# Batch Processing in R and Python

This is meant as a companion notebook for the the SMU MSDS program. Specifically, we will build off of chapter five in the text "Data Science in R", http://www.amazon.com/Data-Science-Approach-Computational-Reasoning/dp/1482234815 (http://www.amazon.com/Data-Science-Approach-Computational-Reasoning/dp/1482234815).

In this notebook we will look at ways for loading and analyzing data out-of-core using a batch processing method. Specifically, we will be using the split-apply-combine approach with a few different APIs. In R, we will use:

- `bigmemory` for memory mapping the variables: https://cran.r-project.org/web/packages/bigmemory/index.html (https://cran.r-project.org/web/packages/bigmemory/index.html)
- `bigtabulate`
- `biganalytics`
- `doMC`

In python, we will also be using a number of different libraries

- `pandas`
- `numpy`
- `graphlab-create`

---

# 1.0 Downloading the dataset (using python)

You can access descriptions of the data files from here: http://stat-computing.org/dataexpo/2009/the-data.html (http://stat-computing.org/dataexpo/2009/the-data.html). You can manually download each of the zipped files OR run the following script to download the files into a folder in the same directory as this notebook called "Data".

In the following blocks of code, we download the data and then decompress the files into .csv files. Each csv contains data for one year of airline flights.

Note: If you want an alternative R version for performing the download operations, see:

- http://www.cybaea.net/journal/2010/08/05/Big-data-for-R/ (http://www.cybaea.net/journal/2010/08/05/Big-data-for-R/)

In [4]:
```
import os, sys
# create a Data directory if not done already
path = "/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData"
if not os.path.exists(path):
    os.mkdir( path, 0755 )
```

In [6]:
```
import urllib # this is part of the standard library for python

years_to_download = range(1987,2009) # get the years 1987 through 2008
baseurl = 'http://stat-computing.org/dataexpo/2009/%d.csv.bz2'

files = []
for year in years_to_download:
    # prepare strings
    url_of_data_file = baseurl%(year) # get the URL for the data file
    save_as_filename = '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
    # save as this
    files += [save_as_filename] # save name of the compressed file

    # download file
    print 'Downloading %s to %s'%(url_of_data_file, save_as_filename) # prog
    urllib.urlretrieve(url_of_data_file, save_as_filename) #execute downloa

print files
```

```
Downloading http://stat-computing.org/dataexpo/2009/1987.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1987.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1987.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1988.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1988.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1988.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1989.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1989.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1989.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1990.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1990.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1990.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1991.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1991.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1991.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1992.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1992.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1992.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1993.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1993.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1993.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1994.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1994.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1994.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1995.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1995.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1995.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1996.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1996.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1996.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1997.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1997.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1997.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1998.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1998.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/1998.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/1999.csv.bz2 (http://
stat-computing.org/dataexpo/2009/1999.csv.bz2) to /Users/mmcgee/Dropbox/2
```

```
016 Summer Courses/MSDS 7333/AirlineData/1999.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2000.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2000.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2000.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2001.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2001.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2001.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2002.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2002.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2002.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2003.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2003.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2003.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2004.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2004.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2004.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2005.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2005.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2005.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2006.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2006.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2006.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2007.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2007.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2007.csv.bz2
Downloading http://stat-computing.org/dataexpo/2009/2008.csv.bz2 (http://
stat-computing.org/dataexpo/2009/2008.csv.bz2) to /Users/mmcgee/Dropbox/2
016 Summer Courses/MSDS 7333/AirlineData/2008.csv.bz2
['/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1987.cs
v.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/
1988.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airli
neData/1989.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 733
3/AirlineData/1990.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/M
SDS 7333/AirlineData/1991.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Co
urses/MSDS 7333/AirlineData/1992.csv.bz2', '/Users/mmcgee/Dropbox/2016 Su
mmer Courses/MSDS 7333/AirlineData/1993.csv.bz2', '/Users/mmcgee/Dropbox/
2016 Summer Courses/MSDS 7333/AirlineData/1994.csv.bz2', '/Users/mmcgee/D
ropbox/2016 Summer Courses/MSDS 7333/AirlineData/1995.csv.bz2', '/Users/m
mcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1996.csv.bz2', '/
Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1997.csv.b
z2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/199
8.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineD
ata/1999.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/A
irlineData/2000.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS
 7333/AirlineData/2001.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summer Cours
es/MSDS 7333/AirlineData/2002.csv.bz2', '/Users/mmcgee/Dropbox/2016 Summe
r Courses/MSDS 7333/AirlineData/2003.csv.bz2', '/Users/mmcgee/Dropbox/201
6 Summer Courses/MSDS 7333/AirlineData/2004.csv.bz2', '/Users/mmcgee/Drop
box/2016 Summer Courses/MSDS 7333/AirlineData/2005.csv.bz2', '/Users/mmcg
ee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/2006.csv.bz2', '/Use
rs/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/2007.csv.bz
2', '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/200
8.csv.bz2']
```

In [7]:
```
import bz2 # this is also part of the python standard library

# Now lets decompress all the files
for filename in files:
    # get file names
    filepath = filename
    newfilepath = filename[:-4]
    print 'Decompressing', filepath,'to', newfilepath

    # go through the decompressed chunks and write out to a decompressed fil
    with open(newfilepath, 'wb') as new_file, bz2.BZ2File(filepath, 'rb') as
        for data in iter(lambda : file.read(100 * 1024), b''):
            new_file.write(data)
```

```
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1987.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1987.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1988.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1988.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1989.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1989.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1990.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1990.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1991.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1991.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1992.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1992.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1993.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1993.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1994.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1994.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1995.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1995.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1996.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1996.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1997.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1997.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1998.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1998.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/1999.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/1999.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2000.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2000.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
```

```
Data/2001.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2001.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2002.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2002.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2003.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2003.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2004.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2004.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2005.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2005.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2006.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2006.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2007.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2007.csv
Decompressing /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airline
Data/2008.csv.bz2 to /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
AirlineData/2008.csv
```

### If you want to delete the compressed files

Run the following block ONLY if you want to delets the compressed bz2 files from your system.

In [11]:
```
!rm '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/'*.bz2
```

---

# 2.0 Loading data into memory

Now that the data has downloaded and been decompressed, we can load a single file into memory to ensure that everything decompressed correctly. For each file, we could load it into memory and then save the length of the file. Let's do this in both python and in R for two files (to keep runtime at a minimum).

In [13]:
```
# Loading individual files in python
import pandas as pd
import numpy as np
import sys

total_length = 0
for year in [1987,1988]:
    # get file name of the csv
    # note that we can also load in the raw .bz2 file in python (or R)
    # but the decompression step for these files sizes takes a huge performa
    csvfile = '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineDa
    print 'loading', csvfile
    sys.stdout.flush()

    # read the file and increment the lines count
    df = pd.read_csv(csvfile) # note that this is a big operation, especiall
    # one way of making this shorter is to filter which columns we are loadi

    total_length += len(df)

print 'Answer from python:', total_length
df.head()
```

```
loading /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1
987.csv
loading /Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1
988.csv
Answer from python: 6513922
```

Out[13]:

| | Year | Month | DayofMonth | DayOfWeek | DepTime | CRSDepTime | ArrTime | CRSArrTime | U |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1988 | 1 | 9 | 6 | 1348 | 1331 | 1458 | 1435 | P |
| **1** | 1988 | 1 | 10 | 7 | 1334 | 1331 | 1443 | 1435 | P |
| **2** | 1988 | 1 | 11 | 1 | 1446 | 1331 | 1553 | 1435 | P |
| **3** | 1988 | 1 | 12 | 2 | 1334 | 1331 | 1438 | 1435 | P |
| **4** | 1988 | 1 | 13 | 3 | 1341 | 1331 | 1503 | 1435 | P |

