Assignment 7: Follow Along (7-2 -- 7-9)

7-2: Dataframe Basics

In [1]:
```
import pandas as pd
df = pd.read_csv("7-2_weather_data.csv") # You can also make a data dict
ionary
df
```

Out[1]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

In [2]:
```
df.shape
```

Out[2]: (6, 4)

In [4]:
```
rows, columns = df.shape #how to print just rows or columns
```

In [5]:
```
rows
```

Out[5]: 6

In [6]:
```
columns
```

Out[6]: 4

In [7]:
```
df.head() #prints a smaller amound of rows for convenience
```

Out[7]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain

In [8]:
```
df.head(2)
```

Out[8]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny

In [11]:
```
df.tail() # prints last 5 rows
```

Out[11]:

	day	temperature	windspeed	event
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

In [12]:
```
df.tail(1)
```

Out[12]:

	day	temperature	windspeed	event
5	1/6/2017	31	2	Sunny

In [14]:
```
df[2:5] # How to print a section, make sure to use brackets
```

Out[14]:

	day	temperature	windspeed	event
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain

In [15]:
```
df[:] # prints everything
```

Out[15]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

In [16]:
```
df # also prints everything
```

Out[16]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

In [17]:
```
df.columns
```

Out[17]:
```
Index(['day', 'temperature', 'windspeed', 'event'], dtype='object')
```

In [18]:
```
df.day
```

Out[18]:
```
0    1/1/2017
1    1/2/2017
2    1/3/2017
3    1/4/2017
4    1/5/2017
5    1/6/2017
Name: day, dtype: object
```

In [19]:
```
df.event
```

Out[19]:
```
0     Rain
1    Sunny
2     Snow
3     Snow
4     Rain
5    Sunny
Name: event, dtype: object
```

In [20]:
```
df['event'] # this formatting also works
```

Out[20]:
```
0     Rain
1    Sunny
2     Snow
3     Snow
4     Rain
5    Sunny
Name: event, dtype: object
```

In [21]:
```
type(df['event']) # reference information
```

Out[21]:
```
pandas.core.series.Series
```

In [23]:
```
df[['event','day']] # how to print certain columns, needs two brackets!
```

Out[23]:

	event	day
0	Rain	1/1/2017
1	Sunny	1/2/2017
2	Snow	1/3/2017
3	Snow	1/4/2017
4	Rain	1/5/2017
5	Sunny	1/6/2017

In [24]:
```
df['temperature'].max()
```

Out[24]:
```
35
```

In [25]:
```
df['temperature'].min()
```

Out[25]:
```
24
```

In [26]:
```
df['temperature'].std()
```

Out[26]:
```
3.8297084310253524
```

In [27]:
```
df.describe() # this just prints statistics on your data
```

Out[27]:

	temperature	windspeed
count	6.000000	6.000000
mean	30.333333	4.666667
std	3.829708	2.338090
min	24.000000	2.000000
25%	28.750000	2.500000
50%	31.500000	5.000000
75%	32.000000	6.750000
max	35.000000	7.000000

In [28]:
```
df[df.temperature>=32]
```

Out[28]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
4	1/5/2017	32	4	Rain

In [29]:
```
df.temperature>=32
# note how this still answers question but formatting is not nice. THa
t's why above method preferred.
```

Out[29]:
```
0     True
1     True
2    False
3    False
4     True
5    False
Name: temperature, dtype: bool
```

In [30]:
```
df[df.temperature==df.temperature.max()] #conditionally select data
```

Out[30]:

	day	temperature	windspeed	event
1	1/2/2017	35	7	Sunny

In [34]:
```
df[df.temperature==df['temperature'].max()]
#does same thing, have to format like this if data set has spaces/gaps
```

Out[34]:

	day	temperature	windspeed	event
1	1/2/2017	35	7	Sunny

In [35]:
```
df['day'][df.temperature==df['temperature'].max()]
#you can also restrict to the data field that you actually want to see
```

Out[35]:
```
1    1/2/2017
Name: day, dtype: object
```

In [38]:
```
df[['day','temperature']][df.temperature==df['temperature'].max()]
# need to add extra brackets when adding temp, don't know why
```

Out[38]:

	day	temperature
1	1/2/2017	35

In [39]:
```
df.index
```

Out[39]:
```
RangeIndex(start=0, stop=6, step=1)
```

In [41]:
```
df
```

Out[41]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

In [43]:
```
df.set_index('day')
#not brackets
#not how this gets rid of random number column on left
#makes a copy of old data frame, does not change original
```

Out[43]:

	temperature	windspeed	event
day			
1/1/2017	32	6	Rain
1/2/2017	35	7	Sunny
1/3/2017	28	2	Snow
1/4/2017	24	7	Snow
1/5/2017	32	4	Rain
1/6/2017	31	2	Sunny

In [48]:
```
df.set_index('day', inplace=True)
#note all within parenthesis
```

In [49]:
```
df.loc['1/4/2017']
# this only works if inplace=True was set
```

Out[49]:
```
temperature      24
windspeed         7
event          Snow
Name: 1/4/2017, dtype: object
```

In [50]:
```
df.reset_index(inplace=True)
#this reverses the set_index
```

In [51]:
```
df #see!
```

Out[51]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow
3	1/4/2017	24	7	Snow
4	1/5/2017	32	4	Rain
5	1/6/2017	31	2	Sunny

In [52]:
```
df.set_index('event', inplace=True)
df
# Note how this gets rid of numbers on left, but event is on the left
```

Out[52]:

	day	temperature	windspeed
event			
Rain	1/1/2017	32	6
Sunny	1/2/2017	35	7
Snow	1/3/2017	28	2
Snow	1/4/2017	24	7
Rain	1/5/2017	32	4
Sunny	1/6/2017	31	2

In [54]:
```
df.loc['Snow']
#will print all occurences where true
```

Out[54]:

	day	temperature	windspeed
event			
Snow	1/3/2017	28	2
Snow	1/4/2017	24	7

7-3: Different Ways of Creating DataFrame

In [58]:
```
import pandas as pd
df = pd.read_csv("7-3_weather_data.csv")
df # same as 7-2 but slightly different set
```

Out[58]:

	day	temperature	windspeed	event	Unnamed: 4
0	1/1/2017	32	6	Rain	NaN
1	1/2/2017	35	7	Sunny	NaN
2	1/3/2017	28	2	Snow	NaN

In [59]:
```
df = pd.read_excel("7-3_weather_data.xlsx")
df
#just a different format that document is saved as
```

Out[59]:

	day	temperature	windspeed	event
0	2017-01-01	32	6	Rain
1	2017-01-02	35	7	Sunny
2	2017-01-03	28	2	Snow

