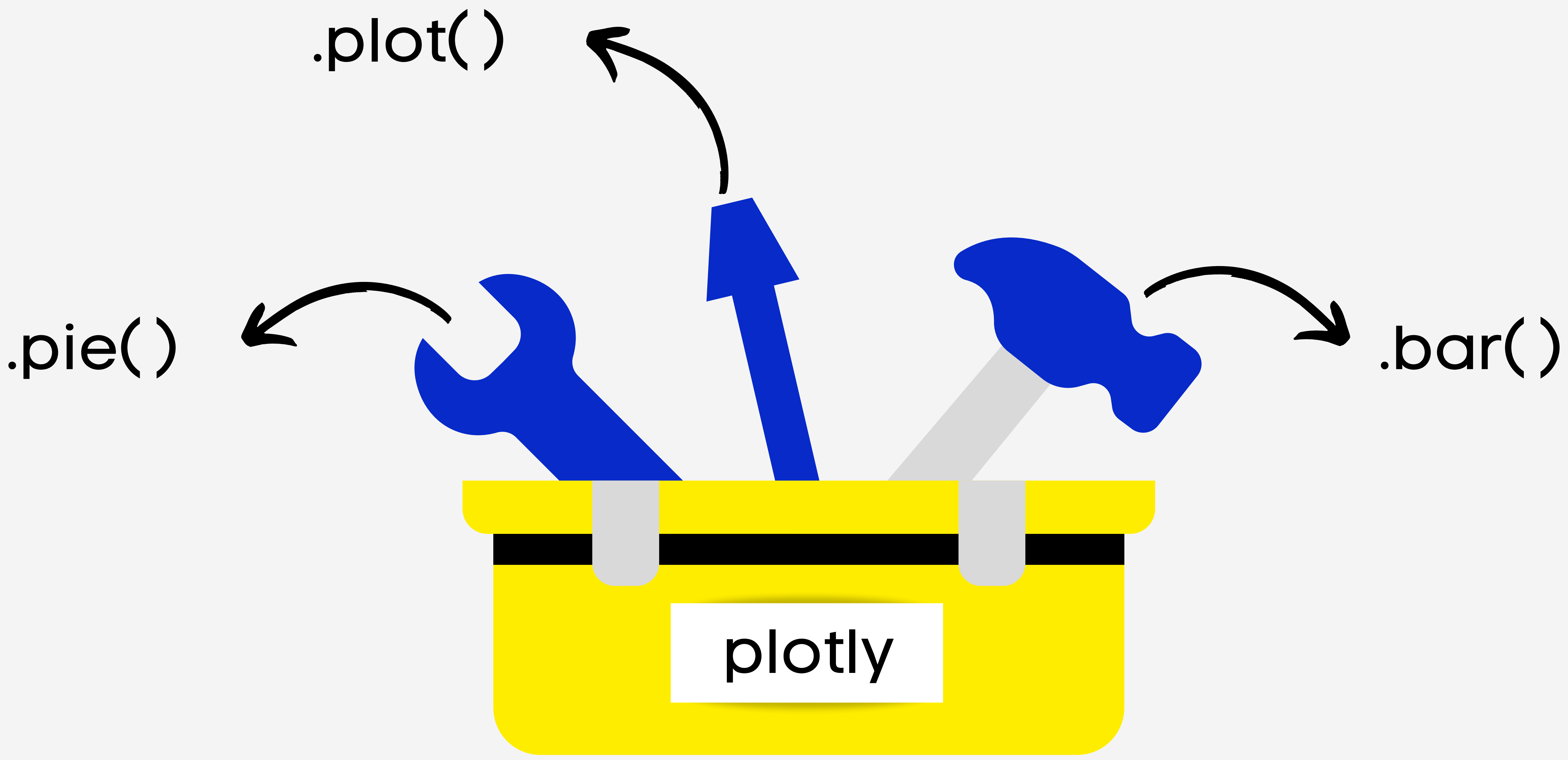
```
my_name.upper()
```


variable
(data)
my_name.upper()
method
(transformation)

```
my_name.upper()
```

```
"JULIE"
```

Python is so powerful because of its **community** that everyday develops **new methods** for people like you and me to use there are methods for almost **everything** methods are often grouped in **libraries** for example, the **Plotly library** has thousands of **methods** for data visualization


.plot( )
.pie( )
.bar( )
plotly

most methods need to be **imported** inside your workspace

```
import plotly
```

```python
from plotly import pie
```

```python
from plotly import pie
```

from the Plotly library, import the method pie

```python
from plotly import pie
```

from the toolbox Plotly, get the pie( ) tool today we'll work with two libraries, **pandas** and **plotly express**

