Data Cleaning

METIS®

Objectives

Develop data cleaning strategies:

- Handling missing values
- Tidying string data
- Cleaning datasets through case studies

What is
Data Cleaning?

METIS®

Data Analysis Workflow

What does the data analysis workflow look like?

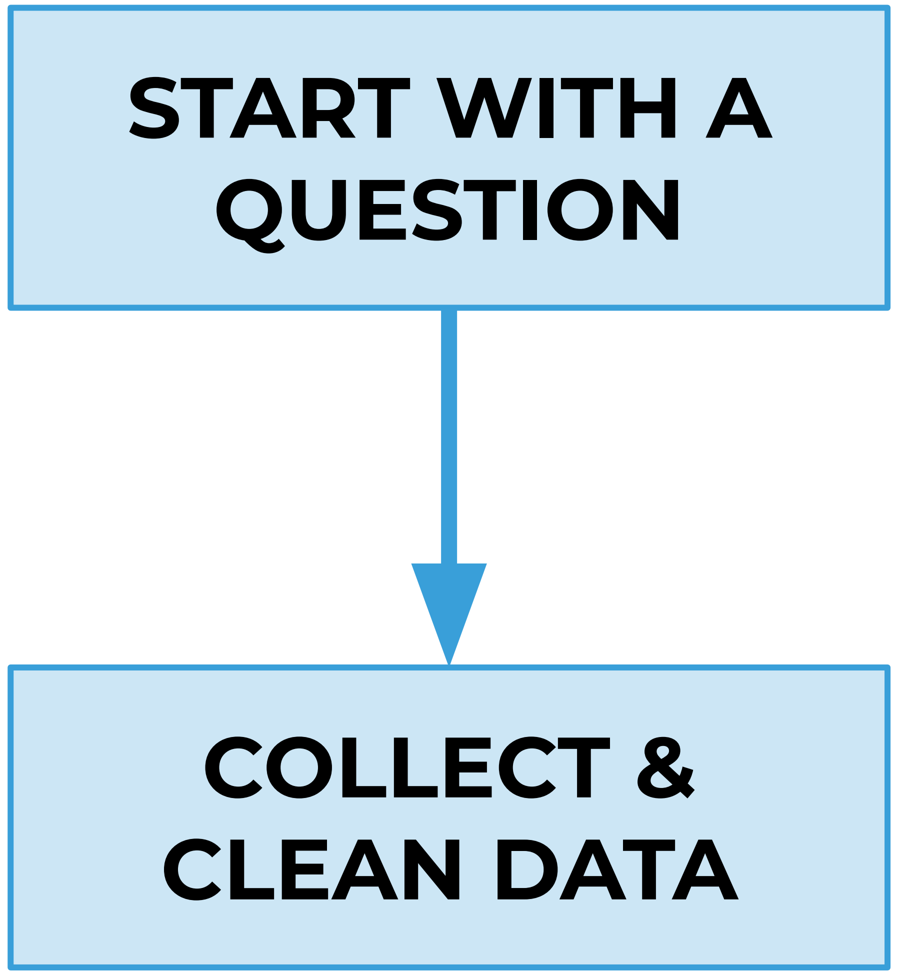
Data Analysis Workflow

What does the data analysis workflow look like?

START WITH A QUESTION

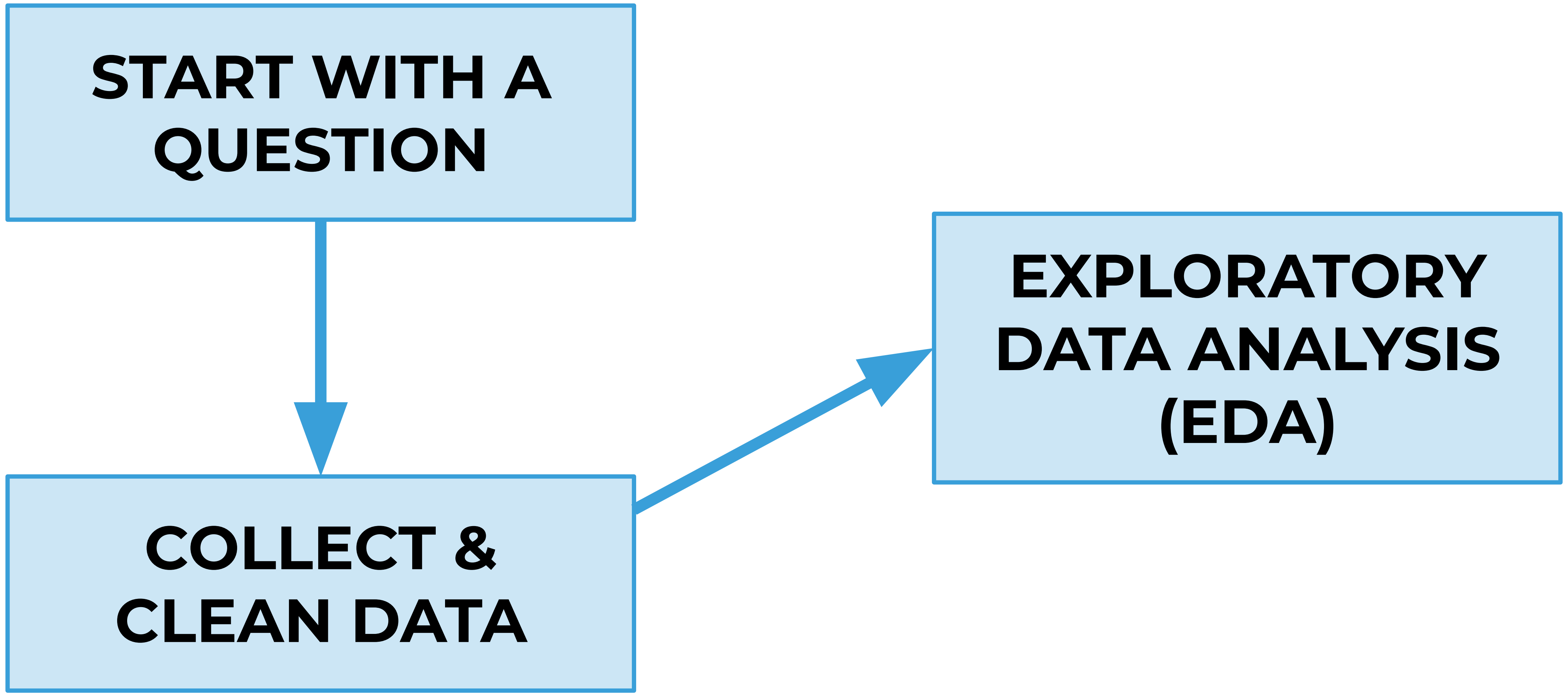
Data Analysis Workflow

What does the data analysis workflow look like?



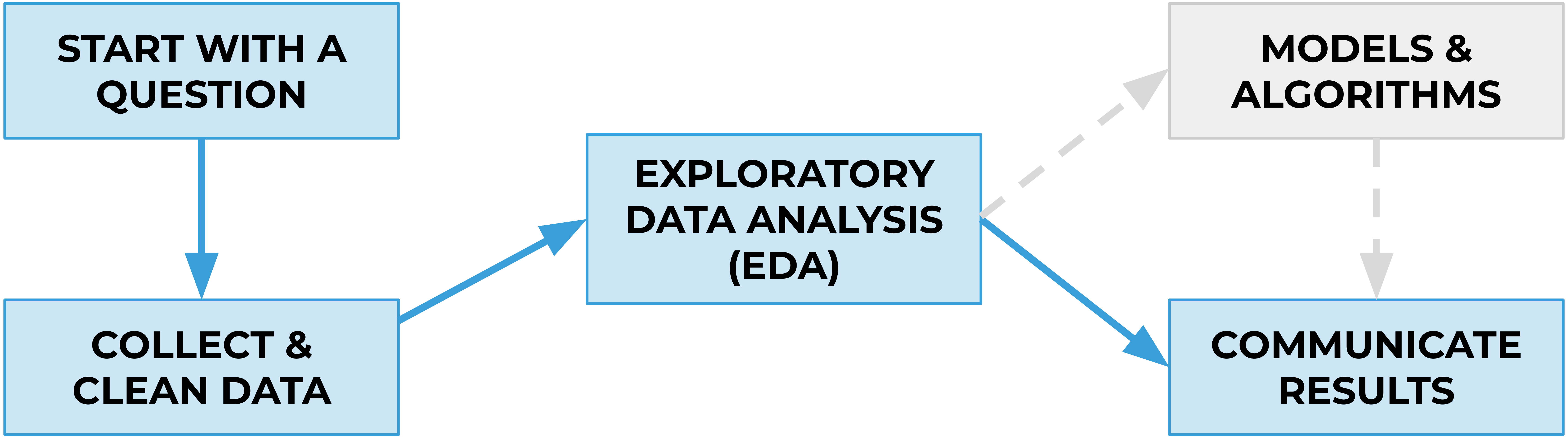
Data Analysis Workflow

What does the data analysis workflow look like?



