Source lessons

1. Learn this first https://pandas.pydata.org/pandas-docs/stable/dsintro.html#dsintro (https://pandas.pydata.org/pandas-docs/stable/dsintro.html#dsintro)
2. Then pandas 10-min intro https://pandas.pydata.org/pandas-docs/stable/10min.html (https://pandas.pydata.org/pandas-docs/stable/10min.html)

miscellaneous pandas and Heidi's data notes

- NaN (not a number) is standard replacement for missing data
- Could be useful for devices that tend to capture great amounts of missing data
- insulin pump example - rows can look very different depending on what was happening
- a small portion of a non-blood-glucose event looks like:

```
15/02/18
17:57:26,,,,,,,,,,,,,,,,,,,,,,,,,,,,,PLGMControllerState,"BEGIN_INTERNAL_STATE
```

- and a blood-glucose row starts like this:

```
16/02/18 11:02:25,,,,,,,,,,,,,,,,,,,,,,,,,9.1,24.93,,GlucoseSensorData
```

- The raw data is available but quite full of a wide variety of data
- The only thing I usually care about in analysis of a day or a week's worth of data is the 9.1 part (blood glucose)

types of structures

Series

- 1-D series, labeled
- index is axis labels
- to create array: `s = pd.Series(data, index=index)`
- if index is omitted, pandas inserts numbered index starting at 0
- contents could be a series of numbers, strings, dictionary, ndarray, etc
- if series is an ndarray, index must have same length as the series

DataFrame

- 2-D array, labeled
- columns can be different types, much like a spreadsheet
- more complex than NumPy's ndarray

In [1]:
```
# usual imports
import numpy as np
import pandas as pd
```

In [2]:
```
# series example, with (ndarray, index) given
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
# with an ndarray in a Series, data and index need equal length
s
```

Out[2]:
```
a    0.093096
b    0.115783
c   -1.673176
d    1.306117
e    0.330326
dtype: float64
```

In [3]:
```
s.index
```

Out[3]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [4]:
```
pd.Series(np.random.randn(5))
```

Out[4]:
```
0   -0.577213
1   -0.271418
2    1.417345
3    0.136241
4   -0.178144
dtype: float64
```

In [5]:
```
# dict example
# data for labels come from index passed, or failing that, from keys in the dict
d = {'a' : 0., 'b' : 1., 'c' : 2.}
pd.Series(d)
```

Out[5]:
```
a    0.0
b    1.0
c    2.0
dtype: float64
```

In [6]:
```
# scalar value example
# must be given index in .Series call
# ie. data = 5; it is repeated for all the index values
pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])
```

Out[6]:
```
a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

In [7]:
```
# let's make a new, more interesting structure
# this one is number of minutes of guitar practice per day
# it will become a pandas Series
minutes = {'Monday' : 34, 'Tuesday' : 12, 'Wednesday' : 28, 'Thursday' : 0, 'Friday'
minutes # note Python doesn't keep order in a dictionary
```

Out[7]:
```
{'Friday': 30, 'Monday': 34, 'Thursday': 0, 'Tuesday': 12, 'Wednesday': 28}
```

In [8]:
```
minutes_s = pd.Series(minutes) # create Series from dictionary
minutes_s
```

Out[8]:
```
Friday       30
Monday       34
Thursday      0
Tuesday      12
Wednesday    28
dtype: int64
```

In [9]:
```
# .series() acts like an ndarray
# but slicing also slices the index
minutes_s[0] # index 0, isn't necessarily the first item contained in the original di
# in this case, the elements appear to be ordered alphabetically, Friday first
```

Out[9]:
```
30
```

In [10]:
```
# luckily we can select or slice by label rather than numerical index
print("You practiced", minutes_s['Monday'], "minutes on Monday.")
```

```
You practiced 34 minutes on Monday.
```

In [11]:
```
minutes_s[:3] # first three values
```

Out[11]:
```
Friday      30
Monday      34
Thursday     0
dtype: int64
```

In [12]:
```
# find Series contents that are above median
# the dictionary had 5 days, and the median value is 12
minutes_s[minutes_s > minutes_s.median()] # [] contains slice of data above median
# in practice so far, the days of the week appear in order
# but to ensure elements are kept in order, a list of tuples would be better
```

Out[12]:
```
Friday    30
Monday    34
dtype: int64
```

In [13]:
```
type(minutes_s) # on that note, let's see what type of object this is
```

Out[13]:
```
pandas.core.series.Series
```

In [14]:
```
minutes_s[[4, 0]] # slice indexes in a certain order
# useful here if you knew Wednesday and Friday (index 4 and 0) had guitar lessons
```

Out[14]:
```
Wednesday    28
Friday       30
dtype: int64
```

In [15]:
```
'Saturday' in minutes_s # basic Python applies to a pandas Series too
```

Out[15]:
```
False
```

In [16]:
```
'Monday' in minutes_s
```

Out[16]:
```
True
```

In [17]:
```
# guitar practice minutes Series can receive arithmetic operations too
# what if you vowed to practice twice as much?
minutes_s2 = minutes_s * 2
minutes_s2
```

Out[17]:
```
Friday       60
Monday       68
Thursday      0
Tuesday      24
Wednesday    56
dtype: int64
```

In [18]:
```
# how many days are over an hour?
minutes_s2 > 60
```

Out[18]:
```
Friday       False
Monday        True
Thursday     False
Tuesday      False
Wednesday    False
dtype: bool
```

In [19]:
```
# a series can accept a list and a name attribute
s = pd.Series([56, 70, 80, 97], index =['July', 'August', 'September', 'December'], r
s
```

Out[19]:
```
July         56
August       70
September    80
December     97
Name: Hydro bill, dtype: int64
```

In [20]:
```
# a series can be renamed
s2 = s.rename("Kilometers walked")
s2.name # find out what s2's name is
```

Out[20]:
```
'Kilometers walked'
```

In [21]:
```
s2
```

Out[21]:
```
July         56
August       70
September    80
December     97
Name: Kilometers walked, dtype: int64
```