SOURCE YOUR DATA

first and foremost, we need **data**

there are lots of places to look for data

local file on your laptop
(e.g. Excel spreasheet, CSV files, ...)

online dataset-sharing platforms

(e.g. Kaggle, Google Dataset Search, ...)

other data sources
(e.g. databases, APIs, web scraping, data extraction from PDFs, ...)

today we're going to work with a dataset from

Kaggle about **exoplanets**

let's import it inside our **Jupyter Notebook** great time to be introduced to **pandas**

MEET PANDAS

pandas is THE data analysis **library** in Python

pandas

let's first **import** pandas into our workspace

let's first **import** pandas into our workspace

```
import pandas
```

let's first **import** pandas into our workspace

```python
import pandas
```

```python
import pandas as pd
```

let's first **import** pandas into our workspace

```
import pandas
```

```
import pandas as pd
```

nickname
of our choice

we can now use pandas to import the CSV file using the **.read_csv( )** method we can now use pandas to import my CSV file using the **.read_csv( )** method

```
pd.read_csv("path/to/our/file")
```

we can now use pandas to import my CSV file using the **.read_csv( )** method

```
pd.read_csv("path/to/our/file")
```

aka pandas

store it inside a variable

```
my_df = pd.read_csv("path/to/our/file")
```

dataframes are the main data types used by pandas

	name	distance	stellar_magnitude	planet_type	discovery_year	mass_multiplier	mass_wrt
0	11 Comae Berenices b	304.0	4.72307	Gas Giant	2007	19.40000	Jupiter
1	11 Ursae Minoris b	409.0	5.01300	Gas Giant	2009	14.74000	Jupiter
2	14 Andromedae b	246.0	5.23133	Gas Giant	2008	4.80000	Jupiter
3	14 Herculis b	58.0	6.61935	Gas Giant	2002	8.13881	Jupiter
4	16 Cygni B b	69.0	6.21500	Gas Giant	1996	1.78000	Jupiter
...	...	...	...	...	...	...	...
5245	XO-7 b	764.0	10.52100	Gas Giant	2019	0.70900	Jupiter
5246	YSES 2 b	357.0	10.88500	Gas Giant	2021	6.30000	Jupiter
5247	YZ Ceti b	12.0	12.07400	Terrestrial	2017	0.70000	Earth
5248	YZ Ceti c	12.0	12.07400	Super Earth	2017	1.14000	Earth
5249	YZ Ceti d	12.0	12.07400	Super Earth	2017	1.09000	Earth

5250 rows × 13 columns

	name	distance	stellar_magnitude	planet_type	discovery_year	mass_multiplier	mass_wrt
0	11 Comae Berenices b	304.0	4.72307	Gas Giant	2007	19.40000	Jupiter
1	11 Ursae Minoris b	409.0	5.01300	Gas Giant	2009	14.74000	Jupiter
2	14 Andromedae b	246.0	5.23133	Gas Giant	2008	4.80000	Jupiter
3	14 Herculis b	58.0	6.61935	Gas Giant	2002	8.13881	Jupiter
4	16 Cygni B b	69.0	6.21500	Gas Giant	1996	1.78000	Jupiter
...	...	...	...	...	...	...	...
5245	XO-7 b	764.0	10.52100	Gas Giant	2019	0.70900	Jupiter
5246	YSES 2 b	357.0	10.88500	Gas Giant	2021	6.30000	Jupiter
5247	YZ Ceti b	12.0	12.07400	Terrestrial	2017	0.70000	Earth
5248	YZ Ceti c	12.0	12.07400	Super Earth	2017	1.14000	Earth
5249	YZ Ceti d	12.0	12.07400	Super Earth	2017	1.09000	Earth

5250 rows × 13 columns

row index

	name	distance	stellar_magnitude	planet_type	discovery_year	mass_multiplier	mass_wrt
0	11 Comae Berenices b	304.0	4.72307	Gas Giant	2007	19.40000	Jupiter
1	11 Ursae Minoris b	409.0	5.01300	Gas Giant	2009	14.74000	Jupiter
2	14 Andromedae b	246.0	5.23133	Gas Giant	2008	4.80000	Jupiter
3	14 Herculis b	58.0	6.61935	Gas Giant	2002	8.13881	Jupiter
4	16 Cygni B b	69.0	6.21500	Gas Giant	1996	1.78000	Jupiter
...	...	...	...	...	...	...	...
5245	XO-7 b	764.0	10.52100	Gas Giant	2019	0.70900	Jupiter
5246	YSES 2 b	357.0	10.88500	Gas Giant	2021	6.30000	Jupiter
5247	YZ Ceti b	12.0	12.07400	Terrestrial	2017	0.70000	Earth
5248	YZ Ceti c	12.0	12.07400	Super Earth	2017	1.14000	Earth
5249	YZ Ceti d	12.0	12.07400	Super Earth	2017	1.09000	Earth

