Missing values in Data

- We have seen how to create `Series` and `DataFrames`, and how to read flat files in to Python using `Pandas`.
- We also learned about indexing, slicing and filtering in Pandas.
- Today we will learn about missing values.
- How missing values are represented in Pandas.
- How missing values affect dataset calculations.
- How to find / remove / replace missing values.

Missing data

- Real world data is rarely 'clean'.
- There will be missing values, outliers, mistakes in the data, poorly named columns, poorly structured data.
- Different sources indicate missing data in different ways eg `NULL`, `NaN`, `NA`, `0`, `None`, `-999`
- Python uses `NaN` (`np.nan` to type it in to Python) and None.
- Pandas usually converts `None` to `NaN` automatically. For example:

In [2]:
```
import numpy as np
import pandas as pd

a = pd.Series([1, np.nan, 3, None, 5, np.nan])

print(a)
```

```
0    1.0
1    NaN
2    3.0
3    NaN
4    5.0
5    NaN
dtype: float64
```

Reading in files with missing data

- When reading in flat files, Pandas automatically converts blank values and NA values to `NaN` values.
- However, it does not notice other missing value codes such as `N/A`, `?`, `-999` as `NaN`.
- These can be included in the read_csv command.
- For example `df = pd.read_csv("df.csv", na_values = ["N/A", "?", "-999", "NA"])`

Like a virus

NaN is a bit like a data virus–it infects any other object it touches.

In [3]:
```
a = pd.Series([1, np.nan, 3, None, 5])

print(a + 1)

print(a * 2)
```

```
0    2.0
1    NaN
2    4.0
3    NaN
4    6.0
dtype: float64
0     2.0
1     NaN
2     6.0
3     NaN
4    10.0
dtype: float64
```

In [25]:
```
b = pd.Series([2, 4, 6, 8, 10])

print(a + b)
```

```
0     3.0
1     NaN
2     9.0
3     NaN
4    15.0
dtype: float64
```

Missing data and summary functions

In Pandas, summary functions such as `a.sum()`, `a.min()`, `a.max()` all ignore the `NaN` values. This is not the case for NumPy arrays.

In [26]:
```
print(a)
print(a.sum())
print(a.min())
print(a.max())
print(a.mean())
```

```
0    1.0
1    NaN
2    3.0
3    NaN
4    5.0
dtype: float64
9.0
1.0
5.0
3.0
```

Operations on missing values

- `df.isna()` returns a Boolean with True for missing values
- `df.isnull()` does the same as `df.isna()`: it returns True for values that are NaN.
- `df.notnull()` (or `df.notna()`) returns True for values that are not NaN.
- Use these as indexes inside square brackets to find rows for which you do or don't have NaNs in certain columns.
  - For example: `df[df.height.notna()]` gives all rows of the DataFrame `df` that do not have a missing height value.
- `df.isna().any()` gives one value for each column / variable which is True if that column contains at least 1 NaN.
- `df.isna().sum()` to see total number of NaNs in each column of a Data Frame.
- `df.dropna()` removes rows with any NaNs from the Series or DataFrame.
- `df.fillna(0)` replaces all NaNs with 0.
- `fill.na("missing")` replaces all NaN values with `"missing"`.

In [27]:
```
print(a)
print(a.isna())
```

```
0    1.0
1    NaN
2    3.0
3    NaN
4    5.0
dtype: float64
0    False
1     True
2    False
3     True
4    False
dtype: bool
```

In [28]:
```
print(a.isnull())
```

```
0    False
1     True
2    False
3     True
4    False
dtype: bool
```

In [29]:
```
print(a[a.isna()])
```

```
1   NaN
3   NaN
dtype: float64
```

In [30]:
```
print(a[a.isnull()])
```

```
1   NaN
3   NaN
dtype: float64
```

In [31]:
```
print(a)
print(a.notna())
```

```
0    1.0
1    NaN
2    3.0
3    NaN
4    5.0
dtype: float64
0     True
1    False
2     True
3    False
4     True
dtype: bool
```

In [32]:
```
print(a.notnull())
```

```
0     True
1    False
2     True
3    False
4     True
dtype: bool
```

In [4]:
```
print(a[a.notna()])
```

```
0    1.0
2    3.0
4    5.0
dtype: float64
```

In [34]:
```
print(a[a.notnull()])
```

```
0    1.0
2    3.0
4    5.0
dtype: float64
```

In [5]:
```
print(a)
print(a.isna().any())
```

```
0    1.0
1    NaN
2    3.0
3    NaN
4    5.0
dtype: float64
True
```

In [6]:
```
print(a.isna())
```

```
0    False
1     True
2    False
3     True
4    False
dtype: bool
```

In [36]:
```
print(a.isna().sum())
```

```
2
```

In [37]:
```
print(a.dropna())
```

```
0    1.0
2    3.0
4    5.0
dtype: float64
```

In [38]:
```
print(a.fillna(0))
```

```
0    1.0
1    0.0
2    3.0
3    0.0
4    5.0
dtype: float64
```

In [39]:
```
print(a.fillna("Missing"))
```

```
0        1.0
1    Missing
2        3.0
3    Missing
4        5.0
dtype: object
```

Examples of operations on missing data in DataFrames

In [18]:
```
df = pd.DataFrame({"name": ["Jack", "Joe", "Charlie", "Frank", "Jonny"],
                   "age": [31, 29, np.nan, 57, np.nan],
                  "height": [np.nan, 193, 194, 183, 155]})
display(df)
```

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0
4	Jonny	NaN	155.0

In [8]:
```
print(df.sum())
```

```
name      JackJoeCharlieFrank
age                     117.0
height                  570.0
dtype: object
```

In [42]:
```
df.min()
```

Out[42]:
```
name      Charlie
age          29.0
height      183.0
dtype: object
```

In [43]:
```
print(df.max())
```

```
name        Joe
age        57.0
height    194.0
dtype: object
```

In [56]:
```
# print(df.mean()) # error,
```

In [76]:
```
df.mean(numeric_only=True)
```

Out[76]:
```
age        39.0
height    190.0
dtype: float64
```

In [9]:
```
df
print(df.isna())
```

```
    name    age  height
0  False  False    True
1  False  False   False
2  False   True   False
3  False  False   False
```

In [47]:
```
print(df.age.isna())
```

```
0    False
1    False
2     True
3    False
Name: age, dtype: bool
```

In [48]:
```
print(df.height.isna())
```

```
0     True
1    False
2    False
3    False
Name: height, dtype: bool
```

In [50]:
```
print(df[df.age.isna()])
```

```
      name  age  height
2  Charlie  NaN   194.0
```

In [51]:
```
print(df.name[df.height.isna()])
```

```
0    Jack
Name: name, dtype: object
```

In [53]:
```
print(df.notna()) # return dataframe with booleans
```

```
   name    age  height
0  True   True   False
1  True   True    True
2  True  False    True
3  True   True    True
```

In [54]:
```
print(df.age.notna()) # returns serie with boolean values
```

```
0     True
1     True
2    False
3     True
Name: age, dtype: bool
```

In [55]:
```
print(df.height.notna())
```

```
0    False
1     True
2     True
3     True
Name: height, dtype: bool
```

In [10]:
```
print(df[df.height.notna()])
```

```
      name   age  height
1      Joe  29.0   193.0
2  Charlie   NaN   194.0
3    Frank  57.0   183.0
```

In [11]:
```
print(df.age[df.height.notna()])
```

```
1    29.0
2     NaN
3    57.0
Name: age, dtype: float64
```

In [60]:
```
print(df.loc[df.height.notna(), ["name", "height"]])
```

```
      name  height
1      Joe   193.0
2  Charlie   194.0
3    Frank   183.0
```

In [20]:
```
display(df)
```

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0
4	Jonny	NaN	155.0

In [16]:
```
df.isna().any()
```

Out[16]:
```
name      False
age        True
height     True
dtype: bool
```

In [19]:
```
df.isna().sum()
```

Out[19]:
```
name      0
age       2
height    1
dtype: int64
```

In [63]:
```
# help(df.fillna)
```

In [21]:
```
df.fillna(0)
```

Out[21]:

	name	age	height
0	Jack	31.0	0.0
1	Joe	29.0	193.0
2	Charlie	0.0	194.0
3	Frank	57.0	183.0
4	Jonny	0.0	155.0

In [22]:
```
df.fillna("Missing")
```

Out[22]:

	name	age	height
0	Jack	31.0	Missing
1	Joe	29.0	193.0
2	Charlie	Missing	194.0
3	Frank	57.0	183.0
4	Jonny	Missing	155.0

In [23]:
```
df.fillna(method = 'ffill', axis = 1) # column
```

Out[23]:

	name	age	height
0	Jack	31.0	31.0
1	Joe	29.0	193.0
2	Charlie	Charlie	194.0
3	Frank	57.0	183.0
4	Jonny	Jonny	155.0

In [24]:
```
df.fillna(method = 'ffill', axis = 0) # row (default)
```

Out[24]:

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	29.0	194.0
3	Frank	57.0	183.0
4	Jonny	57.0	155.0

In [25]:
```
df.fillna(method = 'bfill', axis = 1) # column
```

Out[25]:

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	194.0	194.0
3	Frank	57.0	183.0
4	Jonny	155.0	155.0

In [26]:
```
df.fillna(method = 'bfill', axis = 0) # row
```

Out[26]:

	name	age	height
0	Jack	31.0	193.0
1	Joe	29.0	193.0
2	Charlie	57.0	194.0
3	Frank	57.0	183.0
4	Jonny	NaN	155.0

The `inplace` argument and `pd.fillna` function

Use the `inplace = True` argument to save the filled dataset as the new df (inplace = False by default).

In [27]:
```
#df.fillna(df.mean())
display(df)
```

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0
4	Jonny	NaN	155.0

In [31]:
```
df.fillna(df.mean(numeric_only=True), inplace = True)
display(df)
```

	name	age	height
0	Jack	31.0	181.25
1	Joe	29.0	193.00
2	Charlie	39.0	194.00
3	Frank	57.0	183.00
4	Jonny	39.0	155.00

Drop missing values

In [54]:
```
# Setup df again to overwrite the inplace = True from the last line

df = pd.DataFrame({"name": ["Jack", "Joe", "Charlie", "Frank"],
                   "age": [31, 29, np.nan, 57],
                  "height": [np.nan, 193, 194, 183]})

display(df)
df.dropna()
```

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0

Out[54]:

	name	age	height
1	Joe	29.0	193.0
3	Frank	57.0	183.0

Notice that `dropna()` drops all rows containing a missing value.

Use the argument `axis = "columns"` in dropna to drop all columns containing a missing value.

In [35]:
```
# Drop all columns containing a missing value
df.dropna(axis = "columns")
```

Out[35]:

	name
0	Jack
1	Joe
2	Charlie
3	Frank

`df.dropna(axis = 1)` does the same thing as `df.dropna(axis="columns")`

In [36]:
```
df.dropna(axis = 1)
```

Out[36]:

	name
0	Jack
1	Joe
2	Charlie
3	Frank

If `how='all'` is specified, only rows/cols that are all missing values will be dropped.

In [61]:
```
df.dropna(axis = 'columns', how ='all')
```

Out[61]:

	name	age	height
0	Jack	NaN	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0

Use thresh to specify a minimum number of non-missing values for the row/col to be kept.

In [38]:
```
df.dropna(axis = 'rows', thresh = 1)
```

Out[38]:

	name	age	height
0	Jack	31.0	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0

In [39]:
```
df.dropna(axis = 'rows', thresh = 3)
```

Out[39]:

	name	age	height
1	Joe	29.0	193.0
3	Frank	57.0	183.0

Using the `replace` function for missing values

- Suppose you know that if a certain value appears in a dataset, say 0, it is actually a missing value.
- Then we can use the replace function to replace all `0` values with NaNs.
- `df.replace(0, np.nan)`
- This can also work the other way around, a bit like a simplified `fillna()` : `df.replace(np.nan)`

In [59]:
```
df
```

Out[59]:

	name	age	height
0	Jack	NaN	NaN
1	Joe	29.0	193.0
2	Charlie	NaN	194.0
3	Frank	57.0	183.0

In [58]:
```
df.replace(31, np.nan)
```

Imputation

- Replacing missing values with substituted values is called imputation.
- This can be a very advanced topic.
- There are many methods for imputation, for example `mean imputation`, `regression imputation`.
- We have used `fillna()` and `replace()` for imputation.
- There are other functions in Python for imputation eg: `df.interpolate(method = 'linear', axis = 0)`
- "Interpolation is a type of estimation, a method of constructing new data points within the range of a discrete set of known data points."

Exercises

- `data_missing.csv` is on Moodle. Download the dataset and read it in to Python, making sure all missing values are read in correctly.
- Are there any unexpected values in the OWN_OCCUPIED column? If so, replace them with NaN.
- Use Python code to show which columns have missing values.
- Find the number of missing values in each column.
- Filter the dataset so we only see the rows with no missing values.
- Filter the dataset so we only see the rows with two or less missing values.
- Show the columns ST_NUM and ST_NAME for the rows with missing values in OWN_OCCUPIED.
- Show the columns ST_NUM and ST_NAME for the rows that do not have missing values in OWN_OCCUPIED.
- Fill in the missing values of NUM_BEDROOMS with the previous value in the column.
- Fill in the missing values of NUM_BEDROOMS with the median value of the column. Store this as the new dataset.