In [63]:
```
weather_data= {
    'day' : ['1/1/2017', '1/2/2017', '1/3/2017'],
    'temperature' : [32, 35, 28],
    'windspeed' : [6, 7, 2],
    'event' : ['Rain', 'Sunny', 'Snow']
} # could also creat dictionary, but seems more work intensive
df = pd.DataFrame(weather_data)
df
```

Out[63]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow

In [64]:
```
#fourth method to creat a data frame
#each element in list is a tuple, represent rows in dataframe
weather_data = [
    ('1/1/2017',32,6,'Rain'),
    ('1/2/2017',35,7,'Sunny'),
    ('1/3/2017',28,2,'Snow')
]
df = pd.DataFrame(weather_data, columns=["day", "temperature", "windspee
d", "event"])
# Need to provide column names
df
```

Out[64]:

	day	temperature	windspeed	event
0	1/1/2017	32	6	Rain
1	1/2/2017	35	7	Sunny
2	1/3/2017	28	2	Snow

In [65]:
```
#fifth method: list of dictionaries
weather_data = [
    {"day" : "1/1/2017", "temperature" : 32, "windspeed" : 6, "event" :
"Rain"},
    {"day" : "1/2/2017", "temperature" : 35, "windspeed" : 7, "event" :
"Sunny"},
    {"day" : "1/3/2017", "temperature" : 28, "windspeed" : 2, "event" :
"Snow"},
]
df = pd.DataFrame(weather_data)
df
#there are more ways to create dataframes, but we're only looking at the
se five
```

Out[65]:

	day	event	temperature	windspeed
0	1/1/2017	Rain	32	6
1	1/2/2017	Sunny	35	7
2	1/3/2017	Snow	28	2

7-4: Read Write Excel CSV File

In [81]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx")
df
# you'll run into issues if you have rows with like a title
# I used excel because csv was importing a random empty sixth column
```

Out[81]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

In [82]:
```
# how to skip a row, if necessary
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", skiprows=1)
df
#see below!
```

Out[82]:

	GOOGL	27.82	87	845	larry page
0	WMT	4.61	484	65	n.a.
1	MSFT	-1	85	64	bill gates
2	RIL	not available	50	1023	mukesh ambani
3	TATA	5.6	-1	n.a.	ratan tata

In [83]:
```
# how to skip a row, if necessary
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", header=1)
df
#another option
```

Out[83]:

	GOOGL	27.82	87	845	larry page
0	WMT	4.61	484	65	n.a.
1	MSFT	-1	85	64	bill gates
2	RIL	not available	50	1023	mukesh ambani
3	TATA	5.6	-1	n.a.	ratan tata

In [84]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", skiprows=1, header=None)
df
# Will automatically generate columns, so you can set names
#skipped row of column titles just to demonstrate
```

Out[84]:

	0	1	2	3	4
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

In [85]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", skiprows=1, header=None, names
=["ticker","eps", "revenue", "price", "people",])
df
```

Out[85]:

	ticker	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

In [86]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", nrows=3)
df
#setting the number of rows you want to read (excludes header)
```

Out[86]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1.00	85	64	bill gates

In [87]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", na_values=["not available",
"n.a."])
df
#in some of the cells, those cells are not avaliable,
#note how those values now are NaN
```

Out[87]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845.0	larry page
1	WMT	4.61	484	65.0	NaN
2	MSFT	-1.00	85	64.0	bill gates
3	RIL	NaN	50	1023.0	mukesh ambani
4	TATA	5.60	-1	NaN	ratan tata

In [102]:
```
#revenue cannot be zero, so we know error there, how do we change that w
ithout the one in eps?
import pandas as pd
df = pd.read_csv("7-4_stock_data.csv", na_values={
    "eps": ["not available", "n.a."],
    "revenue": ["not available", "n.a.", -1],
    "people" : ["not available", "n.a."],
    "price" : ["not available", "n.a."],
    })
df
# I have no idea why my -1 is still showing up with excel, but is gone w
ith csv
```

Out[102]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87.0	845.0	larry page
1	WMT	4.61	484.0	65.0	NaN
2	MSFT	-1.00	85.0	64.0	bill gates
3	RIL	NaN	50.0	1023.0	mukesh ambani
4	TATA	5.60	NaN	NaN	ratan tata

In [100]:
```
#how to export new dataframe
#automatically add yours index, unless index=false
df.to_excel("new.xlsx", index=False)
```

In [101]:
```
df.columns
```

Out[101]:
```
Index(['tickers', 'eps', 'revenue', 'price', 'people'], dtype='object')
```

In [258]:
```
#exporting only part of a dataframe
df.to_csv("new.csv", columns=["tickers", "eps"])
```

```
---------------------------------------------------------------------
----
KeyError                                  Traceback (most recent call l
ast)
<ipython-input-258-5d0bb5ea2485> in <module>()
      1 #exporting only part of a dataframe
----> 2 df.to_csv("new.csv", columns=["tickers", "eps"])

~/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py in to_csv
(self, path_or_buf, sep, na_rep, float_format, columns, header, index,
 index_label, mode, encoding, compression, quoting, quotechar, line_ter
minator, chunksize, tupleize_cols, date_format, doublequote, escapecha
r, decimal)
   1742                                      date_format=date_format,
   1743                                      doublequote=doublequote,
-> 1744                                      escapechar=escapechar, decimal
=decimal)
   1745         formatter.save()
   1746

~/anaconda3/lib/python3.6/site-packages/pandas/io/formats/csvs.py in __
init__(self, obj, path_or_buf, sep, na_rep, float_format, cols, header,
 index, index_label, mode, nanRep, encoding, compression, quoting, line
_terminator, chunksize, tupleize_cols, quotechar, date_format, doublequ
ote, escapechar, decimal)
     83             else:
     84                 cols = list(cols)
---> 85             self.obj = self.obj.loc[:, cols]
     86
     87         # update columns to include possible multiplicity of du
pes

~/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py in __ge
titem__(self, key)
   1470             except (KeyError, IndexError):
   1471                 pass
-> 1472             return self._getitem_tuple(key)
   1473         else:
   1474             # we by definition only have the 0th axis

~/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py in _get
item_tuple(self, tup)
    888                 continue
    889
--> 890             retval = getattr(retval, self.name)._getitem_axis(k
ey, axis=i)
    891
    892         return retval

~/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py in _get
item_axis(self, key, axis)
   1899                     raise ValueError('Cannot index with multidi
mensional key')
   1900
-> 1901                 return self._getitem_iterable(key, axis=axis)
   1902
   1903             # nested tuple slicing
```

```
~/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py in _get
item_iterable(self, key, axis)
   1141             if labels.is_unique and Index(keyarr).is_unique:
   1142                 indexer = ax.get_indexer_for(key)
-> 1143                 self._validate_read_indexer(key, indexer, axis)
   1144
   1145                 d = {axis: [ax.reindex(keyarr)[0], indexer]}

~/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py in _val
idate_read_indexer(self, key, indexer, axis)
   1204                 raise KeyError(
   1205                     u"None of [{key}] are in the [{axis}]".form
at(
-> 1206                         key=key, axis=self.obj._get_axis_name(a
xis)))
   1207
   1208             # we skip the warning on Categorical/Interval

KeyError: "None of [['tickers', 'eps']] are in the [columns]"
```

In [112]:
```
#no header
df.to_csv("new.csv", header=False)
```

In [103]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx", "Sheet1")
df
```

Out[103]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

In [104]:
```
import pandas as pd
df = pd.read_excel("7-4_stock_data.xlsx")
df
#why is Sheet1 included in his example?
```

Out[104]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

In [105]:
```
import pandas as pd
def convert_people_cell(cell):
    if cell=="n.a.":
        return "sam walton"
    return cell
df = pd.read_excel("7-4_stock_data.xlsx", "Sheet1", converters = {"peopl
e" : convert_people_cell})
df
```

Out[105]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	sam walton
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

In [108]:
```
import pandas as pd

def convert_people_cell(cell):
    if cell=="n.a.":
        return "sam walton"
    return cell

def convert_eps_cell(cell):
    if cell=="not available":
        return None
    return cell

df = pd.read_excel("7-4_stock_data.xlsx", "Sheet1", converters = {
    "people" : convert_people_cell,
    "eps" : convert_eps_cell
    })
df
```

Out[108]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	sam walton
2	MSFT	-1.00	85	64	bill gates
3	RIL	NaN	50	1023	mukesh ambani
4	TATA	5.60	-1	n.a.	ratan tata

In [116]:
```
# How to move where the data appears on the excel chart
df.to_excel("new.xlsx", sheet_name="stocks", startrow=1, startcol=2, ind
ex=False)
```

In [119]:
```
import pandas as pd
df_stocks = pd.read_excel("7-4_stock_data.xlsx")
df_weather = pd.read_excel("7-4_stocks_weather.xlsx")
with pd.ExcelWriter("7-4_stocks_weather.xlsx") as writer:
    df_stocks.to_excel(writer, sheet_name="stocks")
    df_weather.to_excel(writer, sheet_name="weather")
# weather sheet has no data when I opened it, I assume this is because I
 tried to import the information through read_excel
```

7-5: Handle Missing Data: fillna, dropna, interpolate

In [120]:
```
import pandas as pd
df = pd.read_csv("7-5_weather_data.csv")
df
```

Out[120]:

	day	temperature	windspeed	event
0	1/1/2017	32.0	6.0	Rain
1	1/4/2017	NaN	7.0	Sunny
2	1/5/2017	28.0	NaN	Snow
3	1/6/2017	NaN	7.0	NaN
4	1/7/2017	32.0	NaN	Rain
5	1/8/2017	31.0	2.0	Sunny
6	1/9/2017	NaN	NaN	NaN
7	1/10/2017	34.0	8.0	Cloudy
8	1/11/2017	40.0	12.0	Sunny

In [121]:
```
import pandas as pd
df = pd.read_csv("7-5_weather_data.csv")
type(df.day[0])
```

Out[121]: str

In [123]:
```
# make day column into date
import pandas as pd
df = pd.read_csv("7-5_weather_data.csv", parse_dates=["day"])
df
```

Out[123]:

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-04	NaN	7.0	Sunny
2	2017-01-05	28.0	NaN	Snow
3	2017-01-06	NaN	7.0	NaN
4	2017-01-07	32.0	NaN	Rain
5	2017-01-08	31.0	2.0	Sunny
6	2017-01-09	NaN	NaN	NaN
7	2017-01-10	34.0	8.0	Cloudy
8	2017-01-11	40.0	12.0	Sunny

In [124]:
```
df = pd.read_csv("7-5_weather_data.csv", parse_dates=["day"])
type(df.day[0])
#note the change in output
```

Out[124]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

In [128]:
```
#make day index set
import pandas as pd
df = pd.read_csv("7-5_weather_data.csv", parse_dates=["day"])
df.set_index("day", inplace=True)
#inplace to modify original dataframe
df
```

Out[128]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	NaN	7.0	Sunny
2017-01-05	28.0	NaN	Snow
2017-01-06	NaN	7.0	NaN
2017-01-07	32.0	NaN	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	NaN	NaN	NaN
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [129]:
```
new_df = df.fillna(0) #for making a best guess on value
new_df
```

Out[129]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	0.0	7.0	Sunny
2017-01-05	28.0	0.0	Snow
2017-01-06	0.0	7.0	0
2017-01-07	32.0	0.0	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	0.0	0.0	0
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [130]:
```
#specifying fillna value by column
new_df = df.fillna({
    "temperature" : 0,
    "windspeed" : 0,
    "event" : "no event"
})
new_df
```

Out[130]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	0.0	7.0	Sunny
2017-01-05	28.0	0.0	Snow
2017-01-06	0.0	7.0	no event
2017-01-07	32.0	0.0	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	0.0	0.0	no event
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [131]:
```
#getting a better estimate, carrying forward value from previous day
new_df = df.fillna(method="ffill")
# ffill= forward fill
new_df
```

Out[131]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	32.0	7.0	Sunny
2017-01-05	28.0	7.0	Snow
2017-01-06	28.0	7.0	Snow
2017-01-07	32.0	7.0	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	31.0	2.0	Sunny
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [132]:
```
new_df = df.fillna(method="bfill")
# bfill= backward fill
new_df
```

Out[132]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	28.0	7.0	Sunny
2017-01-05	28.0	7.0	Snow
2017-01-06	32.0	7.0	Rain
2017-01-07	32.0	2.0	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	34.0	8.0	Cloudy
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [133]:
```
new_df = df.fillna(method="bfill", axis="columns")
#This copies value from value to the right, goes horizontally
new_df
```

Out[133]:

	temperature	windspeed	event
day			
2017-01-01	32	6	Rain
2017-01-04	7	7	Sunny
2017-01-05	28	Snow	Snow
2017-01-06	7	7	NaN
2017-01-07	32	Rain	Rain
2017-01-08	31	2	Sunny
2017-01-09	NaN	NaN	NaN
2017-01-10	34	8	Cloudy
2017-01-11	40	12	Sunny

