Table of Contents

Data Wrangling with Pandas

Now that we have been exposed to the basic functionality of Pandas, let's explore some more advanced features that will be useful when addressing more complex data management tasks.

As most statisticians/data analysts will admit, often the lion's share of the time spent implementing an analysis is devoted to preparing the data itself, rather than to coding or running a particular model that uses the data. This is where Pandas and Python's standard library are beneficial, providing high-level, flexible, and efficient tools for manipulating your data as needed.

In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context('notebook')
```

Date/Time data handling

Date and time data are inherently problematic. There are an unequal number of days in every month, an unequal number of days in a year (due to leap years), and time zones that vary over space. Yet information about time is essential in many analyses, particularly in the case of time series analysis.

The `datetime` built-in library handles temporal information down to the nanosecond.

In [2]:
```
from datetime import datetime, date, time
```

In [3]:
```
now = datetime.now()
now
```

Out[3]:
```
datetime.datetime(2023, 9, 25, 11, 19, 58, 759944)
```

In [4]:
```
date(1970, 9, 3)
```

Out[4]:
```
datetime.date(1970, 9, 3)
```

Having a custom data type for dates and times is convenient because we can perform operations on them easily. For example, we may want to calculate the difference between two times:

In [5]:
```
my_age = now - datetime(1970, 1, 1)
my_age.days/365
```

Out[5]:
```
53.76712328767123
```

In this section, we will manipulate data collected from ocean-going vessels on the eastern seaboard. Vessel operations are monitored using the Automatic Identification System (AIS), a safety at sea navigation technology which vessels are required to maintain and that uses transponders to transmit very high frequency (VHF) radio signals containing static information including ship name, call sign, and country of origin, as well as dynamic information unique to a particular voyage such as vessel location, heading, and speed.

The International Maritime Organization's (IMO) International Convention for the Safety of Life at Sea requires functioning AIS capabilities on all vessels 300 gross tons or greater and the US Coast Guard requires AIS on nearly all vessels sailing in U.S. waters. The Coast Guard has established a national network of AIS receivers that provides coverage of nearly all U.S. waters. AIS signals are transmitted several times each minute and the network is capable of handling thousands of reports per minute and updates as often as every two seconds. Therefore, a typical voyage in our study might include the transmission of hundreds or thousands of AIS encoded signals. This provides a rich source of spatial data that includes both spatial and temporal information.

For our purposes, we will use summarized data that describes the transit of a given vessel through a particular administrative area. The data includes the start and end time of the transit segment, as well as information about the speed of the vessel, how far it travelled, etc.

In [6]:
```
segments = pd.read_csv("Data/transit_segments.csv")
segments.head()
```

Out[6]:

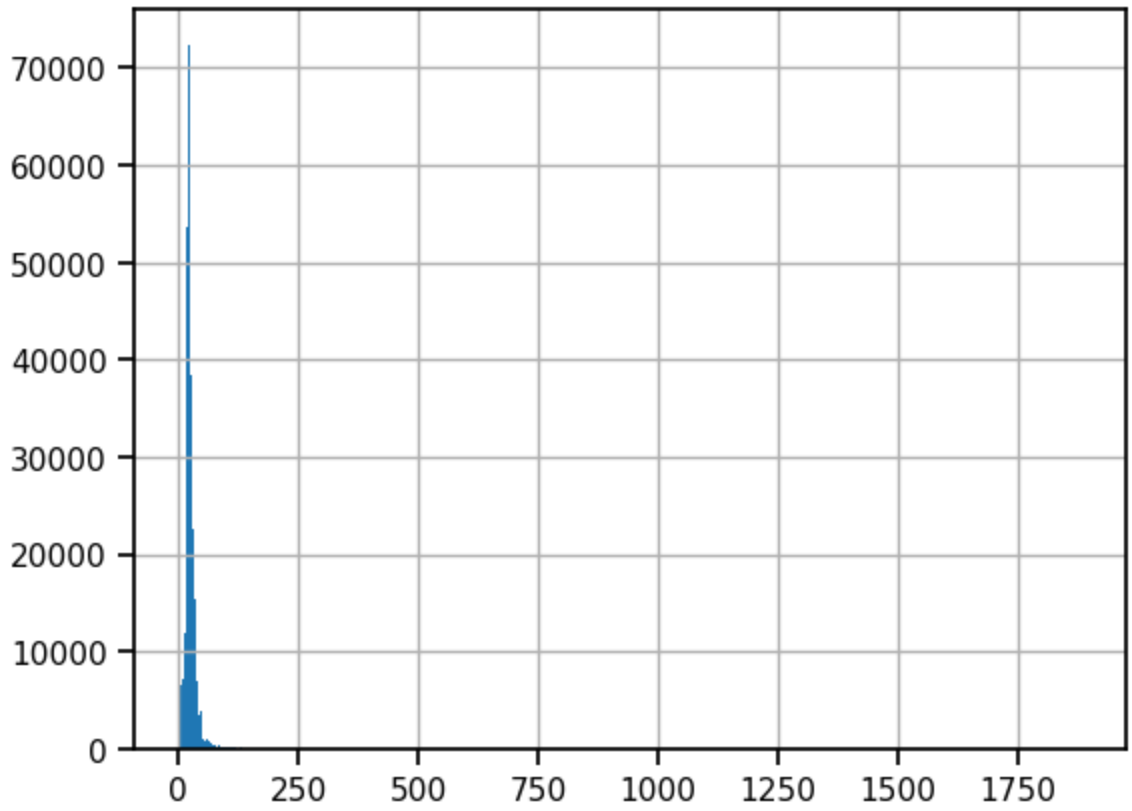
	mmsi	name	transit	segment	seg_length	avg_sog	min_sog	max_sog	pdgt10	st_time	end_time
0	1	Us Govt Ves	1	1	5.1	13.2	9.2	14.5	96.5	2/10/09 16:03	2/10/09 16:27
1	1	Dredge Capt Frank	1	1	13.5	18.6	10.4	20.6	100.0	4/6/09 14:31	4/6/09 15:20
2	1	Us Gov Vessel	1	1	4.3	16.2	10.3	20.5	100.0	4/6/09 14:36	4/6/09 14:55
3	1	Us Gov Vessel	2	1	9.2	15.4	14.5	16.1	100.0	4/10/09 17:58	4/10/09 18:34

4	1	Dredge Capt Frank	2	1	9.2	15.4	14.6	16.2	100.0	4/10/09 17:59	4/10/09 18:35

For example, we might be interested in the distribution of transit lengths, so we can plot them as a histogram:

In [7]:
```
segments.seg_length.hist(bins=500)
```

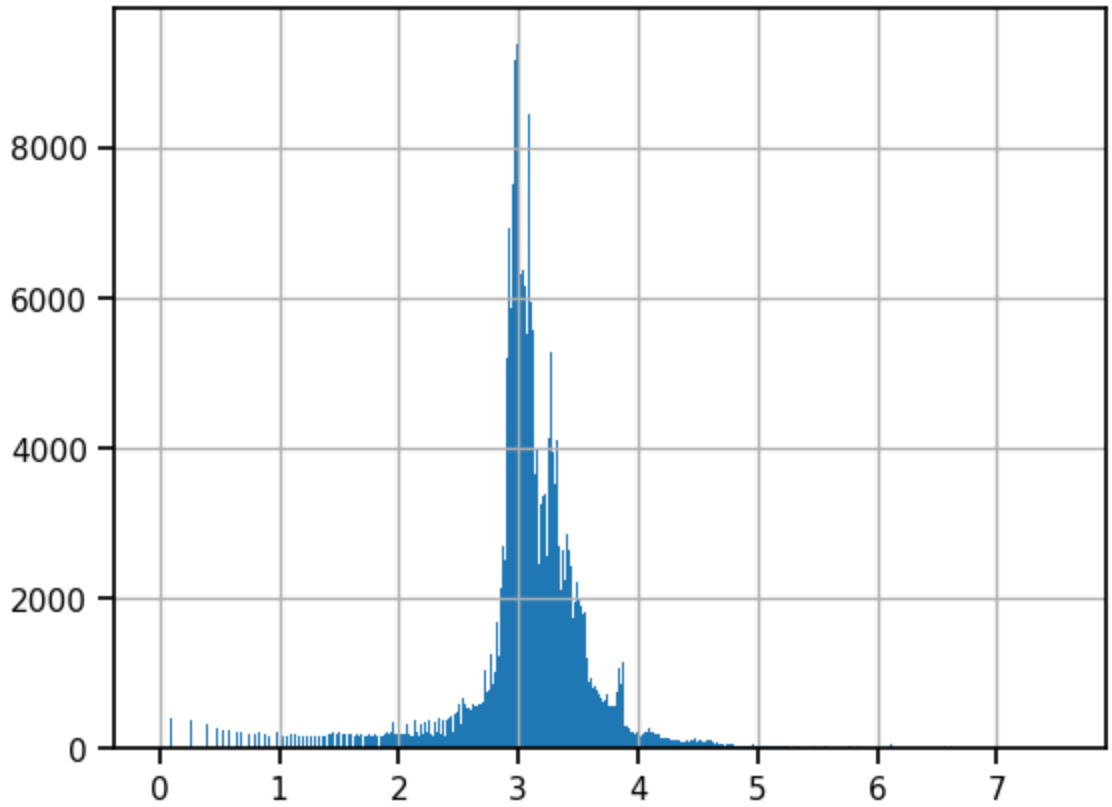
Out[7]:
```
<Axes: >
```



Though most of the transits appear to be short, there are a few longer distances that make the plot difficult to read. This is where a transformation is useful:

In [8]:
```
segments.seg_length.apply(np.log).hist(bins=500)
```

Out[8]:
```
<Axes: >
```



We can see that although there are date/time fields in the dataset, they are not in any specialized format, such as `datetime`.

Our first order of business will be to convert these data to `datetime`. The `strptime` method parses a string representation of a date and/or time field, according to the expected format of this information.

In [9]:
```
datetime.strptime(segments.st_time.iloc[0], '%m/%d/%y %H:%M')
```

Out[9]:
```
datetime.datetime(2009, 2, 10, 16, 3)
```

The `dateutil` package includes a parser that attempts to detect the format of the date strings, and convert them automatically.

In [10]:
```
from dateutil.parser import parse
```

In [11]:
```
parse(segments.st_time.iloc[0])
```

Out[11]:
```
datetime.datetime(2009, 2, 10, 16, 3)
```

We can convert all the dates in a particular column by using the `apply` method.

In [12]:
```
segments.st_time.apply(lambda d: datetime.strptime(d, '%m/%d/%y %H:%M')).head(10)
```

Out[12]:
```
0   2009-02-10 16:03:00
1   2009-04-06 14:31:00
2   2009-04-06 14:36:00
3   2009-04-10 17:58:00
4   2009-04-10 17:59:00
5   2010-03-20 16:06:00
6   2010-03-20 18:05:00
7   2011-05-04 11:28:00
8   2010-06-05 11:23:00
9   2010-06-08 11:03:00
Name: st_time, dtype: datetime64[ns]
```

As a convenience, Pandas has a `to_datetime` method that will parse and convert an entire `Series` of formatted strings into `datetime` objects.

In [13]:
```
pd.to_datetime(segments.st_time[:10], format='%m/%d/%y %H:%M')
```

Out[13]:
```
0   2009-02-10 16:03:00
1   2009-04-06 14:31:00
2   2009-04-06 14:36:00
3   2009-04-10 17:58:00
4   2009-04-10 17:59:00
5   2010-03-20 16:06:00
6   2010-03-20 18:05:00
7   2011-05-04 11:28:00
8   2010-06-05 11:23:00
9   2010-06-08 11:03:00
Name: st_time, dtype: datetime64[ns]
```

Pandas also has a custom NA value for missing datetime objects, `NaT`.