5 rows × 29 columns

In [16]:
```
print 'CSV File Format'
!head '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1987.
```

```
CSV File Format
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,Uni
queCarrier,FlightNum,TailNum,ActualElapsedTime,CRSElapsedTime,AirTime,Arr
Delay,DepDelay,Origin,Dest,Distance,TaxiIn,TaxiOut,Cancelled,Cancellation
Code,Diverted,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircra
ftDelay
1987,10,14,3,741,730,912,849,PS,1451,NA,91,79,NA,23,11,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,15,4,729,730,903,849,PS,1451,NA,94,79,NA,14,-1,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,17,6,741,730,918,849,PS,1451,NA,97,79,NA,29,11,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,18,7,729,730,847,849,PS,1451,NA,78,79,NA,-2,-1,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,19,1,749,730,922,849,PS,1451,NA,93,79,NA,33,19,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,21,3,728,730,848,849,PS,1451,NA,80,79,NA,-1,-2,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,22,4,728,730,852,849,PS,1451,NA,84,79,NA,3,-2,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,23,5,731,730,902,849,PS,1451,NA,91,79,NA,13,1,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,24,6,744,730,908,849,PS,1451,NA,84,79,NA,19,14,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
```

In [19]:
```
# Load up the R extensions for Rpy2
%load_ext rmagic
%load_ext rpy2.ipython

# you might need to execute this block twice
# now, lets use R to load the files and count lengths
```

```
The rmagic extension is already loaded. To reload it, use:
  %reload_ext rmagic
```

In [20]:
```
%%R
# everything in this block is R code
# also note that R is fairly inefficient in its import of CSV files
#    so this may take a while to compute
total_length <- 0
for(year in 1987:1988){
    filename = paste("/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Ai
    x<-read.csv(filename)
    total_length <- total_length + nrow(x)
}
print(total_length)
```

```
[1] 6513922
```

So now we can see that there were about 6.5M different entries in the years 1987 and 1988. However, we really want to be working with one really large descriptor of all years, rather than loading up each year from disc each time. Actually, the operations we had above are much more appropriate for a SQL query, but they are not quite as scalable for some of the other analyses that we want to do later on.

The python code is noticeably faster at loading in the data, so let's use python to pre-process the csv files and make sure all the data is an integer. All the text data must be coded as numeric (an integer) when we use R's big memory package, so lets go file by file and make sure its ready to use with R.

## 2.1 Preprocessing Data in Chunks

In order to replace all the strings in the data frame, we first need to know exactly all the unique strings contained in each column. The first block takes a pass on all the data and finds all the unique string entries in the entire dataset. This means loading all the data files in one pass and concatenating the unique entries into a data structure. I have chosen to use a dictionary as the data structure with the name of each key in the dictionary being the column name of the variable we want to convert to an integer. The value of the key is a `set` of strings.

Once we have this dataset, we can then replace the unique values properly (we will need to load the data files again!!). We can then replace the values with an integer. After replacing them, we can resave the data frame as a csv. Phew!

Please note that the runtime of these blocks aggregates to ~60 minutes (or more) depending on your system. If you want an alternative implementation using R only, please see:

- http://www.cybaea.net/journal/2010/08/05/Big-data-for-R/ (http://www.cybaea.net/journal/2010/08/05/Big-data-for-R/)

In [6]:
```
# In this data, there are multiple variables that are objects
# these need to be appropriately converted to numbers (integers)
# To do this, we must know the uniqe values in each of the columns, let's do
import pandas as pd
import numpy as np
import sys
import time
import cPickle as pickle

unique_values = {} # create an empty dictionary of the column name an the un
for year in range(1987,2009):
    t = time.time()
    # get file name of the csv
    csvfile = '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineDa
    print 'loading',csvfile,
    sys.stdout.flush()

    # read the file
    df = pd.read_csv(csvfile,usecols=['Origin', 'Dest', 'UniqueCarrier','Tai
    #df = df.select_dtypes(exclude=['float64','int64']) # grab only the non-

    print '...finding unique values',
    sys.stdout.flush()

    for col in df.columns:
        # check to see if we have seen this column before
        s = set(df[col].values.astype(np.str))
        if col not in unique_values:
            # if not, then create a key with the unique values for that colu
            unique_values[col] = s
        else:
            # otherwise make sure that the remaining columns are unique
            unique_values[col] |= s

    print '...finished, %.2f seconds'%(time.time()-t)
    sys.stdout.flush()
    del df

# Save out the dictionary for later use
pickle.dump( unique_values, open( "/Users/mmcgee/Dropbox/2016 Summer Courses
```

```
loading Data/1987.csv ...finding unique values ...finished, 3.06 seconds
loading Data/1988.csv ...finding unique values ...finished, 11.16 seconds
loading Data/1989.csv ...finding unique values ...finished, 11.29 seconds
loading Data/1990.csv ...finding unique values ...finished, 11.76 seconds
loading Data/1991.csv ...finding unique values ...finished, 11.38 seconds
loading Data/1992.csv ...finding unique values ...finished, 11.66 seconds
loading Data/1993.csv ...finding unique values ...finished, 11.33 seconds
loading Data/1994.csv ...finding unique values ...finished, 11.48 seconds
loading Data/1995.csv ...finding unique values ...finished, 11.71 seconds
loading Data/1996.csv ...finding unique values ...finished, 11.96 seconds
loading Data/1997.csv ...finding unique values ...finished, 12.19 seconds
loading Data/1998.csv ...finding unique values ...finished, 12.04 seconds
loading Data/1999.csv ...finding unique values ...finished, 12.34 seconds
loading Data/2000.csv ...finding unique values ...finished, 12.77 seconds
loading Data/2001.csv ...finding unique values ...finished, 13.26 seconds
loading Data/2002.csv ...finding unique values ...finished, 11.57 seconds
```

```
loading Data/2003.csv ...finding unique values ...finished, 15.17 seconds
loading Data/2004.csv ...finding unique values ...finished, 16.57 seconds
loading Data/2005.csv ...finding unique values ...finished, 16.49 seconds
loading Data/2006.csv ...finding unique values ...finished, 16.57 seconds
loading Data/2007.csv ...finding unique values ...finished, 17.06 seconds
loading Data/2008.csv ...finding unique values ...finished, 16.40 seconds
```

```
/Library/Python/2.7/site-packages/IPython/core/interactiveshell.py:2868:
 DtypeWarning: Columns (22) have mixed types. Specify dtype option on imp
ort or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [7]:
```
# let's take a look at the dictionary
print unique_values.keys()
print 'One example:',unique_values['CancellationCode']
```

```
['Origin', 'TailNum', 'UniqueCarrier', 'Dest', 'CancellationCode']
One example: set(['A', 'C', 'B', 'D', 'nan'])
```

In [8]:
```
def fast_numpy_replace(np_vector,replace_set):
    # you can look at this function at your leisure, but essentially we use
    # comparison to try and speed up the analysis
    replace_set = np.array(list(replace_set)) # get "possible values" as a r
    n = np.ndarray(np_vector.shape).astype(np.float64) # fill in this matrix

    vector_as_set,idx_back = np.unique(np_vector,return_inverse=True) # get

    # now loop through the unique values for this dataset
    for idx,val in enumerate(vector_as_set):
        # find what number this should be (like a hash)
        category_num = np.nonzero(replace_set == val)[0][0]
        n[idx_back==idx] = category_num # set the values as this category, v

    return n.astype(np.float64)

fileHandle = open('/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Airli
years = range(1987,2009)
for year in years:
    t = time.time()

    # get file name of the csv
    csvfile = '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineDa
    print 'Running...',csvfile,
    sys.stdout.flush()

    # read the file
    df = pd.read_csv(csvfile)

    print 'loaded, ...replacing values',
    sys.stdout.flush()

    # now replace the matching columnar data with the proper number category
    for key in unique_values.keys():
        if key in df:
            print key[0:4],
            sys.stdout.flush()
            tmp = df[key].values.astype(np.str)
            df[key] = fast_numpy_replace(tmp,unique_values[key])

    print '...',
    sys.stdout.flush()

    for col in df:
        df[col] = np.round(df[col].astype(np.float64)) # use floats to keep

    print 'writing',
    sys.stdout.flush()

    # these lines make one large file with the numeric data
    # it also solves a problem with pandas closing the file that takes an ir
    # NOTE: using binary here would be a huge speedup, but I am not sure abo
    # backing file for bigmatrix, so we stick with CSV
    # TODO: find out if the backing file is just a dump of the c struct to r
    if year==years[0]:
        df.to_csv(fileHandle,index=False, index_label=False, na_rep="NA",flo
```

```
    else:
        df.to_csv(fileHandle, mode='a', header=False, index=False, index_lab

    print ', %.2f sec.'%(time.time()-t)
    del df

print 'closing file',
sys.stdout.flush()

fileHandle.close()
print '...Done'
```

```
Running... Data/1987.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 32.90 sec.
Running... Data/1988.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 128.47 sec.
Running... Data/1989.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 126.11 sec.
Running... Data/1990.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 135.32 sec.
Running... Data/1991.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 125.06 sec.
Running... Data/1992.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 124.95 sec.
Running... Data/1993.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 124.91 sec.
Running... Data/1994.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 128.19 sec.
Running... Data/1995.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 191.00 sec.
Running... Data/1996.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 186.53 sec.
Running... Data/1997.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 190.69 sec.
Running... Data/1998.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 189.00 sec.
Running... Data/1999.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 205.32 sec.
Running... Data/2000.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 205.02 sec.
Running... Data/2001.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 225.08 sec.
Running... Data/2002.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 238.68 sec.
Running... Data/2003.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 260.79 sec.
Running... Data/2004.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 304.34 sec.
Running... Data/2005.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 299.36 sec.
Running... Data/2006.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 425.77 sec.
Running... Data/2007.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 321.65 sec.
Running... Data/2008.csv loaded, ...replacing values Orig Tail Uniq Dest
 Canc ... writing , 280.41 sec.
closing file ...Done
```

In [9]:
```
# now lets take a look to see what has actually changed in the file
# let's load the head of 1987 and the big CSV file to see how they compare
print 'New File Format:'
!head '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/Airli
print ''
print 'Old File Format'
!head '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/1987.
```

```
New File Format:
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,Uni
queCarrier,FlightNum,TailNum,ActualElapsedTime,CRSElapsedTime,AirTime,Arr
Delay,DepDelay,Origin,Dest,Distance,TaxiIn,TaxiOut,Cancelled,Cancellation
Code,Diverted,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircra
ftDelay
1987,10,14,3,741,730,912,849,7,1451,9719,91,79,NA,23,11,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA
1987,10,15,4,729,730,903,849,7,1451,9719,94,79,NA,14,-1,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA
1987,10,17,6,741,730,918,849,7,1451,9719,97,79,NA,29,11,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA
1987,10,18,7,729,730,847,849,7,1451,9719,78,79,NA,-2,-1,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA
1987,10,19,1,749,730,922,849,7,1451,9719,93,79,NA,33,19,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA
1987,10,21,3,728,730,848,849,7,1451,9719,80,79,NA,-1,-2,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA
1987,10,22,4,728,730,852,849,7,1451,9719,84,79,NA,3,-2,172,202,447,NA,NA,
0,4,0,NA,NA,NA,NA,NA
1987,10,23,5,731,730,902,849,7,1451,9719,91,79,NA,13,1,172,202,447,NA,NA,
0,4,0,NA,NA,NA,NA,NA
1987,10,24,6,744,730,908,849,7,1451,9719,84,79,NA,19,14,172,202,447,NA,N
A,0,4,0,NA,NA,NA,NA,NA

Old File Format
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,Uni
queCarrier,FlightNum,TailNum,ActualElapsedTime,CRSElapsedTime,AirTime,Arr
Delay,DepDelay,Origin,Dest,Distance,TaxiIn,TaxiOut,Cancelled,Cancellation
Code,Diverted,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircra
ftDelay
1987,10,14,3,741,730,912,849,PS,1451,NA,91,79,NA,23,11,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,15,4,729,730,903,849,PS,1451,NA,94,79,NA,14,-1,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,17,6,741,730,918,849,PS,1451,NA,97,79,NA,29,11,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,18,7,729,730,847,849,PS,1451,NA,78,79,NA,-2,-1,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,19,1,749,730,922,849,PS,1451,NA,93,79,NA,33,19,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,21,3,728,730,848,849,PS,1451,NA,80,79,NA,-1,-2,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,22,4,728,730,852,849,PS,1451,NA,84,79,NA,3,-2,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,23,5,731,730,902,849,PS,1451,NA,91,79,NA,13,1,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
1987,10,24,6,744,730,908,849,PS,1451,NA,84,79,NA,19,14,SAN,SFO,447,NA,NA,
0,NA,0,NA,NA,NA,NA,NA
```

In [10]:
```
# let's now look at the tail of our big dataset and the tail of the 2008 fil
# do they compare nicely?
print 'New File Format:'
!tail '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/Airli
print ''
print 'Old File Format:'
!tail !tail '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData
```

```
New File Format:
2008,12,13,6,1007,847,1149,1010,0,1631,2497,162,143,122,99,80,251,20,689,
8,32,0,4,0,1,0,19,0,79
2008,12,13,6,638,640,808,753,0,1632,10676,90,73,50,15,-2,130,255,270,14,2
6,0,4,0,0,0,15,0,0
2008,12,13,6,756,800,1032,1026,0,1633,959,96,86,56,6,-4,304,255,425,23,1
7,0,4,0,NA,NA,NA,NA,NA
2008,12,13,6,612,615,923,907,0,1635,2826,131,112,103,16,-3,53,272,546,5,2
3,0,4,0,0,0,16,0,0
2008,12,13,6,749,750,901,859,0,1636,9994,72,69,41,2,-1,180,255,215,20,11,
0,4,0,NA,NA,NA,NA,NA
2008,12,13,6,1002,959,1204,1150,0,1636,9994,122,111,71,14,3,251,21,533,6,
45,0,4,0,NA,NA,NA,NA,NA
2008,12,13,6,834,835,1021,1023,0,1637,10045,167,168,139,-2,-1,251,185,87
4,5,23,0,4,0,NA,NA,NA,NA,NA
2008,12,13,6,655,700,856,856,0,1638,12867,121,116,85,0,-5,235,255,545,24,
12,0,4,0,NA,NA,NA,NA,NA
2008,12,13,6,1251,1240,1446,1437,0,1639,9994,115,117,89,9,11,21,255,533,1
3,13,0,4,0,NA,NA,NA,NA,NA
2008,12,13,6,1110,1103,1413,1418,0,1641,10045,123,135,104,-5,7,181,255,87
4,8,11,0,4,0,NA,NA,NA,NA,NA

Old File Format:
2008,12,13,6,1007,847,1149,1010,DL,1631,N909DA,162,143,122,99,80,ATL,IAH,
689,8,32,0,,0,1,0,19,0,79
2008,12,13,6,638,640,808,753,DL,1632,N604DL,90,73,50,15,-2,JAX,ATL,270,1
4,26,0,,0,0,0,15,0,0
2008,12,13,6,756,800,1032,1026,DL,1633,N642DL,96,86,56,6,-4,MSY,ATL,425,2
3,17,0,,0,NA,NA,NA,NA,NA
2008,12,13,6,612,615,923,907,DL,1635,N907DA,131,112,103,16,-3,GEG,SLC,54
6,5,23,0,,0,0,0,16,0,0
2008,12,13,6,749,750,901,859,DL,1636,N646DL,72,69,41,2,-1,SAV,ATL,215,20,
11,0,,0,NA,NA,NA,NA,NA
2008,12,13,6,1002,959,1204,1150,DL,1636,N646DL,122,111,71,14,3,ATL,IAD,53
3,6,45,0,,0,NA,NA,NA,NA,NA
2008,12,13,6,834,835,1021,1023,DL,1637,N908DL,167,168,139,-2,-1,ATL,SAT,8
74,5,23,0,,0,NA,NA,NA,NA,NA
2008,12,13,6,655,700,856,856,DL,1638,N671DN,121,116,85,0,-5,PBI,ATL,545,2
4,12,0,,0,NA,NA,NA,NA,NA
2008,12,13,6,1251,1240,1446,1437,DL,1639,N646DL,115,117,89,9,11,IAD,ATL,5
33,13,13,0,,0,NA,NA,NA,NA,NA
2008,12,13,6,1110,1103,1413,1418,DL,1641,N908DL,123,135,104,-5,7,SAT,ATL,
874,8,11,0,,0,NA,NA,NA,NA,NA
```

In the above code, we also created a large file with all the data from every year inside of it. This is now saved as `AirlineDataAll`. Let's take a look at it.

In [4]:
```
!ls -all '/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineData/'*
```

```
-rw-r--r--@ 1 eclarson  staff  11785518426 Jan 14 17:06 Data/AirlineDataA
ll.csv
```

Now we can see that the `AirlineDataAll` file above is about 12GB. Loading it into main memory like normal is not an option, so lets create some aggregations in both R and python. Lets start with the `bigmemory` package in R and run some of the examples from the book.

---

# 3.0 Analyzing the dataset in R

You need to install the bigmemory package in R. Using the GUI installer from R's IDE is typically the best option. Be sure you have clicked the `also install dependencies` radio edit when installing. You should see output much like this:

---

```
also installing the dependencies ‘memoise’, ‘crayon’, ‘praise’, ‘big
memory.sri’, ‘testthat’

trying URL 'http://cran.revolutionanalytics.com/bin/macosx/maverick
s/contrib/3.1/bigmemory.sri_0.1.3.tgz'
Content type 'application/octet-stream' length 14991 bytes (14 KB)
opened URL
==================================================
downloaded 14 KB

trying URL 'http://cran.revolutionanalytics.com/bin/macosx/maverick
s/contrib/3.1/bigmemory_4.5.8.tgz'
Content type 'application/octet-stream' length 1399658 bytes (1.3 M
B)
opened URL
==================================================
downloaded 1.3 MB


The downloaded binary packages are in
    /var/folders/t4/hzf28kx94nv2vp_fvfsfvkdw0000gn/T//RtmpoYpVF5/dow
nloaded_packages
```

---

Now let's check to be sure everything installed correctly by loading the package here.

In [8]:
```
%%R

library(bigmemory)
```

Okay! So now we have bigmemory installed! But what we really need to do is create a binary file that memory maps the data we are interested in. This is called a 'backing file'. To create that binary bakcing file we must read the data using the `big.matrix` package.

There are several things that we should be aware of:

- The `bigmemory` package only supports backing of R `matrices` which means all the data must be the exact same type. For this dataset we are using mostly just timestamps and integer values (replacing many of the category variables with integers).
  - This is important because it means the binary backing file size will depend on the datatype of the matrix. It also means floating point numbers will be truncated and we will lose precision if not careful.
- Even though its a matrix, `bigmemory` allows us to use headers, so we can access it similarly to a `data.frame`
- We will not be able to store sparse matrices easily using `bigmemory`-it can only save dense matrices. However, needing to memory map a sparse matrix is typically an edge case for Data Scientists and we can just use sparse representations.
- Under the hood, we are just using a pointer on disk to memory for a c++ structure.
- `bigmemory` basically threw out all their Windows support in 2013, so no binaries exist for R > 2.15.3
- But... it supports mac and linux just fine!
- The big advantage to using bigmemory: it is very versatile and allows deep control over different grouping operations and apply functions.

The operation below will take about 25 minutes to complete.

In [13]:
```
%%R

#So now lets actually go through the pain of creating a memory mapped large
# we need to run this command to create the decriptor
# if you have already run this, don't run it again!! Instead just run the ne
x <- read.big.matrix("/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Ai
                     backingpath="Data",  backingfile="airline.bin",
                     descriptorfile="airline.desc",
                     type="integer", extraCols="age")

print(dim(x))
```

```
[1] 123534969        30
```

In [7]:
```
# how big was the binary file it created?
!ls -all Data/*.bin
# This is a 14 GB file??? That means the data is not compressed on disk!
# Well, I guess that's okay, because we really want to know how fast the ana
```

```
-rw-r--r--@ 1 eclarson  staff  14824196281 Jan 15 12:08 Data/airline.bin
```

In [9]:
```
# CONVENIENCE BLOCK FOR THOSE LOADING SESSION FROM DISK
# Load up the R extensions for Rpy2
%load_ext rmagic
%load_ext rpy2.ipython

import cPickle as pickle

unique_values = pickle.load( open( "/Users/mmcgee/Dropbox/2016 Summer Course
# you might need to execut this block twice
# now, lets use R to load the files and count lengths
```

```
The rmagic extension is already loaded. To reload it, use:
  %reload_ext rmagic
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

In [10]:
```
%%R

# attach the binary backing file through its descriptor
x <- attach.big.matrix("/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/
```

## 3.1 Analyzing Busiest Airports in R

Using big memory is a lot like using R data.frames but there is decreased functionality. Basically, we need to use the optimized functions whenever possible. That means using functions like `bigsplit` in place of using `split`.

- this is also a great resource: http://www.bytemining.com/wp-content/uploads/2010/08/r_hpc_II.pdf (http://www.bytemining.com/wp-content/uploads/2010/08/r_hpc_II.pdf)

---

So now we are setup to do some analysis, but we actually need to install more packages that are sister repositories to big memory. Specifically, you will need:

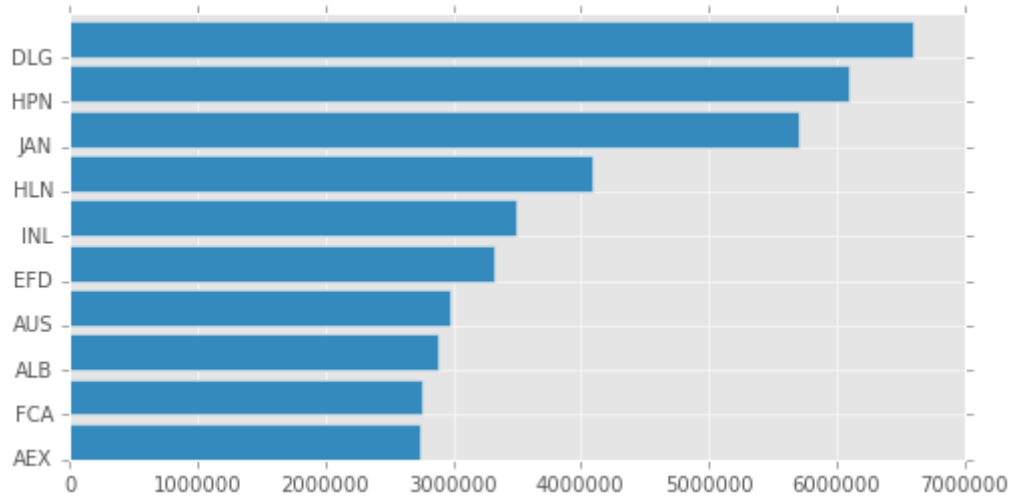
- `bigtabulate` (in order to split the big.matrix)
- `doMC` (to perform parallel calculations)

Let's now use bigtabulate to get this show on the road and start grouping our massive matrix!

In [15]:
```
%%R

# lets do some basics by graphing aggregated data
library(bigtabulate)
# split up the x data based on the unique airport it flew from
origin_indices <- bigsplit(x, 'Origin')
```

In [16]:
```
%%R -o counts
# recall that with the -o command, this variable is available in the python
counts <- sapply(origin_indices , function(i) length(i))
```

In [17]:
```
from matplotlib import pyplot as plt
import numpy as np

%matplotlib inline
plt.style.use('ggplot')

# just get the most frequent 10 airport origins
idxs = np.argsort(counts)[-10:]
counts_large = counts[idxs]

fig = plt.figure(figsize=(8,4))
plt.barh(range(len(counts_large)),counts_large)

# now we can use the previous array of values to index back to what the name
xticks_labels = np.array(list(unique_values['Origin']))[idxs]

# and set them on the plot
plt.yticks(range(len(xticks_labels)), xticks_labels)

plt.show()
```




## 3.2 Analyzing Age of Plane in R

Okay, so now let's go ahead and calculate the age of the different planes when they took their flights. We will estimate the first flight of the plane in the dataset as its 'birthmonth'.

