PANDAS DATA PROCESSING

DATA TRANSFORMATION AND CLEANING

CONCATENATION

- The addition of one dataset to another
- Typically used to extend a dataset with extra rows or columns
- To achieve this we can use the pandas `.concat()` method

CONCATENATING DATAFRAMES

```
import numpy as np # we'll use this later
import pandas as pd

q1 = pd.read_csv('data/new-drugs-q1.csv')
q2 = pd.read_csv('data/new-drugs-q2.csv')
q3 = pd.read_csv('data/new-drugs-q3.csv')
```

Each file contains information about new prescription drugs made available in California during a given quarter of 2019 (based on data from OSHPD).

```
for q in [q1, q2, q3]:
    print(q.shape, q.columns)
```

```
(49, 4) Index(['NDC Number', 'Date Introduced to Market', 'Manufacturer Na
me',
       'Drug Product Description'],
      dtype='object')
(111, 4) Index(['NDC Number', 'Date Introduced to Market', 'Manufacturer N
ame',
       'Drug Product Description'],
      dtype='object')
(46, 4) Index(['NDC Number', 'Date Introduced to Market', 'Manufacturer Na
me',
       'Drug Product Description'],
      dtype='object')
```

- We have read the data and created a DataFrame from each CSV file
- We have confirmed that they contain the same columns in the same order

```
df = pd.concat([q1, q2, q3], axis=0)
```

```
print(df.shape)
df.head(3)
```

```
(206, 4)
```

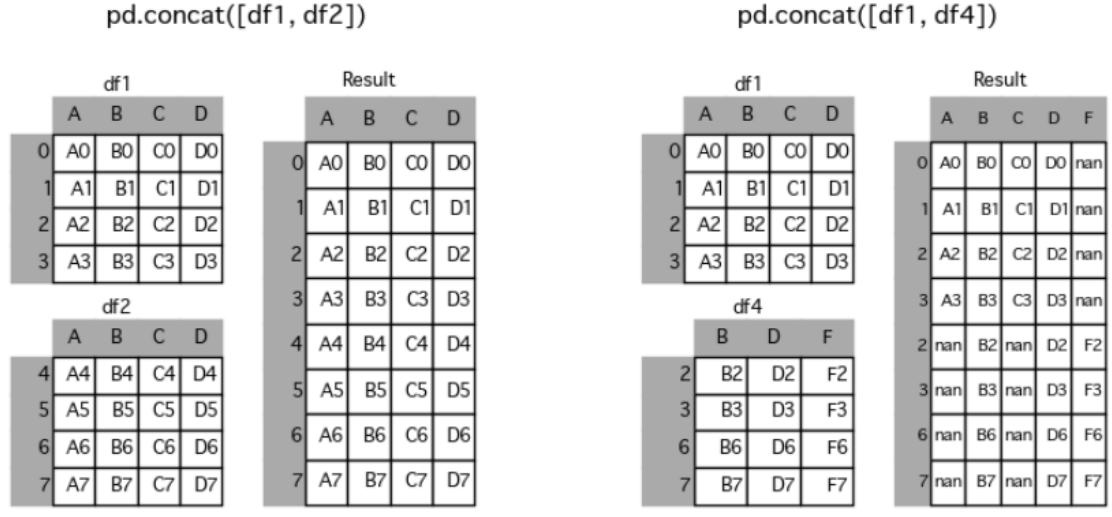
	NDC Number	Date Introduced to Market	Manufacturer Name	Drug Product Description
0	72626260101	2019-01-02	Asegua Therapeutics LLC	agLDV/SOF (ledipasvir 90 mg/sofosbuvir 400 mg)...
1	72626270101	2019-01-02	Asegua Therapeutics LLC	agSOF/VEL (sofosbuvir 400 mg/velpatasvir 100 m...
2	93765256	2019-01-03	Teva Pharmaceuticals USA	VARDENAFIL HCL TABLETS 2.5MG 30

- We used the pandas `.concat()` method, passing a list of DataFrames as the only argument
- The `axis` parameter determines whether to concatenate along rows or columns
    - The default `0` is used here to combine rows from DataFrames with shared column names
    - `axis=1` would be used to extend a dataset with additional columns

```
df.index
```

```
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
            ...
            36, 37, 38, 39, 40, 41, 42, 43, 44, 45],
           dtype='int64', length=206)
```

- Notice that the index labels are unchanged from what they were in each individual DataFrame, i.e. there are duplicated values
- We can create new unique row index labels if we want to by passing `ignore_index=True` to the `.concat()` method


pd.concat([df1, df2])
df1
A B C D
0 A0 B0 C0 D0
1 A1 B1 C1 D1
2 A2 B2 C2 D2
3 A3 B3 C3 D3
df2
A B C D
4 A4 B4 C4 D4
5 A5 B5 C5 D5
6 A6 B6 C6 D6
7 A7 B7 C7 D7
Result
A B C D
0 A0 B0 C0 D0
1 A1 B1 C1 D1
2 A2 B2 C2 D2
3 A3 B3 C3 D3
4 A4 B4 C4 D4
5 A5 B5 C5 D5
6 A6 B6 C6 D6
7 A7 B7 C7 D7

pd.concat([df1, df4])
df1
A B C D
0 A0 B0 C0 D0
1 A1 B1 C1 D1
2 A2 B2 C2 D2
3 A3 B3 C3 D3
df4
B D F
2 B2 D2 F2
3 B3 D3 F3
6 B6 D6 F6
7 B7 D7 F7
Result
A B C D F
0 A0 B0 C0 D0 nan
1 A1 B1 C1 D1 nan
2 A2 B2 C2 D2 nan
3 A3 B3 C3 D3 nan
2 nan B2 nan D2 F2
3 nan B3 nan D3 F3
6 nan B6 nan D6 F6
7 nan B7 nan D7 F7

```
df['NDC Number'].value_counts().max()
```

```
1
```

```
df_new = df.set_index('NDC Number')
df_new.head(2)
```

NDC Number	Date Introduced to Market	Manufacturer Name	Drug Product Description
72626260101	2019-01-02	Asegua Therapeutics LLC	agLDV/SOF (ledipasvir 90 mg/sofosbuvir 400 mg)...
72626270101	2019-01-02	Asegua Therapeutics LLC	agSOF/VEL (sofosbuvir 400 mg/velpatasvir 100 m...

- We can see from using the `.value_counts()` Series method that `NDC Number` contains no duplicate values
- We used the `.set_index()` method to use `NDC Number` as our index in a new DataFrame assigned to `df_new`

JOINING DATASETS

- We often need to join (combine) datasets which have some relationship with one another
- The relationship (or association) requires a common key in each dataset so that they can be combined

one-to-one joins

Each dataset contains the same number of shared, unique values in the key

many-to-one joins

The first dataset has numerous instances of one or more of the values in the key while the second datset only has one instance of each value

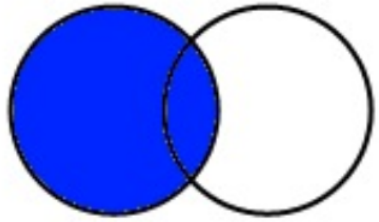
many-to-many joins

Both datasets have numerous instances of one or more of the values in the key

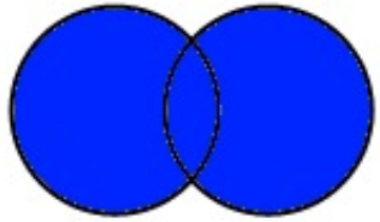
USING THE PANDAS `.merge()` METHOD

pandas uses terminology borrowed from SQL (a popular language for querying databases) in the syntax for its methods which provide functionality for joining datasets.

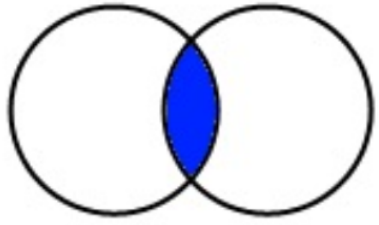
LEFT JOIN



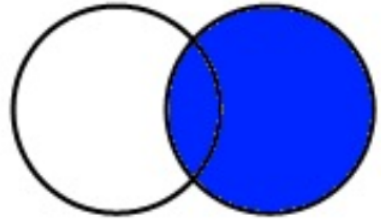
FULL OUTER JOIN



INNER JOIN



RIGHT JOIN



EXAMPLES OF DIFFERENT JOINS WITH SMALL DATASETS

There are several small DataFrames created in the `dataframes.py` file; let's take a look at two of them and then see how they can be merged:

```
from dataframes import students, residents

display(students, residents)
```

	Name	Subject
0	Jesse	Physics
1	Kotryna	Biochemistry
2	Xiaoyi	Chemistry
3	David	Medicine

	Name	Age
0	Jesse	21
1	Kotryna	22
2	Xiaoyi	23
3	Raoul	29

We would like to add the `Age` column data to the `students` table:

```
students.merge(residents, how="inner", left_on="Name", right_on="Name")
```

	Name	Subject	Age
0	Jesse	Physics	21
1	Kotryna	Biochemistry	22
2	Xiaoyi	Chemistry	23

- `how=inner` means that only matches from the both tables are retained; no details for `Raoul` or `David` are used
- the `left_on` and `right_on` arguments are used to identify the column on which to join the tables
    - although here the column name is the same, they could be different in other scenarios

If we wanted to retain all of the names from the right table (`residents`), we can use `how="right"` :

```
students.merge(residents, how="right")
```

	Name	Subject	Age
0	Jesse	Physics	21
1	Kotryna	Biochemistry	22
2	Xiaoyi	Chemistry	23
3	Raoul	NaN	29

- There is no data for `Subject` for `Raoul`, so this is returned as a `NaN` (missing value)

A similar result could be achieved by reversing the table order and using `left` instead of `right`:

```
residents.merge(students, how="left")
```

	Name	Age	Subject
0	Jesse	21	Physics
1	Kotryna	22	Biochemistry
2	Xiaoyi	23	Chemistry
3	Raoul	29	NaN

- The only difference here is the column order
    - `Age` precedes `Subject` because `Age` was part of the `left` table, i.e. the table specified in the `how` argument

Outer joins will retain data found in one table but not the other.
The `indicator` parameter is useful if we want to see which of the original tables the data in the other columns cam from:

```
residents.merge(students, how="outer", indicator=True)
```

	Name	Age	Subject	_merge
0	Jesse	21.0	Physics	both
1	Kotryna	22.0	Biochemistry	both
2	Xiaoyi	23.0	Chemistry	both
3	Raoul	29.0	NaN	left_only
4	David	NaN	Medicine	right_only

JOINING DATAFRAMES USING EACH INDEX AS THE KEY

```
data_a = pd.read_csv('data/drugs-data-a.csv', index_col='NDC Number')
display(data_a.head(2))
data_a.shape
```

NDC Number	Date Introduced to Market	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Date	Acquisition Price	Acquisition Price Non-Public Indicator
47335093640	2019-03-01	705.67	1.0	NaN	NaN	NaN	NaN	NaN	NaN
47335023683	2019-04-25	7500.00	NaN	0.0	NaN	NaN	NaN	NaN	NaN

```
(79, 9)
```

- Notice how we used the `index_col` parameter with the pandas `read_csv()` method to use the values in the `NDC Number` column as our index
- The `.shape` DataFrame attribute tells us that there are less rows in `data_a` than our previous DataFrame `df_new`

```
df_extra_on_index = df_new.merge(data_a, how='left', left_index=True, right_index=True)
df_extra_on_index.head(2)
```

NDC Number	Date Introduced to Market_x	Manufacturer Name	Drug Product Description	Date Introduced to Market_y	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	I
72626260101	2019-01-02	Asegua Therapeutics LLC	agLDV/SOF (ledipasvir 90 mg/sofosbuvir 400 mg)...	NaN	NaN	NaN	NaN	NaN	
		Asegua	agSOF/VEL (sofosbuvir 400						

- Here we used a LEFT JOIN (`how=left`), since we want to retain all data in the original DataFrame `df_new` and supplement it with associated data from the DataFrame `data_a`
- We used `left_index=True` and `right_index=True` to specify the `index` of each DataFrame as the key on which they will be joined
- We can see that the default values for the `suffix` parameter have been used, since there was a column in both of the DataFrames labelled `Date Introduced to Market`

JOINING DATAFRAMES USING MULTIPLE COLUMNS AS THE KEY

```
df_extra = df_new.merge(data_a, how='left', on=['NDC Number', 'Date Introduced to Ma
rket'])
display(df_extra.head(2))
df_extra.shape
```

NDC Number	Date Introduced to Market	Manufacturer Name	Drug Product Description	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Date	Acquis Pri
72626260101	2019-01-02	Asegua Therapeutics LLC	agLDV/SOF (ledipasvir 90 mg/sofosbuvir 400 mg)...	NaN	NaN	NaN	NaN	NaN	NaN	Na
		Asegua	agSOF/VEL (sofosbuvir 400							

```
(206, 11)
```

- Here we have used the `on` parameter to provide a list of column labels which are present in both DataFrames
- Notice that this list can include the row index name (in this case, `NDC Number`)
- By including `Date Introduced to Market` in the key, we only see one instance of the column in the resulting DataFrame

INNER JOIN

```
df_extra_inner = df_new.merge(data_a, on=['NDC Number', 'Date Introduced to Market'
])
display(df_extra_inner.head(2))
df_extra_inner.shape
```

NDC Number	Date Introduced to Market	Manufacturer Name	Drug Product Description	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Date	Acquisitio Price
93765256	2019-01-03	Teva Pharmaceuticals USA	VARDENAFIL HCL TABLETS 2.5MG 30	704.59	1.0	101361.0	NaN	NaN	NaN	NaN
		Teva	VARDENAFIL HCL							

```
(79, 11)
```

- Using `.merge()` with the default `how` parameter results in an INNER JOIN
    - The resulting DataFrame has fewer rows; there were 79 rows with matching keys

RIGHT JOIN

```
df_extra_right = df_new.merge(data_a, how='right', on=['NDC Number', 'Date Introduced to Market'])
display(df_extra_right.head(2))
df_extra_right.shape
```

NDC Number	Date Introduced to Market	Manufacturer Name	Drug Product Description	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Date	Acquisitio Price
93765256	2019-01-03	Teva Pharmaceuticals USA	VARDENAFIL HCL TABLETS 2.5MG 30	704.59	1.0	101361.0	NaN	NaN	NaN	NaN
		Teva	VARDENAFIL HCL							

(79, 11)

- Notice how in this instance the resulting DataFrame from a RIGHT JOIN is the same as that returned when using an INNER JOIN
    - This is to be expected if all of the keys in the right DataFrame are present in the left DataFrame

```
pd.merge(left, right, how='left', on=['key1', 'key2'
])
```

These keys are only found in **left** so have NaN values in columns C and D in **Result**

This key is found once in both **left** and **right**, so is found once in the **Result**

This key is found twice in **right**, so is found twice in **Result** with the same values (from the same entry in **left**) in columns A and B

This row in **right** has no matching key in **left**, so is dropped

PANDAS .MERGE() IN DETAIL

Let's work through the arguments (args) and keyword arguments (kwargs) for the DataFrame `.merge()` method found in the documentation:

```
DataFrame.merge(self, right, how='inner',
                on=None, left_on=None, right_on=None,
                left_index=False, right_index=False,
                sort=False, suffixes=('_x', '_y'),
                copy=True, indicator=False, validate=
None)
```

```
DataFrame.merge(self, right,
```

- `self` refers to the DataFrame on which the method is being called, and is passed automatically to it; this DataFrame can be considered the left circle in each of the previous Venn diagrams
- `right` refers to the other DataFrame which we want to join with the original (left) DataFrame, and is represented by the right circle in the diagrams

```
DataFrame.merge(self, right, how='inner',
```

- how is the first optional parameter or keyword argument, all of which have default values
- The value given for how will determine how pandas attempts to join the two DataFrames
- In this case inner is the default value, which means that pandas will attempt an inner join

```
DataFrame.merge(self, right, how='inner',
                on=None, left_on=None, right_on=None,
                left_index=False, right_index=False,
```

- The next parameters tell pandas which column(s) in each DataFrame contain(s) the key with which we want to join them
    - `on`, `left_on` and `right_on` can all take either a single label or a list of labels; where more than one label is used, the values in all given columns in both DataFrames must match for rows to be associated
    - `left_index` and `right_index` take Boolean values

- `on` gives us a way to associate the DataFrames using a single argument; useful if the key is in columns or indexes with the same label in both DataFrames
- if we don't use `on`, then we need to provide the key for each DataFrame separately:
    - For the original (left) DataFrame provide either `left_on=` with label(s) or `left_index=True`
    - For the additional (right) DataFrame provide either `right_on=` with label(s) or `right_index=True`

```
DataFrame.merge(self, right, how='inner',
                on=None, left_on=None, right_on=None,
                left_index=False, right_index=False,
                sort=False, suffixes=('_x', '_y'),
```

- `sort=True` would sort the resulting DataFrame by the key
- `suffixes` will append the given strings to any column labels present in both DataFrames (but not part of the key) so that they can be distinguished in the new DataFrame
    - Consider adding these columns to the key if they contain identical values; they will then only appear once in the new DataFrame

```
DataFrame.merge(self, right, how='inner',
                on=None, left_on=None, right_on=None,
                left_index=False, right_index=False,
                sort=False, suffixes=('_x', '_y'),
                copy=True, indicator=False, validate=
None)
```

- It's less likely that you will want to modify these keyword arguments, but:
    - `copy=False` could be used help to save memory usage
    - `indicator=True` adds a column giving information about the source of each row in relation to the join
    - `validate` allows checking of whether the merge is of a specified type, such as one-to-many, many-to-one

- As always, remember that you can and should refer to the documentation if:
    - You have an unusual scenario to deal with or your output is not as expected
    - You regularly use a given method - the optional parameters often provide a quick way to carry out common further processing tasks

Part 1: Merging datasets

processing-pandas-workbook.ipynb

DATA PREPARATION

Data pre-processing or cleaning is often required to get our raw data into a more usable state:

- Removal of data not required for our task
- Conversion of values to an appropriate data type
- Checking for missing values and fixing errors

REMOVAL OF SUPERFLUOUS DATA

We can use methods and syntax previously seen to reduce the number of rows and columns in our dataset.

```
df_data = pd.read_csv('data/drugs.csv', index_col='NDC Number')
df_data.head(1)
```

NDC Number	Manufacturer Name	Drug Product Description	Date Introduced to Market	WAC at Introduction	Marketing/Pricing Plan Description	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakth Ther Indic
47335093640	SUN PHARMACEUTICALS	Leuprolide Acetate Injection 1Mg/0.2Ml, 2.8Ml	2019-03-01	705.67	NaN	1.0	NaN	Na

```
cols_to_drop = ['Date Introduced to Market', 'Acquisition Date', \
                'Acquisition Price','Marketing/Pricing Plan Description',\
                'Acquisition Price Comment', 'General Comments', \
                'Supporting Documents']

df_cols = df_data.drop(cols_to_drop, axis=1)
df_cols.head(1)
```

NDC Number	Manufacturer Name	Drug Product Description	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Price Non-Public Indicator
47335093640	SUN PHARMACEUTICALS	Leuprolide Acetate Injection 1Mg/0.2Ml, 2.8Ml	705.67	1.0	NaN	NaN	NaN	NaN

- The DataFrame `.drop()` method allows us to drop any columns (`axis=1`) which are not required

```
df_sub = df_cols[df_cols['Manufacturer Name']!='Kyowa Kirin, Inc.'].copy()
display(df_cols.shape)
df_sub.shape
```

```
(206, 8)
```

```
(204, 8)
```

- We can remove rows which have particular values in a given column
- Using `.copy()` ensures that `df_sub` is a distinct object in memory and that subsequent changes to it will not affect the original DataFrame

MAKING DATA MORE USABLE

We may encounter datasets where particular states or values are represented in a way which are not suitable or optimal for analysis using our chosen tools (such as pandas and Python), for example:

- Boolean values are represented differently (`Yes` | `No` or `1` | `0`)
- Percentages have inconsistent formatting (`0.42` or `42%`)
- Ambiguous dates have been misinterpreted (`dd-mm-yy` or `mm-dd-yy`)

After examining the dataset we notice that all of the `Indicator` columns contain values which are either `1.0` or `NaN`; we decide that replacing the `NaN` values with zeros will help with our analysis:

```
df_sub.columns
```

```
Index(['Manufacturer Name', 'Drug Product Description', 'WAC at Introducti
on',
       'Marketing/Pricing Plan Non-Public Indicator',
       'Estimated Number of Patients', 'Breakthrough Therapy Indicator',
       'Priority Review Indicator', 'Acquisition Price Non-Public Indicato
r'],
      dtype='object')
```

```
indicator_columns = df_sub.columns[df_sub.columns.str.contains("Indicator")]
df_sub[indicator_columns] = df_sub[indicator_columns].fillna(0).astype(int)
df_sub.head(2)
```

	Manufacturer Name	Drug Product Description	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Price Non-Public Indicator
NDC Number								
47335093640	SUN PHARMACEUTICALS	Leuprolide Acetate Injection 1Mg/0.2Ml, 2.8Ml	705.67	1	NaN	0	0	0
		Ambrisentan						

- We used the `.columns` attribute to access the column labels, and the `.contains()` method on the string (`.str`) of each one to identify those which contain `'Indicator'`
- The `.fillna()` method replaced the `NaN` values with zeros in the identified `indicator_columns`
- `.astype(int)` makes our DataFrame more readable and less ambiguous

We also notice that in the `Estimated Number of Patients` column we see both `NaN` values and `0.0` values:

```
display(df_sub['Estimated Number of Patients'].isna().sum())
df_sub[df_sub['Estimated Number of Patients'] == 0].shape[0]
```

```
73
```

```
25
```

We determine that the `0.0` values should in fact be `NaN` values, because we assume they must be missing (rather than there being an expectation at `Estimated Number of Patients` will actually be zero):

```
df_sub['Estimated Number of Patients'] = df_sub['Estimated Number of Patients'].repl
ace(0, np.nan)
df_clean = df_sub.copy()
df_clean.head(3)
```

	Manufacturer Name	Drug Product Description	WAC at Introduction	Marketing/Pricing Plan Non-Public Indicator	Estimated Number of Patients	Breakthrough Therapy Indicator	Priority Review Indicator	Acquisition Price Non-Public Indicator
NDC Number								
47335093640	SUN PHARMACEUTICALS	Leuprolide Acetate Injection 1Mg/0.2Ml, 2.8Ml	705.67	1	NaN	0	0	0
		Ambrisentan						

- We used the `.replace()` method to replace zeros with `NaN` values, which can be created using `np.nan`
    - `np` is the alias we used when importing `numpy` earlier
    - the `.nan` attribute defines `NaN` values
- We assigned a copy of the cleaned DataFrame to `df_clean`

Part 2: Data preparation and cleaning

processing-pandas-workbook.ipynb

USING THE PANDAS `.apply()` METHOD

The `.apply()` method allows us to apply our own functions to our data

- Typically this is used to create a new column or update an existing one with the results of calling the function with an existing column of values

```
from dataframes import dimensions
dimensions
```

	length (cm)	width (cm)	length (m)	width (m)	area (m2)
0	500	450	5.0	4.5	22.5
1	220	250	2.2	2.5	5.5
2	150	800	1.5	8.0	12.0

We'd like to have the dimensions in metres rather than centimetres. Here's a function we can apply:

```
def cm_to_m(cm):
    return cm / 100
```

```
dimensions[['length (m)', 'width (m)']] = dimensions[['length (cm)', 'width (cm)']].
apply(cm_to_m)
dimensions
```

	length (cm)	width (cm)	length (m)	width (m)	area (m2)
0	500	450	5.0	4.5	22.5
1	220	250	2.2	2.5	5.5
2	150	800	1.5	8.0	12.0

- We have applied the function to two columns, assigning the results to two new columns

We can also use `.apply()` using multiple values from a given row of a DataFrame:

```
def area(row):
    return row['length (m)'] * row['width (m)']
```

```
dimensions['area (m2)'] = dimensions.apply(area, axis=1)
dimensions
```

	length (cm)	width (cm)	length (m)	width (m)	area (m2)
0	500	450	5.0	4.5	22.5
1	220	250	2.2	2.5	5.5
2	150	800	1.5	8.0	12.0

- Notice that here the function relies on those columns being present in the DataFrame
- `axis=1` is requred to do this in a column-wise manner

In our drugs example, we notice that we have several instances where multiple entries in `Drug Product Description` refer to the same drug, but wiith differing dosage levels. We create a function which returns only the first word:

```
def first_word(description):

    return description.split(' ')[0] if ' ' in description else description
```

- This function has a single parameter `description` and needs to be called with a string
- The `.split()` method will split the string wherever there is a space (`' '`) and then the first element (`[0]`) in the resulting list will be accessed
    - In the absence of a space,`description` will be returned

```
df_clean['Short Description'] = df_clean['Drug Product Description'].apply(first_wor
d)
```

- We then used the `.apply()` method to apply our `first_word` function to `Drug Product Description`, creating the new column `Short Description`

USING THE `.nunique()` METHOD

The `.nunique()` method allows us to identify the number of unique values in a Series:

```
df_clean['Drug Product Description'].nunique()
```

```
196
```

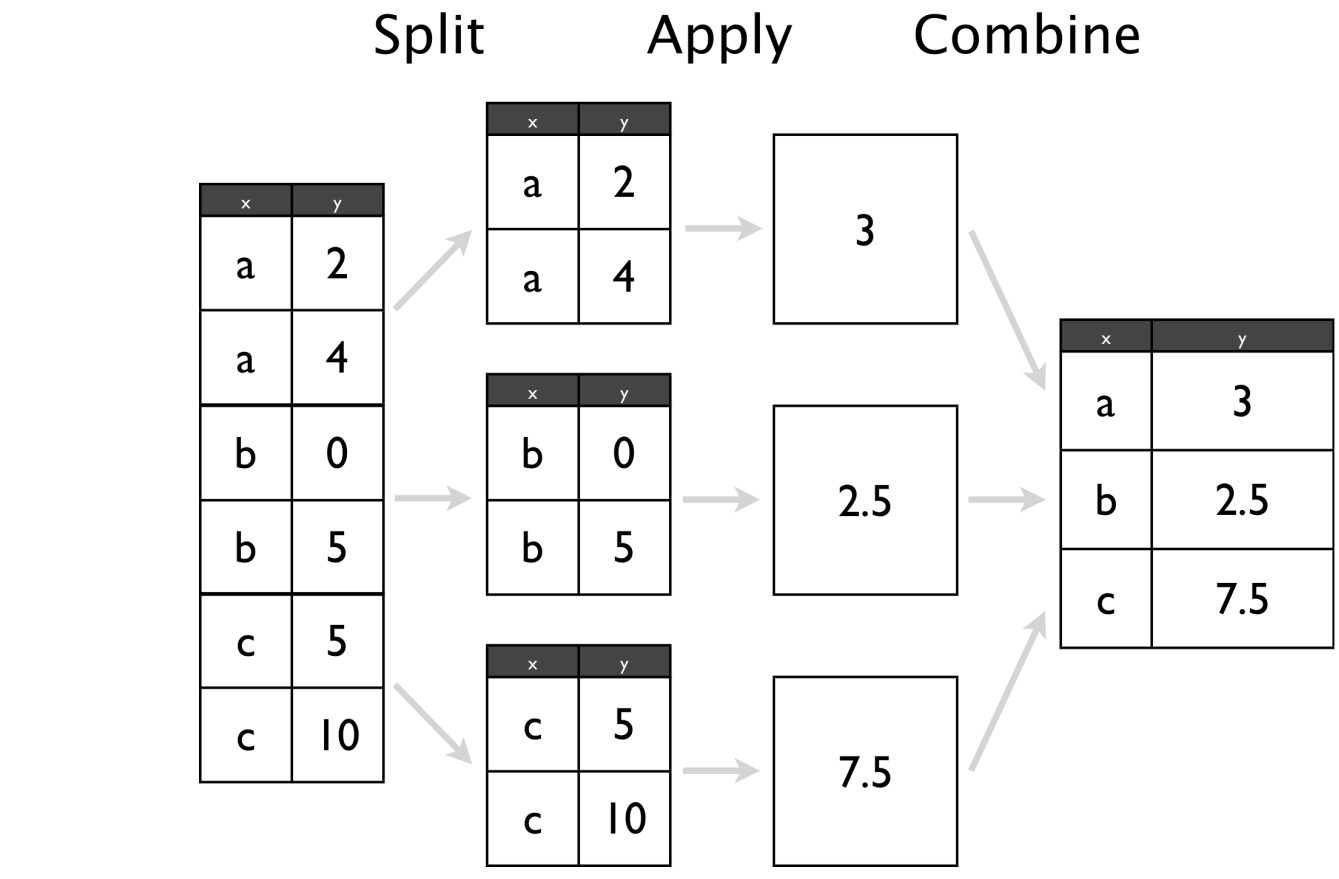
```
df_clean['Short Description'].nunique()
```

```
87
```

In the example, we can see that our new `Short Description` column has less than half the number of unique values.

DATA GROUPING AND AGGREGATION

- We often need to calculate metrics for subsets (or groups) of our data
- A dataset can be split into groups of rows with common values in a given column
- Calculations can be applied to all groups simultaneously
- The results of these calculations can then be combined back together



Source: Github

USING THE PANDAS .groupby() DATAFRAME METHOD

```
sac = pd.DataFrame({'x': ['a', 'a', 'b', 'b', 'c', 'c'], 'y': [2, 4, 0, 5, 5, 10]})
sac
```

	x	y
0	a	2
1	a	4
2	b	0
3	b	5
4	c	5
5	c	10

```
sac.groupby('x')[['y']].mean()
```

	y
x	
a	3.0
b	2.5
c	7.5

with the `sac` DataFrame, `groupby` column `x` and calculate for column(s) `['y']` the `mean` of each group

- In this particular example, the result would be the same with the omission of `[['y']]`; if no columns are specified, the function is applied to all columns in the DataFrame
- Using [single parentheses] is possible when only specifying a single column to perform the operation on, but will result in a Series rather than a DataFrame being returned

You're unlikely to need to use the following code snippets in isolation, but let's examine each to help clarify Split - Apply - Combine:

```
gb = sac.groupby('x')
display(gb)
len(gb)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fe1208e4b10>
```

```
3
```

- the .groupby() method created a groupby object, which has a length of 3, i.e. the number of groups it holds (a, b, and c)

```
gb.get_group('a')
```

	x	y
0	a	2
1	a	4

- The `get_group()` method of a groupby object allows us to access each individual group which has been created

```
gb.get_group('a').mean()
```

```
y    3.0
dtype: float64
```

- The original .groupby() statement applies an operation such as the .mean() method used above to each group and combines the results into a new DataFrame

Another example showing a different method (`.count()`) being applied, using a different column to group by:

```
sac.groupby('y')[['x']].count()
```

	x
y	
0	1
2	1
4	1
5	2
10	1

- The row index labels show the unique values in column `y` of the original DataFrame, by which the values for column `x` have been grouped

Here's an example which applies the `.sum()` method to a subset of columns, with rows grouped by `Manufacturer Name`:

```
indicators = df_clean.groupby('Manufacturer Name')\
            [['Priority Review Indicator',
              'Breakthrough Therapy Indicator',
              'Marketing/Pricing Plan Non-Public Indicator',
              'Acquisition Price Non-Public Indicator']]\
            .sum()

indicators.tail(3)
```

	Priority Review Indicator	Breakthrough Therapy Indicator	Marketing/Pricing Plan Non-Public Indicator	Acquisition Price Non-Public Indicator
Manufacturer Name				
Valeant Pharmaceuticals North America, LLC	0	0	1	0
ViiV Healthcare	0	0	1	0
Zydus Pharmaceuticals (USA) Inc.	0	0	8	0

```
indicators.columns=[ 'Priority', 'Breakthrough', 'Marketing', 'Acquisition']
ind_total = indicators.sort_values(by=['Priority'], ascending=False)
ind_total.head(5)
```

Manufacturer Name	Priority	Breakthrough	Marketing	Acquisition
AveXis	22	22	0	0
Janssen	7	7	0	0
Teva Pharmaceuticals USA	6	0	29	0
Karyopharm Therapeutics Inc.	4	0	4	0
Paratek Pharmaceuticals, Inc.	4	0	4	0

- Here we have updated the column labels and used the `sort_values()` method to improve readability

USING THE PANDAS .agg() METHOD

```
ind_total.agg(['sum', 'mean'])
```

	Priority	Breakthrough	Marketing	Acquisition
sum	64.000000	42.000000	132.000000	9.000000
mean	1.391304	0.913043	2.869565	0.195652

- Using .agg() on a DataFrame applies the functions in the [list] on every column (the axis parameter has a default value of 0)

```
ind_total.agg(['sum'], axis=1).sort_values(by='sum', ascending=False)\
         .rename(columns={'sum':'Total Indicators'}).head(3)
```

	Total Indicators
AveXis	44
Teva Pharmaceuticals USA	35
Par Pharmaceutical	15

- `axis=1` applies the function(s) to each row
- The function name is used for the resulting column label by default; here we used the `.rename()` method to update it

COMBINING `.groupby()` WITH `.agg()`

```
manu_agg = df_clean.groupby('Manufacturer Name')\
                   .agg(entries=('Manufacturer Name', 'size'), \
                        patient_estimates=('Estimated Number of Patients', 'count'))

manu_agg['missing_estimates'] = manu_agg['entries'] - manu_agg['patient_estimates']

manu_agg.sort_values(by='missing_estimates', ascending=False).head(3)
```

	entries	patient_estimates	missing_estimates
Manufacturer Name			
SUN PHARMACEUTICALS	10	0	10
Zydus Pharmaceuticals (USA) Inc.	8	0	8
EMD Serono, Inc.	8	0	8

