Pandas

In addition to what's in Anaconda, this lecture will need the following libraries:

In [ ]:
```
!pip install --upgrade pandas-datareader
!pip install --upgrade yfinance
```

```
Defaulting to user installation because normal site-packages is not writea
ble
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting pandas-datareader
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/3f/16/56c9d648b50
3619ebe96f726b5f642b68e299b34162ed2d6faa9d7966b7d/pandas_datareader-0.10.0
-py3-none-any.whl (109 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 109.5/109.5 kB 395.9 kB/s eta
0:00:00a 0:00:01
Collecting lxml
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/51/79/9f7d249850c
9f8357538055359bffa91cc9f0606fcea72b6881fdea9ee39/lxml-5.2.2-cp39-cp39-mac
osx_10_9_universal2.whl (8.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.1/8.1 MB 6.1 MB/s eta 0:0
0:0000:0100:01
Requirement already satisfied: requests>=2.19.0 in /Users/cheney_gao/Libra
ry/Python/3.9/lib/python/site-packages (from pandas-datareader) (2.31.0)
Requirement already satisfied: pandas>=0.23 in /Users/cheney_gao/Library/P
ython/3.9/lib/python/site-packages (from pandas-datareader) (2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /Users/cheney_gao/Library/
Python/3.9/lib/python/site-packages (from pandas>=0.23->pandas-datareader)
(1.24.2)
Requirement already satisfied: tzdata>=2022.7 in /Users/cheney_gao/Librar
y/Python/3.9/lib/python/site-packages (from pandas>=0.23->pandas-datareade
r) (2024.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/cheney_ga
o/Library/Python/3.9/lib/python/site-packages (from pandas>=0.23->pandas-d
atareader) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/cheney_gao/Library/P
ython/3.9/lib/python/site-packages (from pandas>=0.23->pandas-datareader)
(2024.1)
Requirement already satisfied: certifi>=2017.4.17 in /Users/cheney_gao/Lib
rary/Python/3.9/lib/python/site-packages (from requests>=2.19.0->pandas-da
tareader) (2024.2.2)
Requirement already satisfied: idna<4,>=2.5 in /Users/cheney_gao/Library/P
ython/3.9/lib/python/site-packages (from requests>=2.19.0->pandas-dataread
er) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/cheney_gao/Lib
rary/Python/3.9/lib/python/site-packages (from requests>=2.19.0->pandas-da
tareader) (2.2.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/cheney_g
ao/Library/Python/3.9/lib/python/site-packages (from requests>=2.19.0->pan
das-datareader) (3.3.2)
Requirement already satisfied: six>=1.5 in /Library/Developer/CommandLineT
ools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-
packages (from python-dateutil>=2.8.2->pandas>=0.23->pandas-datareader)
(1.15.0)
Installing collected packages: lxml, pandas-datareader
Successfully installed lxml-5.2.2 pandas-datareader-0.10.0

[notice] A new release of pip is available: 23.0.1 -> 24.1.2
[notice] To update, run: /Library/Developer/CommandLineTools/usr/bin/pytho
n3 -m pip install --upgrade pip
Defaulting to user installation because normal site-packages is not writea
ble
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting yfinance
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/db/fc/10b7d339ccf
6725e13408d76fb1e944f512590a949af426503c38d4af712/yfinance-0.2.41-py2.py3-
none-any.whl (73 kB)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.5/73.5 kB 3.1 MB/s eta 0:
00:00
Requirement already satisfied: pytz>=2022.5 in /Users/cheney_gao/Library/P
ython/3.9/lib/python/site-packages (from yfinance) (2024.1)
Requirement already satisfied: numpy>=1.16.5 in /Users/cheney_gao/Library/
Python/3.9/lib/python/site-packages (from yfinance) (1.24.2)
Collecting beautifulsoup4>=4.11.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/b1/fe/e8c672695b3
7eecc5cbf43e1d0638d88d66ba3a44c4d321c796f4e59167f/beautifulsoup4-4.12.3-py
3-none-any.whl (147 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 147.9/147.9 kB 4.9 MB/s eta
0:00:00
Collecting html5lib>=1.1
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/6c/dd/a834df64821
47d48e225a49515aabc28974ad5a4ca3215c18a882565b028/html5lib-1.1-py2.py3-non
e-any.whl (112 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 112.2/112.2 kB 1.7 MB/s eta
0:00:00a 0:00:01
Requirement already satisfied: lxml>=4.9.1 in /Users/cheney_gao/Library/Py
thon/3.9/lib/python/site-packages (from yfinance) (5.2.2)
Collecting frozendict>=2.3.4
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/02/e5/b9794781c97
2db003ec136d72f1f7e1aa57756df094554160cb185616607/frozendict-2.4.4-cp39-cp
39-macosx_11_0_arm64.whl (37 kB)
Requirement already satisfied: platformdirs>=2.0.0 in /Users/cheney_gao/Li
brary/Python/3.9/lib/python/site-packages (from yfinance) (3.9.1)
Requirement already satisfied: requests>=2.31 in /Users/cheney_gao/Librar
y/Python/3.9/lib/python/site-packages (from yfinance) (2.31.0)
Collecting multitasking>=0.0.7
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/3e/8a/bb3160e76e8
44db9e69a413f055818969c8acade64e1a9ac5ce9dfdcf6c1/multitasking-0.0.11-py3-
none-any.whl (8.5 kB)
Requirement already satisfied: pandas>=1.3.0 in /Users/cheney_gao/Library/
Python/3.9/lib/python/site-packages (from yfinance) (2.2.2)
Collecting peewee>=3.16.2
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/bd/be/e9c886b4601
a19f4c34a1b75c5fe8b98a2115dd964251a76b24c977c369d/peewee-3.17.6.tar.gz (3.
0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 9.8 MB/s eta 0:0
0:00ta 0:00:01
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting soupsieve>1.2
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/4c/f3/038b302fdfb
e3be7da016777069f26ceefe11a681055ea1f7817546508e3/soupsieve-2.5-py3-none-a
ny.whl (36 kB)
Collecting webencodings
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/f4/24/2a3e3df7323
93fed8b3ebf2ec078f05546de641fe1b667ee316ec1dcf3b7/webencodings-0.5.1-py2.p
y3-none-any.whl (11 kB)
Requirement already satisfied: six>=1.9 in /Library/Developer/CommandLineT
ools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-
packages (from html5lib>=1.1->yfinance) (1.15.0)
Requirement already satisfied: tzdata>=2022.7 in /Users/cheney_gao/Librar
y/Python/3.9/lib/python/site-packages (from pandas>=1.3.0->yfinance) (202
4.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/cheney_ga
o/Library/Python/3.9/lib/python/site-packages (from pandas>=1.3.0->yfinanc
e) (2.8.2)
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/cheney_g
ao/Library/Python/3.9/lib/python/site-packages (from requests>=2.31->yfina
nce) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/cheney_gao/Lib
rary/Python/3.9/lib/python/site-packages (from requests>=2.31->yfinance)
(2.2.1)
Requirement already satisfied: idna<4,>=2.5 in /Users/cheney_gao/Library/P
ython/3.9/lib/python/site-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: certifi>=2017.4.17 in /Users/cheney_gao/Lib
rary/Python/3.9/lib/python/site-packages (from requests>=2.31->yfinance)
(2024.2.2)
Building wheels for collected packages: peewee
  Building wheel for peewee (pyproject.toml) ... done
  Created wheel for peewee: filename=peewee-3.17.6-cp39-cp39-macosx_10_9_u
niversal2.whl size=390128 sha256=fe5e3e361146fd30f9a9e0fe5a668407b4543055b
3ec06ca04ae468f53e427a5
  Stored in directory: /Users/cheney_gao/Library/Caches/pip/wheels/b4/fe/b
7/ddfae4d159c4b4b86890ab3c691bae86456b186e4111ef6d46
Successfully built peewee
Installing collected packages: webencodings, peewee, multitasking, soupsie
ve, html5lib, frozendict, beautifulsoup4, yfinance
Successfully installed beautifulsoup4-4.12.3 frozendict-2.4.4 html5lib-1.1
multitasking-0.0.11 peewee-3.17.6 soupsieve-2.5 webencodings-0.5.1 yfinanc
e-0.2.41

[notice] A new release of pip is available: 23.0.1 -> 24.1.2
[notice] To update, run: /Library/Developer/CommandLineTools/usr/bin/pytho
n3 -m pip install --upgrade pip
```

Overview

Pandas is a package of fast, efficient data analysis tools for Python.

Its popularity has surged in recent years, coincident with the rise of fields such as data science and machine learning.

Just as NumPy provides the basic array data type plus core array operations, pandas

1. defines fundamental structures for working with data and
2. endows them with methods that facilitate operations such as

- reading in data
- adjusting indices
- working with dates and time series
- sorting, grouping, re-ordering and general data munging [1]
- dealing with missing values, etc., etc.

More sophisticated statistical functionality is left to other packages, such as statsmodels and scikit-learn, which are built on top of pandas.

This lecture will provide a basic introduction to pandas.

Throughout the lecture, we will assume that the following imports have taken place

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
```

```
/Users/cheney_gao/Library/Python/3.9/lib/python/site-packages/urllib3/__in
it__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, cu
rrently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://g
ithub.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Two important data types defined by pandas are `Series` and `DataFrame`.

You can think of a `Series` as a "column" of data, such as a collection of observations on a single variable.

A `DataFrame` is a two-dimensional object for storing related columns of data.

Series

Let's start with Series.

We begin by creating a series of four random observations

In [ ]:
```
s = pd.Series(np.random.randn(4), name='daily returns')
s
```

Out[ ]:
```
0    1.052282
1    0.361496
2   -0.648000
3   -0.585500
Name: daily returns, dtype: float64
```

Here you can imagine the indices `0, 1, 2, 3` as indexing four listed companies, and the values being daily returns on their shares.

Pandas `Series` are built on top of NumPy arrays and support many similar operations

In [ ]:
```
s * 100
```

Out[ ]:
```
0    105.228216
1     36.149575
2    -64.799964
3    -58.550002
Name: daily returns, dtype: float64
```

In [ ]:
```
np.abs(s)
```

Out[ ]:
```
0    1.052282
1    0.361496
2    0.648000
3    0.585500
Name: daily returns, dtype: float64
```

But `Series` provide more than NumPy arrays.

Not only do they have some additional (statistically oriented) methods

In [ ]:
```
s.describe()
```

Out[ ]:
```
count    4.000000
mean     0.045070
std      0.814978
min     -0.648000
25%     -0.601125
50%     -0.112002
75%      0.534192
max      1.052282
Name: daily returns, dtype: float64
```

But their indices are more flexible

In [ ]:
```
s.index = ['AMZN', 'AAPL', 'MSFT', 'GOOG']
s
```

Out[ ]:
```
AMZN    1.052282
AAPL    0.361496
MSFT   -0.648000
GOOG   -0.585500
Name: daily returns, dtype: float64
```

Viewed in this way, `Series` are like fast, efficient Python dictionaries (with the restriction that the items in the dictionary all have the same type—in this case, floats).

In fact, you can use much of the same syntax as Python dictionaries

In [ ]:
```
s['AMZN']
```

Out[ ]:
```
1.0522821587952125
```

In [ ]:
```
s['AMZN'] = 0
s
```

Out[ ]:
```
AMZN    0.000000
AAPL    0.361496
MSFT   -0.648000
GOOG   -0.585500
Name: daily returns, dtype: float64
```

In [ ]:
```
'AAPL' in s
```

Out[ ]:
```
True
```

DataFrames

While a `Series` is a single column of data, a `DataFrame` is several columns, one for each variable.

In essence, a `DataFrame` in pandas is analogous to a (highly optimized) Excel spreadsheet.

Thus, it is a powerful tool for representing and analyzing data that are naturally organized into rows and columns, often with descriptive indexes for individual rows and individual columns.

Let's look at an example that reads data from the CSV file `pandas/data/test_pwt.csv`, which is taken from the Penn World Tables.

The dataset contains the following indicators

Variable Name	Description
POP	Population (in thousands)
XRAT	Exchange Rate to US Dollar
tcgdp	Total PPP Converted GDP (in million international dollar)
cc	Consumption Share of PPP Converted GDP Per Capita (%)
cg	Government Consumption Share of PPP Converted GDP Per Capita (%)

We'll read this in from a URL using the `pandas` function `read_csv`.

In [ ]:
```
df = pd.read_csv('https://raw.githubusercontent.com/QuantEcon/lecture-pyt
type(df)
```

Out[ ]:
```
pandas.core.frame.DataFrame
```

Here's the content of `test_pwt.csv`

In [ ]:
```
df
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.999500	2.950722e+05	75.716805	5
1	Australia	AUS	2000	19053.186	1.724830	5.418047e+05	67.759026	6
2	India	IND	2000	1006300.297	44.941600	1.728144e+06	64.575551	14
3	Israel	ISR	2000	6114.570	4.077330	1.292539e+05	64.436451	10
4	Malawi	MWI	2000	11801.505	59.543808	5.026222e+03	74.707624	11
5	South Africa	ZAF	2000	45064.098	6.939830	2.272424e+05	72.718710	5
6	United States	USA	2000	282171.957	1.000000	9.898700e+06	72.347054	6
7	Uruguay	URY	2000	3219.793	12.099592	2.525596e+04	78.978740	5

Select Data by Position

In practice, one thing that we do all the time is to find, select and work with a subset of the data of our interests.

We can select particular rows using standard Python array slicing notation

In [ ]:
```
df[2:5]
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
2	India	IND	2000	1006300.297	44.941600	1.728144e+06	64.575551	14.
3	Israel	ISR	2000	6114.570	4.077330	1.292539e+05	64.436451	10.
4	Malawi	MWI	2000	11801.505	59.543808	5.026222e+03	74.707624	11.

To select columns, we can pass a list containing the names of the desired columns represented as strings

In [ ]:
```
df[['country', 'tcgdp']]
```

Out[ ]:

	country	tcgdp
0	Argentina	2.950722e+05
1	Australia	5.418047e+05
2	India	1.728144e+06
3	Israel	1.292539e+05
4	Malawi	5.026222e+03
5	South Africa	2.272424e+05
6	United States	9.898700e+06
7	Uruguay	2.525596e+04

To select both rows and columns using integers, the `iloc` attribute should be used with the format `.iloc[rows, columns]`.

