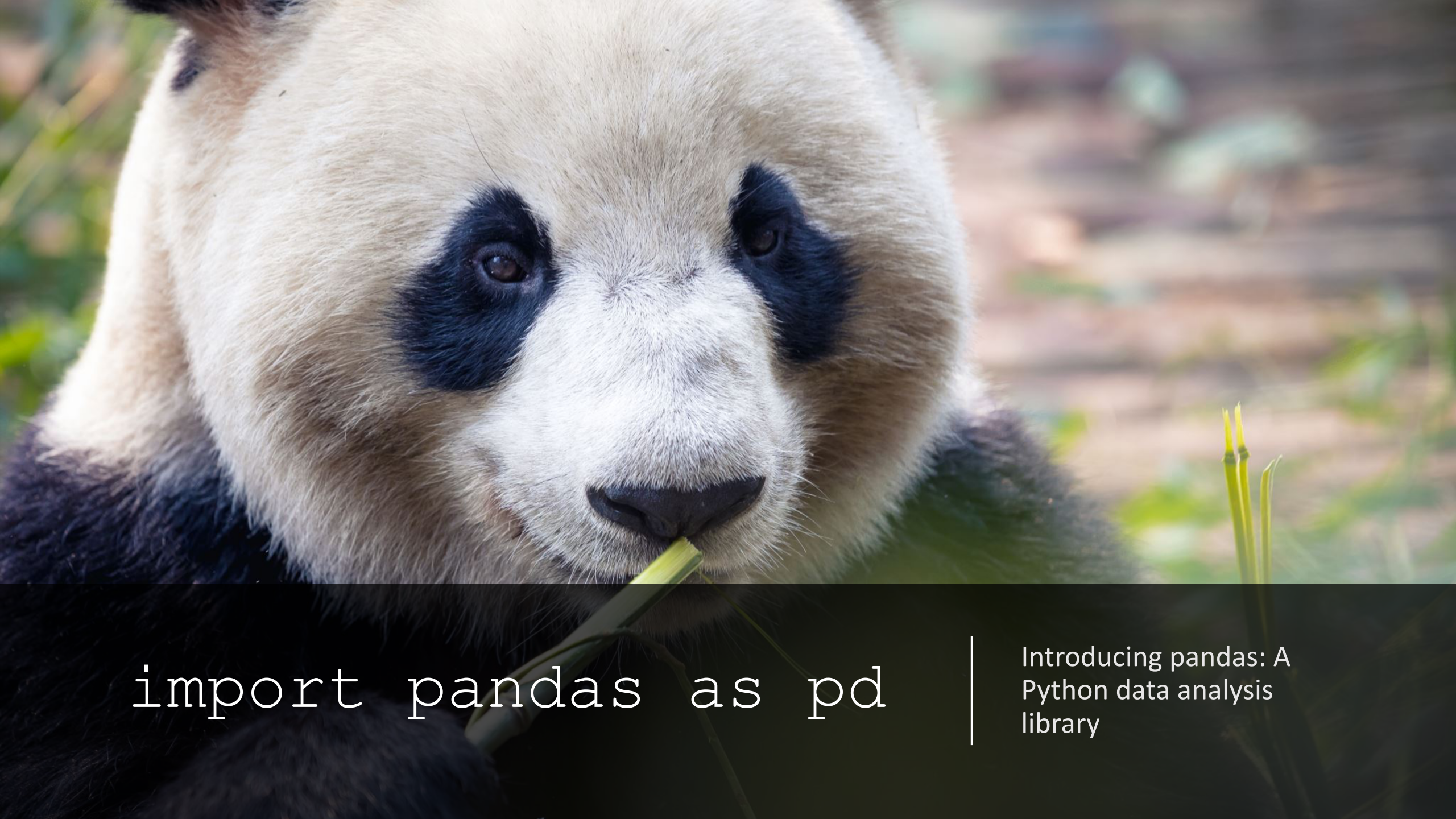

import pandas as pd

Introducing pandas: A Python data analysis library

```
[4] import pandas as pd
```

```
[5] data = pd.read_csv('/content/gdrive/My Drive/[YCMI_CBDS Summer Course] Data/diabetes.csv')
```

```
[6] data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

pd.read_csv

Loads data from a CSV table, returns a DataFrame.

Related:

pd.to_csv
pd.read_sql
pd.read_json
pd.read_excel
pd.read_html

Common `pd.read_csv` options

- Only load specific columns
  - `pd.read_csv("filename.csv", usecols=['patient_id', 'age'])`
- For files where columns are separated with tabs instead of commas
  - `pd.read_csv("warfarin.tsv", sep="\t")`
- For files without a header row (columns will be numbered from 0)
  - `pd.read_csv("pec_traindata.csv", header=None)`

Creating your own DataFrame