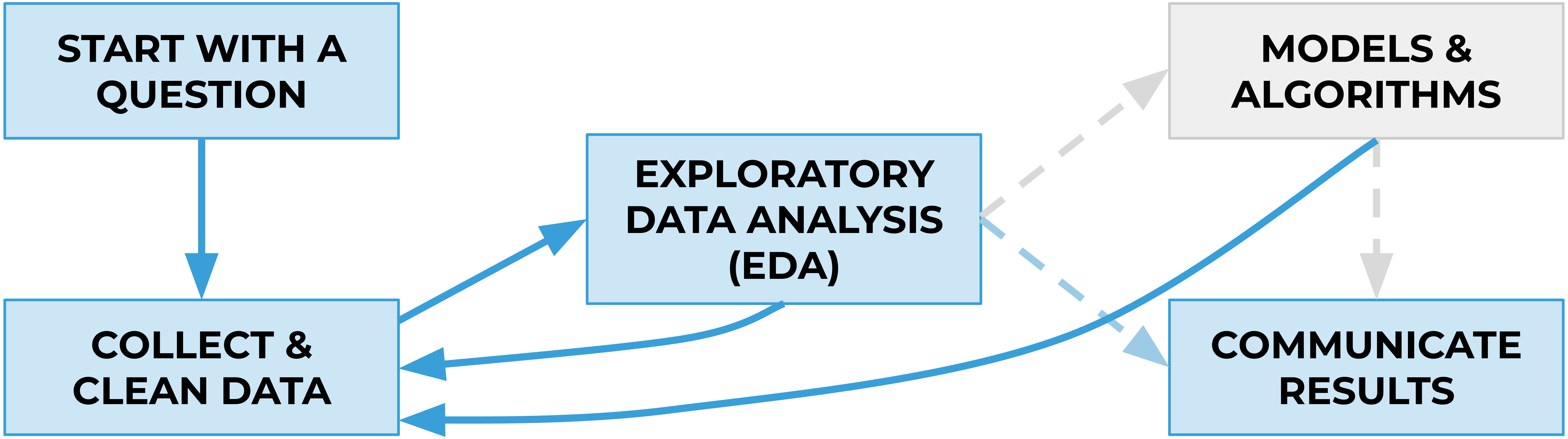
Data Analysis Workflow

What does the data analysis workflow look like?



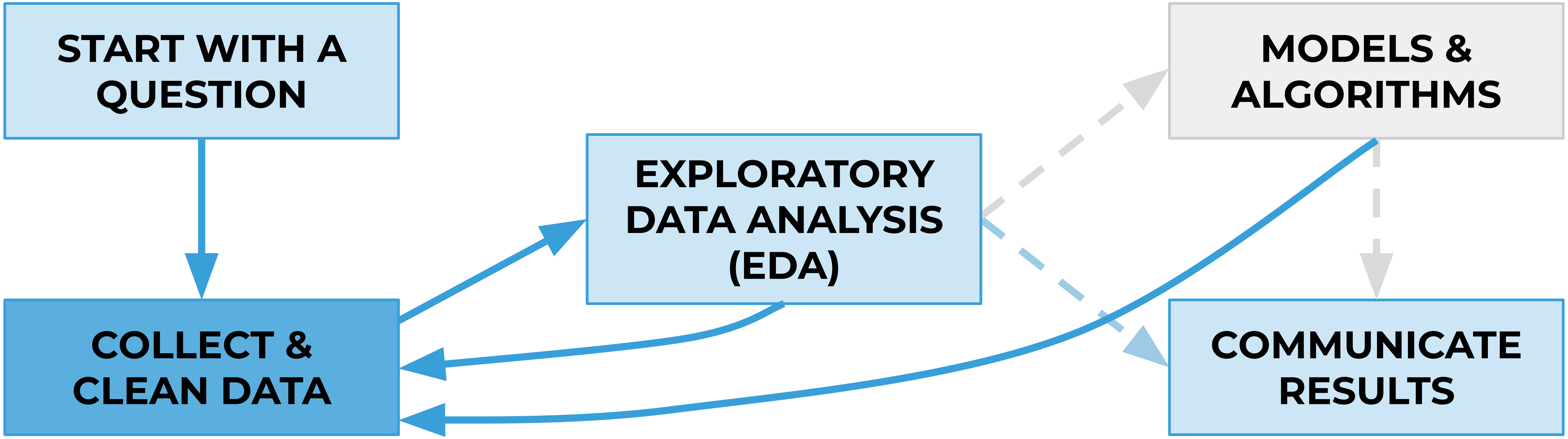
Data Analysis Workflow

What does the data analysis workflow look like?



Data Analysis Workflow

What does the data analysis workflow look like?



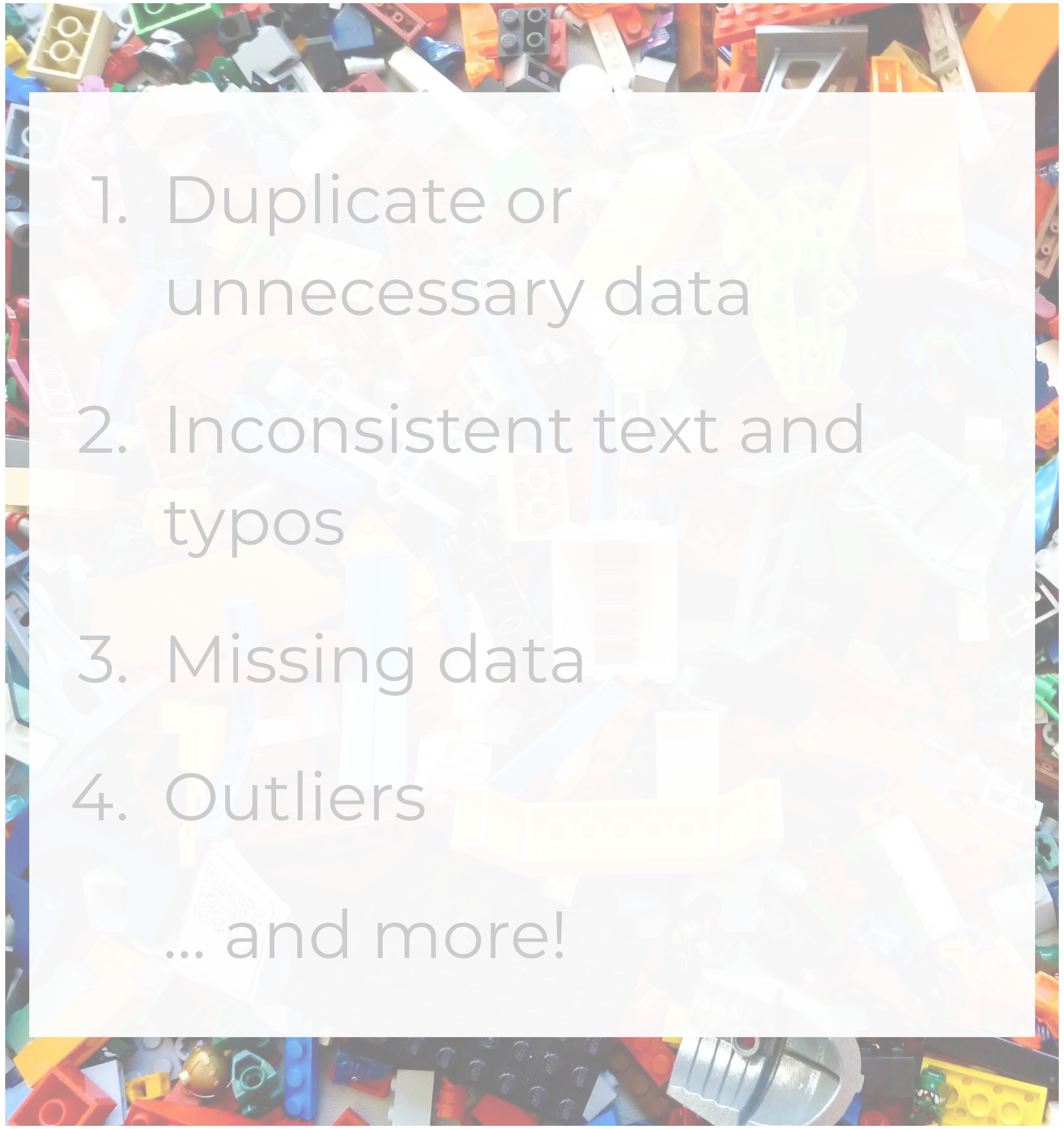
How can data be messy?

How can data be messy?

1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. Missing data
4. Outliers

... and more!

How can data be messy?



Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	Facebook	512,000,000
5	amazon.com	492 million
6	yelp.com	---
7	reddit.com	184,000,000
36	netflix.com	37,000,000

How can data be messy?

1. **Duplicate or unnecessary data**
2. Inconsistent text and typos
3. Missing data
4. Outliers

... and more!

Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	Facebook	512,000,000
5	amazon.com	492 million
6	yelp.com	---
7	reddit.com	184,000,000
36	netflix.com	37,000,000

How can data be messy?

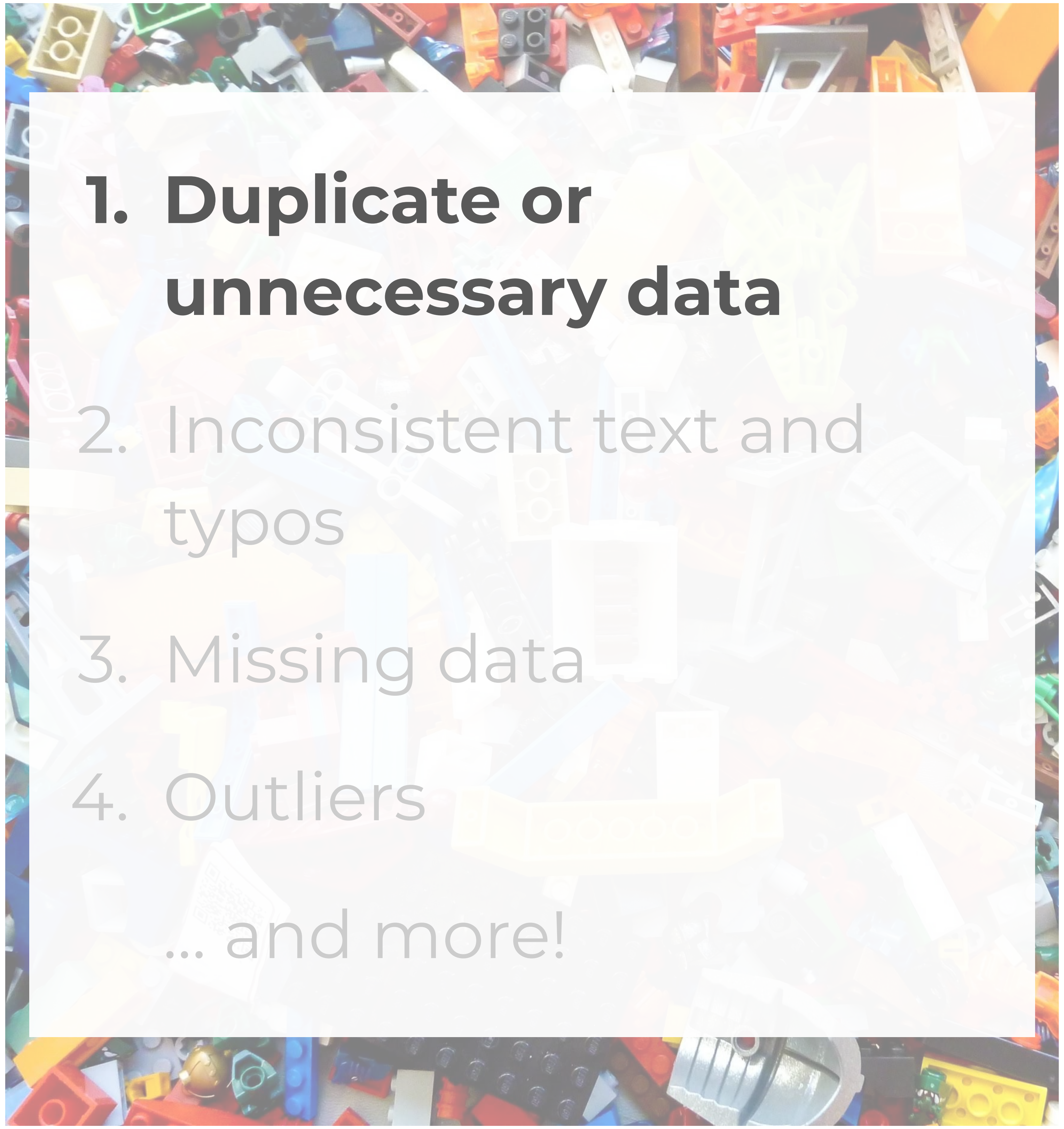
1. **Duplicate or unnecessary data**
2. Inconsistent text and typos
3. Missing data
4. Outliers

... and more!

Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	Facebook	512,000,000
5	amazon.com	492 million
6	yelp.com	---
7	reddit.com	184,000,000
36	netflix.com	37,000,000

How can data be messy?



1. **Duplicate or unnecessary data**
2. Inconsistent text and typos
3. Missing data
4. Outliers

... and more!

- Look for duplicates and **dig into why** there are multiple values
- **Filter data down** as appropriate

How can data be messy?

1. Duplicate or unnecessary data
2. **Inconsistent text and typos**
3. Missing data
4. Outliers

... and more!

Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	Facebook	512,000,000
5	amazon.com	492 million
6	yelp.com	---
7	reddit.com	184,000,000

How can data be messy?

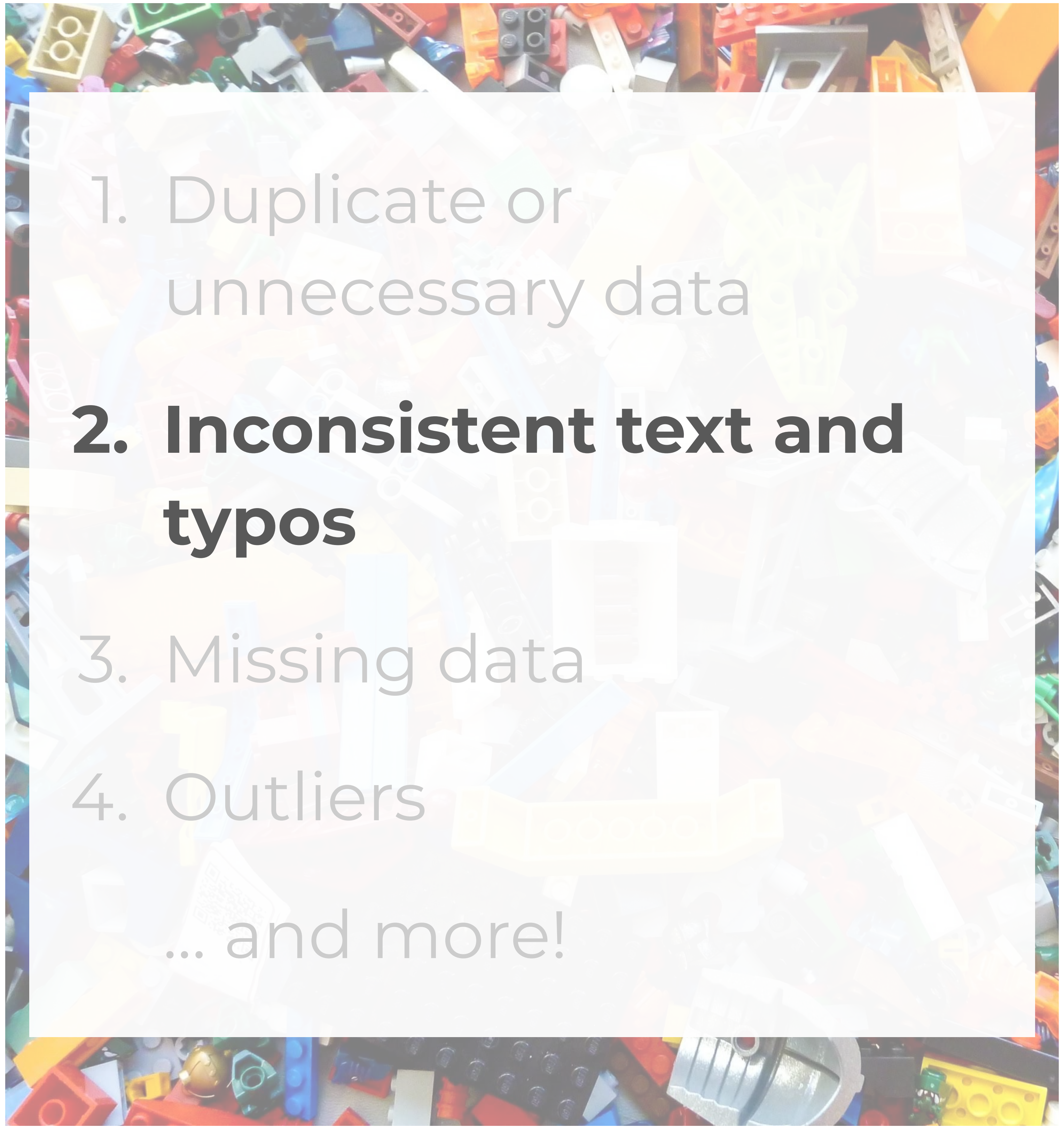
1. Duplicate or unnecessary data
2. **Inconsistent text and typos**
3. Missing data
4. Outliers

