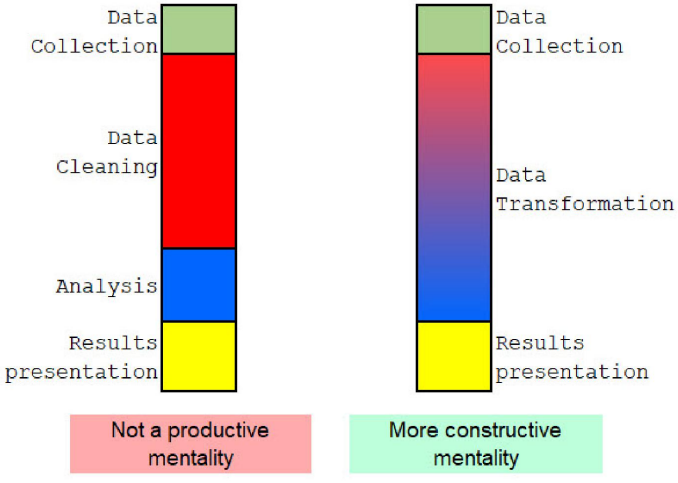
Handling Missing Data & `missingno` Library

Disclaimer: These notes and examples are an adaptation from the references listed at the end. They are compiled to fit the scope of this specific course.

Introduction

Nobody likes *dirty* data-sets! Nobody likes missing values either! We all love to get a clean data set without missing, inaccurate values and start doing *data analysis*, running *regressions*, and *plotting pretty plots*, *training machine learning models*, ...

But unfortunately, real-world data sets are dirty and have missing values! Any data analysis based on low-quality data-set will lead to misleading results. Transforming a low-quality data set into a more reliable, clean data-set is a significant part of data analysis. Data cleaning is an integral part of data analysis, and separating data cleaning from data analysis is simply short-sighted. According to IBM Data Analytics (https://www.ibm.com/cloud/blog) you can expect to spend up to `80%` of your time cleaning data. I hope that the following image changes your mentality regarding data cleaning:



Breakdown of phases in a data analysis project

Your data-set may have missing values for many reasons. Just a few of those reasons are:

- There was a programming error in some earlier part of your code.
- Survey participants decided not to fill out parts of a survey.
- The data entry person made a mistake in entering the data.
- Data got lost in transferring between databases.

This handout will give you an overview of the `missingno` library for visualizing missing data and `pandas` capabilities in handling missing values. As usual, this is meant to jump-start your understanding of `pandas` capabilities, and it is not meant to be comprehensive.

For illustration, we will work with `dirty_house_data.csv` file. Let's define

```
import numpy as np
import pandas as pd
```

```
pd.options.display.precision = 2
dirty_df = pd.read_csv('dirty_house_data.csv')
dirty_df
```

yields

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price
0	Harbor	3/9/2021	Milo	NaN	2.0	1.0	126.0	NaN	2001.0	NaN
1	Harbor	3/12/2021	Lola	30671.0	2.0	1.0	202.0	NaN	NaN	1.48e+06
2	Harbor	3/9/2021	Chubby	32061.0	2.0	1.0	120.0	NaN	1995.0	1.28e+06
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Camden	8/10/2020	Milo	30672.0	1.0	1.0	0.0	NaN	NaN	3.00e+05
5	Camden	8/10/2020	Milo	30673.0	3.0	1.0	220.0	NaN	2005.0	NaN
6	Camden	8/10/2020	Lola	30673.0	NaN	NaN	NaN	NaN	NaN	5.42e+05
7	Harbor	3/9/2020	Chubby	30001.0	2.0	1.0	159.0	NaN	NaN	1.46e+06
8	Harbor	4/3/2020	Lola	NaN	3.0	2.0	211.0	NaN	1890.0	2.85e+06
9	Harbor	4/3/2021	Babushka	NaN	3.0	2.0	128.0	NaN	1890.0	1.85e+06
10	Harbor	4/3/2021	Chubby	32065.0	2.0	1.0	215.0	NaN	1900.0	NaN
11	Harbor	4/3/2021	Lola	32066.0	2.0	1.0	130.0	NaN	1900.0	1.44e+06
12	Harbor	6/8/2020	Babushka	32066.0	2.0	2.0	336.0	NaN	1890.0	NaN
13	Camden	4/2/2021	Milo	30676.0	2.0	1.0	156.0	NaN	2005.0	1.04e+06
14	Camden	4/2/2021	Lola	20000.0	3.0	2.0	0.0	NaN	NaN	NaN
15	Camden	4/3/2021	Milo	30676.0	3.0	2.0	134.0	NaN	1900.0	1.46e+06
16	Camden	4/3/2021	Babushka	30675.0	3.0	2.0	94.0	NaN	NaN	8.50e+05
17	Camden	4/6/2021	Lola	30675.0	3.0	1.0	120.0	NaN	2014.0	1.60e+06
18	Fells	6/8/2020	Chubby	30677.0	3.0	2.0	400.0	NaN	2006.0	NaN
19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
20	Fells	6/8/2020	Lola	30000.0	3.0	2.0	202.0	NaN	2010.0	NaN
21	Fells	7/5/2020	Milo	NaN	2.0	1.0	181.0	NaN	NaN	9.41e+05

Both `pandas` and `numpy` show missing values with `NaN` (`Not a Number`). `dirty_df` has several missing values in different rows and columns. We will first learn how to visualize these missing values and then discuss handling and possibly filling these missing values.

`missingno` library

`missingno` library visualizes the distribution of `NaN` values. Installing `missingno` is straightforward and can be done by running

```
pip install missingno
```

According to its documentation, `missingno` should be imported as `msno` as in

```
import missingno as msno
```

`missingno` has several practical methods, such as:

- `nullity_filter()`: used for filtering a `DataFrame` according to its nullity
- `nullity_sort()`: used for sorting a `DataFrame` according to its nullity
- `matrix()`, `heatmap()`, and `bar()` is the most useful visual tools that `missingno` offers for shedding insights into patterns of `NaN` values in a `DataFrame`.

Let's explore visual tools offered by `missingno`.

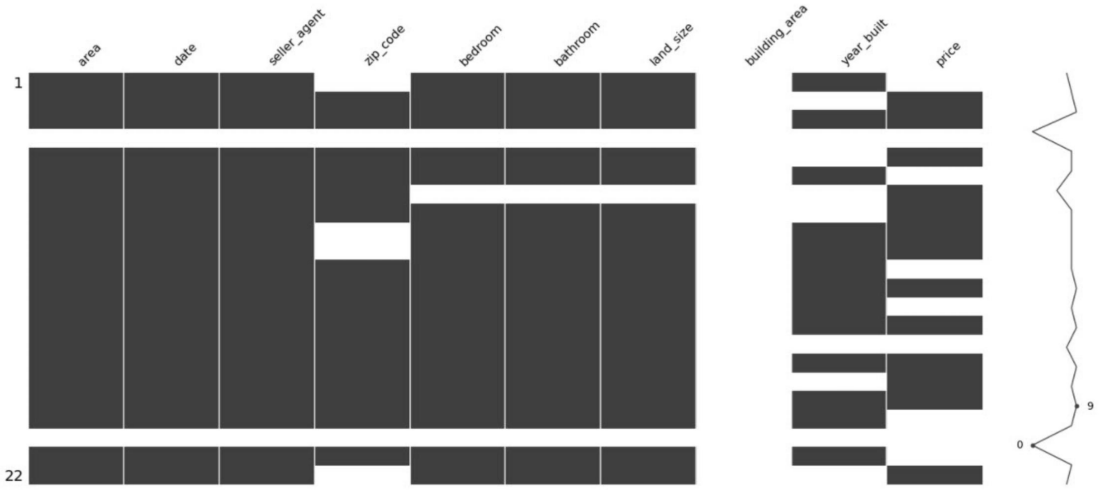
`Visual missing-ness matrix`

```
import missingno as msno
import matplotlib.pyplot as plt

dirty_df = pd.read_csv('dirty_house_data.csv')

msno.matrix(dirty_df)  # establish a matrix
```

yields a matrix that can help us find the patterns of missing-ness in the `DataFrame`

