Load data from file

Most often data will come from somewhere, often csv files, and using `pd.read_csv()` will allow smooth creation of DataFrames.

Let's load that same heart-attack.csv that we used in Numpy before:

In [14]:
```
import numpy as np
import pandas as pd
```

In [15]:
```
headers = pd.pandas.read_fwf('auto-mpg.names')
headers = headers["Title: Auto-Mpg Data"].iloc[23:32].tolist()
data = pd.read_fwf('auto-mpg.data',header=None, names=headers)
```

After loading data, it is good practice to check what we have. Usually, the sequences is:

1. Check dimension
2. Peek at the first rows
3. Get info on data types and missing values
4. Summarize columns

In [16]:
```
# Check dimension (rows, columns)
data.shape
```

Out[16]:
```
(398, 9)
```

In [17]:
```
# Peek at the first rows
data.head()
```

Out[17]:

	1. mpg: continuous	2. cylinders: multi-valued discrete	3. displacement: continuous	4. horsepower: continuous	5. weight: continuous	6. acceleration: continuous	7. model year: multi-valued discrete	c
0	18.0	8	307.0	130.0	3504.0	12.0	70	
1	15.0	8	350.0	165.0	3693.0	11.5	70	
2	18.0	8	318.0	150.0	3436.0	11.0	70	
3	16.0	8	304.0	150.0	3433.0	12.0	70	
4	17.0	8	302.0	140.0	3449.0	10.5	70	

In [18]:
```
# Column names are
data.columns
```

Out[18]:
```
Index(['1. mpg:           continuous',
       '2. cylinders:     multi-valued discrete',
       '3. displacement:  continuous', '4. horsepower:    continuous',
       '5. weight:        continuous', '6. acceleration:  continuous',
       '7. model year:    multi-valued discrete',
       '8. origin:        multi-valued discrete',
       '9. car name:      string (unique for each instance)'],
      dtype='object')
```

In [19]:
```
# Get info on data types and missing values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column                                          Non-Null Count  Dt
ype
---  ------                                          --------------  --
---
 0   1. mpg:           continuous                    398 non-null    fl
oat64
 1   2. cylinders:     multi-valued discrete         398 non-null    in
t64
 2   3. displacement:  continuous                    398 non-null    fl
oat64
 3   4. horsepower:    continuous                    398 non-null    ob
ject
 4   5. weight:        continuous                    398 non-null    fl
oat64
 5   6. acceleration:  continuous                    398 non-null    fl
oat64
 6   7. model year:    multi-valued discrete         398 non-null    in
t64
 7   8. origin:        multi-valued discrete         398 non-null    in
t64
 8   9. car name:      string (unique for each instance)  398 non-null    ob
ject
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

Summarize values

What is the mean, std, min, max in each column?

In [20]:
```
data.mean()
```

```
/var/folders/pj/h_6374bs7s772tx06t7bfvl00000gn/T/ipykernel_13065/531903386.p
y:1: FutureWarning: The default value of numeric_only in DataFrame.mean is d
eprecated. In a future version, it will default to False. In addition, speci
fying 'numeric_only=None' is deprecated. Select only valid columns or specif
y the value of numeric_only to silence this warning.
  data.mean()
```

Out[20]:
```
1. mpg:           continuous                 23.514573
2. cylinders:     multi-valued discrete       5.454774
3. displacement:  continuous                193.425879
5. weight:        continuous               2970.424623
6. acceleration:  continuous                 15.568090
7. model year:    multi-valued discrete      76.010050
8. origin:        multi-valued discrete       1.572864
dtype: float64
```

In [21]:
```
# where are the other columns? Check data types
data.dtypes
```

Out[21]:
```
1. mpg:           continuous                          float64
2. cylinders:     multi-valued discrete                 int64
3. displacement:  continuous                          float64
4. horsepower:    continuous                           object
5. weight:        continuous                          float64
6. acceleration:  continuous                          float64
7. model year:    multi-valued discrete                 int64
8. origin:        multi-valued discrete                 int64
9. car name:      string (unique for each instance)    object
dtype: object
```

Notice that many columns are of type object, which is not a number. Maybe this has to do with missing values? We know from peeking at the first rows that there are '?' values in there. Let's replace these with the string NaN for not-a-number.

In [22]:
```
# replace '?' with 'NaN'
data = data.replace({'?': 'NaN'})
data.head()
```

Out[22]:

	1. mpg: continuous	2. cylinders: multi-valued discrete	3. displacement: continuous	4. horsepower: continuous	5. weight: continuous	6. acceleration: continuous	7. model year: multi-valued discrete	c
0	18.0	8	307.0	130.0	3504.0	12.0	70	
1	15.0	8	350.0	165.0	3693.0	11.5	70	
2	18.0	8	318.0	150.0	3436.0	11.0	70	
3	16.0	8	304.0	150.0	3433.0	12.0	70	
4	17.0	8	302.0	140.0	3449.0	10.5	70	

Pandas knows that 'NaN' probably means that numbers are missing. Now we can convert the data type from object to float

In [23]:
```
# convert dtypes, removed string column of car names to remove error when co
data = data.loc[:, data.columns != '9. car name:      string (unique for eac
data.dtypes
```

Out[23]:
```
1. mpg:           continuous               float64
2. cylinders:     multi-valued discrete    float64
3. displacement:  continuous               float64
4. horsepower:    continuous               float64
5. weight:        continuous               float64
6. acceleration:  continuous               float64
7. model year:    multi-valued discrete    float64
8. origin:        multi-valued discrete    float64
dtype: object
```

We could have loaded the data with the `na_values` argument to indicate that '?' means missing number:

In [24]:
```
headers = pd.pandas.read_fwf('auto-mpg.names')
headers = headers["Title: Auto-Mpg Data"].iloc[23:32].tolist()
data = pd.read_fwf('auto-mpg.data',header=None, names=headers, na_values='?')
data.dtypes
```

Out[24]:
```
1. mpg:           continuous                          float64
2. cylinders:     multi-valued discrete                 int64
3. displacement:  continuous                          float64
4. horsepower:    continuous                          float64
5. weight:        continuous                          float64
6. acceleration:  continuous                          float64
7. model year:    multi-valued discrete                 int64
8. origin:        multi-valued discrete                 int64
9. car name:      string (unique for each instance)    object
dtype: object
```

This worked nicely. Now we can describe all columns, meaning printing basic statistics. Note that by default Pandas ignores NaN, whereas Numpy does not.

In [25]:
```
data.describe() # ignores NaN
```

Out[25]:

	1. mpg: continuous	2. cylinders: multi-valued discrete	3. displacement: continuous	4. horsepower: continuous	5. weight: continuous	6. acceleration: continuous	ye
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	39
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	

We could be interested by these statistics in each of the genders. To get these, we first group values by gender, then ask for the description. We will only look at age for clarity

In [26]:
```
data.groupby(by='8. origin:         multi-valued discrete').describe()['1. mp
```

Out[26]:

8. origin: multi-valued discrete	count	mean	std	min	25%	50%	75%	max
1	249.0	20.083534	6.402892	9.0	15.0	18.5	24.00	39.0
2	70.0	27.891429	6.723930	16.2	24.0	26.5	30.65	44.3
3	79.0	30.450633	6.090048	18.0	25.7	31.6	34.05	46.6

Find NaNs

How many NaNs in each column?

We can ask which entries are null, which produces a boolean array

In [27]:
```
data.isnull()
```

Out[27]:

	1. mpg: continuous	2. cylinders: multi-valued discrete	3. displacement: continuous	4. horsepower: continuous	5. weight: continuous	6. acceleration: continuous	7. model year: multi-valued discrete
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
393	False	False	False	False	False	False	False
394	False	False	False	False	False	False	False
395	False	False	False	False	False	False	False
396	False	False	False	False	False	False	False
397	False	False	False	False	False	False	False

398 rows × 9 columns

Applying `sum()` to this boolean array will count the number of `True` values in each column

In [28]:
```
data.isnull().sum()
```

Out[28]:
```
1. mpg:           continuous                          0
2. cylinders:     multi-valued discrete               0
3. displacement:  continuous                          0
4. horsepower:    continuous                          6
5. weight:        continuous                          0
6. acceleration:  continuous                          0
7. model year:    multi-valued discrete               0
8. origin:        multi-valued discrete               0
9. car name:      string (unique for each instance)   0
dtype: int64
```

We get complementary information from `info()`

In [29]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column                                                Non-Null Count  Dt
ype
---  ------                                                --------------  --
---
 0   1. mpg:           continuous                          398 non-null    fl
oat64
 1   2. cylinders:     multi-valued discrete               398 non-null    in
t64
 2   3. displacement:  continuous                          398 non-null    fl
oat64
 3   4. horsepower:    continuous                          392 non-null    fl
oat64
 4   5. weight:        continuous                          398 non-null    fl
oat64
 5   6. acceleration:  continuous                          398 non-null    fl
oat64
 6   7. model year:    multi-valued discrete               398 non-null    in
t64
 7   8. origin:        multi-valued discrete               398 non-null    in
t64
 8   9. car name:      string (unique for each instance)   398 non-null    ob
ject
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

We can fill (replace) these missing values, for example with the minimum value in each column

In [30]:
```
data.fillna(data.min()).describe()
```

Out[30]:

	1. mpg: continuous	2. cylinders: multi-valued discrete	3. displacement: continuous	4. horsepower: continuous	5. weight: continuous	6. acceleration: continuous	ye
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	39
mean	23.514573	5.454774	193.425879	103.587940	2970.424623	15.568090	7
std	7.815984	1.701004	104.269838	38.859575	846.841774	2.757689	
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	7
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	7
50%	23.000000	4.000000	148.500000	92.000000	2803.500000	15.500000	7
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	7
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	8

Count unique values (a histogram)

We finish off, with our good friend the histogram

In [31]:
```
data['8. origin:          multi-valued discrete'].value_counts()
```

Out[31]:
```
1    249
3     79
2     70
Name: 8. origin:          multi-valued discrete, dtype: int64
```