... and more!

Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	Facebook	512,000,000
5	amazon.com	492 million
6	yelp.com	---
7	reddit.com	184,000,000

How can data be messy?



1. Duplicate or unnecessary data
2. **Inconsistent text and typos**
3. Missing data
4. Outliers

... and more!

Check **summary statistics** for each column of data.

- Minimum and maximum of numerical values
- Unique values of categoricals

How can data be messy?

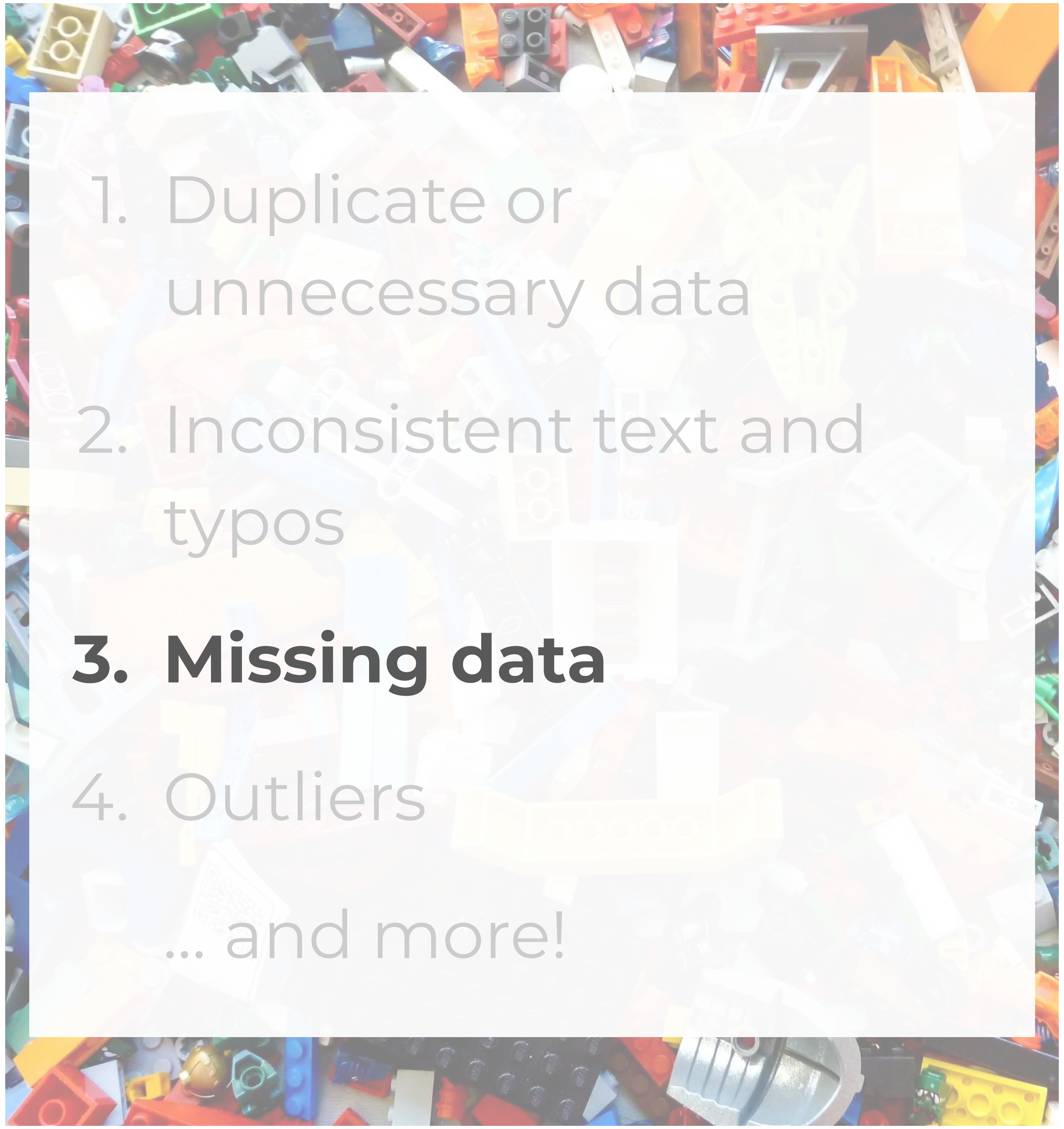
1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. **Missing data**
4. Outliers

... and more!

Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	facebook.com	512,000,000
5	amazon.com	492,000,000
6	yelp.com	---
7	reddit.com	184,000,000

How can data be messy?



1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. **Missing data**
4. Outliers

... and more!

Most Visited US Websites (as of 2020)

rank	website	monthly_traffic
1	youtube.com	1,626,000,000
2	en.wikipedia.org	1,032,000,000
3	twitter.com	536,000,000
4	facebook.com	512,000,000
5	amazon.com	492,000,000
6	yelp.com	---
7	reddit.com	184,000,000

How can data be messy?

1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. Missing data
4. **Outliers**

... and more!

Outliers

- Are **distant** from other observations
- Do not accurately represent real world
- Can **significantly impact** analysis

How can data be messy?

1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. Missing data
4. **Outliers**

... and more!

How to **find** outliers:

- Plots
- Statistics

How can data be messy?

1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. Missing data
4. **Outliers**

... and more!

How to **find** outliers:

- Plots
- Statistics

How to **deal with** outliers:

- Remove them
- Assign mean or median value
- Predict value with model

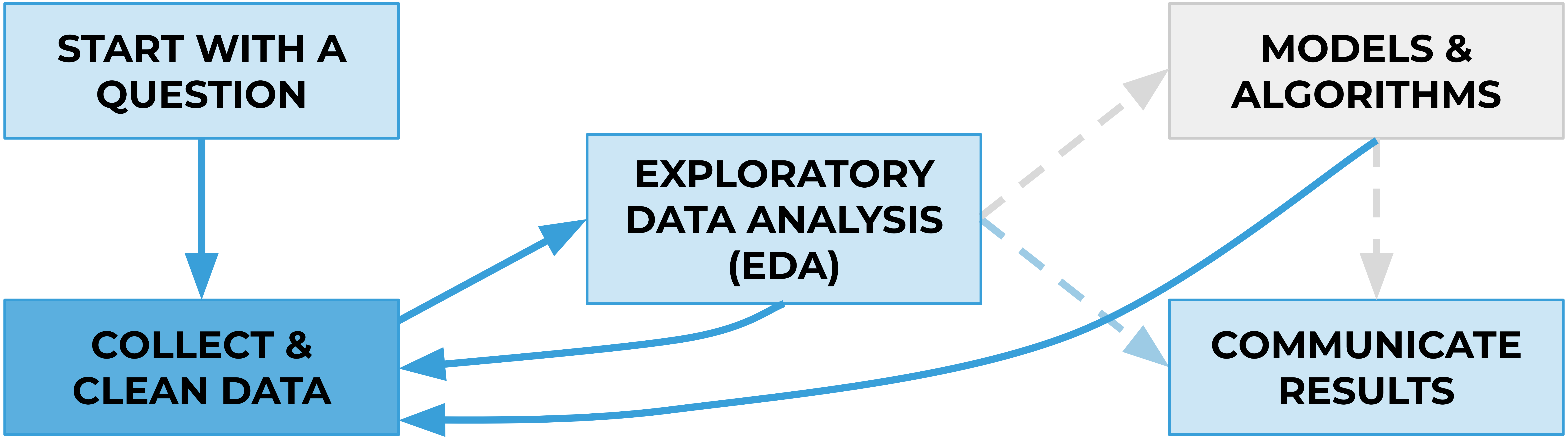
How can data be messy?

1. Duplicate or unnecessary data
2. Inconsistent text and typos
3. Missing data
4. Outliers

... and more!

Data Analysis Workflow

What does the data analysis workflow look like?



Handling Missing Values

Missing Values

- Unfortunately very common
- Occur for many reasons
- Detect with pandas
- Several ways to handle missings

Detecting Missing Values

Load in data about coffee

```
import pandas as pd
df = pd.read_csv("coffee.csv")
df
```

	price_lb	shipping	kind
Bold Bean	8.33	3.00	arabica
Morning Jumpstart	NaN	NaN	arabica
Barista's Dream	15.29	1.99	robusta
Eastern Roast	27.88	5.49	liberica
Cup o' Joe	9.99	NaN	arabica
Wide Awake	13.99	4.00	robusta
Daily Grind	9.15	2.50	arabica

Detecting Missing Values

Dataset contains several missings

```
import pandas as pd
df = pd.read_csv("coffee.csv")
df
```

	price_lb	shipping	kind
Bold Bean	8.33	3.00	arabica
Morning Jumpstart	NaN	NaN	arabica
Barista's Dream	15.29	1.99	robusta
Eastern Roast	27.88	5.49	liberica
Cup o' Joe	9.99	NaN	arabica
Wide Awake	13.99	4.00	robusta
Daily Grind	9.15	2.50	arabica

Detecting Missing Values

Quickly check missings with `.info()`

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, Bold Bean to Daily Grind
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   price_lb   6 non-null      float64
 1   shipping   5 non-null      float64
 2   kind       7 non-null      object
dtypes: float64(2), object(1)
memory usage: 544.0+ bytes
```

Detecting Missing Values

Use `.isna()` for elementwise True/False values

```
df.isna()
```

	price_lb	shipping	kind
Bold Bean	False	False	False
Morning Jumpstart	True	True	False
Barista's Dream	False	False	False
Eastern Roast	False	False	False
Cup o' Joe	False	True	False
Wide Awake	False	False	False
Daily Grind	False	False	False

Detecting Missing Values

Use `.isna()` for elementwise True/False values

```
df.isna()
```

	price_lb	shipping	kind
Bold Bean	False	False	False
Morning Jumpstart	True	True	False
Barista's Dream	False	False	False
Eastern Roast	False	False	False
Cup o' Joe	False	True	False
Wide Awake	False	False	False
Daily Grind	False	False	False

How to Detect Missing Values

Use `.isna()` result as data mask

```
~df.shipping.isna()
```

```
Bold Bean            True
Morning Jumpstart   False
Barista's Dream      True
Eastern Roast        True
Cup o' Joe          False
Wide Awake           True
Daily Grind          True
Name: shipping, dtype: bool
```

How to Detect Missing Values

Use `.isna()` result as data mask

```
~df.shipping.isna()
```

```
Bold Bean            True
Morning Jumpstart   False
Barista's Dream      True
Eastern Roast        True
Cup o' Joe          False
Wide Awake           True
Daily Grind          True
Name: shipping, dtype: bool
```

```
df[~df.shipping.isna()]
```

	price_lb	shipping	kind
Bold Bean	8.33	3.00	arabica
Barista's Dream	15.29	1.99	robusta
Eastern Roast	27.88	5.49	liberica
Wide Awake	13.99	4.00	robusta
Daily Grind	9.15	2.50	arabica

Methods to Handle Missings

1. Drop rows with missing values
2. Fill missing values with a standard value such as zero
3. Impute missings with mean or median

Dropping Missing Values

Use pandas to drop with `.dropna()`

- Drops all rows with any missing by default

```
df.dropna()
```

	price_lb	shipping	kind
Bold Bean	8.33	3.00	arabica
Barista's Dream	15.29	1.99	robusta
Eastern Roast	27.88	5.49	liberica
Wide Awake	13.99	4.00	robusta
Daily Grind	9.15	2.50	arabica

Dropping Missing Values

Use pandas to drop with `.dropna()`

- Drops all rows with any missing by default
- Use `subset` to drop only some missings

```
df.dropna(subset=["price_lb"])
```

	price_lb	shipping	kind
Bold Bean	8.33	3.00	arabica
Barista's Dream	15.29	1.99	robusta
Eastern Roast	27.88	5.49	liberica
Cup o' Joe	9.99	NaN	arabica
Wide Awake	13.99	4.00	robusta
Daily Grind	9.15	2.50	arabica

Filling Missings with a Value

Use pandas `.fillna()` to fill missings

```
df.shipping.fillna(0)
```

Filling Missings with a Value

Use pandas `.fillna()` to fill missings

- **Only fill with reasonable values!**

```
df.shipping.fillna(0)
```

```
Bold Bean             3.00
Morning Jumpstart     0.00
Barista's Dream       1.99
Eastern Roast         5.49
Cup o' Joe            0.00
Wide Awake            4.00
Daily Grind           2.50
Name: shipping, dtype: float64
```

Imputing Missings

Use pandas `.fillna()` to fill missings with mean or median values

```
price_avg = df.price_lb.mean()
price_avg
```

```
14.105
```

Imputing Missings

Use pandas `.fillna()` to fill missings with mean or median values

```
price_avg = df.price_lb.mean()
df.price_lb.fillna(price_avg)
```

```
Bold Bean              8.330
Morning Jumpstart     14.105
Barista's Dream       15.290
Eastern Roast         27.880
Cup o' Joe             9.990
Wide Awake            13.990
Daily Grind            9.150
Name: price_lb, dtype: float64
```

Detecting Missing Values

`.info()`

- Count of non-null values for each column

`.isna()`

- Boolean True/False for each element
- Can be used as data mask

Methods to Handle Missings

`.dropna()`

1. Drop rows with missing values

`.fillna()`

2. Fill missing values with a standard value such as zero
3. Impute missings with mean or median

4. Use a model to predict missing (advanced)

Managing Columns of Data

METIS®

Rename Data Columns

Load in data about coffee

```
import pandas as pd
df = pd.read_csv("coffee_data.csv")
df
```

	Price per Pound	Shipping Price	Favorite?
Bold Bean	8.33	3.00	Yes
Morning Jumpstart	9.49	0.00	Yes
Barista's Dream	15.29	1.99	Yes
Eastern Roast	27.88	5.49	Yes
Cup o' Joe	9.99	0.00	Yes
Wide Awake	13.99	4.00	Yes
Daily Grind	9.15	2.50	Yes

Rename Data Columns

None of these column names are valid Python variables

- Rename to make analysis easier

```
df.columns
```

```
Index(['Price per Pound', 'Shipping
Price', 'Favorite?'], dtype='object')
```

Rename Data Columns

Pass a dictionary to the `columns` argument of pandas `.rename()`

```
df.rename(columns={
   'Price per Pound': 'price_lb',
   'Shipping Price': 'shipping',
   'Favorite?': 'favorite'
   }, inplace=True)
```

Rename Data Columns

Pass a dictionary to the `columns` argument of pandas `.rename()`

```
df.rename(columns={
   'Price per Pound': 'price_lb',
   'Shipping Price': 'shipping',
   'Favorite?': 'favorite'
   }, inplace=True)
```

```
df
```

	price_lb	shipping	favorite
Bold Bean	8.33	3.00	Yes
Morning Jumpstart	9.49	0.00	Yes
Barista's Dream	15.29	1.99	Yes
Eastern Roast	27.88	5.49	Yes
Cup o' Joe	9.99	0.00	Yes
Wide Awake	13.99	4.00	Yes
Daily Grind	9.15	2.50	Yes

What is the average shipping?

Why does using `.mean()` on shipping column causes error?

```
df.shipping.mean()
```

```
TypeError: Could not convert
3.000.001.995.490.004.002.50 to numeric
```

What is the average shipping?

Why does using `.mean()` on shipping column causes error?

```
df.shipping.mean()
```

```
TypeError: Could not convert
3.000.001.995.490.004.002.50 to numeric
```

```
df.head(3)
```

	price_lb	shipping	favorite
Bold Bean	8.33	3.00	Yes
Morning Jumpstart	9.49	0.00	Yes
Barista's Dream	15.29	1.99	Yes

What is the average shipping?

Checking `.dtypes` shows the shipping column contains strings

```
df.dtypes
```

```
price_lb      float64
shipping       object
favorite       object
dtype: object
```

Updating a Column Datatype

Convert a column's datatype with the `.astype()` method

```
df['shipping'] = df.shipping.astype('float')
```

Updating a Column Datatype

Convert a column's datatype with the `.astype()` method

```
df['shipping'] = df.shipping.astype('float')
df.dtypes
```

```
price_lb      float64
shipping      float64
favorite       object
dtype: object
```

Updating a Column Datatype

Convert a column's datatype with the `.astype()` method

```
df['shipping'] = df.shipping.astype('float')
df.dtypes
```

```
price_lb      float64
shipping      float64
favorite       object
dtype: object
```

```
df.shipping.mean()
```

```
2.4257142857142857
```

Dropping Columns

The favorite column contains the same value for every row

```
df.favorite.value_counts()
```

```
Yes      7
Name: favorite, dtype: int64
```

Dropping Columns

Drop unnecessary columns with pandas `.drop()`

- `axis=0` refers to the row dimension
- `axis=1` refers to column dimension

```
df.drop('favorite', axis=1)
```

Dropping Columns

Drop unnecessary columns with pandas `.drop()`

- `axis=0` refers to the row dimension
- `axis=1` refers to column dimension

```
df.drop('favorite', axis=1)
```

	price_lb	shipping
Bold Bean	8.33	3.00
Morning Jumpstart	9.49	0.00
Barista's Dream	15.29	1.99
Eastern Roast	27.88	5.49
Cup o' Joe	9.99	0.00
Wide Awake	13.99	4.00
Daily Grind	9.15	2.50

Managing Columns of Data

- Rename columns by passing an update dictionary into `.rename()`
- Convert column's datatype with `.astype()`
- Use `.drop()` and `axis=1` to drop a column from the dataframe

Cleaning String Data

METIS®

Analyzing Text Data

Text data is notoriously messy.

- Inconsistent text
- Typos
- Extra whitespace
- Extra characters in numerical values (e.g. commas, dollar signs)

Analyzing Text Data

Load in data about US cities

```
import pandas as pd
df = pd.read_csv("cities.csv")
df
```

	city	state	population
0	Chicago	IL	2,706,000
1	Los Angeles	ca	3,990,000
2	Omaha	NE	468,300
3	Dallas	TX	1,345,000
4	Philadelphia	Pa	1,584,000
5	Los Alamos	NM	12,373

Convert Column to Upper- or Lowercase

- Inconsistencies in state column

```
df
```

	city	state	population
0	Chicago	IL	2,706,000
1	Los Angeles	ca	3,990,000
2	Omaha	NE	468,300
3	Dallas	TX	1,345,000
4	Philadelphia	Pa	1,584,000
5	Los Alamos	NM	12,373

Convert Column to Upper- or Lowercase

- Inconsistencies in state column
- Convert column to uppercase
- Reference string methods, `.str`

```
df.state = df.state.str.upper()
df.state
```

```
0     IL
1     CA
2     NE
3     TX
4     PA
5     NM
Name: state, dtype: object
```

Remove Specific Characters

- Commas in population column

```
df
```

	city	state	population
0	Chicago	IL	2,706,000
1	Los Angeles	CA	3,990,000
2	Omaha	NE	468,300
3	Dallas	TX	1,345,000
4	Philadelphia	PA	1,584,000
5	Los Alamos	NM	12,373

Remove Specific Characters

- Commas in population column
- Remove by replacing commas with the empty string

```
pop = df.population.str.replace(',', '')
pop
```

```
0    2706000
1    3990000
2     468300
3    1345000
4    1584000
5      12373
Name: population, dtype: object
```

Remove Specific Characters

- Commas in population column
- Remove by replacing commas with the empty string

```
pop = df.population.str.replace(',', '')
pop
```

```
0    2706000
1    3990000
2     468300
3    1345000
4    1584000
5      12373
Name: population, dtype: object
```

```
pop.astype('int')
```

```
0    2706000
1    3990000
2     468300
3    1345000
4    1584000
5      12373
Name: population, dtype: int64
```

Analyzing Text Data

```
df
```

	city	state	population
0	Chicago	IL	2706000
1	Los Angeles	CA	3990000
2	Omaha	NE	468300
3	Dallas	TX	1345000

Analyzing Text Data

```
df
```

	city	state	population
0	Chicago	IL	2706000
1	Los Angeles	CA	3990000
2	Omaha	NE	468300
3	Dallas	TX	1345000

```
df.city.unique()
```

```
array(['Chicago ', 'Los Angeles ',
  'Omaha ', 'Dallas ', 'Philadelphia ',
  'Los Alamos '], dtype=object)
```

Removing Whitespace

Strip whitespace from front or end of string data

- Whitespace includes spaces, tabs, new line characters, etc.

```
city =  df.city.str.strip()
```

Removing Whitespace

Strip whitespace from front or end of string data

- Whitespace includes spaces, tabs, new line characters, etc.

```
city =  df.city.str.strip()
city.unique()
```

```
array(['Chicago', 'Los Angeles',
  'Omaha', 'Dallas', 'Philadelphia',
  'Los Alamos'], dtype=object)
```

Checking for Substrings

Which cities contain “Los”?

- Check elementwise with `.contains()`

```
df.city.str.contains('Los')
```

```
0    False
1     True
2    False
3    False
4    False
5     True
Name: city, dtype: bool
```

Checking for Substrings

Which cities contain “Los”?

- Check elementwise with `.contains()`
- Use result as data mask

```
df.city.str.contains('Los')
```

```
0    False
1     True
2    False
3    False
4    False
5     True
Name: city, dtype: bool
```

```
df[df.city.str.contains('Los')]
```

	city	state	population
1	Los Angeles	CA	3990000
5	Los Alamos	NM	12373

Analyzing Text Data

Text data is notoriously messy.

Inconsistent text or typos

- `.str.upper()`, `.str.lower()`
- `.str.replace()`

Extra whitespace

- `.str.strip()`

Characters in numerical values

- `.str.replace()`

Searching for substrings

- `.str.contains()`

Case Study #1: Exploring New Data

METIS®

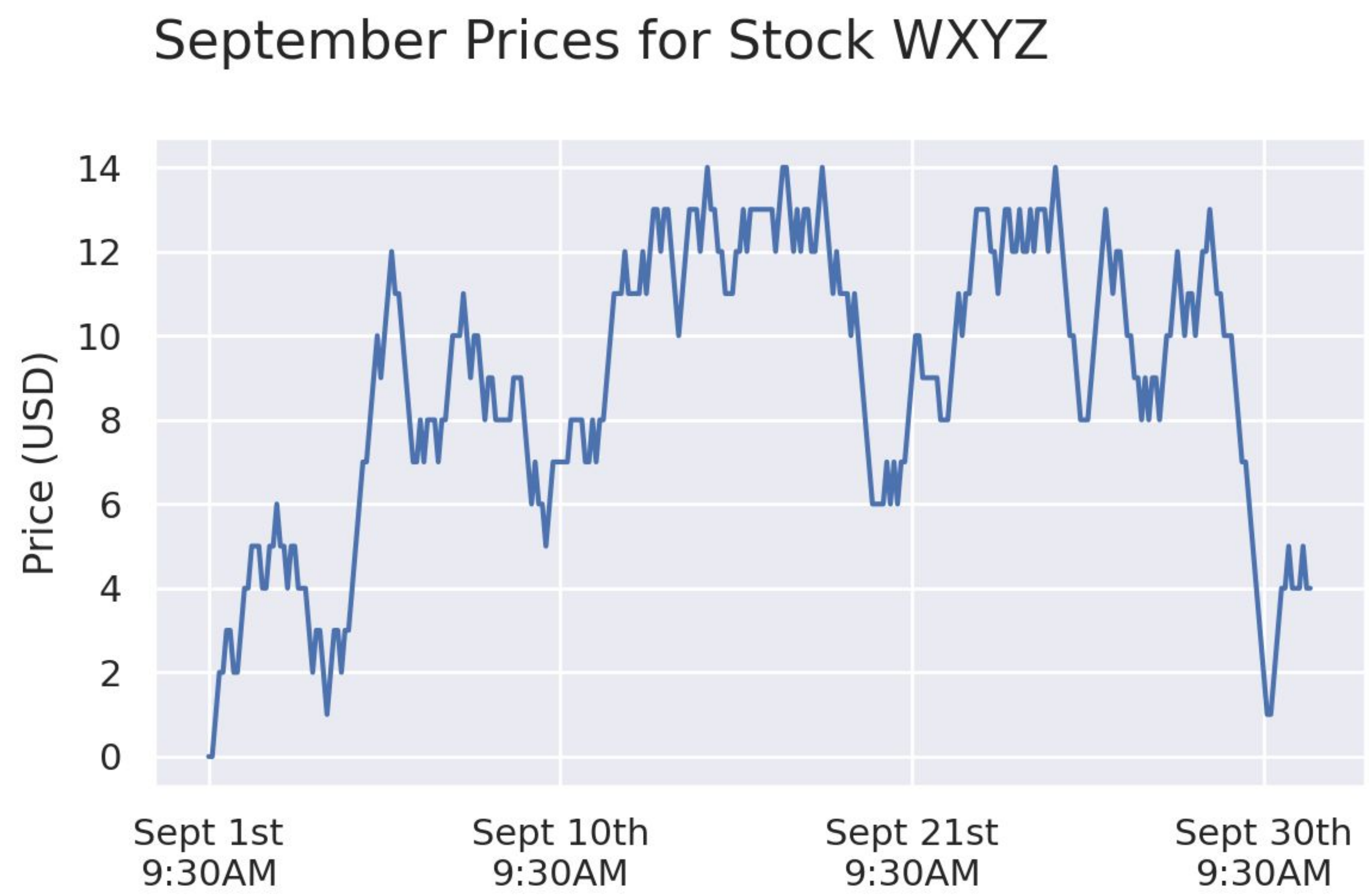
What should we do when exploring new data?

Ask many questions and be skeptical.

- How do these data help answer the project question?
- What kind of data is given in each column?
- Do the data contain missing values?
- What steps are necessary to make these data ready for analysis?

Stock Prices Case Study

Which Monday time saw the highest stock price in September?



Stock Prices Case Study

What kind of data do we have?

```
import pandas as pd
df = pd.read_csv("wxyz.csv")
df.head()
```

	day	time	price
0	9/1/2020	9:30 AM	$0.00
1	9/1/2020	10:00 AM	$0.00
2	9/1/2020	10:30 AM	$1.00
3	9/1/2020	11:00 AM	$2.00
4	9/1/2020	11:30 AM	$2.00

Stock Prices Case Study

What kind of data do we have?

```
df.shape
```

```
(308, 3)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308 entries, 0 to 307
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   day     308 non-null    object
 1   time    308 non-null    object
 2   price   308 non-null    object
dtypes: object(3)
memory usage: 7.3+ KB
```

Stock Prices Case Study

What kind of data do we have?

```
df.shape
```

```
(308, 3)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308 entries, 0 to 307
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   day     308 non-null    object
 1   time    308 non-null    object
 2   price   308 non-null    object
dtypes: object(3)
memory usage: 7.3+ KB
```

All columns contain strings

Convert Strings to Numerics

Convert the price column to numerical values

```
df.head()
```

	day	time	price
0	9/1/2020	9:30 AM	$0.00
1	9/1/2020	10:00 AM	$0.00
2	9/1/2020	10:30 AM	$1.00
3	9/1/2020	11:00 AM	$2.00
4	9/1/2020	11:30 AM	$2.00

Convert Strings to Numerics

Convert the price column to numerical values

- Remove dollar signs

```
df.price = df.price.str.replace('$', '')
df.head()
```

	day	time	price
0	9/1/2020	9:30 AM	0.00
1	9/1/2020	10:00 AM	0.00
2	9/1/2020	10:30 AM	1.00
3	9/1/2020	11:00 AM	2.00
4	9/1/2020	11:30 AM	2.00

Convert Strings to Numerics

Convert the price column to numerical values

- Remove dollar signs
- Convert from strings to floats

```
df.price = df.price.astype('float')
df.dtypes
```

```
day          object
time         object
price       float64
dtype: object
```

Stock Prices Case Study

Which Monday time saw the highest stock price in September?

```
df.sample(5)
```

	day	time	price
143	9/15/2020	11:00 AM	12.0
157	9/16/2020	11:00 AM	13.0
243	9/24/2020	12:00 PM	8.0
276	9/28/2020	2:30 PM	11.0
296	9/30/2020	10:30 AM	1.0

Create Datetime Column

- Combine day and time columns

```
df['date_time'] = df.day + ' ' + df.time
df.head()
```

	day	time	price	date_time
0	9/1/2020	9:30 AM	0.0	9/1/2020 9:30 AM
1	9/1/2020	10:00 AM	0.0	9/1/2020 10:00 AM
2	9/1/2020	10:30 AM	1.0	9/1/2020 10:30 AM
3	9/1/2020	11:00 AM	2.0	9/1/2020 11:00 AM
4	9/1/2020	11:30 AM	2.0	9/1/2020 11:30 AM

Create Datetime Column

- Combine day and time columns
- Convert to datetime

```
df['date_time'] = df.day + ' ' + df.time
df.head()
```

	day	time	price	date_time
0	9/1/2020	9:30 AM	0.0	9/1/2020 9:30 AM
1	9/1/2020	10:00 AM	0.0	9/1/2020 10:00 AM
2	9/1/2020	10:30 AM	1.0	9/1/2020 10:30 AM
3	9/1/2020	11:00 AM	2.0	9/1/2020 11:00 AM
4	9/1/2020	11:30 AM	2.0	9/1/2020 11:30 AM

```
df['date_time'] = pd.to_datetime(df.date_time)
df.dtypes
```

```
day                  object
time                 object
price               float64
date_time    datetime64[ns]
dtype: object
```

Create Day of Week Column

Use `weekday` property of datetime

```
df['day_of_week'] = df.date_time.dt.weekday
df.sample(5)
```

	day	time	price	date_time	day_of_week
157	9/16/2020	11:00 AM	13.0	2020-09-16 11:00:00	2
296	9/30/2020	10:30 AM	1.0	2020-09-30 10:30:00	2
143	9/15/2020	11:00 AM	12.0	2020-09-15 11:00:00	1
243	9/24/2020	12:00 PM	8.0	2020-09-24 12:00:00	3
276	9/28/2020	2:30 PM	11.0	2020-09-28 14:30:00	0

Stock Prices Case Study

Which Monday time saw the highest stock price in September?

- Select Mondays

```
mondays = df[df.day_of_week == 0]
mondays.sample(5)
```

	day	time	price	date_time	day_of_week
58	9/7/2020	10:30 AM	7.0	2020-09-07 10:30:00	0
129	9/14/2020	11:00 AM	12.0	2020-09-14 11:00:00	0
130	9/14/2020	11:30 AM	11.0	2020-09-14 11:30:00	0
199	9/21/2020	11:00 AM	9.0	2020-09-21 11:00:00	0
279	9/28/2020	4:00 PM	13.0	2020-09-28 16:00:00	0

Stock Prices Case Study

Which Monday time saw the highest stock price in September?

- Select Mondays
- Sort to find the maximum price

```
(mondays[['date_time', 'price']]
  .sort_values('price', ascending=False)
  .head(3))
```

	date_time	price
139	2020-09-14 16:00:00	14.0
279	2020-09-28 16:00:00	13.0
138	2020-09-14 15:30:00	13.0

What should we do when exploring new data?

`.head(), .info(), .dtypes, .shape`

- What kind of data is given in each column?
- Do the data contain missing values?

`.replace(), .to_datetime()`

- What steps are necessary to make data ready for analysis?

- How do these data help answer the project question?

Case Study #2:
Diagnosing Errors

METIS®

How can we diagnose errors?

Data inconsistencies may cause errors when operating on columns.

Build custom functions to:

- Include print statements
- Add conditional statements
- Use exceptions

Circus Performers Case Study

Which type of performer is the most experienced on average?

```python
import pandas as pd
df = pd.read_csv("circus.csv")
df.head()
```

	email	performances
0	jamie50@liontamer.org	16
1	mya62@liontamer.org	3
2	bertie74@clown.inc	6
3	jonathan@clown.inc	24
4	kasper112@juggle.xyz	28

Circus Performers Case Study

Which type of performer is the most experienced on average?

```
df.shape
```

```
(50, 2)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   email         50 non-null     object
 1   performances  50 non-null     int64
dtypes: int64(1), object(1)
memory usage: 928.0+ bytes
```

Manipulating Email Data

Create new columns for domain

```
str.split("jamie50@liontamer.org", "@")
```

```
['jamie50', 'liontamer.org']
```

```
str.split("jamie50@liontamer.org", "@")[1]
```

```
'liontamer.org’
```

Manipulating Email Data

Create new columns for domain

```
df.loc[:2].email.map(
   lambda x: str.split(x, "@")[1]
)
```

```
0     liontamer.org
1     liontamer.org
2         clown.inc
Name: email, dtype: object
```

Manipulating Email Data

Create new columns for domain

```
df.email.map(
   lambda x: str.split(x, "@")[1]
)
```

```
IndexError: list index out of range
```

Creating Custom Function

Create a custom Python function to diagnose error

```
def check_for_at_symbol(email):
    if '@' not in email:
        print(email)
    return ('@' in email)

at_symbol_test = df.email.map(check_for_at_symbol)
```

```
jonathan104ringmaster.net
```

Creating Custom Function

Create a custom Python function to create domain column

```
def get_domain(email):
   if '@' not in email:
      return None
   return str.split(email, '@')[1]

df['domain'] = df.email.map(get_domain)
```

Creating Custom Function

Create a custom Python function to create domain column

```
def get_domain(email):
    if '@' not in email:
        return None
    return str.split(email, '@')[1]

df['domain'] = df.email.map(get_domain)
df.loc[13:16]
```

	email	performances	domain
13	nicoletta71@liontamer.org	23	liontamer.org
14	mya_6@ringmaster.net	5	ringmaster.net
15	jonathan104ringmaster.net	14	None
16	natasha.114@ringmaster.net	12	ringmaster.net

Using Python Exceptions

Create domain column by catching errors with exception

```
def get_domain_exception(email):
    try:
        return str.split(email, '@')[1]
    except IndexError:
        print(email)
        return None

df['domain'] = df.email.map(get_domain_exception)
```

```
jonathan104ringmaster.net
```

Circus Performers Case Study

Which type of performer is the most experienced on average?

```
df.head(3)
```

	email	performances	domain
0	jamie50@liontamer.org	16	liontamer.org
1	mya62@liontamer.org	3	liontamer.org
2	bertie74@clown.inc	6	clown.inc

```
df.shape
```

```
(50, 2)
```

```
df.email.nunique()
```

```
40
```

Remove Duplicate Rows

Use pandas to remove duplicate rows with `.drop_duplicates()`

- Entire row must be exact match

```
df.drop_duplicates().shape()
```

```
(40, 3)
```

Remove Duplicate Rows

Use pandas to remove duplicate rows with `.drop_duplicates()`

- Entire row must be exact match
- Use `subset` argument with column name or list to match specific columns

```
df.drop_duplicates().shape()
```

```
(40, 3)
```

```
df.drop_duplicates(subset='email').shape()
```

```
(40, 3)
```

Circus Performers Case Study

Which type of performer is the most experienced on average?

```
df.drop_duplicates(inplace=True)
(df.groupby('domain')
   .performances
   .mean()
   .sort_values(ascending=False))
```

Circus Performers Case Study

Which type of performer is the most experienced on average?

```
df.drop_duplicates(inplace=True)
(df.groupby('domain')
   .performances
   .mean()
   .sort_values(ascending=False))
```

```
domain
trapeze          23.0
clown.inc        17.4
juggle.xyz       17.2
ringmaster.net   15.0
acrobatics.com   12.5
liontamer.org    10.4
Name: performances, dtype: float64
```

Case Study #3:

Comparing Against Group Statistics

METIS®

Penguins Case Study

Standardize the penguin masses by species and sort by standard mass ascending.

```python
import seaborn as sns
df = sns.load_dataset("penguins")
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

Penguins Case Study

Standardize the penguin masses by species and sort by standard mass ascending.

$$\frac{x - \mu_{spec}}{\sigma_{spec}}$$

```python
import seaborn as sns
df = sns.load_dataset("penguins")
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

Penguins Case Study

What kind of data do we have?

```
df.shape
```

```
(344, 7)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

Handling Missing Values

What kind of missings do we have?

```
df[df.bill_length_mm.isna()]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN

```
df.dropna(subset=["bill_length_mm"], inplace=True)
```

```
df.shape
```

```
(342, 7)
```

Handling Missing Values

What kind of missings do we have?

```
df[df.sex.isna()]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN
10	Adelie	Torgersen	37.8	17.1	186.0	3300.0	NaN
11	Adelie	Torgersen	37.8	17.3	180.0	3700.0	NaN
47	Adelie	Dream	37.5	18.9	179.0	2975.0	NaN
246	Gentoo	Biscoe	44.5	14.3	216.0	4100.0	NaN
286	Gentoo	Biscoe	46.2	14.4	214.0	4650.0	NaN
324	Gentoo	Biscoe	47.3	13.8	216.0	4725.0	NaN
336	Gentoo	Biscoe	44.5	15.7	217.0	4875.0	NaN

Penguins Case Study

Standardize the penguin masses by species and sort by standard mass ascending.

$$\boxed{\frac{x - \mu_{spec}}{\sigma_{spec}}}$$

Pandas Transform

Use transform to produce group aggregates for each row

```
df.groupby("species").body_mass_g.mean()
```

```
species
Adelie       3700.662252
Chinstrap    3733.088235
Gentoo       5076.016260
Name: body_mass_g, dtype: float64
```

Pandas Transform

Use transform to produce group aggregates for each row

```
df["mass_species_mean"] = (df.groupby("species").body_mass_g
                                   .transform(lambda x: x.mean()))
```

Pandas Transform

Use transform to produce group aggregates for each row

```
df["mass_species_mean"] = (df.groupby("species").body_mass_g
                                  .transform(lambda x: x.mean()))
df[["species", "body_mass_g", "mass_species_mean"]].sample(5)
```

	species	body_mass_g	mass_species_mean
45	Adelie	4600.0	3700.662252
199	Chinstrap	4300.0	3733.088235
320	Gentoo	4850.0	5076.016260
68	Adelie	3050.0	3700.662252
332	Gentoo	4650.0	5076.016260

Standard Penguin Mass

Standardize the penguin masses by species.

$$\frac{x - \mu_{spec}}{\sigma_{spec}}$$

```
df["mass_standard"] = (df.groupby("species").body_mass_g
                          .transform(lambda x: (x - x.mean())/x.std()))
```

Standard Penguin Mass

Standardize the penguin masses by species.

$$\frac{x - \mu_{spec}}{\sigma_{spec}}$$

```
df["mass_standard"] = (df.groupby("species").body_mass_g
                                    .transform(lambda x: (x - x.mean())/x.std()))
df[["species", "body_mass_g", "mass_species_mean", "mass_standard"]].sample(5)
```

	species	body_mass_g	mass_species_mean	mass_standard
45	Adelie	4600.0	3700.662252	1.961195
199	Chinstrap	4300.0	3733.088235	1.475046
320	Gentoo	4850.0	5076.016260	-0.448342
68	Adelie	3050.0	3700.662252	-1.418906
332	Gentoo	4650.0	5076.016260	-0.845075

Penguins Case Study

Standardize the penguin masses by species and sort by standard mass ascending.

```
df.sort_values("mass_standard").head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	mass_species_mean	mass_standard
190	Chinstrap	Dream	46.9	16.6	192.0	2700.0	Female	3733.088235	-2.687988
260	Gentoo	Biscoe	42.7	13.7	208.0	3950.0	Female	5076.016260	-2.233644
174	Chinstrap	Dream	43.2	16.6	187.0	2900.0	Female	3733.088235	-2.167609
246	Gentoo	Biscoe	44.5	14.3	216.0	4100.0	NaN	5076.016260	-1.936094
58	Adelie	Biscoe	36.5	16.6	181.0	2850.0	Female	3700.662252	-1.855048