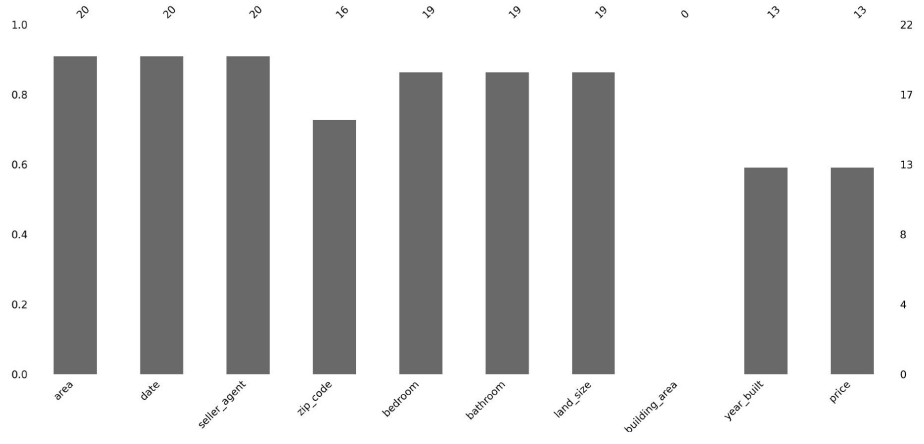


According to this output, the column of `building_area` and two rows are entirely missing. The most complete row in this `DataFrame` has 9 non-`NaN` values.

Bar Chart

```
msno.bar(dirty_df)  # establish a bar chart
```

The Bar chart provides insights into the number of missing values in each column.



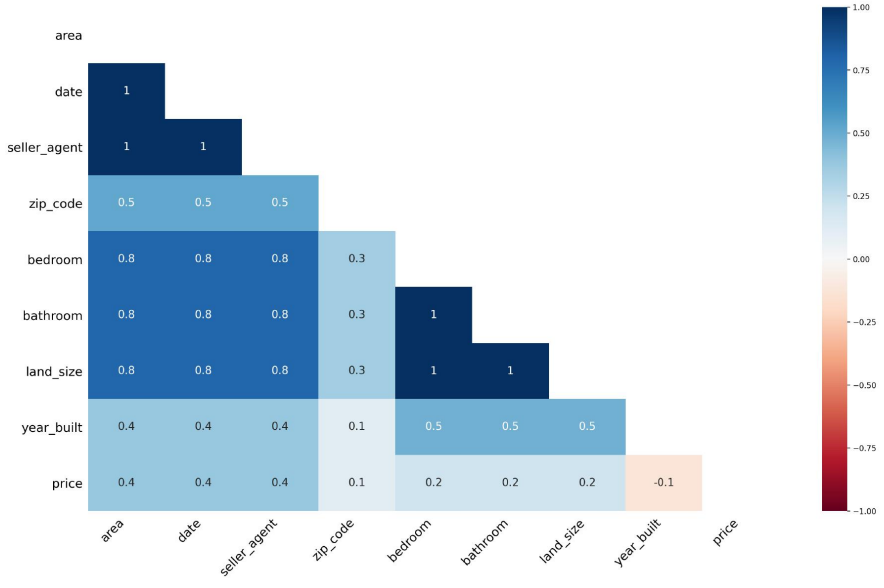
According to this output, besides `building_area` that is completely missing, `year_built` and `price` followed by `zip_code` are the least complete columns in this `DataFrame`.

heatmap

`heatmap` visualizes the correlation of missing-ness between every 2 columns. This is how we can get it

```
msno.heatmap(dirty_df)  # building a heatmap
```

yields



In this heatmap, a value near -1 means that if one variable appears, the other variable is likely to be missing and vice versa. A value near 0 means there is no dependence between the occurrence of missing values of two variables. A value near 1 means if one variable appears, the other variable is likely also present. In our example, `seller_agent` and `date` have correlation of 1; meaning when one is missing, the other is missing as well.

Missing values in `pandas`

`pandas` capabilities in handling missing values will be the main focus of the rest of this handout. Recall that

```
dirty_df = pd.read_csv('dirty_house_data.csv')
dirty_df
```

yields

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price
0	Harbor	3/9/2021	Milo	NaN	2.0	1.0	126.0	NaN	2001.0	NaN
1	Harbor	3/12/2021	Lola	30671.0	2.0	1.0	202.0	NaN	NaN	1.48e+06
2	Harbor	3/9/2021	Chubby	32061.0	2.0	1.0	120.0	NaN	1995.0	1.28e+06
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Camden	8/10/2020	Milo	30672.0	1.0	1.0	0.0	NaN	NaN	3.00e+05
5	Camden	8/10/2020	Milo	30673.0	3.0	1.0	220.0	NaN	2005.0	NaN
6	Camden	8/10/2020	Lola	30673.0	NaN	NaN	NaN	NaN	NaN	5.42e+05
7	Harbor	3/9/2020	Chubby	30001.0	2.0	1.0	159.0	NaN	NaN	1.46e+06
8	Harbor	4/3/2020	Lola	NaN	3.0	2.0	211.0	NaN	1890.0	2.85e+06
9	Harbor	4/3/2021	Babushka	NaN	3.0	2.0	128.0	NaN	1890.0	1.85e+06
10	Harbor	4/3/2021	Chubby	32065.0	2.0	1.0	215.0	NaN	1900.0	NaN
11	Harbor	4/3/2021	Lola	32066.0	2.0	1.0	130.0	NaN	1900.0	1.44e+06
12	Harbor	6/8/2020	Babushka	32066.0	2.0	2.0	336.0	NaN	1890.0	NaN
13	Camden	4/2/2021	Milo	30676.0	2.0	1.0	156.0	NaN	2005.0	1.04e+06
14	Camden	4/2/2021	Lola	20000.0	3.0	2.0	0.0	NaN	NaN	NaN
15	Camden	4/3/2021	Milo	30676.0	3.0	2.0	134.0	NaN	1900.0	1.46e+06
16	Camden	4/3/2021	Babushka	30675.0	3.0	2.0	94.0	NaN	NaN	8.50e+05
17	Camden	4/6/2021	Lola	30675.0	3.0	1.0	120.0	NaN	2014.0	1.60e+06
18	Fells	6/8/2020	Chubby	30677.0	3.0	2.0	400.0	NaN	2006.0	NaN
19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
20	Fells	6/8/2020	Lola	30000.0	3.0	2.0	202.0	NaN	2010.0	NaN
21	Fells	7/5/2020	Milo	NaN	2.0	1.0	181.0	NaN	NaN	9.41e+05

By calling

```
dirty_df.info()
```

we get

RangeIndex: 22 entries, 0 to 21

Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	area	20 non-null	object
1	date	20 non-null	object
2	seller_agent	20 non-null	object
3	zip_code	20 non-null	float64
4	bedroom	19 non-null	float64
5	bathroom	19 non-null	float64
6	land_size	19 non-null	float64
7	building_area	0 non-null	float64
8	year_built	12 non-null	float64
9	price	13 non-null	float64

dtypes: float64(7), object(3)

memory usage: 1.8+ KB

> According to this table, there are 22 rows; each column contains varying non-null (not missing) values.

Finding missing values

`.isna()`, `.isnull()`, & `notna()`

To detect `NaN` values, `pandas` uses either `.isna()` or `.isnull()`; these two methods of `isnull()` or `.isna()` do the same thing. They both return the `DataFrame` with `boolean` values indicating missing ones.

> **Example**
>
> ```
> dirty_df.isna()
> ```

yields

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price
0	False	False	False	True	False	False	False	True	False	True
1	False	False	False	False	False	False	False	True	True	False
2	False	False	False	False	False	False	False	True	False	False
3	True	True	True	True	True	True	True	True	True	True
4	False	False	False	False	False	False	False	True	True	False
5	False	False	False	False	False	False	False	True	False	True
6	False	False	False	False	True	True	True	True	True	False
7	False	False	False	False	False	False	False	True	True	False
8	False	False	False	True	False	False	False	True	False	False
9	False	False	False	True	False	False	False	True	False	False
10	False	False	False	False	False	False	False	True	False	True
11	False	False	False	False	False	False	False	True	False	False
12	False	False	False	False	False	False	False	True	False	True
13	False	False	False	False	False	False	False	True	False	False
14	False	False	False	False	False	False	False	True	True	True
15	False	False	False	False	False	False	False	True	False	False
16	False	False	False	False	False	False	False	True	True	False
17	False	False	False	False	False	False	False	True	False	False
18	False	False	False	False	False	False	False	True	False	True
19	True	True	True	True	True	True	True	True	True	True
20	False	False	False	False	False	False	False	True	False	True
21	False	False	False	True	False	False	False	True	True	False

You can also use `notna()` which is just the opposite of `isna()`.

Inspecting columns

`.isna().any()`

`.isna().any()` returns a `boolean` value for each column. If there is at least one missing value in that column, the result is `True`.

> ```
> dirty_df.isna().any()
> ```
>
> returns
>
> | | |
> |---|---|
> | area | True |
> | date | True |
> | seller_agent | True |
> | zip_code | True |
> | bedroom | True |
> | bathroom | True |
> | land_size | True |
> | building_area | True |
> | year_built | True |
> | price | False |
> | dtype: bool | |
>
> This shows all of the columns have at least one missing values.
>
> **Note:** In a similar manner, one can check which rows have at least one missing values by passing `axis = 1` to `any()` function. Here is an example:
>
> ```
> dirty_df.isna().any(axis = 1)
> ```
>
> returns

0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True
10	True
11	True
12	True
13	True
14	True
15	True
16	True
17	True
18	True
19	True
20	True

0	True
21	True
dtype: bool	

which implies each row has at least one missing value.

`.isna().sum()`

`.isna().sum()` counts the number of `NaN` values in each column.

```
dirty_df.isna().sum()
```

returns

area	2
date	2
seller_agent	2
zip_code	6
bedroom	3
bathroom	3
land_size	3
building_area	22
year_built	9
price	0
dtype: int64	

`building_area` has most missing values and each colum has at least `2` missing values.

In a very similar fashin, `axis = 1` can be passed to `sum()` function as well in order to get number of missing values in each row of the `DataFrame`. Here is an example

```
dirty_df.isna().sum(axis = 1)
```

returns

0	3
1	2
2	1
3	10
4	2
5	2
6	5
7	2
8	2
9	2
10	2
11	1
12	2
13	1
14	3
15	1
16	2
17	1
18	2
19	10
20	2
21	3

indicating that row index 3 and 19 have the most missing values (10 each)

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

`.isna().sum().sum()` goes one level deeper to find total number of missing values in the `DataFrame`.

> ```
> dirty_df.isna().sum().sum()
> ```
>
> returns `58`. There are total of `58` missing values from this `DataFrame`.

Dropping missing values

A quick visual inspection of `dirty_df` in

> | | area | date | seller_agent | zip_code | bedroom | bathroom | land_size | building_area | year_built | price |
> |---|---|---|---|---|---|---|---|---|---|---|
> | **0** | False | False | False | True | False | False | False | True | False | True |
> | **1** | False | False | False | False | False | False | False | True | True | False |
> | **2** | False | False | False | False | False | False | False | True | False | False |
> | **3** | True | True | True | True | True | True | True | True | True | True |
> | **4** | False | False | False | False | False | False | False | True | True | False |
> | **5** | False | False | False | False | False | False | False | True | False | True |
> | **6** | False | False | False | False | True | True | True | True | True | False |
> | **7** | False | False | False | False | False | False | False | True | True | False |
> | **8** | False | False | False | True | False | False | False | True | False | False |
> | **9** | False | False | False | True | False | False | False | True | False | False |
> | **10** | False | False | False | False | False | False | False | True | False | True |
> | **11** | False | False | False | False | False | False | False | True | False | False |
> | **12** | False | False | False | False | False | False | False | True | False | True |
> | **13** | False | False | False | False | False | False | False | True | False | False |
> | **14** | False | False | False | False | False | False | False | True | True | True |
> | **15** | False | False | False | False | False | False | False | True | False | False |
> | **16** | False | False | False | False | False | False | False | True | True | False |

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price
17	False	False	False	False	False	False	False	True	False	False
18	False	False	False	False	False	False	False	True	False	True
19	True	True	True	True	True	True	True	True	True	True
20	False	False	False	False	False	False	False	True	False	True
21	False	False	False	True	False	False	False	True	True	False

reveals that rows at indices `3` and `19` all all missing. Additionally, `building_area` has all the values missing as well. Rows `3` and `19` and column `buildimng_area` can be dropped using `drop()` method of `pandas`. The corresponding command would be

```
dirty_df.drop(columns = ['building_area'], inplace = True)
dirty_df.drop(index  = [3,19], inplace = True)
```

STOP: A much cleaner and more concise way of doing exactly same thing is to drop columns and rows in just one line of snippet as in

```
dirty_df.drop(columns = ['building_area'], index  = [3,19], inplace = True)
```

which yields exactly same output.

However, this visual inspection for a slightly bigger data-set is proven to be very slow and erroneous. Instead we can use `dropna()` method.

`.dropna()`

`.dropna()` is used for dropping a row or column with missing values.

```
dirty_df.dropna()
```

returns

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price

This is an empty `DataFrame` as there is at least one missing value in each row. The `.dropna()` has removed all the rows with at least one missing value. `how` parameter controls behavior of `dropna()`.

`how` parameter is used to set condition for dropping:

- `how='any'` : drop if there is any missing value. `any` is the default value for the `how` parameter.
- `how='all'` : drop if all values are missing

```
dirty_df.dropna(how = 'all')
```

removes all the rows that `all` their values are missing and gives out

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price
0	Harbor	3/9/2021	Milo	NaN	2.0	1.0	126.0	NaN	2001.0	0.00e+00
1	Harbor	3/12/2021	Lola	30671.0	2.0	1.0	202.0	NaN	NaN	1.48e+06
2	Harbor	3/9/2021	Chubby	32061.0	2.0	1.0	120.0	NaN	1995.0	1.28e+06
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00e+00
4	Camden	8/10/2020	Milo	30672.0	1.0	1.0	0.0	NaN	NaN	3.00e+05
5	Camden	8/10/2020	Milo	30673.0	3.0	1.0	220.0	NaN	2005.0	0.00e+00
6	Camden	8/10/2020	Lola	30673.0	NaN	NaN	NaN	NaN	NaN	5.42e+05
7	Harbor	3/9/2020	Chubby	30001.0	2.0	1.0	159.0	NaN	NaN	1.46e+06
8	Harbor	4/3/2020	Lola	NaN	3.0	2.0	211.0	NaN	1890.0	2.85e+06
9	Harbor	4/3/2021	Babushka	NaN	3.0	2.0	128.0	NaN	1890.0	1.85e+06
10	Harbor	4/3/2021	Chubby	32065.0	2.0	1.0	215.0	NaN	1900.0	0.00e+00
11	Harbor	4/3/2021	Lola	32066.0	2.0	1.0	130.0	NaN	1900.0	1.44e+06
12	Harbor	6/8/2020	Babushka	32066.0	2.0	2.0	336.0	NaN	1890.0	0.00e+00
13	Camden	4/2/2021	Milo	30676.0	2.0	1.0	156.0	NaN	2005.0	1.04e+06
14	Camden	4/2/2021	Lola	20000.0	3.0	2.0	0.0	NaN	NaN	0.00e+00
15	Camden	4/3/2021	Milo	30676.0	3.0	2.0	134.0	NaN	1900.0	1.46e+06
16	Camden	4/3/2021	Babushka	30675.0	3.0	2.0	94.0	NaN	NaN	8.50e+05
17	Camden	4/6/2021	Lola	30675.0	3.0	1.0	120.0	NaN	2014.0	1.60e+06
18	Fells	6/8/2020	Chubby	30677.0	3.0	2.0	400.0	NaN	2006.0	0.00e+00
19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00e+00

	area	date	seller_agent	zip_code	bedroom	bathroom	land_size	building_area	year_built	price
20	Fells	6/8/2020	Lola	30000.0	3.0	2.0	202.0	NaN	2010.0	0.00e+00
21	Fells	7/5/2020	Milo	NaN	2.0	1.0	181.0	NaN	NaN	9.41e+05

- By default `dropna()` removes rows; by setting `axis = 1`, we can remove columns as well.
- `dropna()` has an optional argument of `inplace`.
- `dropna()` has an optional argument of `thresh`, as well. The `thresh` argument dictates the threshold of number of non-missing values for displaying. As an example setting `thresh = 9` will guarantee the rows with at least 9 non-missing values will be displayed.