In [22]:
```
%%R

library(bigtabulate)
# split up the x data based on the unique TailNum value
acindices <- bigsplit(x, 'TailNum')
# the acindices are a list of the different groups
```

In [23]:
```
%%R
# define a 'birthplace' function
# we use the very first recorded flight for a specific plane as an estimate

# let's first just define e function to get the
birthmonth <- function(y) {
    # assume that the input is one matric of values from one plane

    # get minimum year for this plane
    minYear <- min(y[,'Year'], na.rm=TRUE)
    # get a subset of the dataset for only this minimum year
    these <- which(y[,'Year']==minYear)
    # get the minimum month from the years
    minMonth <- min(y[these,'Month'], na.rm=TRUE)
    # now just return the number of months since 00 AD
    return(12*minYear + minMonth - 1)
}
```

In [24]:
```
%%R
# recall that sapply will send the different indices into the given functio
# here, we send in the indices of the plane, with the year and month
# this will be done for every plane based upon the acindices grouping
acStart <- sapply(acindices , function(i) birthmonth(x[i, c('Year','Month'),
```

In [21]:
```
%%R
#of course, we really wanted this to be a fast computation
# and we could split the operations using a foreach parallel loop

# setup the parallel for package
library(doMC)
registerDoMC(cores=3)
```

```
/Library/Python/2.7/site-packages/rpy2/robjects/functions.py:106: UserWar
ning: Loading required package: iterators

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/Library/Python/2.7/site-packages/rpy2/robjects/functions.py:106: UserWar
ning: Loading required package: parallel

  res = super(Function, self).__call__(*new_args, **new_kwargs)
```

In [25]:
```
%%R -o acStart

# now run it--in this example we only get e mild speedup because the computa
# fairly quick and there are a lot of groups to be made

# note that this package handles a bit more gracefully the attached memory r
# this works similarly to the parallel for loop from your book, but slightly
acStart <- foreach(i=acindices , .combine=c)%dopar% {
    return(birthmonth(x[i, c('Year', 'Month'), drop=FALSE]))
}
```

In [27]:
```
# We are back in python here!
# which plane is the youngest?
idx = np.argmax(acStart)

print 'The youngest plane is', list(unique_values['TailNum'])[idx],
print 'and flew starting in the year %.0f'%(acStart[idx]/12.0)
```

```
The youngest plane is N5WCAA and flew starting in the year 2009
```

In [28]:
```
%%R
# now lets save the age of th plane at the time it flew a given route
x[,"age"] <- x[,"Year"]*12+x[,"Month"]-acStart[x[,"TailNum"]]
```

```
Assignment will down cast from double to integer
Hint: To remove this warning type:  options(bigmemory.typecast.warning=FA
LSE)
```

In the above block of code, we filled in the values for 'age' in the big.matrix. This operation was possible because we used the `extraCols` argument when we created the backing file for the big.matrix. If we has not told the API that we planned to add this extra column, then the performance hit for creating a new column is HUGE. That's because we would need to completetly recreate the binary backing file on the hard drive. By telling the API we wnated it upfront, it went ahead and stored NaNs in the column when it created the binary file. That means we just needed to update the file incstead of write a complete new one.

As you can see, while it takes some time to get the files formatted correctly, they are easy to perform analysis on them using the technique of splitting, applying, and combining (essentially this is map-reduce in an embarassingly parallel context). Once the preprocessing is done it is also extremely easy (and fast) to load the memory mapped file onto into your R session. Like many things in Data Science, its the preprocessing that takes the most time.

## 3.3 Linear Regression of Massive Data in R

For our FInal R analysis, lets create a regression model based upon the age of the plane. We want to know if the age of the plane is a factor in determining if it will have a delay associated with it. We can achieve this in R with ease by using the `biganalytics` package. This package uses batch analysis (like we talked about and used in Data Mining) in order to use mini-batch (or stochastic) gradient descent upon the linear regression model.

In [29]:
```
%%R
library(biganalytics)

blm <- biglm.big.matrix( ArrDelay ~ age, data=x)
```

In [30]:
```
%%R
summary(blm)
```

```
Large data regression model: biglm(formula = formula, data = data, ...)
Sample size =  120947440
              Coef   (95%    CI)     SE p
(Intercept) 7.1203 7.1146 7.1260 0.0028 0
age         0.0045 0.0044 0.0046 0.0000 0
```

As your book states, there is a positive association of age and delays, but the model is ignoring all other variables, so the association might be very weak. It is actually quite amazing that we can just use much of this analytics engine out of the box and perform model training. In this scenario, we really do NOT NEED parallelism because the linear regression model must be sequentially built anyway. Thus, the our-of-core memory was the big problem. But now you have the tools to perform some really interesting analysis.

But of course, we are not done! There were some downsides to the R analysis. My biggest concerns were that:

- All the data in the matrix must be one type
  - ...took long preprocessing steps to get the CSV in the correct format
- Columns are not easily created or deleted from the file
- Loading and saving of the binary file was totally uncompressed
  - our binary data file ended up being bigger than the CSV file

Now, let's see what the alternative looks like in python. Keep in mind, while the python alternative does solve many of the above issues, it will not be quite as versatile as the above R coding architecture. Depending on your application, you might use Python, R, or both!

---

# 4.0 Analyzing the data using only Python

Now let's analyze the data using graphlab create (from the company Dato). As you have seen before, the graphlab-create API uses something called a scalable data frame (or SFrame) that handles all of the out-of-core memory management for you. It is by far one of the most optimized tools for handling table data out-of-core. Really. With that said, lets also give a list of what to expect:

- Loading and saving large amounts of data will be very fast
- Manipulating and adding columns is not much overhead, as everything is saved piece meal
- Parallelization is mostly handled for you in the background
- Operations are 'queued up' before execution, so that they can be simplified and parallelized (if possible and easy)
- However, there will be slightly less flexibility in the split-apply-combine technique
  - We need to use premade aggregation function from Dato's API
  - That means we can't just write a custom function to work in parallel on the different groups (like in R)
  - But we can concatenate operations to perform rich analysis (albeit not using standard python syntax)
  - In my opinion, this is the biggest downside (that custom aggregators cannot be built), but future version of graphlab might start to support this

- Grouping and applying cannot be separated from each other
  - This is because of the queueing of operations, grouping does not happen until absolutely neccessary
  - This also optimizes the memory management, which can be a huge time savings
- We CAN write custom apply functions that work on each row of data (just not the grouped splits)
- At the time of writing this, graphlab is only supported for python 2.7 (ouch!)
  - This is unfortunate because graphlab is the only reason I am not updating to python 3 ...
- Support on windows is really good, but mac/linux is slightly better.
  - This is especially true when using Dato's numpy extensions (which only works on mac/linux)
  - And the extensions make numpy operations completely usable out-of-core, with many operations optimized for sequential access (slightly more optimized than Numpy's builtin memmap)

## 4.1 Loading 12GB of Data in Python

Alright, then. Let's load up graphlab in order to get an idea about the power of this tool and what the syntax looks like.

