

# COMP41680

## More Advanced Pandas

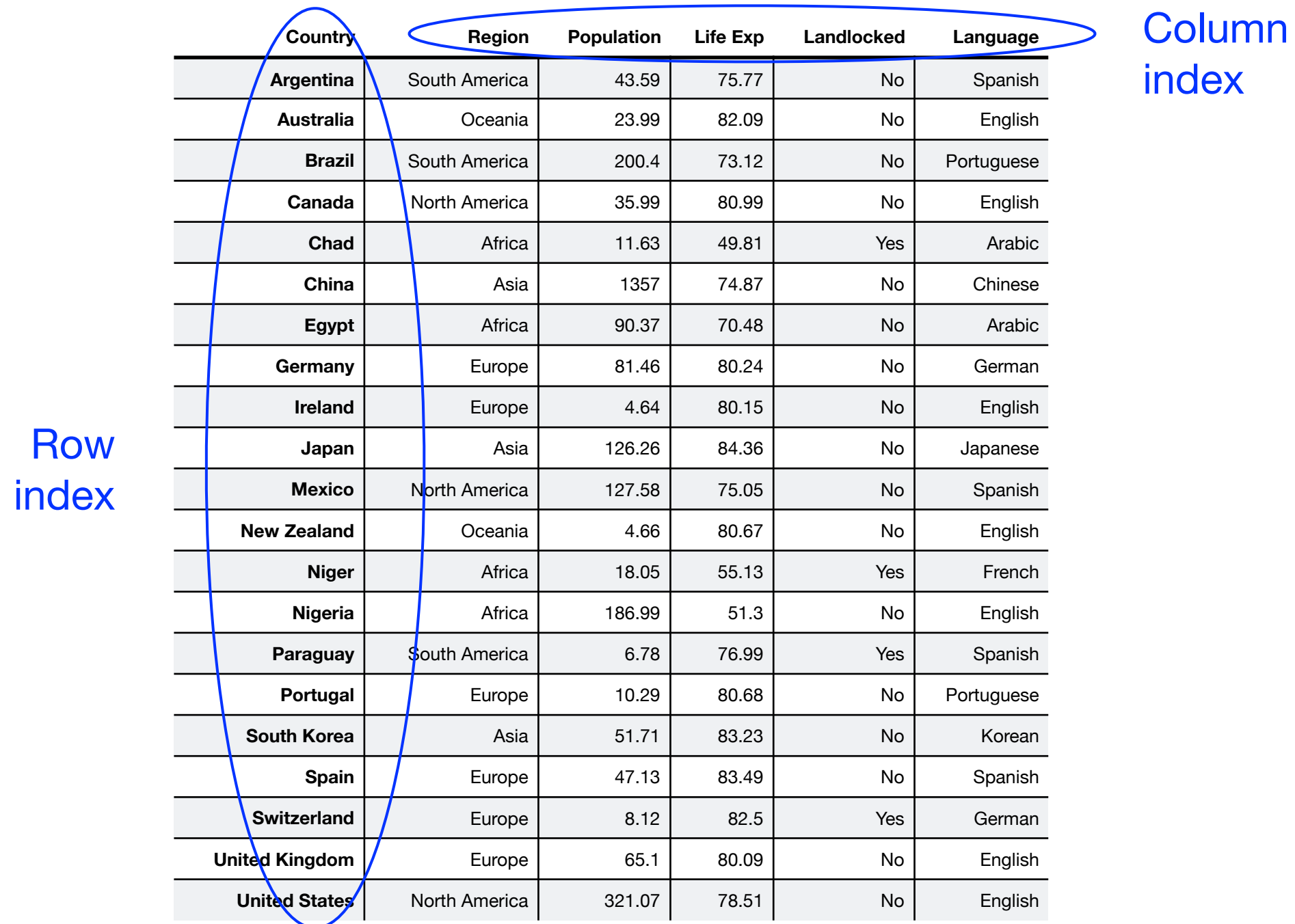
**Derek Greene**

**UCD School of Computer Science**  
**Spring 2023**



# Reminder: DataFrames

- A Pandas DataFrame is a flexible two-dimensional tabular data structure, where rows and columns both have index labels.



The diagram illustrates a Pandas DataFrame with 20 rows and 7 columns. The columns are labeled: Country, Region, Population, Life Exp, Landlocked, and Language. The rows are labeled with country names. A blue oval on the left side of the table, labeled "Row index", encompasses the entire table. A blue oval on the top of the table, labeled "Column index", encompasses the header row. The data is as follows:

Country	Region	Population	Life Exp	Landlocked	Language
Argentina	South America	43.59	75.77	No	Spanish
Australia	Oceania	23.99	82.09	No	English
Brazil	South America	200.4	73.12	No	Portuguese
Canada	North America	35.99	80.99	No	English
Chad	Africa	11.63	49.81	Yes	Arabic
China	Asia	1357	74.87	No	Chinese
Egypt	Africa	90.37	70.48	No	Arabic
Germany	Europe	81.46	80.24	No	German
Ireland	Europe	4.64	80.15	No	English
Japan	Asia	126.26	84.36	No	Japanese
Mexico	North America	127.58	75.05	No	Spanish
New Zealand	Oceania	4.66	80.67	No	English
Niger	Africa	18.05	55.13	Yes	French
Nigeria	Africa	186.99	51.3	No	English
Paraguay	South America	6.78	76.99	Yes	Spanish
Portugal	Europe	10.29	80.68	No	Portuguese
South Korea	Asia	51.71	83.23	No	Korean
Spain	Europe	47.13	83.49	No	Spanish
Switzerland	Europe	8.12	82.5	Yes	German
United Kingdom	Europe	65.1	80.09	No	English
United States	North America	321.07	78.51	No	English

# Data Normalisation

---

- Many data analysis techniques require that all input features in a dataset share a common scale.
- **Normalisation**: a data pre-processing step, which involves changing the values of numeric columns in the data to a common scale, without distorting differences in the ranges of values.
- Many different normalisation approaches:
  - **Mean normalisation**: subtract feature's mean value from all values for that feature.
  - **Max normalisation**: divide all values for a feature by the maximum value for that feature.
  - **Z-score normalisation**: for all values for a feature, subtract the feature mean and divide by the feature's standard deviation.

$$X' = \frac{X - \mu_X}{\sigma_X}$$

# Data Normalisation

- Min-max normalisation**: rescales the range of a feature's values to  $[0,1]$ , based on its minimum and maximum values.

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

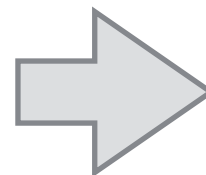
Example: Life expectancy values per country

Country	Life Exp
Argentina	75.77
Australia	82.09
Brazil	73.12
Canada	80.99
Chad	49.81
China	74.87
Egypt	70.48
Germany	80.24
Ireland	80.15
Japan	84.36
Mexico	75.05
...	...

$$X_{\min} = 49.81$$

$$X_{\max} = 84.36$$

$$X' = \frac{X - 49.81}{84.36 - 49.81}$$



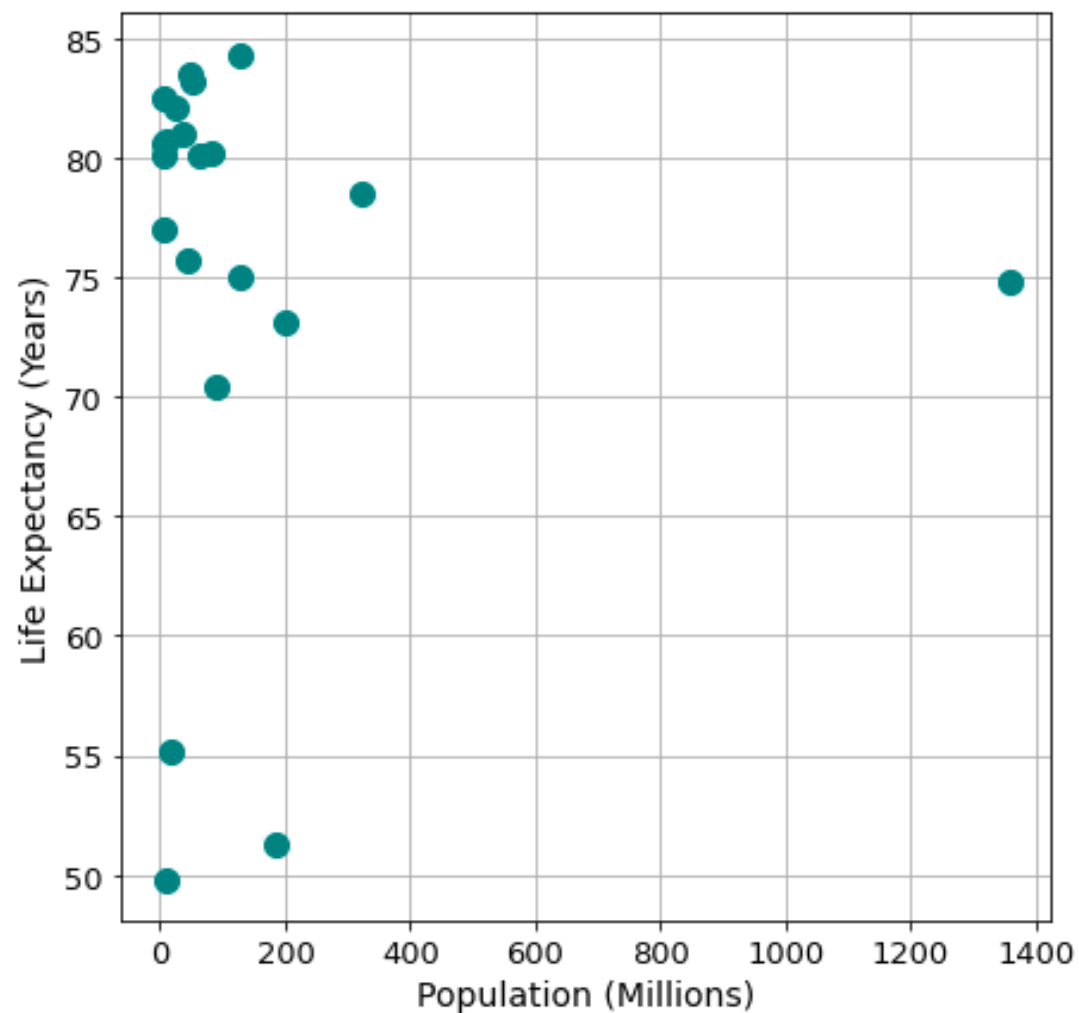
Country	Norm Life Exp
Argentina	0.7514
Australia	0.9343
Brazil	0.6747
Canada	0.9025
Chad	0.0000
China	0.7253
Egypt	0.5983
Germany	0.8808
Ireland	0.8781
Japan	1.0000
Mexico	0.7305
...	...



