

# Python Packages

# Outline

- How to import packages
- Numpy
- Pandas
- Matplotlib

- Use `import` to load a library module into a program's memory.
- Then refer to things from the module as `module_name.thing_name`.
  - Python uses `.` to mean “part of”.
- Use `from ... import ...` to load only specific items from a library module.
- Then refer to them directly without library name as prefix.
- Use `import ... as ...` to give a library a short *alias* while importing it.
- Then refer to items in the library using that shortened name.

```
import math
print('pi is', math.pi)
print('cos(pi) is',
math.cos(math.pi))
```

```
from math import cos, pi
print('cos(pi) is', cos(pi))
```

```
import math as m
print('cos(pi) is', m.cos(m.pi))
```

# Numpy

```
import numpy as np
```

<https://numpy.org/doc/stable/reference/index.html>

# Numpy

- `np.array([Arrays Arrays Arrays])`
- `>>> a = np.array([1, 2, 3, 4, 5, 6])`
- `>>> a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])`

Command

`np.array([1,2,3])`



NumPy Array

1
2
3



# Pandas

```
import pandas as pd
```



<https://pandas.pydata.org/docs/>



# Pandas

- Great for reading data
- Data manipulation with integrated indexing.

diamonds mtcars iris										
Show	10	entries	Search:							
	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.52	Ideal	D	VS2	61.4	56	1664	5.16	5.19	3.18
2	0.5	Very Good	F	SI1	62.3	60	1250	5.07	5.11	3.17
3	0.61	Ideal	G	VVS2	61.6	54	2242	5.45	5.49	3.37
4	0.36	Premium	G	VS2	62.5	58	756	4.55	4.51	2.83
5	0.7	Very Good	E	VS2	63.5	54	2889	5.62	5.66	3.58
6	0.56	Ideal	F	VS1	61.7	56	2016	5.32	5.28	3.27
7	1.19	Premium	E	I1	60.2	61	3572	6.91	6.87	4.15
8	0.52	Ideal	F	IF	60.6	57	2575	5.21	5.22	3.16
9	0.38	Ideal	E	IF	62.7	55	1433	4.61	4.67	2.91
10	0.51	Ideal	E	VS2	62.1	54	1608	5.13	5.15	3.19
Showing 1 to 10 of 1,000 entries										
Previous 1 2 3 4 5 ... 100 Next										

Type all of the table???????







# Pandas- DataFrames

- A [DataFrame](#) is a collection of [Series](#)
- The DataFrame is the way Pandas represents a table
- Series is the data-structure Pandas use to represent a column.
- Built on top of the [Numpy](#) library
- Access individual records of the table
- Handling of missing values
- Databases operations between DataFrames.



# Pandas-Read

```
import pandas as pd
data = pd.read_csv('data/gapminder_gdp_oceania.csv')
print(data)

data.info() data = pd.read_csv('data/gapminder_gdp_oceania.csv',
index_col='country')

print(data.columns)
print(data.T)
print(data.describe())
```



# Pandas-Practice

- Read the data in [gapminder\\_gdp\\_americas.csv](#) into a variable called `americas` and display its summary statistics.
- What method call will display the first three rows of this data?
- What method call will display the last three columns of this data?



# Pandas-Practice

- Read the data in `gapminder_gdp_americas.csv` into a variable called `americas` and display its summary statistics.
  - `americas = pd.read_csv('data/gapminder_gdp_americas.csv', index_col='country')`
  - `americas.describe()`
- What method call will display the first three **rows** of this data?
  - `americas.head(n=3)`
- What method call will display the last three **columns** of this data?
  - `americas_flipped = americas.T`
  - `americas_flipped.tail(n=3)`
  - `americas_flipped.tail(n=3).T`



# Pandas- DataFrame

- Use `DataFrame.iloc[..., ...]` to select values by their (entry) position
- `data = pd.read_csv('data/gapminder_gdp_europe.csv', index_col='country')`
- `print(data.iloc[0, 0])`
- `print(data.loc["Albania", "gdpPercap_1952"])`
- 
-



# Pandas-DataFrame

- Use `:` on its own to mean all columns or all rows.
- **`print(data.loc["Albania", :])`**
- **`print(data.loc[:, "gdpPercap_1952"])`**



# Pandas- DataFrame

- Select multiple columns or rows using `DataFrame.loc` and a named slice.
- **`print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972'])`**



# Pandas-DataFrame

*# Use a subset of data to keep output readable.*

```
subset = data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972']  
print('Subset of data:\n', subset)
```

*# Which values were greater than 10000 ?*

```
print('\nWhere are values large?\n', subset > 10000)
```

```
mask = subset > 10000  
print(subset[mask])
```

```
print(subset[subset > 10000].describe())
```





# Pandas-DataFrame Practice

1. Write an expression to find the Per Capita GDP of Serbia in 2007.
2. Do the two statements below produce the same output?

```
print(df.iloc[0:2, 0:2])
```

```
print(df.loc['Albania':'Belgium', 'gdpPercap_1952':'gdpPercap_1962'])
```

3. GDP per capita for all countries in 1982.
4. GDP per capita for Denmark for all years.
5. GDP per capita for all countries for years *after* 1985.
6. GDP per capita for each country in 2007 as a multiple of GDP per capita for that country in 1952.



# Pandas-DataFrame Practice

1. Write an expression to find the Per Capita GDP of Serbia in 2007.

- `print(df.loc['Serbia', 'gdpPercap_2007'])`

2. Do the two statements below produce the same output?

```
print(df.iloc[0:2, 0:2])
```

```
print(df.loc['Albania':'Belgium', 'gdpPercap_1952':'gdpPercap_1962'])
```

- No, they do not produce the same output!
- numerical slice, 0:2, *omits* the final index (i.e. index 2)
- named slice, 'gdpPercap\_1952':'gdpPercap\_1962', *includes* the final element.



# Pandas-DataFrame Practice

1. GDP per capita for all countries in 1982.
  - `data['gdpPercap_1982']`
2. GDP per capita for Denmark for all years.
  - `data.loc['Denmark',:]`
3. GDP per capita for all countries for years *after* 1985.
  - `data.loc[:, 'gdpPercap_1985':]`
4. GDP per capita for each country in 2007 as a multiple of GDP per capita for that country in 1952.
  - `data['gdpPercap_2007']/data['gdpPercap_1952']`



# Keypoints

- Use the Pandas library to get basic statistics out of tabular data.
- Use `index_col` to specify that a column's values should be used as row headings.
- Use `DataFrame.info` to find out more about a dataframe.
- The `DataFrame.columns` variable stores information about the dataframe's columns.
- Use `DataFrame.T` to transpose a dataframe.
- Use `DataFrame.describe` to get summary statistics about data.

# Keypoints

- Use `DataFrame.iloc[..., ...]` to select values by integer location.
- Use `:` on its own to mean all columns or all rows.
- Select multiple columns or rows using `DataFrame.loc` and a named slice.
- Result of slicing can be used in further operations.
- Use comparisons to select data based on value.
- Select values or NaN using a Boolean mask.

# Matplotlib

```
import matplotlib.pyplot as plt
```

<https://matplotlib.org/stable/index.html>

<https://matplotlib.org/stable/gallery/index.html>

# Plotting

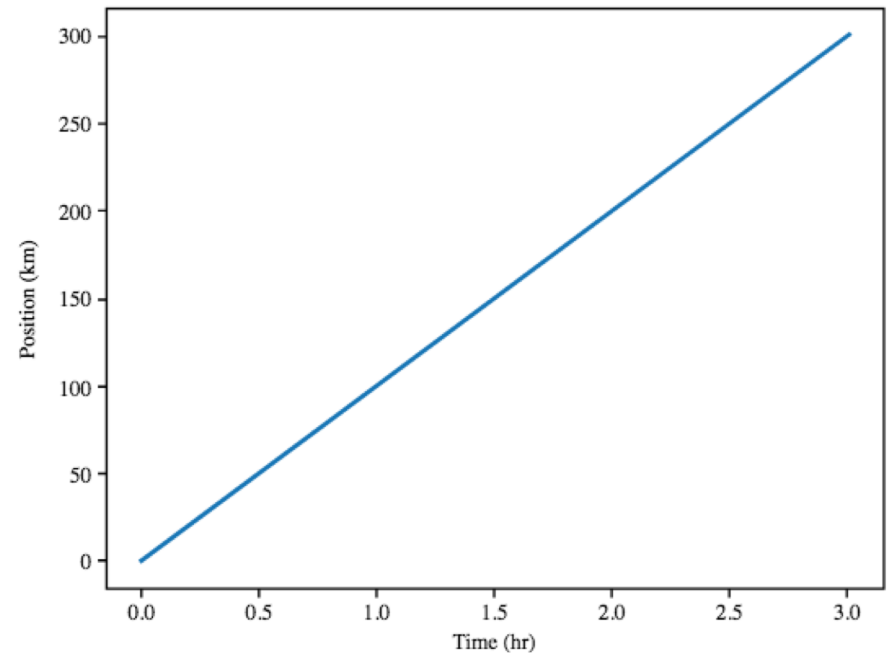
```
time = [0, 1, 2, 3] position = [0, 100, 200, 300]
```

```
plt.plot(time, position)
```

```
plt.xlabel('Time (hr)')
```

```
plt.ylabel('Position (km)')
```

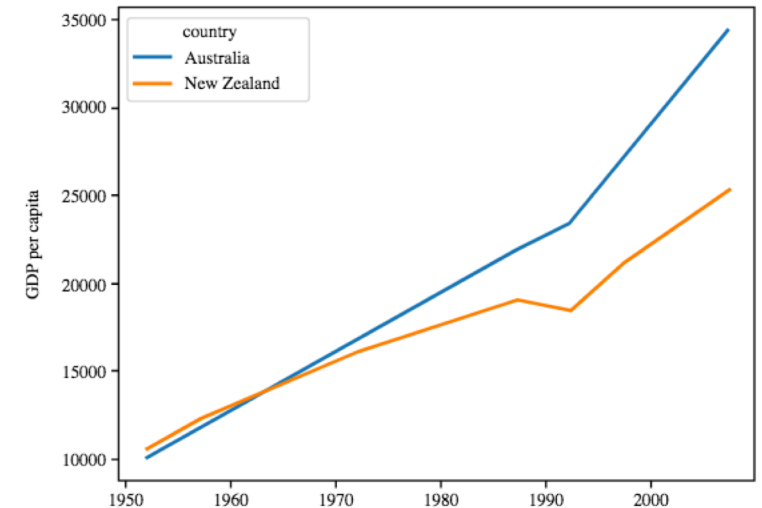
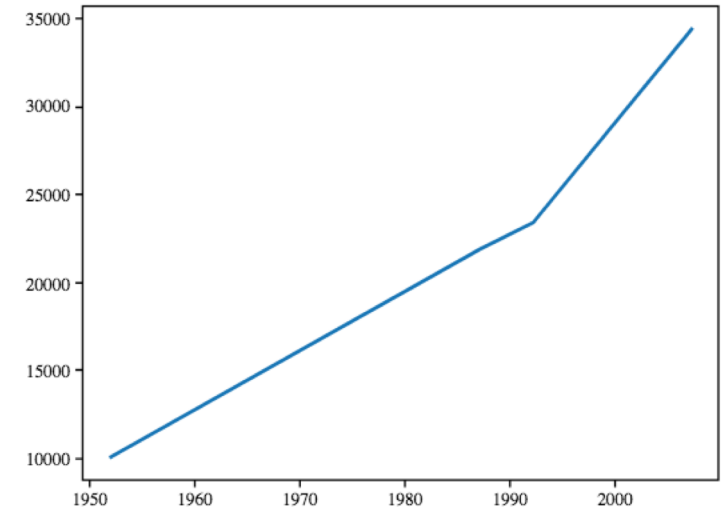
```
plt.show()
```



# Plotting + DataFrame

```
import pandas as pd
data =
pd.read_csv('data/gapminder_gdp_oceania.csv',
index_col='country')
years = data.columns.str.strip('gdpPercap_')
data.columns = years.astype(int)
data.loc['Australia'].plot()
```

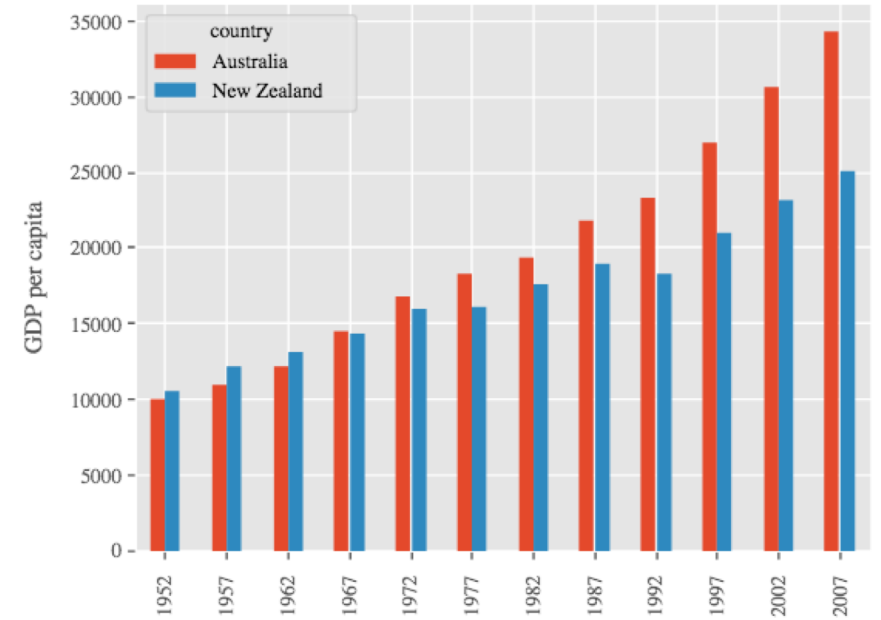
```
data.T.plot()
plt.ylabel('GDP per capita')
```



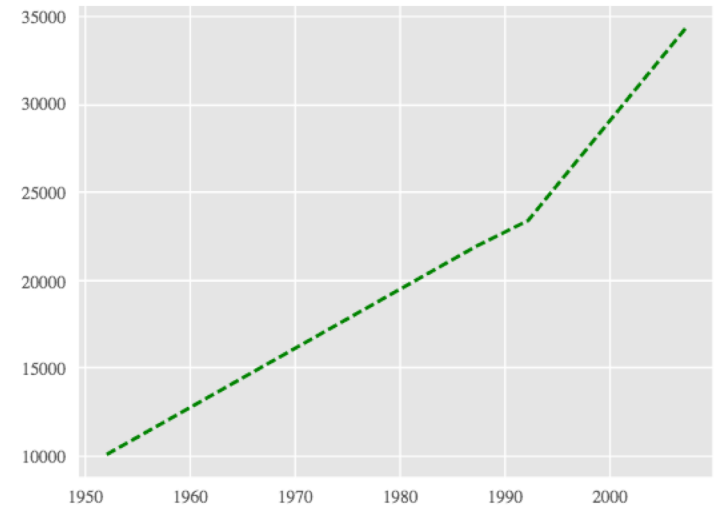


# Plotting + DataFrames

```
plt.style.use('ggplot')  
data.T.plot(kind='bar')  
plt.ylabel('GDP per capita')
```



```
years = data.columns  
gdp_australia = data.loc['Australia']  
plt.plot(years, gdp_australia, 'g--')
```



# Plotting

*# Select two countries' worth of data.*

```
gdp_australia = data.loc['Australia']
```

```
gdp_nz = data.loc['New Zealand']
```

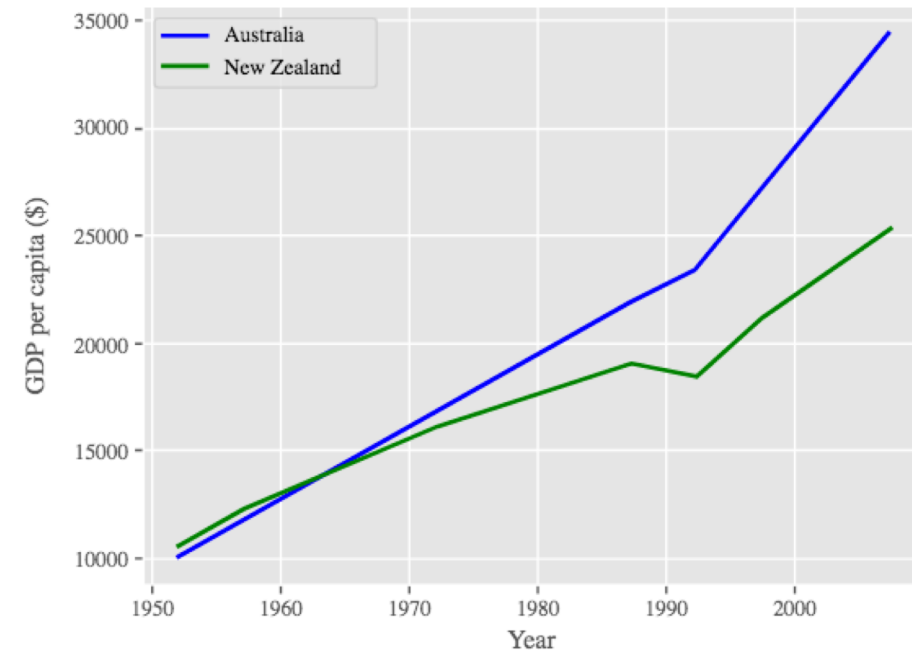
```
plt.plot(years, gdp_australia, 'b-', label='Australia')
```

```
plt.plot(years, gdp_nz, 'g-', label='New Zealand')
```

```
plt.legend(loc='upper left')
```

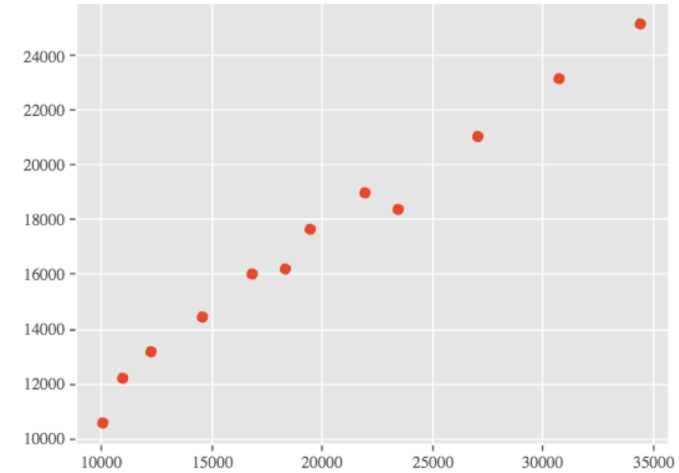
```
plt.xlabel('Year')
```

```
plt.ylabel('GDP per capita ($)')
```

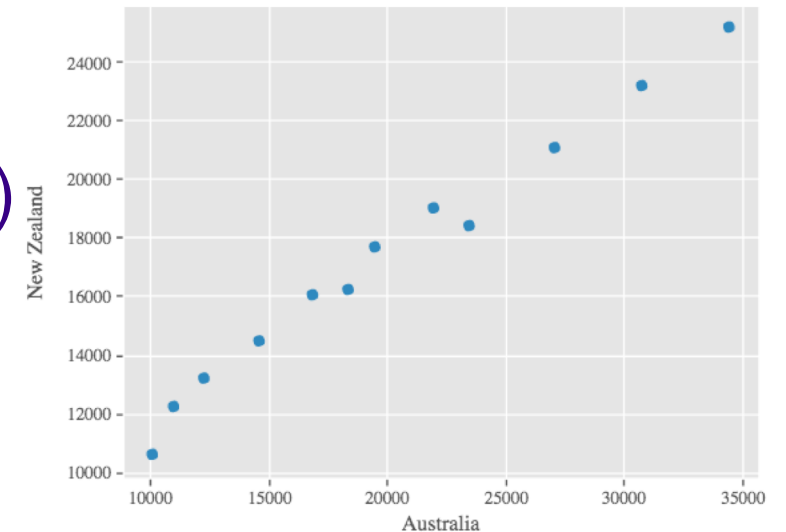


# Plotting

```
plt.scatter(gdp_australia, gdp_nz)
```



```
data.T.plot.scatter(x = 'Australia', y = 'New Zealand')
```

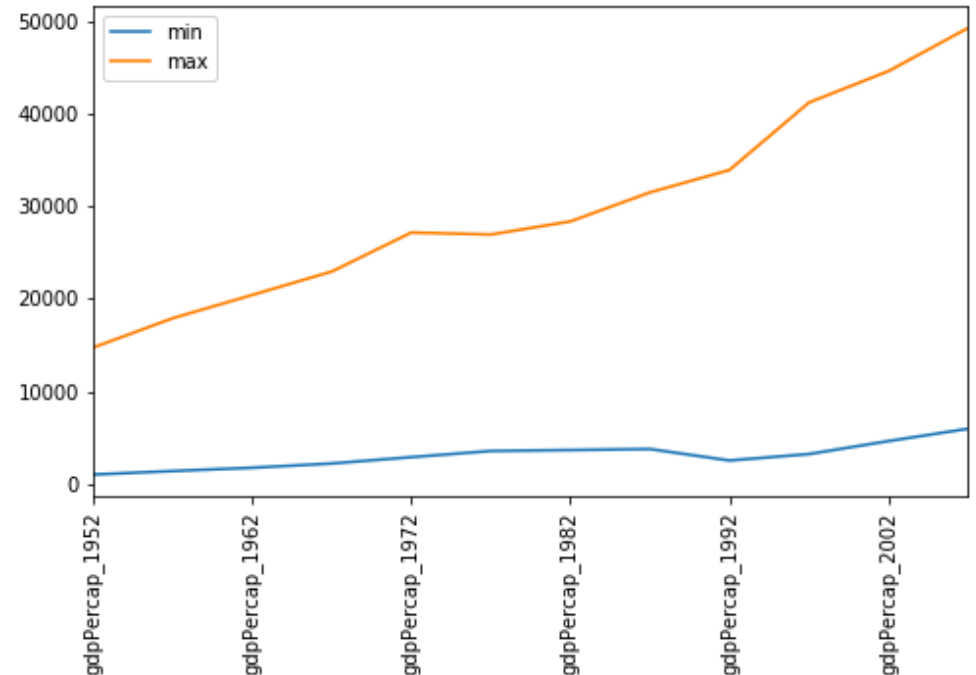


# Plotting-Practice

Fill in the blanks below to plot the minimum GDP per capita over time for all the countries in Europe. Modify it again to plot the maximum GDP per capita over time for Europe.

```
data_europe = pd.read_csv('data/gapmind  
index_col='country')
```

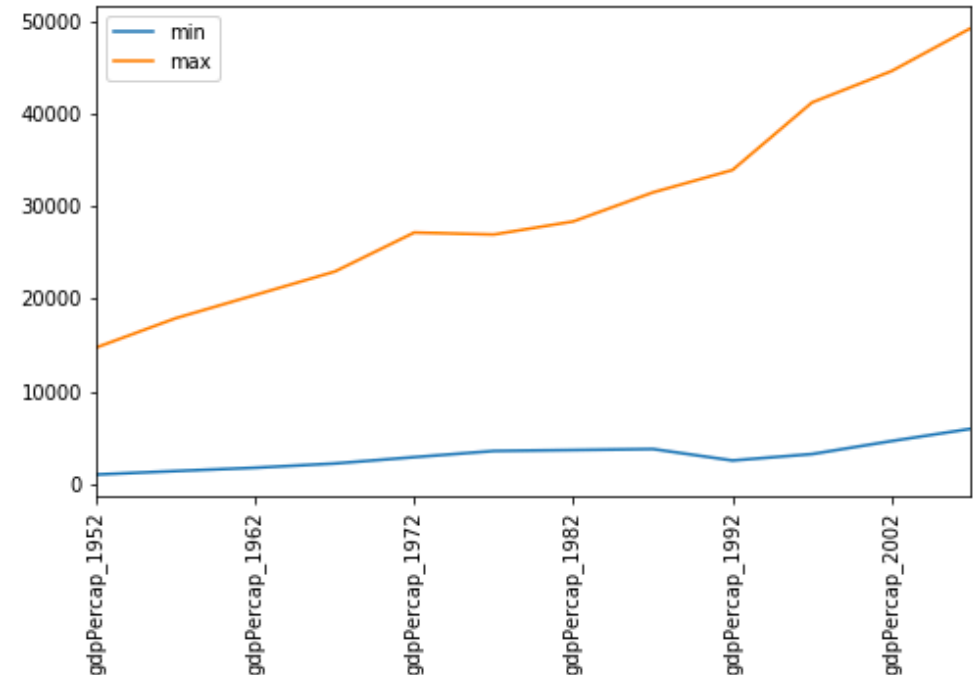
```
data_europe._____.plot(label='min')  
data_europe._____  
plt.legend(loc='best')  
plt.xticks(rotation=90)
```



# Plotting-Practice

```
data_europe = pd.read_csv('data/gapminder_gdp_europe.csv',  
index_col='country')
```

```
data_europe.min().plot(label='min')  
data_europe.max().plot(label='max')  
plt.legend(loc='best')  
plt.xticks(rotation=90)
```



# Plotting-Saving

- `plt.savefig('my_figure.png')`
- `plt` refer to a global figure variable and after a figure has been displayed to the screen
- \*Make sure you call `plt.savefig` before the plot is displayed to the\* screen,
- `fig = plt.gcf()` *# get current figure*
- `data.plot(kind='bar')`
- `fig.savefig('my_figure.png')`

# Keypoints

- [matplotlib](#) is the most widely used scientific plotting library in Python.
- Plot data directly from a Pandas dataframe.
- Select and transform data, then plot it.
- Many styles of plot are available: see the [Python Graph Gallery](#) for more options.
- Can plot many sets of data together.