week3

October 15, 2017

In [87]:
```
# As always, start with the following

import pandas as pd

# Let's load the weather data
# Notice how you can load directly from a url
weather_df = pd.read_csv("https://github.com/vega/vega-datasets/raw/gh-pag
```

1 Indexing/Slicing Dataframes

- Using square brackets
- Using loc
- Using iloc

2 Using Square Brackets

In [88]:
```
# Labels are used to select columns
weather_df["location"].head()
```

Out[88]:
```
0    Seattle
1    Seattle
2    Seattle
3    Seattle
4    Seattle
Name: location, dtype: object
```

In [89]:
```
# List or tuple of labels can also be passed to select multiple columns ar

weather_df[["wind","location"]].head()
```

Out[89]:

	wind	location
0	4.7	Seattle
1	4.5	Seattle
2	2.3	Seattle
3	4.7	Seattle
4	6.1	Seattle

```
In [90]: # When using numeric slicing, it is for selecting/slicing rows
         # Works exactly like python list slicing
         weather_df[1:5]

         #Can you fetch the last 5 records?
```

Out[90]:

	location	date	precipitation	temp_max	temp_min	wind	weat
1	Seattle	2012-01-02 00:00	10.9	10.6	2.8	4.5	r
2	Seattle	2012-01-03 00:00	0.8	11.7	7.2	2.3	r
3	Seattle	2012-01-04 00:00	20.3	12.2	5.6	4.7	r
4	Seattle	2012-01-05 00:00	1.3	8.9	2.8	6.1	r

3 Using loc

Used for **labeled** slicing of both rows and columns

NOTE: loc is used with square brackets

```
In [91]: # Fetch rows based on index number Look at the far left column

         weather_df.loc[1]
```

```
Out[91]: location                   Seattle
         date             2012-01-02 00:00
         precipitation                10.9
         temp_max                     10.6
         temp_min                      2.8
         wind                          4.5
         weather                      rain
         Name: 1, dtype: object
```

```
In [92]: # You can also use the python list slicing syntax to fetch multiple rows
         weather_df.loc[5:10]
```

Out[92]:

	location	date	precipitation	temp_max	temp_min	wind	wea
5	Seattle	2012-01-06 00:00	2.5	4.4	2.2	2.2	
6	Seattle	2012-01-07 00:00	0.0	7.2	2.8	2.3	
7	Seattle	2012-01-08 00:00	0.0	10.0	2.8	2.0	
8	Seattle	2012-01-09 00:00	4.3	9.4	5.0	3.4	
9	Seattle	2012-01-10 00:00	1.0	6.1	0.6	3.4	
10	Seattle	2012-01-11 00:00	0.0	6.1	-1.1	5.1	

```
In [93]: # You can use a list of ids to fetch
         weather_df.loc[[1,5,7,10]]
```

Out[93]:

	location	date	precipitation	temp_max	temp_min	wind	wea
1	Seattle	2012-01-02 00:00	10.9	10.6	2.8	4.5	
5	Seattle	2012-01-06 00:00	2.5	4.4	2.2	2.2	
7	Seattle	2012-01-08 00:00	0.0	10.0	2.8	2.0	
10	Seattle	2012-01-11 00:00	0.0	6.1	-1.1	5.1	

```
In [ ]: # Negative indecies don't work!

        weather_df.loc[-1]

        # Why?

In [ ]: # you can also set the column you want like so
        weather_df.loc[1:5, ["location","weather"]]
```

4 Using iloc

It is exactly like loc, but uses numeric indecies

```
In [ ]: weather_df.loc[1:5]

In [ ]: # Negative indicies work this time!
        weather_df.iloc[-1]

        # why?

In [ ]: # selecting columns is also numeric
        weather_df.iloc[:,0:2].head()
```