- Here we have used `.agg()` to apply multiple functions to a `groupby()` object
- Notice how each element in the tuple passed to `.agg()` is constructed as follows:

```
result_column_name=('source_column', 'functio
n')
```

- 'size' returns the number of values in the given column, including NaN values (as such, the choice of column on which to apply it is unimportant, since all columns in a given DataFrame will have the same number)
- 'count' excludes NaN values, so here the difference between the columns in manu_agg tells us how many entries do not have patient_estimates

Part 3: Data grouping and aggregation

processing-pandas-workbook.ipynb

IDENTIFYING AND FIXING UNUSUAL DATA ISSUES

At some point we may encounter unexpected results in the output of our code

- in practice these may only be identified after further work with the dataset
- we need to identify and isolate potential causes of the issue
- we may need to do some manual updating to solve the problem
- if the issue is likely to recur, consider discussing with the author of the data source

PRACTICAL EXAMPLE

When creating this notebook, at some point we noticed that some entries in `Short Description` had not been shortened as expected:

```
df_clean[df_clean['Short Description'].str.contains('Cinacalcet')][
                ['Drug Product Description', 'Short Description']]
```

NDC Number	Drug Product Description	Short Description
47335037983	Cinacalcet HCL Oral Tablet 30MG	Cinacalcet
47335038083	Cinacalcet HCL Oral Tablet 60MG	Cinacalcet
47335060083	Cinacalcet HCL Oral Tablet 90MG	Cinacalcet
378619793	Cinacalcet Hydrochloride Tablets, 30mg, 30s	Cinacalcet
378619693	Cinacalcet Hydrochloride Tablets, 60mg, 30s	Cinacalcet
378619593	Cinacalcet Hydrochloride Tablets, 90mg, 30s	Cinacalcet
67877050330	Cinacalcet 30mg 30 Tabs	Cinacalcet 30mg 30 Tabs
67877050430	Cinacalcet 60mg 30 Tabs	Cinacalcet 60mg 30 Tabs

```
df_clean.loc[67877050530, 'Drug Product Description']
```

```
'Cinacalcet\xa090mg\xa030\xa0Tabs'
```

- Having checked our code for the `first_word()` function we used earlier looks ok, we used `.loc()` to look at an example of a specific value which was not being processed as expected
- We can see that, in place of spaces, we see instances of `\xa0`, which are not visible when the DataFrame is displayed

Finding help with unusual issues

Turning to a Google search for help, we can find more information:

- A very helpful Stack Overflow page with some possible solutions
- Some understanding of what kinds of characters may cause such problems

The top answer on the Stack Overflow page is in fact all we need:

```
string = string.replace(u'\xa0', u' ')
```

```
df_clean['Drug Product Description'] = df_clean['Drug Product Description'].str.repl
ace(u'\xa0', u' ')
df_clean['Short Description'] = df_clean['Drug Product Description'].apply(first_wor
d)
df_clean.loc[67877050530, 'Short Description']
```

```
'Cinacalcet'
```

We have fixed our problem by:

- Checking our code first
- Searching for help online
- Adapting some code we found

We have learned that sometimes issues can occur due to the encoding of unusual characters; next time we will know that this can be an issue and how we might go about resolving it.

THE IMPORTANCE OF CONTEXT AND DOMAIN KNOWLEDGE

When using data from third parties, always keep in mind that the author may not have produced the dataset for the purposes you intend to use it for.

- Before working with a dataset, do what you can to understand why it was created and how the data was collected
- When working with a dataset, be alert for unusual patterns, inconsistencies, and missing data

PRACTICAL EXAMPLE

Consider the list of results and values for the entries with a `Short Description` of `Cinacalcet` (only the last 5 are shown below):

```
df_clean[df_clean['Short Description'] == 'Cinacalcet'].tail()\
        [['Short Description', 'Manufacturer Name', 'Drug Product Description', \
          'WAC at Introduction', 'Estimated Number of Patients']]
```

NDC Number	Short Description	Manufacturer Name	Drug Product Description	WAC at Introduction	Estimated Number of Patients
378619693	Cinacalcet	Mylan Pharmaceuticals Inc	Cinacalcet Hydrochloride Tablets, 60mg, 30s	1371.39	NaN
378619593	Cinacalcet	Mylan Pharmaceuticals Inc	Cinacalcet Hydrochloride Tablets, 90mg, 30s	2057.09	NaN
67877050330	Cinacalcet	Ascend Laboratories, LLC	Cinacalcet 30mg 30 Tabs	685.50	468000.0
67877050430	Cinacalcet	Ascend Laboratories,	Cinacalcet 60mg 30 Tabs	1371.25	468000.0

Notice that we have entries with:

- A different `Manufacturer Name` and a similar `Drug Product Description`
  ... *Can they be treated as the same drug?*
- Similar but not identical values for `WAC at Introduction`
  ... *Should these be considered to be equal?*
- Matching values for `Estimated Number of Patients` for the same drug at different doses
  ... *Is the total the sum of the values or just one of them?*

We need to understand the underlying data, and the context in which it has been collected.