`pandas`'s handling of `NaN` in mathematical operations

`pandas` shows missing values with `NaN` and ignores them in its calculations. As an example

> **Example**
>
> ```
> dirty_df.price.mean()
> ```
>
> will find `mean` of values in the column of `price` and during its calculation it will ignore `NaN` cells as if they do not exist. It will report `1313769.23`. Please note that `pandas` will report `NaN` in its calculations if all values in that `Series` or `DataFrame` are `NaN`.

Filling missing values

We can use `fillna()` method to replace `NaN` values with any particular values.

Replace `NaN` with a specific scalar

Take a look at the following snippet

> **Example**
>
> ```
> dirty_df.price.fillna(0, inplace = True)
> ```
>
> will replace all the NaN values in the price column with 0. This is not necessarily a great idea, as it is unrealistic to assume that the selling price of a house is 0!

Here is another example

Example

```
price_mean = dirty_df.price.mean()
dirty_df.price.fillna(price_mean, inplace = True)
```

will replace all the `NaN` values in the column of `price` with the `mean` value of `price` for the existing `price` values.

Important Note: While using `mean` for filling the missing value cells is common, we must acknowledge that `mean` is greatly affected by outliers in your data. As a result, if outliers are present in our data, then `median` will be a better option.

Replace `NaN` with values before or after

`NaN` values can be filled based on the last known value. We can use the `method` parameter of `fillna()` for that purpose. First, let's have a look at the `price` column.

```
dirty_df.price
```

yields

0	NaN
1	1.48e+06
2	1.28e+06
3	NaN
4	3.00e+05
5	NaN
6	5.42e+05
7	1.46e+06
8	2.85e+06

0	NaN
9	1.85e+06
10	NaN
11	1.44e+06
12	NaN
13	1.04e+06
14	NaN
15	1.46e+06
16	8.50e+05
17	1.60e+06
18	NaN
19	NaN
20	NaN
21	9.41e+05
Name: price, dtype: float64	

now let's replace `NaN` values with the last known valid value. >

Example

```
dirty_df.price.fillna(method = 'ffill', inplace = True)
dirty_df.price
```

yields

0	NaN
1	1.48e+06
2	1.28e+06
3	1.28e+06
4	3.00e+05
5	3.00e+05
6	5.42e+05
7	1.46e+06
8	2.85e+06
9	1.85e+06
10	1.85e+06
11	1.44e+06
12	1.44e+06
13	1.04e+06
14	1.04e+06
15	1.46e+06
16	8.50e+05
17	1.60e+06
18	1.60e+06
19	1.60e+06
20	1.60e+06

0	NaN
21	9.41e+05
Name: price, dtype: float64	

`ffill` for `forward fill` replaces missing values with those in the previous row. For example, `NaN` value in index 3 was replaced by the value in index 2 (`1.28e+06`). As the first value in column `price` is missing, `ffill` cannot replace it with anything. For that, you can use `bfill`

You can also choose `bfill` for `backward fill` . Here is another example

Example

```
dirty_df.price.fillna(method = 'ffill', inplace = True)
dirty_df.price.fillna(method = 'bfill', inplace = True)
dirty_df.price
```

will first replace all `NaN` with the `ffill` method. Then using the `bfill` method, it will fill the `NaN` values atop the `price` column. Here is the outcome >

0	1.48e+06
1	1.48e+06
2	1.28e+06
3	1.28e+06
4	3.00e+05
5	3.00e+05
6	5.42e+05
7	1.46e+06
8	2.85e+06
9	1.85e+06
10	1.85e+06

0	1.48e+06
11	1.44e+06
12	1.44e+06
13	1.04e+06
14	1.04e+06
15	1.46e+06
16	8.50e+05
17	1.60e+06
18	1.60e+06
19	1.60e+06
20	1.60e+06
21	9.41e+05
Name: price, dtype: float64	

as you can see, the first `NaN` value was replaced by the value after it in index `1`, which is `1.48e+06`.