In [135]:
```
new_df = df.fillna(method="ffill", limit=1)
# This stops a value from copying down more than once, wasn't happening
 in our set
new_df
```

Out[135]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	32.0	7.0	Sunny
2017-01-05	28.0	7.0	Snow
2017-01-06	28.0	7.0	Snow
2017-01-07	32.0	7.0	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	31.0	2.0	Sunny
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [136]:
```
#getting an even better guess of missing value
#interpolate, fill missing values with mean of value directly before and
 after
new_df = df.interpolate()
new_df
#linear interpolation, default of interpolate is linear
```

Out[136]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	30.0	7.0	Sunny
2017-01-05	28.0	7.0	Snow
2017-01-06	30.0	7.0	NaN
2017-01-07	32.0	4.5	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	32.5	5.0	NaN
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [138]:
```
#now considering jump in missing days
#IMPORTANT FEATURE
new_df = df.interpolate(method="time")
new_df
```

Out[138]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	29.0	7.0	Sunny
2017-01-05	28.0	7.0	Snow
2017-01-06	30.0	7.0	NaN
2017-01-07	32.0	4.5	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	32.5	5.0	NaN
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [140]:
```
#dropping all rows with missing values
new_df = df.dropna()
new_df
```

Out[140]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [141]:
```
#dropping rows that have all three values missing
new_df = df.dropna(how = "all")
new_df
#note 1/9 is gone
```

Out[141]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	NaN	7.0	Sunny
2017-01-05	28.0	NaN	Snow
2017-01-06	NaN	7.0	NaN
2017-01-07	32.0	NaN	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [142]:
```
#"If I have at least 1 filled value, keep the row", alternative to how=
 "all"
new_df = df.dropna(thresh = 1)
new_df
```

Out[142]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	NaN	7.0	Sunny
2017-01-05	28.0	NaN	Snow
2017-01-06	NaN	7.0	NaN
2017-01-07	32.0	NaN	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [143]:
```
new_df = df.dropna(thresh = 2)
new_df
#now 1/6 is dropped too. Requires 2 valid values
```

Out[143]:

	temperature	windspeed	event
day			
2017-01-01	32.0	6.0	Rain
2017-01-04	NaN	7.0	Sunny
2017-01-05	28.0	NaN	Snow
2017-01-07	32.0	NaN	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

In [145]:
```
#how to add missing dates
dt = pd.date_range("01-01-2017", "01-11-2017")
#set date range
idx=pd.DatetimeIndex(dt)
#create a date tme index
df = df.reindex(idx)
#reindex with the new date time index
df
#then ypu'd use a fillna value to fill missing dates with an estimate
```

Out[145]:

	temperature	windspeed	event
2017-01-01	32.0	6.0	Rain
2017-01-02	NaN	NaN	NaN
2017-01-03	NaN	NaN	NaN
2017-01-04	NaN	7.0	Sunny
2017-01-05	28.0	NaN	Snow
2017-01-06	NaN	7.0	NaN
2017-01-07	32.0	NaN	Rain
2017-01-08	31.0	2.0	Sunny
2017-01-09	NaN	NaN	NaN
2017-01-10	34.0	8.0	Cloudy
2017-01-11	40.0	12.0	Sunny

7-6: Handle Missing Data: replace function

In [155]:
```
import pandas as pd
import numpy as np #What is this? He doesn't say
df = pd.read_csv("7-6_weather_data.csv")
df
```

Out[155]:

	day	temperature	windspeed	event
0	1/1/17	32	6	Rain
1	1/2/17	-99999	7	Sunny
2	1/3/17	28	-99999	Snow
3	1/4/17	-99999	7	0
4	1/5/17	32	-99999	Rain
5	1/6/17	31	2	Sunny
6	1/6/17	34	5	0

In [156]:
```
new_df = df.replace(-99999, np.NaN) #why np.? He doens't explain
new_df
```

Out[156]:

	day	temperature	windspeed	event
0	1/1/17	32.0	6.0	Rain
1	1/2/17	NaN	7.0	Sunny
2	1/3/17	28.0	NaN	Snow
3	1/4/17	NaN	7.0	0
4	1/5/17	32.0	NaN	Rain
5	1/6/17	31.0	2.0	Sunny
6	1/6/17	34.0	5.0	0

In [158]:
```
new_df = df.replace([-99999, -88888, 0], np.NaN)
#make a list if you want to replace multiple different values
new_df
```

Out[158]:

	day	temperature	windspeed	event
0	1/1/17	32.0	6.0	Rain
1	1/2/17	NaN	7.0	Sunny
2	1/3/17	28.0	NaN	Snow
3	1/4/17	NaN	7.0	0
4	1/5/17	32.0	NaN	Rain
5	1/6/17	31.0	2.0	Sunny
6	1/6/17	34.0	5.0	0

In [162]:
```
new_df = df.replace({
    "temperature" : -99999,
    "windspeed" : -99999,
    "event" : "0"
    },np.NaN)

new_df
```

Out[162]:

	day	temperature	windspeed	event
0	1/1/17	32.0	6.0	Rain
1	1/2/17	NaN	7.0	Sunny
2	1/3/17	28.0	NaN	Snow
3	1/4/17	NaN	7.0	NaN
4	1/5/17	32.0	NaN	Rain
5	1/6/17	31.0	2.0	Sunny
6	1/6/17	34.0	5.0	NaN

In [166]:
```
#create a mapping of what you want to replace.
import pandas as pd
import numpy as np
df = pd.read_csv("7-6_weather_data.csv")
new_df = df.replace({
    -99999: np.NaN,
    "no event" : "Sunny"
    })

new_df
```

Out[166]:

	day	temperature	windspeed	event
0	1/1/17	32.0	6.0	Rain
1	1/2/17	NaN	7.0	Sunny
2	1/3/17	28.0	NaN	Snow
3	1/4/17	NaN	7.0	Sunny
4	1/5/17	32.0	NaN	Rain
5	1/6/17	31.0	2.0	Sunny
6	1/6/17	34.0	5.0	Sunny

In [167]:
```
import pandas as pd
import numpy as np
df = pd.read_csv("7-6_weather_data.csv")
df # changed dataframe to have units, like he did
```

Out[167]:

	day	temperature	windspeed	event
0	1/1/17	32 F	6 mph	Rain
1	1/2/17	-99999	7 mph	Sunny
2	1/3/17	28	-99999	Snow
3	1/4/17	-99999	7	no event
4	1/5/17	32 C	-99999	Rain
5	1/6/17	31	2	Sunny
6	1/6/17	34	5	no event