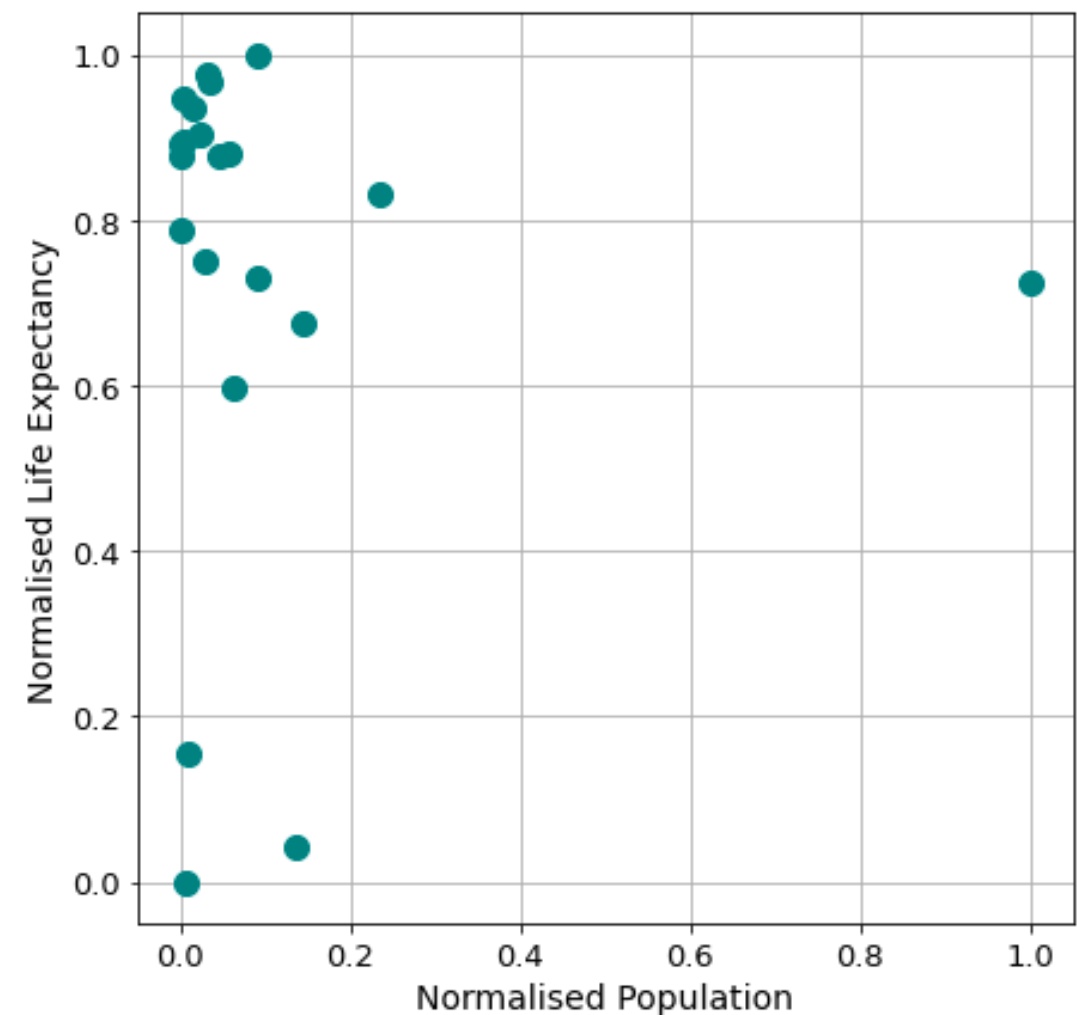
# Data Normalisation

- **Min-max normalisation**: rescales the range of a feature's values to  $[0,1]$ , based on its minimum and maximum values.

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$



Original numeric values for life expectancy and population



Min-max normalised values for life expectancy and population

# Data Normalisation

---

- We can easily apply min-max normalisation to a Series or individual column of a DataFrame:

Example: Life expectancy values per country

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

```
life_min = df["Life Exp"].min()  
life_max = df["Life Exp"].max()  
df["Life Exp"] = (df["Life Exp"]-life_min)/(life_max-life_min)
```

- We can also apply other types of normalisation by applying arithmetic operators to a Series or DataFrame:

```
df - df.mean()
```

Subtract the mean  
column values from  
each row in df

```
df / df.max()
```

Divide values in each  
column by the maximum  
value for that column

```
df / df.sum()
```

Divide values in each  
column by the total  
for that column

# Frequency Tables

- When working with a Series with categorical values, frequency tables provide a way of counting the frequency of different values.
- The function `value_counts()` returns a new Series with counts of unique values.
- We can also normalise the values, to calculate the relative frequencies of the unique values (i.e. percentages)

```
df["Region"].value_counts()
```

Europe	6
Africa	4
South America	3
North America	3
Asia	3
Oceania	2

```
df["Language"].value_counts()
```

English	7
Spanish	4
Portuguese	2
Arabic	2
German	2
Chinese	1
Japanese	1
French	1
Korean	1

```
df["Language"].value_counts(normalize=True)
```

English	0.333333
Spanish	0.190476
Portuguese	0.095238
Arabic	0.095238
German	0.095238
Chinese	0.047619
Japanese	0.047619
French	0.047619
Korean	0.047619

# Cross Tabulation

- **Cross tabulation** allows us to quantitatively analyse the relationship between multiple variables.
- In Pandas, this involves counting the frequency with which values from different columns in a DataFrame co-occur.

Example: If we divide countries by "Region", how many countries are "Landlocked" in each region?

Country	Region	Population	Life Exp	Landlocked	Language
Argentina	South America	43.59	75.77	No	Spanish
Australia	Oceania	23.99	82.09	No	English
Brazil	South America	200.4	73.12	No	Portuguese
Canada	North America	35.99	80.99	No	English
Chad	Africa	11.63	49.81	Yes	Arabic
China	Asia	1357	74.87	No	Chinese
Egypt	Africa	90.37	70.48	No	Arabic
Germany	Europe	81.46	80.24	No	German
Ireland	Europe	4.64	80.15	No	English
Japan	Asia	126.26	84.36	No	Japanese
Mexico	North America	127.58	75.05	No	Spanish
New Zealand	Oceania	4.66	80.67	No	English
Niger	Africa	18.05	55.13	Yes	French
Nigeria	Africa	186.99	51.3	No	English
Paraguay	South America	6.78	76.99	Yes	Spanish
Portugal	Europe	10.29	80.68	No	Portuguese
South Korea	Asia	51.71	83.23	No	Korean
Spain	Europe	47.13	83.49	No	Spanish
Switzerland	Europe	8.12	82.5	Yes	German
United Kingdom	Europe	65.1	80.09	No	English
United States	North America	321.07	78.51	No	English



# Cross Tabulation

- We can perform this using the `pd.crosstab()` function. The output is a new DataFrame, which represents a frequency table.
- We can compare values in one column (the sources of values) relative to another (our new index).

```
pd.crosstab(df["Region"],  
            df["Landlocked"])
```

Region vs Landlocked

Region	No	Yes
Africa	2	2
Asia	3	0
Europe	5	1
North America	3	0
Oceania	2	0
South America	2	1

```
pd.crosstab(df["Language"],  
            df["Landlocked"])
```

Language vs Landlocked

Region	No	Yes
Arabic	1	1
Chinese	1	0
English	7	0
French	0	1
German	1	1
Japanese	1	0
Korean	1	0
Portuguese	2	0
Spanish	3	1

# Aggregating Data

- Categorising a dataset into groups, and applying functions to each group is often an important step in data analysis.
- We can use the `groupby()` function to group data based on the values in a categorical column.

```
groups1 = df.groupby("Region")
```

Group each country by the categorical "Region" column

Country	Region	Population	Life Exp	Landlocked	Language
Argentina	South America	43.59	75.77	No	Spanish
Australia	Oceania	23.99	82.09	No	English
Brazil	South America	200.4	73.12	No	Portuguese
Canada	North America	35.99	80.99	No	English
Chad	Africa	11.63	49.81	Yes	Arabic
China	Asia	1357	74.87	No	Chinese
Egypt	Africa	90.37	70.48	No	Arabic
Germany	Europe	81.46	80.24	No	German
Ireland	Europe	4.64	80.15	No	English
Japan	Asia	126.26	84.36	No	Japanese
Mexico	North America	127.58	75.05	No	Spanish
New Zealand	Oceania	4.66	80.67	No	English
Niger	Africa	18.05	55.13	Yes	French
Nigeria	Africa	186.99	51.3	No	English
Paraguay	South America	6.78	76.99	Yes	Spanish
Portugal	Europe	10.29	80.68	No	Portuguese
South Korea	Asia	51.71	83.23	No	Korean
Spain	Europe	47.13	83.49	No	Spanish
Switzerland	Europe	8.12	82.5	Yes	German
United Kingdom	Europe	65.1	80.09	No	English
United States	North America	321.07	78.51	No	English

# Aggregating Data

- Categorising a dataset into groups, and applying functions to each group is often an important step in data analysis.
- We can use the `groupby()` function to group data based on the values in a categorical column.

```
groups1 = df.groupby("Region")
```

Group each country by the categorical "Region" column

Country	Region	Population	Life Exp	Landlocked	Language
Chad	Africa	11.63	49.81	Yes	Arabic
Egypt	Africa	90.37	70.48	No	Arabic
Niger	Africa	18.05	55.13	Yes	French
Nigeria	Africa	186.99	51.3	No	English
China	Asia	1357	74.87	No	Chinese
Japan	Asia	126.26	84.36	No	Japanese
South Korea	Asia	51.71	83.23	No	Korean
Germany	Europe	81.46	80.24	No	German
Ireland	Europe	4.64	80.15	No	English
Portugal	Europe	10.29	80.68	No	Portuguese
Spain	Europe	47.13	83.49	No	Spanish
Switzerland	Europe	8.12	82.5	Yes	German
United Kingdom	Europe	65.1	80.09	No	English
Canada	North America	35.99	80.99	No	English
Mexico	North America	127.58	75.05	No	Spanish
United States	North America	321.07	78.51	No	English
Australia	Oceania	23.99	82.09	No	English
New Zealand	Oceania	4.66	80.67	No	English
Argentina	South America	43.59	75.77	No	Spanish
Brazil	South America	200.4	73.12	No	Portuguese
Paraguay	South America	6.78	76.99	Yes	Spanish

# Aggregating Data

- The result of the `groupby()` function is a `GroupBy` object. Nothing has been computed yet, but it can be used to apply different operations to each of the groups, which result in new `DataFrames`.
- For example, we could compute the mean or the sum of the numeric features of the countries in each region:

```
groups1 = df.groupby("Region")
```

```
groups1.mean()
```

Region	Population	Life Exp
Africa	76.76	56.68
Asia	511.66	80.82
Europe	36.12	81.19
North America	161.55	78.18
Oceania	14.33	81.38
South America	83.59	75.29

```
groups1.sum()
```

Region	Population	Life Exp
Africa	307.04	226.72
Asia	1534.97	242.46
Europe	216.74	487.15
North America	484.64	234.55
Oceania	28.65	162.76
South America	250.77	225.88

# Aggregating Data

- We can apply the same grouping approach for the other columns.

Country	Region	Population	Life Exp	Landlocked	Language
Argentina	South America	43.59	75.77	No	Spanish
Australia	Oceania	23.99	82.09	No	English
Brazil	South America	200.4	73.12	No	Portuguese
Canada	North America	35.99	80.99	No	English
Chad	Africa	11.63	49.81	Yes	Arabic
China	Asia	1357	74.87	No	Chinese
Egypt	Africa	90.37	70.48	No	Arabic
Germany	Europe	81.46	80.24	No	German
Ireland	Europe	4.64	80.15	No	English
Japan	Asia	126.26	84.36	No	Japanese
Mexico	North America	127.58	75.05	No	Spanish
New Zealand	Oceania	4.66	80.67	No	English
Niger	Africa	18.05	55.13	Yes	French
Nigeria	Africa	186.99	51.3	No	English
Paraguay	South America	6.78	76.99	Yes	Spanish
Portugal	Europe	10.29	80.68	No	Portuguese
South Korea	Asia	51.71	83.23	No	Korean
Spain	Europe	47.13	83.49	No	Spanish
Switzerland	Europe	8.12	82.5	Yes	German
United Kingdom	Europe	65.1	80.09	No	English
United States	North America	321.07	78.51	No	English

```
groups2 = df.groupby("Landlocked")
groups2.mean()
```

Landlocked	Population	Life Exp
No	163.43	77.36
Yes	11.15	66.11

```
groups2 = df.groupby("Landlocked")
groups2.sum()
```

Landlocked	Population	Life Exp
No	2778.23	1315.09
Yes	44.58	264.43

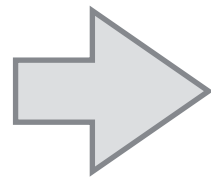
Group each country by the categorical "Landlocked" column



# Filtering Duplicate Rows

- To drop duplicate rows in a DataFrame, call the associated `drop_duplicates()` function. By default, it matches duplicate rows based on all columns and only keeps the first occurrence.

	coffee	size	strength
0	Napoli	110	4
1	Napoli	110	4
2	Napoli	40	4
3	Venezia	110	3
4	Venezia	110	3
5	Firenze	40	2



```
df.drop_duplicates()
```

	coffee	size	strength
0	Napoli	110	4
2	Napoli	40	4
3	Venezia	110	3
5	Firenze	40	2

First occurrence  
of duplicates kept  
(#2 and #5)

- To remove duplicates based on values in specific column(s), use the `subset` argument.

```
df.drop_duplicates(  
    subset=["coffee", "strength"])
```

	coffee	size	strength
0	Napoli	110	4
3	Venezia	110	3
5	Firenze	40	2

Only compare on  
2 columns

# Missing Values

---

- Many real datasets have missing values, either because the data existed but was not collected, or because it never existed. Generally these cases can be divided into 3 scenarios:
  1. **Missing Completely at Random (MCAR)**: The fact that a certain value is missing has nothing to do with any values, observed or missing. There is nothing systematic that makes some values more likely to be missing than others.
  2. **Missing at Random (MAR)**: The propensity for a value to be missing is not related to the missing data, but it is related to some of the observed data (e.g. people with a certain age might be less likely to reveal their salary).
  3. **Missing not at Random (MNAR)**: There is a systematic relationship between the propensity of a value to be missing and its values (e.g. people with high salaries might be less likely to reveal their salary).

# Missing Values in Pandas

- In Pandas missing data can be represented in two different ways:
  - NaN (Not a Number)**: indicating a missing floating point value.
  - None**: a standard Python null value for any other type of value.
- We can use the functions `isnull()` and `notnull()` to identify and count missing and non-missing values:

```
df.isnull()
```

Country	City	Population	GDP-BN
Ireland	Dublin	4613000	250.8
Belgium	Brussels	NaN	524.4
France	Paris	66627000	NaN
Spain	Madrid	46439000	1619.0

Country	City	Population	GDP-BN
Ireland	False	False	False
Belgium	False	True	False
France	False	False	True
Spain	False	False	False

```
df.notnull()
```

Country	City	Population	GDP-BN
Ireland	True	True	True
Belgium	True	False	True
France	True	True	False
Spain	True	True	True

# Handling Missing Values

---

- There are two general approaches to handling missing values. The appropriate approach depends on the data and the task at hand.

## 1. Filter out missing data

- Filter by example: Delete any row containing missing values from the dataset.
- Filter by feature: Delete any column containing missing values from the dataset.

## 2. Fill in missing data

- Fill with a global constant: Replace all missing values in the dataset with a fixed value.
- Fill with mean/median: Replace missing values based on the row or column mean/median from existing data.
- Prediction: Use algorithm to predict most likely replacements based on existing values (e.g. regression, interpolation).

# Filtering Out Missing Data

- Filtering eliminates the presence of rows and/or columns in a DataFrame which contain one or more missing values.
- Note this can also remove potentially useful data.
- We could call `drop()` to remove one or more rows from a DataFrame, by specifying a list of row indexes. This creates a copy of the original DataFrame.

Country	City	Population	GDP-BN
Ireland	Dublin	4613000	250.8
Belgium	Brussels	NaN	524.4
France	Paris	66627000	NaN
Spain	Madrid	46439000	1619.0

```
names = ["Belgium", "France"]  
df.drop(names, axis=0)
```

Note: `axis=0` means drop rows

Country	City	Population	GDP-BN
Ireland	Dublin	4613000	250.8
Spain	Madrid	46439000	1619.0



# Filtering Out Missing Data

- Instead of manually specifying rows, we can call `dropna()` to remove all rows containing missing values:

```
df.dropna(axis=0)
```

Country	City	Population	GDP-BN
Ireland	Dublin	4613000	250.8
Spain	Madrid	46439000	1619.0

- We can also use `drop()` and `dropna()` to remove columns containing missing values from a DataFrame by specifying `axis=1`.

```
cols = ["Population", "GDP-BN"]  
df.drop(cols, axis=1)
```

```
df.dropna(axis=1)
```

Country	City
Ireland	Dublin
Belgium	Brussels
France	Paris
Spain	Madrid

# Example: Missing Values in Pandas

- Example:** In the Titanic dataset, 86 out of 418 values in the numeric "Age" column are missing, as indicated by NaN values.

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S
8	900	3	Abraham, Mrs. Joseph (Sophie Halaut Easu)	female	18.0	0	0	2657	7.2292	NaN	C
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	S
10	902	3	Ilieff, Mr. Ylio	male	NaN	0	0	349220	7.8958	NaN	S
11	903	1	Jones, Mr. Charles Cresson	male	46.0	0	0	694	26.0000	NaN	S
12	904	1	Snyder, Mrs. John Pillsbury (Nelle Stevenson)	female	23.0	1	0	21228	82.2667	B45	S
13	905	2	Howard, Mr. Benjamin	male	63.0	1	0	24065	26.0000	NaN	S
14	906	1	Chaffee, Mrs. Herbert Fuller (Carrie Constance...	female	47.0	1	0	W.E.P. 5734	61.1750	E31	S
15	907	2	del Carlo, Mrs. Sebastiano (Argenia Genovesi)	female	24.0	1	0	SC/PARIS 2167	27.7208	NaN	C
16	908	2	Keane, Mr. Daniel	male	35.0	0	0	233734	12.3500	NaN	Q
17	909	3	Assaf, Mr. Gerios	male	21.0	0	0	2692	7.2250	NaN	C
18	910	3	Ilmakangas, Miss. Ida Livija	female	27.0	1	0	STON/O2. 3101270	7.9250	NaN	S
19	911	3	Assaf Khalil, Mrs. Mariana (Miriam)"	female	45.0	0	0	2696	7.2250	NaN	C
20	912	1	Rothschild, Mr. Martin	male	55.0	1	0	PC 17603	59.4000	NaN	C
21	913	3	Olsen, Master. Artur Karl	male	9.0	0	1	C 17368	3.1708	NaN	S
22	914	1	Flegenheim, Mrs. Alfred (Antoinette)	female	NaN	0	0	PC 17598	31.6833	NaN	S

```
df.isnull().sum()
```

PassengerId	0
Pclass	0
Name	0
Sex	0
Age	86
SibSp	0
Parch	0
Ticket	0
Fare	1
Cabin	327
Embarked	0

Sum the number of null (NaN) values in each column in the DataFrame

# Example: Missing Values in Pandas

- If we expect a column or row to provide important information, we may want to keep it and fill/estimate the missing values.

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S
5	897	3	Svensson, Mr. Johan Cervin	male	14.0	0	0	7538	9.2250	NaN	S
6	898	3	Connolly, Miss. Kate	female	30.0	0	0	330972	7.6292	NaN	Q
7	899	2	Caldwell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	S
8	900	3	Abraham, Mrs. Joseph (Sophie Halaut Easu)	female	18.0	0	0	2657	7.2292	NaN	C
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	A/4 48871	24.1500	NaN	\$
10	902	3	Ilieff, Mr. Ylio	male	NaN	0	0	349220	7.8958	NaN	\$
11	903	1	Jones, Mr. Charles Cresson	male	46.0	0	0	694	26.0000	NaN	\$
12	904	1	Snyder, Mrs. John Pillsbury (Nelle Stevenson)	female	23.0	1	0	21228	82.2667	B45	\$
13	905	2	Howard, Mr. Benjamin	male	63.0	1	0	24065	26.0000	NaN	\$
14	906	1	Chaffee, Mrs. Herbert Fuller (Carrie Constance...	female	47.0	1	0	W.E.P. 5734	61.1750	E31	\$
15	907	2	del Carlo, Mrs. Sebastiano (Argenia Genovesi)	female	24.0	1	0	SC/PARIS 2167	27.7208	NaN	C
16	908	2	Keane, Mr. Daniel	male	35.0	0	0	233734	12.3500	NaN	Q
17	909	3	Assaf, Mr. Gerios	male	21.0	0	0	2692	7.2250	NaN	C
18	910	3	Ilmakangas, Miss. Ida Livija	female	27.0	1	0	STON/O2. 3101270	7.9250	NaN	S
19	911	3	Assaf Khalil, Mrs. Mariana (Miriam")"	female	45.0	0	0	2696	7.2250	NaN	C
20	912	1	Rothschild, Mr. Martin	male	55.0	1	0	PC 17603	59.4000	NaN	C
21	913	3	Olsen, Master. Artur Karl	male	9.0	0	1	C 17368	3.1708	NaN	S
22	914	1	Flegenheim, Mrs. Alfred (Antoinette)	female	NaN	0	0	PC 17598	31.6833	NaN	S

The columns "Age" and "Cabin" could be strongly indicative of survival.

# Filling In Missing Data

- We could replace all missing values in a DataFrame with the same value using `fillna()`. This creates a copy of the DataFrame.

```
df2 = df.fillna(0)
```

Replace every missing value  
with 0

- A more sensible approach would be to fill in missing values using a column's mean value:

```
mean_age = df["Age"].mean()  
df["Age"] = df["Age"].fillna(mean_age)
```

Fill in every NaN value with  
mean value 30.27

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.50000	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.00000	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.00000	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.00000	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.00000	1	1	3101298	12.2875	NaN	S
5	897	3	Svensson, Mr. Johan Cervin	male	14.00000	0	0	7538	9.2250	NaN	S
6	898	3	Connolly, Miss. Kate	female	30.00000	0	0	330972	7.6292	NaN	Q
7	899	2	Caldwell, Mr. Albert Francis	male	26.00000	1	1	248738	29.0000	NaN	S
8	900	3	Abraham, Mrs. Joseph (Sophie Halaut Easu)	female	18.00000	0	0	2657	7.2292	NaN	C
9	901	3	Davies, Mr. John Samuel	male	21.00000	2	0	A/4 48871	24.1500	NaN	S
10	902	3	Ilieff, Mr. Ylio	male	30.27259	0	0	349220	7.8958	NaN	S

NaN has been  
replaced with  
30.27

# Exporting DataFrames

- After modifying a DataFrame (e.g. applying filtering or cleaning), we might often want to save the modified data as a new CSV file.
- With Pandas, we can use the `to_csv()` function. The `sep` parameter specifies the string used to separate values in the file.

```
df2 = df.head(5)
```

	name	population	life_expect
code			
<b>BEL</b>	Belgium	11338476	80.99
<b>ESP</b>	Spain	46484533	82.83
<b>FIN</b>	Finland	5495303	81.78
<b>FRA</b>	France	66892205	82.27
<b>IRL</b>	Ireland	4749777	81.61

```
df2.to_csv("countries5.csv")
```

```
df2.to_csv("countries5b.csv", sep=";")
```

```
code,name,population,life_expect
BEL,Belgium,11338476,80.99
ESP,Spain,46484533,82.83
FIN,Finland,5495303,81.78
FRA,France,66892205,82.27
IRL,Ireland,4749777,81.61
```

Output file: countries5.csv

```
code;name;population;life_expect
BEL;Belgium;11338476;80.99
ESP;Spain;46484533;82.83
FIN;Finland;5495303;81.78
FRA;France;66892205;82.27
IRL;Ireland;4749777;81.61
```

Output file: countries5b.csv



# Exporting DataFrames

- We can also serialise a Pandas DataFrame so that we can use it later.
- We can easily serialise this DataFrame, using the Pandas wrapper for Pickle, via the `to_pickle()` function.
- Then we can load it back later using the `pd.read_pickle()` function.
- The version we have read is an exact copy of the original DataFrame.

	Country	Region	Population	Life Exp
0	Argentina	South America	43.59	75.77
1	Australia	Oceania	23.99	82.09
2	Brazil	South America	200.40	73.12
3	Canada	North America	35.99	80.99

Specify the output file path

```
countries.to_pickle("pd-example.pkl")
```

Specify the input file path

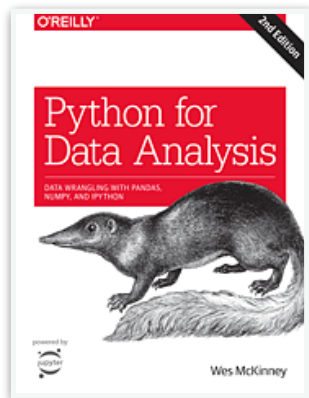
```
df = pd.read_pickle("pd-example.pkl")  
df
```

	Country	Region	Population	Life Exp
0	Argentina	South America	43.59	75.77
1	Australia	Oceania	23.99	82.09
2	Brazil	South America	200.40	73.12
3	Canada	North America	35.99	80.99

# Further Reading

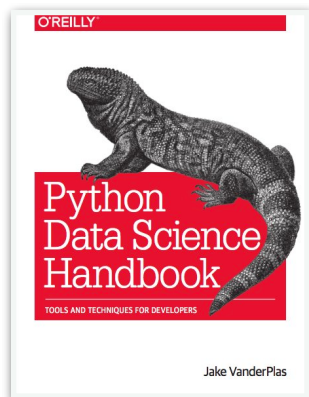
---

For further details on using Pandas for data manipulation see:



Python for Data Analysis, 2nd Edition  
William McKinney

<http://shop.oreilly.com/product/0636920050896.do>



Python Data Science Handbook  
Jake VanderPlas

<http://shop.oreilly.com/product/0636920034919.do>



## Getting started

New to *pandas*? Check out the getting started guides. They contain an introduction to *pandas*' main concepts and links to additional tutorials.

[To the getting started guides](#)



## User guide

The user guide provides in-depth information on the key concepts of *pandas* with useful background information and explanation.

[To the user guide](#)

Official Pandas Documentation

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