5 Sorting

- Sort rows based on values of column(s)
- Descending or ascending order

```
In [ ]: # Sort based on temp_max
        weather_df.sort_values(by="temp_max").head()

        # is this ascending or  descending order?

In [ ]: # To sort in descending order, set ascending argument to False
        weather_df.sort_values(by="temp_max", ascending=False).head()

        # Seems like New York has the highest and lowest tempratures!

In [ ]: # Sort weather_df by percipitation in ascending order

In [ ]: # Notice how percipitation is 0 for many observations
        # To sort by percipitation, then by wind speed, both ascending, do the foll
        weather_df.sort_values(by=["precipitation", "wind"]).head()

In [ ]: # To sort by percipitation ascending, then by wind speed descending, do the
        weather_df.sort_values(by=["precipitation", "wind"], ascending=[True, False
```

6 Filteration

Selecting rows based on logical conditions. e.g., weather observations in New York, or observations where wind speed is higher than 10

You use conditions very similar to Python conditions in syntax, with some slight variation

```
In [ ]: # to fetch observations for New York
        weather_df[weather_df["location"] == "New York"].head()

In [ ]: # Perform the same filter using dot notation

In [ ]: # Filter observations where wind is higher than 10

In [ ]: # Now try to find out how many observations there are using 2 different met

In [ ]: # Filter all observations where temp_min is less than or equal to zero and
        weather_df[(weather_df.temp_min < 0) & (weather_df.weather == "rain")].head

        # The parantheses are important!

In [ ]: # You can write it over multiple lines to be easier to read

        weather_df[
                (weather_df.temp_min < 0) &
                (weather_df.weather == "rain")
            ].head()

In [ ]: # You can also use 'or' in the condition
        weather_df[
                (weather_df.temp_min < 0) &
                (
                    (weather_df.weather == "rain") |
                    (weather_df.weather == "snow")
                )
            ].head(10)
```

7 Some useful functions used in filteration

- isin(values)
- isnull(), notnull()
- duplicated

You can use these in filteration conditions

```
In [ ]: # Filter using method isin to find observations where whether is either ra
```

8 Data Manipulation

- Operations can be performed on columns
- All values in a column will have the same operation performed on them
- When operating on two or more columns, the operations are performed on items in the same position
  - Columns must match in size

9 Useful methods and Operators

- Almost all the mathematical operators are available
- Useful methods to perform calculations on columns are:
  - max, min, mean, median, mode, std, var, count, sum, mod
- Method **apply** will accept a function that takes a single argument, and returns a value
  - The function is applied to every item in the column and a new column is created with the results
- Useful methods to clean the dataframe are:
  - dropna, drop_duplicates, fillna

```
In [ ]: # Calculating the temprature range


        # Try to store it in a column called temp_range
        # be sure to try dot and index notations

In [ ]: # Calculate the mean range and store it in a column called mean_range

In [ ]: # Calculate the mean centered value of range
        # mean centering = temp_range - mean_range
        # tells us how much the observation is different from the mean
        # name the collumn mc_range

In [ ]: # Caclulate the square of mc_range
        # name the new column mc_range_sq

In [ ]: # calculate the natural log of mc_range and use name mc_range_log
        # tip: search numpy
        # be sure to examine the data, what do you see?
        # What should you do?
        # Is fillna(0) a good idea?

In [ ]: # based on what you know so far,
        # try to plot range, mc_range, and mc_range_sq

In [ ]: # try to plot the distributions for the new range columns
        # hint: search for histograms
```

```
In [ ]: # try to count the number of observations where the temprature change is ab

        # can you produce a scalar value instead of a column?

        # Can you calculate the ratio?

        # Can you calculate the percentage?

In [ ]: # Calculate the average temprature for the day
        # hint: use temp_max and temp_min

In [ ]: # plot the average temp

In [ ]: # plot the distribution for average temp

In [ ]: # compare the distribution of average temp with mc_range

In [ ]: # find the days in which the average temprature is below zero and it is sno
        # calculate the percentage of these days
```