In [168]:
```
#regular expressions: used to detect patterns
new_df = df.replace("[A-Za-z]","",regex=True)
#brackets mean any letter is replaced with the blank that we have commae
d wth it
#regex=True must be present for it to work
new_df
#Note that the output removed the entire event column values
```

Out[168]:

	day	temperature	windspeed	event
0	1/1/17	32	6	
1	1/2/17	-99999	7	
2	1/3/17	28	-99999	
3	1/4/17	-99999	7	
4	1/5/17	32	-99999	
5	1/6/17	31	2	
6	1/6/17	34	5	

In [169]:
```
#how to replace by column
new_df = df.replace({
    "temperature" : "[A-Za-z]",
    "windspeed" : "[A-Za-z]",
    },"",regex=True)
new_df
```

Out[169]:

	day	temperature	windspeed	event
0	1/1/17	32	6	Rain
1	1/2/17	-99999	7	Sunny
2	1/3/17	28	-99999	Snow
3	1/4/17	-99999	7	no event
4	1/5/17	32	-99999	Rain
5	1/6/17	31	2	Sunny
6	1/6/17	34	5	no event

In [170]:
```
df =pd.DataFrame({
    "score" : ["exceptional", "average", "good", "poor", "average", "exc
eptional"],
    "student" : ["rob", "maya", "parthiv", "tom", "julian", "erica"]
})
df
```

Out[170]:

	score	student
0	exceptional	rob
1	average	maya
2	good	parthiv
3	poor	tom
4	average	julian
5	exceptional	erica

In [173]:
```
new_df = df.replace(["poor", "average", "good", "exceptional"], [1,2,3,4
])
new_df
```

Out[173]:

	score	student
0	4	rob
1	2	maya
2	3	parthiv
3	1	tom
4	2	julian
5	4	erica

7-7: Group By (Split Apply Combine)

In [176]:
```
import pandas as pd
df = pd.read_csv("7-7_weather_data.csv")
df
```

Out[176]:

	day	city	temperature	windspeed	event
0	1/1/2017	new york	32	6	Rain
1	1/2/2017	new york	36	7	Sunny
2	1/3/2017	new york	28	12	Snow
3	1/4/2017	new york	33	7	Sunny
4	1/1/2017	mumbai	90	5	Sunny
5	1/2/2017	mumbai	85	12	Fog
6	1/3/2017	mumbai	87	15	Fog
7	1/4/2017	mumbai	92	5	Rain
8	1/1/2017	paris	45	20	Sunny
9	1/2/2017	paris	50	13	Cloudy
10	1/3/2017	paris	54	8	Cloudy
11	1/4/2017	paris	42	10	Cloudy

In [178]:
```
g = df.groupby("city") #mention column name that you want to group by
g
```

Out[178]: <pandas.core.groupby.groupby.DataFrameGroupBy object at 0x113647be0>

In [179]:
```
#need to do the above step for this to work
for city, city_df in g:
    print(city)
    print(city_df)
```

```
mumbai
        day    city  temperature  windspeed  event
4  1/1/2017  mumbai           90          5  Sunny
5  1/2/2017  mumbai           85         12    Fog
6  1/3/2017  mumbai           87         15    Fog
7  1/4/2017  mumbai           92          5   Rain
new york
        day      city  temperature  windspeed  event
0  1/1/2017  new york           32          6   Rain
1  1/2/2017  new york           36          7  Sunny
2  1/3/2017  new york           28         12   Snow
3  1/4/2017  new york           33          7  Sunny
paris
         day   city  temperature  windspeed   event
8   1/1/2017  paris           45         20   Sunny
9   1/2/2017  paris           50         13  Cloudy
10  1/3/2017  paris           54          8  Cloudy
11  1/4/2017  paris           42         10  Cloudy
```

In [180]:
```
#way to see one data frame by city
g.get_group("mumbai")
```

Out[180]:

	day	city	temperature	windspeed	event
4	1/1/2017	mumbai	90	5	Sunny
5	1/2/2017	mumbai	85	12	Fog
6	1/3/2017	mumbai	87	15	Fog
7	1/4/2017	mumbai	92	5	Rain

In [182]:
```
# max for each city for each variable
# these processes together called Split Apply Combine
g.max()
```

Out[182]:

	day	temperature	windspeed	event
city				
mumbai	1/4/2017	92	15	Sunny
new york	1/4/2017	36	12	Sunny
paris	1/4/2017	54	20	Sunny

In [183]:
```
g.mean()
```

Out[183]:

	temperature	windspeed
city		
mumbai	88.50	9.25
new york	32.25	8.00
paris	47.75	12.75

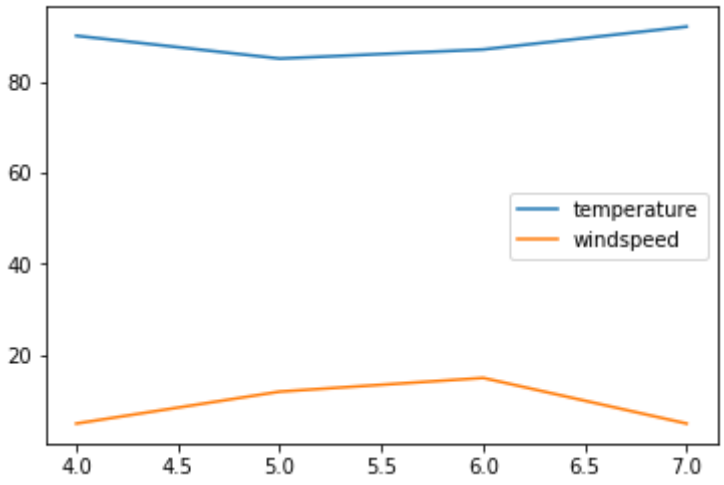
In [184]:
```
# all the info in one shot
g.describe()
```

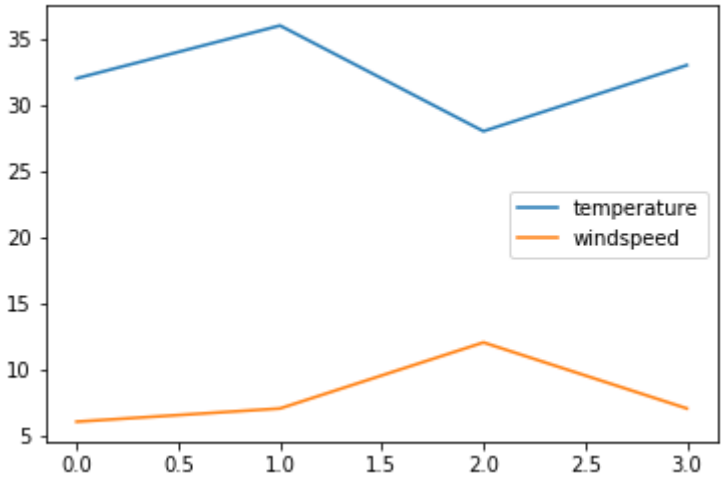
Out[184]:

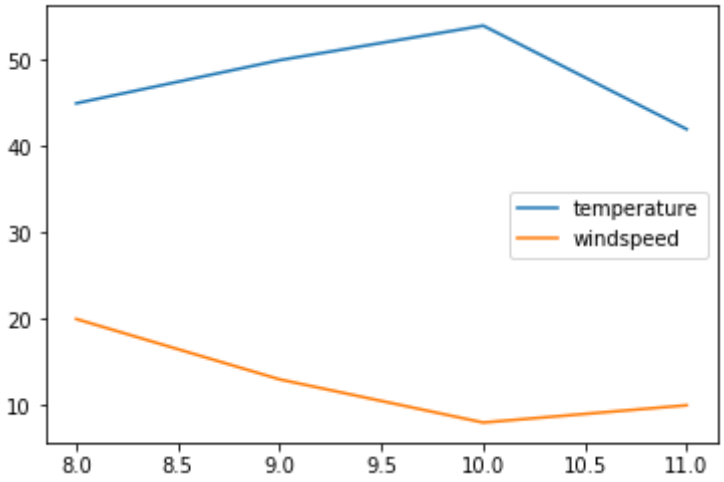
	temperature								windspeed			
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**std**	**m**
city												
mumbai	4.0	88.50	3.109126	85.0	86.50	88.5	90.50	92.0	4.0	9.25	5.057997	5.
new york	4.0	32.25	3.304038	28.0	31.00	32.5	33.75	36.0	4.0	8.00	2.708013	6.
paris	4.0	47.75	5.315073	42.0	44.25	47.5	51.00	54.0	4.0	12.75	5.251984	8.

In [185]:
```
%matplotlib inline
g.plot()
```

Out[185]:
```
city
mumbai      AxesSubplot(0.125,0.125;0.775x0.755)
new york    AxesSubplot(0.125,0.125;0.775x0.755)
paris       AxesSubplot(0.125,0.125;0.775x0.755)
dtype: object
```







7-8: How do I work with dates and times in pandas?

Why have we changed narrators? Miss the other guy

In [186]:
```
import pandas as pd
ufo =pd.read_csv("7-8_uforeports.csv")
ufo.head()
```

Out[186]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00

In [187]:
```
ufo.dtypes
#tells us they are stored as strings
```

Out[187]:
```
City               object
Colors Reported    object
Shape Reported     object
State              object
Time               object
dtype: object
```

In [192]:
```
ufo.Time.str.slice(-5, -3).astype(int).head()
#I don't understand what he has sliced, and how -5, -3 got it
#"5 characters from the end", the end of what?
#gives us the hour
```

Out[192]:
```
0    22
1    20
2    14
3    13
4    19
Name: Time, dtype: int64
```

In [194]:
```
#convert time column to panda's date time formate
ufo["Time"] = pd.to_datetime(ufo.Time)
```

In [195]:
```
ufo.head()
#data is the same, formatting changed
```

Out[195]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	1930-06-01 22:00:00
1	Willingboro	NaN	OTHER	NJ	1930-06-30 20:00:00
2	Holyoke	NaN	OVAL	CO	1931-02-15 14:00:00
3	Abilene	NaN	DISK	KS	1931-06-01 13:00:00
4	New York Worlds Fair	NaN	LIGHT	NY	1933-04-18 19:00:00

In [196]:
```
ufo.dtypes
#but no longer string, that's what actually changed
```

Out[196]:
```
City                         object
Colors Reported              object
Shape Reported               object
State                        object
Time                 datetime64[ns]
dtype: object
```

In [198]:
```
ufo.Time.dt.hour.head()
#dt. calls that we are using a datetime series property
```

Out[198]:
```
0    22
1    20
2    14
3    13
4    19
Name: Time, dtype: int64
```

In [199]:
```
ufo.Time.dt.weekday_name.head()
```

Out[199]:
```
0     Sunday
1     Monday
2     Sunday
3     Monday
4    Tuesday
Name: Time, dtype: object
```

In [201]:
```
ufo.Time.dt.dayofyear.head()
```

Out[201]:
```
0    152
1    181
2     46
3    152
4    108
Name: Time, dtype: int64
```

In [203]:
```
ts = pd.to_datetime("1/1/1999")
#you can use timestamps as part of comparisons
```

In [205]:
```
ufo.loc[ufo.Time >= ts, :].head()
#colon means show me all columns
# shows all times that happened after 1/1/1999
```

Out[205]:

	City	Colors Reported	Shape Reported	State	Time
12832	Loma Rica	NaN	LIGHT	CA	1999-01-01 02:30:00
12833	Bauxite	NaN	NaN	AR	1999-01-01 03:00:00
12834	Florence	NaN	CYLINDER	SC	1999-01-01 14:00:00
12835	Lake Henshaw	NaN	CIGAR	CA	1999-01-01 15:00:00
12836	Wilmington Island	NaN	LIGHT	GA	1999-01-01 17:15:00

In [206]:
```
#most recent timestamp in the series
ufo.Time.max()
```

Out[206]:
```
Timestamp('2000-12-31 23:59:00')
```

In [207]:
```
# time delta object
ufo.Time.max() - ufo.Time.min()
```

Out[207]:
```
Timedelta('25781 days 01:59:00')
```

In [209]:
```
#choose unit
(ufo.Time.max() - ufo.Time.min()).days
```

Out[209]:
```
25781
```

In [211]:
```
%matplotlib inline
ufo["Year"] = ufo.Time.dt.year
```

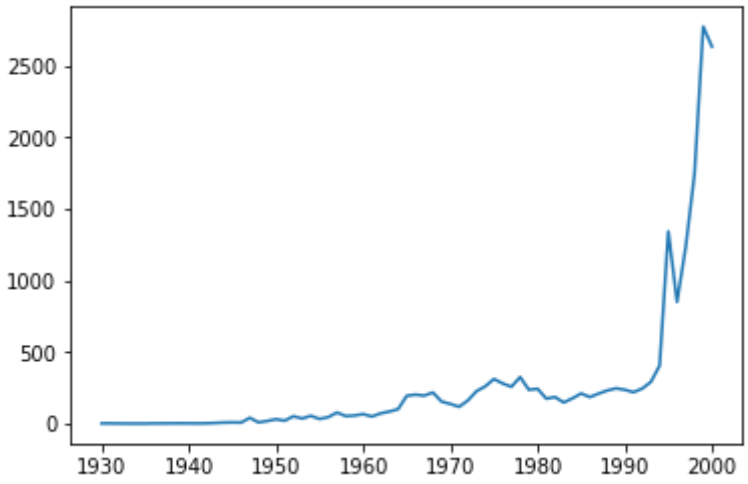
In [210]:
```
ufo.head()
```

Out[210]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	1930-06-01 22:00:00
1	Willingboro	NaN	OTHER	NJ	1930-06-30 20:00:00
2	Holyoke	NaN	OVAL	CO	1931-02-15 14:00:00
3	Abilene	NaN	DISK	KS	1931-06-01 13:00:00
4	New York Worlds Fair	NaN	LIGHT	NY	1933-04-18 19:00:00

In [216]:
```
ufo.Year.value_counts().sort_index().plot()
```

Out[216]: <matplotlib.axes._subplots.AxesSubplot at 0x11efc83c8>



7-9: Pivot Table

In [218]:
```
import pandas as pd
df = pd.read_csv("7-9_weather.csv")
df
```

Out[218]:

	date	city	temperature	humidity
0	5/1/2017	new york	65	56
1	5/2/2017	new york	66	58
2	5/3/2017	new york	68	60
3	5/1/2017	mumbai	75	80
4	5/2/2017	mumbai	78	83
5	5/3/2017	mumbai	82	85
6	5/1/2017	beijing	80	26
7	5/2/2017	beijing	77	30
8	5/3/2017	beijing	79	35

In [219]:
```
df.pivot(index="date", columns="city")
```

Out[219]:

	temperature			humidity		
city	**beijing**	**mumbai**	**new york**	**beijing**	**mumbai**	**new york**
date						
5/1/2017	80	75	65	26	80	56
5/2/2017	77	78	66	30	83	58
5/3/2017	79	82	68	35	85	60

In [220]:
```
df.pivot(index="date", columns="city", values="humidity")
```

Out[220]:

city	**beijing**	**mumbai**	**new york**
date			
5/1/2017	26	80	56
5/2/2017	30	83	58
5/3/2017	35	85	60

In [221]:
```
df= pd.read_csv("7-9_weather2.csv")
df
#note how there are now multiple temperatures for a day
```

Out[221]:

	date	**city**	**temperature**	**humidity**
0	5/1/2017	new york	65	56
1	5/1/2017	new york	61	54
2	5/2/2017	new york	70	60
3	5/2/2017	new york	72	62
4	5/1/2017	mumbai	75	80
5	5/1/2017	mumbai	78	83
6	5/2/2017	mumbai	82	85
7	5/2/2017	mumbai	80	26

In [222]:
```
#this automatically takes the average
df.pivot_table(index="city", columns="date")
```

Out[222]:

	humidity		temperature	
date	**5/1/2017**	**5/2/2017**	**5/1/2017**	**5/2/2017**
city				
mumbai	81.5	55.5	76.5	81.0
new york	55.0	61.0	63.0	71.0

In [223]:
```
#aggfunc can be set, the default is mean and does not need to be specifi
ed
df.pivot_table(index="city", columns="date", aggfunc="sum")
```

Out[223]:

	humidity		temperature	
date	**5/1/2017**	**5/2/2017**	**5/1/2017**	**5/2/2017**
city				
mumbai	163	111	153	162
new york	110	122	126	142

In [224]:
```
#shows number of instances
df.pivot_table(index="city", columns="date", aggfunc="count")
```

Out[224]:

	humidity		temperature	
date	**5/1/2017**	**5/2/2017**	**5/1/2017**	**5/2/2017**
city				
mumbai	2	2	2	2
new york	2	2	2	2

In [225]:
```
df.pivot_table(index="city", columns="date", margins=True)
#shows average in each direction of the table
```

Out[225]:

	humidity			temperature		
date	**5/1/2017**	**5/2/2017**	**All**	**5/1/2017**	**5/2/2017**	**All**
city						
mumbai	81.50	55.50	68.50	76.50	81.0	78.750
new york	55.00	61.00	58.00	63.00	71.0	67.000
All	68.25	58.25	63.25	69.75	76.0	72.875

In [226]:
```
df = pd.read_csv("7-9_weather3.csv")
df
```

Out[226]:

	date	city	temperature	humidity
0	5/1/2017	new york	65	56
1	5/2/2017	new york	61	54
2	5/3/2017	new york	70	60
3	12/1/2017	new york	30	50
4	12/2/2017	new york	28	52
5	12/3/2017	new york	25	51

In [227]:
```
df["date"]=pd.to_datetime(df["date"])
df.pivot_table(index=pd.Grouper(freq="M", key="date"), columns ="city")
```

Out[227]:

	humidity	temperature
city	**new york**	**new york**
date		
2017-05-31	56.666667	65.333333
2017-12-31	51.000000	27.666667

Part B!

7-1:

Using the skills from the videos, choose a dataset you from the Data Sets links on the course Module 7 page or any other source that interests you. The links are to sources for free datasets or at the bottom of that page are CSV, Excel, and TXT data sets that you can use for this project.

After reviewing that data, define at least two or three questions that a researcher (you) might ask about your raw data that pandas could answer. Example: Using the NOAH Weather Seattle Area dataset, what was the average precipitation for each reporting station? How many days did it rain at each station?

I have selected the movie comments one. Part of why I chose this is because for a research project I did a vader sentimental analysis of comments left by users at a library. I'm interested in the idea of trying to use code to handle large ideas. I think it's sometimes a little harder to wrap your head around datasets like this.

Question One:

How many movies has each reviewer reviewed? What is the average number of reviews written by reviewers?

In [388]:
```
import pandas as pd
df = pd.read_csv("reviews.csv", usecols=["critic", "publication"])
df.columns = ["critic", "number of reviews"]

df.pivot_table(index="critic", aggfunc="count")
print(df.pivot_table(index="critic", aggfunc="count").mean())
%matplotlib inline
df.pivot_table(index="critic", aggfunc="count").plot()
df.pivot_table(index="critic", aggfunc="count")
```

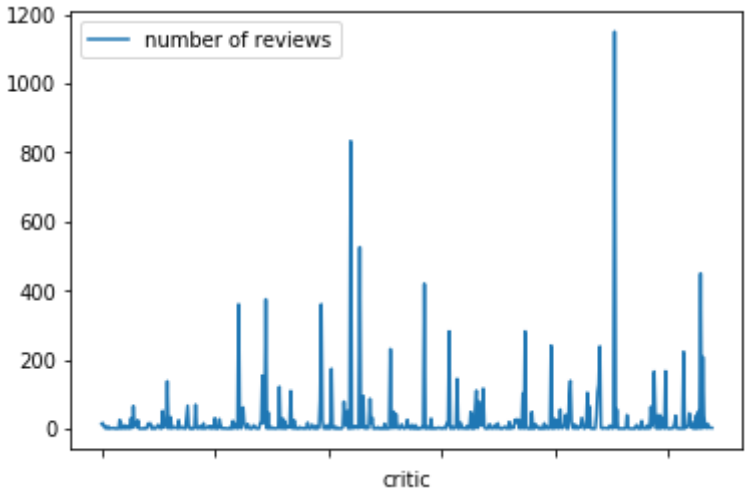
```
number of reviews    23.587037
dtype: float64
```

Out[388]:

	number of reviews
critic	
A.D. Murphy	14
A.H. Weiler	17
A.O. Scott	10
Aaron Hillis	2
Abel Green	5
Achy Obejas	1
Adam Graham	7
Adam Markovitz	1
Akiva Gottlieb	2
Al Brumley	2
Alan Scherstuhl	2
Alfred Rushford Greason	3
Alissa Simon	1
Allen Barra	1
Allison Benedikt	1
Amos Barshad	1
Amy Biancolli	26
Amy Dawes	2
Amy E. Schwartz	1
Amy Simmons	2
Amy Taubin	11
Ana Marie Cox	1
Andre Sennwald	5
Andrea C. Basora	2
Andrea Gronvall	9
Andrew Geller	1
Andrew O'Hehir	31
Andrew Ross	3
Andrew Sarris	66
Anita Gates	14
...	...

	number of reviews
critic	
Thomas O'Connor	1
Tim Grierson	1
Tim Purtell	1
Todd Gilchrist	1
Todd McCarthy	223
Tom Charity	1
Tom Huddleston	5
Tom Huddlestone	7
Tom Keogh	9
Tom Long	45
Tom Maurstad	5
Tom Milne	20
Tom Russo	2
Tom Sime	2
Tony Wong	1
Trevor Johnston	38
Trevor Lewis	1
Ty Burr	50
V.A. Musetto	2
Variety Staff	450
Vic Vogler	1
Vincent Canby	208
Wally Hammond	38
Walter Goodman	7
Walter V. Addiego	2
Wesley Morris	15
Whitney Willaims	4
Whittaker Chambers	3
William Brogdon	4
William Goss	3

540 rows × 1 columns



Question Two:

How many times did each publisher publish? Min? Max? Standard Deviation? Mean?

I wanted to do a question more dissimilar from the first one, but couldn't really think of something else. I considered looking at the dates, but they were incredibly messy in this dataset. I also initially wanted to see the occurences of fresh vs. rotten for each movie title. I could figure out how to group by movie but could not figure out how to look at occurences within a column applied to each grouping. I eventually had to move onto another idea.

In [387]:
```
import pandas as pd
df = pd.read_csv("reviews.csv", usecols=["publication", "title"])
df.set_index('publication')
df.columns = ["publication", "number of publications"]

df.pivot_table(index="publication", aggfunc="count")
print("Max:")
print(df.pivot_table(index="publication", aggfunc="count").max())
print("Min:")
print(df.pivot_table(index="publication", aggfunc="count").min())
print("Mean:")
print(df.pivot_table(index="publication", aggfunc="count").mean())
print("Standard Deviation:")
print(df.pivot_table(index="publication", aggfunc="count").std())
%matplotlib inline
df.pivot_table(index="publication", aggfunc="count").plot()
df.pivot_table(index="publication", aggfunc="count")
```

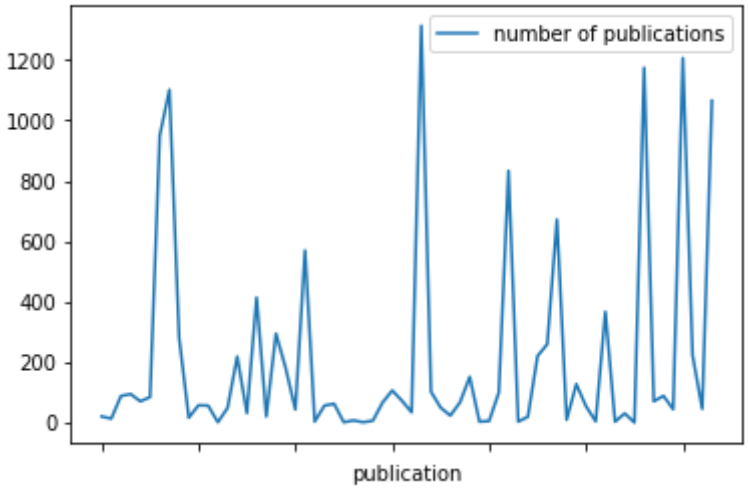
```
Max:
number of publications    1313
dtype: int64
Min:
number of publications    1
dtype: int64
Mean:
number of publications    210.03125
dtype: float64
Standard Deviation:
number of publications    341.163135
dtype: float64
```

Out[387]:

	number of publications
publication	
Arizona Republic	21
Associated Press	14
At the Movies	89
Atlanta Journal-Constitution	95
Boston Globe	71
CNN.com	85
Chicago Reader	949
Chicago Sun-Times	1101
Chicago Tribune	282
Christian Science Monitor	17
Dallas Morning News	58
Denver Post	57
Denver Rocky Mountain News	2
Detroit Free Press	49
Detroit News	219
Ebert & Roeper	32
Entertainment Weekly	414
Film.com	21
Globe and Mail	295
Houston Chronicle	181
L.A. Weekly	44
Los Angeles Times	570
MSN Movies	4
Miami Herald	57
Minneapolis Star Tribune	63
MovieTime, ABC Radio National	2
Mr. Showbiz	8
NPR	2
NPR.org	7
New York Daily News	67
...	...

	number of publications
publication	
New Yorker	103
Newark Star-Ledger	50
Newsday	24
Newsweek	68
Orlando Sentinel	152
Passionate Moviegoer	4
Philadelphia Daily News	6
Philadelphia Inquirer	100
ReelViews	833
Richard Roeper.com	4
Rolling Stone	20
Sacramento Bee	221
Salon.com	260
San Francisco Chronicle	672
San Jose Mercury News	10
Seattle Times	128
Slate	55
St. Louis Post-Dispatch	5
TIME Magazine	367
The Atlantic	4
The New Republic	31
The Wrap	1
Time Out	1174
Time Out New York	71
Toronto Star	89
USA Today	45
Variety	1206
Village Voice	224
Wall Street Journal	46
Washington Post	1065

64 rows × 1 columns


number of publications
1200
1000
800
600
400
200
0
publication