5250 rows × 13 columns

	name	distance	stellar_magnitude	planet_type	discovery_year	mass_multiplier	mass_wrt
0	11 Comae Berenices b	304.0	4.72307	Gas Giant	2007	19.40000	Jupiter
1	11 Ursae Minoris b	409.0	5.01300	Gas Giant	2009	14.74000	Jupiter
2	14 Andromedae b	246.0	5.23133	Gas Giant	2008	4.80000	Jupiter
3	14 Herculis b	58.0	6.61935	Gas Giant	2002	8.13881	Jupiter
4	16 Cygni B b	69.0	6.21500	Gas Giant	1996	1.78000	Jupiter
...	...	...	...	...	...	...	...
5245	XO-7 b	764.0	10.52100	Gas Giant	2019	0.70900	Jupiter
5246	YSES 2 b	357.0	10.88500	Gas Giant	2021	6.30000	Jupiter
5247	YZ Ceti b	12.0	12.07400	Terrestrial	2017	0.70000	Earth
5248	YZ Ceti c	12.0	12.07400	Super Earth	2017	1.14000	Earth
5249	YZ Ceti d	12.0	12.07400	Super Earth	2017	1.09000	Earth

5250 rows × 13 columns

column name

	name	distance	stellar_magnitude	planet_type	discovery_year	mass_multiplier	mass_wrt
0	11 Comae Berenices b	304.0	4.72307	Gas Giant	2007	19.40000	Jupiter
1	11 Ursae Minoris b	409.0	5.01300	Gas Giant	2009	14.74000	Jupiter
2	14 Andromedae b	246.0	5.23133	Gas Giant	2008	4.80000	Jupiter
3	14 Herculis b	58.0	6.61935	Gas Giant	2002	8.13881	Jupiter
4	16 Cygni B b	69.0	6.21500	Gas Giant	1996	1.78000	Jupiter
...	...	...	...	...	...	...	...
5245	XO-7 b	764.0	10.52100	Gas Giant	2019	0.70900	Jupiter
5246	YSES 2 b	357.0	10.88500	Gas Giant	2021	6.30000	Jupiter
5247	YZ Ceti b	12.0	12.07400	Terrestrial	2017	0.70000	Earth
5248	YZ Ceti c	12.0	12.07400	Super Earth	2017	1.14000	Earth
5249	YZ Ceti d	12.0	12.07400	Super Earth	2017	1.09000	Earth

5250 rows × 13 columns

how can I retrieve a single **column?**

how can I retrieve a single **column**?

```
my_df["column_name"]
```

how can I retrieve a single **column**?

```
my_df["column_name"]
```

variable name

how can I retrieve multiple **columns**?

how can I retrieve multiple **columns**?

```
my_df[["column_name_1", "column_name_2"]]
```

how can I **filter** my dataframe?

say I only want to get the exoplanets that have been detected using the "transit" method?

I can use **boolean indexing**

I can use **boolean indexing**

```
my_df[my_df["detection_method"] == "transit]
```

I can use **boolean indexing**

```
my_df[my_df["detection_method"] == "transit"]
```

"in my dataframe, retrieve all the rows where the **detection method** is equal to **transit**"

EXPLORE YOUR DATA

how big is your dataframe?

how big is your dataframe?

```
my_df.shape
```

what columns are inside your dataframe?

what columns are inside your dataframe?

```
my_df.columns
```

what type of data is in your dataframe?

what type of data is in your dataframe?

```
my_df.dtypes
```

how far is the furthest exoplanet?

how far is the furthest exoplanet?

```
my_df["distance"].max()
```

I can store the value inside a variable

```
max_distance = my_df["distance"].max()
```

which exoplanet is the furthest?

which exoplanet is the furthest?

```
my_df[my_df["distance"] == max_distance]
```

which exoplanet is the furthest?

```
my_df[my_df["distance"] == max_distance]
```

"in my dataframe, retrieve all the rows where the **distance** is equal to **max_distance**" how many different detection methods are there?

how many different detection methods are there?

```
my_df["detection_method"].unique()
```

which detection method is the most common?

which detection method is the most common?

```
my_df["detection_method"].value_counts()
```

what is the **average** distance of each **type of exoplanet**?

what is the **average** distance of each **type of exoplanet**?

```
my_df.groupby("planet_type").mean()
```

what is the **average** distance of each **type of exoplanet**?

```
my_df.groupby("planet_type").mean()
```

"in my dataframe, for each **planet type** compute the **mean**"

.shape

.columns

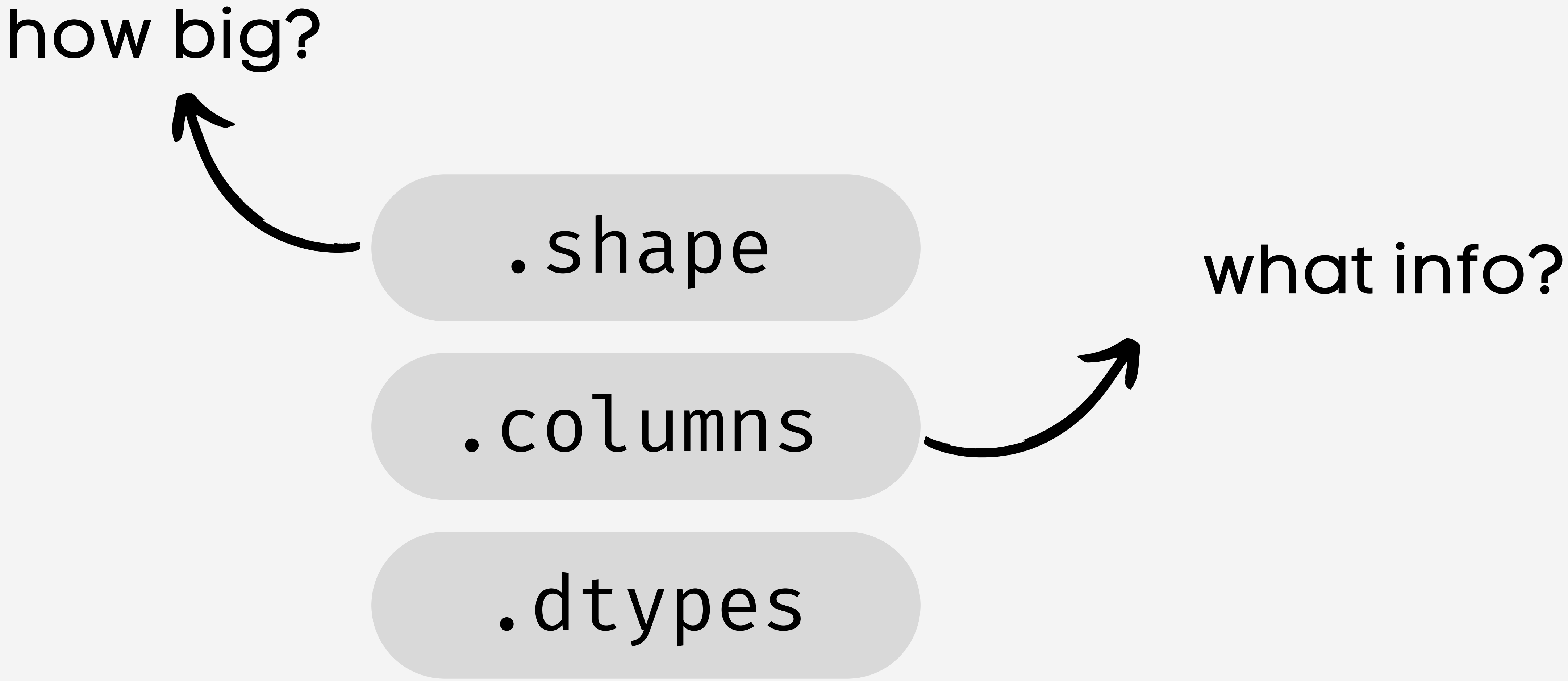
.dtypes

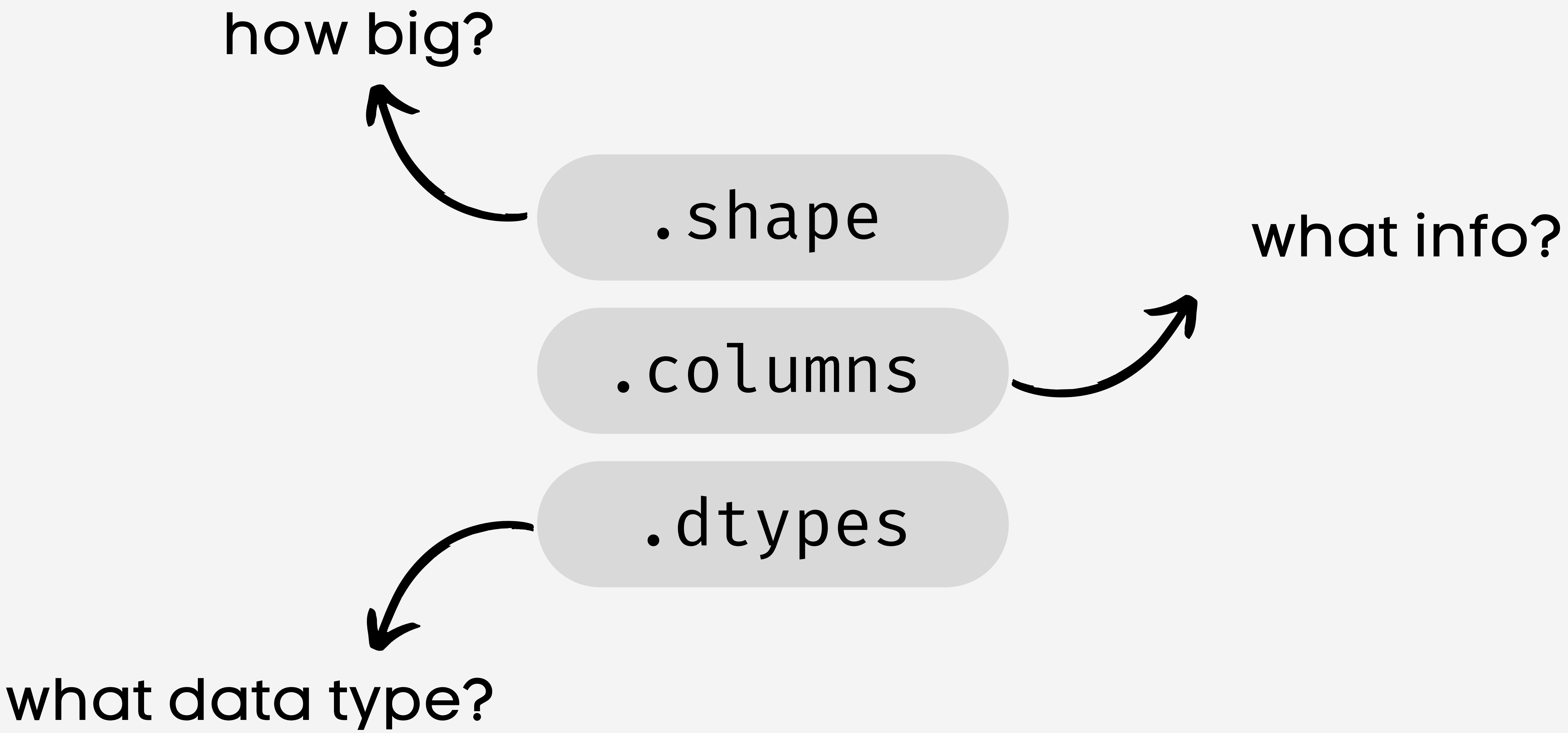
how big?

`.shape`

`.columns`

`.dtypes`


how big?
.shape
.columns
.dtypes
what info?


how big?
.shape
.columns
.dtypes
what info?
what data type?

.unique()

.value_counts()

.groupby()

what values in my column?

```
.unique()
```

```
.value_counts()
```

```
.groupby()
```

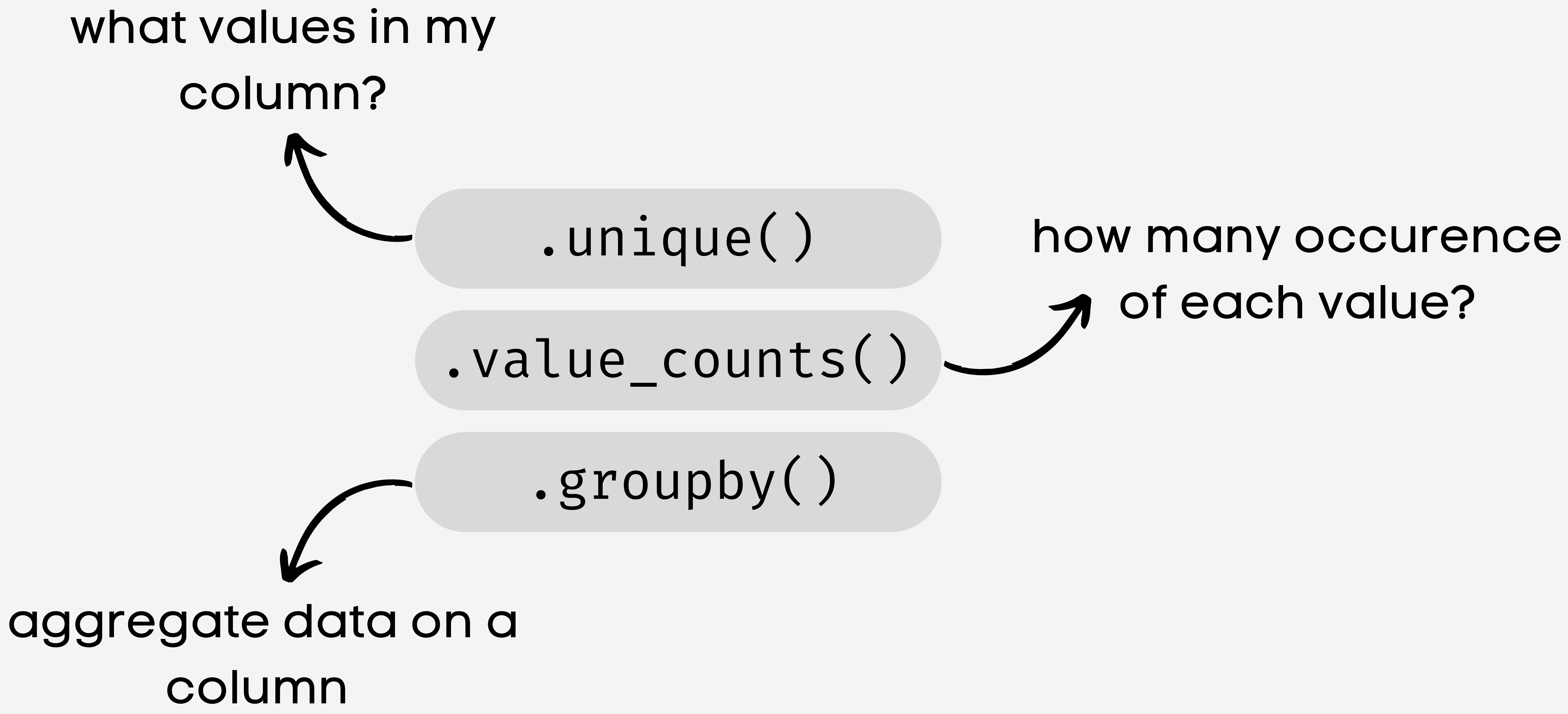
what values in my column?

`.unique()`

`.value_counts()`

`.groupby()`

how many occurence of each value?


what values in my column?
.unique()
.value_counts()
how many occurence of each value?
.groupby()
aggregate data on a column

PREPARE YOUR DATA

is my dataset **clean**? are there any **null values**?

is my dataset **clean**? are there any **null values**?

```
my_df.isna().sum()
```

is my dataset **clean**? are there any **null values**?

```
my_df.isna().sum()
```

tests if each cell is null
(returns **T**rue or **F**alse)

is my dataset **clean**? are there any **null values**?

```
my_df.isna().sum()
```

tests if each cell is null
(returns **T**rue or **F**alse)

sums up all
the **T**rue values

NaN is equivalent to **no data**

what to do with NaNs? **keep** them or **remove** them?

it depends...

sometimes **no data** is data

in our case of exoplanets, **no data** means that the detection method did not pick up any signal in this case, **no data** is a valuable information that you should probably keep do we have all the data we need?

let's create a **new feature** (i.e. column)

I want to know the mass of each exoplanet relative the Earth's

I know that Jupiter is 318 times heavier than Earth

$$M_{Jupiter} = 318 \times M_{Earth}$$

Step 1: create new column where "**Jupiter**" is replaced by **318** and "**Earth**" is replaced by **1**

Step 1: create new column where "**Jupiter**" is replaced by **318** and "**Earth**" is replaced by **1**

```
my_df["conv_factor"] = my_df["mass_wrt"].map({"Jupiter": 318, "Earth": 1})
```

Step 2: compute the exoplanet mass by multiplying **mass_multiplier** to **earth_mass**

$$M_{Exoplanet} = mass\_multiplier \times conv\_factor$$

Step 2: compute the exoplanet mass by multiplying **mass_multiplier** to **earth_mass**

```
my_df["mass_wrt_earth"] = my_df["mass_multiplier"] x my_df["conv_factor"]
```

VISUALIZE YOUR DATA

now that our dataset is ready, let's answer some questions using **data visualizations** to do so, we'll use the Python library **ploty.express**

how many exoplanets have been detected throughout the years?

how many exoplanets have been detected throughout the years?

```
my_df.groupby("discovery_year").count()
```

how many exoplanets have been detected throughout the years?

```
my_df.groupby("discovery_year").count()
```

try it yourself in the playground

how many exoplanets have been detected throughout the years?

```
my_df.groupby("discovery_year", as_index = False).count()
```

store your aggregated dataframe into a new variable

```
yearly_discoveries = my_df.groupby("discovery_year", as_index = False).count()
```

store your aggregated dataframe into a new variable

```
yearly_discoveries = my_df.groupby("discovery_year", as_index = False).count()
```

try it yourself in
the playground

create your line chart

```
px.line(x = yearly_discoveries["discovery_year"], y = yearly_discoveries["name"])
```

create your line chart

```
px.line(x = yearly_discoveries["discovery_year"], y = yearly_discoveries["name"])
```

try it yourself in
the playground

how many exoplanets have been detected using each detection method?

how many exoplanets have been detected using each detection method?

```
my_df["detection_method"].value_counts()
```

store your aggregated dataframe into a new variable

```
method_discoveries = my_df["detection_method"].value_counts()
```

better to go for a bar chart!

```
px.bar(x = method_discoveries["detection_method"], y = method_discoveries["name"])
```

better to go for a bar chart!

```
px.bar(x = method_discoveries["detection_method"], y = method_discoveries["name"])
```

try it yourself in
the playground

what is the proportion of each exoplanet type?

what is the proportion of each exoplanet type?

```
px.pie(my_df["planet_type"])
```

how are stellar magnitudes distributed?

how are stellar magnitudes distributed?

```
px.histogram(my_df["stellar_magnitude"])
```

for each planet type, what is the distribution of their mass?

for each planet type, what is the distribution of their mass?

```
px.box(x = my_df["planet_type"], y = my_df["exoplanet_mass"])
```

are there any correlations between features (i.e. columns)?

are there any correlations between features (i.e. columns)?

```
my_df.corr()
```

are there any correlations between features (i.e. columns)?

```
my_df.corr()
```

```
corr_matrix = my_df.corr()
```

are there any correlations between features (i.e. columns)?

```
my_df.corr()
```

```
corr_matrix = my_df.corr()
```

```
px.imshow(corr_matrix)
```

```
.scatter()
.bar()
.pie()
.box()
.imshow()
```

y vs x plots of
continuous variables

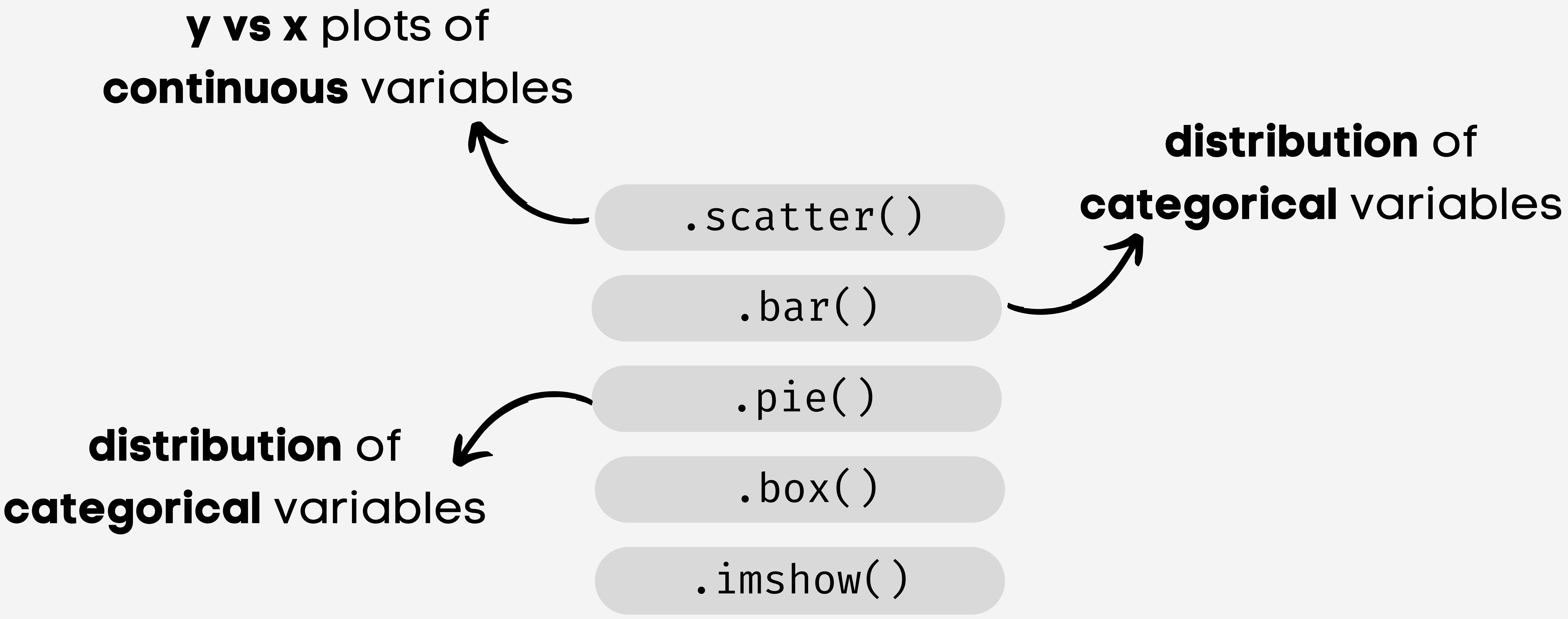
`.scatter()`

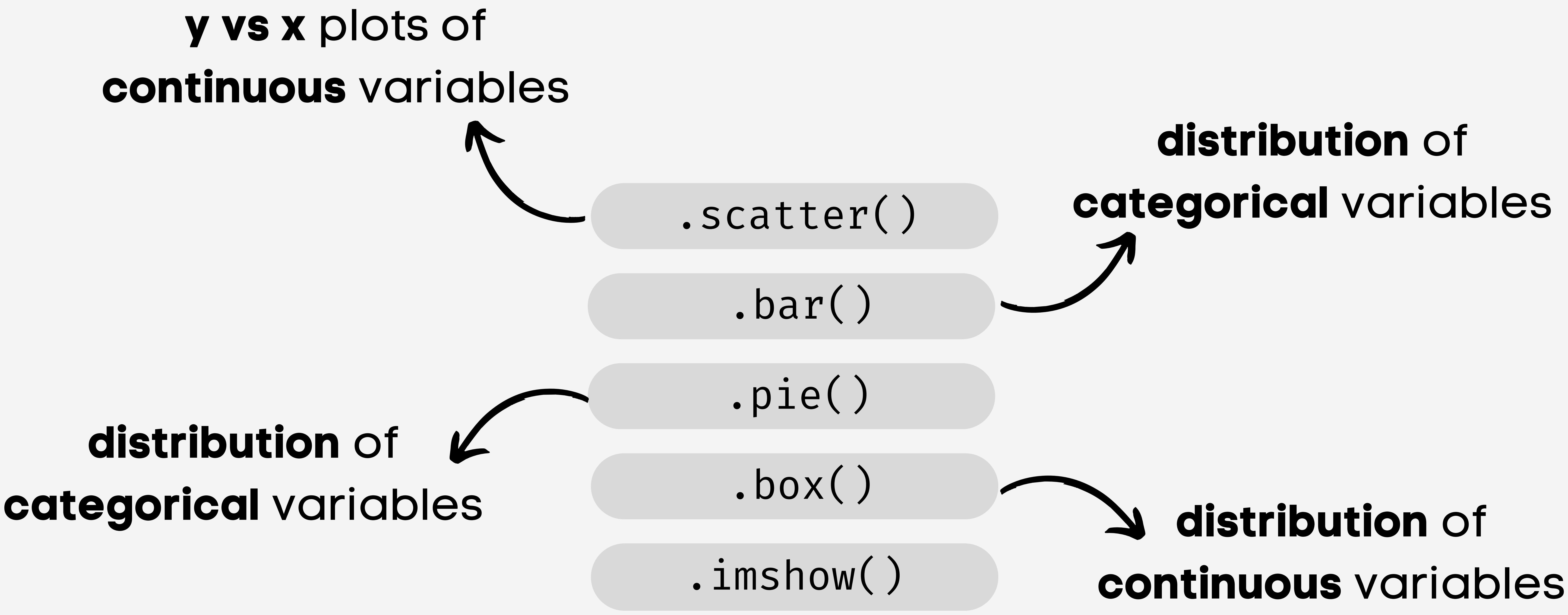
`.bar()`

`.pie()`

`.box()`

`.imshow()`


y vs x plots of
continuous variables
.scatter()
.bar()
.pie()
.box()
.imshow()
distribution of
categorical variables
distribution of
categorical variables


y vs x plots of **continuous** variables
.scatter()
.bar()
.pie()
.box()
.imshow()
distribution of **categorical** variables
distribution of **categorical** variables
distribution of **continuous** variables

y vs x plots of **continuous** variables

`.scatter()`

`.bar()`

distribution of **categorical** variables

`.pie()`

distribution of **categorical** variables

`.box()`

distribution of **continuous** variables

`.imshow()`

display an image