Question #1 - Code to collect SNMP data

My sample code is shown in the cell below - the solution repeatedly runs the **snmpget** commandline tool with different arguments. The output is collected using the Python **subprocess** module, which handles I/O redirection for jobs. By default, this code will not gather the data unless you set the "gatherData" variable to true.

Another solution would be to use a **bash** script to call **snmpget** to collect the data; however, I find the data processing (multi-dimensional arrays) much harder to do in shell script.

In [2]:
```
gatherData = False
```

In [4]:

```
import subprocess
import pprint
import time

pp = pprint.PrettyPrinter(indent=4, depth=6)

MINUTES_BETWEEN_SAMPLES = 10
NUMBER_RAW_SAMPLES=20
TARGET="128.138.207.5:7777"
IF = [1, 2, 3, 6, 7, 9, 10, 11 ]
MIBS = { "inOctets" : ".1.3.6.1.2.1.2.2.1.10",
         "inUcastPkts" : ".1.3.6.1.2.1.2.2.1.11",
         "inNUcastPkts" : ".1.3.6.1.2.1.2.2.1.12",
         "outOctets" : ".1.3.6.1.2.1.2.2.1.16",
         "outUCastPkts" : ".1.3.6.1.2.1.2.2.1.17",
         "outNUCastPkts" : ".1.3.6.1.2.1.2.2.1.18"
         }

def getMib(target, mib, interface):
    """
    Get a MIB value via snmpget
    """
    cmd = [ "/usr/bin/snmpget",
            "-v", "2c",
            "-c", "public",
            target,
            "%s.%d" %( MIBS[mib], interface)
            ]
    output = subprocess.check_output(cmd)
    output = output.split()
    return int(output[-1])

def delta(low,high):
    "Compute difference and account for overflow"
    diff = high - low
    if diff < 0:
        diff += (2**32)
    return diff

def dump(filename, data):
    f = open(filename, "w")
    headers = ['sample', 'if'] + [mib for mib in MIBS]
    f.write( ','.join(headers) + '\n' )
    for sample in range(len(data)):
        for ifs in IF:
            line = [ str(sample), str(ifs) ] + [ str(data[sample][ifs][
            f.write( ','.join(line) + '\n' )
    f.close()

if gatherData:
    RAWSAMPLES = {}
    for sample in range(NUMBER_RAW_SAMPLES):
        print "Collect sample", sample
        IFDATA = {}
```

```
        for ifs in IF:
            MIBDATA={}
            for mib in MIBS:
                value = getMib(TARGET, mib, ifs)
                MIBDATA[mib] = value
            IFDATA[ifs] = MIBDATA
        RAWSAMPLES[sample] = IFDATA
        time.sleep(MINUTES_BETWEEN_SAMPLES * 60)

    dump("snmp-raw.csv", RAWSAMPLES)

    SAMPLES = {}
    for sample in range(len(RAWSAMPLES)-1):
        IFDATA = {}
        for ifs in IF:
            MIBDATA = {}
            for mib in MIBS:
                value = delta( RAWSAMPLES[sample][ifs][mib],
                               RAWSAMPLES[sample+1][ifs][mib] )
                MIBDATA[mib] = value
            IFDATA[ifs] = MIBDATA
        SAMPLES[sample] = IFDATA

    dump("snmp-delta.csv", SAMPLES)
else:
    print("Not collecting and dumping data")
```

```
Not collecting and dumping data
```

Question #2 - Analyzing the data

I'm going to do basic summarization of the data using the Python Pandas library, which is similar to the R data frame

In [55]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [8]:
```
raw = pd.read_csv("ref-snmp-raw.csv")
delta = pd.read_csv("ref-snmp-delta.csv")
```

In [11]:
```
# Lets dump out the first few rows of the delta data to see what's ther
delta[:4]
```

Out[11]:

	sample	if	outUCastPkts	inOctets	outOctets	inUcastPkts	outNUCastPkts	inNU
0	0	1	0	0	0	0	0	0
1	0	2	672	184683	90820	2226	0	0
2	0	3	96	116273	14370	866	0	0
3	0	6	119	97039	16290	751	0	0

Mean Packet Size

In [39]:
```
# Now, let's compute the mean packet size by dividing
# the octests (both in and out) by the number of packets.
#
def sumOfCol(data, iface, val):
    return np.sum(data[data['if'] == iface][val].values)

print "inOctets is ", sumOfCol(delta, 2, 'inOctets'), \
    "outOctets is ", sumOfCol(delta, 2, 'outOctets'), \
    "combined is ", sumOfCol(delta, 2, ['inOctets','outOctets'])

def avgPktSize(data, iface):
    return sumOfCol(delta, iface, ['inOctets','outOctets']) / \
        sumOfCol(delta, iface, ['outUCastPkts','inUcastPkts', 'outNUCas

print "Average packet size for interface #2 is", avgPktSize(delta, 2),
print "Average packet size across all interfaces is", \
    [(iface, avgPktSize(delta, iface)) for iface in IF]
```

```
inOctets is  2776493 outOctets is  1169827 combined is  3946320
Average packet size for interface #2 is 101 bytes
Average packet size across all interfaces is [(1, 0), (2, 101), (3,
146), (6, 143), (7, 89), (9, 0), (10, 92), (11, 92)]
```

So the output shows the mean packet size per interface. Note that we're taking the average number of bytes and dividing by the average number of packets to get the average packet size.

[(1, 0), (2, 101), (3, 146), (6, 143), (7, 89), (9, 0), (10, 92), (11, 92)]

We can plot this, but to do so, it's easiest to use Pandas fetures to compute the number of bytes and packets per observation and then compute the average packet size.

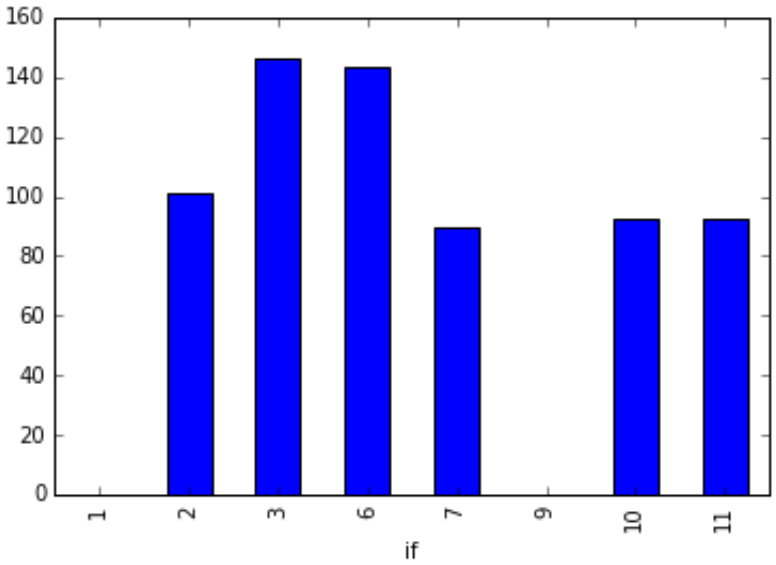
In [94]:
```
delta['bytes'] = delta['inOctets'] + delta['outOctets']
delta['pkts'] = delta['outUCastPkts'] + delta['inUcastPkts'] + delta['o
delta[:5] # show first 5 rows
```

Out[94]:

	sample	if	outUCastPkts	inOctets	outOctets	inUcastPkts	outNUCastPkts	inNU
0	0	1	0	0	0	0	0	0
1	0	2	672	184683	90820	2226	0	0
2	0	3	96	116273	14370	866	0	0
3	0	6	119	97039	16290	751	0	0
4	0	7	678	149297	92040	2201	0	0

In [62]:
```
# Now, plot out the average packet size by interface
byIf = delta.groupby('if').aggregate(sum)
(byIf['bytes'] / byIf['pkts']).plot('bar')
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x11455fc90>



Are packet sizes similar?

There's two ways to answer this - one is to look at the graph above and see that links 3 and 6 appear to have similar averages and that 2, 7, 10 and 11 also have similar averages. From this, we would say that not all links have similarly sizes packets.

Another way to answer this is to look at the correlation coefficient - if there's a high correlation between two links, then not only are the average packet sizes similar, but they appear in a similar order as well. Let's compute the pair-wise correlation coefficient:

In [71]:
```
nzIF = [2, 3, 6, 7, 10, 11]
pd.DataFrame( np.corrcoef([ delta['bytes'][delta['if'] == iface] \
                            for iface in nzIF ] ),
              columns = nzIF )
```

Out[71]:

	2	3	6	7	10	11
0	1.000000	0.198431	0.191663	0.997174	0.226503	0.226744
1	0.198431	1.000000	0.997520	0.202815	0.171412	0.171759
2	0.191663	0.997520	1.000000	0.198648	0.146870	0.147246
3	0.997174	0.202815	0.198648	1.000000	0.184177	0.184430
4	0.226503	0.171412	0.146870	0.184177	1.000000	0.999995
5	0.226744	0.171759	0.147246	0.184430	0.999995	1.000000

From this, we see that

- links 2 and 7 have a strong correlation (0.99),
- links 3 and 6 have a strong correlation (0.99),
- links 10 and 11 have a strong correlation (0.99)

The correlation refines the answer derived from the averages by looking at the time series of the data as well as the overall averages -- it's a more nuanced measure.

Either method (averages or correlation) are considered correct for this problem.

Which link is most "bursty"?

For this we'll use the coefficient of variation (cov) of the packet size or the mean divided by the standard deviation of the packet size. This is almost overkill because the means are so close (and thus the standard deviations would be directly comparable), but it's easy enough to do.

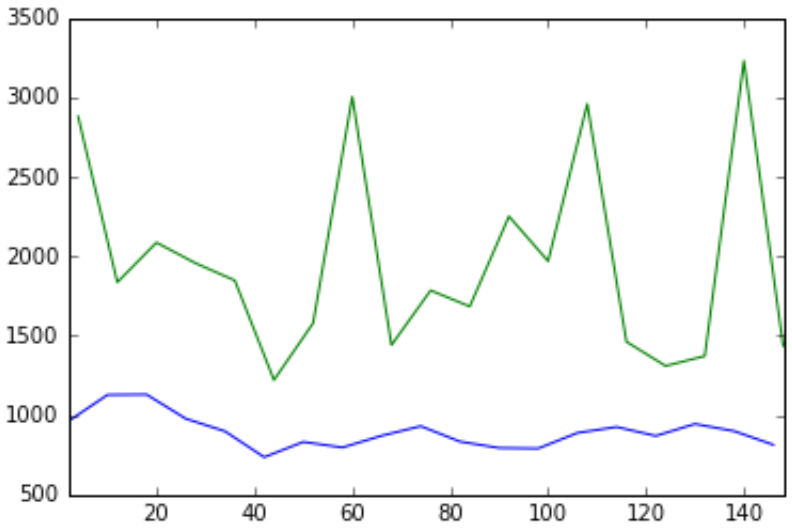
In [91]:
```
def cov(data): return np.std(data) / np.mean(data)
print "Comparing link packets \n", \
  pd.DataFrame([ (iface,  \
                  np.mean(delta['pkts'][delta['if']==iface]), \
                  np.std(delta['pkts'][delta['if']==iface]), \
                  cov(delta['pkts'][delta['if']==iface]) \
                 ) for iface in nzIF ] ,\
                columns = ['iface', 'mean', 'std. dev', 'cov'])
```

```
Comparing link packets
   iface         mean    std. dev       cov
0      2  2055.473684  599.235247  0.291531
1      3   896.157895  101.839535  0.113640
2      6   803.157895   99.506288  0.123894
3      7  1963.473684  608.031368  0.309671
4     10  1041.736842  306.782083  0.294491
5     11  1041.736842  307.042744  0.294741
```

By these measures, we'd say that ***link #7 is the "most bursty"***, or the link that has the highest coefficient of variation. Lets plot link #3 (low cov) and #7 to see if the COV matches our intuitive notion of "less bursty" and "more bursty"

In [93]:
```
delta['pkts'][delta['if'] == 3].plot()
delta['pkts'][delta['if'] == 7].plot()
```

Out[93]: `<matplotlib.axes._subplots.AxesSubplot at 0x11484df50>`



The lower line shows the packets for link #3 (a low average number of packets and also low variation) while the upper plot shows link #7 (large average, large cov). This confirms that our metric (COV) is a good choice and helps us select a "bursty" link.

Given the similarity of the means for this data, it would also be appropriate to use the variance or standard deviation alone to assess "burstiness".

In [ ]: