CAMBRIDGE SPARK

FURTHER PANDAS

PANDAS BASICS

- Useful methods and functions
    - `describe`, `get_dummies`, etc.
- Fancy indexing
    - `loc` vs. `iloc`
- Column-wise operations
    - `arithmetic`, `apply`, etc.
- SQL inspired functions
    - `merge`, `join`, `groupby`

ADVANCED PROCESSING FUNCTIONS

- Pivot Tables
  - `pivot`, `pivot_table`, `melt`
- Multi-Index
  - `MultiIndex`, `stack`, `crosstab`
- Pandas `Series`
  - `factorize`, `cut`

PIVOT TABLES

- Similar to pivot tables in Excel
- Reshape a dataset, so each row has a category and subcategory associated to a value
    - example: Great Britain's (category: country) online sales (subcategory: online) make a profit (value: profit) of X

EXAMPLE

```
import pandas as pd
df = pd.read_csv('data/profits.csv')

df.head()
```

	Country	Type	Profit
0	GB	online	50
1	US	online	25
2	GB	in-store	30
3	US	in-store	100

- The category is the country.
- The sub-category is the type.
- The value is the profit

We want to pivot the table so that there is only one copy of each country in the country column.

EXAMPLE `df.pivot`:

```
df.pivot(index='Country', columns='Type')
```

	Profit	
Type	**in-store**	**online**
Country		
GB	30	50
US	100	25

PIVOT TABLES

There are two functions to build pivot tables in pandas:

- `.pivot(index, columns, [values])`
    - numerical and categorical data
    - no aggregation - will fail if there are duplicate rows with same categories
- `.pivot_table(index, columns, [values], [aggfunc])`
    - only with numerical data
    - can do aggregation (similar to `groupby`)

`df.pivot`

- `.pivot` works with numerical and categorical data (if rows are unique per subcategory)
    - `index`: the column that will become the new index
    - `columns`: the subcategories to split by
    - `values`: [optional] if you want to keep subset of columns

df.pivot

In the previous example, what was the original index?

```
df.head()
```

	Country	Type	Profit
0	GB	online	50
1	US	online	25
2	GB	in-store	30
3	US	in-store	100

Is there another column that we could use as the index?

`df.pivot`

Yes! We could pivot on the `Type`:

```
df.pivot(index='Type', columns='Country')
```

	Profit	
Country	**GB**	**US**
Type		
in-store	30	100
online	50	25

df.pivot_table

- pivot_table only numerical data and we can add an aggregator:
  - index: the columns that will become the new index
  - columns: the subcategories to split by
  - values: [optional] subset of columns to keep
  - aggfunc: [optional] aggregation function for duplicate rows

EXAMPLE df.pivot_table:

```
df = pd.read_csv('data/profits_with_duplicates.csv')
df
```

	Country	Type	Profit
0	GB	online	50
1	US	in-store	25
2	GB	online	30
3	US	in-store	100
4	GB	in-store	25
5	GB	in-store	30
6	GB	online	75
7	US	online	110
8	US	in-store	10

EXAMPLE df.pivot_table:

- df.pivot will fail because it can't aggregate

```
df = df.pivot_table(index='Country', columns='Type', aggfunc='sum')
df
```

	Profit	
Type	**in-store**	**online**
Country		
GB	55	155
US	135	110

melt

- Reverses the pivot operation
- Reshape so we have a column of categories (e.g. country) which has one value (e.g. profit) per subcategory)
- Arguments:
  - id_vars: the index column to be stretched out
  - value_vars: the new subcategory (could be multiple of these)

EXAMPLE melt:

```python
df = pd.read_csv('data/profits.csv')
df = df.pivot(index='Country', columns='Type')

# MultiIndex processing - this will be covered in the next section
df = df['Profit'].reset_index()

df
```

Type	Country	in-store	online
0	GB	30	50
1	US	100	25

```python
pd.melt(df, id_vars='Country', var_name='Type', value_name='Profit')
```

	Country	Type	Profit
0	GB	in-store	30
1	US	in-store	100
2	GB	online	50
3	US	online	25

Practical 1 - pivot, pivot_table, melt

practical1.ipynb

MULTIINDEX

- Hierarchical index over several columns or rows
    - `pivot` (and some other commands) create a multi-index by default

```
df = pd.read_csv('data/profits.csv')
df = df.pivot(index='Country', columns='Type')
df
```

	Profit	
Type	**in-store**	**online**
Country		
GB	30	50
US	100	25

'Profit' is a multi-index with subcolumns.

COLUMN MULTIINDEX

Imagine our data is set up in lists:

```
# [online, in-store]
gb = [30, 40]
us = [100, 25]

df = pd.DataFrame([gb,us], columns=['online','in-store'])
df
```

	online	in-store
0	30	40
1	100	25

We could combine online and in-store into a MultiIndex called "profit".

COLUMN MULTIINDEX

Create in three ways: `from_tuples` , `from_arrays` , `from_frames`

```
# single index:
df = pd.DataFrame([gb,us], columns=['online','in-store'])

# MultiIndex:
columns = pd.MultiIndex.from_tuples([('profit','online'), ('profit','in-store')])
df = pd.DataFrame([gb,us], columns=columns)
df
```

	profit	
	online	**in-store**
0	30	40
1	100	25

COLUMN MULTI-INDEX

We can chain index or use `loc`:

```
df['profit']['online']
```

```
0     30
1    100
Name: online, dtype: int64
```

```
df.loc[:, ('profit', 'online')]
```

```
0     30
1    100
Name: (profit, online), dtype: int64
```

ROW MULTIINDEX

```
london    = [100,20]
cambridge = [200,30]
new_york  = [300,40]

index = pd.MultiIndex.from_tuples([('GB','London'),('GB','Cambridge'),('US','New York')])

df = pd.DataFrame([london, cambridge, new_york], columns=['profit', 'revenue'], index=index)

df
```

		profit	revenue
GB	**London**	100	20
	Cambridge	200	30
US	**New York**	300	40

ROW MULTIINDEX

We select row subsets in a similar way:

```
df.loc["GB", :]
```

	profit	revenue
London	100	20
Cambridge	200	30

```
df.loc[("GB", "London"),:]
```

```
profit     100
revenue     20
Name: (GB, London), dtype: int64
```

STACK AND UNSTACK

- Stack: converts inner column MultiIndex to a row MultiIndex
- Unstack: converts inner row MultiIndex to a column MultiIndex

STACK

Consider the column MultiIndex we created earlier:

```
gb = [30, 40]
us = [100, 25]

columns = pd.MultiIndex.from_tuples([('profit','online'), ('profit','in-store')])
df = pd.DataFrame([gb,us], columns=columns, index=['GB','US'])
df
```

	profit	
	online	**in-store**
GB	30	40
US	100	25

`stack` will flatten "profit" into a single column, multi-indexed by rows "online" and "in-store".

STACK

```
df.stack()
```

		profit
GB	in-store	40
	online	30
US	in-store	25
	online	100

UNSTACK

Unstack will do the opposite, converting the row MultiIndex back to a column MultiIndex:

```
df = df.stack()
df.unstack()
```

	profit	
	in-store	**online**
GB	40	30
US	25	100

Practical 2 - MultiIndex, stack, unstack

practical2.ipynb

CROSSTAB

Use `crosstab` to compare aggregations of different values.

```python
df = pd.read_csv('data/profits_with_duplicates.csv')
df
```

	Country	Type	Profit
0	GB	online	50
1	US	in-store	25
2	GB	online	30
3	US	in-store	100
4	GB	in-store	25
5	GB	in-store	30
6	GB	online	75
7	US	online	110
8	US	in-store	10

e.g. `crosstab` could `sum` the profits for each country.

`crosstab` EXAMPLE

```
pd.crosstab(df['Country'], columns=df['Type'], values=df['Profit'], aggfunc='sum')
```

Type	**in-store**	**online**
Country		
GB	55	155
US	135	110

`cut`

Transform numerical data into categorical data

- group numbers into histogram bins
- each value is replaced by the number it bins to

```
a = np.linspace(0, 9, 10)
a
```

```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

- e.g. imagine we want to split these numbers up into two bins (<=5 and >5).

cut

```
binned = pd.cut(a, bins=2)
binned
```

```
[(-0.009, 4.5], (-0.009, 4.5], (-0.009, 4.5], (-0.009, 4.5], (-0.009, 4.
5], (4.5, 9.0], (4.5, 9.0], (4.5, 9.0], (4.5, 9.0], (4.5, 9.0]]
Categories (2, interval[float64]): [(-0.009, 4.5] < (4.5, 9.0]]
```

```
binned.categories
```

```
IntervalIndex([(-0.009, 4.5], (4.5, 9.0]],
              closed='right',
              dtype='interval[float64]')
```

This might look a bit strange at first. `pandas` has replaced every value in the list with the range it fits into (either 0 to 4.5 or 4.5 to 9). It also has a `categories` attribute which tells us what the two intervals are.

The range opens with a curved bracket and ends with a square bracket, with the square bracket meaning that that number is inclusive. So in the second bin 4.5 is exclusive, and 9.0 is inclusive. This means that the number 4.5 would be grouped into the first bin.

We can now use this numerical data as categorical data instead, or use it to easily create histograms.

factorize

- factorize splits data into a dataset codes, uniques, assigning a code to each unique value

```
codes, uniques = pd.factorize(['GB','GB','US','FR','GB','FR','FR'])
print(codes)
print(uniques)
```

```
[0 0 1 2 0 2 2]
['GB' 'US' 'FR']
```

```
uniques[codes[0]]
```

```
'GB'
```

Practical 3 - crosstab, cut, factorize

practical3.ipynb