In [ ]:
```
df.iloc[2:5, 0:4]
```

Out[ ]:

	country	country isocode	year	POP
2	India	IND	2000	1006300.297
3	Israel	ISR	2000	6114.570
4	Malawi	MWI	2000	11801.505

To select rows and columns using a mixture of integers and labels, the `loc` attribute can be used in a similar way

In [ ]:
```
df.loc[df.index[2:5], ['country', 'tcgdp']]
```

Out[ ]:

	country	tcgdp
2	India	1.728144e+06
3	Israel	1.292539e+05
4	Malawi	5.026222e+03

Select Data by Conditions

Instead of indexing rows and columns using integers and names, we can also obtain a sub-dataframe of our interests that satisfies certain (potentially complicated) conditions.

This section demonstrates various ways to do that.

The most straightforward way is with the `[]` operator.

In [ ]:
```
df[df.POP >= 20000]
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.99950	2.950722e+05	75.716805	5.5
2	India	IND	2000	1006300.297	44.94160	1.728144e+06	64.575551	14.(
5	South Africa	ZAF	2000	45064.098	6.93983	2.272424e+05	72.718710	5.7
6	United States	USA	2000	282171.957	1.00000	9.898700e+06	72.347054	6.(

To understand what is going on here, notice that `df.POP >= 20000` returns a series of boolean values.

In [ ]:
```
df.POP >= 20000
```

Out[ ]:
```
0     True
1    False
2     True
3    False
4    False
5     True
6     True
7    False
Name: POP, dtype: bool
```

In this case, `df[___]` takes a series of boolean values and only returns rows with the `True` values.

Take one more example,

In [ ]:
```
df[(df.country.isin(['Argentina', 'India', 'South Africa'])) & (df.POP >
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
2	India	IND	2000	1006300.297	44.94160	1.728144e+06	64.575551	14.07
5	South Africa	ZAF	2000	45064.098	6.93983	2.272424e+05	72.718710	5.72

However, there is another way of doing the same thing, which can be slightly faster for large dataframes, with more natural syntax.

In [ ]:
```
# the above is equivalent to
df.query("POP >= 20000")
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.99950	2.950722e+05	75.716805	5.5
2	India	IND	2000	1006300.297	44.94160	1.728144e+06	64.575551	14.(
5	South Africa	ZAF	2000	45064.098	6.93983	2.272424e+05	72.718710	5.7
6	United States	USA	2000	282171.957	1.00000	9.898700e+06	72.347054	6.(

In [ ]:
```
df.query("country in ['Argentina', 'India', 'South Africa'] and POP > 400
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
2	India	IND	2000	1006300.297	44.94160	1.728144e+06	64.575551	14.07
5	South Africa	ZAF	2000	45064.098	6.93983	2.272424e+05	72.718710	5.72

We can also allow arithmetic operations between different columns.

In [ ]:
```
df[(df.cc + df.cg >= 80) & (df.POP <= 20000)]
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
4	Malawi	MWI	2000	11801.505	59.543808	5026.221784	74.707624	11.658
7	Uruguay	URY	2000	3219.793	12.099592	25255.961693	78.978740	5.108

In [ ]:
```
# the above is equivalent to
df.query("cc + cg >= 80 & POP <= 20000")
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
4	Malawi	MWI	2000	11801.505	59.543808	5026.221784	74.707624	11.658
7	Uruguay	URY	2000	3219.793	12.099592	25255.961693	78.978740	5.108

For example, we can use the conditioning to select the country with the largest household consumption - gdp share `cc`.

In [ ]:
```
df.loc[df.cc == max(df.cc)]
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	cg
7	Uruguay	URY	2000	3219.793	12.099592	25255.961693	78.97874	5.108068

When we only want to look at certain columns of a selected sub-dataframe, we can use the above conditions with the `.loc[__ , __]` command.

The first argument takes the condition, while the second argument takes a list of columns we want to return.

In [ ]:
```
df.loc[(df.cc + df.cg >= 80) & (df.POP <= 20000), ['country', 'year', 'PO
```

Out[ ]:

	country	year	POP
4	Malawi	2000	11801.505
7	Uruguay	2000	3219.793

Application: Subsetting Dataframe

Real-world datasets can be enormous.

It is sometimes desirable to work with a subset of data to enhance computational efficiency and reduce redundancy.

Let's imagine that we're only interested in the population (`POP`) and total GDP (`tcgdp`).

One way to strip the data frame `df` down to only these variables is to overwrite the dataframe using the selection method described above

In [ ]:
```
df_subset = df[['country', 'POP', 'tcgdp']]
df_subset
```

Out[ ]:

	country	POP	tcgdp
0	Argentina	37335.653	2.950722e+05
1	Australia	19053.186	5.418047e+05
2	India	1006300.297	1.728144e+06
3	Israel	6114.570	1.292539e+05
4	Malawi	11801.505	5.026222e+03
5	South Africa	45064.098	2.272424e+05
6	United States	282171.957	9.898700e+06
7	Uruguay	3219.793	2.525596e+04

We can then save the smaller dataset for further analysis.

```
df_subset.to_csv('pwt_subset.csv', index=False)
```

Apply Method

Another widely used Pandas method is `df.apply()`.

It applies a function to each row/column and returns a series.

This function can be some built-in functions like the `max` function, a `lambda` function, or a user-defined function.

Here is an example using the `max` function

In [ ]:
```
df[['year', 'POP', 'XRAT', 'tcgdp', 'cc', 'cg']].apply(max)
```

Out[ ]:
```
year     2.000000e+03
POP      1.006300e+06
XRAT     5.954381e+01
tcgdp    9.898700e+06
cc       7.897874e+01
cg       1.407221e+01
dtype: float64
```

This line of code applies the `max` function to all selected columns.

`lambda` function is often used with `df.apply()` method

A trivial example is to return itself for each row in the dataframe

In [ ]:
```
df.apply(lambda row: row, axis=1)
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.999500	2.950722e+05	75.716805	5
1	Australia	AUS	2000	19053.186	1.724830	5.418047e+05	67.759026	6
2	India	IND	2000	1006300.297	44.941600	1.728144e+06	64.575551	14
3	Israel	ISR	2000	6114.570	4.077330	1.292539e+05	64.436451	10
4	Malawi	MWI	2000	11801.505	59.543808	5.026222e+03	74.707624	11
5	South Africa	ZAF	2000	45064.098	6.939830	2.272424e+05	72.718710	5
6	United States	USA	2000	282171.957	1.000000	9.898700e+06	72.347054	6
7	Uruguay	URY	2000	3219.793	12.099592	2.525596e+04	78.978740	5

> **Note**
>
> For the `.apply()` method

- axis = 0 – apply function to each column (variables)
- axis = 1 – apply function to each row (observations)
- axis = 0 is the default parameter

We can use it together with `.loc[]` to do some more advanced selection.

In [ ]:
```
complexCondition = df.apply(
    lambda row: row.POP > 40000 if row.country in ['Argentina', 'India',
    axis=1), ['country', 'year', 'POP', 'XRAT', 'tcgdp']
```

`df.apply()` here returns a series of boolean values rows that satisfies the condition specified in the if-else statement.

In addition, it also defines a subset of variables of interest.

In [ ]:
```
complexCondition
```

Out[ ]:
```
(0    False
 1     True
 2     True
 3     True
 4     True
 5     True
 6    False
 7     True
 dtype: bool,
 ['country', 'year', 'POP', 'XRAT', 'tcgdp'])
```

When we apply this condition to the dataframe, the result will be

In [ ]:
```
df.loc[complexCondition]
```

Out[ ]:

	country	year	POP	XRAT	tcgdp
1	Australia	2000	19053.186	1.724830	5.418047e+05
2	India	2000	1006300.297	44.941600	1.728144e+06
3	Israel	2000	6114.570	4.077330	1.292539e+05
4	Malawi	2000	11801.505	59.543808	5.026222e+03
5	South Africa	2000	45064.098	6.939830	2.272424e+05
7	Uruguay	2000	3219.793	12.099592	2.525596e+04

Make Changes in DataFrames

The ability to make changes in dataframes is important to generate a clean dataset for future analysis.

1. We can use `df.where()` conveniently to "keep" the rows we have selected and replace the rest rows with any other values

In [ ]:
```
df.where(df.POP >= 20000, False)
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.9995	295072.21869	75.716805	5.5
1	False	False	False	False	False	False	False	
2	India	IND	2000	1006300.297	44.9416	1728144.3748	64.575551	14.0
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
5	South Africa	ZAF	2000	45064.098	6.93983	227242.36949	72.71871	5.7
6	United States	USA	2000	282171.957	1.0	9898700.0	72.347054	6.0
7	False	False	False	False	False	False	False	

2. We can simply use `.loc[]` to specify the column that we want to modify, and assign values

In [ ]:
```
df.loc[df.cg == max(df.cg), 'cg'] = np.nan
df
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.999500	2.950722e+05	75.716805	5
1	Australia	AUS	2000	19053.186	1.724830	5.418047e+05	67.759026	6
2	India	IND	2000	1006300.297	44.941600	1.728144e+06	64.575551	
3	Israel	ISR	2000	6114.570	4.077330	1.292539e+05	64.436451	10
4	Malawi	MWI	2000	11801.505	59.543808	5.026222e+03	74.707624	11
5	South Africa	ZAF	2000	45064.098	6.939830	2.272424e+05	72.718710	5
6	United States	USA	2000	282171.957	1.000000	9.898700e+06	72.347054	6
7	Uruguay	URY	2000	3219.793	12.099592	2.525596e+04	78.978740	5

3. We can use the `.apply()` method to modify *rows/columns as a whole*

In [ ]:
```
def update_row(row):
    # modify POP
    row.POP = np.nan if row.POP<= 10000 else row.POP

    # modify XRAT
    row.XRAT = row.XRAT / 10
    return row

df.apply(update_row, axis=1)
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	
0	Argentina	ARG	2000	37335.653	0.099950	2.950722e+05	75.716805	5.
1	Australia	AUS	2000	19053.186	0.172483	5.418047e+05	67.759026	6.
2	India	IND	2000	1006300.297	4.494160	1.728144e+06	64.575551	
3	Israel	ISR	2000	NaN	0.407733	1.292539e+05	64.436451	10.
4	Malawi	MWI	2000	11801.505	5.954381	5.026222e+03	74.707624	11.
5	South Africa	ZAF	2000	45064.098	0.693983	2.272424e+05	72.718710	5.
6	United States	USA	2000	282171.957	0.100000	9.898700e+06	72.347054	6.
7	Uruguay	URY	2000	NaN	1.209959	2.525596e+04	78.978740	5.

4. We can use the `.applymap()` method to modify all *individual entries* in the dataframe altogether.

In [ ]:
```
# Round all decimal numbers to 2 decimal places
df.applymap(lambda x : round(x,2) if type(x)!=str else x)
```

```
/var/folders/v4/8yvlmdh17719kc18c21wy27r0000gn/T/ipykernel_80483/2333807478.py:2: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  df.applymap(lambda x : round(x,2) if type(x)!=str else x)
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc	cg
0	Argentina	ARG	2000	37335.65	1.00	295072.22	75.72	5.58
1	Australia	AUS	2000	19053.19	1.72	541804.65	67.76	6.72
2	India	IND	2000	1006300.30	44.94	1728144.37	64.58	NaN
3	Israel	ISR	2000	6114.57	4.08	129253.89	64.44	10.27
4	Malawi	MWI	2000	11801.50	59.54	5026.22	74.71	11.66
5	South Africa	ZAF	2000	45064.10	6.94	227242.37	72.72	5.73
6	United States	USA	2000	282171.96	1.00	9898700.00	72.35	6.03
7	Uruguay	URY	2000	3219.79	12.10	25255.96	78.98	5.11

Application: Missing Value Imputation

Replacing missing values is an important step in data munging.

Let's randomly insert some NaN values

In [ ]:
```
for idx in list(zip([0, 3, 5, 6], [3, 4, 6, 2])):
    df.iloc[idx] = np.nan

df
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc
0	Argentina	ARG	2000.0	NaN	0.999500	2.950722e+05	75.716805
1	Australia	AUS	2000.0	19053.186	1.724830	5.418047e+05	67.759026
2	India	IND	2000.0	1006300.297	44.941600	1.728144e+06	64.575551
3	Israel	ISR	2000.0	6114.570	NaN	1.292539e+05	64.436451
4	Malawi	MWI	2000.0	11801.505	59.543808	5.026222e+03	74.707624
5	South Africa	ZAF	2000.0	45064.098	6.939830	2.272424e+05	NaN
6	United States	USA	NaN	282171.957	1.000000	9.898700e+06	72.347054
7	Uruguay	URY	2000.0	3219.793	12.099592	2.525596e+04	78.978740

The `zip()` function here creates pairs of values from the two lists (i.e. [0,3], [3,4] ...)

We can use the `.applymap()` method again to replace all missing values with 0

In [ ]:
```
# replace all NaN values by 0
def replace_nan(x):
    if type(x)!=str:
        return  0 if np.isnan(x) else x
    else:
        return x

df.applymap(replace_nan)
```

```
/var/folders/v4/8yvlmdh17719kc18c21wy27r0000gn/T/ipykernel_80483/966822939.py:8: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  df.applymap(replace_nan)
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc
0	Argentina	ARG	2000.0	0.000	0.999500	2.950722e+05	75.716805
1	Australia	AUS	2000.0	19053.186	1.724830	5.418047e+05	67.759026
2	India	IND	2000.0	1006300.297	44.941600	1.728144e+06	64.575551
3	Israel	ISR	2000.0	6114.570	0.000000	1.292539e+05	64.436451
4	Malawi	MWI	2000.0	11801.505	59.543808	5.026222e+03	74.707624
5	South Africa	ZAF	2000.0	45064.098	6.939830	2.272424e+05	0.000000
6	United States	USA	0.0	282171.957	1.000000	9.898700e+06	72.347054
7	Uruguay	URY	2000.0	3219.793	12.099592	2.525596e+04	78.978740

Pandas also provides us with convenient methods to replace missing values.

For example, single imputation using variable means can be easily done in pandas

In [ ]:
```
df = df.fillna(df.iloc[:,2:8].mean())
df
```

Out[ ]:

	country	country isocode	year	POP	XRAT	tcgdp	cc
0	Argentina	ARG	2000.0	1.962465e+05	0.999500	2.950722e+05	75.716805
1	Australia	AUS	2000.0	1.905319e+04	1.724830	5.418047e+05	67.759026
2	India	IND	2000.0	1.006300e+06	44.941600	1.728144e+06	64.575551
3	Israel	ISR	2000.0	6.114570e+03	18.178451	1.292539e+05	64.436451
4	Malawi	MWI	2000.0	1.180150e+04	59.543808	5.026222e+03	74.707624
5	South Africa	ZAF	2000.0	4.506410e+04	6.939830	2.272424e+05	71.217322
6	United States	USA	2000.0	2.821720e+05	1.000000	9.898700e+06	72.347054
7	Uruguay	URY	2000.0	3.219793e+03	12.099592	2.525596e+04	78.978740

Missing value imputation is a big area in data science involving various machine learning techniques.

There are also more advanced tools in python to impute missing values.

Standardization and Visualization

Let's imagine that we're only interested in the population (`POP`) and total GDP (`tcgdp`).

One way to strip the data frame `df` down to only these variables is to overwrite the dataframe using the selection method described above

In [ ]:
```
df = df[['country', 'POP', 'tcgdp']]
df
```

Out[ ]:

	country	POP	tcgdp
0	Argentina	1.962465e+05	2.950722e+05
1	Australia	1.905319e+04	5.418047e+05
2	India	1.006300e+06	1.728144e+06
3	Israel	6.114570e+03	1.292539e+05
4	Malawi	1.180150e+04	5.026222e+03
5	South Africa	4.506410e+04	2.272424e+05
6	United States	2.821720e+05	9.898700e+06
7	Uruguay	3.219793e+03	2.525596e+04

Here the index `0, 1,..., 7` is redundant because we can use the country names as an index.

To do this, we set the index to be the `country` variable in the dataframe

In [ ]:
```
df = df.set_index('country')
df
```

Out[ ]:

country	POP	tcgdp
Argentina	1.962465e+05	2.950722e+05
Australia	1.905319e+04	5.418047e+05
India	1.006300e+06	1.728144e+06
Israel	6.114570e+03	1.292539e+05
Malawi	1.180150e+04	5.026222e+03
South Africa	4.506410e+04	2.272424e+05
United States	2.821720e+05	9.898700e+06
Uruguay	3.219793e+03	2.525596e+04

Let's give the columns slightly better names

In [ ]:
```
df.columns = 'population', 'total GDP'
df
```

Out[ ]:

country	population	total GDP
Argentina	1.962465e+05	2.950722e+05
Australia	1.905319e+04	5.418047e+05
India	1.006300e+06	1.728144e+06
Israel	6.114570e+03	1.292539e+05
Malawi	1.180150e+04	5.026222e+03
South Africa	4.506410e+04	2.272424e+05
United States	2.821720e+05	9.898700e+06
Uruguay	3.219793e+03	2.525596e+04

The `population` variable is in thousands, let's revert to single units

In [ ]:
```
df['population'] = df['population'] * 1e3
df
```

Out[ ]:

country	population	total GDP
Argentina	1.962465e+08	2.950722e+05
Australia	1.905319e+07	5.418047e+05
India	1.006300e+09	1.728144e+06
Israel	6.114570e+06	1.292539e+05
Malawi	1.180150e+07	5.026222e+03
South Africa	4.506410e+07	2.272424e+05
United States	2.821720e+08	9.898700e+06
Uruguay	3.219793e+06	2.525596e+04

Next, we're going to add a column showing real GDP per capita, multiplying by 1,000,000 as we go because total GDP is in millions

In [ ]:
```
df['GDP percap'] = df['total GDP'] * 1e6 / df['population']
df
```

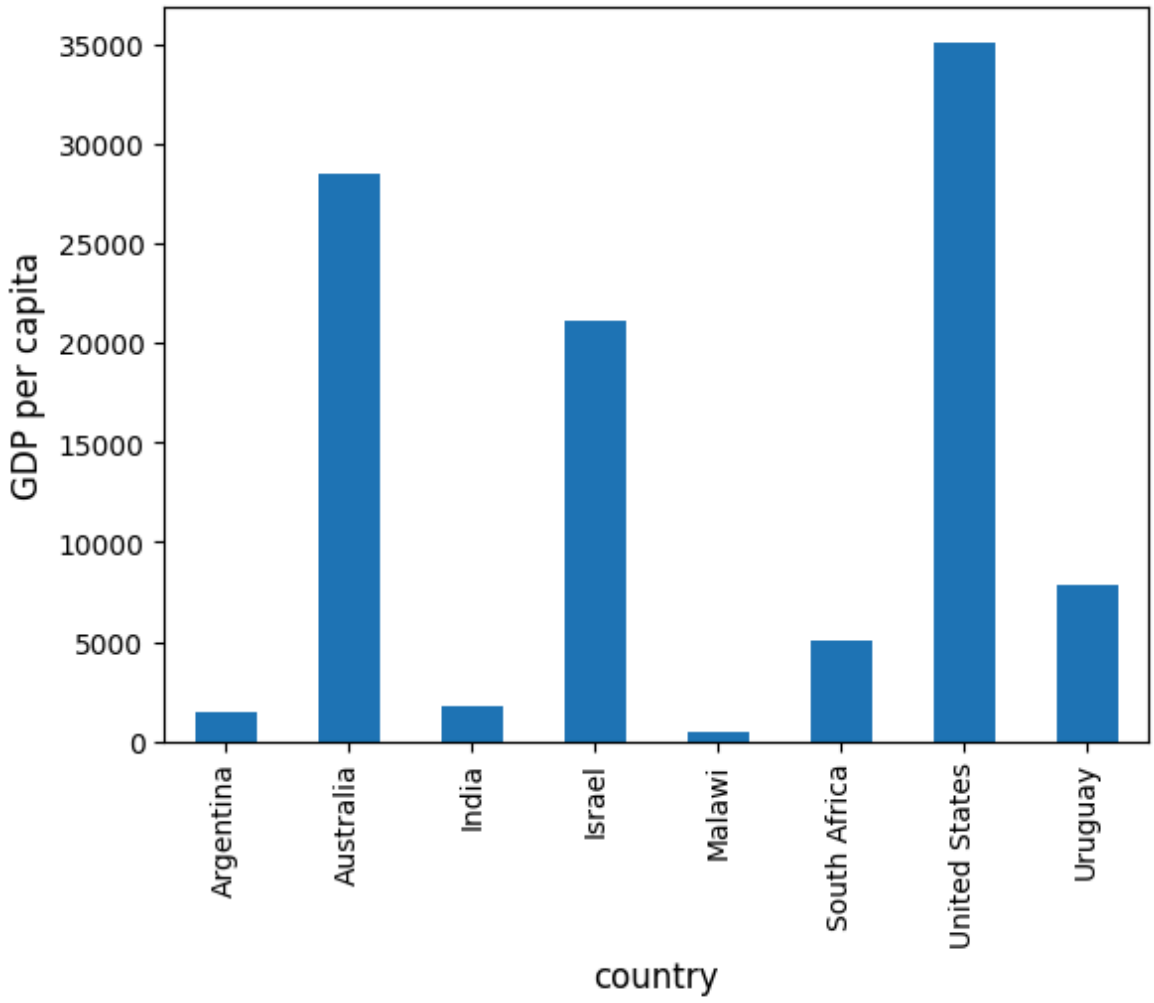
Out[ ]:

country	population	total GDP	GDP percap
Argentina	1.962465e+08	2.950722e+05	1503.579625
Australia	1.905319e+07	5.418047e+05	28436.433261
India	1.006300e+09	1.728144e+06	1717.324719
Israel	6.114570e+06	1.292539e+05	21138.672749
Malawi	1.180150e+07	5.026222e+03	425.896679
South Africa	4.506410e+07	2.272424e+05	5042.647686
United States	2.821720e+08	9.898700e+06	35080.381854
Uruguay	3.219793e+06	2.525596e+04	7843.970620

One of the nice things about pandas `DataFrame` and `Series` objects is that they have methods for plotting and visualization that work through Matplotlib.

For example, we can easily generate a bar plot of GDP per capita

In [ ]:
```
ax = df['GDP percap'].plot(kind='bar')
ax.set_xlabel('country', fontsize=12)
ax.set_ylabel('GDP per capita', fontsize=12)
plt.show()
```



At the moment the data frame is ordered alphabetically on the countries—let's change it to GDP per capita

In [ ]:
```
df = df.sort_values(by='GDP percap', ascending=False)
df
```

Out[ ]:

country	population	total GDP	GDP percap
United States	2.821720e+08	9.898700e+06	35080.381854
Australia	1.905319e+07	5.418047e+05	28436.433261
Israel	6.114570e+06	1.292539e+05	21138.672749
Uruguay	3.219793e+06	2.525596e+04	7843.970620
South Africa	4.506410e+07	2.272424e+05	5042.647686
India	1.006300e+09	1.728144e+06	1717.324719
Argentina	1.962465e+08	2.950722e+05	1503.579625
Malawi	1.180150e+07	5.026222e+03	425.896679

Plotting as before now yields

In [ ]:
```
ax = df['GDP percap'].plot(kind='bar')
ax.set_xlabel('country', fontsize=12)
ax.set_ylabel('GDP per capita', fontsize=12)
plt.show()
```