Software Engineering for Data Scientists

Manipulating Data with Python

CSE 599 B1

Today's Objectives

1. Opening & Navigating the IPython Notebook

2. Simple Math in the IPython Notebook

3. Loading data with pandas

4. Cleaning and Manipulating data with pandas

5. Visualizing data with pandas

1. Opening and Navigating the IPython Notebook

We will start today with the interactive environment that we will be using often through the course: the IPython/Jupyter Notebook (http://ipython.org).

We will walk through the following steps together:

1. Download miniconda () (be sure to get Version 3.5) and install it on your system (hopefully you have done this before coming to class)
2. Use the `conda` command-line tool to update your package listing and install the IPython notebook:

   Update `conda`'s listing of packages for your system:

   ```
   $ conda update conda
   ```

   Install IPython notebook and all its requirements

   ```
   $ conda install ipython-notebook
   ```

3. Navigate to the directory containing the course material. For example:

   ```
   $ cd ~/courses/CSE599/
   ```

   You should see a number of files in the directory, including these:

   ```
   $ ls
   ...
   Breakout-Simple-Math.ipynb
   CSE599_Lecture_2.ipynb
   ...
   ```

4. Type `ipython notebook` in the terminal to start the notebook

   ```
   $ ipython notebook
   ```

   If everything has worked correctly, it should automatically launch your default browser
5. Click on `CSE599_Lecture_2.ipynb` to open the notebook containing the content for this lecture.

With that, you're set up to use the IPython notebook!

2. Simple Math in the IPython Notebook

Now that we have the IPython notebook up and running, we're going to do a short breakout exploring some of the mathematical functionality that Python offers.

Please open Breakout-Simple-Math.ipynb (Breakout-Simple-Math.ipynb), find a partner, and make your way through that notebook, typing and executing code along the way.

3. Loading data with pandas

With this simple Python computation experience under our belt, we can now move to doing some more interesting analysis.

Python's Data Science Ecosystem

In addition to Python's built-in modules like the `math` module we explored above, there are also many often-used third-party modules that are core tools for doing data science with Python. Some of the most important ones are:

numpy (http://numpy.org/): Numerical Python

Numpy is short for "Numerical Python", and contains tools for efficient manipulation of arrays of data. If you have used other computational tools like IDL or MatLab, Numpy should feel very familiar.

`scipy` (http://scipy.org/): Scientific Python

Scipy is short for "Scientific Python", and contains a wide range of functionality for accomplishing common scientific tasks, such as optimization/minimization, numerical integration, interpolation, and much more. We will not look closely at Scipy today, but we will use its functionality later in the course.

pandas (http://pandas.pydata.org/): Labeled Data Manipulation in Python

Pandas is short for "Panel Data", and contains tools for doing more advanced manipulation of labeled data in Python, in particular with a columnar data structure called a *Data Frame*. If you've used the R (http://rstats.org) statistical language (and in particular the so-called "Hadley Stack"), much of the functionality in Pandas should feel very familiar.

`matplotlib` (http://matplotlib.org): Visualization in Python

Matplotlib started out as a Matlab plotting clone in Python, and has grown from there in the 15 years since its creation. It is the most popular data visualization tool currently in the Python data world (though other recent packages are starting to encroach on its monopoly).

Installing Pandas & friends

Because the above packages are not included in Python itself, you need to install them separately. While it is possible to install these from source (compiling the C and/or Fortran code that does the heavy lifting under the hood) it is much easier to use a package manager like `conda`. All it takes is to run

```
$ conda install numpy scipy pandas matplotlib
```

and (so long as your conda setup is working) the packages will be downloaded and installed on your system.

Loading Data with Pandas

In [1]:
```
import numpy
numpy.__path__
```

Out[1]: `['/home/ubuntu/miniconda2/envs/python3/lib/python2.7/site-packages/numpy']`

In [2]:
```
import pandas
```

Because we'll use it so much, we often import under a shortened name using the `import ... as ...` pattern:

In [3]:
```
import pandas as pd
```

Now we can use the `read_csv` command to read the comma-separated-value data:

In [4]:
```
data = pd.read_csv('2015_trip_data.csv')
```

Note: strings in Python can be defined either with double quotes or single quotes

Viewing Pandas Dataframes

The `head()` and `tail()` methods show us the first and last rows of the data

In [5]:
```
data.head()
```

Out[5]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_name	fro
0	431	10/13/2014 10:31	10/13/2014 10:48	SEA00298	985.935	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
1	432	10/13/2014 10:32	10/13/2014 10:48	SEA00195	926.375	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
2	433	10/13/2014 10:33	10/13/2014 10:48	SEA00486	883.831	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
3	434	10/13/2014 10:34	10/13/2014 10:48	SEA00333	865.937	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
4	435	10/13/2014 10:34	10/13/2014 10:49	SEA00202	923.923	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI

In [6]:
```
data.tail()
```

Out[6]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_nam
142841	156796	10/12/2015 20:41	10/12/2015 20:47	SEA00358	377.183	E Pine St & 16th Ave	Summit Ave & E Denny Way
142842	156797	10/12/2015 20:43	10/12/2015 20:48	SEA00399	303.330	Bellevue Ave & E Pine St	Summit Ave E & E Republican St
142843	156798	10/12/2015 21:03	10/12/2015 21:06	SEA00204	165.597	Harvard Ave & E Pine St	E Harrison St & Broadway Ave E
142844	156799	10/12/2015 21:35	10/12/2015 21:41	SEA00073	388.576	Pine St & 9th Ave	3rd Ave & Broad St
142845	156800	10/12/2015 22:45	10/12/2015 22:51	SEA00033	391.885	NE 42nd St & University Way NE	Eastlake Ave E E Allison St

The `shape` attribute shows us the number of elements:

In [7]:
```
data.shape
```

Out[7]:
```
(142846, 12)
```

The `columns` attribute gives us the column names

In [8]:
```
data.columns
```

Out[8]:
```
Index([u'trip_id', u'starttime', u'stoptime', u'bikeid', u'tripduration',
       u'from_station_name', u'to_station_name', u'from_station_id',
       u'to_station_id', u'usertype', u'gender', u'birthyear'],
      dtype='object')
```

The `index` attribute gives us the index names

In [9]:
```
data.index
```

Out[9]:
```
RangeIndex(start=0, stop=142846, step=1)
```

The `dtypes` attribute gives the data types of each column:

In [10]:
```
data.dtypes
```

Out[10]:
```
trip_id                int64
starttime             object
stoptime              object
bikeid                object
tripduration         float64
from_station_name     object
to_station_name       object
from_station_id       object
to_station_id         object
usertype              object
gender                object
birthyear            float64
dtype: object
```

4. Manipulating data with pandas

Here we'll cover some key features of manipulating data with pandas

Access columns by name using square-bracket indexing:

In [11]:
```
data["usertype"]
```

```
Out[11]: 0                   Annual Member
         1                   Annual Member
         2                   Annual Member
         3                   Annual Member
         4                   Annual Member
         5                   Annual Member
         6                   Annual Member
         7                   Annual Member
         8                   Annual Member
         9                   Annual Member
         10                  Annual Member
         11                  Annual Member
         12                  Annual Member
         13                  Annual Member
         14                  Annual Member
         15                  Annual Member
         16                  Annual Member
         17                  Annual Member
         18                  Annual Member
         19                  Annual Member
         20                  Annual Member
         21                  Annual Member
         22                  Annual Member
         23                  Annual Member
         24                  Annual Member
         25                  Annual Member
         26                  Annual Member
         27                  Annual Member
         28                  Annual Member
         29                  Annual Member
                             ...
         142816              Annual Member
         142817              Annual Member
         142818              Annual Member
         142819              Annual Member
         142820    Short-Term Pass Holder
         142821    Short-Term Pass Holder
         142822    Short-Term Pass Holder
         142823    Short-Term Pass Holder
         142824              Annual Member
         142825              Annual Member
         142826              Annual Member
         142827              Annual Member
         142828              Annual Member
         142829              Annual Member
         142830              Annual Member
         142831              Annual Member
         142832              Annual Member
         142833    Short-Term Pass Holder
         142834              Annual Member
         142835              Annual Member
         142836              Annual Member
         142837              Annual Member
         142838              Annual Member
         142839              Annual Member
         142840              Annual Member
         142841              Annual Member
         142842              Annual Member
         142843              Annual Member
         142844    Short-Term Pass Holder
         142845              Annual Member
         Name: usertype, dtype: object
```

Mathematical operations on columns happen *element-wise*:

In [12]:
```
data['tripduration'] / 60
```

```
Out[12]: 0         16.432250
         1         15.439583
         2         14.730517
         3         14.432283
         4         15.398717
         5         13.480083
         6          9.945250
         7          9.868850
         8          9.772450
         9          9.793900
         10         9.414983
         11        10.335683
         12        10.568117
         13        10.238933
         14        10.024383
         15        10.313017
         16        10.284750
         17        10.000833
         18         8.328900
         19         9.588450
         20         9.530117
         21         9.396050
         22         7.294817
         23         8.052567
         24         7.989700
         25         8.892200
         26         8.027150
         27         7.679533
         28         7.652750
         29        11.340950
                     ...
         142816     8.671450
         142817    14.607983
         142818     4.938550
         142819    11.882017
         142820    21.947550
         142821    21.883067
         142822    21.240667
         142823    18.747133
         142824    22.390117
         142825    12.643100
         142826     4.613267
         142827     6.036600
         142828     4.168183
         142829     4.278083
         142830     6.128367
         142831     7.455983
         142832     2.109933
         142833     9.045133
         142834     4.308700
         142835    15.671783
         142836    10.276817
         142837    10.726967
         142838     9.055200
         142839     3.375017
         142840     8.962267
         142841     6.286383
         142842     5.055500
         142843     2.759950
         142844     6.476267
         142845     6.531417
         Name: tripduration, dtype: float64
```

Columns can be created (or overwritten) with the assignment operator. Let's create a *tripminutes* column with the number of minutes for each trip

In [13]:
```
data['tripminutes'] = data['tripduration'] / 60
```

In [14]:
```
data.head()
```

Out[14]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_name	fro
0	431	10/13/2014 10:31	10/13/2014 10:48	SEA00298	985.935	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
1	432	10/13/2014 10:32	10/13/2014 10:48	SEA00195	926.375	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
2	433	10/13/2014 10:33	10/13/2014 10:48	SEA00486	883.831	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
3	434	10/13/2014 10:34	10/13/2014 10:48	SEA00333	865.937	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI
4	435	10/13/2014 10:34	10/13/2014 10:49	SEA00202	923.923	2nd Ave & Spring St	Occidental Park / Occidental Ave S & S Washing...	CBI

Working with Times

One trick to know when working with columns of times is that Pandas `DateTimeIndex` provides a nice interface for working with columns of times:

In [15]:
```
times = pd.DatetimeIndex(data['starttime'])
```

With it, we can extract, the hour of the day, the day of the week, the month, and a wide range of other views of the time:

In [16]:
```
times
```

Out[16]:
```
DatetimeIndex(['2014-10-13 10:31:00', '2014-10-13 10:32:00',
               '2014-10-13 10:33:00', '2014-10-13 10:34:00',
               '2014-10-13 10:34:00', '2014-10-13 10:34:00',
               '2014-10-13 11:35:00', '2014-10-13 11:35:00',
               '2014-10-13 11:35:00', '2014-10-13 11:35:00',
               ...
               '2015-10-12 20:09:00', '2015-10-12 20:11:00',
               '2015-10-12 20:18:00', '2015-10-12 20:39:00',
               '2015-10-12 20:41:00', '2015-10-12 20:41:00',
               '2015-10-12 20:43:00', '2015-10-12 21:03:00',
               '2015-10-12 21:35:00', '2015-10-12 22:45:00'],
              dtype='datetime64[ns]', length=142846, freq=None)
```

In [17]:
```
times.dayofweek
```

Out[17]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int32)
```

In [18]:
```
times.month
```

Out[18]:
```
array([10, 10, 10, ..., 10, 10, 10], dtype=int32)
```

Note: math functionality can be applied to columns using the NumPy package: for example:

In [19]:
```
import numpy as np
np.exp(data['tripminutes'])
```

```
Out[19]: 0         1.369101e+07
         1         5.073712e+06
         2         2.496791e+06
         3         1.852939e+06
         4         4.870546e+06
         5         7.150325e+05
         6         2.085294e+04
         7         1.931911e+04
         8         1.754370e+04
         9         1.792407e+04
         10        1.227087e+04
         11        3.081273e+04
         12        3.887539e+04
         13        2.797127e+04
         14        2.257015e+04
         15        3.012217e+04
         16        2.928264e+04
         17        2.204483e+04
         18        4.141859e+03
         19        1.459523e+04
         20        1.376820e+04
         21        1.204073e+04
         22        1.472647e+03
         23        3.141849e+03
         24        2.950412e+03
         25        7.275007e+03
         26        3.063000e+03
         27        2.163610e+03
         28        2.106430e+03
         29        8.419998e+04
                       ...
         142816    5.833952e+03
         142817    2.208852e+06
         142818    1.395677e+02
         142819    1.446420e+05
         142820    3.401730e+09
         142821    3.189298e+09
         142822    1.677661e+09
         142823    1.386043e+08
         142824    5.295465e+09
         142825    3.096197e+05
         142826    1.008129e+02
         142827    4.184678e+02
         142828    6.459799e+01
         142829    7.210211e+01
         142830    4.586864e+02
         142831    1.730185e+03
         142832    8.247691e+00
         142833    8.477182e+03
         142834    7.434378e+01
         142835    6.399839e+06
         142836    2.905125e+04
         142837    4.556826e+04
         142838    8.562950e+03
         142839    2.922477e+01
         142840    7.803024e+03
         142841    5.372069e+02
         142842    1.568830e+02
         142843    1.579905e+01
         142844    6.495415e+02
         142845    6.863699e+02
         Name: tripminutes, dtype: float64
```

Simple Grouping of Data

The real power of Pandas comes in its tools for grouping and aggregating data. Here we'll look at *value counts* and the basics of *group-by* operations.

Value Counts

Pandas includes an array of useful functionality for manipulating and analyzing tabular data. We'll take a look at two of these here.

The `pandas.value_counts` returns statistics on the unique values within each column.

We can use it, for example, to break down rides by gender:

In [20]:
```
pd.value_counts(data['gender'])
```

Out[20]:
```
Male      67608
Female    18245
Other      1507
Name: gender, dtype: int64
```

Or to break down rides by age:

In [21]:
```
pd.value_counts(data['birthyear']).sort_index()
```

```
Out[21]: 1936.0       6
         1939.0      23
         1942.0       2
         1943.0      11
         1944.0       1
         1945.0     115
         1946.0      39
         1947.0     244
         1948.0      34
         1949.0     154
         1950.0     657
         1951.0     251
         1952.0     204
         1953.0     337
         1954.0     152
         1955.0     397
         1956.0     481
         1957.0     224
         1958.0     160
         1959.0     696
         1960.0     429
         1961.0     875
         1962.0    1769
         1963.0     828
         1964.0    1374
         1965.0    1510
         1966.0     761
         1967.0    1354
         1968.0    1085
         1969.0    1185
         1970.0    1056
         1971.0    1279
         1972.0    1921
         1973.0    1076
         1974.0    1497
         1975.0    1969
         1976.0    1577
         1977.0    2465
         1978.0    2063
         1979.0    1976
         1980.0    2236
         1981.0    4779
         1982.0    4629
         1983.0    3965
         1984.0    3815
         1985.0    5370
         1986.0    3492
         1987.0    9320
         1988.0    4188
         1989.0    2755
         1990.0    3605
         1991.0    2912
         1992.0    1798
         1993.0    1126
         1994.0     549
         1995.0     412
         1996.0     121
         1997.0      21
         1998.0      25
         1999.0       5
         Name: birthyear, dtype: int64
```

What else might we break down rides by?

In [22]:
```
pd.value_counts(times.dayofweek)
```

Out[22]:
```
3    21505
0    21266
4    21097
2    20748
1    20465
5    20358
6    17407
dtype: int64
```

We can sort by the index rather than the counts if we wish:

In [23]:
```
pd.value_counts(times.dayofweek, sort=False)
```

Out[23]:
```
0    21266
1    20465
2    20748
3    21505
4    21097
5    20358
6    17407
dtype: int64
```

In [24]:
```
pd.value_counts(times.month)
```

Out[24]:
```
7     18808
8     17046
6     15999
5     15548
9     13134
4     12898
10    11081
3      9980
11     7823
1      7368
2      7330
12     5831
dtype: int64
```

In [25]:
```
pd.value_counts(times.month, sort=False)
```

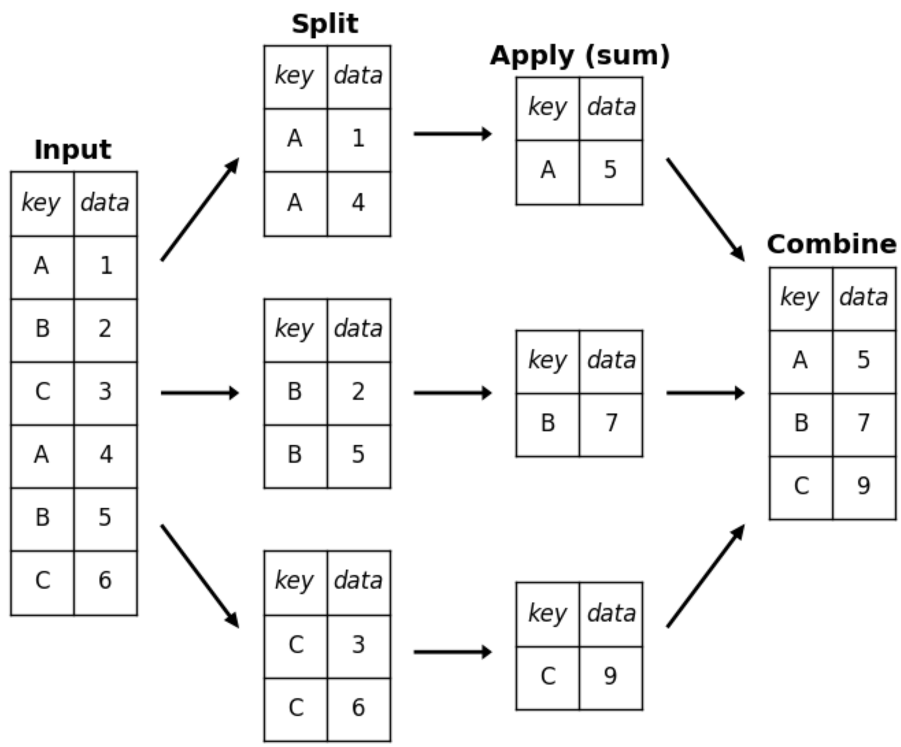
Out[25]:
```
1      7368
2      7330
3      9980
4     12898
5     15548
6     15999
7     18808
8     17046
9     13134
10    11081
11     7823
12     5831
dtype: int64
```

Group-by Operation

One of the killer features of the Pandas dataframe is the ability to do group-by operations. You can visualize the group-by like this (image borrowed from the Python Data Science Handbook (http://shop.oreilly.com/product/0636920034919.do))

In [26]:
```
from IPython.display import Image
Image('split_apply_combine.png')
```

Out[26]:



Let's break take this in smaller steps. First, let's look at the data by hour across all days in the year.

In [27]:
```
pd.value_counts(times.hour)
```

Out[27]:
```
17    14163
16    11629
8     10967
18    10382
15     9850
9      9751
13     9575
12     9571
14     9096
11     8864
10     7761
19     6939
7      6093
20     4792
21     3730
22     2484
6      1855
23     1749
0      1022
5       905
1       682
2       478
4       316
3       192
dtype: int64
```

groupby allows us to look at the number of values for each column and each value.

In [28]:
```
data.groupby(times.hour).count()
```

Out[28]:

	trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_name	from_stat
0	1022	1022	1022	1022	1022	1022	1022	1022
1	682	682	682	682	682	682	682	682
2	478	478	478	478	478	478	478	478
3	192	192	192	192	192	192	192	192
4	316	316	316	316	316	316	316	316
5	905	905	905	905	905	905	905	905
6	1855	1855	1855	1855	1855	1855	1855	1855
7	6093	6093	6093	6093	6093	6093	6093	6093
8	10967	10967	10967	10967	10967	10967	10967	10967
9	9751	9751	9751	9751	9751	9751	9751	9751
10	7761	7761	7761	7761	7761	7761	7761	7761
11	8864	8864	8864	8864	8864	8864	8864	8864
12	9571	9571	9571	9571	9571	9571	9571	9571
13	9575	9575	9575	9575	9575	9575	9575	9575
14	9096	9096	9096	9096	9096	9096	9096	9096
15	9850	9850	9850	9850	9850	9850	9850	9850
16	11629	11629	11629	11629	11629	11629	11629	11629
17	14163	14163	14163	14163	14163	14163	14163	14163
18	10382	10382	10382	10382	10382	10382	10382	10382
19	6939	6939	6939	6939	6939	6939	6939	6939
20	4792	4792	4792	4792	4792	4792	4792	4792
21	3730	3730	3730	3730	3730	3730	3730	3730
22	2484	2484	2484	2484	2484	2484	2484	2484
23	1749	1749	1749	1749	1749	1749	1749	1749

Now, let's find the average length of a ride as a function of time of day:

In [29]:
```
data.groupby(times.hour)['tripminutes'].mean()
```

Out[29]:
```
0     18.293162
1     16.812000
2     26.467510
3     22.643443
4     18.595762
5     13.565035
6     12.091993
7     12.378344
8     12.544350
9     15.175861
10    23.558911
11    25.645489
12    26.052903
13    27.878785
14    28.354453
15    26.164124
16    21.257375
17    17.388788
18    16.706635
19    16.886609
20    17.463562
21    15.227905
22    15.931296
23    16.006255
Name: tripminutes, dtype: float64
```

You can specify a groupby using the names of table columns and compute other functions, such as the mean.

In [30]:
```
data.groupby(['gender'])['tripminutes'].mean()
```

Out[30]:
```
gender
Female    12.137525
Male       9.547313
Other     10.898911
Name: tripminutes, dtype: float64
```

The simplest version of a groupby looks like this, and you can use almost any aggregation function you wish (mean, median, sum, minimum, maximum, standard deviation, count, etc.)

```
<data object>.groupby(<grouping values>).<aggregate>()
```

You can even group by multiple values: for example we can look at the trip duration by time of day and by gender:

In [31]:
```
grouped = data.groupby([times.hour, 'gender'])['tripminutes'].mean()
grouped
```

```
Out[31]:     gender
          0   Female     9.016608
              Male       8.750898
              Other      8.123788
          1   Female     8.721844
              Male       7.865786
              Other     13.765683
          2   Female     8.371583
              Male       8.880909
              Other      7.344022
          3   Female    16.796675
              Male       8.087312
              Other      5.643350
          4   Female     9.123506
              Male      12.134917
              Other      2.810767
          5   Female    12.970906
              Male       8.261415
          6   Female    10.954091
              Male       8.038184
              Other      9.752973
          7   Female    12.392548
              Male       9.461130
              Other     10.525419
          8   Female    11.309898
              Male       9.219013
              Other     10.129815
          9   Female    10.881655
              Male       9.019535
              Other     11.167447
          10  Female    12.152443
                          ...
          14  Female    13.085510
              Male      10.097013
              Other     11.780128
          15  Female    12.521131
              Male      10.586964
              Other     10.580798
          16  Female    12.614844
              Male      10.534829
              Other     13.845402
          17  Female    12.927452
              Male      10.507247
              Other     11.990486
          18  Female    12.414073
              Male       9.432061
              Other      9.477819
          19  Female    12.008509
              Male       9.068067
              Other      9.579921
          20  Female    11.716320
              Male       9.253255
              Other      8.051081
          21  Female    11.241531
              Male       8.859545
              Other      7.606792
          22  Female    11.088358
              Male       8.733132
              Other      6.618971
          23  Female    11.032336
              Male       8.085837
              Other      5.183287
          Name: tripminutes, dtype: float64
```

The `unstack()` operation can help make sense of this type of multiply-grouped data. What this technically does is split a multiple-valued index into an index plus columns:

In [32]:
```
grouped.unstack()
```

Out[32]:

gender	Female	Male	Other
0	9.016608	8.750898	8.123788
1	8.721844	7.865786	13.765683
2	8.371583	8.880909	7.344022
3	16.796675	8.087312	5.643350
4	9.123506	12.134917	2.810767
5	12.970906	8.261415	NaN
6	10.954091	8.038184	9.752973
7	12.392548	9.461130	10.525419
8	11.309898	9.219013	10.129815
9	10.881655	9.019535	11.167447
10	12.152443	9.275972	11.615609
11	11.763075	9.066147	13.589810
12	12.436949	9.859569	10.503408
13	13.031992	9.675181	10.693493
14	13.085510	10.097013	11.780128
15	12.521131	10.586964	10.580798
16	12.614844	10.534829	13.845402
17	12.927452	10.507247	11.990486
18	12.414073	9.432061	9.477819
19	12.008509	9.068067	9.579921
20	11.716320	9.253255	8.051081
21	11.241531	8.859545	7.606792
22	11.088358	8.733132	6.618971
23	11.032336	8.085837	5.183287

5. Visualizing data with pandas

Of course, looking at tables of data is not very intuitive. Fortunately Pandas has many useful plotting functions built-in, all of which make use of the `matplotlib` library to generate plots.

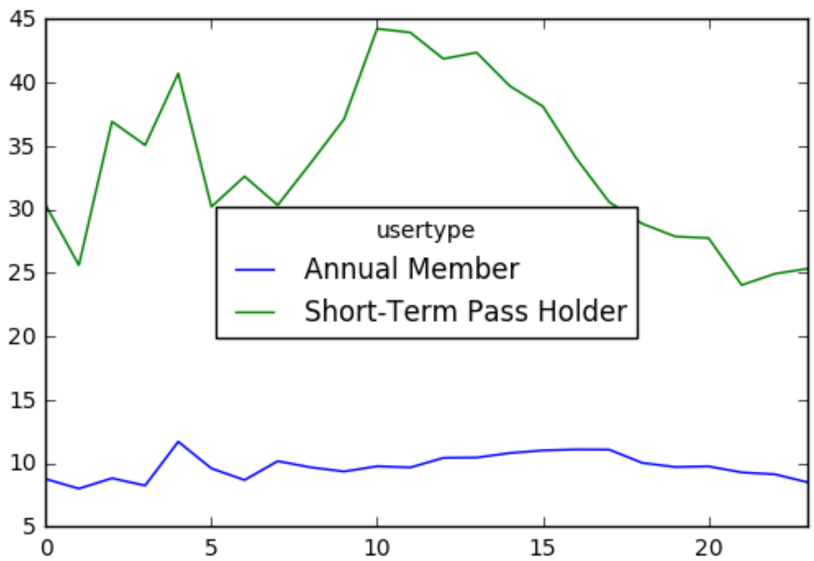
Whenever you do plotting in the IPython notebook, you will want to first run this *magic command* which configures the notebook to work well with plots:

In [33]:
```
%matplotlib inline
```

Now we can simply call the `plot()` method of any series or dataframe to get a reasonable view of the data:

In [34]:
```
data.groupby([times.hour, 'usertype'])['tripminutes'].mean().unstack().plot()
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7f56dc760c50>



Adjusting the Plot Style

The default formatting is not very nice; I often make use of the Seaborn (http://stanford.edu/~mwaskom/software/seaborn/) library for better plotting defaults.

You should do this in bash

```
$ conda install seaborn
```

Then this in python

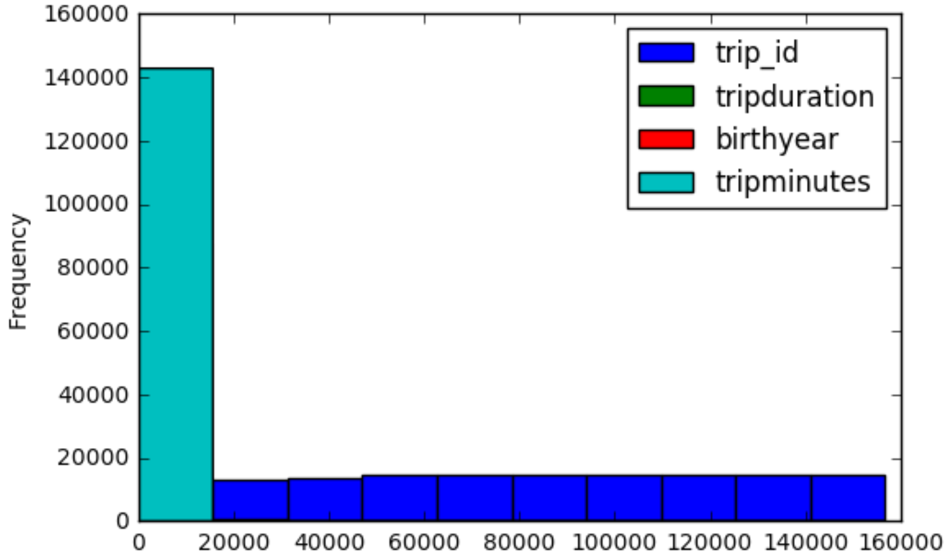
```
import seaborn
seaborn.set()
data.groupby([times.hour, 'usertype'])['tripminutes'].mean().unstack().plot()
```

Other plot types

Pandas supports a range of other plotting types; you can find these by using the autocomplete on the `plot` method:

In [35]:
```
data.plot.hist()
```

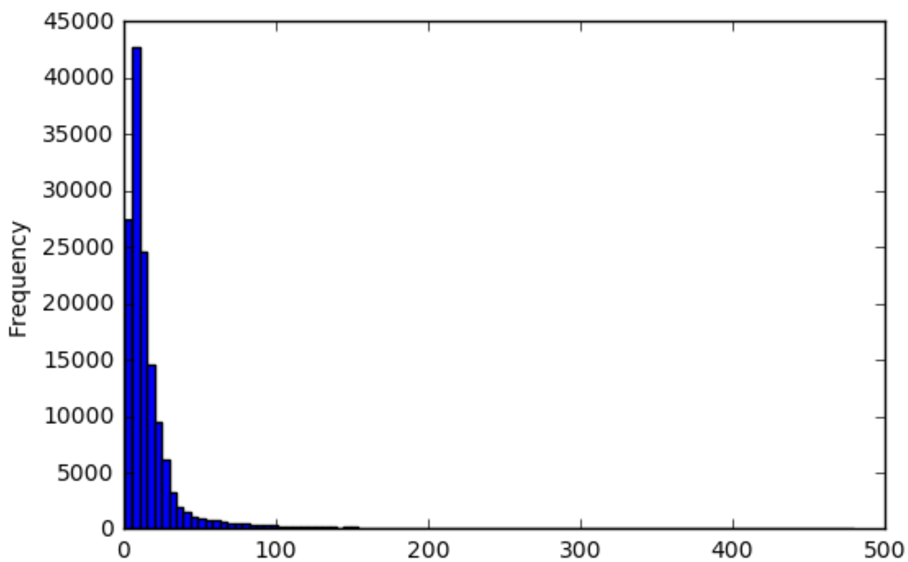
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f56dc3f0290>



For example, we can create a histogram of trip durations:

In [36]:
```
data['tripminutes'].plot.hist(bins=100)
```

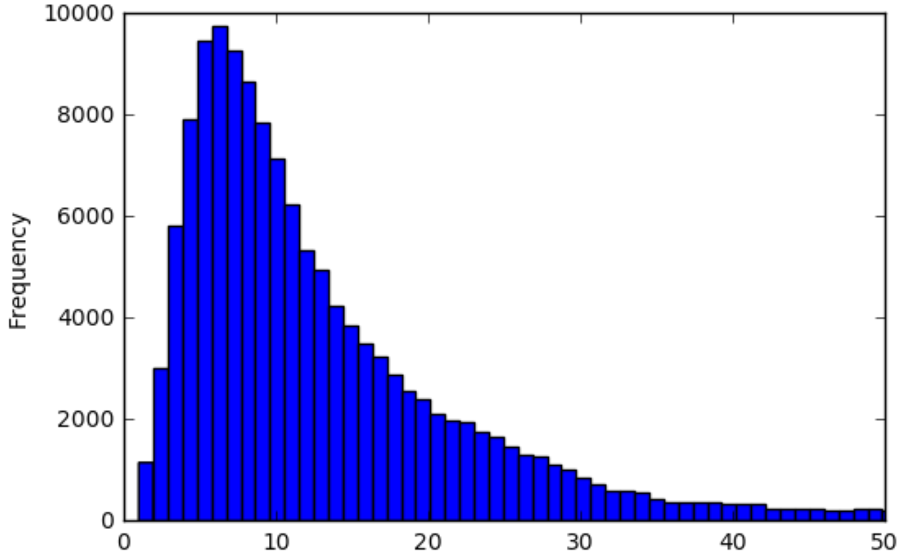
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f56cd03b190>



If you'd like to adjust the x and y limits of the plot, you can use the `set_xlim()` and `set_ylim()` method of the resulting object:

In [37]:
```
plot = data['tripminutes'].plot.hist(bins=500)
plot.set_xlim(0, 50)
```

Out[37]: (0, 50)



Breakout: Exploring the Data

1. Make a plot of the total number of rides as a function of month of the year (You'll need to extract the month, use a `groupby`, and find the appropriate aggregation to count the number in each group).

In [ ]:

1. Split this plot by gender. Do you see any seasonal ridership patterns by gender?

In [ ]:

1. Split this plot by user type. Do you see any seasonal ridership patterns by usertype?

In [ ]:

1. Repeat the above three steps, counting the number of rides by time of day rather thatn by month.

In [ ]:

1. Are there any other interesting insights you can discover in the data using these tools?

In [ ]:

In [ ]:

```

```

Looking Forward to Homework

In the homework this week, you will have a chance to apply some of these patterns to a brand new (but closely related) dataset.