In [14]:
```
pd.to_datetime([None])
```

Out[14]:
```
DatetimeIndex(['NaT'], dtype='datetime64[ns]', freq=None)
```

Also, if `to_datetime()` has problems parsing any particular date/time format, you can pass the spec in using the `format=` argument.

The `read_*` functions now have an optional `parse_dates` argument that try to convert any columns passed to it into `datetime` format upon import:

In [15]:
```
segments = pd.read_csv("Data/transit_segments.csv", parse_dates=['st_time', 'end_time'],
```

In [16]:
```
segments.dtypes
```

Out[16]:
```
mmsi                   int64
name                  object
transit                int64
segment                int64
seg_length           float64
avg_sog              float64
min_sog              float64
max_sog              float64
pdgt10               float64
st_time       datetime64[ns]
end_time      datetime64[ns]
dtype: object
```

Columns of the `datetime` type have an **accessor** to easily extract properties of the data type. This will return a `Series`, with the same row index as the `DataFrame`. For example:

In [17]:
```
segments.st_time.dt.month.head()
```

Out[17]:
```
0    2
1    4
2    4
3    4
4    4
Name: st_time, dtype: int32
```

This can be used to easily filter rows by particular temporal attributes:

In [18]:
```
segments[segments.st_time.dt.month==2].head()
```

Out[18]:

	mmsi	name	transit	segment	seg_length	avg_sog	min_sog	max_sog	pdgt10	st_time	end_
0	1	Us Govt Ves	1	1	5.1	13.2	9.2	14.5	96.5	2009-02-10 16:03:00	2009 16:2
78	3011	Charleston	16	1	18.9	0.3	0.0	16.1	0.5	2010-02-07 07:26:00	2010 19:2
79	3011	Charleston	17	1	19.2	0.3	0.0	6.4	0.0	2010-02-11 16:56:00	2010 14:3
80	3011	Charleston	18	1	24.7	0.3	0.0	5.7	0.0	2010-02-19 11:53:00	2010 16:5
81	3011	Charleston	19	1	40.1	0.4	0.0	16.3	0.1	2010-02-23 15:15:00	2010 14:2

Merging and joining DataFrame objects

Now that we have the vessel transit information as we need it, we may want a little more information regarding the vessels themselves. In the `data/AIS` folder there is a second table that contains information about each of the ships that traveled the segments in the `segments` table.

In [19]:
```
vessels = pd.read_csv("Data/vessel_information.csv", index_col='mmsi')
vessels.head()
```

Out[19]:

mmsi	num_names	names	sov	flag	flag_type	num_loas	
1	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138
9	3	000000009/Raven/Shearwater	N	Unknown	Unknown	2	
21	1	Us Gov Vessel	Y	Unknown	Unknown	1	
74	2	Mcfaul/Sarah Bell	N	Unknown	Unknown	1	
103	3	Ron G/Us Navy Warship 103/Us Warship 103	Y	Unknown	Unknown	2	

The challenge is that several ships have travelled multiple segments, so there is not a one-to-one relationship between the rows of the two tables. **The table of vessel information has a *one-to-many* relationship with the segments**.

In Pandas, we can combine tables according to the value of one or more *keys* that are used to identify rows, much like an index. Using a trivial example:

In [20]:
```
df1 = pd.DataFrame(dict(id=range(4), age=np.random.randint(18, 31, size=4)))
df2 = pd.DataFrame(dict(id=list(range(3))+list(range(3)),
                        score=np.random.random(size=6)))

df1
```

Out[20]:

	id	age

0	0	25
1	1	24
2	2	26
3	3	26

In [21]:
```
df2
```

Out[21]:

	id	score
0	0	0.828877
1	1	0.429877
2	2	0.929862
3	0	0.876232
4	1	0.238272
5	2	0.164953

In [22]:
```
pd.merge(df1, df2)
```

Out[22]:

	id	age	score
0	0	25	0.828877
1	0	25	0.876232
2	1	24	0.429877
3	1	24	0.238272
4	2	26	0.929862
5	2	26	0.164953

Notice that without any information about which column to use as a key, Pandas did the right thing and used the `id` column in both tables. Unless specified otherwise, `merge` will used any common column names as keys for merging the tables.

Notice also that `id=3` from `df1` was omitted from the merged table. This is because, by default, `merge` performs an **inner join** on the tables, meaning that the merged table represents an intersection of the two tables.

In [23]:
```
pd.merge(df1, df2, how='outer')
```

Out[23]:

	id	age	score
0	0	25	0.828877
1	0	25	0.876232
2	1	24	0.429877
3	1	24	0.238272
4	2	26	0.929862
5	2	26	0.164953
6	3	26	NaN

The **outer join** above yields the union of the two tables, so all rows are represented, with missing values inserted as appropriate. One can also perform **right** and **left** joins to include all rows of the right or left table (*i.e.* first or second argument to `merge`), but not necessarily the other.

Looking at the two datasets that we wish to merge:

In [24]:
```
segments.head(1)
```

Out[24]:

	mmsi	name	transit	segment	seg_length	avg_sog	min_sog	max_sog	pdgt10	st_time	end_time
0	1	Us Govt Ves	1	1	5.1	13.2	9.2	14.5	96.5	2009-02-10 16:03:00	2009-02-10 16:27:00

In [25]:
```
vessels.head(1)
```

Out[25]:

mmsi	num_names	names	sov	flag	flag_type	num_loas	
1	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/15

we see that there is a `mmsi` value (a vessel identifier) in each table, but it is used as an index for the `vessels` table. In this case, we have to specify to join on the index for this table, and on the `mmsi` column for the other.

In [26]:
```
segments_merged = pd.merge(vessels, segments, left_index=True, right_on='mmsi')
```

In [27]:
```
segments_merged.head()
```

Out[27]:

	num_names	names	sov	flag	flag_type	num_loas	loa
0	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
1	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
2	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
3	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
4	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0

5 rows × 21 columns

In this case, the default inner join is suitable; we are not interested in observations from either table that do not have corresponding entries in the other.

Notice that `mmsi` field that was an index on the `vessels` table is no longer an index on the merged table.

Here, we used the `merge` function to perform the merge; we could also have used the `merge` *method* for either of the tables:

In [28]:
```
vessels.merge(segments, left_index=True, right_on='mmsi').head()
```

Out[28]:

	num_names	names	sov	flag	flag_type	num_loas	loa
0	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
1	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
2	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
3	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
4	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0

5 rows × 21 columns

Occasionally, there will be fields with the same in both tables that we do not wish to use to join the tables; they may contain different information, despite having the same name. In this case, Pandas will by default append suffixes `_x` and `_y` to the columns to uniquely identify them.

In [29]:
```
segments['type'] = 'foo'
pd.merge(vessels, segments, left_index=True, right_on='mmsi').head()
```

Out[29]:

	num_names	names	sov	flag	flag_type	num_loas	loa
0	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
1	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
2	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0

3	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0
4	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138.0/154.0/156.0

5 rows × 22 columns

This behavior can be overridden by specifying a `suffixes` argument, containing a list of the suffixes to be used for the columns of the left and right columns, respectively.

Exercise 6

Fix the following `merge` in order to return a non-empty `DataFrame`.

In [30]:
```
segments.merge(vessels).head(10) #empty
segments.merge(vessels, right_index=True, left_on='mmsi').head(10) #non-empty
```

Out[30]:

	mmsi	name	transit	segment	seg_length	avg_sog	min_sog	max_sog	pdgt10	st_time	...	nu
0	1	Us Govt Ves	1	1	5.1	13.2	9.2	14.5	96.5	2009-02-10 16:03:00	...	
1	1	Dredge Capt Frank	1	1	13.5	18.6	10.4	20.6	100.0	2009-04-06 14:31:00	...	
2	1	Us Gov Vessel	1	1	4.3	16.2	10.3	20.5	100.0	2009-04-06 14:36:00	...	
3	1	Us Gov Vessel	2	1	9.2	15.4	14.5	16.1	100.0	2009-04-10 17:58:00	...	
4	1	Dredge Capt Frank	2	1	9.2	15.4	14.6	16.2	100.0	2009-04-10 17:59:00	...	
5	1	Bil Holman Dredge	1	1	17.4	34.3	33.9	35.0	100.0	2010-03-20 16:06:00	...	
6	1	Bil Holman Dredge	1	2	76.0	34.4	33.8	34.9	100.0	2010-03-20 18:05:00	...	
7	1	S.d. Gumel	1	1	13.7	6.9	3.0	14.0	38.2	2011-05-04 11:28:00	...	
8	9	Shearwater	4	1	11.6	8.8	7.9	10.3	0.0	2010-06-05 11:23:00	...	
9	9	Shearwater	8	1	11.6	8.3	7.4	9.2	0.0	2010-06-08 11:03:00	...	

10 rows × 22 columns

Concatenation

A common data manipulation is appending rows or columns to a dataset that already conform to the dimensions of the exsiting rows or colums, respectively:

In [31]:
```
np.concatenate([np.random.random(5), np.random.random(5)])
```

Out[31]:
```
array([0.39198087, 0.41995382, 0.43967486, 0.1565683 , 0.36624927,
       0.45438236, 0.84902765, 0.03488581, 0.67308845, 0.43687262])
```

This operation is also called *binding* or *stacking*.

With Pandas' indexed data structures, there are additional considerations as the overlap in index values between two data structures affects how they are concatenate.

Lets import two microbiome datasets, each consisting of counts of microorganiams from a particular patient. We will use the first column of each dataset as the index.

In [32]:
```
mb1 = pd.read_excel('Data/microbiome_MID1.xls', 'Sheet 1', index_col=0, header=None)
mb2 = pd.read_excel('Data/microbiome_MID2.xls', 'Sheet 1', index_col=0, header=None)
mb1.columns = mb2.columns = ['Count']
mb1.index.name = mb2.index.name = 'Taxon'
mb1.shape, mb2.shape
```

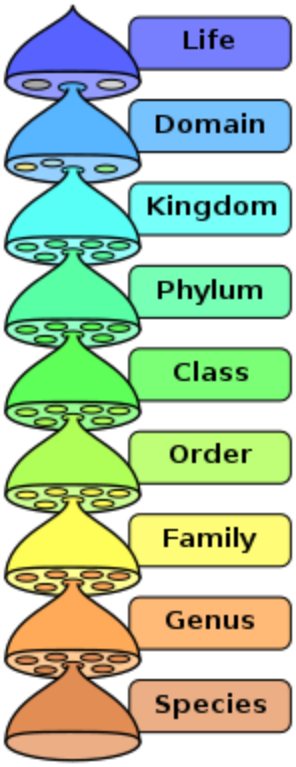
Out[32]:
```
((272, 1), (288, 1))
```

In [33]:
```
mb1.head()
```

Out[33]:

Taxon	Count
Archaea "Crenarchaeota" Thermoprotei Desulfurococcales Desulfurococcaceae Ignisphaera	7
Archaea "Crenarchaeota" Thermoprotei Desulfurococcales Pyrodictiaceae Pyrolobus	2
Archaea "Crenarchaeota" Thermoprotei Sulfolobales Sulfolobaceae Stygiolobus	3
Archaea "Crenarchaeota" Thermoprotei Thermoproteales Thermofilaceae Thermofilum	3
Archaea "Euryarchaeota" "Methanomicrobia" Methanocellales Methanocellaceae Methanocella	7

The index of these data is the unique biological classification of each organism, beginning with *domain*, *phylum*, *class*, and for some organisms, going all the way down to the genus level.



In [34]:
```
mb1.index.is_unique
```

Out[34]:
```
True
```

If we concatenate along `axis=0` (the default), we will obtain another data frame with the the rows concatenated:

In [35]:
```
pd.concat([mb1, mb2], axis=0).head()
```

Out[35]:

Taxon	Count
Archaea "Crenarchaeota" Thermoprotei Desulfurococcales Desulfurococcaceae Ignisphaera	7
Archaea "Crenarchaeota" Thermoprotei Desulfurococcales Pyrodictiaceae Pyrolobus	2
Archaea "Crenarchaeota" Thermoprotei Sulfolobales Sulfolobaceae Stygiolobus	3
Archaea "Crenarchaeota" Thermoprotei Thermoproteales Thermofilaceae Thermofilum	3
Archaea "Euryarchaeota" "Methanomicrobia" Methanocellales Methanocellaceae Methanocella	7

However, the index is no longer unique, due to overlap between the two `DataFrames`.

In [36]:
```
pd.concat([mb1, mb2], axis=0).index.is_unique
```

Out[36]:
```
False
```

Concatenating along `axis=1` will concatenate column-wise, but respecting the indices of the two `DataFrames`.

In [37]:
```
pd.concat([mb1, mb2], axis=1).shape
```

Out[37]:
```
(438, 2)
```

In [38]:
```
pd.concat([mb1, mb2], axis=1).head()
```

Out[38]:

	Count	Count
Taxon		
Archaea "Crenarchaeota" Thermoprotei Desulfurococcales Desulfurococcaceae Ignisphaera	7.0	23.0
Archaea "Crenarchaeota" Thermoprotei Desulfurococcales Pyrodictiaceae Pyrolobus	2.0	2.0
Archaea "Crenarchaeota" Thermoprotei Sulfolobales Sulfolobaceae Stygiolobus	3.0	10.0
Archaea "Crenarchaeota" Thermoprotei Thermoproteales Thermofilaceae Thermofilum	3.0	9.0
Archaea "Euryarchaeota" "Methanomicrobia" Methanocellales Methanocellaceae Methanocella	7.0	9.0

Reshaping DataFrame objects

In the context of a single DataFrame, we are often interested in re-arranging the layout of our data.

This dataset is from Table 6.9 of Statistical Methods for the Analysis of Repeated Measurements by Charles S. Davis, pp. 161-163 (Springer, 2002). These data are from a multicenter, randomized controlled trial of botulinum toxin type B (BotB) in patients with cervical dystonia from nine U.S. sites.

- Randomized to placebo (N=36), 5000 units of BotB (N=36), 10,000 units of BotB (N=37)
- Response variable: total score on Toronto Western Spasmodic Torticollis Rating Scale (TWSTRS), measuring severity, pain, and disability of cervical dystonia (high scores mean more impairment)
- TWSTRS measured at baseline (week 0) and weeks 2, 4, 8, 12, 16 after treatment began

In [39]:
```
cdystonia = pd.read_csv("Data/cdystonia.csv", index_col=None)
cdystonia.head()
```

Out[39]:

	patient	obs	week	site	id	treat	age	sex	twstrs
0	1	1	0	1	1	5000U	65	F	32
1	1	2	2	1	1	5000U	65	F	30
2	1	3	4	1	1	5000U	65	F	24
3	1	4	8	1	1	5000U	65	F	37
4	1	5	12	1	1	5000U	65	F	39

This dataset includes repeated measurements of the same individuals (longitudinal data). Its possible to present such information in (at least) two ways: showing each repeated measurement in their own row, or in multiple columns representing multiple measurements.

The `stack` method rotates the data frame so that columns are represented in rows:

In [40]:
```
stacked = cdystonia.stack()
stacked.head(15)
```

Out[40]:
```
0  patient      1
   obs          1
   week         0
   site         1
   id           1
   treat    5000U
   age         65
```

```
   sex            F
   twstrs        32
1  patient        1
   obs            2
   week           2
   site           1
   id             1
   treat       5000U
dtype: object
```

To complement this, `unstack` pivots from rows back to columns.

In [41]:
```
stacked.unstack().head()
```

Out[41]:

	patient	obs	week	site	id	treat	age	sex	twstrs
0	1	1	0	1	1	5000U	65	F	32
1	1	2	2	1	1	5000U	65	F	30
2	1	3	4	1	1	5000U	65	F	24
3	1	4	8	1	1	5000U	65	F	37
4	1	5	12	1	1	5000U	65	F	39

For this dataset, it makes sense to create a hierarchical index based on the patient and observation:

Pivoting

The `pivot` method allows a DataFrame to be transformed easily between long and wide formats in the same way as a pivot table is created in a spreadsheet. It takes three arguments: `index`, `columns` and `values`, corresponding to the DataFrame index (the row headers), columns and cell values, respectively.

For example, we may want the `twstrs` variable (the response variable) in wide format according to patient, as we saw with the unstacking method above:

In [42]:
```
twstrs_wide = cdystonia.pivot(index='patient', columns='obs', values='twstrs').head()
twstrs_wide
```

Out[42]:

obs	1	2	3	4	5	6
patient						
1	32.0	30.0	24.0	37.0	39.0	36.0
2	60.0	26.0	27.0	41.0	65.0	67.0
3	44.0	20.0	23.0	26.0	35.0	35.0
4	53.0	61.0	64.0	62.0	NaN	NaN
5	53.0	35.0	48.0	49.0	41.0	51.0

In [43]:
```
cdystonia_wide = (cdystonia[['patient','site','id','treat','age','sex']]
                  .drop_duplicates()
                  .merge(twstrs_wide, right_index=True, left_on='patient', how='inner')
                  .head())
cdystonia_wide
```

Out[43]:

	patient	site	id	treat	age	sex	1	2	3	4	5	6

0	1	1	1	5000U	65	F	32.0	30.0	24.0	37.0	39.0	36.0
6	2	1	2	10000U	70	F	60.0	26.0	27.0	41.0	65.0	67.0
12	3	1	3	5000U	64	F	44.0	20.0	23.0	26.0	35.0	35.0
18	4	1	4	Placebo	59	F	53.0	61.0	64.0	62.0	NaN	NaN
22	5	1	5	10000U	76	F	53.0	35.0	48.0	49.0	41.0	51.0

To convert our "wide" format back to long, we can use the `melt` function, appropriately parameterized. This function is useful for `DataFrame`s where one or more columns are identifier variables (`id_vars`), with the remaining columns being measured variables (`value_vars`). The measured variables are "unpivoted" to the row axis, leaving just two non-identifier columns, a *variable* and its corresponding *value*, which can both be renamed using optional arguments.

In [44]:
```
pd.melt(cdystonia_wide, id_vars=['patient','site','id','treat','age','sex'], var_name='o
```

Out[44]:

	patient	site	id	treat	age	sex	obs	twsters
0	1	1	1	5000U	65	F	1	32.0
1	2	1	2	10000U	70	F	1	60.0
2	3	1	3	5000U	64	F	1	44.0
3	4	1	4	Placebo	59	F	1	53.0
4	5	1	5	10000U	76	F	1	53.0

This illustrates the two formats for longitudinal data: **long** and **wide** formats. Its typically better to store data in long format because additional data can be included as additional rows in the database, while wide format requires that the entire database schema be altered by adding columns to every row as data are collected.

The preferable format for analysis depends entirely on what is planned for the data, so it is imporant to be able to move easily between them.

A related method, `pivot_table`, creates a spreadsheet-like table with a hierarchical index, and allows the values of the table to be populated using an arbitrary aggregation function.

In [45]:
```
cdystonia.pivot_table(index=['site', 'treat'], columns='week', values='twstrs', aggfunc=
```

```
/var/folders/1n/kf80j0zn4bn5q4md0z8h3mbm0000gp/T/ipykernel_15566/2681402665.py:1: Future
Warning: The provided callable <built-in function max> is currently using DataFrameGroup
By.max. In a future version of pandas, the provided callable will be used directly. To k
eep current behavior pass the string "max" instead.
  cdystonia.pivot_table(index=['site', 'treat'], columns='week', values='twstrs', aggfun
c=max).head(10)
```

Out[45]:

	week	0	2	4	8	12	16
site	**treat**						
1	**10000U**	60	41	48	49	65	67
	5000U	44	32	34	43	42	46
	Placebo	53	61	64	62	32	38
2	**10000U**	65	60	60	64	67	66
	5000U	67	64	65	64	62	64

	Placebo	53	56	52	57	61	54
3	**10000U**	50	43	51	46	49	56
	5000U	52	44	47	50	50	49
	Placebo	43	38	40	48	49	44
4	**10000U**	54	52	52	54	51	57

For a simple cross-tabulation of group frequencies, the `crosstab` function (not a method) aggregates counts of data according to factors in rows and columns. The factors may be hierarchical if desired.

In [46]:
```
pd.crosstab(cdystonia.sex, cdystonia.site)
```

Out[46]:

site	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
sex									
F	52	53	42	30	22	54	66	48	28
M	18	29	30	18	11	33	6	58	33

Data transformation

There are a slew of additional operations for DataFrames that we would collectively refer to as "transformations" which include tasks such as removing duplicate values, replacing values, and grouping values.

Dealing with duplicates

We can easily identify and remove duplicate values from `DataFrame` objects. For example, say we want to removed ships from our `vessels` dataset that have the same name:

In [47]:
```
vessels.duplicated(subset='names').head(30)
```

Out[47]:
```
mmsi
1          False
9          False
21         False
74         False
103        False
310        False
3011       False
4731       False
15151      False
46809      False
80404      False
82003      False
298716     False
366235     False
439541     False
453556     False
505843     False
527918     False
565026     False
572329     False
587370     False
641114     False
```

```
642262     False
693559     False
883085      True
1193046    False
1193946    False
1233916    False
1239468    False
3041300    False
dtype: bool
```

In [48]:
```
vessels.drop_duplicates(['names']).head()
```

Out[48]:

mmsi	num_names	names	sov	flag	flag_type	num_loas	
1	8	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	Y	Unknown	Unknown	7	42.0/48.0/57.0/90.0/138
9	3	000000009/Raven/Shearwater	N	Unknown	Unknown	2	
21	1	Us Gov Vessel	Y	Unknown	Unknown	1	
74	2	Mcfaul/Sarah Bell	N	Unknown	Unknown	1	
103	3	Ron G/Us Navy Warship 103/Us Warship 103	Y	Unknown	Unknown	2	

Value replacement

Frequently, we get data columns that are encoded as strings that we wish to represent numerically for the purposes of including it in a quantitative analysis. For example, consider the treatment variable in the cervical dystonia dataset:

In [49]:
```
cdystonia.treat.value_counts()
```

Out[49]:
```
treat
10000U     213
5000U      211
Placebo    207
Name: count, dtype: int64
```

A logical way to specify these numerically is to change them to integer values, perhaps using "Placebo" as a baseline value. If we create a dict with the original values as keys and the replacements as values, we can pass it to the `map` method to implement the changes.

In [50]:
```
cdystonia['treatment'] = cdystonia.treat.map({'Placebo': 0, '5000U': 1, '10000U': 2})
cdystonia.treatment.head(10)
```

Out[50]:
```
0    1
1    1
2    1
3    1
4    1
5    1
6    2
7    2
8    2
9    2
Name: treatment, dtype: int64
```

We can also perform the same replacement that we used `map` for with `replace`:

In [51]:
```
cdystonia.treat.replace({0:'Placebo', 1:'5000U', 2:'10000U'}).head(10)
```

Out[51]:
```
0     5000U
1     5000U
2     5000U
3     5000U
4     5000U
5     5000U
6    10000U
7    10000U
8    10000U
9    10000U
Name: treat, dtype: object
```

Inidcator variables

For some statistical analyses (*e.g.* regression models or analyses of variance), categorical or group variables need to be converted into columns of indicators--zeros and ones--to create a so-called **design matrix**. The Pandas function `get_dummies` (indicator variables are also known as *dummy variables*) makes this transformation straightforward.