```
pd.DataFrame({
    'name': ['Jones', 'Rodriguez', 'Smith'],
    'age': [42, 61, 24]
})
```

	name	age
0	Jones	42
1	Rodriguez	61
2	Smith	24

Creating your own DataFrame

```
pd.DataFrame([
      [78, 1, 57, 'June 12, 2020'],
      [85, 6, 42, 'January 17, 5611'],
      [93, 1, 25, 'March 13, 2047']
],
    columns=['DiaBP',
             'Surgeries',
             'Age',
             'Date'])
```

	DiaBP	Surgeries	Age	Date
0	78	1	57	June 12, 2020
1	85	6	42	January 17, 5611
2	93	1	25	March 13, 2047

```
[7] data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Summary statistics with `.describe()`

- By default, displays only numeric data, but passing `include='all'` gives some information about other data types
- We can use row and column labels to grab specific facts, e.g. `data.describe().loc['mean', 'Pregnancies']`

```
[8] data['BMI'].describe()
```

```
count    768.000000
mean      31.992578
std        7.884160
min        0.000000
25%       27.300000
50%       32.000000
75%       36.600000
max       67.100000
Name: BMI, dtype: float64
```

```
[12] temps = pd.Series([37.2, 38.3, 36.9])
     temps.describe()
```

```
count     3.000000
mean     37.466667
std       0.737111
min      36.900000
25%      37.050000
50%      37.200000
75%      37.750000
max      38.300000
dtype: float64
```

```
[11] names = pd.Series(['James', 'Cole',
                        'Scheele', 'James'])
     names.describe()
```

```
count          4
unique         3
top        James
freq           2
dtype: object
```

Describe works for `pd.Series` as well

```
[37] patients = pd.DataFrame({
       'id': [515, 614, 617, 669, 717],
       'gender': ['m', 'f', 'm', 'm', 'f'],
       'age': [51, 27, 62, 43, 57]
     })
     patients.describe()
```

	id	age
count	5.000000	5.000000
mean	626.400000	48.000000
std	75.291434	13.711309
min	515.000000	27.000000
25%	614.000000	43.000000
50%	617.000000	51.000000
75%	669.000000	57.000000
max	717.000000	62.000000

```
[41] patients.groupby('gender').describe()
```

	id								age							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
gender																
f	2.0	665.500000	72.831998	614.0	639.75	665.5	691.25	717.0	2.0	42.0	21.213203	27.0	34.5	42.0	49.5	57.0
m	3.0	600.333333	78.341134	515.0	566.00	617.0	643.00	669.0	3.0	52.0	9.539392	43.0	47.0	51.0	56.5	62.0

describe with groupby

Groupby takes a column name or list of column names. You can also use it for other analyses.

```
[56] data['Age'].value_counts().head(5)
```

```
22    72
21    63
25    48
24    46
23    38
Name: Age, dtype: int64
```

```
[58] data['Age'].value_counts(bins=5)
```

```
(20.939, 33.0]    474
(33.0, 45.0]      176
(45.0, 57.0]       76
(57.0, 69.0]       39
(69.0, 81.0]        3
Name: Age, dtype: int64
```

```
[63] data.groupby('State')['Age'].value_counts(bins=5)
```

```
State
Connecticut     (20.953, 30.2]    140
                (30.2, 39.4]       47
                (39.4, 48.6]       38
                (48.6, 57.8]       18
                (57.8, 67.0]       14
Massachusetts   (20.95, 30.8]     136
                (30.8, 40.6]       56
                (40.6, 50.4]       40
                (50.4, 60.2]       23
                (60.2, 70.0]        5
New York        (20.939, 33.0]    164
                (33.0, 45.0]       49
                (45.0, 57.0]       19
                (57.0, 69.0]       17
                (69.0, 81.0]        2
Name: Age, dtype: int64
```

```
[6] data.sort_values(by=['BMI'], ascending=False)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
177	0	129	110	46	130	67.1	0.319	26	1
445	0	180	78	63	14	59.4	2.420	25	1
673	3	123	100	35	240	57.3	0.880	22	0
125	1	88	30	42	99	55.0	0.496	26	1
120	0	162	76	56	100	53.2	0.759	25	1
...	...	...	...	...	...	...	...	...	...
81	2	74	0	0	0	0.0	0.102	22	0
371	0	118	64	23	89	0.0	1.731	21	0
522	6	114	0	0	0	0.0	0.189	26	0
49	7	105	0	0	0	0.0	0.305	24	0
706	10	115	0	0	0	0.0	0.261	30	1

768 rows × 9 columns

`sort_values`

Returns a new DataFrame (by default does not change the original) sorted by one or more columns.

Some options:

`ascending`
True or False

`ignore_index`
True or **False**

`na_position`
'first' or '**last**'

```
[6] data.sort_values(by=['BMI'], ascending=False)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
177	0	129	110	46	130	67.1	0.319	26	1
445	0	180	78	63	14	59.4	2.420	25	1
673	3	123	100	35	240	57.3	0.880	22	0
125	1	88	30	42	99	55.0	0.496	26	1
120	0	162	76	56	100	53.2	0.759	25	1
...	...	...	...	...	...	...	...	...	...
81	2	74	0	0	0	0.0	0.102	22	0
371	0	118	64	23	89	0.0	1.731	21	0
522	6	114	0	0	0	0.0	0.189	26	0
49	7	105	0	0	0	0.0	0.305	24	0
706	10	115	0	0	0	0.0	0.261	30	1

768 rows × 9 columns

```
[7] data.head(2)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0

`sort_values`

Returns a new DataFrame (by default does not change the original) sorted by one or more columns.

Some options:

`ascending`
True or False
`ignore_index`
True or **False**
`na_position`
'first' or '**last**'

```
[14] data[['Age', 'BMI']].head(5)
```

	Age	BMI
0	50	33.6
1	31	26.6
2	32	23.3
3	21	28.1
4	33	43.1

```
[17] data['BMI'].head(5)
```

```
0    33.6
1    26.6
2    23.3
3    28.1
4    43.1
Name: BMI, dtype: float64
```

Selecting columns

- We can also load in just a subset of columns when doing a `pd.read_csv` by passing in a `usecols` argument; we'll see an example later.

```
[46] patients
```

	id	gender	age
0	515	m	51
1	614	f	27
2	617	m	62
3	669	m	43
4	717	f	57

```
[47] patients.drop('age', axis=1)
```

	id	gender
0	515	m
1	614	f
2	617	m
3	669	m
4	717	f

Selecting all but a few columns

- The `drop` method can be used as shown to select all but a column or all but a list of columns.
- Calling this with `axis=0` drops rows instead.
- When used as above, it returns a new `DataFrame`; the previous `DataFrame` is unchanged.

Comparing values

```
[18] data['BMI'] > 30
```

```
0       True
1      False
2      False
3      False
4       True
      ...
763     True
764     True
765    False
766     True
767     True
Name: BMI, Length: 768, dtype: bool
```

```
[19] data['is_obese'] = data['BMI'] > 30
```

```
[30] data[['Age', 'BMI', 'is_obese']].head(5)
```

	Age	BMI	is_obese
0	50	33.6	True
1	31	26.6	False
2	32	23.3	False
3	21	28.1	False
4	33	43.1	True

Comparing values

- Build more complicated queries with:
  - and &
  - or |
  - not ~
- Always group these operators with parentheses.

```
[33] data['thirty_something'] = (data['Age'] >= 30) & (data['Age'] < 40)
```

```
[34] data[['Age', 'BMI', 'is_obese', 'thirty_something']].head(5)
```

	Age	BMI	is_obese	thirty_something
0	50	33.6	True	False
1	31	26.6	False	True
2	32	23.3	False	True
3	21	28.1	False	False
4	33	43.1	True	True

```
[53] data['Age']

0      50
1      31
2      32
3      21
4      33
       ..
763    63
764    27
765    30
766    47
767    23
Name: Age, Length: 768, dtype: int64
```

```
[48] data['Age'].isin(range(33, 60))

0       True
1      False
2      False
3      False
4       True
       ...
763    False
764    False
765    False
766     True
767    False
Name: Age, Length: 768, dtype: bool
```

```
[54] ~data['Age'].isin([50, 21])

0      False
1       True
2       True
3      False
4       True
       ...
763     True
764     True
765     True
766     True
767     True
Name: Age, Length: 768, dtype: bool
```

Comparing values: `isin`

Remember you can use ~ to do the logical negation

Comparing values:
`.str.contains`

```
[74] diagnoses
```

	id	conditions
0	6116	Alopecia
1	7457	Alopecia, Alzheimer's
2	25671	Alzheimer's, Crohn's disease

```
[75] diagnoses['conditions'].str.contains('Alopecia')
```

```
0     True
1     True
2    False
Name: conditions, dtype: bool
```

Selecting rows: use []

```
[66] data[data['Age'] > 65]
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
123	5	132	80	0	0	26.8	0.186	69	0
221	2	158	90	0	0	31.6	0.805	66	1
363	4	146	78	0	0	38.5	0.520	67	1
453	2	119	0	0	0	19.6	0.832	72	0
459	9	134	74	33	60	25.9	0.460	81	0
489	8	194	80	0	0	26.1	0.551	67	0
495	6	166	74	0	0	26.6	0.304	66	0
537	0	57	60	0	0	21.7	0.735	67	0
552	6	114	88	0	0	27.8	0.247	66	0
666	4	145	82	18	0	32.5	0.235	70	1
674	8	91	82	0	0	35.6	0.587	68	0
684	5	136	82	0	0	0.0	0.640	69	0
759	6	190	92	0	0	35.5	0.278	66	1

```
[102] patients
```

	id	gender	favorite_letter
0	1712	m	m
1	2343	female	t
2	3734	f	m
3	4753	male	f
4	5672	m	t
5	8234	f	f

```
[103] patients = patients.replace({
          'gender': {'m': 'male', 'f': 'female'}
      })
```

```
[104] patients
```

	id	gender	favorite_letter
0	1712	male	m
1	2343	female	t
2	3734	female	m
3	4753	male	f
4	5672	male	t
5	8234	female	f

Replacing values

- Use `.replace` to replace values.
- Can specify different replacement rules for different columns.
- This returns a new `DataFrame`, so assign result to a variable.

Dates

- `pd.to_datetime` automatically recognizes several date formats.
- Works with a single date or a list or series of dates.
- Can derive information from dates and times like the `.day_name`, `.month`, `.year`, `.day`
- Can subtract two dates.
- If working with dates, tell pandas that they are dates otherwise it will treat them as strings.

```
[87] today = pd.to_datetime('June 8, 2020')
     today.day_name()
```

```
'Monday'
```

```
[88] some_other_day = pd.to_datetime('12 June 2020')
     some_other_day - today
```

```
Timedelta('4 days 00:00:00')
```

```
[89] (some_other_day - today).days
```

```
4
```

```
[90] patients['dob'] = pd.to_datetime(patients['dob'])
```

Categorical variables

```
[98] data['Outcome']
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
[99] data['Outcome'].describe()
```

```
count    768.000000
mean       0.348958
std        0.476951
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: Outcome, dtype: float64
```

```
[100] data['Outcome'] = data['Outcome'].astype('category')
      data['Outcome'].describe()
```

```
count      768
unique       2
top          0
freq       500
Name: Outcome, dtype: int64
```

Concatenating DataFrame objects

```
[25] data1
```

	patient_id	patient_name
0	42	Albertson
1	71	Wu

```
[26] data2
```

	patient_id	patient_name
0	61	Gonzalez
1	16	Cohen

```
[41] pd.concat([data1, data2])
```

	patient_id	patient_name
0	42	Albertson
1	71	Wu
0	61	Gonzalez
1	16	Cohen

Concatenating DataFrame objects

```
[25] data1
```

	patient_id	patient_name
0	42	Albertson
1	71	Wu

```
[26] data2
```

	patient_id	patient_name
0	61	Gonzalez
1	16	Cohen

```
[42] pd.concat([data1, data2],
               ignore_index=True)
```

	patient_id	patient_name
0	42	Albertson
1	71	Wu
2	61	Gonzalez
3	16	Cohen