Replace `NaN` using interpolation

`interpolate()` method could be used to perform linear (by default) interpolation on our data. Here is an example

Example

```
dirty_df.price.interpolate()
```

yields

0	0.00e+00
1	1.48e+06
2	1.28e+06
3	0.00e+00
4	3.00e+05
5	0.00e+00
6	5.42e+05
7	1.46e+06
8	2.85e+06
9	1.85e+06
10	0.00e+00
11	1.44e+06
12	0.00e+00
13	1.04e+06
14	0.00e+00
15	1.46e+06
16	8.50e+05
17	1.60e+06
18	0.00e+00
19	0.00e+00
20	0.00e+00

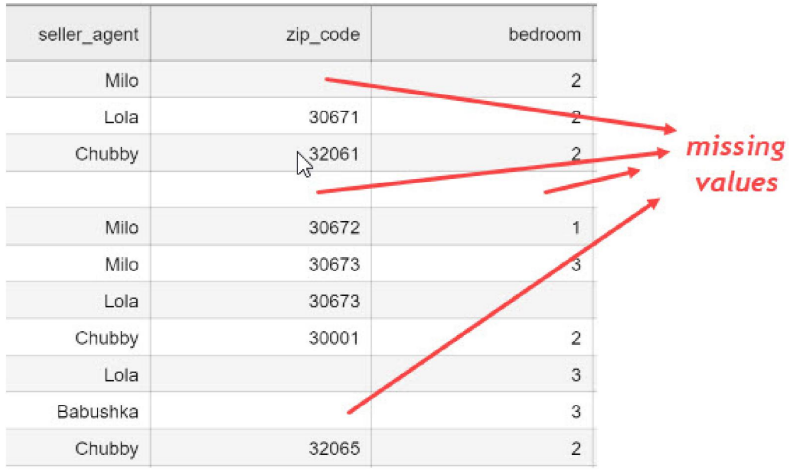
0	0.00e+00
21	9.41e+05

Note: `interpolate` method fills missing values using an interpolation method. Default is a `linear` interpolation; however, one can change the type of interpolation using the argument of `method`.

Handling non-standard missing values

So far, we have assumed that all the missing values in the input `csv` file have no values (are missing). This is depicted in the following snapshot from the `dirty_house_data.csv` file

seller_agent	zip_code	bedroom
Milo		2
Lola	30671	2
Chubby	32061	2
Milo	30672	1
Milo	30673	3
Lola	30673	
Chubby	30001	2
Lola		3
Babushka		3
Chubby	32065	2



`pandas` recognizes these empty cells as `NaN`. However, often missing values have different formats. For example, missing values could be entered as `n/a`, `-`, `--`, `?`, `na`, `Na`, `NA`, `nA`, `???`. This is a common problem as users have different preferences in entering missing values. One possible solution for this problem is recognizing these values while reading the `csv` file. Here is an example:

Example

```
missing_values_char = ["-", "--", "?", "???", "na","Na","nA", "NA","n/a"]
df = pd.read_csv("dirty_house_data.csv", na_values = missing_values_char)
```

puts all of the missing value indicators in a list and feeds that list to the `na_values` argument of `read_csv`. Then when we import the data, `pandas` will recognize them immediately.

References

This document is an adaption from the following references:

1- *Python online documentation*; available at https://docs.python.org/3/

2- *Exploratory Data Analysis with Python*; Suresh Kumar Mukhiya and Usman Ahmed, Packt publications, 2020

3- *Introduction to programming in Python*; available at https://introcs.cs.princeton.edu/Python/home

4- *Introduction to Computation and Programming Using Python: With Application to Understanding Data*, John Guttag, The MIT Press, 2nd edition, 2016

5- *Introduction to Python for Computer Science and Data Science*, Paul J. Deitel, Harvey Deitel, Pearson, 1st edition, 2019

6- *Data Mining for Business Analytics*, Galit Shmueli, Peter C. Bruce, Peter Gedeck, Nitin R. Patel, Wiley, 2020

7- *Interactive Data Visualization with Python*; Abha Belorkar, Sharath Chandra Guntuka, Shubhangi Hora, and Anshu Kumar. Packt publications, 2020

8- https://www.learnpython.org/

9- https://numpy.org/

10- https://pandas.pydata.org/

11- *Time Series Analysis and Its Applications*; Robert H. Shumway, David S. Stoffer, Springer publications, 2017

12- *Pandas Cookbook*, Matt Harrison and Theodore Petrou, Packt publications, 2020

13- https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html