In [52]:
```
pd.get_dummies(vessels.head(10).type)
```

Out[52]:

mmsi	Dredging/MilOps/Reserved/Towing	Other	Pleasure/Tug	Tanker/Unknown	Towing	Tug	Unknown
1	True	False	False	False	False	False	False
9	False	False	True	False	False	False	False
21	False	False	False	False	False	False	True
74	False	False	False	False	False	False	True
103	False	False	False	True	False	False	False
310	False	False	False	False	False	False	True
3011	False	True	False	False	False	False	False
4731	False	False	False	False	False	False	True
15151	False	False	False	False	False	True	False
46809	False	False	False	False	True	False	False

Categorical Data

Pandas provides a convenient `dtype` for reprsenting categorical (factor) data, called `category`.

For example, the `treat` column in the cervical dystonia dataset represents three treatment levels in a clinical trial, and is imported by default as an `object` type, since it is a mixture of string characters.

In [53]:
```
cdystonia.treat.head()
```

Out[53]:
```
0    5000U
1    5000U
2    5000U
3    5000U
4    5000U
Name: treat, dtype: object
```

We can convert this to a `category` type either by the `Categorical` constructor, or casting the column using `astype`:

In [54]:
```
pd.Categorical(cdystonia.treat)
```

Out[54]:
```
['5000U', '5000U', '5000U', '5000U', '5000U', ..., '5000U', '5000U', '5000U', '5000U',
'5000U']
Length: 631
Categories (3, object): ['10000U', '5000U', 'Placebo']
```

In [55]:
```
cdystonia['treat'] = cdystonia.treat.astype('category')
```

In [56]:
```
cdystonia.treat.describe()
```

Out[56]:
```
count        631
unique         3
top       10000U
freq         213
Name: treat, dtype: object
```

The important difference between the `category` type and the `object` type is that `category` is represented by an **underlying array of integers**, which is then mapped to character labels.

In [57]:
```
cdystonia.treat.cat.codes.head(20)
```

Out[57]:
```
0     1
1     1
2     1
3     1
4     1
5     1
6     0
7     0
8     0
9     0
10    0
11    0
12    1
13    1
14    1
15    1
16    1
17    1
18    2
19    2
dtype: int8
```

Notice that these are 8-bit integers, which are essentially single bytes of data, making memory usage lower.

There is also a performance benefit. Consider an operation such as calculating the total segment lengths for each ship in the `segments` table (this is also a preview of pandas' `groupby` operation!):

In [58]:
```
%time segments.groupby(segments.name).seg_length.sum().sort_values(ascending=False, inpl
```

Out[58]:
```
CPU times: user 1.48 ms, sys: 483 µs, total: 1.96 ms
Wall time: 1.53 ms
name
Nauticast           19190.5
Asphalt Seminole     6528.1
Majestic             2997.9
Neva Belle           2737.9
```

```
Zaandam              2296.7
Name: seg_length, dtype: float64
```

In [59]:
```
segments['name'] = segments.name.astype('category')
```

In [60]:
```
%time segments.groupby(segments.name).seg_length.sum().sort_values(ascending=False, inpl
```

```
CPU times: user 2.39 ms, sys: 696 µs, total: 3.08 ms
Wall time: 2.7 ms
<timed eval>:1: FutureWarning: The default of observed=False is deprecated and will be c
hanged to True in a future version of pandas. Pass observed=False to retain current beha
vior or observed=True to adopt the future default and silence this warning.
```

Out[60]:
```
name
Nauticast           19190.5
Asphalt Seminole     6528.1
Majestic             2997.9
Neva Belle           2737.9
Zaandam              2296.7
Name: seg_length, dtype: float64
```

Hence, we get a considerable speedup simply by using the appropriate `dtype` for our data.

Discretization

Pandas' `cut` function can be used to group continuous or countable data in to bins. Discretization is generally a very **bad idea** for statistical analysis, so use this function responsibly!

Lets say we want to bin the ages of the cervical dystonia patients into a smaller number of groups:

In [61]:
```
cdystonia.age.describe()
```

Out[61]:
```
count    631.000000
mean      55.616482
std       12.123910
min       26.000000
25%       46.000000
50%       56.000000
75%       65.000000
max       83.000000
Name: age, dtype: float64
```

Let's transform these data into decades, beginnnig with individuals in their 20's and ending with those in their 80's:

In [62]:
```
pd.cut(cdystonia.age, [20,30,40,50,60,70,80,90])[:20]
```

Out[62]:
```
0     (60, 70]
1     (60, 70]
2     (60, 70]
3     (60, 70]
4     (60, 70]
5     (60, 70]
6     (60, 70]
7     (60, 70]
8     (60, 70]
9     (60, 70]
10    (60, 70]
11    (60, 70]
12    (60, 70]
13    (60, 70]
14    (60, 70]
15    (60, 70]
```

```
16    (60, 70]
17    (60, 70]
18    (50, 60]
19    (50, 60]
Name: age, dtype: category
Categories (7, interval[int64, right]): [(20, 30] < (30, 40] < (40, 50] < (50, 60] < (6
0, 70] < (70, 80] < (80, 90]]
```

The parentheses indicate an open interval, meaning that the interval includes values up to but *not including* the endpoint, whereas the square bracket is a closed interval, where the endpoint is included in the interval. We can switch the closure to the left side by setting the `right` flag to `False`:

In [63]:
```
pd.cut(cdystonia.age, [20,30,40,50,60,70,80,90], right=False)[:20]
```

Out[63]:
```
0     [60, 70)
1     [60, 70)
2     [60, 70)
3     [60, 70)
4     [60, 70)
5     [60, 70)
6     [70, 80)
7     [70, 80)
8     [70, 80)
9     [70, 80)
10    [70, 80)
11    [70, 80)
12    [60, 70)
13    [60, 70)
14    [60, 70)
15    [60, 70)
16    [60, 70)
17    [60, 70)
18    [50, 60)
19    [50, 60)
Name: age, dtype: category
Categories (7, interval[int64, left]): [[20, 30) < [30, 40) < [40, 50) < [50, 60) < [60,
70) < [70, 80) < [80, 90)]
```

Since the data are now **ordinal**, rather than numeric, we can give them labels:

In [64]:
```
pd.cut(cdystonia.age, [20,40,60,80,90], labels=['young','middle-aged','old','really old'
```

Out[64]:
```
0             old
1             old
2             old
3             old
4             old
5             old
6             old
7             old
8             old
9             old
10            old
11            old
12            old
13            old
14            old
15            old
16            old
17            old
18    middle-aged
19    middle-aged
Name: age, dtype: category
Categories (4, object): ['young' < 'middle-aged' < 'old' < 'really old']
```

A related function `qcut` uses empirical quantiles to divide the data. If, for example, we want the quartiles -- (0-25%], (25-50%], (50-70%], (75-100%] -- we can just specify 4 intervals, which will be equally-spaced by default:

In [65]:
```
pd.qcut(cdystonia.age, 4)[:20]
```

Out[65]:
```
0     (56.0, 65.0]
1     (56.0, 65.0]
2     (56.0, 65.0]
3     (56.0, 65.0]
4     (56.0, 65.0]
5     (56.0, 65.0]
6     (65.0, 83.0]
7     (65.0, 83.0]
8     (65.0, 83.0]
9     (65.0, 83.0]
10    (65.0, 83.0]
11    (65.0, 83.0]
12    (56.0, 65.0]
13    (56.0, 65.0]
14    (56.0, 65.0]
15    (56.0, 65.0]
16    (56.0, 65.0]
17    (56.0, 65.0]
18    (56.0, 65.0]
19    (56.0, 65.0]
Name: age, dtype: category
Categories (4, interval[float64, right]): [(25.999, 46.0] < (46.0, 56.0] < (56.0, 65.0]
< (65.0, 83.0]]
```

Alternatively, one can specify custom quantiles to act as cut points:

In [66]:
```
quantiles = pd.qcut(segments.seg_length, [0, 0.01, 0.05, 0.95, 0.99, 1])
quantiles[:20]
```

Out[66]:
```
0     (1.7, 455.8]
1     (1.7, 455.8]
2     (1.7, 455.8]
3     (1.7, 455.8]
4     (1.7, 455.8]
5     (1.7, 455.8]
6     (1.7, 455.8]
7     (1.7, 455.8]
8     (1.7, 455.8]
9     (1.7, 455.8]
10    (1.7, 455.8]
11    (1.7, 455.8]
12    (1.7, 455.8]
13    (1.7, 455.8]
14    (1.7, 455.8]
15    (1.7, 455.8]
16    (1.7, 455.8]
17    (1.7, 455.8]
18    (1.7, 455.8]
19    (1.7, 455.8]
Name: seg_length, dtype: category
Categories (5, interval[float64, right]): [(0.999, 1.1] < (1.1, 1.7] < (1.7, 455.8] < (4
55.8, 456.801] < (456.801, 701.6]]
```

Permutation and sampling

For some data analysis tasks, such as simulation, we need to be able to randomly reorder our data, or draw random values from it. Calling NumPy's `permutation` function with the length of the sequence

you want to permute generates an array with a permuted sequence of integers, which can be used to re-order the sequence.

In [67]:
```
new_order = np.random.permutation(len(segments))
new_order[:30]
```

Out[67]:
```
array([767, 369, 308, 908, 327,  32, 667, 493, 147, 786, 683, 605, 803,
       505, 180, 146, 714, 341, 916, 634, 222, 421, 248, 305, 904, 925,
       594, 148, 297, 512])
```

Using this sequence as an argument to the `take` method results in a reordered DataFrame:

In [68]:
```
segments.take(new_order).head()
```

Out[68]:

	mmsi	name	transit	segment	seg_length	avg_sog	min_sog	max_sog	pdgt10	st_time	e
767	1193046	Nauticast	58	57	455.8	0.1	0.0	0.2	0.0	2009-04-16 02:22:00	2
369	641114	Samantha Miller	15	1	12.9	2.3	0.0	9.0	0.0	2009-01-02 13:27:00	2
308	587370	Dorothy Moran	64	1	1.2	6.9	2.6	10.7	14.2	2010-08-27 11:54:00	2
908	1193046	Lady Simpson	7	2	19.7	11.5	2.1	17.2	50.8	2009-08-12 19:16:00	2
327	587370	Dorothy Moran	94	1	26.8	11.7	10.9	13.4	100.0	2010-10-13 23:00:00	2

Compare this ordering with the original:

In [69]:
```
segments.head()
```

Out[69]:

	mmsi	name	transit	segment	seg_length	avg_sog	min_sog	max_sog	pdgt10	st_time	end_time
0	1	Us Govt Ves	1	1	5.1	13.2	9.2	14.5	96.5	2009-02-10 16:03:00	2009-02-10 16:27:00
1	1	Dredge Capt Frank	1	1	13.5	18.6	10.4	20.6	100.0	2009-04-06 14:31:00	2009-04-06 15:20:00
2	1	Us Gov Vessel	1	1	4.3	16.2	10.3	20.5	100.0	2009-04-06 14:36:00	2009-04-06 14:55:00
3	1	Us Gov Vessel	2	1	9.2	15.4	14.5	16.1	100.0	2009-04-10 17:58:00	2009-04-10 18:34:00
4	1	Dredge Capt Frank	2	1	9.2	15.4	14.6	16.2	100.0	2009-04-10 17:59:00	2009-04-10 18:35:00

For random sampling, `DataFrame` and `Series` objects have a `sample` method that can be used to draw samples, with or without replacement:

In [70]:
```
vessels.sample(n=10)
```

Out[70]:

mmsi	num_names	names	sov	flag	flag_type	num_loas	loa	ma
311062400	1	Ditlev Reefer	N	Bahamas (Commonwealth of the)	Foreign	1	164.0	
259776000	1	Goya	N	Norway	Foreign	1	225.0	
367316790	2	367316790/Janice/julie	N	United States of America	Domestic	1	30.0	
636091350	1	Hs Elektra	N	Liberia (Republic of)	Foreign	1	243.0	
353704000	1	Suez Canal Bridge	N	Panama (Republic of)	Foreign	2	285.0/289.0	
565621000	1	Sichem Contester	N	Singapore (Republic of)	Foreign	1	147.0	
305266000	1	Bbc New York	N	Antigua and Barbuda	Foreign	1	132.0	
419797000	1	Desh Mahima	N	India (Republic of)	Foreign	1	250.0	
240435000	1	Minerva Ellie	N	Greece	Foreign	1	244.0	
367168860	1	Starr	N	United States of America	Domestic	1	23.0	

In [71]:
```
vessels.sample(n=10, replace=True)
```

Out[71]:

mmsi	num_names	names	sov	flag	flag_type	num_loas	loa	max_loa	n
477999100	1	Excellent Ace	N	Hong Kong (Special Administrative Region of Ch...	Foreign	2	199.0/200.0	200.0	
566389000	1	Oocl Kaohsiung	N	Singapore (Republic of)	Foreign	1	261.0	261.0	
305048000	1	Bbc Greenland	N	Antigua and Barbuda	Foreign	1	120.0	120.0	
370488000	1	Sunlight Ocean	N	Panama (Republic of)	Foreign	1	225.0	225.0	
710465000	1	F Constituicao	Y	Brazil (Federative Republic of)	Foreign	1	129.0	129.0	
338931000	1	Red Cloud	N	United States of America	Domestic	1	289.0	289.0	
657411000	1	Ugonwaafor1	N	Nigeria (Federal Republic of)	Foreign	1	33.0	33.0	
306589000	1	Blue Marlin	N	Netherlands Antilles	Foreign	1	225.0	225.0	
367059950	1	El Puma Grande	N	United States of America	Domestic	3	30.0/31.0/33.0	33.0	

636013656		1	Golden Seas	N	Liberia (Republic of)	Foreign		1	225.0	225.0

Data aggregation and GroupBy operations

One of the most powerful features of Pandas is its **GroupBy** functionality. On some occasions we may want to perform operations on *groups* of observations within a dataset. For example:

- **aggregation**, such as computing the sum of mean of each group, which involves applying a function to each group and returning the aggregated results
- **slicing** the DataFrame into groups and then doing something with the resulting slices (*e.g.* plotting)
- group-wise **transformation**, such as standardization/normalization

In [72]:
```
cdystonia_grouped = cdystonia.groupby(cdystonia.patient)
```

However, the grouping is only an intermediate step; for example, we may want to **iterate** over each of the patient groups:

In [73]:
```
for patient, group in cdystonia_grouped:
    print('patient', patient)
    print('group', group)
```

```
patient 1
group    patient  obs  week  site  id  treat  age sex  twstrs  treatment
0        1    1     0     1   1  5000U   65   F      32          1
1        1    2     2     1   1  5000U   65   F      30          1
2        1    3     4     1   1  5000U   65   F      24          1
3        1    4     8     1   1  5000U   65   F      37          1
4        1    5    12     1   1  5000U   65   F      39          1
5        1    6    16     1   1  5000U   65   F      36          1
patient 2
group     patient  obs  week  site  id   treat  age sex  twstrs  treatment
6         2    1     0     1   2  10000U   70   F      60          2
7         2    2     2     1   2  10000U   70   F      26          2
8         2    3     4     1   2  10000U   70   F      27          2
9         2    4     8     1   2  10000U   70   F      41          2
10        2    5    12     1   2  10000U   70   F      65          2
11        2    6    16     1   2  10000U   70   F      67          2
patient 3
group     patient  obs  week  site  id  treat  age sex  twstrs  treatment
12        3    1     0     1   3  5000U   64   F      44          1
13        3    2     2     1   3  5000U   64   F      20          1
14        3    3     4     1   3  5000U   64   F      23          1
15        3    4     8     1   3  5000U   64   F      26          1
16        3    5    12     1   3  5000U   64   F      35          1
17        3    6    16     1   3  5000U   64   F      35          1
patient 4
group     patient  obs  week  site  id    treat  age sex  twstrs  treatment
18        4    1     0     1   4  Placebo   59   F      53          0
19        4    2     2     1   4  Placebo   59   F      61          0
20        4    3     4     1   4  Placebo   59   F      64          0
21        4    4     8     1   4  Placebo   59   F      62          0
patient 5
group     patient  obs  week  site  id   treat  age sex  twstrs  treatment
22        5    1     0     1   5  10000U   76   F      53          2
23        5    2     2     1   5  10000U   76   F      35          2
24        5    3     4     1   5  10000U   76   F      48          2
25        5    4     8     1   5  10000U   76   F      49          2
26        5    5    12     1   5  10000U   76   F      41          2
27        5    6    16     1   5  10000U   76   F      51          2
```

patient 6

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
28	6	1	0	1	6	10000U	59	F	49	2
29	6	2	2	1	6	10000U	59	F	34	2
30	6	3	4	1	6	10000U	59	F	43	2
31	6	4	8	1	6	10000U	59	F	48	2
32	6	5	12	1	6	10000U	59	F	48	2
33	6	6	16	1	6	10000U	59	F	51	2

patient 7

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
34	7	1	0	1	7	5000U	72	M	42	1
35	7	2	2	1	7	5000U	72	M	32	1
36	7	3	4	1	7	5000U	72	M	32	1
37	7	4	8	1	7	5000U	72	M	43	1
38	7	5	12	1	7	5000U	72	M	42	1
39	7	6	16	1	7	5000U	72	M	46	1

patient 8

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
40	8	1	0	1	8	Placebo	40	M	34	0
41	8	2	2	1	8	Placebo	40	M	33	0
42	8	3	4	1	8	Placebo	40	M	21	0
43	8	4	8	1	8	Placebo	40	M	27	0
44	8	5	12	1	8	Placebo	40	M	32	0
45	8	6	16	1	8	Placebo	40	M	38	0

patient 9

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
46	9	1	0	1	9	5000U	52	F	41	1
47	9	2	2	1	9	5000U	52	F	32	1
48	9	3	4	1	9	5000U	52	F	34	1
49	9	4	8	1	9	5000U	52	F	35	1
50	9	5	12	1	9	5000U	52	F	37	1
51	9	6	16	1	9	5000U	52	F	36	1

patient 10

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
52	10	1	0	1	10	Placebo	47	M	27	0
53	10	2	2	1	10	Placebo	47	M	10	0
54	10	3	4	1	10	Placebo	47	M	31	0
55	10	4	8	1	10	Placebo	47	M	32	0
56	10	5	12	1	10	Placebo	47	M	6	0
57	10	6	16	1	10	Placebo	47	M	14	0

patient 11

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
58	11	1	0	1	11	10000U	57	F	48	2
59	11	2	2	1	11	10000U	57	F	41	2
60	11	3	4	1	11	10000U	57	F	32	2
61	11	4	8	1	11	10000U	57	F	35	2
62	11	5	12	1	11	10000U	57	F	57	2
63	11	6	16	1	11	10000U	57	F	51	2

patient 12

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
64	12	1	0	1	12	Placebo	47	F	34	0
65	12	2	2	1	12	Placebo	47	F	19	0
66	12	3	4	1	12	Placebo	47	F	21	0
67	12	4	8	1	12	Placebo	47	F	24	0
68	12	5	12	1	12	Placebo	47	F	28	0
69	12	6	16	1	12	Placebo	47	F	28	0

patient 13

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
70	13	1	0	2	1	Placebo	70	F	49	0
71	13	2	2	2	1	Placebo	70	F	47	0
72	13	3	4	2	1	Placebo	70	F	44	0
73	13	4	8	2	1	Placebo	70	F	48	0
74	13	5	12	2	1	Placebo	70	F	44	0
75	13	6	16	2	1	Placebo	70	F	44	0

patient 14

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
76	14	1	0	2	2	5000U	49	F	46	1
77	14	2	2	2	2	5000U	49	F	35	1
78	14	3	4	2	2	5000U	49	F	45	1
79	14	4	8	2	2	5000U	49	F	49	1
80	14	5	12	2	2	5000U	49	F	53	1
81	14	6	16	2	2	5000U	49	F	56	1

patient 15

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
82	15	1	0	2	3	10000U	59	F	56	2
83	15	2	2	2	3	10000U	59	F	44	2
84	15	3	4	2	3	10000U	59	F	48	2
85	15	4	8	2	3	10000U	59	F	54	2
86	15	5	12	2	3	10000U	59	F	49	2
87	15	6	16	2	3	10000U	59	F	60	2

patient 16

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
88	16	1	0	2	4	5000U	64	M	59	1
89	16	2	2	2	4	5000U	64	M	48	1
90	16	3	4	2	4	5000U	64	M	56	1
91	16	4	8	2	4	5000U	64	M	55	1
92	16	5	12	2	4	5000U	64	M	57	1
93	16	6	16	2	4	5000U	64	M	58	1

patient 17

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
94	17	1	0	2	5	10000U	45	F	62	2
95	17	2	2	2	5	10000U	45	F	60	2
96	17	3	4	2	5	10000U	45	F	60	2
97	17	4	8	2	5	10000U	45	F	64	2
98	17	5	12	2	5	10000U	45	F	67	2
99	17	6	16	2	5	10000U	45	F	66	2

patient 18

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
100	18	1	0	2	6	Placebo	66	F	50	0
101	18	2	2	2	6	Placebo	66	F	53	0
102	18	3	4	2	6	Placebo	66	F	52	0
103	18	4	8	2	6	Placebo	66	F	57	0
104	18	5	12	2	6	Placebo	66	F	61	0
105	18	6	16	2	6	Placebo	66	F	54	0

patient 19

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
106	19	1	0	2	7	10000U	49	F	42	2
107	19	2	2	2	7	10000U	49	F	42	2
108	19	3	4	2	7	10000U	49	F	43	2
109	19	4	8	2	7	10000U	49	F	33	2
110	19	5	12	2	7	10000U	49	F	37	2
111	19	6	16	2	7	10000U	49	F	43	2

patient 20

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
112	20	1	0	2	8	Placebo	54	F	53	0
113	20	2	2	2	8	Placebo	54	F	56	0
114	20	3	4	2	8	Placebo	54	F	52	0
115	20	4	8	2	8	Placebo	54	F	54	0
116	20	5	12	2	8	Placebo	54	F	55	0
117	20	6	16	2	8	Placebo	54	F	51	0

patient 21

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
118	21	1	0	2	9	5000U	47	F	67	1
119	21	2	2	2	9	5000U	47	F	64	1
120	21	3	4	2	9	5000U	47	F	65	1
121	21	4	8	2	9	5000U	47	F	64	1
122	21	5	12	2	9	5000U	47	F	62	1
123	21	6	16	2	9	5000U	47	F	64	1

patient 22

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
124	22	1	0	2	10	Placebo	31	M	44	0
125	22	2	2	2	10	Placebo	31	M	40	0

126	22	3	4	2	10	Placebo	31	M	32	0
127	22	4	8	2	10	Placebo	31	M	36	0
128	22	5	12	2	10	Placebo	31	M	42	0
129	22	6	16	2	10	Placebo	31	M	43	0

patient 23

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
130	23	1	0	2	11	10000U	53	F	65	2
131	23	2	2	2	11	10000U	53	F	58	2
132	23	3	4	2	11	10000U	53	F	55	2
133	23	5	12	2	11	10000U	53	F	56	2
134	23	6	16	2	11	10000U	53	F	60	2

patient 24

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
135	24	1	0	2	12	5000U	61	M	56	1
136	24	2	2	2	12	5000U	61	M	54	1
137	24	3	4	2	12	5000U	61	M	52	1
138	24	4	8	2	12	5000U	61	M	48	1
139	24	5	12	2	12	5000U	61	M	52	1
140	24	6	16	2	12	5000U	61	M	53	1

patient 25

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
141	25	1	0	2	13	Placebo	40	M	30	0
142	25	2	2	2	13	Placebo	40	M	33	0
143	25	3	4	2	13	Placebo	40	M	25	0
144	25	4	8	2	13	Placebo	40	M	29	0
145	25	5	12	2	13	Placebo	40	M	32	0
146	25	6	16	2	13	Placebo	40	M	32	0

patient 26

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
147	26	1	0	2	14	5000U	67	M	47	1
148	26	3	4	2	14	5000U	67	M	54	1
149	26	4	8	2	14	5000U	67	M	43	1
150	26	5	12	2	14	5000U	67	M	46	1
151	26	6	16	2	14	5000U	67	M	50	1

patient 27

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
152	27	1	0	3	1	10000U	54	F	50	2
153	27	2	2	3	1	10000U	54	F	43	2
154	27	3	4	3	1	10000U	54	F	51	2
155	27	4	8	3	1	10000U	54	F	46	2
156	27	5	12	3	1	10000U	54	F	49	2
157	27	6	16	3	1	10000U	54	F	53	2

patient 28

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
158	28	1	0	3	2	Placebo	41	F	34	0
159	28	2	2	3	2	Placebo	41	F	29	0
160	28	3	4	3	2	Placebo	41	F	27	0
161	28	4	8	3	2	Placebo	41	F	21	0
162	28	5	12	3	2	Placebo	41	F	22	0
163	28	6	16	3	2	Placebo	41	F	22	0

patient 29

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
164	29	1	0	3	3	5000U	66	M	39	1
165	29	2	2	3	3	5000U	66	M	41	1
166	29	3	4	3	3	5000U	66	M	33	1
167	29	4	8	3	3	5000U	66	M	39	1
168	29	5	12	3	3	5000U	66	M	37	1
169	29	6	16	3	3	5000U	66	M	37	1

patient 30

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
170	30	1	0	3	4	Placebo	68	F	43	0
171	30	2	2	3	4	Placebo	68	F	31	0
172	30	3	4	3	4	Placebo	68	F	29	0
173	30	4	8	3	4	Placebo	68	F	28	0
174	30	5	12	3	4	Placebo	68	F	33	0
175	30	6	16	3	4	Placebo	68	F	38	0

patient 31

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
176	31	1	0	3	5	10000U	41	F	46	2
177	31	2	2	3	5	10000U	41	F	26	2
178	31	3	4	3	5	10000U	41	F	29	2
179	31	4	8	3	5	10000U	41	F	33	2
180	31	5	12	3	5	10000U	41	F	45	2
181	31	6	16	3	5	10000U	41	F	56	2

patient 32

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
182	32	1	0	3	6	5000U	77	M	52	1
183	32	2	2	3	6	5000U	77	M	44	1
184	32	3	4	3	6	5000U	77	M	47	1
185	32	4	8	3	6	5000U	77	M	50	1
186	32	5	12	3	6	5000U	77	M	50	1
187	32	6	16	3	6	5000U	77	M	49	1

patient 33

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
188	33	1	0	3	7	10000U	41	M	38	2
189	33	2	2	3	7	10000U	41	M	19	2
190	33	3	4	3	7	10000U	41	M	20	2
191	33	4	8	3	7	10000U	41	M	27	2
192	33	5	12	3	7	10000U	41	M	29	2
193	33	6	16	3	7	10000U	41	M	32	2

patient 34

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
194	34	1	0	3	8	Placebo	56	M	33	0
195	34	2	2	3	8	Placebo	56	M	38	0
196	34	3	4	3	8	Placebo	56	M	40	0
197	34	4	8	3	8	Placebo	56	M	48	0
198	34	5	12	3	8	Placebo	56	M	49	0
199	34	6	16	3	8	Placebo	56	M	44	0

patient 35

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
200	35	1	0	3	9	5000U	46	F	28	1
201	35	2	2	3	9	5000U	46	F	16	1
202	35	3	4	3	9	5000U	46	F	11	1
203	35	4	8	3	9	5000U	46	F	7	1
204	35	5	12	3	9	5000U	46	F	13	1
205	35	6	16	3	9	5000U	46	F	21	1

patient 36

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
206	36	1	0	3	10	10000U	46	F	34	2
207	36	2	2	3	10	10000U	46	F	23	2
208	36	3	4	3	10	10000U	46	F	16	2
209	36	4	8	3	10	10000U	46	F	15	2
210	36	5	12	3	10	10000U	46	F	17	2
211	36	6	16	3	10	10000U	46	F	29	2

patient 37

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
212	37	1	0	3	11	Placebo	47	F	39	0
213	37	2	2	3	11	Placebo	47	F	37	0
214	37	3	4	3	11	Placebo	47	F	39	0
215	37	4	8	3	11	Placebo	47	F	39	0
216	37	5	12	3	11	Placebo	47	F	45	0
217	37	6	16	3	11	Placebo	47	F	43	0

patient 38

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
218	38	1	0	3	12	5000U	35	M	29	1
219	38	2	2	3	12	5000U	35	M	42	1
220	38	3	4	3	12	5000U	35	M	35	1
221	38	4	8	3	12	5000U	35	M	24	1
222	38	5	12	3	12	5000U	35	M	29	1
223	38	6	16	3	12	5000U	35	M	42	1

patient 39

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment

224	39	1	0	4	1	Placebo	58	M	52	0
225	39	2	2	4	1	Placebo	58	M	55	0
226	39	3	4	4	1	Placebo	58	M	51	0
227	39	4	8	4	1	Placebo	58	M	52	0
228	39	5	12	4	1	Placebo	58	M	54	0
229	39	6	16	4	1	Placebo	58	M	57	0

patient 40

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
230	40	1	0	4	2	5000U	62	F	52	1
231	40	2	2	4	2	5000U	62	F	30	1
232	40	3	4	4	2	5000U	62	F	43	1
233	40	4	8	4	2	5000U	62	F	45	1
234	40	5	12	4	2	5000U	62	F	47	1
235	40	6	16	4	2	5000U	62	F	46	1

patient 41

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
236	41	1	0	4	3	10000U	73	F	54	2
237	41	2	2	4	3	10000U	73	F	52	2
238	41	3	4	4	3	10000U	73	F	52	2
239	41	4	8	4	3	10000U	73	F	54	2
240	41	5	12	4	3	10000U	73	F	51	2
241	41	6	16	4	3	10000U	73	F	57	2

patient 42

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
242	42	1	0	4	4	10000U	52	F	52	2
243	42	2	2	4	4	10000U	52	F	44	2
244	42	3	4	4	4	10000U	52	F	33	2
245	42	4	8	4	4	10000U	52	F	54	2
246	42	5	12	4	4	10000U	52	F	46	2
247	42	6	16	4	4	10000U	52	F	47	2

patient 43

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
248	43	1	0	4	5	Placebo	53	F	47	0
249	43	2	2	4	5	Placebo	53	F	45	0
250	43	3	4	4	5	Placebo	53	F	41	0
251	43	4	8	4	5	Placebo	53	F	45	0
252	43	5	12	4	5	Placebo	53	F	43	0
253	43	6	16	4	5	Placebo	53	F	41	0

patient 44

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
254	44	1	0	4	6	5000U	69	M	44	1
255	44	2	2	4	6	5000U	69	M	34	1
256	44	3	4	4	6	5000U	69	M	29	1
257	44	4	8	4	6	5000U	69	M	28	1
258	44	5	12	4	6	5000U	69	M	35	1
259	44	6	16	4	6	5000U	69	M	41	1

patient 45

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
260	45	1	0	4	7	Placebo	55	M	42	0
261	45	2	2	4	7	Placebo	55	M	39	0
262	45	3	4	4	7	Placebo	55	M	38	0
263	45	4	8	4	7	Placebo	55	M	47	0
264	45	5	12	4	7	Placebo	55	M	39	0
265	45	6	16	4	7	Placebo	55	M	39	0

patient 46

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
266	46	1	0	4	8	10000U	52	F	42	2
267	46	2	2	4	8	10000U	52	F	14	2
268	46	3	4	4	8	10000U	52	F	9	2
269	46	4	8	4	8	10000U	52	F	9	2
270	46	5	12	4	8	10000U	52	F	16	2
271	46	6	16	4	8	10000U	52	F	33	2

patient 47

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
272	47	1	0	5	1	10000U	51	F	44	2
273	47	2	2	5	1	10000U	51	F	34	2

274	47	3	4	5	1	10000U	51	F	32	2
275	47	4	8	5	1	10000U	51	F	35	2
276	47	5	12	5	1	10000U	51	F	54	2
277	47	6	16	5	1	10000U	51	F	53	2

patient 48

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
278	48	1	0	5	2	Placebo	56	F	60	0
279	48	2	2	5	2	Placebo	56	F	57	0
280	48	3	4	5	2	Placebo	56	F	53	0
281	48	4	8	5	2	Placebo	56	F	52	0
282	48	5	12	5	2	Placebo	56	F	53	0
283	48	6	16	5	2	Placebo	56	F	58	0

patient 49

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
284	49	1	0	5	3	5000U	65	F	60	1
285	49	2	2	5	3	5000U	65	F	53	1
286	49	3	4	5	3	5000U	65	F	55	1
287	49	4	8	5	3	5000U	65	F	62	1
288	49	5	12	5	3	5000U	65	F	67	1

patient 50

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
289	50	1	0	5	4	10000U	35	F	50	2
290	50	2	2	5	4	10000U	35	F	50	2
291	50	4	8	5	4	10000U	35	F	46	2
292	50	5	12	5	4	10000U	35	F	50	2
293	50	6	16	5	4	10000U	35	F	57	2

patient 51

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
294	51	1	0	5	5	5000U	43	M	38	1
295	51	2	2	5	5	5000U	43	M	27	1
296	51	3	4	5	5	5000U	43	M	16	1
297	51	4	8	5	5	5000U	43	M	19	1
298	51	5	12	5	5	5000U	43	M	23	1
299	51	6	16	5	5	5000U	43	M	26	1

patient 52

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
300	52	1	0	5	6	Placebo	61	M	44	0
301	52	3	4	5	6	Placebo	61	M	46	0
302	52	4	8	5	6	Placebo	61	M	26	0
303	52	5	12	5	6	Placebo	61	M	30	0
304	52	6	16	5	6	Placebo	61	M	34	0

patient 53

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
305	53	1	0	6	1	Placebo	43	M	54	0
306	53	2	2	6	1	Placebo	43	M	53	0
307	53	3	4	6	1	Placebo	43	M	51	0
308	53	4	8	6	1	Placebo	43	M	56	0
309	53	5	12	6	1	Placebo	43	M	39	0
310	53	6	16	6	1	Placebo	43	M	9	0

patient 54

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
311	54	1	0	6	2	10000U	64	F	54	2
312	54	2	2	6	2	10000U	64	F	32	2
313	54	3	4	6	2	10000U	64	F	40	2
314	54	4	8	6	2	10000U	64	F	52	2
315	54	5	12	6	2	10000U	64	F	42	2
316	54	6	16	6	2	10000U	64	F	47	2

patient 55

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
317	55	1	0	6	3	5000U	57	M	56	1
318	55	2	2	6	3	5000U	57	M	55	1
319	55	3	4	6	3	5000U	57	M	44	1
320	55	4	8	6	3	5000U	57	M	50	1
321	55	5	12	6	3	5000U	57	M	53	1
322	55	6	16	6	3	5000U	57	M	52	1

patient 56

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
323	56	1	0	6	4	5000U	60	F	51	1
324	56	2	2	6	4	5000U	60	F	50	1
325	56	3	4	6	4	5000U	60	F	50	1
326	56	4	8	6	4	5000U	60	F	56	1
327	56	5	12	6	4	5000U	60	F	59	1
328	56	6	16	6	4	5000U	60	F	53	1

patient 57

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
329	57	1	0	6	5	10000U	44	F	53	2
330	57	2	2	6	5	10000U	44	F	56	2
331	57	3	4	6	5	10000U	44	F	47	2
332	57	4	8	6	5	10000U	44	F	53	2
333	57	5	12	6	5	10000U	44	F	51	2
334	57	6	16	6	5	10000U	44	F	51	2

patient 58

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
335	58	1	0	6	6	Placebo	41	F	36	0
336	58	2	2	6	6	Placebo	41	F	29	0
337	58	3	4	6	6	Placebo	41	F	24	0
338	58	4	8	6	6	Placebo	41	F	32	0
339	58	5	12	6	6	Placebo	41	F	45	0
340	58	6	16	6	6	Placebo	41	F	36	0

patient 59

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
341	59	1	0	6	7	5000U	51	F	59	1
342	59	2	2	6	7	5000U	51	F	53	1
343	59	3	4	6	7	5000U	51	F	45	1
344	59	4	8	6	7	5000U	51	F	44	1
345	59	5	12	6	7	5000U	51	F	50	1
346	59	6	16	6	7	5000U	51	F	48	1

patient 60

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
347	60	1	0	6	8	Placebo	57	F	49	0
348	60	2	2	6	8	Placebo	57	F	50	0
349	60	3	4	6	8	Placebo	57	F	48	0
350	60	4	8	6	8	Placebo	57	F	56	0
351	60	5	12	6	8	Placebo	57	F	49	0
352	60	6	16	6	8	Placebo	57	F	57	0

patient 61

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
353	61	1	0	6	9	10000U	42	F	50	2
354	61	2	2	6	9	10000U	42	F	38	2
355	61	3	4	6	9	10000U	42	F	42	2
356	61	4	8	6	9	10000U	42	F	43	2
357	61	5	12	6	9	10000U	42	F	42	2
358	61	6	16	6	9	10000U	42	F	46	2

patient 62

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
359	62	1	0	6	10	Placebo	48	F	46	0
360	62	2	2	6	10	Placebo	48	F	48	0
361	62	3	4	6	10	Placebo	48	F	46	0
362	62	4	8	6	10	Placebo	48	F	57	0
363	62	5	12	6	10	Placebo	48	F	57	0
364	62	6	16	6	10	Placebo	48	F	49	0

patient 63

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
365	63	1	0	6	11	10000U	57	M	55	2
366	63	2	2	6	11	10000U	57	M	34	2
367	63	3	4	6	11	10000U	57	M	26	2
368	63	4	8	6	11	10000U	57	M	40	2
369	63	5	12	6	11	10000U	57	M	49	2
370	63	6	16	6	11	10000U	57	M	47	2

patient 64

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
371	64	1	0	6	12	5000U	39	M	46	1

372	64	2	2	6	12	5000U	39	M	44	1	
373	64	3	4	6	12	5000U	39	M	47	1	
374	64	4	8	6	12	5000U	39	M	50	1	
375	64	5	12	6	12	5000U	39	M	46	1	
376	64	6	16	6	12	5000U	39	M	51	1	

patient 65

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
377	65	1	0	6	13	10000U	67	M	34	2
378	65	2	2	6	13	10000U	67	M	31	2
379	65	3	4	6	13	10000U	67	M	25	2

patient 66

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
380	66	1	0	6	14	5000U	39	F	57	1
381	66	2	2	6	14	5000U	39	F	48	1
382	66	3	4	6	14	5000U	39	F	50	1
383	66	4	8	6	14	5000U	39	F	50	1
384	66	5	12	6	14	5000U	39	F	50	1
385	66	6	16	6	14	5000U	39	F	49	1

patient 67

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
386	67	1	0	6	15	Placebo	69	M	41	0
387	67	2	2	6	15	Placebo	69	M	40	0
388	67	3	4	6	15	Placebo	69	M	42	0
389	67	4	8	6	15	Placebo	69	M	38	0
390	67	5	12	6	15	Placebo	69	M	50	0
391	67	6	16	6	15	Placebo	69	M	56	0

patient 68

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
392	68	1	0	7	1	5000U	54	F	49	1
393	68	2	2	7	1	5000U	54	F	25	1
394	68	3	4	7	1	5000U	54	F	30	1
395	68	4	8	7	1	5000U	54	F	41	1
396	68	5	12	7	1	5000U	54	F	41	1
397	68	6	16	7	1	5000U	54	F	31	1

patient 69

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
398	69	1	0	7	2	Placebo	67	F	42	0
399	69	2	2	7	2	Placebo	67	F	30	0
400	69	3	4	7	2	Placebo	67	F	40	0
401	69	4	8	7	2	Placebo	67	F	43	0
402	69	5	12	7	2	Placebo	67	F	36	0
403	69	6	16	7	2	Placebo	67	F	45	0

patient 70

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
404	70	1	0	7	3	10000U	58	F	31	2
405	70	2	2	7	3	10000U	58	F	18	2
406	70	3	4	7	3	10000U	58	F	23	2
407	70	4	8	7	3	10000U	58	F	26	2
408	70	5	12	7	3	10000U	58	F	33	2
409	70	6	16	7	3	10000U	58	F	41	2

patient 71

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
410	71	1	0	7	4	Placebo	72	F	50	0
411	71	2	2	7	4	Placebo	72	F	27	0
412	71	3	4	7	4	Placebo	72	F	43	0
413	71	4	8	7	4	Placebo	72	F	32	0
414	71	5	12	7	4	Placebo	72	F	40	0
415	71	6	16	7	4	Placebo	72	F	47	0

patient 72

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
416	72	1	0	7	5	10000U	65	F	35	2
417	72	2	2	7	5	10000U	65	F	24	2
418	72	3	4	7	5	10000U	65	F	34	2
419	72	4	8	7	5	10000U	65	F	28	2
420	72	5	12	7	5	10000U	65	F	34	2
421	72	6	16	7	5	10000U	65	F	28	2

patient 73

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
422	73	1	0	7	6	5000U	68	F	38	1
423	73	2	2	7	6	5000U	68	F	25	1
424	73	3	4	7	6	5000U	68	F	21	1
425	73	4	8	7	6	5000U	68	F	33	1
426	73	5	12	7	6	5000U	68	F	42	1
427	73	6	16	7	6	5000U	68	F	53	1

patient 74

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
428	74	1	0	7	7	10000U	75	F	53	2
429	74	2	2	7	7	10000U	75	F	40	2
430	74	3	4	7	7	10000U	75	F	38	2
431	74	4	8	7	7	10000U	75	F	44	2
432	74	5	12	7	7	10000U	75	F	47	2
433	74	6	16	7	7	10000U	75	F	53	2

patient 75

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
434	75	1	0	7	8	Placebo	26	F	42	0
435	75	2	2	7	8	Placebo	26	F	48	0
436	75	3	4	7	8	Placebo	26	F	26	0
437	75	4	8	7	8	Placebo	26	F	37	0
438	75	5	12	7	8	Placebo	26	F	37	0
439	75	6	16	7	8	Placebo	26	F	43	0

patient 76

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
440	76	1	0	7	9	5000U	36	F	53	1
441	76	2	2	7	9	5000U	36	F	45	1
442	76	3	4	7	9	5000U	36	F	52	1
443	76	4	8	7	9	5000U	36	F	51	1
444	76	5	12	7	9	5000U	36	F	52	1
445	76	6	16	7	9	5000U	36	F	53	1

patient 77

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
446	77	1	0	7	10	10000U	72	M	46	2
447	77	2	2	7	10	10000U	72	M	47	2
448	77	3	4	7	10	10000U	72	M	45	2
449	77	4	8	7	10	10000U	72	M	45	2
450	77	5	12	7	10	10000U	72	M	50	2
451	77	6	16	7	10	10000U	72	M	52	2

patient 78

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
452	78	1	0	7	11	Placebo	54	F	50	0
453	78	2	2	7	11	Placebo	54	F	42	0
454	78	3	4	7	11	Placebo	54	F	52	0
455	78	4	8	7	11	Placebo	54	F	60	0
456	78	5	12	7	11	Placebo	54	F	54	0
457	78	6	16	7	11	Placebo	54	F	59	0

patient 79

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
458	79	1	0	7	12	5000U	64	F	43	1
459	79	2	2	7	12	5000U	64	F	24	1
460	79	3	4	7	12	5000U	64	F	17	1
461	79	4	8	7	12	5000U	64	F	37	1
462	79	5	12	7	12	5000U	64	F	36	1
463	79	6	16	7	12	5000U	64	F	38	1

patient 80

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
464	80	1	0	8	1	Placebo	39	F	46	0
465	80	2	2	8	1	Placebo	39	F	39	0
466	80	3	4	8	1	Placebo	39	F	25	0
467	80	4	8	8	1	Placebo	39	F	15	0
468	80	5	12	8	1	Placebo	39	F	21	0
469	80	6	16	8	1	Placebo	39	F	25	0

patient 81

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
470	81	1	0	8	2	10000U	54	M	41	2
471	81	2	2	8	2	10000U	54	M	30	2
472	81	3	4	8	2	10000U	54	M	44	2
473	81	4	8	8	2	10000U	54	M	46	2
474	81	5	12	8	2	10000U	54	M	46	2
475	81	6	16	8	2	10000U	54	M	44	2

patient 82

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
476	82	1	0	8	3	5000U	48	M	33	1
477	82	2	2	8	3	5000U	48	M	27	1
478	82	3	4	8	3	5000U	48	M	25	1
479	82	4	8	8	3	5000U	48	M	30	1
480	82	5	12	8	3	5000U	48	M	28	1
481	82	6	16	8	3	5000U	48	M	30	1

patient 83

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
482	83	1	0	8	4	5000U	83	F	36	1
483	83	2	2	8	4	5000U	83	F	15	1
484	83	3	4	8	4	5000U	83	F	16	1
485	83	4	8	8	4	5000U	83	F	17	1
486	83	5	12	8	4	5000U	83	F	22	1
487	83	6	16	8	4	5000U	83	F	41	1

patient 84

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
488	84	1	0	8	5	10000U	74	M	33	2
489	84	2	2	8	5	10000U	74	M	32	2
490	84	3	4	8	5	10000U	74	M	31	2
491	84	4	8	8	5	10000U	74	M	27	2
492	84	5	12	8	5	10000U	74	M	49	2
493	84	6	16	8	5	10000U	74	M	60	2

patient 85

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
494	85	1	0	8	6	Placebo	41	M	37	0

patient 86

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
495	86	1	0	8	7	10000U	65	F	24	2
496	86	2	2	8	7	10000U	65	F	29	2
497	86	3	4	8	7	10000U	65	F	18	2
498	86	4	8	8	7	10000U	65	F	20	2
499	86	5	12	8	7	10000U	65	F	25	2
500	86	6	16	8	7	10000U	65	F	41	2

patient 87

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
501	87	1	0	8	8	5000U	79	M	42	1
502	87	2	2	8	8	5000U	79	M	23	1
503	87	3	4	8	8	5000U	79	M	30	1
504	87	4	8	8	8	5000U	79	M	36	1
505	87	5	12	8	8	5000U	79	M	41	1
506	87	6	16	8	8	5000U	79	M	43	1

patient 88

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
507	88	1	0	8	9	Placebo	63	M	30	0
508	88	2	2	8	9	Placebo	63	M	22	0
509	88	3	4	8	9	Placebo	63	M	21	0
510	88	4	8	8	9	Placebo	63	M	25	0
511	88	5	12	8	9	Placebo	63	M	26	0
512	88	6	16	8	9	Placebo	63	M	33	0

patient 89

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
513	89	1	0	8	10	Placebo	63	F	42	0
514	89	2	2	8	10	Placebo	63	F	46	0
515	89	3	4	8	10	Placebo	63	F	41	0
516	89	4	8	8	10	Placebo	63	F	43	0
517	89	5	12	8	10	Placebo	63	F	49	0
518	89	6	16	8	10	Placebo	63	F	54	0

patient 90

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
519	90	1	0	8	11	10000U	34	F	49	2
520	90	2	2	8	11	10000U	34	F	25	2
521	90	3	4	8	11	10000U	34	F	30	2
522	90	4	8	8	11	10000U	34	F	49	2
523	90	5	12	8	11	10000U	34	F	55	2
524	90	6	16	8	11	10000U	34	F	58	2

patient 91

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
525	91	1	0	8	12	5000U	42	M	58	1
526	91	2	2	8	12	5000U	42	M	46	1
527	91	3	4	8	12	5000U	42	M	46	1
528	91	4	8	8	12	5000U	42	M	50	1
529	91	5	12	8	12	5000U	42	M	56	1
530	91	6	16	8	12	5000U	42	M	60	1

patient 92

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
531	92	1	0	8	13	Placebo	57	M	26	0
532	92	2	2	8	13	Placebo	57	M	26	0
533	92	3	4	8	13	Placebo	57	M	27	0
534	92	4	8	8	13	Placebo	57	M	22	0
535	92	5	12	8	13	Placebo	57	M	38	0
536	92	6	16	8	13	Placebo	57	M	35	0

patient 93

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
537	93	1	0	8	14	5000U	68	M	37	1
538	93	3	4	8	14	5000U	68	M	23	1
539	93	4	8	8	14	5000U	68	M	18	1
540	93	5	12	8	14	5000U	68	M	34	1
541	93	6	16	8	14	5000U	68	M	36	1

patient 94

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
542	94	1	0	8	15	10000U	51	M	40	2
543	94	2	2	8	15	10000U	51	M	24	2
544	94	3	4	8	15	10000U	51	M	25	2
545	94	4	8	8	15	10000U	51	M	37	2
546	94	6	16	8	15	10000U	51	M	38	2

patient 95

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
547	95	1	0	8	16	5000U	51	F	33	1
548	95	2	2	8	16	5000U	51	F	10	1
549	95	3	4	8	16	5000U	51	F	13	1
550	95	4	8	8	16	5000U	51	F	16	1
551	95	5	12	8	16	5000U	51	F	32	1
552	95	6	16	8	16	5000U	51	F	16	1

patient 96

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
553	96	1	0	8	17	10000U	61	F	41	2
554	96	2	2	8	17	10000U	61	F	50	2
555	96	3	4	8	17	10000U	61	F	22	2
556	96	4	8	8	17	10000U	61	F	28	2
557	96	5	12	8	17	10000U	61	F	34	2
558	96	6	16	8	17	10000U	61	F	36	2

patient 97

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
559	97	1	0	8	18	Placebo	42	M	46	0
560	97	3	4	8	18	Placebo	42	M	41	0
561	97	4	8	8	18	Placebo	42	M	41	0
562	97	5	12	8	18	Placebo	42	M	58	0
563	97	6	16	8	18	Placebo	42	M	53	0

patient 98

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
564	98	1	0	8	19	10000U	73	F	40	2
565	98	2	2	8	19	10000U	73	F	28	2
566	98	3	4	8	19	10000U	73	F	29	2
567	98	4	8	8	19	10000U	73	F	30	2

568	98	5	12	8	19	10000U	73	F	37	2
569	98	6	16	8	19	10000U	73	F	44	2

patient 99

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
570	99	1	0	9	1	10000U	57	M	40	2
571	99	2	2	9	1	10000U	57	M	16	2
572	99	3	4	9	1	10000U	57	M	18	2
573	99	4	8	9	1	10000U	57	M	25	2
574	99	5	12	9	1	10000U	57	M	33	2
575	99	6	16	9	1	10000U	57	M	48	2

patient 100

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
576	100	1	0	9	2	Placebo	59	M	61	0
577	100	2	2	9	2	Placebo	59	M	52	0
578	100	3	4	9	2	Placebo	59	M	61	0
579	100	4	8	9	2	Placebo	59	M	68	0
580	100	5	12	9	2	Placebo	59	M	59	0
581	100	6	16	9	2	Placebo	59	M	71	0

patient 101

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
582	101	1	0	9	3	5000U	57	M	35	1
583	101	2	2	9	3	5000U	57	M	21	1
584	101	3	4	9	3	5000U	57	M	29	1
585	101	4	8	9	3	5000U	57	M	30	1
586	101	5	12	9	3	5000U	57	M	35	1
587	101	6	16	9	3	5000U	57	M	48	1

patient 102

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
588	102	1	0	9	4	Placebo	68	F	58	0
589	102	2	2	9	4	Placebo	68	F	38	0
590	102	3	4	9	4	Placebo	68	F	50	0
591	102	4	8	9	4	Placebo	68	F	53	0
592	102	5	12	9	4	Placebo	68	F	47	0
593	102	6	16	9	4	Placebo	68	F	59	0

patient 103

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
594	103	1	0	9	5	5000U	55	F	49	1
595	103	2	2	9	5	5000U	55	F	45	1
596	103	3	4	9	5	5000U	55	F	36	1
597	103	5	12	9	5	5000U	55	F	40	1
598	103	6	16	9	5	5000U	55	F	52	1

patient 104

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
599	104	1	0	9	6	10000U	46	F	52	2
600	104	2	2	9	6	10000U	46	F	46	2
601	104	3	4	9	6	10000U	46	F	36	2
602	104	5	12	9	6	10000U	46	F	45	2
603	104	6	16	9	6	10000U	46	F	54	2

patient 105

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
604	105	1	0	9	7	Placebo	79	F	45	0
605	105	2	2	9	7	Placebo	79	F	46	0
606	105	3	4	9	7	Placebo	79	F	33	0
607	105	4	8	9	7	Placebo	79	F	44	0
608	105	5	12	9	7	Placebo	79	F	46	0
609	105	6	16	9	7	Placebo	79	F	48	0

patient 106

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment
610	106	1	0	9	8	5000U	43	M	67	1
611	106	2	2	9	8	5000U	43	M	63	1
612	106	3	4	9	8	5000U	43	M	71	1
613	106	4	8	9	8	5000U	43	M	66	1
614	106	5	12	9	8	5000U	43	M	68	1
615	106	6	16	9	8	5000U	43	M	71	1

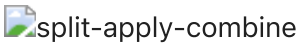
patient 107

group	patient	obs	week	site	id	treat	age	sex	twstrs	treatment

```
616      107    1     0     9    9  10000U   50   M      57          2
617      107    3     4     9    9  10000U   50   M      36          2
618      107    4     8     9    9  10000U   50   M      23          2
619      107    6    16     9    9  10000U   50   M      52          2
patient 108
group      patient  obs  week  site  id   treat  age sex  twstrs  treatment
620      108    1     0     9   10  10000U   39   F      63          2
621      108    2     2     9   10  10000U   39   F      51          2
622      108    3     4     9   10  10000U   39   F      46          2
623      108    4     8     9   10  10000U   39   F      50          2
624      108    5    12     9   10  10000U   39   F      50          2
625      108    6    16     9   10  10000U   39   F      54          2
patient 109
group      patient  obs  week  site  id  treat  age sex  twstrs  treatment
626      109    1     0     9   11  5000U   57   M      53          1
627      109    2     2     9   11  5000U   57   M      38          1
628      109    4     8     9   11  5000U   57   M      33          1
629      109    5    12     9   11  5000U   57   M      36          1
630      109    6    16     9   11  5000U   57   M      51          1
```

A common data analysis procedure is the **split-apply-combine** operation, which groups subsets of data together, applies a function to each of the groups, then recombines them into a new data table.

For example, we may want to aggregate our data with with some function.



(figure taken from "Python for Data Analysis", p.251)

We can aggregate in Pandas using the `aggregate` (or `agg`, for short) method:

In [74]:
```
cdystonia_grouped = cdystonia.drop(columns=['treat', 'sex']).groupby(cdystonia.patient)
cdystonia_grouped.agg('mean').head()
cdystonia_grouped.mean().head() #equivalent
```

Out[74]:

patient	patient	obs	week	site	id	age	twstrs	treatment
1	1.0	3.5	7.0	1.0	1.0	65.0	33.000000	1.0
2	2.0	3.5	7.0	1.0	2.0	70.0	47.666667	2.0
3	3.0	3.5	7.0	1.0	3.0	64.0	30.500000	1.0
4	4.0	2.5	3.5	1.0	4.0	59.0	60.000000	0.0
5	5.0	3.5	7.0	1.0	5.0	76.0	46.166667	2.0

Since it does not make sense to aggregate string variables, we dropped the `treat` and `sex` variables and they are **not included in the aggregation**.

Some aggregation functions are so common that Pandas has a convenience method for them, such as `mean`:

The `add_prefix` and `add_suffix` methods can be used to give the columns of the resulting table labels that reflect the transformation:

In [75]:
```
cdystonia_grouped.mean().add_suffix('_mean').head()
```

Out[75]:

patient	patient_mean	obs_mean	week_mean	site_mean	id_mean	age_mean	twstrs_mean	treatment_r

patient							
1	1.0	3.5	7.0	1.0	1.0	65.0	33.000000
2	2.0	3.5	7.0	1.0	2.0	70.0	47.666667
3	3.0	3.5	7.0	1.0	3.0	64.0	30.500000
4	4.0	2.5	3.5	1.0	4.0	59.0	60.000000
5	5.0	3.5	7.0	1.0	5.0	76.0	46.166667

In [76]:
```
# The median of the `twstrs` variable
cdystonia_grouped['twstrs'].quantile(0.5).head(10)
```

Out[76]:
```
patient
1     34.0
2     50.5
3     30.5
4     61.5
5     48.5
6     48.0
7     42.0
8     32.5
9     35.5
10    20.5
Name: twstrs, dtype: float64
```

If we wish, we can easily aggregate according to multiple keys:

In [77]:
```
cdystonia.drop(columns=['treat', 'sex']).groupby(['week','site']).mean().head()
```

Out[77]:

		patient	obs	id	age	twstrs	treatment
week	**site**						
0	**1**	6.5	1.0	6.5	59.000000	43.083333	1.000000
	2	19.5	1.0	7.5	53.928571	51.857143	0.928571
	3	32.5	1.0	6.5	51.500000	38.750000	1.000000
	4	42.5	1.0	4.5	59.250000	48.125000	1.000000
	5	49.5	1.0	3.5	51.833333	49.333333	1.000000

Apply

We can generalize the split-apply-combine methodology by using `apply` function. This allows us to invoke any function we wish on a grouped dataset and recombine them into a DataFrame.

The function below takes a DataFrame and a column name, sorts by the column, and takes the `n` largest values of that column. We can use this with `apply` to return the largest values from every group in a DataFrame in a single call.

In [78]:
```
def top(df, column, n=5):
    return df.sort_values(by=column, ascending=False)[:n]
```

To see this in action, consider the vessel transit segments dataset (which we merged with the vessel information to yield `segments_merged`). Say we wanted to return the 3 longest segments travelled by each ship:

In [79]:
```
top3segments = segments_merged.groupby('mmsi').apply(top, column='seg_length', n=3)[['na
top3segments.head(15)
```

Out[79]:

		names	seg_length
mmsi			
1	**6**	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	76.0
	5	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	17.4
	7	Bil Holman Dredge/Dredge Capt Frank/Emo/Offsho...	13.7
9	**15**	000000009/Raven/Shearwater	47.2
	14	000000009/Raven/Shearwater	31.4
	13	000000009/Raven/Shearwater	19.3
21	**16**	Us Gov Vessel	48.7
	25	Us Gov Vessel	25.3
	30	Us Gov Vessel	21.7
74	**35**	Mcfaul/Sarah Bell	7.4
	34	Mcfaul/Sarah Bell	1.4
103	**37**	Ron G/Us Navy Warship 103/Us Warship 103	87.5
	41	Ron G/Us Navy Warship 103/Us Warship 103	62.6
	43	Ron G/Us Navy Warship 103/Us Warship 103	59.1
310	**51**	Arabella	77.4

Notice that additional arguments for the applied function can be passed via `apply` after the function name. It assumes that the DataFrame is the first argument.

References

Python for Data Analysis Wes McKinney