In [31]:
```
import graphlab as gl
```

```
A newer version of GraphLab Create (v1.8.3) is available! Your current ve
rsion is v1.6.1.

You can use pip to upgrade the graphlab-create package. For more informat
ion see https://dato.com/products/create/upgrade. (https://dato.com/produ
cts/create/upgrade.)
```

In [32]:
```
sf = gl.SFrame('/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/AirlineI
```

```
[INFO] This non-commercial license of GraphLab Create is assigned to ecla
rson@smu.edu and will expire on November 20, 2016. For commercial licensi
ng options, visit https://dato.com/buy/. (https://dato.com/buy/.)

[INFO] Start server at: ipc:///tmp/graphlab_server-25879 - Server binary:
 /Library/Python/2.7/site-packages/graphlab/unity_server - Server log: /t
mp/graphlab_server_1457550445.log
[INFO] GraphLab Server Version: 1.6.1
```

```
PROGRESS: Finished parsing file /Users/eclarson/Dropbox/2U_QTW/Notebooks/
Data/AirlineDataAll.csv
PROGRESS: Parsing completed. Parsed 100 lines in 3.88496 secs.
------------------------------------------------------
Inferred types from first line of file as
column_type_hints=[int,int,int,int,int,int,int,int,int,int,int,int,int,st
r,int,int,int,int,int,str,str,int,int,int,str,str,str,str,str]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
PROGRESS: Read 541148 lines. Lines per second: 130969
PROGRESS: Read 2173116 lines. Lines per second: 196554
PROGRESS: Read 3811683 lines. Lines per second: 234181
PROGRESS: Read 5447032 lines. Lines per second: 244385
PROGRESS: Read 6526272 lines. Lines per second: 235956
PROGRESS: Read 7616835 lines. Lines per second: 226936
PROGRESS: Read 8707731 lines. Lines per second: 224239
PROGRESS: Read 10342982 lines. Lines per second: 222921
PROGRESS: Read 11421982 lines. Lines per second: 218713
PROGRESS: Read 13054670 lines. Lines per second: 225628
PROGRESS: Read 14689137 lines. Lines per second: 226708
PROGRESS: Read 15775671 lines. Lines per second: 225393
PROGRESS: Read 17397783 lines. Lines per second: 229081
PROGRESS: Read 18485787 lines. Lines per second: 226396
PROGRESS: Read 20120148 lines. Lines per second: 230760
PROGRESS: Read 21203108 lines. Lines per second: 229848
PROGRESS: Read 22828622 lines. Lines per second: 232983
PROGRESS: Read 24461038 lines. Lines per second: 234532
PROGRESS: Read 25550328 lines. Lines per second: 232368
PROGRESS: Read 27169062 lines. Lines per second: 234952
PROGRESS: Read 28256032 lines. Lines per second: 233188
PROGRESS: Read 29887885 lines. Lines per second: 233148
PROGRESS: Read 31511778 lines. Lines per second: 232137
PROGRESS: Read 33136496 lines. Lines per second: 232988
PROGRESS: Read 34767102 lines. Lines per second: 233546
PROGRESS: Read 36392631 lines. Lines per second: 236058
PROGRESS: Read 38018102 lines. Lines per second: 237374
PROGRESS: Read 40206514 lines. Lines per second: 241416
PROGRESS: Read 41844332 lines. Lines per second: 243771
PROGRESS: Read 44029607 lines. Lines per second: 247384
PROGRESS: Read 46214930 lines. Lines per second: 250681
PROGRESS: Read 47840837 lines. Lines per second: 251919
PROGRESS: Read 49480836 lines. Lines per second: 253095
PROGRESS: Read 51664904 lines. Lines per second: 255465
PROGRESS: Read 53290048 lines. Lines per second: 255087
```

```
PROGRESS: Read 54928042 lines. Lines per second: 254578
PROGRESS: Read 56564800 lines. Lines per second: 255770
PROGRESS: Read 58191404 lines. Lines per second: 257075
PROGRESS: Read 59822855 lines. Lines per second: 258040
PROGRESS: Read 62004894 lines. Lines per second: 259918
PROGRESS: Read 63633709 lines. Lines per second: 260899
PROGRESS: Read 65260703 lines. Lines per second: 261192
PROGRESS: Read 66891682 lines. Lines per second: 262163
PROGRESS: Read 68523240 lines. Lines per second: 262380
PROGRESS: Read 70140914 lines. Lines per second: 263458
PROGRESS: Read 72314727 lines. Lines per second: 264950
PROGRESS: Read 73945239 lines. Lines per second: 265366
PROGRESS: Read 75570243 lines. Lines per second: 265827
PROGRESS: Read 77736095 lines. Lines per second: 267367
PROGRESS: Read 79360647 lines. Lines per second: 267162
PROGRESS: Read 81512882 lines. Lines per second: 268704
PROGRESS: Read 83139516 lines. Lines per second: 269536
PROGRESS: Read 84813217 lines. Lines per second: 269686
PROGRESS: Read 86517396 lines. Lines per second: 269494
PROGRESS: Read 88214689 lines. Lines per second: 270041
PROGRESS: Read 89923815 lines. Lines per second: 269252
PROGRESS: Read 91633751 lines. Lines per second: 269140
PROGRESS: Read 93340532 lines. Lines per second: 269587
PROGRESS: Read 95034995 lines. Lines per second: 269335
PROGRESS: Read 96744114 lines. Lines per second: 268382
PROGRESS: Read 98451867 lines. Lines per second: 268821
PROGRESS: Read 100159724 lines. Lines per second: 268056
PROGRESS: Read 101287399 lines. Lines per second: 267217
PROGRESS: Read 102986784 lines. Lines per second: 267567
PROGRESS: Read 104694628 lines. Lines per second: 267379
PROGRESS: Read 106401064 lines. Lines per second: 266921
PROGRESS: Read 108097454 lines. Lines per second: 267162
PROGRESS: Read 109791760 lines. Lines per second: 267357
PROGRESS: Read 111495903 lines. Lines per second: 267603
PROGRESS: Read 112634842 lines. Lines per second: 266984
PROGRESS: Read 114339392 lines. Lines per second: 267188
PROGRESS: Read 116029489 lines. Lines per second: 267273
PROGRESS: Read 117678096 lines. Lines per second: 267336
PROGRESS: Read 119314108 lines. Lines per second: 267849
PROGRESS: Read 121492728 lines. Lines per second: 268808
PROGRESS: Read 122571182 lines. Lines per second: 250558
PROGRESS: Finished parsing file /Users/eclarson/Dropbox/2U_QTW/Notebooks/
Data/AirlineDataAll.csv
PROGRESS: Parsing completed. Parsed 123534969 lines in 491.683 secs.
```

Wow! That was really fast to read the entire csv structure! On my system, it took about 6 minutes to prepare the SFrame. But--is it really the same length as the data we saw above? How could this be 5 times faster than the R code for loading and parsing the data? Let's check the dimensions.

In [33]:
```
sf.shape
```

Out[33]:
```
(123534969, 29)
```

...Yes. It is the same size! And if we dump this to file, it will be compressed to under 2GB and be written in about a minute to disk (and almost instantaneously when read--just like R). That's a really

great advantage when working with large data like this. The space/time tradeoffs have really been optimized for this package.

It also means that file access is quicker because the DiskIO has to load fewer bytes. It needs to decompress the bytes, but it does so typically much quicker than it would be to read them from disk. If you are not impressed right now, then I am not sure you ever will be (that's a joke, relax)!

If you wanted to save it, you could say:

- `sf.save('Data/airline_data_sframe_directory') # write out as binary compressed file (very compressed)`
- Then to load it back up:
  - `gl.load_sframe('Data/airline_data_sframe_directory')`

---

## 4.2 Preprocessing the CSV data in Python with Graphlab

So great, we can load up the same file as R did. It still took forever to preprocess that file and get it ready. So, we really want to answer this question: **Is there an easier way to preprocess and concatenate all the files into one memory map using SFrames?**

SFrames are more forgiving in terms of the data types that they can hold. Moreover, they load much more quickly than the parser used by `pandas`. Maybe we should try to read in each of the original CSV files as SFrames and concatenate them together into one SFrame. Would this be a quicker way to create the memory mapped file than all the preprocessing we did above?

The answer is, in fact, **yes**. It is much faster and the code is much easier to understand than what we performed earlier. And because `graphlab` supports more than one data type, we do not need to preprocess any of the data. Plus, when we write out to file, it happens more quickly and in a compressed version (on disk it will be compressed to about 2GB).

Here is the code to perform all the concatenation we did earlier. I am also supplying graphlab with the data type for each column to ensure that the data is consistent. However, if I did not, Graphlab would try and guess the data type based on the first 100 lines of the csv file.

In [ ]:
```
del sf # get rid of the old thing
# What about just loading up all the data using the SFrame Utility for loadi
# We will need to make sure that the SFrame has consistent datatypes, so we
column_hints=[int,int,int,int,int,int,int,int,str,int,str,int,int,int,int,in

t = time.time()
# now load the first SFrame
sf = gl.SFrame() #.read_csv('Data/1987.csv',column_type_hints=column_hints)

# and then append each SFrame in a for loop
for year in range(1987,2009):
    print 'read %d lines, reading next file %d.csv'%(sf.shape[0],year)
    sys.stdout.flush()
    sftmp = gl.SFrame.read_csv('/Users/mmcgee/Dropbox/2016 Summer Courses/MS
    sf = sf.append(sftmp)

print 'It took %.2f seconds to concatenate the memory mapped file'%(time.tim

t = time.time()
print 'Saving...',
sf.save('Data/sframe_directory') # save a compressed version of this SFrame
print 'took %.2f seconds'%(time.time()-t),'Shape of SFrame is',sf.shape
```

So yeah, we concatenated and loaded the file in ~320 seconds and saved a compressed binary version in about 1 minute (on my machine). That's quite a speedup.

But can we perform operations upon it? Let's check how versatile these data structures are. Let's first try to replicate the functionality of finding the most popular airports to fly out of, like we did with `bigmemory`.

In [34]:
```
# CONVENIENCE BLOCK FOR THOSE LOADING FROM DISK HERE

# If you have already run the notebook above and just want to load up the da
# then you can reload the SFrame here
import graphlab as gl

sf = gl.load_sframe('/Users/mmcgee/Dropbox/2016 Summer Courses/MSDS 7333/Air
```

## 4.3 Analyzing popular Airports to fly from in Python

We now need to perform some operations using the split-apply-combine technique. However, in graphlab this all happens via one syntax (the `groupby` function). Let's see how it works.

In [37]:
```
# to perform grouping and splitting
# we need to specify (1) which column(s) to group the SFrame using, and
#                    (2) what function we want to perform on the group
# in graphlab, we only have a few options for performing on each of the grou
# Here, lets keep it simple--let's group by the airport origin and then
#  use the builtin 'count' function to aggregate the results
# The result is another SFrame with the Unique origin names as a column and
#  number of entries in each group in another column
%time sf_counts = sf.groupby('Origin', {'num_flights':gl.aggregate.COUNT()})
sf_counts
```

```
CPU times: user 44.5 ms, sys: 56.1 ms, total: 101 ms
Wall time: 1min 18s
```

Out[37]:

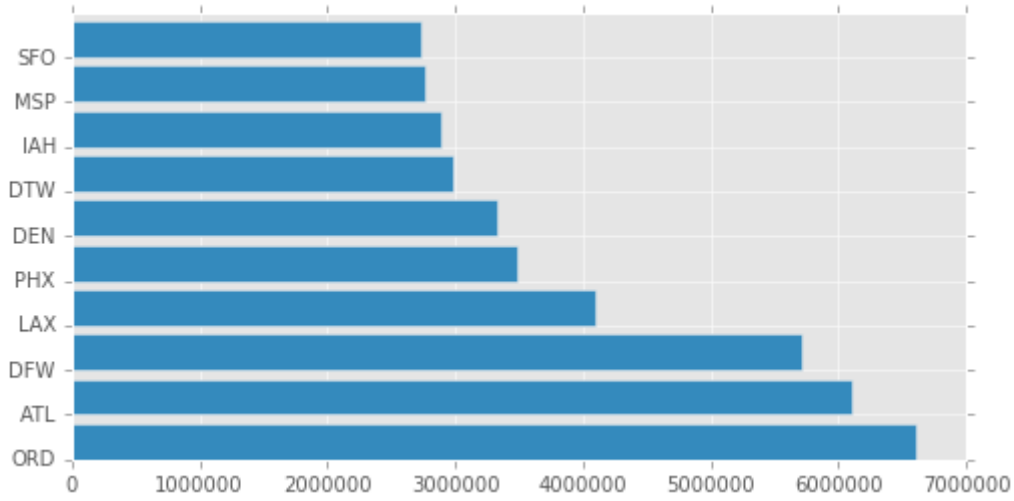
| Origin | num_flights |
|---|---|
| TYR | 20739 |
| TOL | 58747 |
| ABY | 8035 |
| SHV | 129284 |
| HSV | 158743 |
| FAT | 136998 |
| ATL | 6100953 |
| TEX | 683 |
| CWA | 7881 |
| EAU | 2167 |

[347 rows x 2 columns]

Note: Only the head of the SFrame is printed.

You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

In [38]:
```
from matplotlib import pyplot as plt
import numpy as np

%matplotlib inline
plt.style.use('ggplot')

# As seen above, the sf_counts SFrame has the origin of the flight on the le
# and the count of flights on the right

# let's grab the top 10 entries
sf_top = sf_counts.topk('num_flights',10) # this is builtin command in graph

airports = np.array(sf_top['Origin'])
counts = np.array(sf_top['num_flights'])

fig = plt.figure(figsize=(8,4))
plt.barh(range(len(counts)),counts)

# and set them on the plot
plt.yticks(range(len(airports)), airports)

plt.show()
```




Which is exactly what we got before using the big.matrix package! To me, this was a much simpler implementation, but we still need to see what the limitations are.

---

## 4.4 Analyzing Depature Delays at Specific Times of the Day in Python

Now, let's try to do something a little more interesting, such as trying to perform the same operations that are in the Nolan text. Specifically, let's try to find the percentiles for late flights based upon the hour of the day that they depart. To do this we will:

- Create a new column for the hour of the day that a plane departed.
- Fix the `hours==24` and `hours==0` to be consistent
- Group by the hours of departure
- Take percentiles of each group to see which ones are the latest

In [39]:
```
from math import floor
# first, let's create a new column in this SFrame that has the departure tir
sf['DepTimeByHour'] = sf['CRSDepTime'].apply(lambda x: floor(x/100),dtype=ir
sf['DepTimeByHour'] # and print a few of them (note: the column has not beer
```

Out[39]:
```
dtype: int
Rows: 123534969
[7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 1
5, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 1
6, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 7, 8, 8, 8, 8, 8, ... ]
```

In [40]:
```
# Let's now change the hours of the day that are equal to 24
# we need to be careful here becasue each column is not immutable
# in pandas this would be:
#    df.DepTimeByHour[df.DepTimeByHour==24] = 0
# but we cant just change a few values in the column, we need to change ther
# don't worry though, Graphlab does this smartly
sf['DepTimeByHour'] = sf['DepTimeByHour'].apply(lambda x: 0 if x==24 else x)
# again, this column has not been evaluated yet because that value has not y
```

In [42]:
```
# now lets group the SFrame by the hours and calculate the percentiles of ea
# here is where the lazy evaluation actually happens so this takes a little

# the groupby function will partition our SFrame into groups based upon the
# next, we need to tell graphlab what operations to perform on the group and
# to do that, we send in a dictionary of names and 'operations'
# We did a similar operation above with the 'COUNT' aggregator
# there are only a certain number of operators we can choose from, we will c
#   aggregator on the column 'DepDelay'. We want to take the percentiles [0.
# We can also perform other operations by adding entries in the dictonary
# So we will also take the 'MAX' of each group
import time
t = time.time()
delay_sf = sf.groupby('DepTimeByHour',
                           {'delay_quantiles':gl.aggregate.QUANTILE('DepDel
                            'delay_max':gl.aggregate.MAX('DepDelay')})
# this returns a new SFrame with the specified columns from each aggregation

print 'Took %.2f seconds to run'%(time.time()-t)
```

```
Took 191.67 seconds to run
```

In [43]:
```
# sort it by when departed and display it
delay_sf = delay_sf.sort('DepTimeByHour')
delay_sf
```

Out[43]:

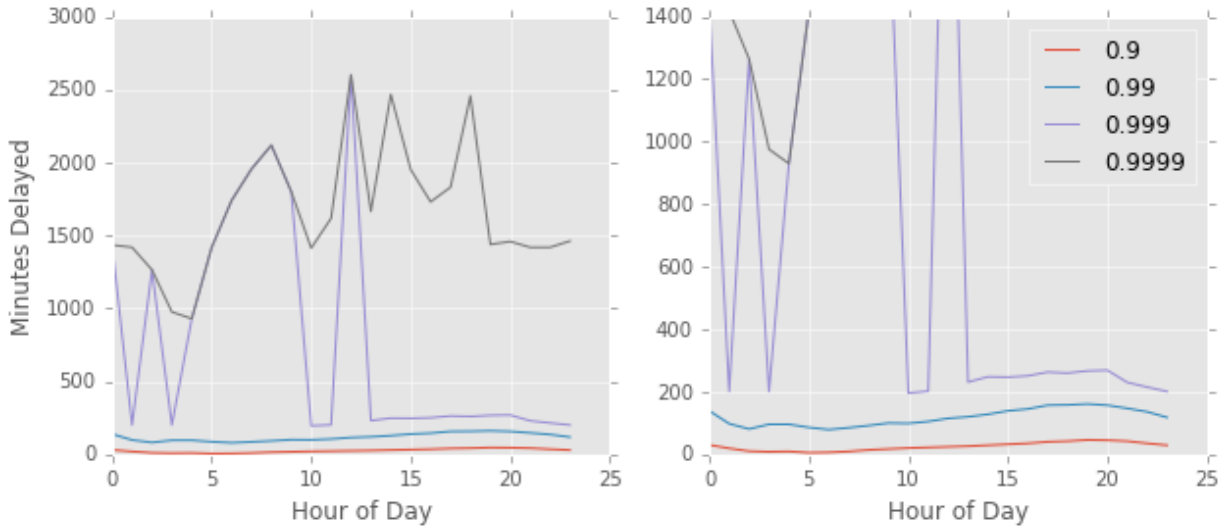
| DepTimeByHour | delay_max | delay_quantiles |
|---|---|---|
| 0 | 1435 | [30.0, 139.0, 1435.0, 1435.0] ... |
| 1 | 1419 | [19.0, 98.0, 201.0, 1419.0] ... |
| 2 | 1265 | [10.0, 81.0, 1265.0, 1265.0] ... |
| 3 | 976 | [8.0, 96.0, 201.0, 976.0] |
| 4 | 930 | [9.0, 96.0, 930.0, 930.0] |
| 5 | 1418 | [5.0, 86.0, 1418.0, 1418.0] ... |
| 6 | 1740 | [6.0, 79.0, 1740.0, 1740.0] ... |
| 7 | 1956 | [9.0, 85.0, 1956.0, 1956.0] ... |
| 8 | 2119 | [14.0, 92.0, 2119.0, 2119.0] ... |
| 9 | 1800 | [17.0, 100.0, 1800.0, 1800.0] ... |

[24 rows x 3 columns]

Note: Only the head of the SFrame is printed.

You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

In [44]:
```
# to use matplotlib, we need to convert over to numpy arrays
# this is a fine operation because the new aggregated SFrame we are
# working (delay_sf) with is quite small
x = np.array(delay_sf['DepTimeByHour'])
y = np.array(delay_sf['delay_quantiles'])

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(x,y)
plt.ylabel('Minutes Delayed')
plt.xlabel('Hour of Day')


plt.subplot(1,2,2)
plt.plot(x,y)
plt.xlabel('Hour of Day')
plt.ylim(0,1400) # make the same axes as in the book
plt.legend(['0.9','0.99','0.999','0.9999'])

plt.show()
```




Oh no! This doesn't look like the graph from the book at all!! Actually, there is a really great reason for that...

Let's investigate further. The bottom two lines (percentiles 0.90 and 0.99) look very similar to what your book had, but the other two percentile values (0.999 and 0.9999) seemingly did not evaluate properly.

Find out why by looking at the documentation for the quartile aggregator.

- https://dato.com/products/create/docs/graphlab.data_structures.aggregation.html#module-graphlab.aggregate (https://dato.com/products/create/docs/graphlab.data_structures.aggregation.html#module-graphlab.aggregate)
- Hint: there is a reason that the quantile calculation was really, really, really fast.

---

## 4.5 Calculate Plane Age with Dato

So let's keep moving and try to calculate the plane's age like we did in R.

In [45]:
```
# only use years where the tail number was recorded
# we can manipulate the SFrame fairly easily in graphlab, so let's do it
sf_tmp = sf[['TailNum','Year','Month','DepDelay']][sf['Year']>1994]
```

In [46]:
```
# lets try to make a function for getting the age of the plane
# First lets just save the plane's age in years
sf_tmp['FlightAge'] = 12*sf_tmp['Year']+sf_tmp['Month']-1

# and take the minimum of that in order to get its first flight
t = time.time()
sf_min_ages = sf_tmp[['TailNum','FlightAge']].groupby('TailNum',{'FirstFligh
print 'Took %.2f seconds to run'%(time.time()-t)
```

```
Took 102.38 seconds to run
```

In [47]:
```
# Now transform the FirstFlight Column into the original dataframe size
# to do that we can just do a join on a few columns of our sf
# this will save the flight age and the minimum in a new SFrame
%time sf_fewcols = sf_tmp[['TailNum','FlightAge']].join(sf_min_ages,on='Tail
```

```
CPU times: user 193 ms, sys: 300 ms, total: 494 ms
Wall time: 7min 39s
```

In [48]:
```
# and now we can simply subtract the new calculated quantity and add to the
sf_tmp['Age'] = sf_fewcols['FlightAge']-sf_fewcols['FirstFlight']
```

## 4.6 Calculate a linear model from massive data with Python

Graphlab (much like `biganalytics`) will perform mini-batch linear regression and it will do it fast. Let's see what kind of output we get.

In [49]:
```
# now look at the age and delay time in terms of regression (like your book,
%time lin_model = gl.linear_regression.create(sf_tmp['DepDelay','Age'].dropr
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This
may take a while.
          You can set ``validation_set=None`` to disable validation track
ing.

PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 80183731
PROGRESS: Number of features          : 1
PROGRESS: Number of unpacked features : 1
PROGRESS: Number of coefficients    : 2
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+---
------------------+---------------+-----------------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error | Va
lidation-max_error | Training-rmse | Validation-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+---
------------------+---------------+-----------------+
PROGRESS: | 1         | 2        | 9.125101     | 2591.597051        | 19
47.347461          | 30.926499     | 30.826641       |
PROGRESS: +-----------+----------+--------------+--------------------+---
------------------+---------------+-----------------+
CPU times: user 271 ms, sys: 128 ms, total: 399 ms
Wall time: 2min 20s
```

In [50]:
```
lin_model['coefficients']
```

Out[50]:

| name | index | value |
|---|---|---|
| (intercept) | None | 8.59440814545 |
| Age | None | 0.00528458023677 |

[2 rows x 3 columns]

Which again, gives us a positive association of Age and departure delay time (but could be very weak because we model nothing else). The value is of the same magnitude as what we found using R. It is slightly different because we need to use gradient descent (not batch methods, which are deterministic). The number of iterations and starting point for gradient methods will slightly (or not so slightly sometimes) affect the final coefficient values.

---

# 5.0 Conclusions

Now that we have seen SFrames and bigmemory, which one should you choose? **As always, it depends.** I would recommend performing preprocessing with SFrames, almost always. They are much easier to work with. But their use of split-apply-combine is still lacking behind the abilities of R's bigmemory. Some operations are optimized, and if you can get away with only using those operations, then try just using SFrames. Since SFrames are great for adding and subtracting columns, that is another way to play around with the data before trying to use R.

However, if you need a more flexible solution, bigmemory will need to be what you go with. And it is a really wonderful program... That's especially true when the aggregation function you use needs to do something rather complex. If you are running windows... SFrames are your goto. But, really, why are you running windows? Oh right, for SAS... Time to install a linux partition maybe... Or buy a mac and run bootcamp... You get the idea.

Another thing to keep in mind: Graphlab is actively being developed (and most of it is open source, like SFrames). It will one day have an R interface. It will probably get custom aggregation functions one day soon.

The solutions I have provided here are just examples. Large datasets can be handled using databases really well (but eventually you run into limitations).

And, as always, happy analyzing!

---

*This Notebook was created and is managed by Professor Eric Larson. Please direct comments and questions to eclarson@smu.edu.*

In [ ]:

In [ ]:

In [ ]: