

PROGRAMMING WITH R  
ANALYTICAL QUESTIONS  
EXPLORATORY DATA ANALYSIS  
DATA REPRESENTATION  
VECTORS  
DATA FRAMES  
LOAD DATA  
TIDY DATA  
STRINGS  
TRANSFORM DATA  
UNSTRUCTURED DATA  
MACHINE LEARNING  
VISUALIZE DATA  
COMMUNICATE DATA

# PROGRAMMING WITH R

variables

# control structures

loops

functions

libraries

# ANALYTICAL QUESTIONS



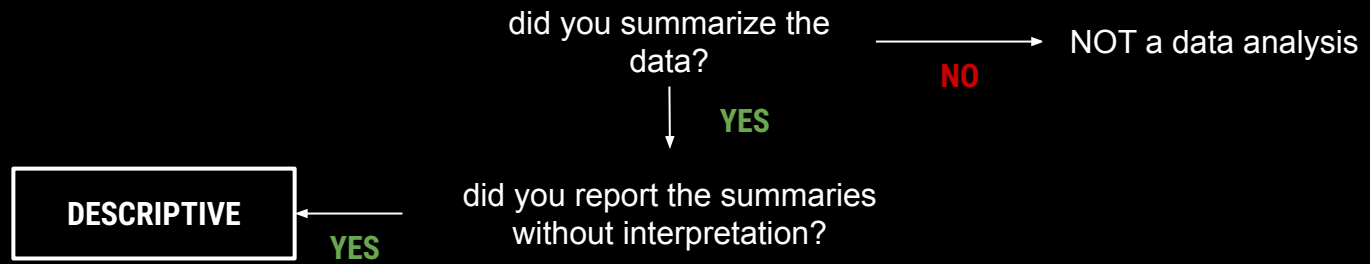
did you summarize the  
data?

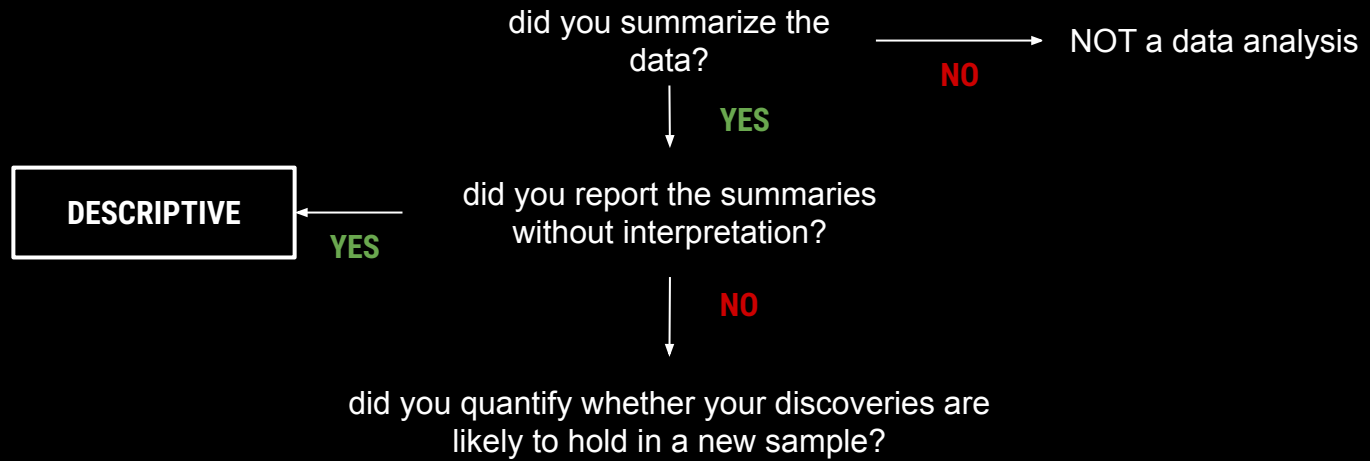
did you summarize the  
data?

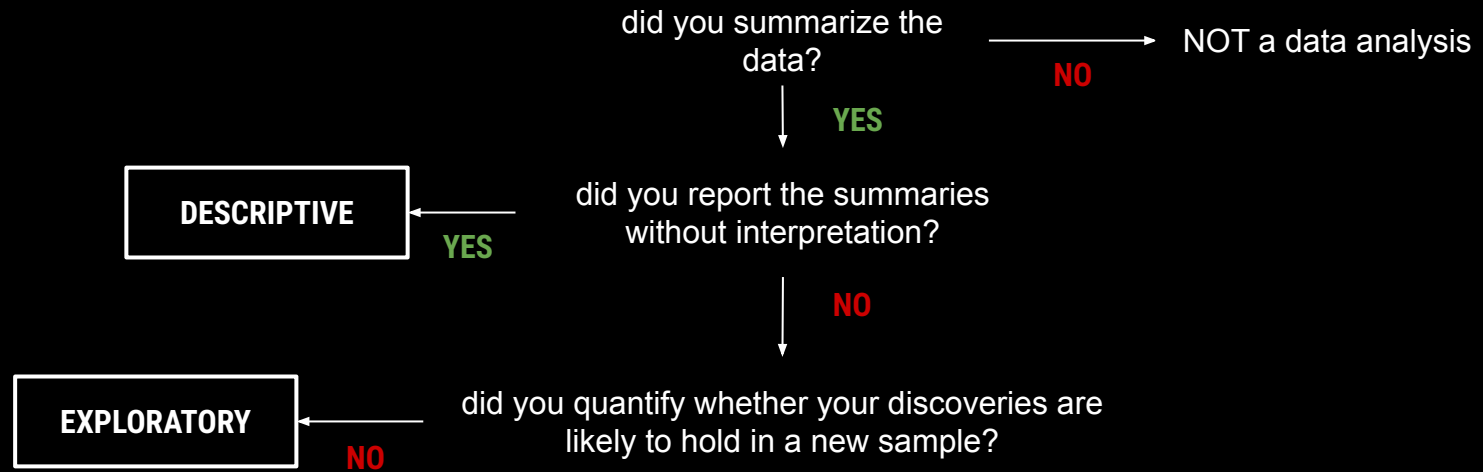
NO

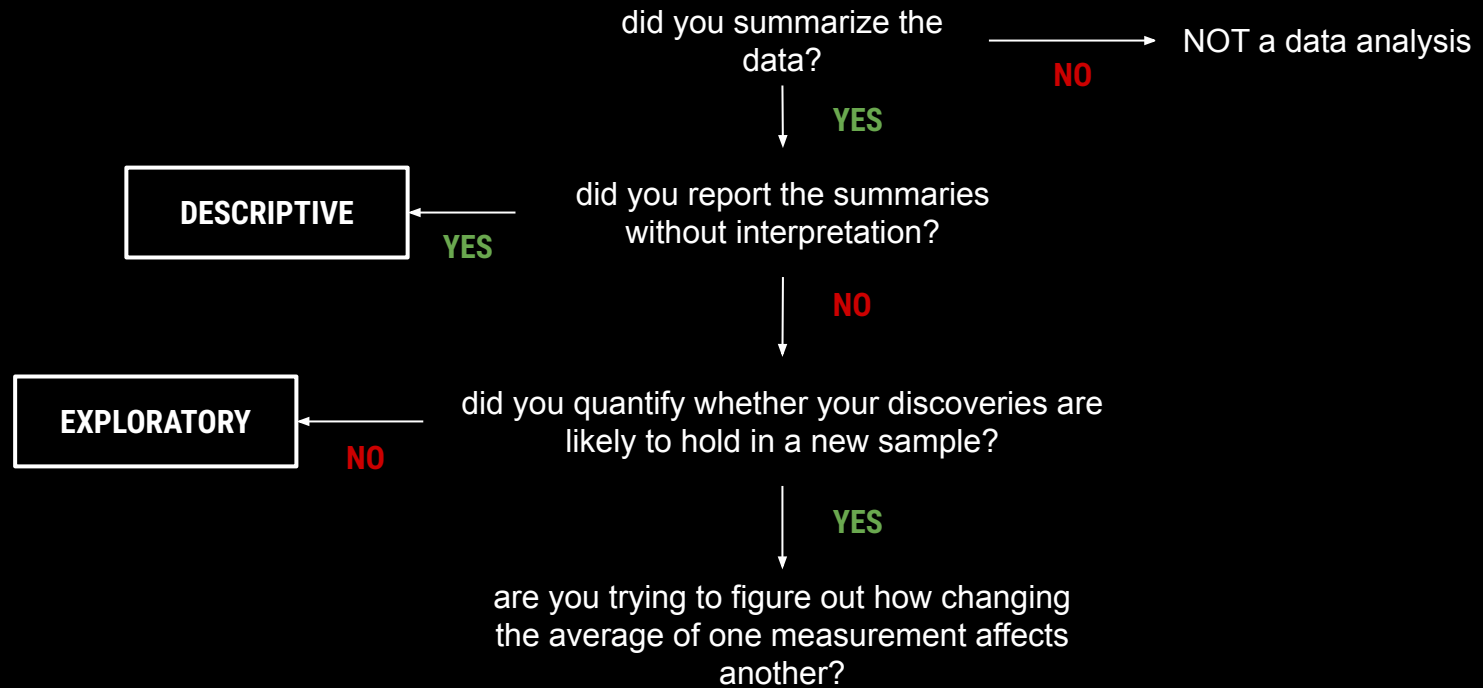
NOT a data analysis

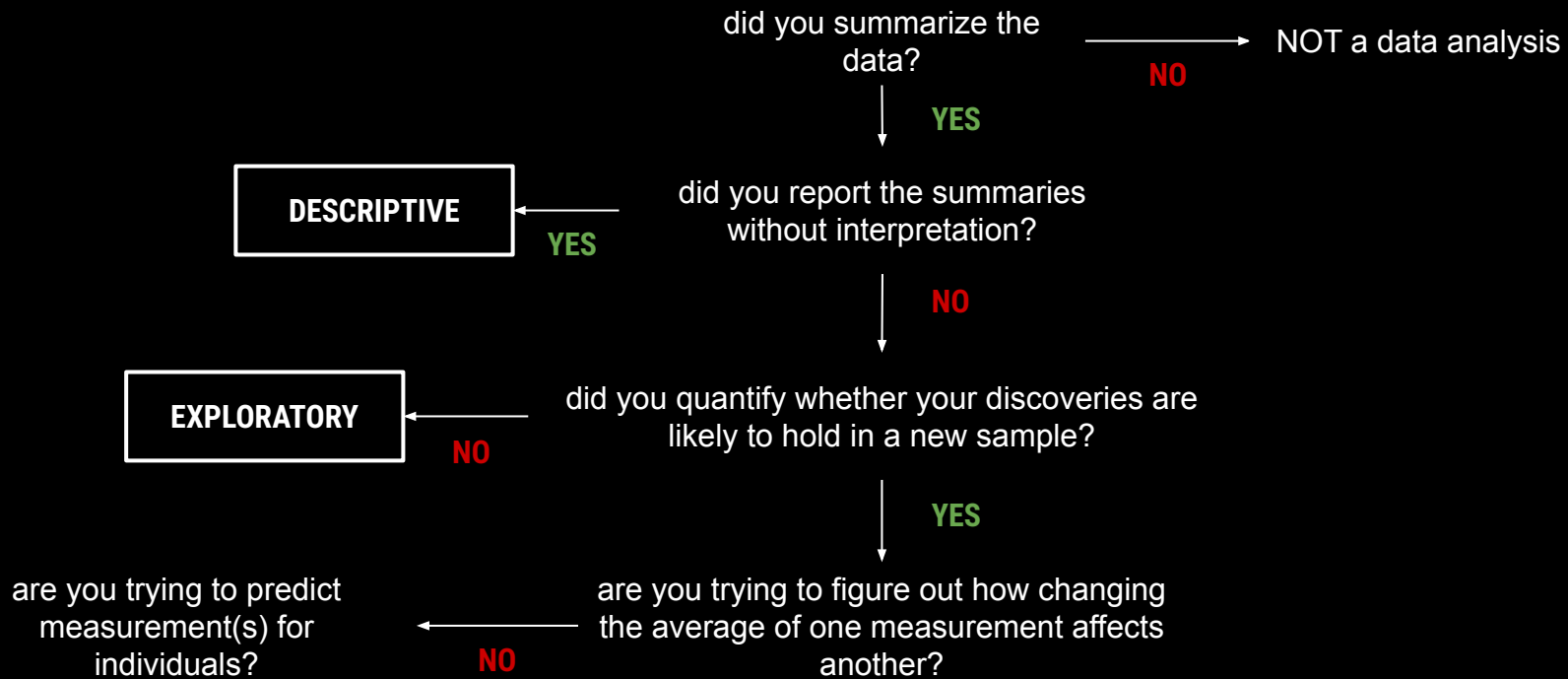




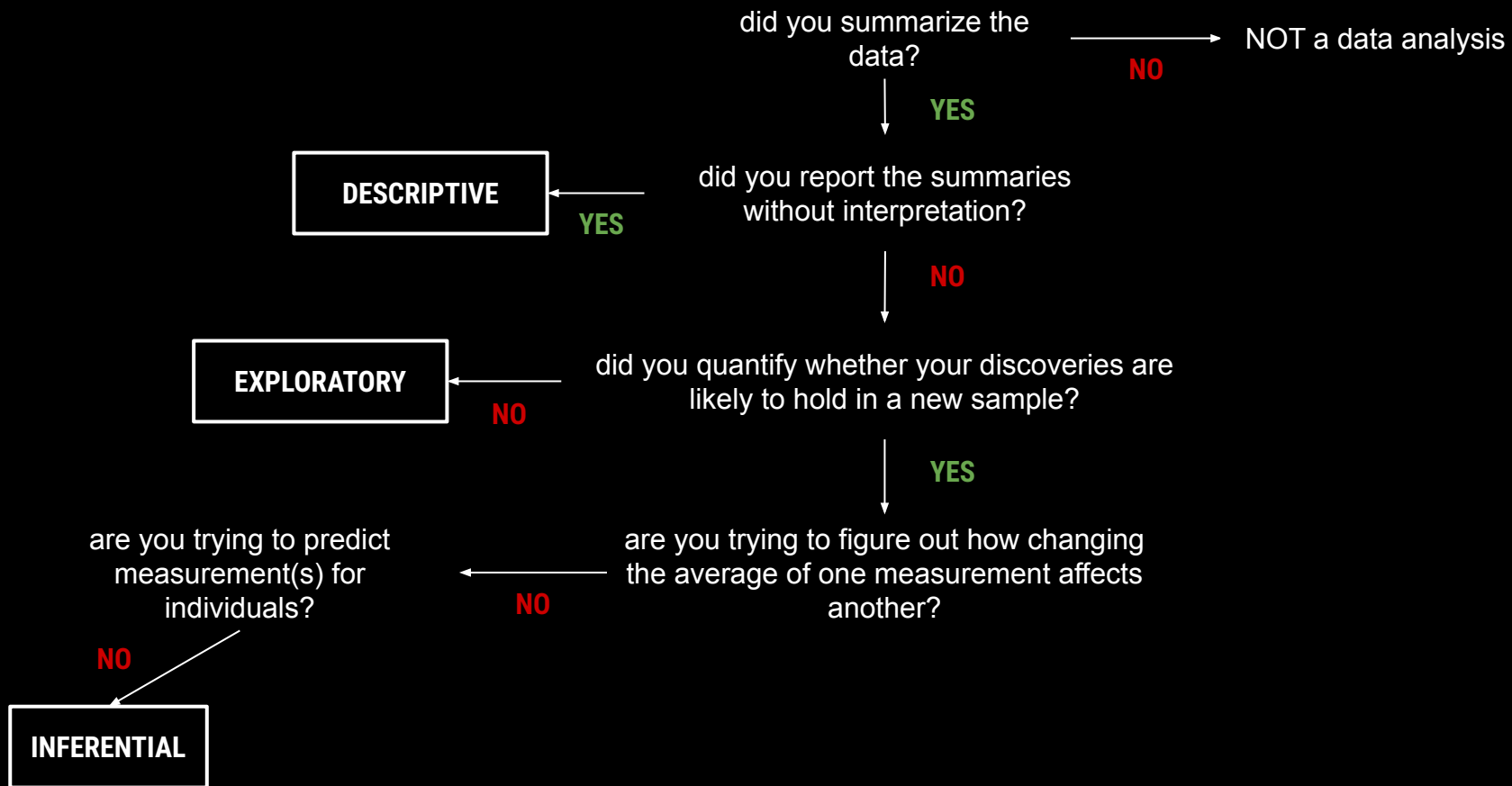


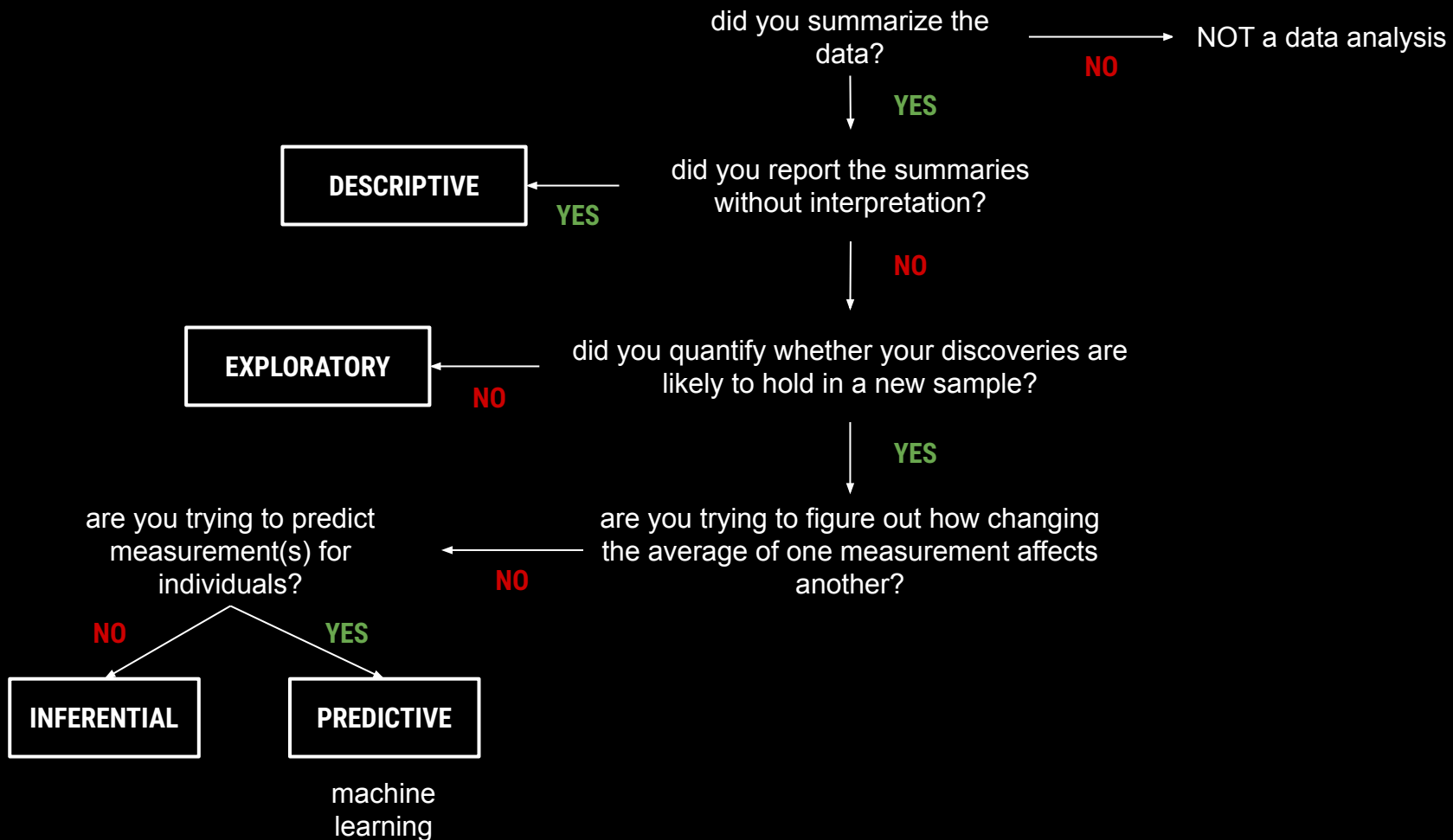


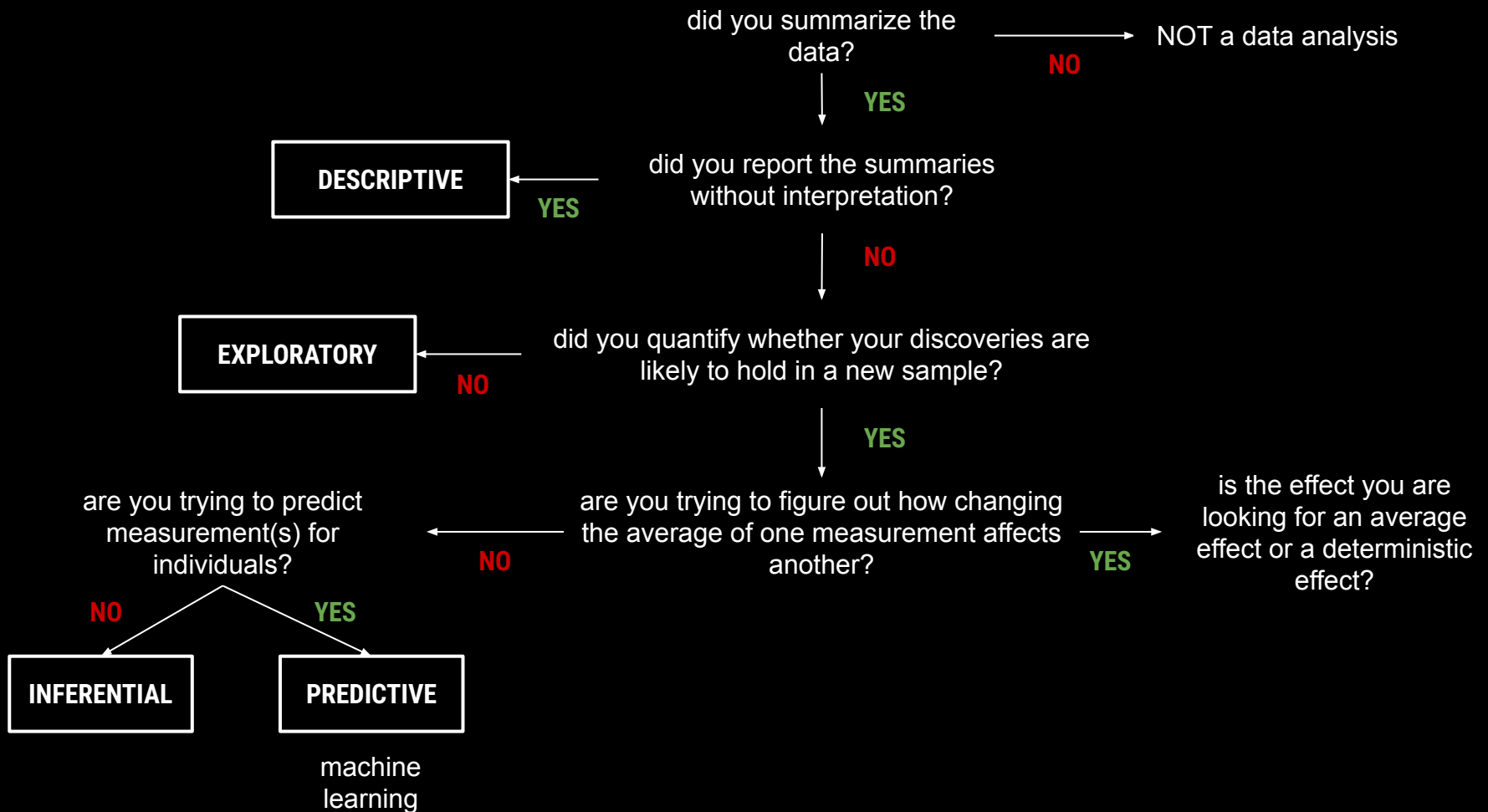


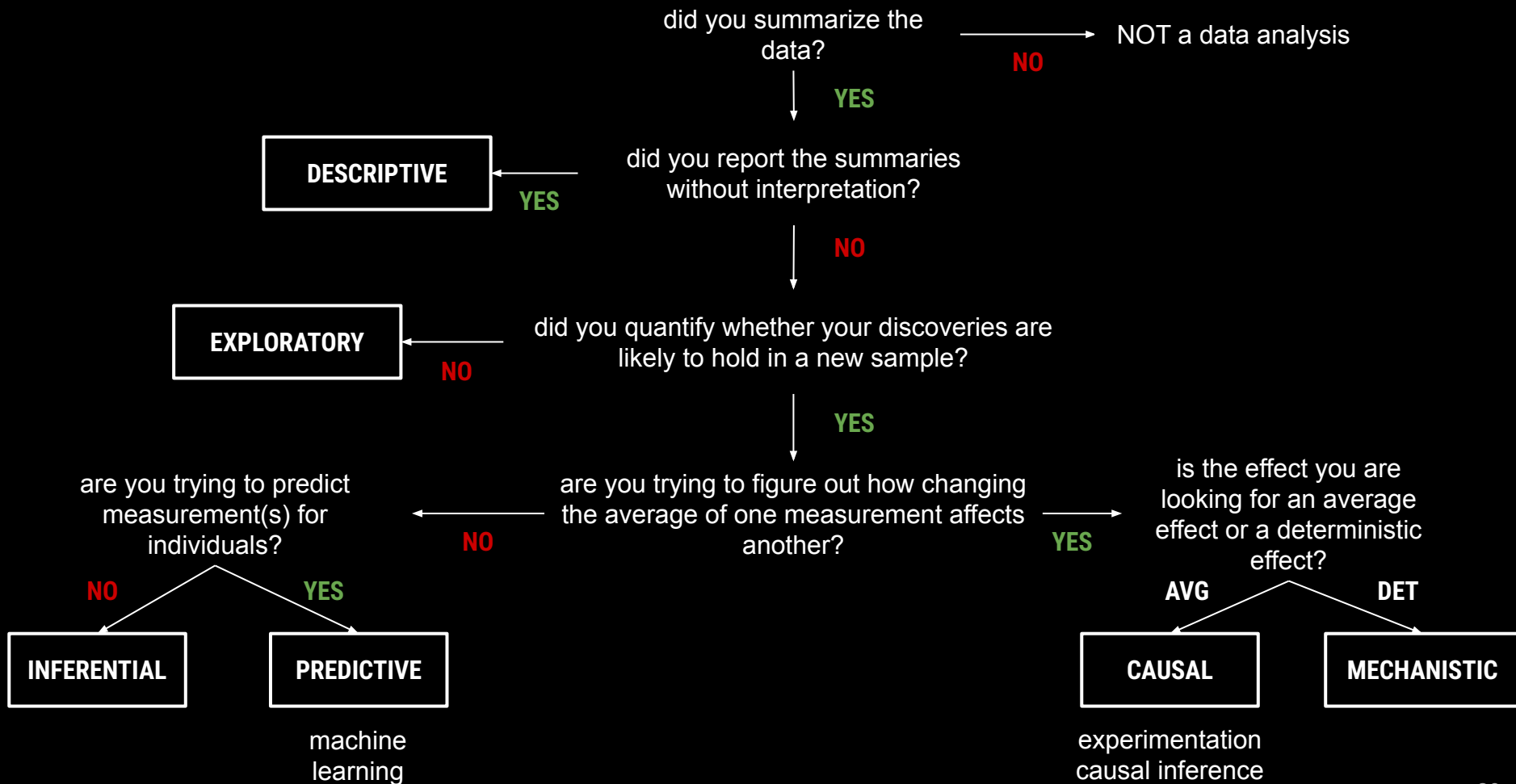


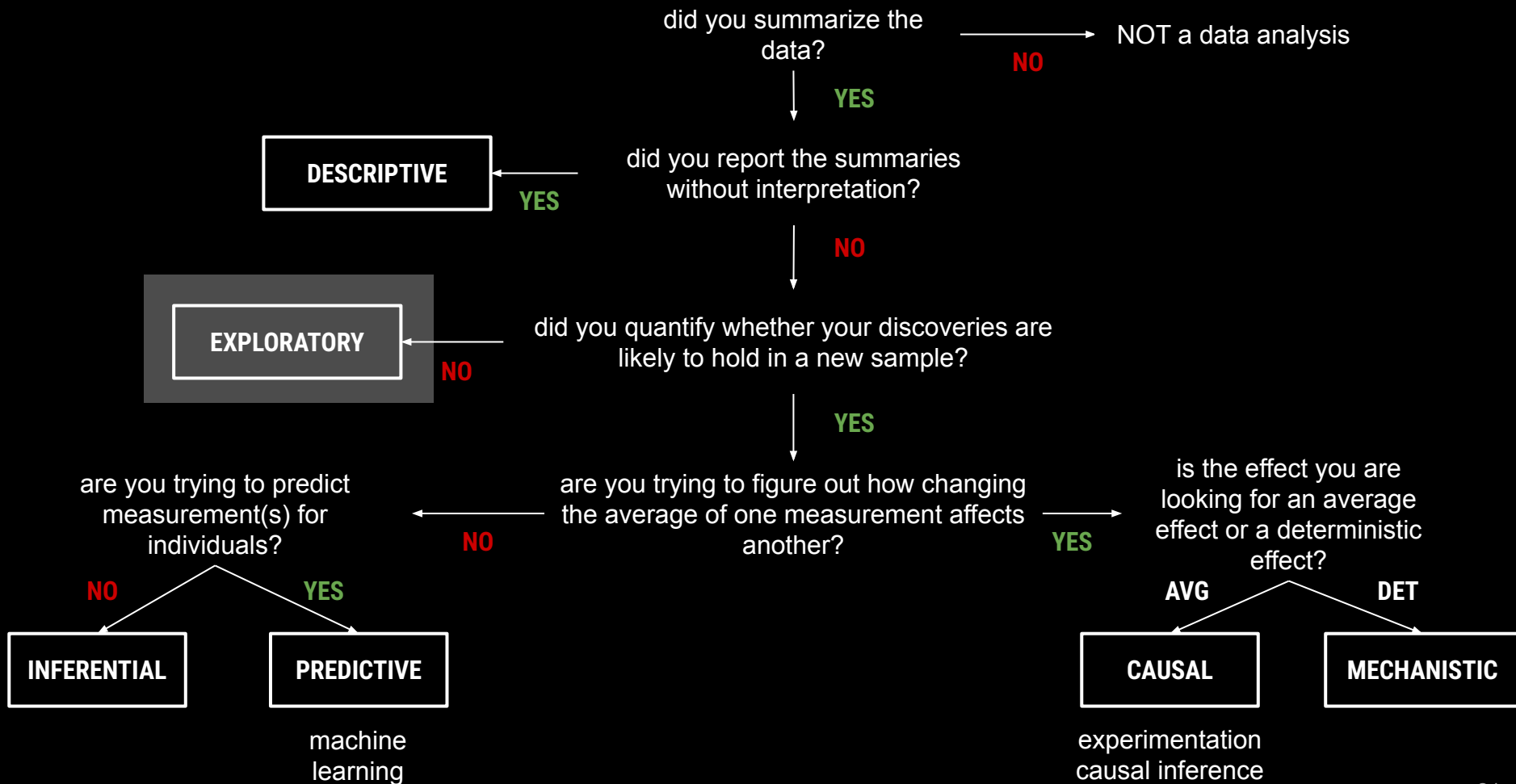






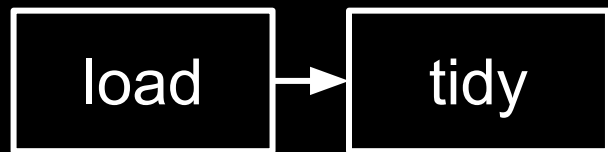




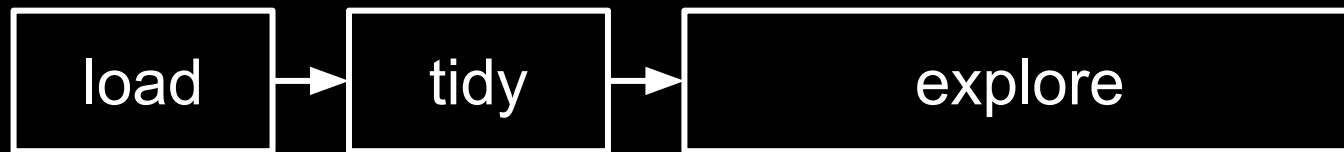


# EXPLORATORY DATA ANALYSIS

load

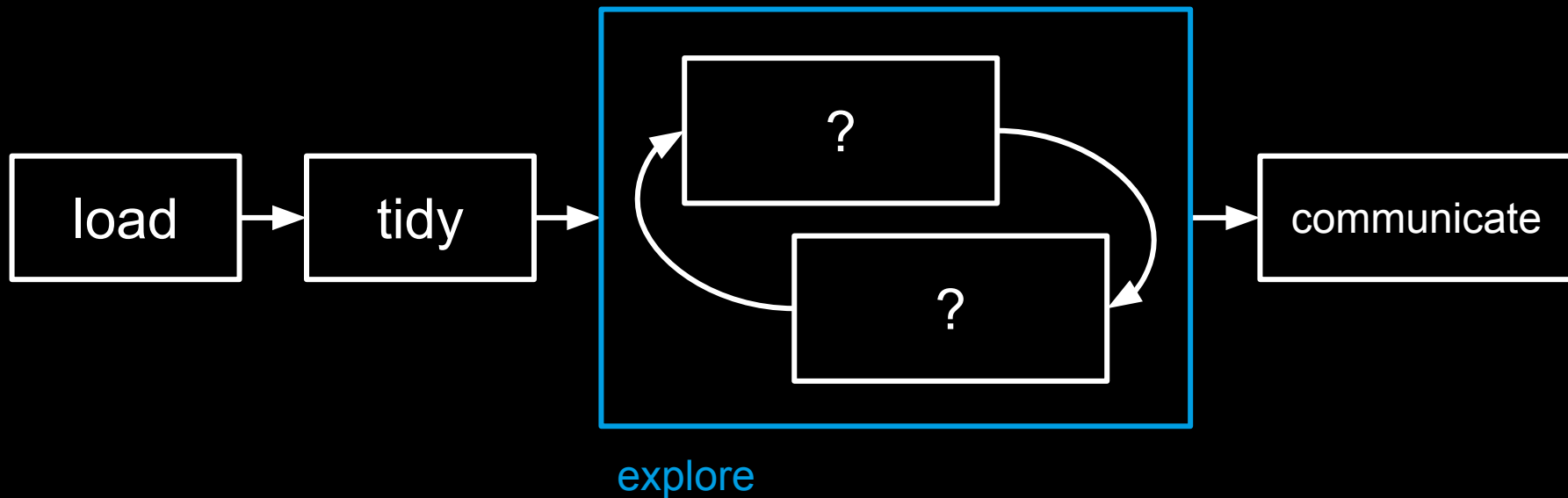


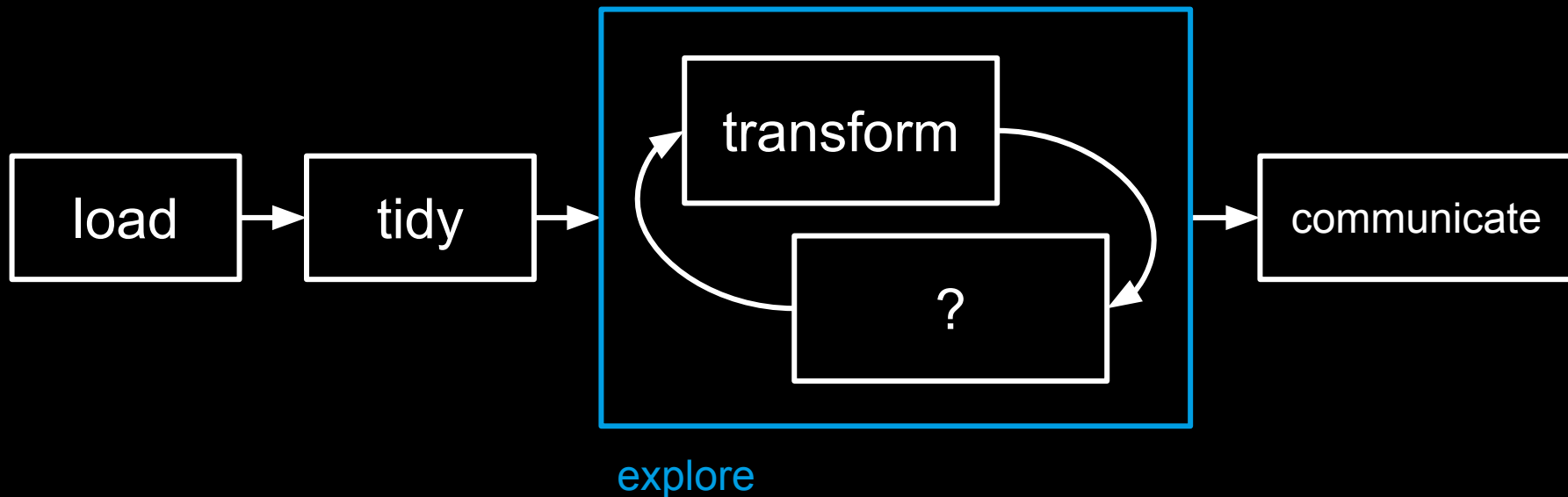


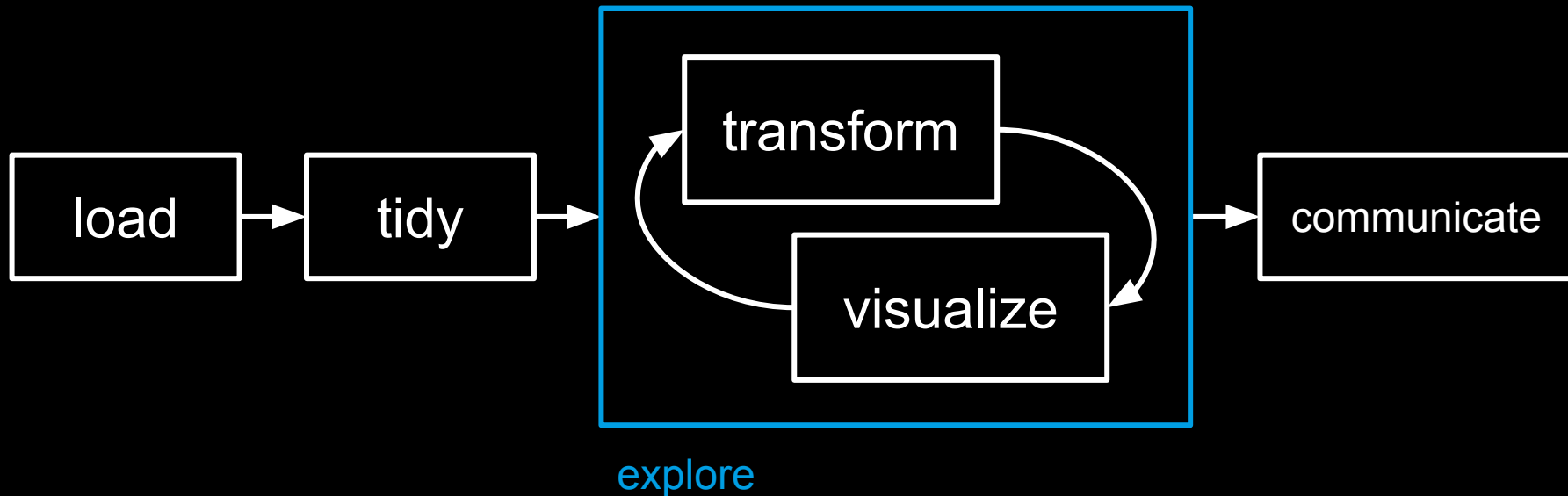


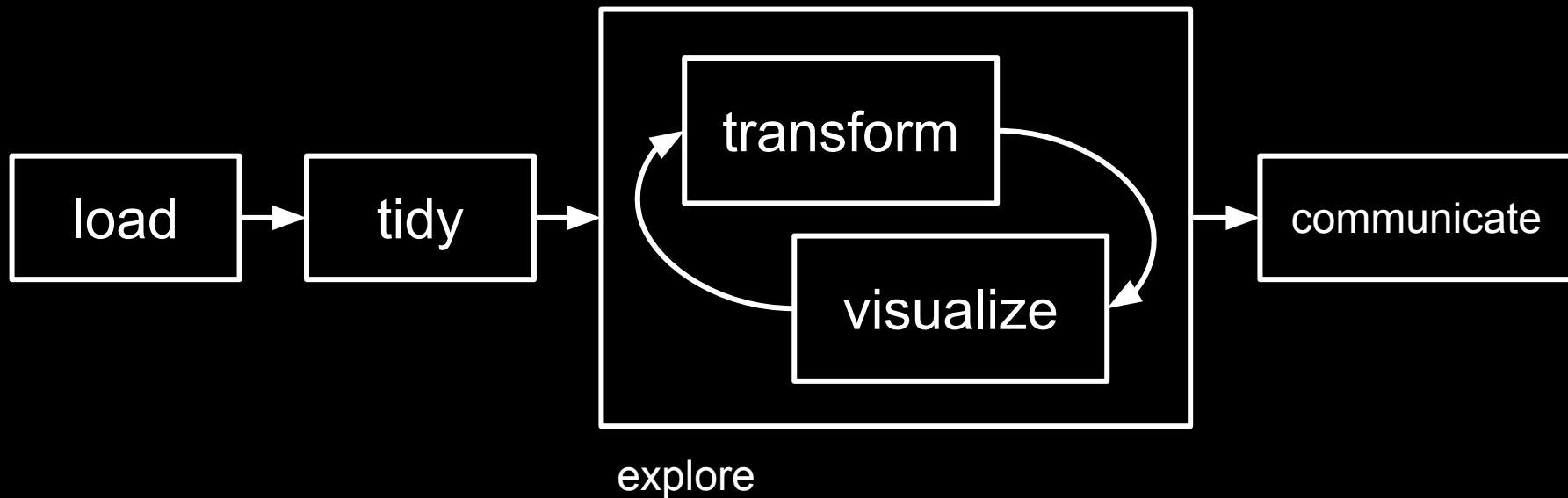


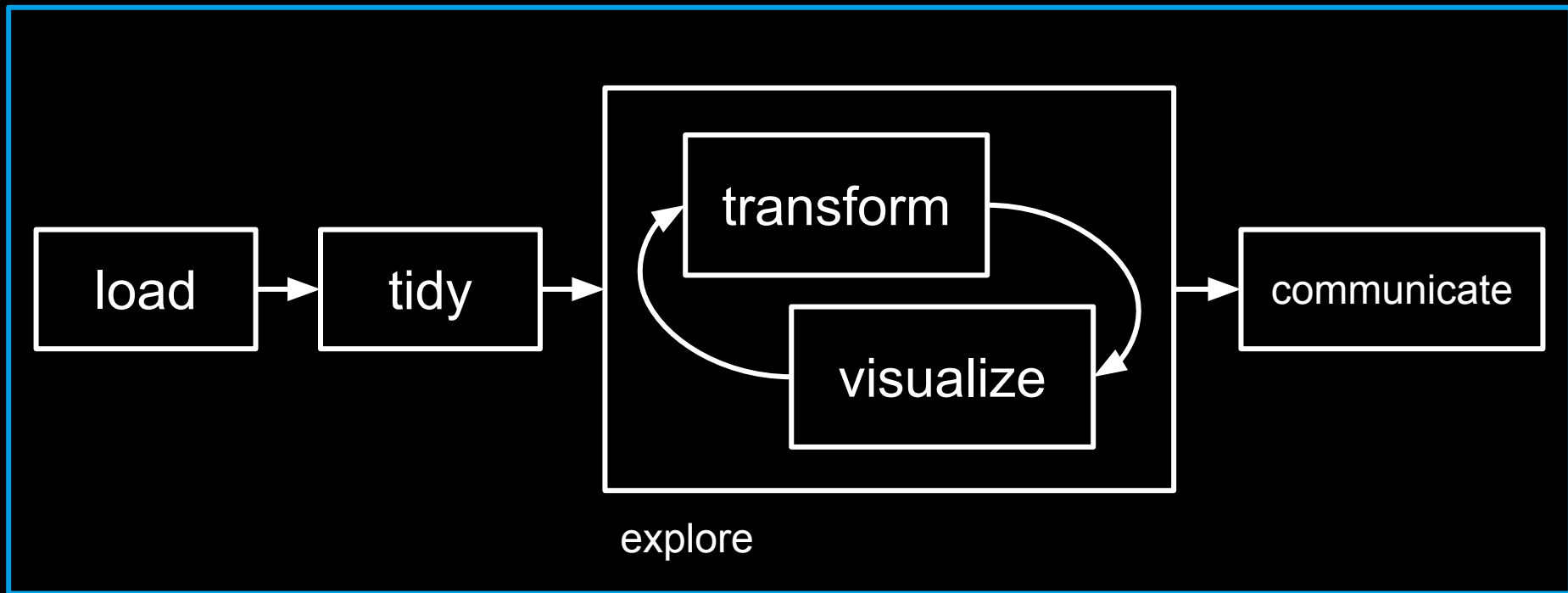






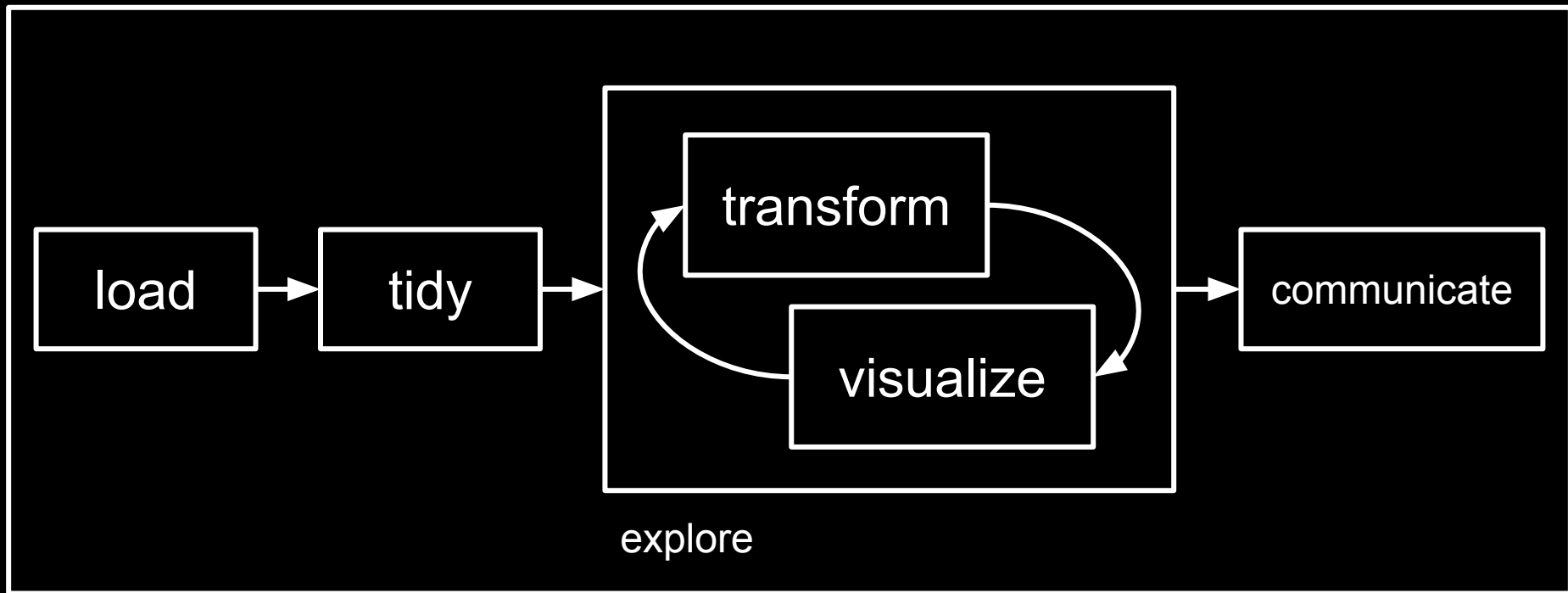






program

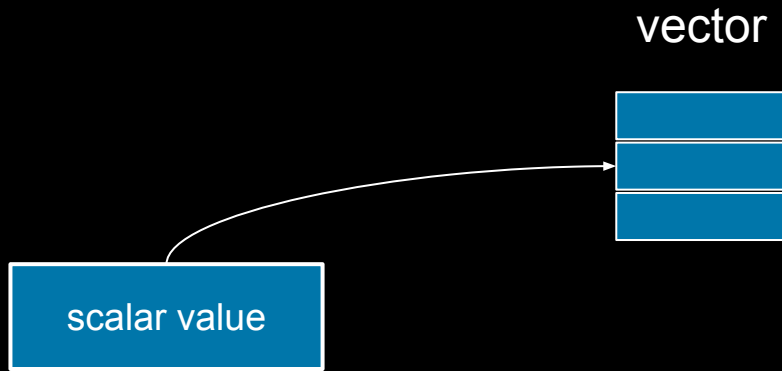


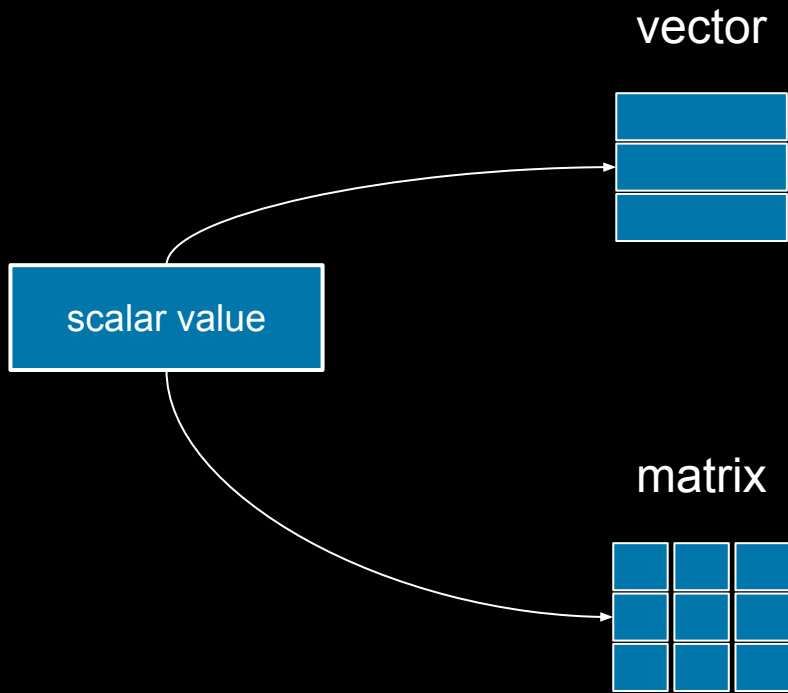


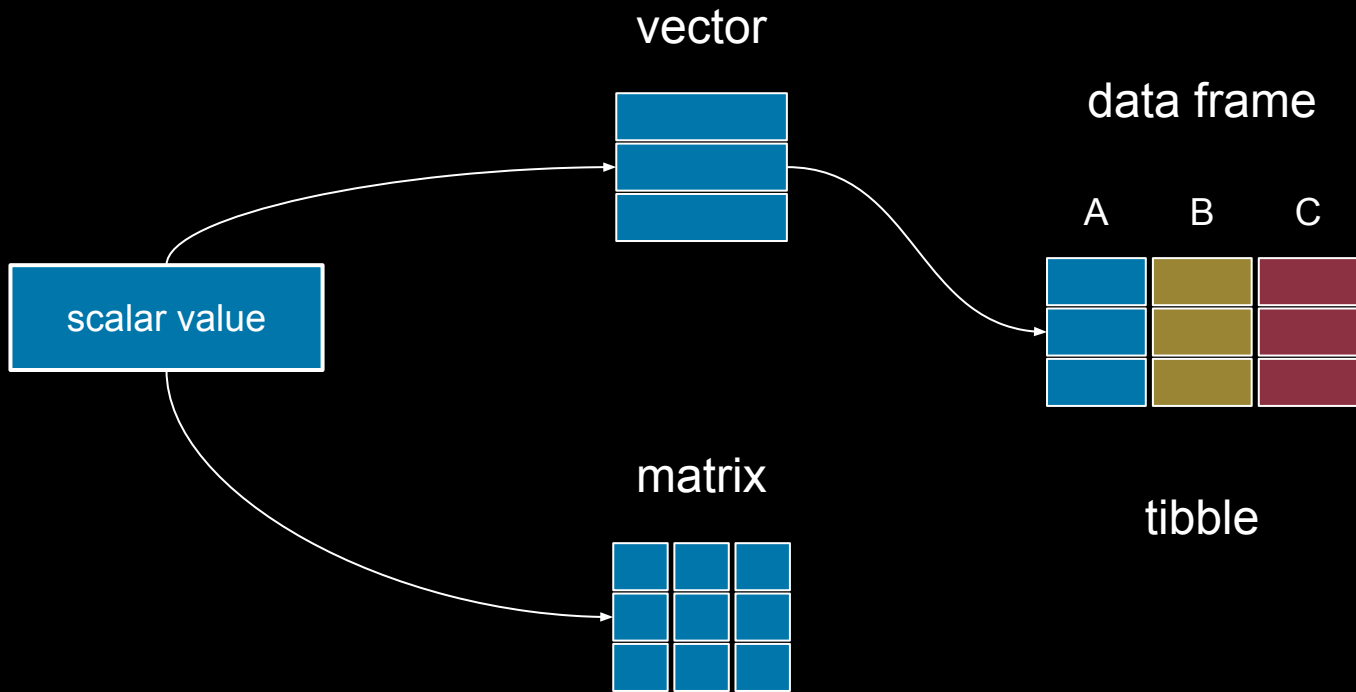
program

# DATA REPRESENTATION

scalar value







# VECTORS

apple
pear
orange



list of values with  
the same storage mode

list of values with  
the same storage mode

character  
double  
integer  
logical

```
v <- c("apple", "pear", "orange")
```

v[1]

apple
pear
orange

v[2]

apple
pear
orange

v[3]

apple
pear
orange

```
weight <- c(91, 75.5, 61, 88.5, 120)
```



```
weight <- c(91, 75.5, 61, 88.5, 120)  
mean(weight)
```

sum	length
mean	sort
median	cumsum
sd	prod
var	quantile
min	abs
max	range

91
75.5
61
88.5
120

```
weight_after_diet <-  
  c(89.5, 75, 56, 96.5, 115)
```

weight

weight\_after\_diet

91
75.5
61
88.5
120

—

89.5
75
56
96.5
115

weight

91
75.5
61
88.5
120

weight\_after\_diet

89.5
75
56
96.5
115

weight\_loss

1.5
0.5
5
-8
5

—

=

```
weight_loss <-  
  weight - weight_after_diet
```

subsetting vectors

weight[1]



weight[1]

weight[-1]

```
weight[1]
```

```
weight[-1]
```

```
weight[2:5]
```

```
weight[1]
```

```
weight[-1]
```

```
weight[2:5]
```

```
weight[1:length(weight)-1]
```

```
weight[1]
```

```
weight[-1]
```

```
weight[2:5]
```

```
weight[1:length(weight)-1]
```

```
weight[c(TRUE, FALSE, TRUE, TRUE, FALSE)]
```

```
weight[1]
```

```
weight[-1]
```

```
weight[2:5]
```

```
weight[1:length(weight)-1]
```

```
weight[c(TRUE, FALSE, TRUE, TRUE, FALSE)]
```

```
weight[weight > 80]
```

```
weight[1]
```

```
weight[-1]
```

```
weight[2:5]
```

```
weight[1:length(weight)-1]
```

```
weight[c(TRUE, FALSE, TRUE, TRUE, FALSE)]
```

```
weight[weight > 80]
```

```
weight[weight > 80 & weight < 100]
```

special values

NA

NULL

NaN

Inf

-Inf



factors

```
category <- factor(c("heavy", "medium", "light", "medium", "heavy"))
```

```
category <- factor(c("heavy", "medium", "light", "medium", "heavy"))
```

```
levels(weight_category)
```

```
category <- factor(c("heavy", "medium", "light", "medium", "heavy"))
```

```
levels(weight_category)
```

```
category_reordered <- factor(category,  
                               levels = c("light", "medium", "heavy"))
```

```
category <- factor(c("heavy", "medium", "light", "medium", "heavy"))
```

```
levels(weight_category)
```

```
category_reordered <- factor(category,  
                              levels = c("light", "medium", "heavy"))
```

```
category_ordered <- factor(category,  
                             levels = c("light", "medium", "heavy"),  
                             ordered = TRUE)
```

{{ forcats }}

`as_factor()`

fct\_reorder  
fct\_relevel  
fct\_infreq  
fct\_rev  
fct\_lump



# DATA FRAMES

"apple"

"pear"

"orange"

"apple"	TRUE
"pear"	TRUE
"orange"	FALSE

"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

fruit

"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

fruit	domestic	
"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

fruit	domestic	sugar
"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

fruit	domestic	sugar
"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0



data frame "fruits"

fruit	domestic	sugar
"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

creating data frames

```
data.frame()  
read.csv()
```

comma separated values (CSV)

data frame meta data

`ncol()`

`nrow()`

`dim()`

`colnames()`

accessing data frames

accessing data frames  
accessing columns



```
monty$prize_door  
monty$contestant_choice  
monty$decision
```

```
monty$prize_door  
monty$contestant_choice  
monty$decision
```



return a vector

```
monty["prize_door"]  
monty["contestant_choice"]  
monty["decision"]
```

```
monty["prize_door"]  
monty["contestant_choice"]  
monty["decision"]
```



return a data  
frame

```
# multiple columns by name  
monty(c("prize_door", "contestant_choice"))
```

```
monty[, 1]           # first column  
monty[, 1:2]         # first two columns  
monty[, ncol(monty)] # last column
```

accessing data frames  
accessing rows

```
monty[1,]           # first row
monty[1:10,]        # first 10 rows
monty[nrow(monty),] # last row
```



changing columns

```
monty$decision <- as.factor(monty$decision)
```

adding columns

```
monty$correct_guess <-  
  monty$contestant_choice == monty$prize_door
```

rename columns

```
colnames(monty)[2] <- "choice"
```

subsetting data frames

```
switched <-  
  monty[monty_hall$decision == "switch, ]
```



```
switched <-  
  monty[  
    monty_hall$decision == "switch &  
    monty$won == TRUE, ]
```

`subset()`

```
subset(monty, decision == "switch")
```

```
subset(  
    monty,  
    decision == "switch" & won == TRUE  
)
```

sorting rows

```
monty[order(monty$prize_door),]
```

```
monty[order(monty$prize_door),]
```

```
monty[order(  
    monty$prize_door,  
    decreasing = TRUE  
),]
```

saving data frames



```
write.csv()
```

tibbles

{{ tibble }}

data frame "fruits"

fruit	domestic	sugar
"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

*some sugar*

data frame "fruits"

fruit	domestic	sugar
"apple"	TRUE	10.0
"pear"	TRUE	16.5
"orange"	FALSE	14.0

tibble "tbl\_fruits"

```
as_tibble()
```

*some sugar*

better printing

subsets stay tibbles

better data type guessing

support for extended data types

...

# LOAD DATA



{{ readr }}

```
read_csv()  
read_delim()
```

{{ readxl }}

`read_excel()`

# TIDY DATA

# tidy data

each variable is a column;  
each column is a variable.

each observation is a row;  
each row is an observation.

each value is a cell;  
each cell is a single value.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898

variables



country	year	cases	population
Afghanistan	1999	745	19937071
Afghanistan	2000	2666	20095360
Brazil	1999	37737	17296362
Brazil	2000	60488	17494898

observations

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898

values

country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898

longer



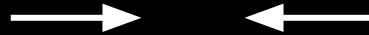
country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898

wider



country	cases_1999	cases_2000	pop_1999	pop_2000
Afghanistan	745	2666	19987071	20595360
Brazil	37737	172006362	80488	174504898

compressed



country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898

# tidy

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898

tidy

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898

vector



{{ tidyrr }}

```
pivot_wider()
```

`pivot_longer()`

# STRINGS

{{ stringr }}

```
str_trim()  
str_squish()
```

str\_starts()  
str\_ends()  
str\_detect()

“Annabel Miller”



“Annabel Miller”

```
str_starts(txt, "Anna")
```

“Annabel Miller”

```
str_ends(txt, "Miller")
```

“Annabel Miller”

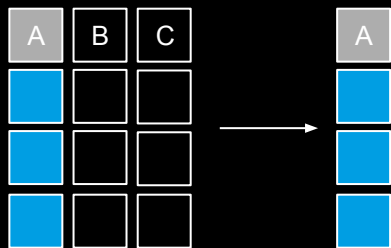
```
str_detect(txt, "Miller")
```

# TRANSFORM DATA

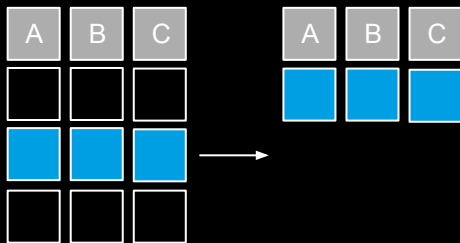
{{ dplyr }}

types of transformations

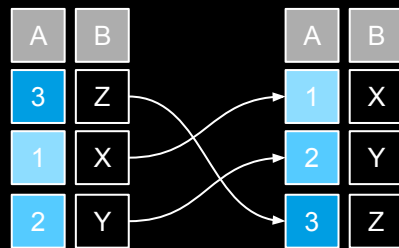
`select()`



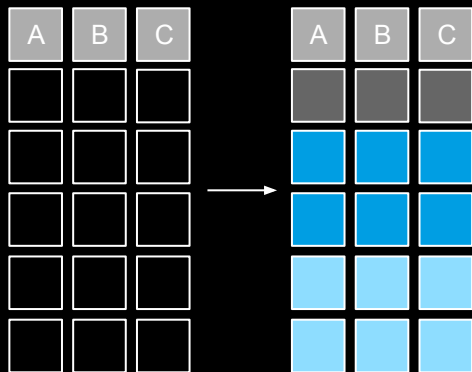
`filter()`



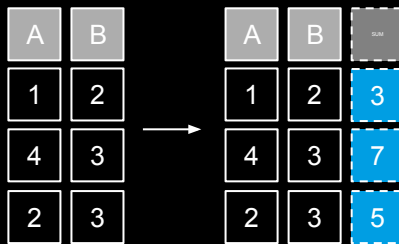
`arrange()`



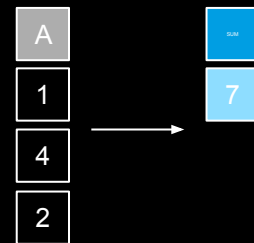
`group_by()`



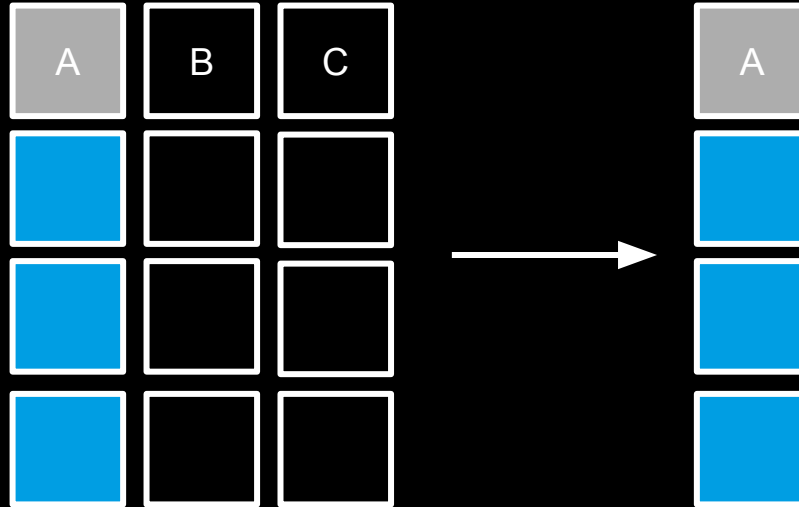
`mutate()`



`summarize()`

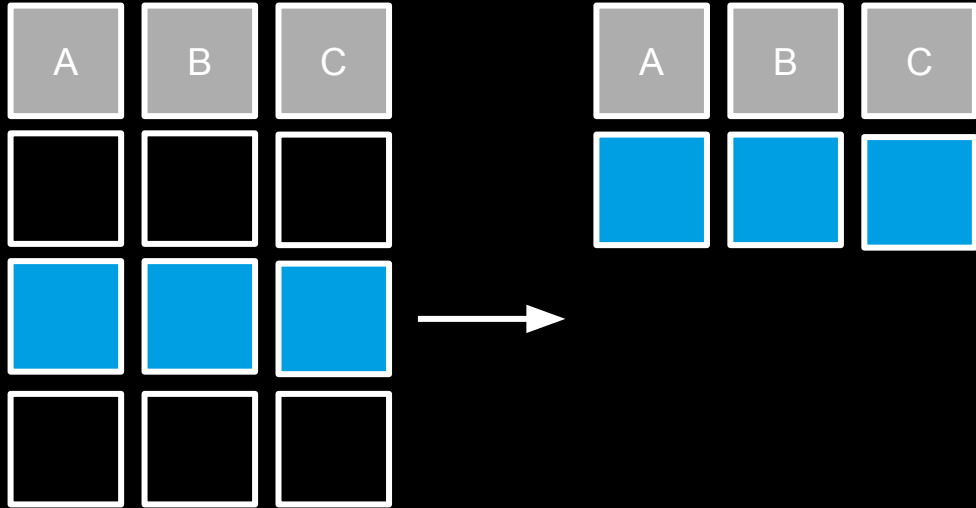


`select()`

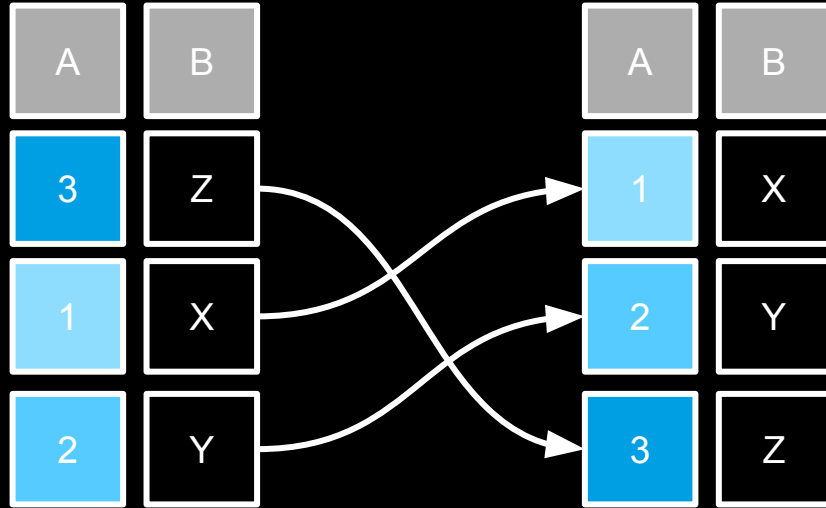




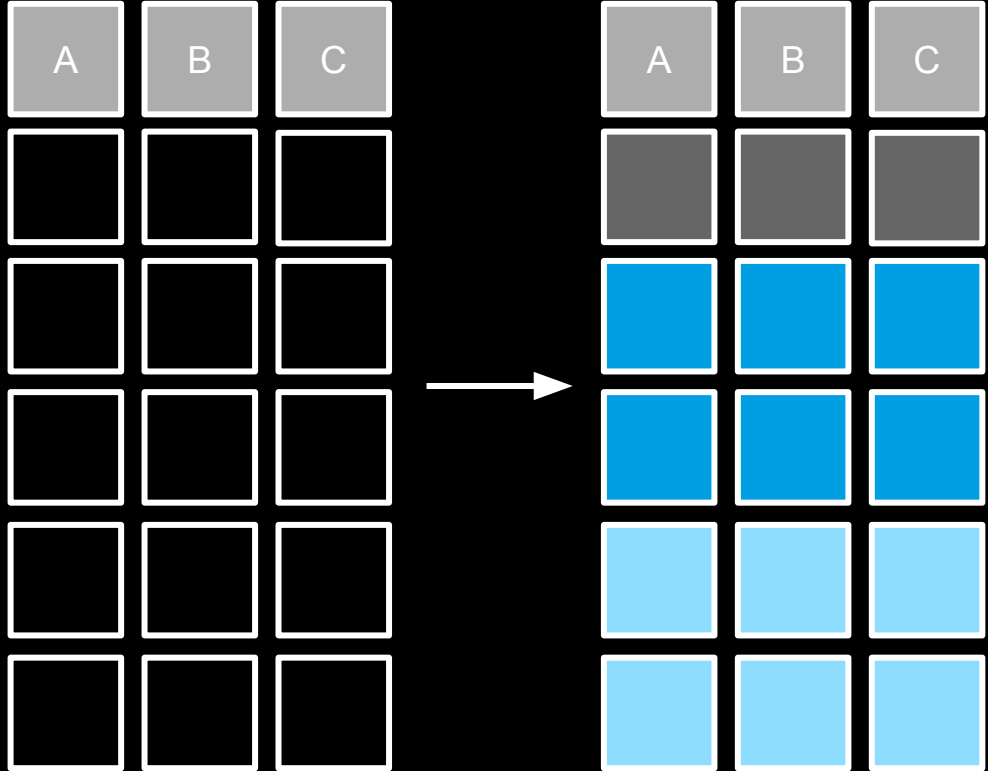
`filter()`



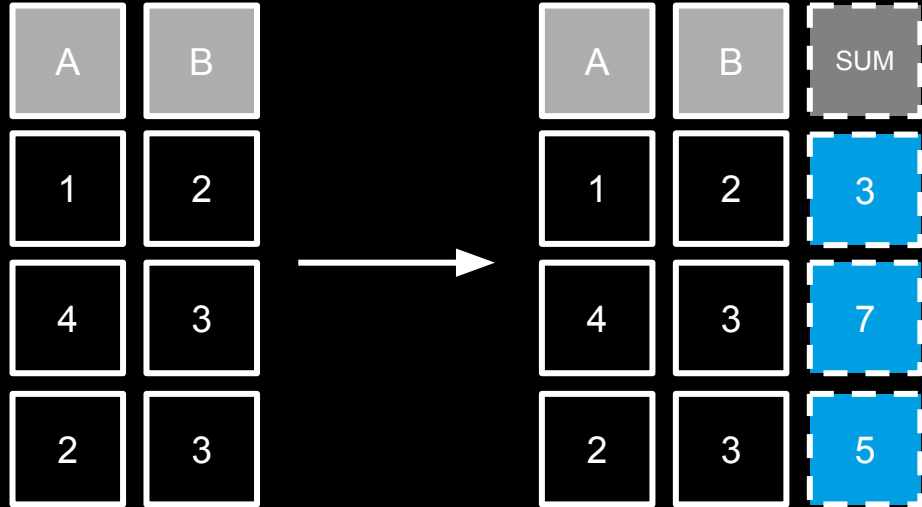
arrange()



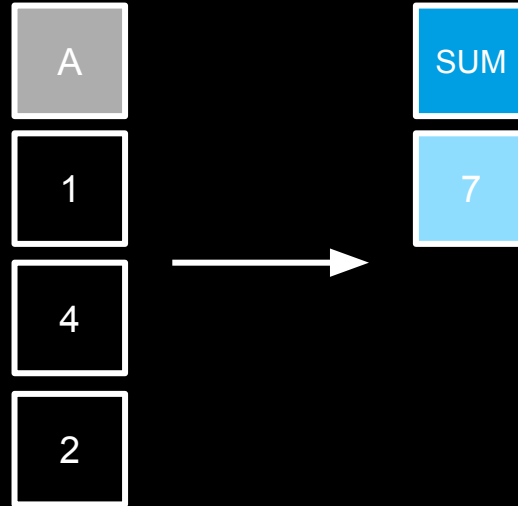
`group_by()`



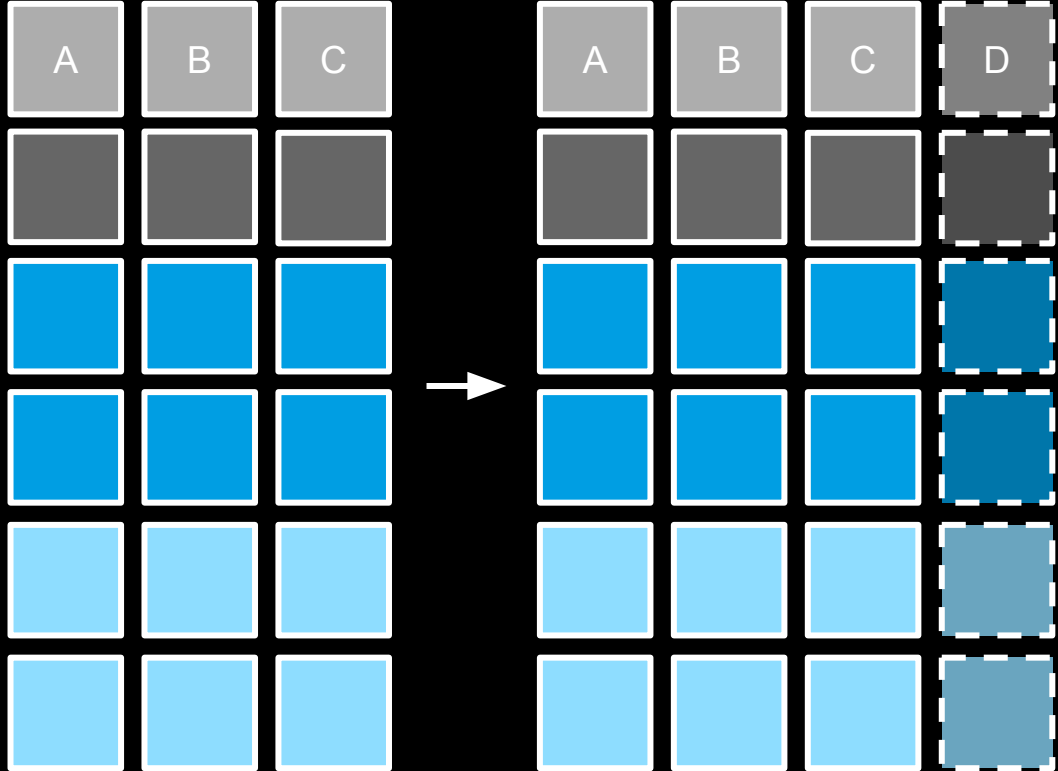
mutate()



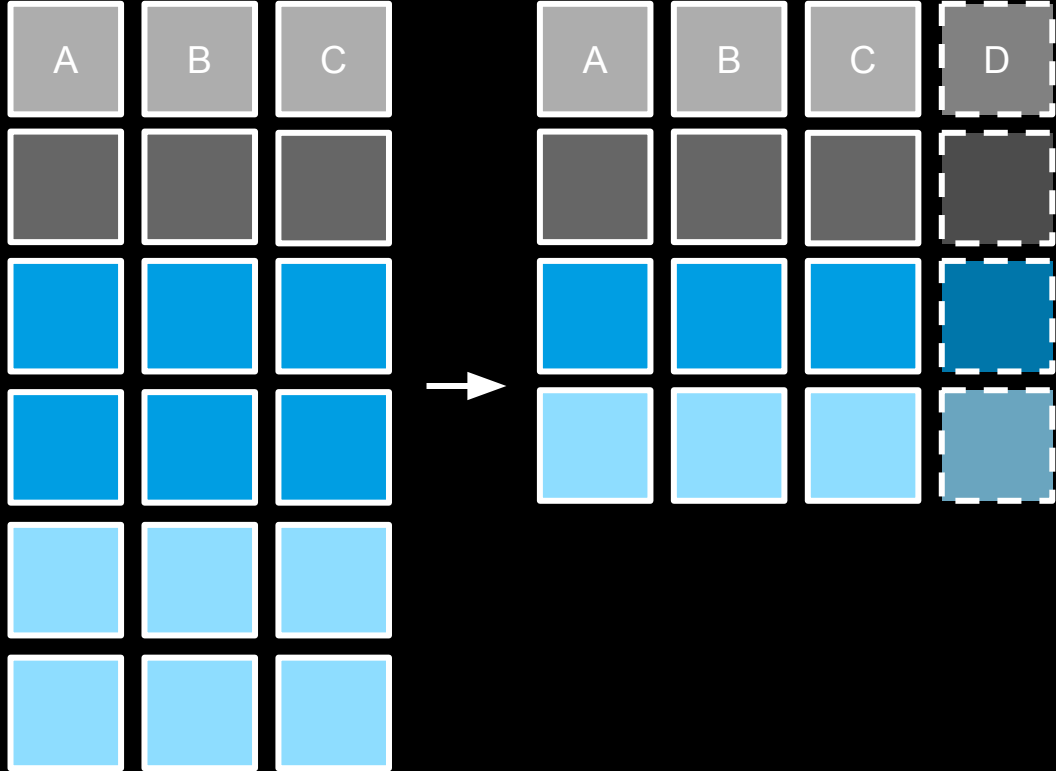
`summarize()`



`group_by()`  
+  
`mutate()`



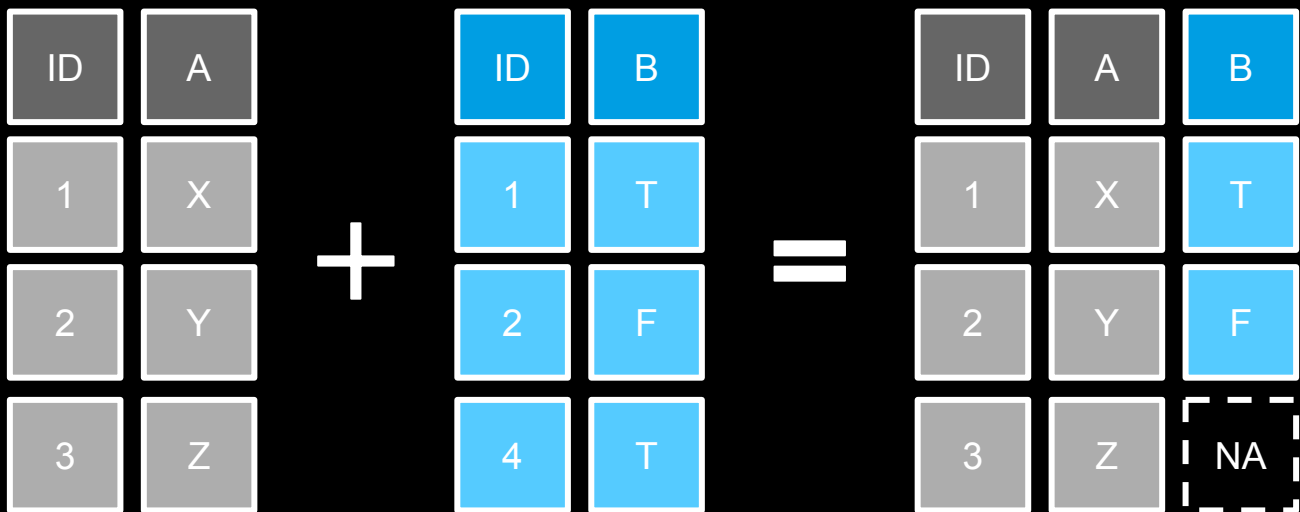
`group_by()`  
+  
`summarize()`



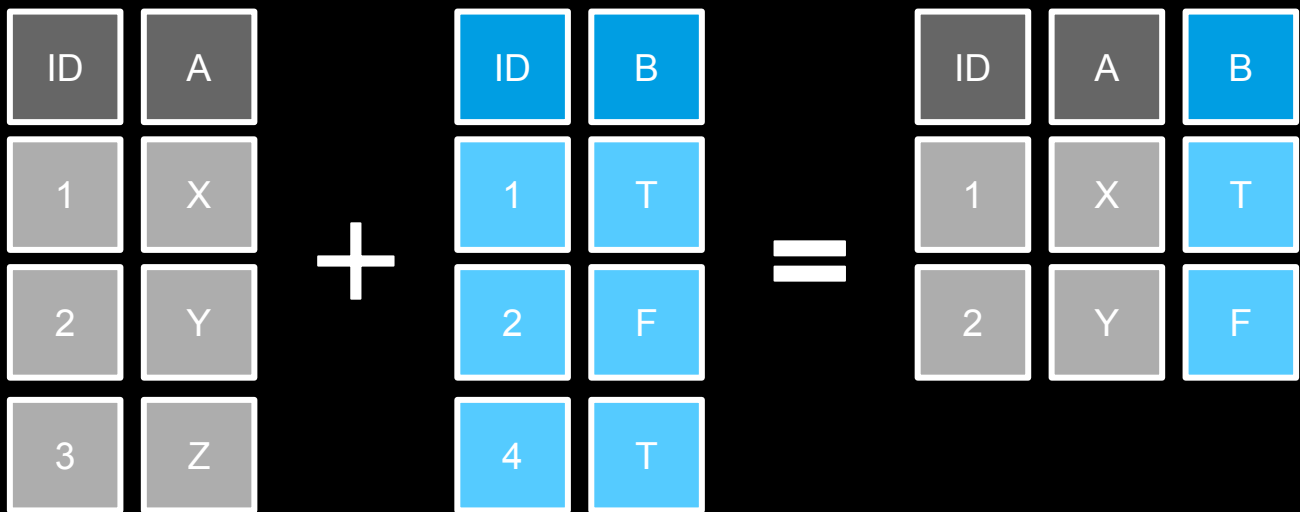
joining data



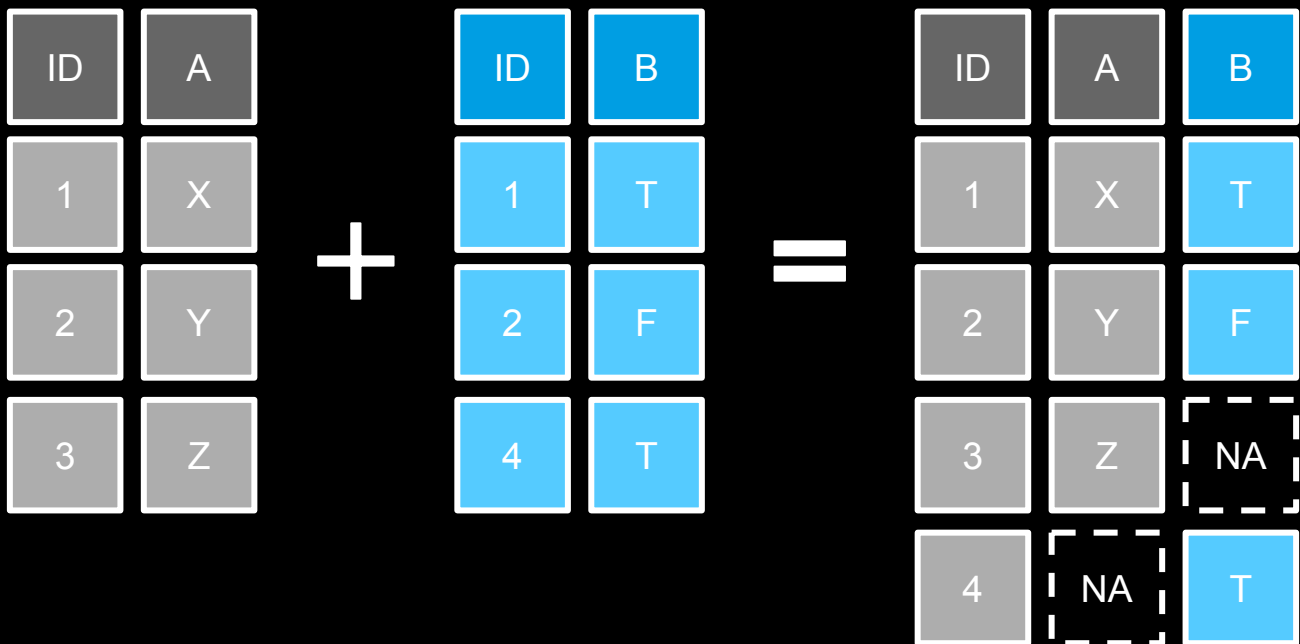
`left_join()`



`inner_join()`



```
full_join()
```

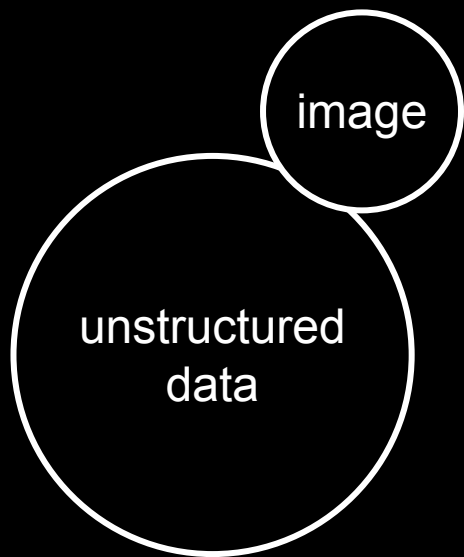


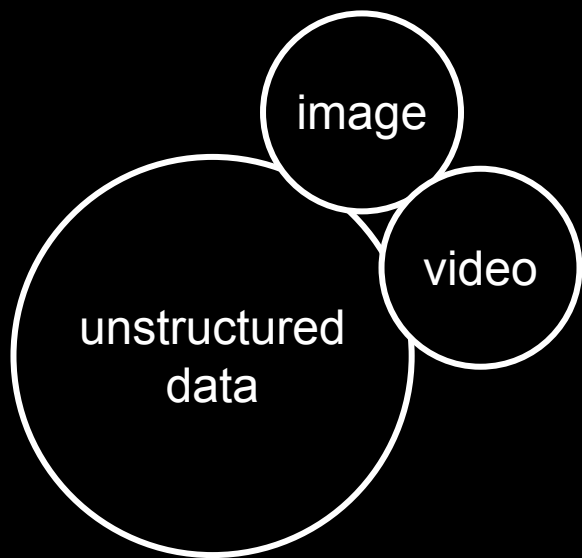
# UNSTRUCTURED DATA

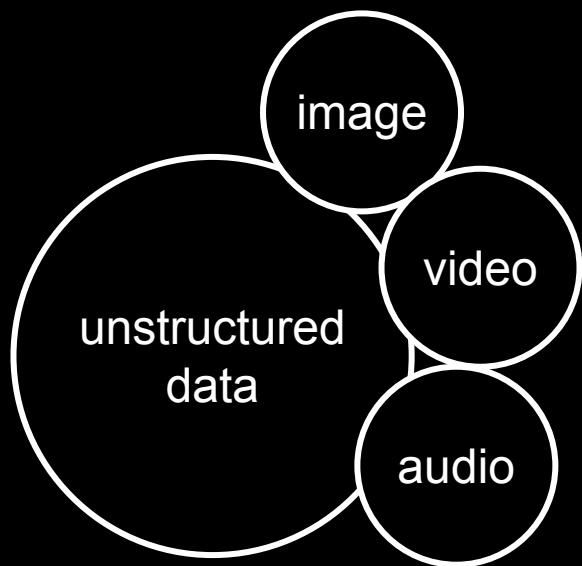


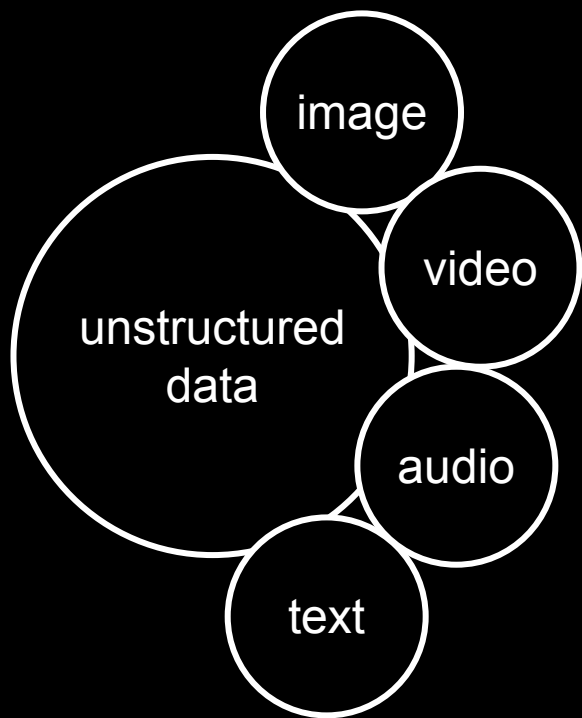
unstructured  
data



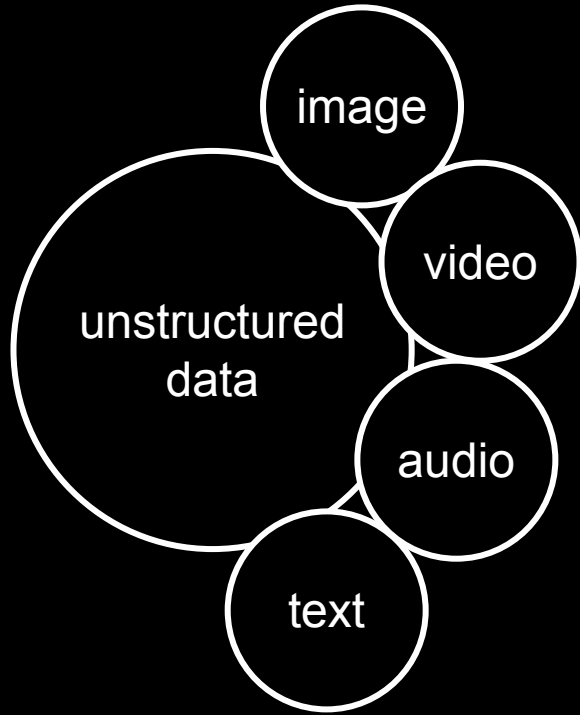




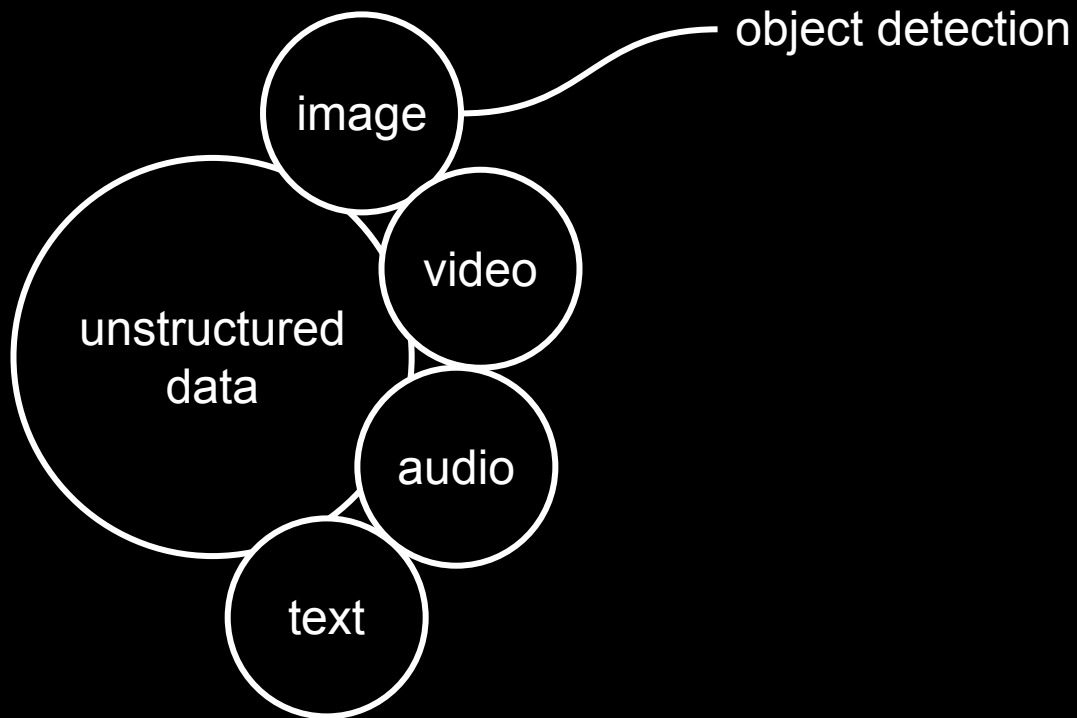




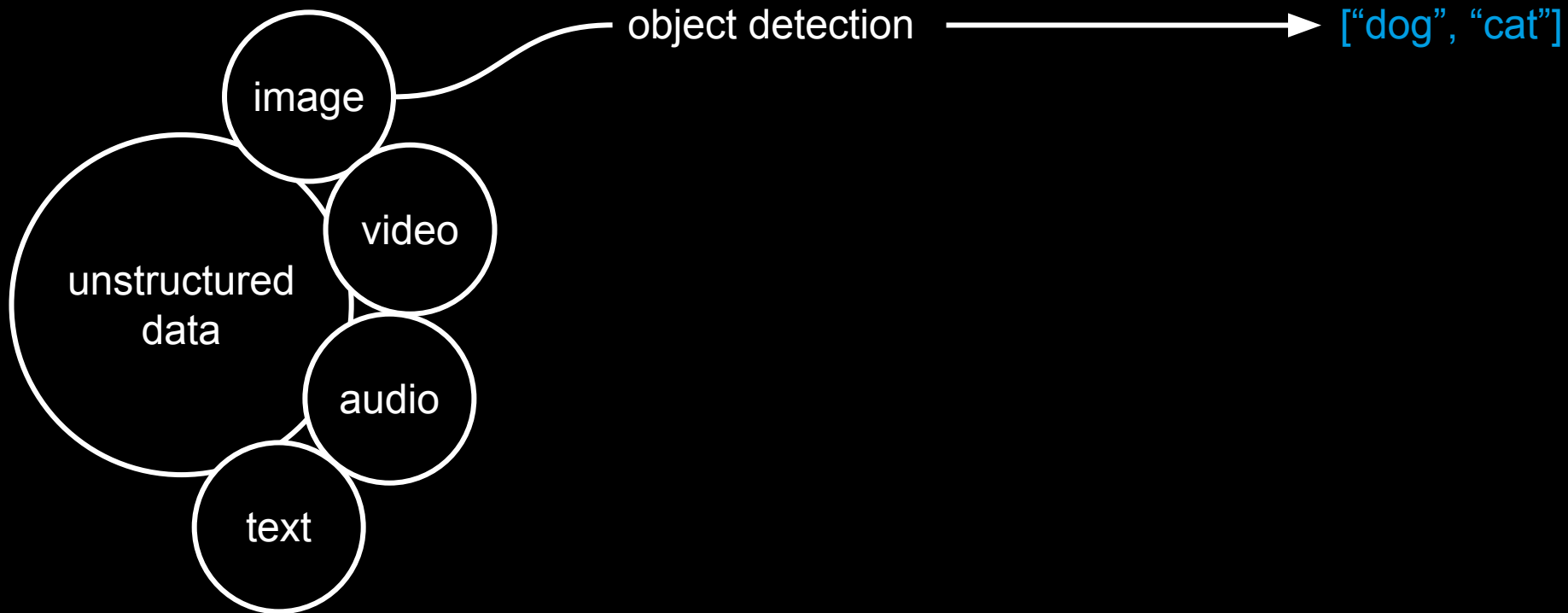
no handles to grab



no handles to grab



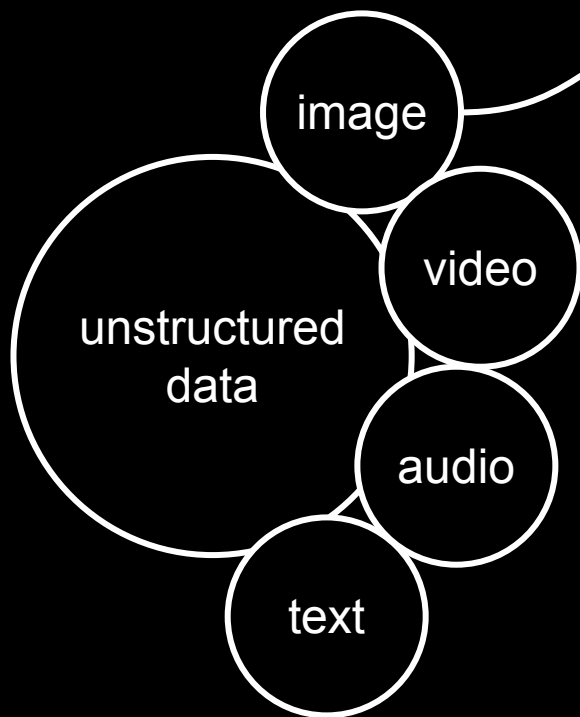
no handles to grab



no handles to grab

algorithm

extracted, structured information



object detection



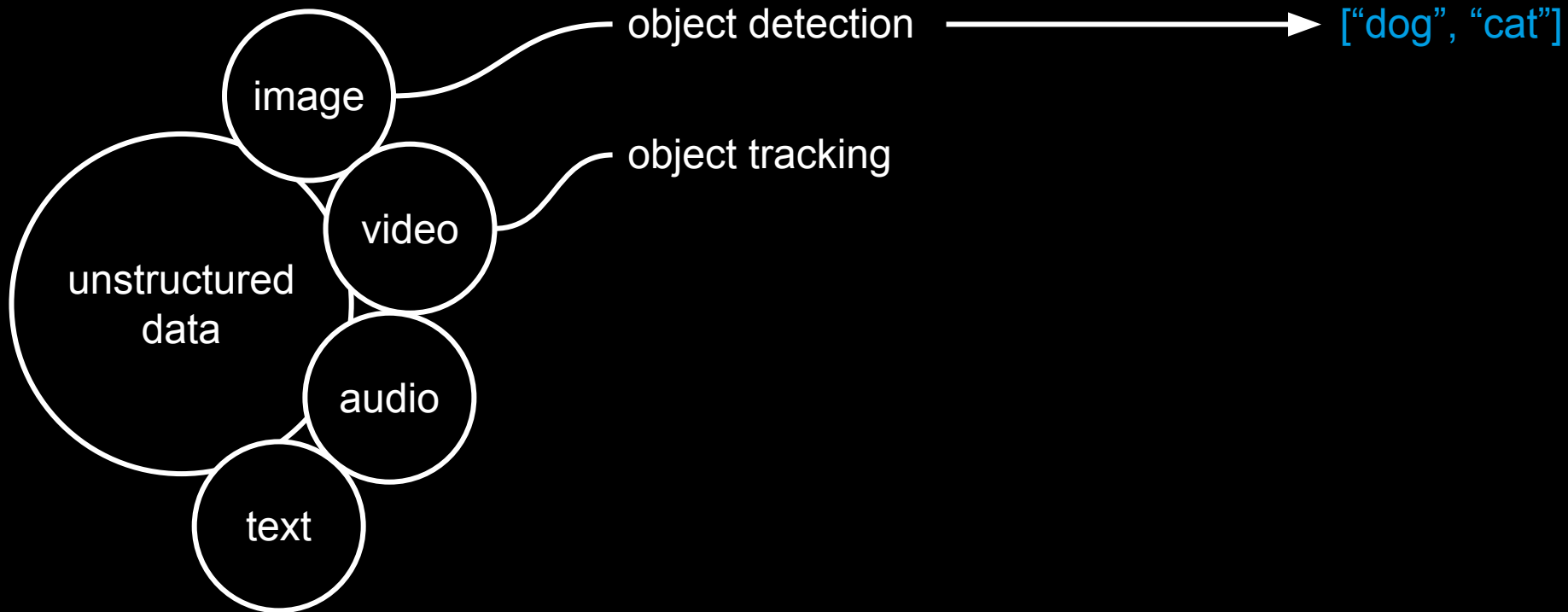
`["dog", "cat"]`



no handles to grab

algorithm

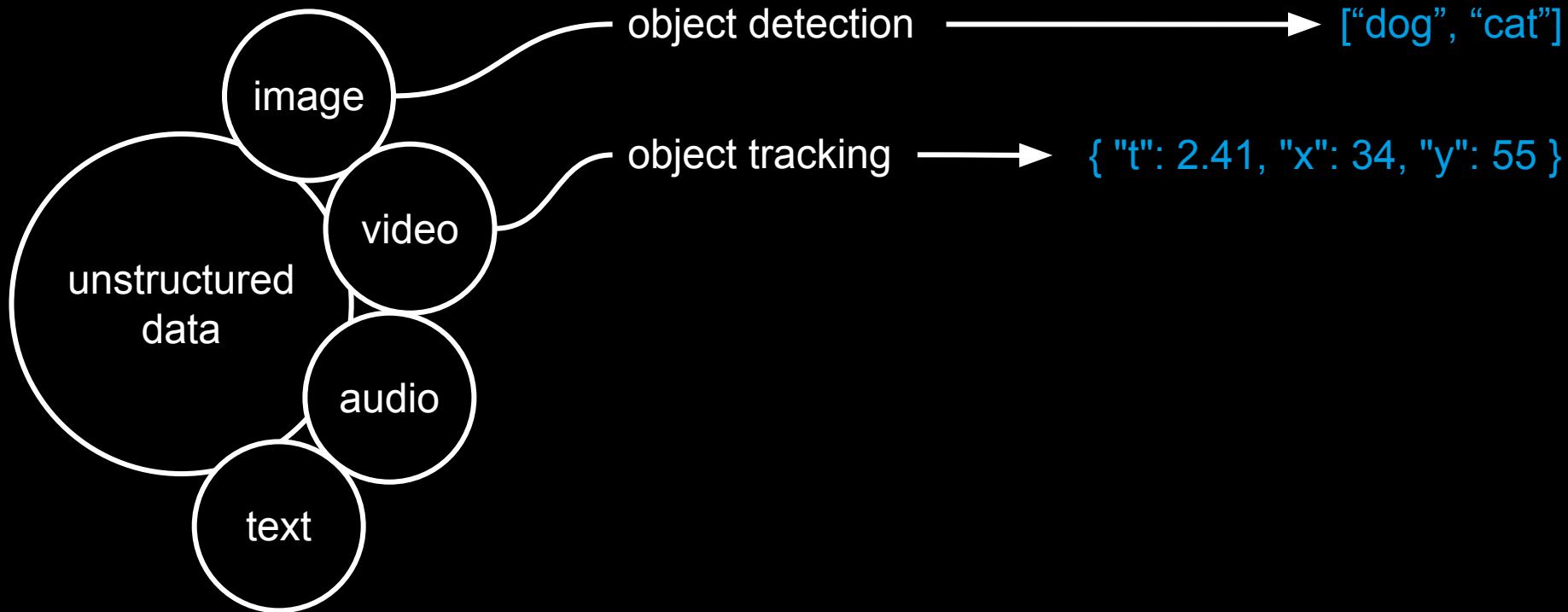
extracted, structured information



no handles to grab

algorithm

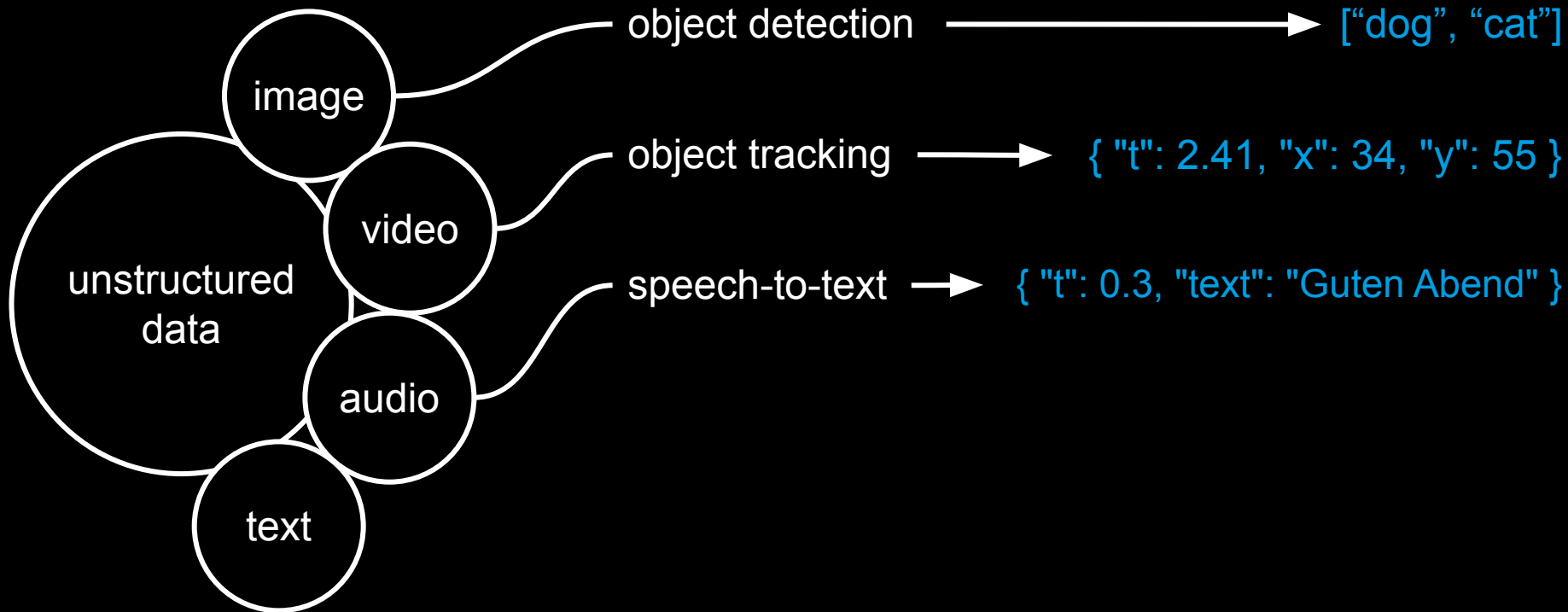
extracted, structured information



no handles to grab

algorithm

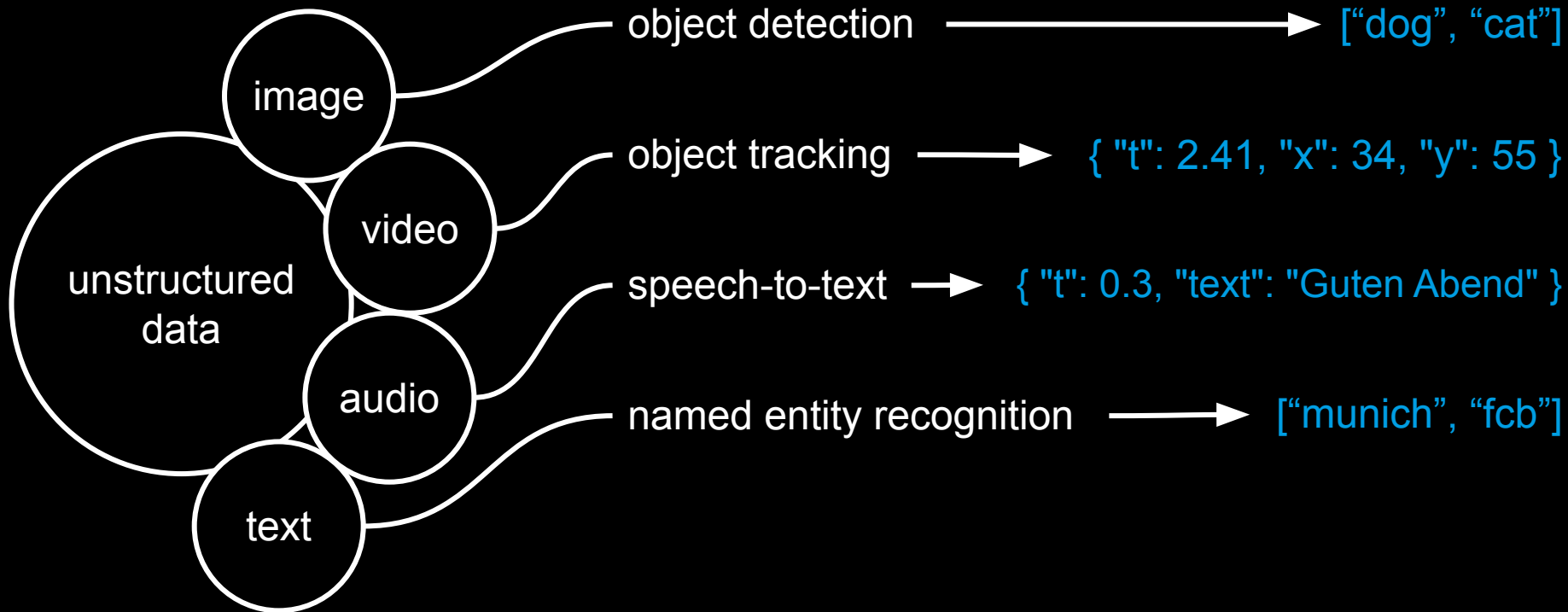
extracted, structured information



no handles to grab

algorithm

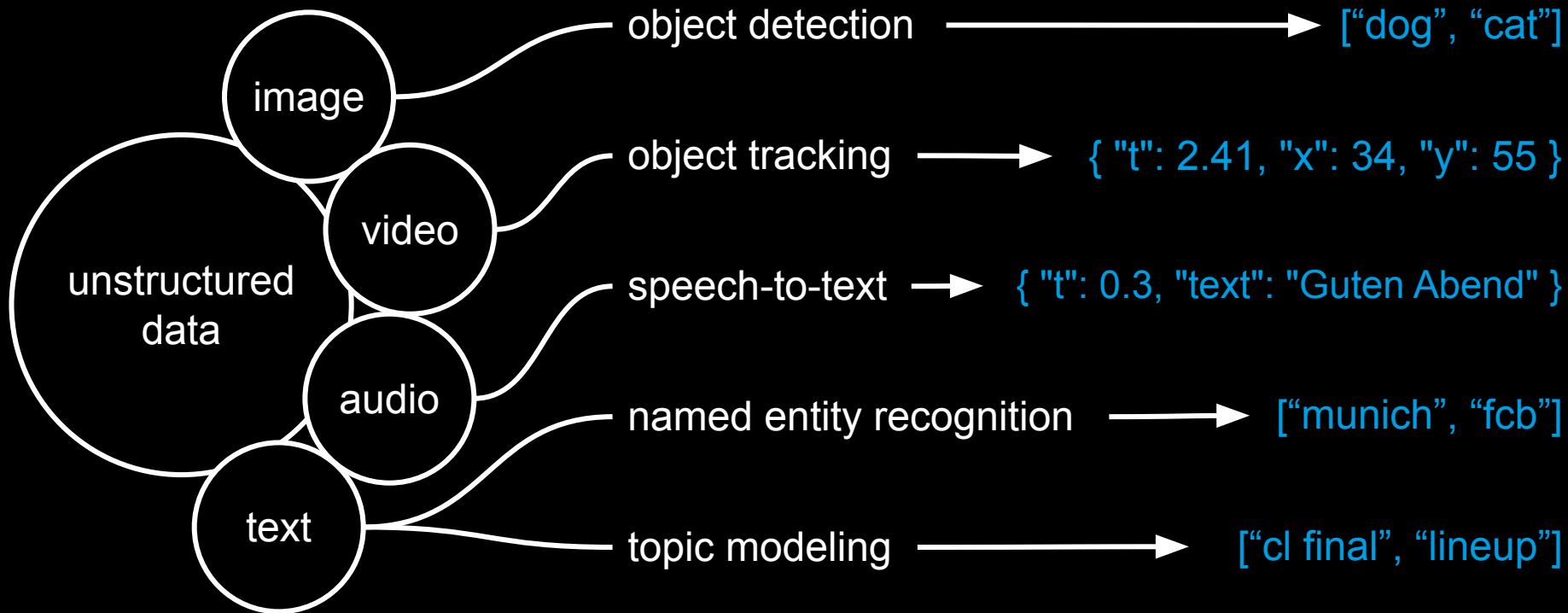
extracted, structured information



no handles to grab

algorithm

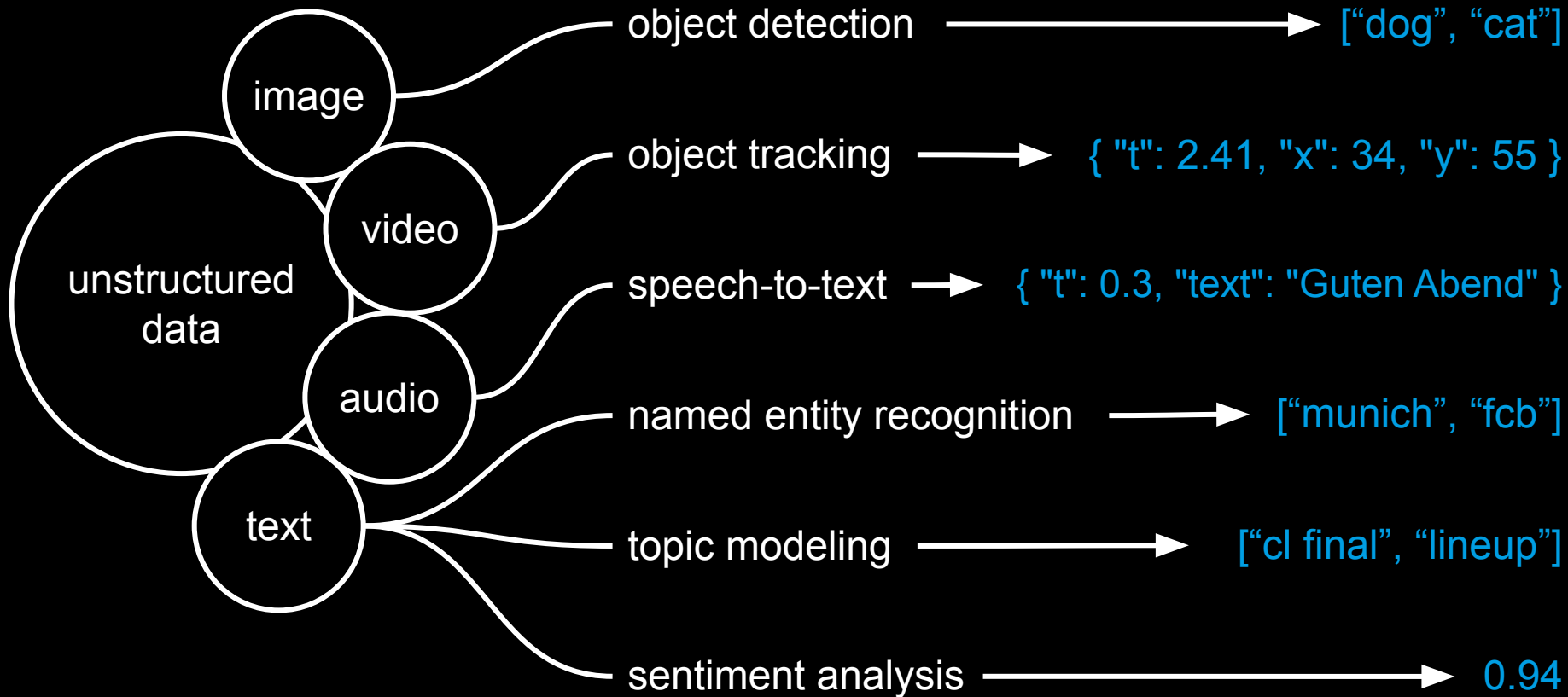
extracted, structured information



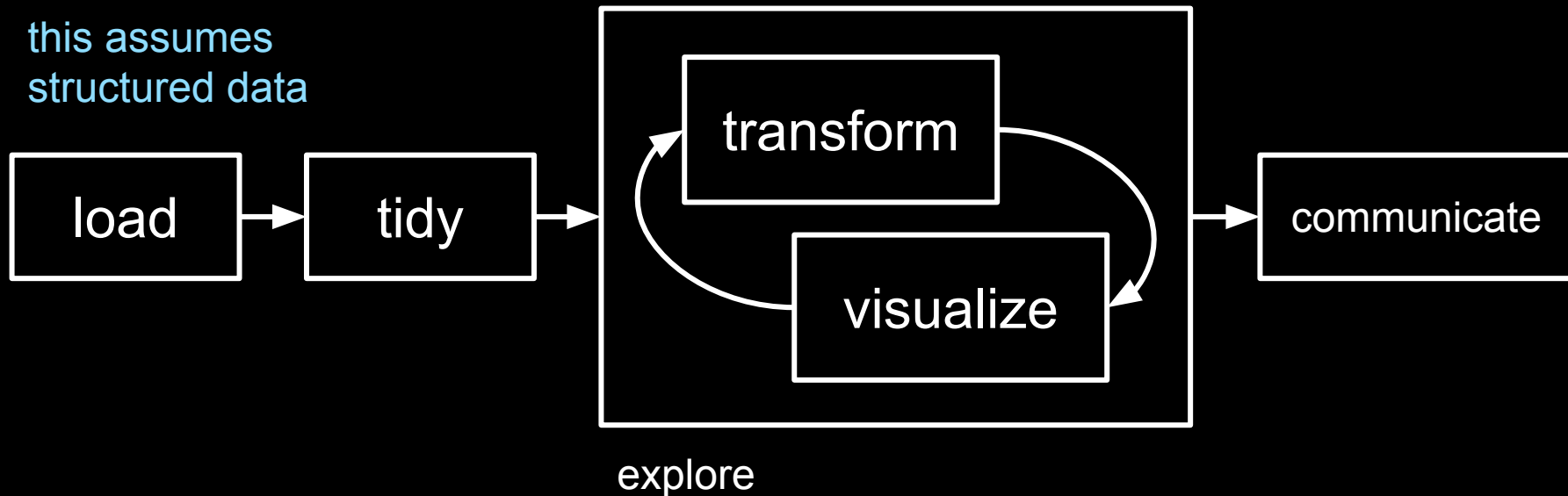
no handles to grab

algorithm

extracted, structured information

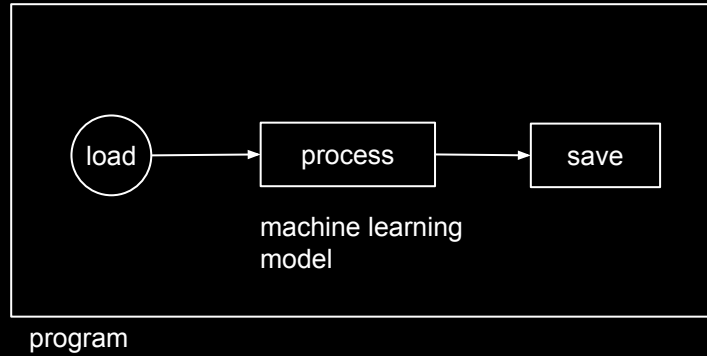


this assumes  
structured data

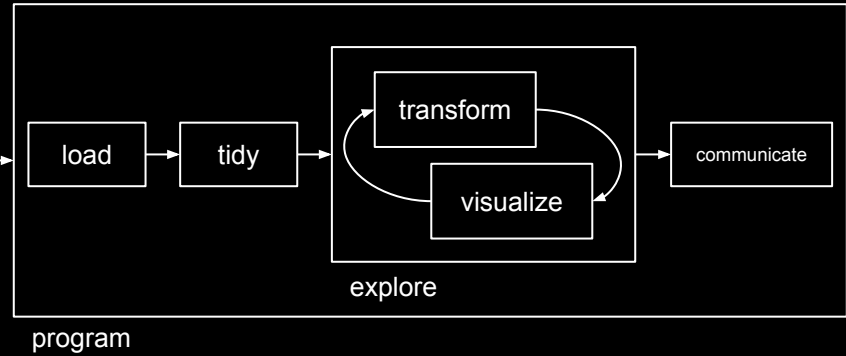


program

pre-process  
unstructured data

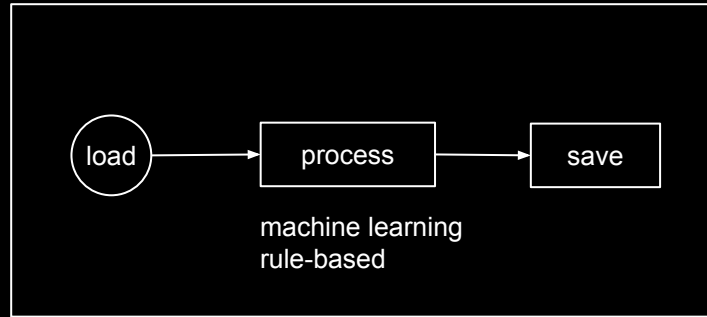


exploratory data  
analysis





pre-process  
unstructured data



program

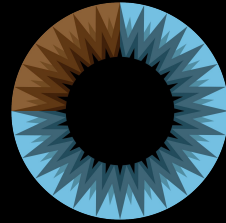


exploratory data  
analysis



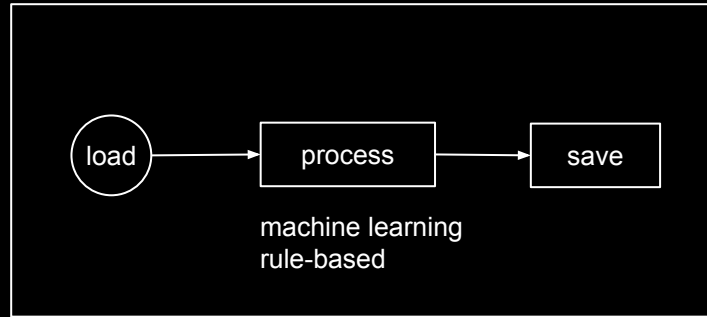
# MACHINE LEARNING

Highly recommended for  
background information



3Blue1Brown's YouTube Course on Neural  
Networks and Deep Learning

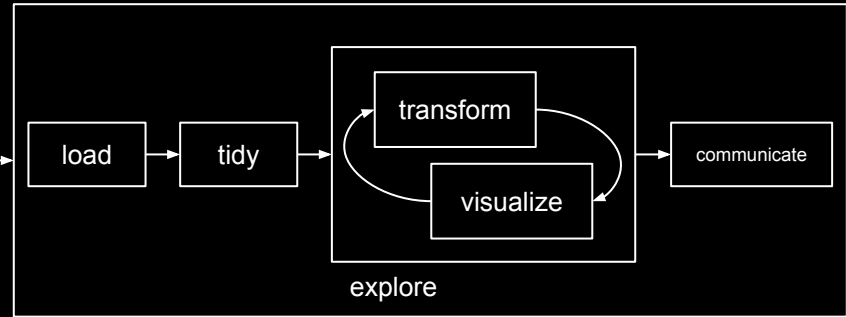
pre-process  
unstructured data



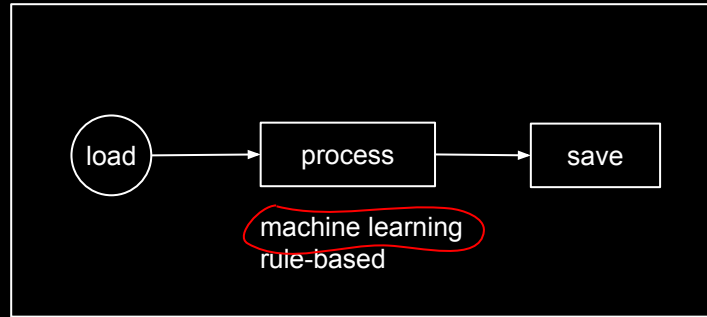
program



exploratory data  
analysis



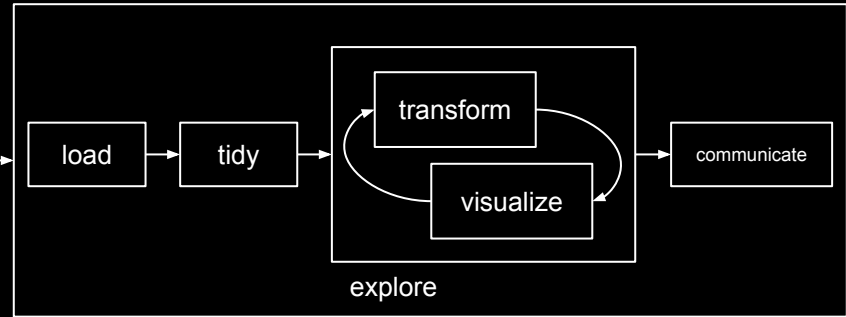
pre-process  
unstructured data



program



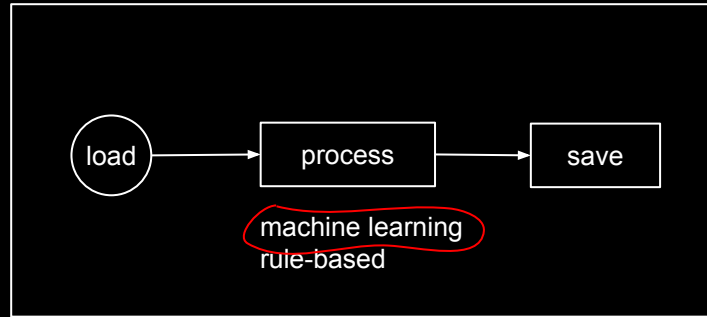
exploratory data  
analysis



program



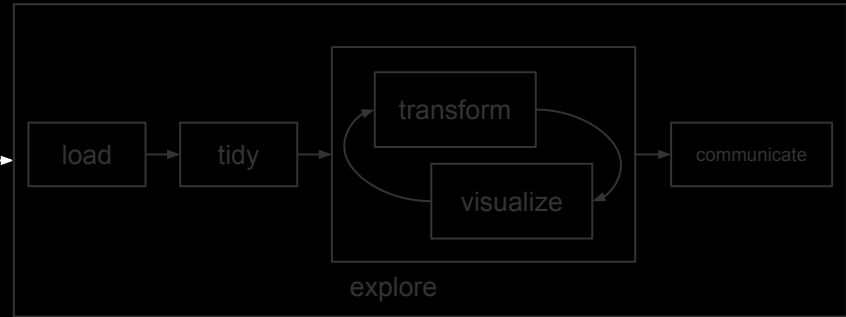
pre-process  
unstructured data



program



exploratory data  
analysis



program





machine learning

program



YouTube



process

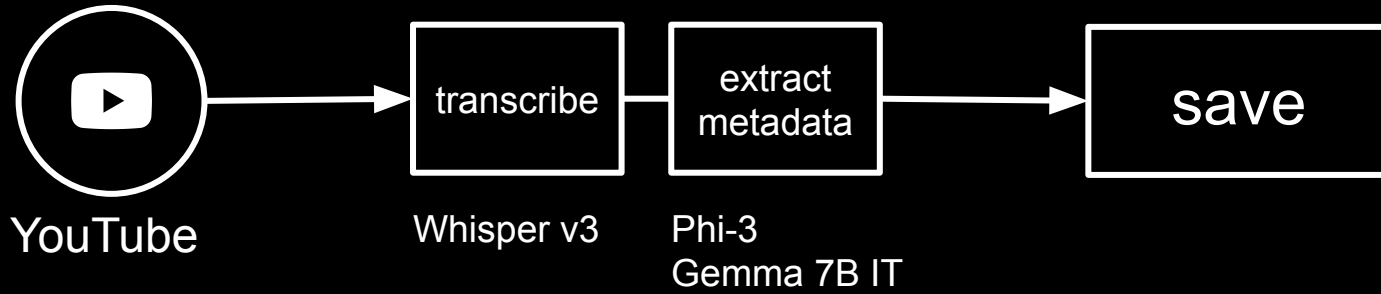
machine learning



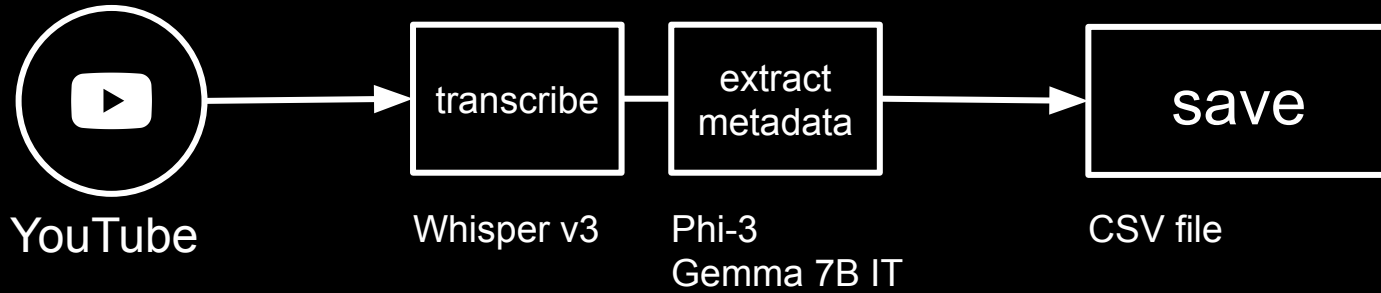
save

program





program



program

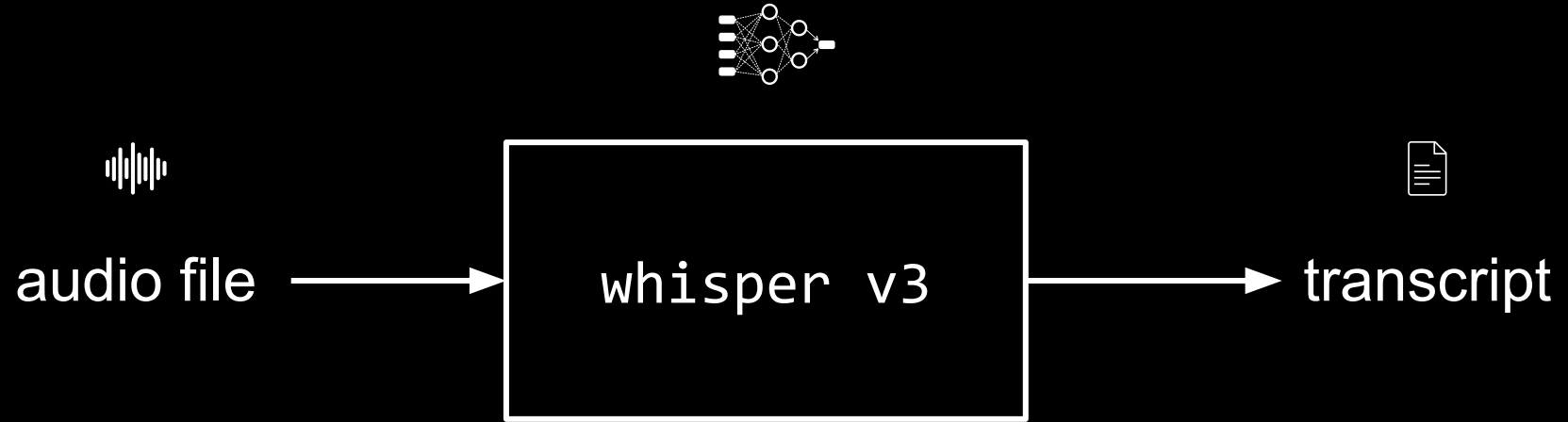
# YouTube API

# Whisper v3

<https://arxiv.org/abs/2212.04356>



<https://huggingface.co/openai/whisper-large-v3>



# Large Language Models (LLM)

what has been said so far?  
(*prompt / context*)



what has been said so far?  
(*prompt / context*)



prediction of next token based on  
learnt probability distribution

what has been said so far?  
(*prompt / context*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

what has been said so far?  
(*prompt / context*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

+

(filter)

(*discriminating, insulting content*)

what has been said so far?  
(*prompt / context*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

+

(filter)

(*discriminating, insulting content*)

next word (*token*)



what has been said so far?  
(*prompt / context*)



prediction of next token based on  
learnt probability distribution

+

(randomness)

+

(filter)

(*discriminating, insulting content*)



next word (*token*)



# PROMPTING

<https://www.promptingguide.ai/>



elements of a prompt

<instruction>

<context>

<input data>

<output indicator>



elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

example prompt

Explain the binary number system.

elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

example prompt

Explain the binary number system.

start simple

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

You are a friendly tutor and your task is to explain complex concepts as simple as possible.

Explain the binary number system.

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

You are a friendly tutor and your task is to explain complex concepts as simple as possible.

Your answers are never longer than 10 sentences.

Explain the binary number system.

# ZERO-SHOT PROMPTING

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

Classify the text into neutral,  
negative or positive.

Text: "What a great dinner!"

Sentiment:

## elements of a prompt

<instruction>

<context>

<input data>

<output indicator>

## example prompt

Classify the text into neutral,  
negative or positive.

Text: "What a great dinner!"

Sentiment:

this will be replaced with  
data later...

# FEW-SHOT PROMPTING

IN-CONTEXT LEARNING



## examples in the context to learn from

Extract all references to countries and their continent in the following text using the format from the examples below.

Example 1: "They played the team called 'Die Mannschaft' in the world cup final"

Correct answer: Germany, Europe

Example 2: "The Three Lions once again lost to Germany in a semi final"

Correct answer: England, Europe, Germany, Europe

Text: "The Selecao was destroyed 1:7 by the DFB selection in their home stadium."

Answer:

## examples in the context to learn from

Extract all references to countries and their continent in the following text using the format from the examples below.

Example 1: "They played the team called 'Die Mannschaft' in the world cup final"

Correct answer: Germany, Europe

Example 2: "The Three Lions once again lost to Germany in a semi final"

Correct answer: England, Europe, Germany, Europe

Text: "The Selecao was destroyed 1:7 by the DFB selection in their home stadium."

Answer:

more prompting strategies

chain-of-thought (CoT)

self-consistency

generate knowledge prompting

prompt chaining (subtasks)

tree-of-thoughts (ToT)

retrieval-augmented-generation (RAG)

...

# Phi-3

<https://arxiv.org/abs/2404.14219>



~~<https://huggingface.co/microsoft/Phi-3-mini-128k-instruct>~~

<https://huggingface.co/microsoft/Phi-3-medium-128k-instruct>

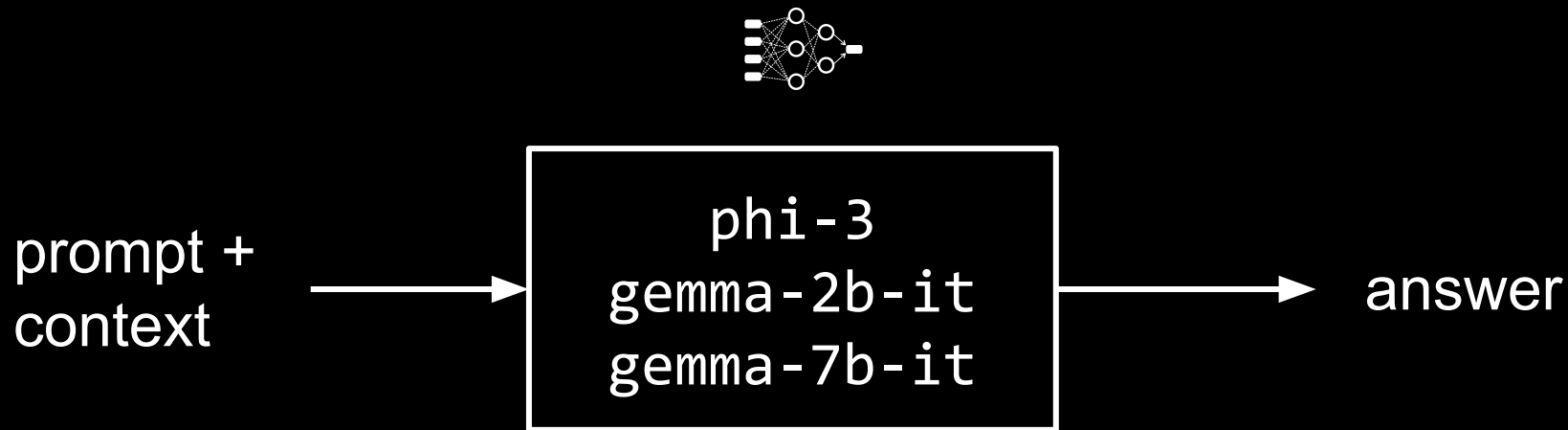
# Gemma 2B / 7B Instruct

<https://arxiv.org/abs/2403.08295>



<https://huggingface.co/google/gemma-2b-it>

<https://huggingface.co/google/gemma-7b-it>





# OpenAI's GPT-4.1 family

<https://platform.openai.com/docs/models>

4.1 nano

GPT-4.1 nano

Default

Fastest, most cost-effective GPT-4.1 model

Compare

Try in Playground

INTELLIGENCE

Average

SPEED

Very fast

PRICE

\$0.1 • \$0.4

Input • Output

INPUT

Text, image

OUTPUT

Text

GPT-4.1 nano is the fastest, most cost-effective GPT-4.1 model.

✦ 1,047,576 context window

↪ 32,768 max output tokens

🗓 Jun 01, 2024 knowledge cutoff

Pricing

Pricing is based on the number of tokens used. For tool-specific models, like search and computer use, there's a fee per tool call. See details in the [pricing page](#).

Text tokens

Per 1M tokens • Batch API price

Input

\$0.10

Cached input

\$0.025

Output

\$0.40

<https://platform.openai.com/docs/models/gpt-4.1-nano>

1 mio. token →  
~2500 pages

**GPT-4.1 nano** Default ⓘ ⓘ  
Fastest, most cost-effective GPT-4.1 model

Compare Try in Playground

INTELLIGENCE	SPEED	PRICE	INPUT	OUTPUT
● ● Average	⚡ ⚡ ⚡ ⚡ ⚡ Very fast	\$0.1 • \$0.4 Input • Output	📄 🖼️ 🗑️ Text, image	📄 🗑️ 🗑️ Text

GPT-4.1 nano is the fastest, most cost-effective GPT-4.1 model.

- ✦ 1,047,576 context window
- ↪ 32,768 max output tokens
- 🗓 Jun 01, 2024 knowledge cutoff

**Pricing**

Pricing is based on the number of tokens used. For tool-specific models, like search and computer use, there's a fee per tool call. See details in the [pricing page](#).

Text tokens Per 1M tokens • Batch API price ☐

Input	Cached input	Output
\$0.10	\$0.025	\$0.40

<https://platform.openai.com/docs/models/gpt-4.1-nano>

1 mio. token →  
~2500 pages

roughly 10 cents  
as input

The screenshot shows the OpenAI GPT-4.1 nano model page. It features a header with the model name, a 'Default' dropdown, and a 'Try in Playground' button. Below this is a comparison table with columns for Intelligence, Speed, Price, Input, and Output. The Price column shows '\$0.1 • \$0.4' for 'Input • Output'. The Input column shows 'Text, image' and the Output column shows 'Text'. A text block below the table states 'GPT-4.1 nano is the fastest, most cost-effective GPT-4.1 model.' and lists '1,047,576 context window', '32,768 max output tokens', and 'Jun 01, 2024 knowledge cutoff'. A 'Pricing' section explains that pricing is based on the number of tokens used and provides a table for 'Text tokens' with 'Per 1M tokens' and 'Batch API price' columns. The 'Input' column shows '\$0.10', 'Cached input' shows '\$0.025', and 'Output' shows '\$0.40'. Two orange arrows point from the text on the left to the '\$0.10' and '1,047,576 context window' values.

GPT-4.1 nano Default

Fastest, most cost-effective GPT-4.1 model

Compare Try in Playground

INTELLIGENCE	SPEED	PRICE	INPUT	OUTPUT
 Average	 Very fast	\$0.1 • \$0.4 Input • Output	 Text, image	 Text

GPT-4.1 nano is the fastest, most cost-effective GPT-4.1 model.

- ✦ 1,047,576 context window
- ↪ 32,768 max output tokens
- 📅 Jun 01, 2024 knowledge cutoff

Pricing

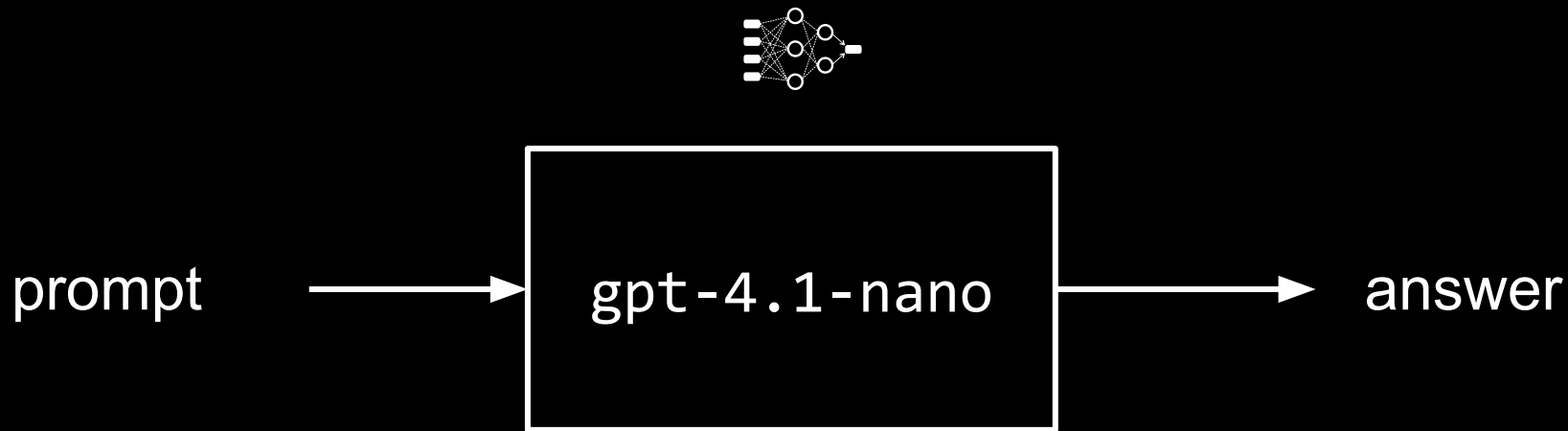
Pricing is based on the number of tokens used. For tool-specific models, like search and computer use, there's a fee per tool call. See details in the [pricing page](#).

Text tokens

Per 1M tokens • Batch API price ☐

Input \$0.10	Cached input \$0.025	Output \$0.40
-----------------	-------------------------	------------------

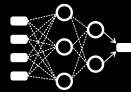
<https://platform.openai.com/docs/models/gpt-4.1-nano>



prompt +  
image



answer



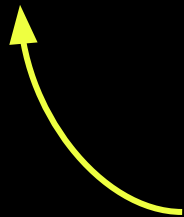
prompt +  
image



gpt-4.1-nano



answer



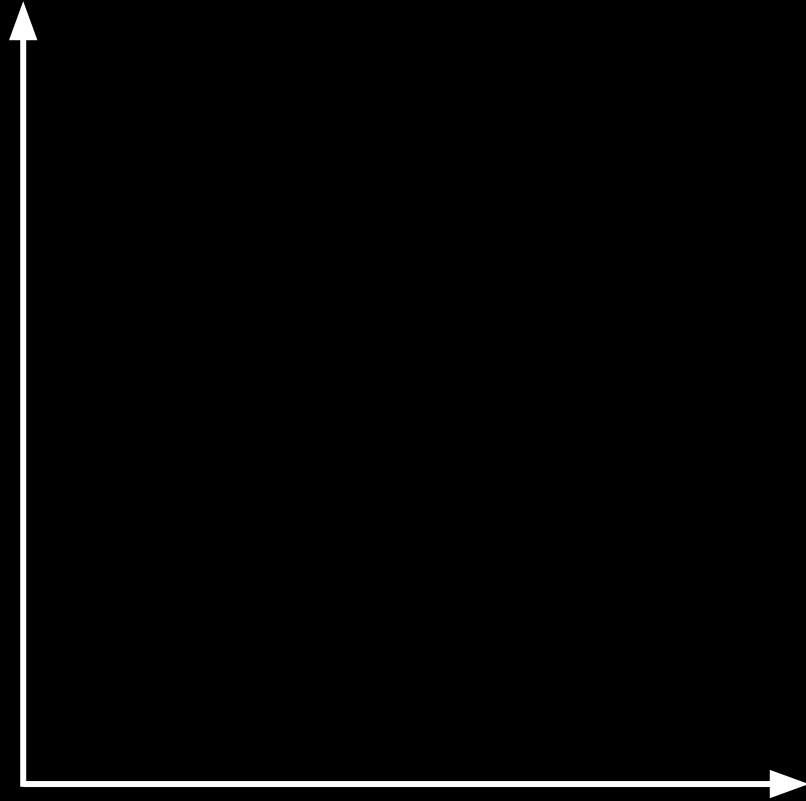
multimodality

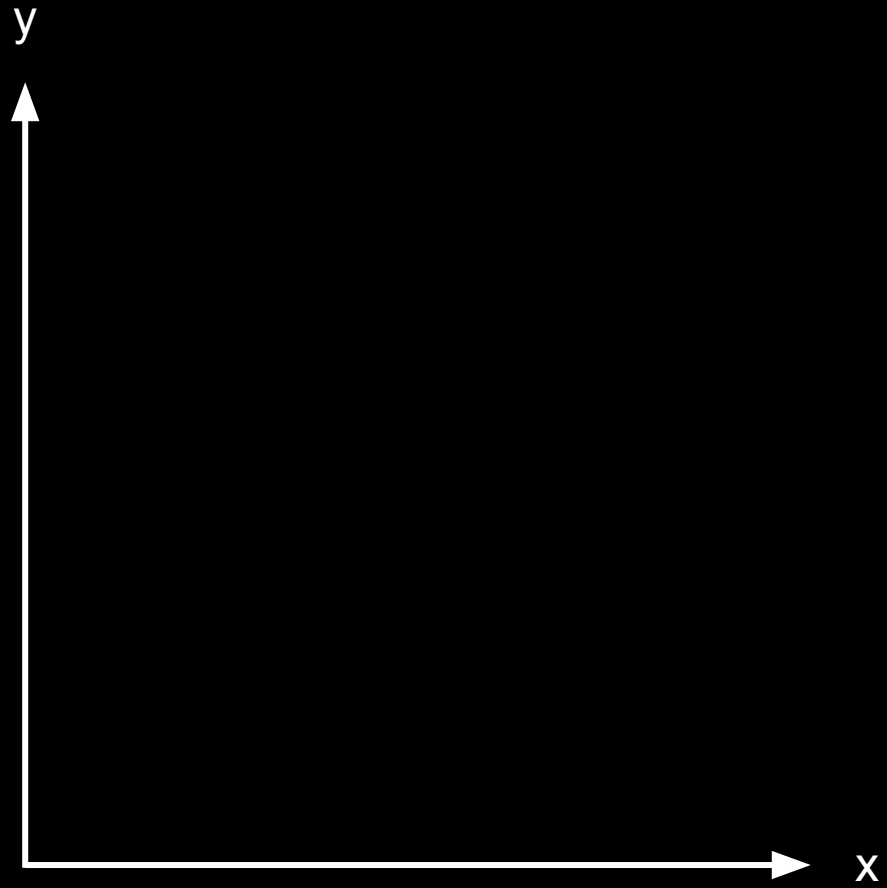
# VISUALIZE DATA

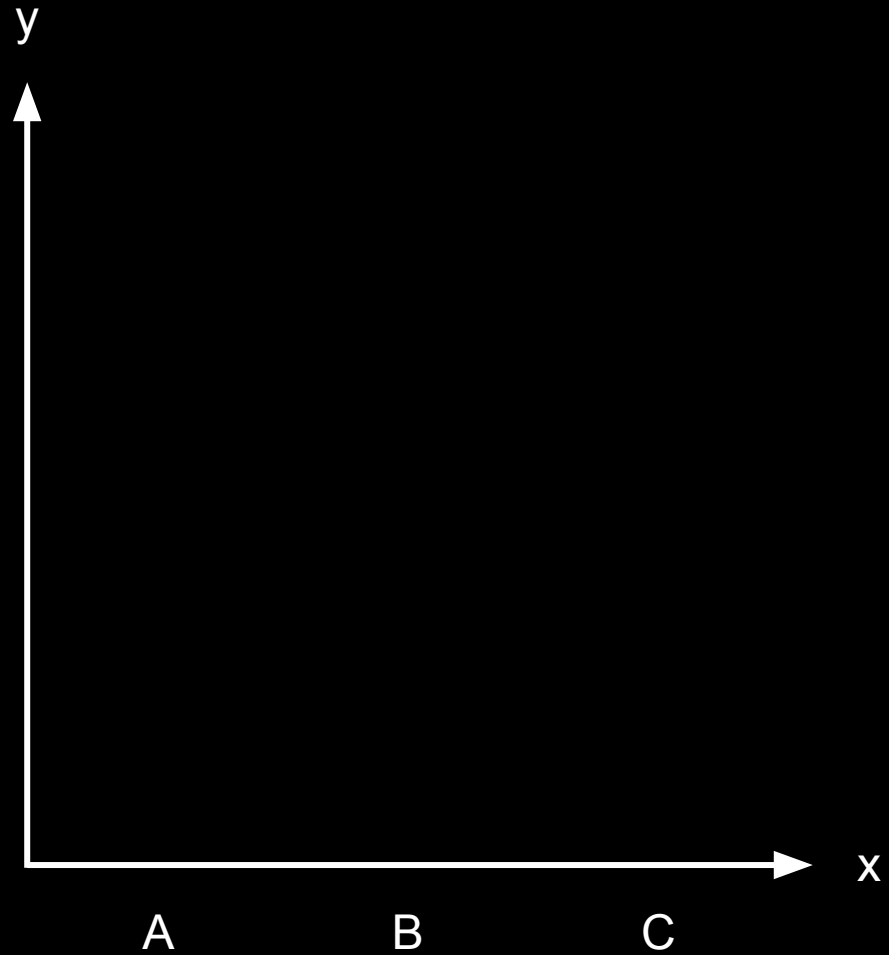


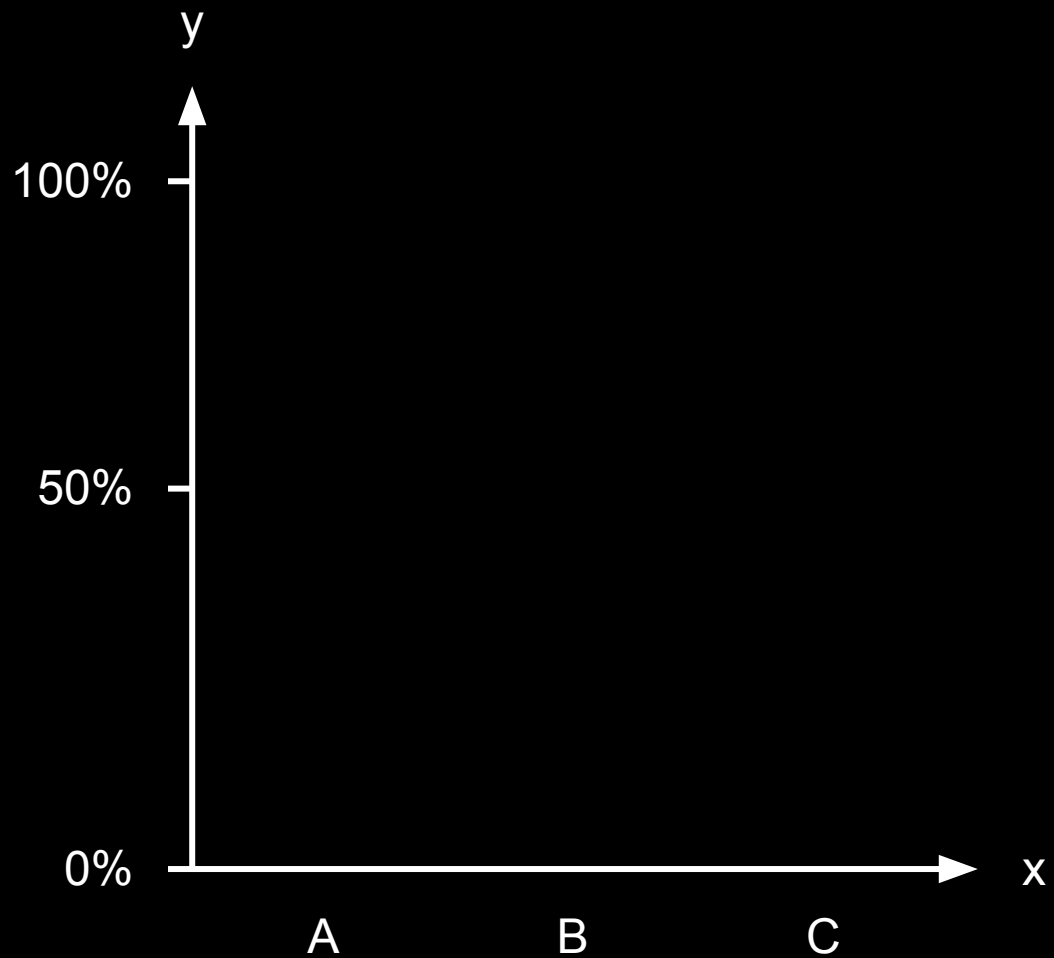
# data

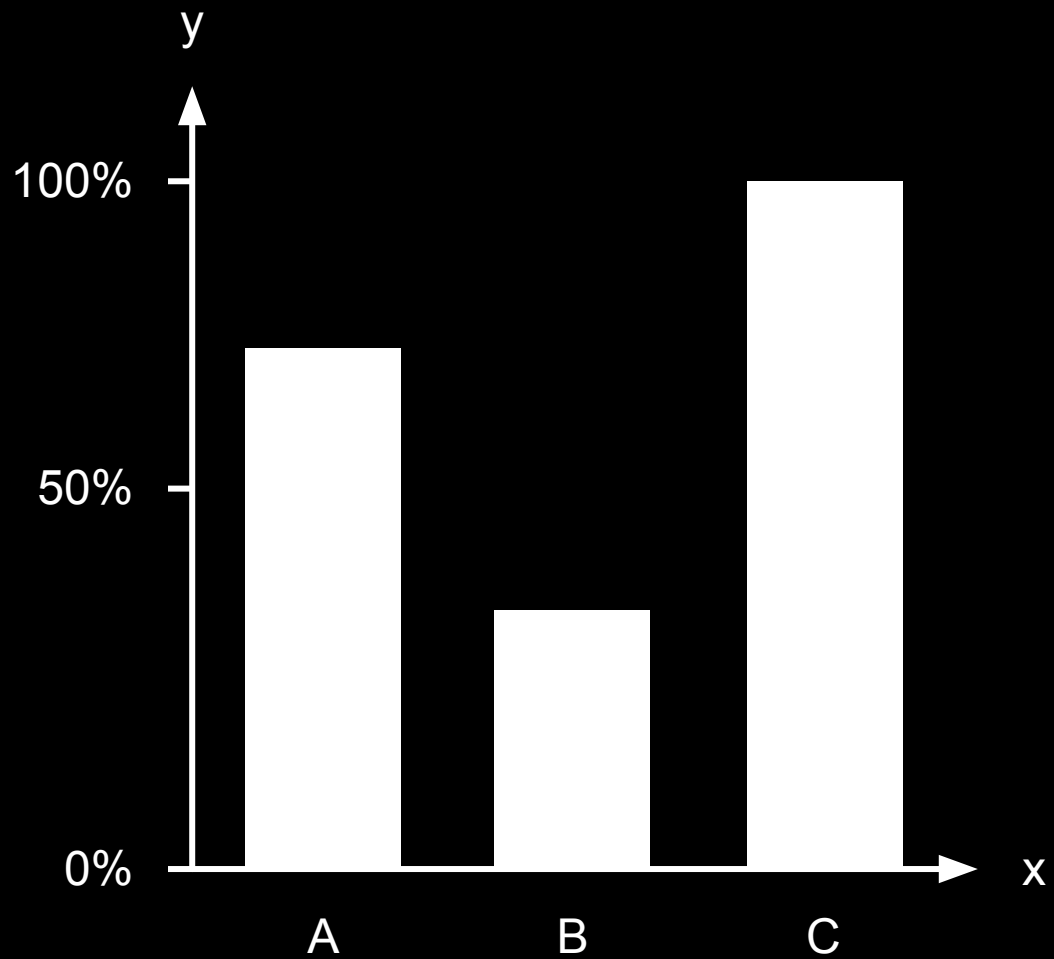
category	pct
A	75
B	33
C	100



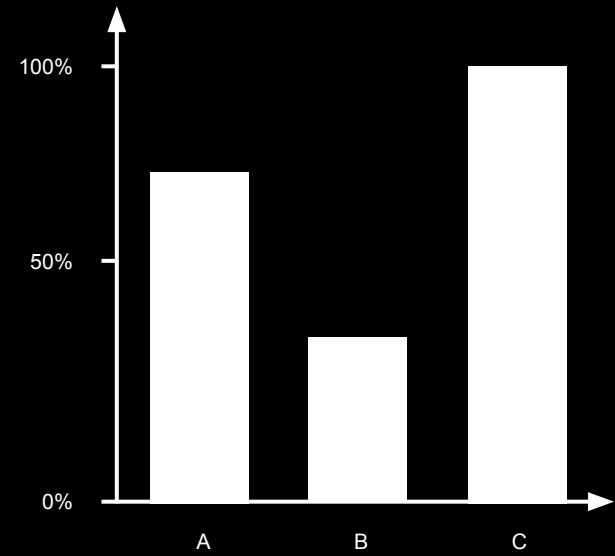








category	pct
A	75
B	33
C	100



{{ ggplot2 }}

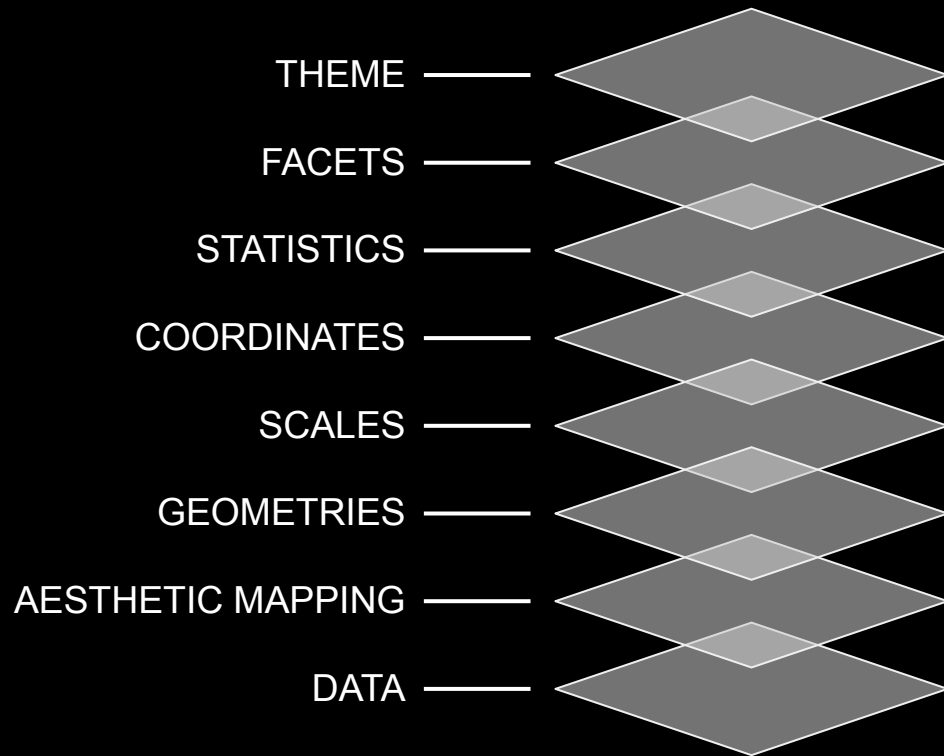


why visualize?

{{ ggplot2 }}

grammar of graphics

any  
data  
visualization



has useful defaults

mandatory



THEME

FACETS

STATISTICS

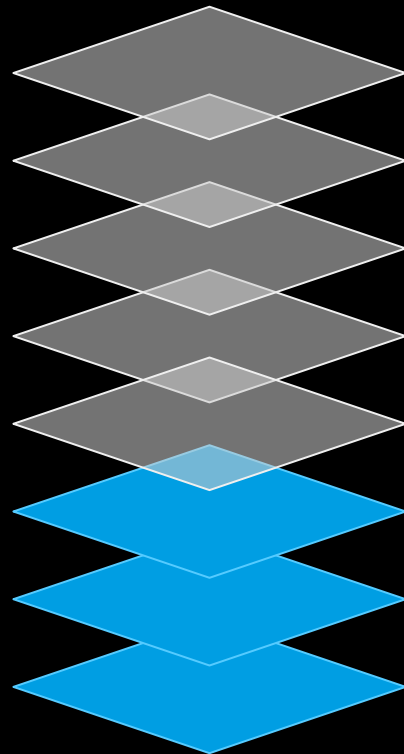
COORDINATES

SCALES

GEOMETRIES

AESTHETIC MAPPING

DATA

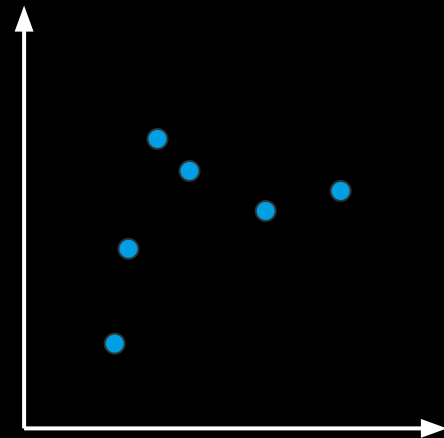


`ggplot()`

```
ggplot() +  
  aes()
```

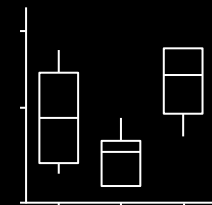
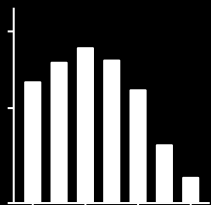
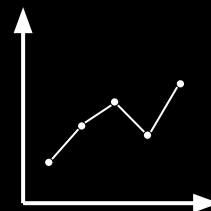
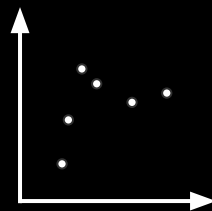
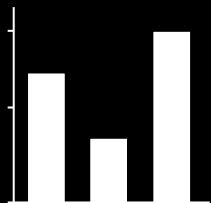
```
ggplot() +  
  aes() +  
  geom_point()
```

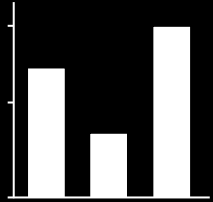
```
ggplot() +  
  aes() +  
  geom_point()
```



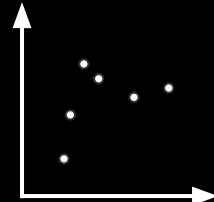


basic plots

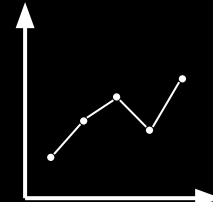




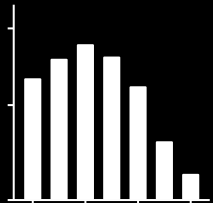
bar chart



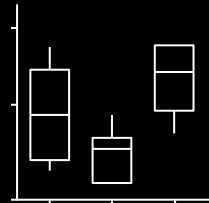
scatter plot



line chart

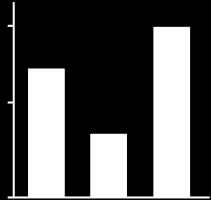


histogram



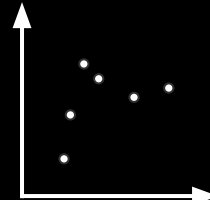
box plot

amounts  
proportions  
distributions (discrete)



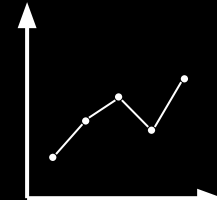
bar chart

associations  
patterns



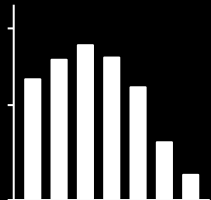
scatter plot

trends  
developments



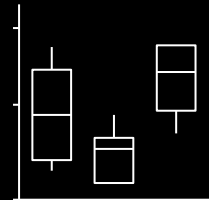
line chart

distributions (continuous)



histogram

compare distributions (continuous)



box plot

# COMMUNICATE DATA

quarto

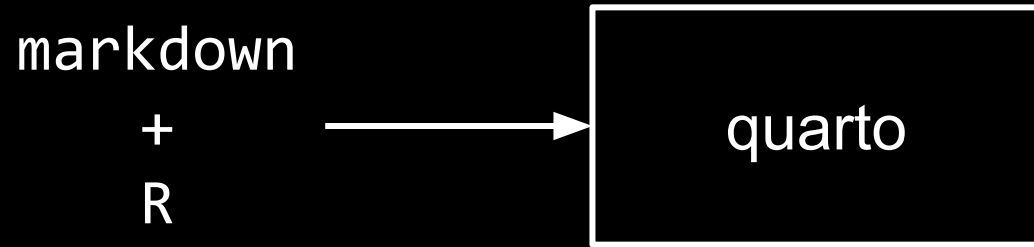
{{ quarto }}

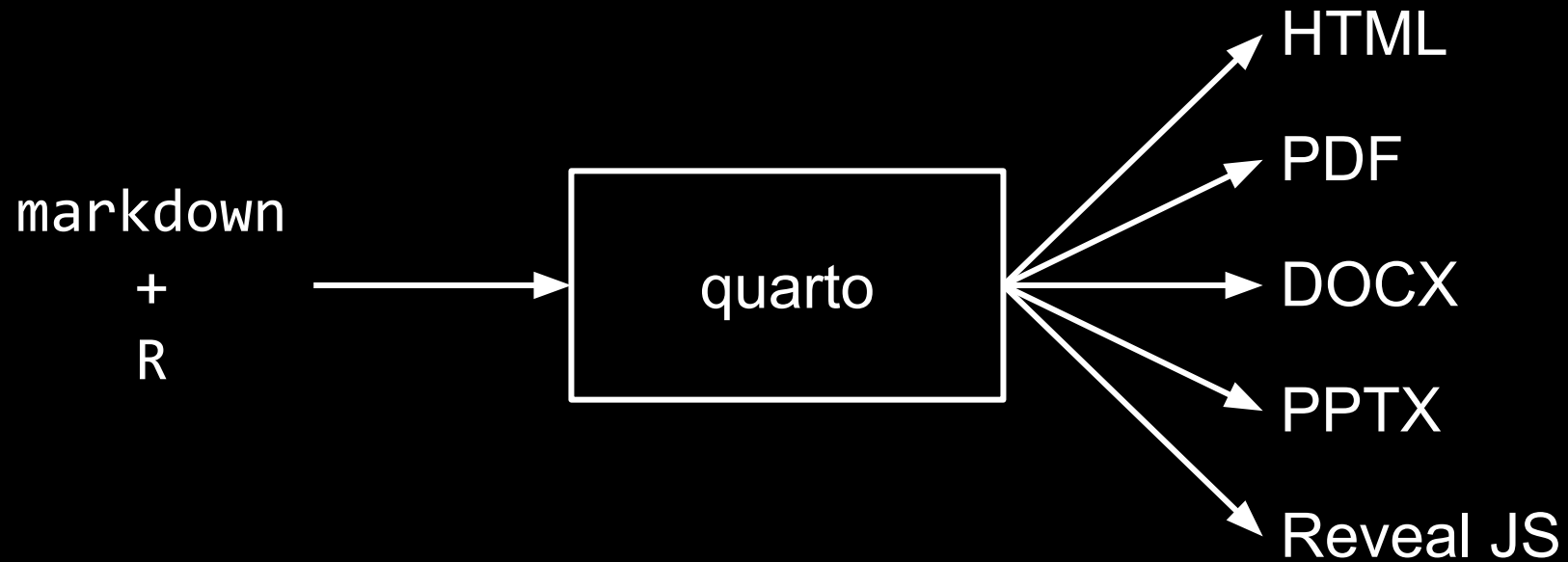
markdown

+

R







markdown

# Heading 1

## Heading 2

### Heading 3

#### Heading 4

This is *\*italic\**,

and this **\*\*bold\*\***

This is rendered  
as `code`.

- List item A
- List item B
- List item C

1. First

2. Second

3. Third



`![Image title](path/to/image.png)`

```
![Image title](path/to/image.png){width=200}
```

```
![Image title](path/to/image.png){#fig-myimage}
```

For more details see [@fig-myimage](#).

[Linked text](https://quarto.org)

...{r}

1 + 1

...

code options

```
` `` {r}
```

```
#| echo: false
```

```
1 + 1
```

```
` `` `
```

```
` `` {r}
```

```
#| eval: false
```

```
x <- 1 + 1
```

```
` ``
```



```
` `` {r}
```

```
#| include: false
```

```
library(tidyverse)
```

```
` ``
```

```
` `` {r}
```

```
#| message: false
```

```
data <- read_csv("data.csv")
```

```
` ``
```

```
` `` {r}
```

```
#| warning: false
```

```
data <- read_csv("data.csv")
```

```
` ``
```

figures

```
` `` {r}  
#| label: fig-tweets-per-user  
#| fig-cap: "Tweets per User"  
tweets |>  
  ggplot() +  
  aes(x = screen_name) +  
  geom_bar()  
` ``
```

```
` `` {r}  
#| code-fold: true  
#| code-summary: "Show code"  
tweets |>  
  ggplot() +  
  aes(x = screen_name) +  
  geom_bar()  
` ``
```

cross references

```
```{r}
#| label: fig-tweets-per-user
#| fig-cap: "Tweets per User"
...
```
```

In `@fig-tweets-per-user` you can see an overview of the number of tweets per user in the data set.



```
# Introduction {#sec-introduction}
```

```
...
```

```
# Analysis
```

```
As stated in @sec-introduction, the goal of this paper is  
to analyze the user behavior with regard to the content  
they tweet.
```

# citation & bibliography

<https://quarto.org/docs/authoring/citations.html>

output formats