# Homework 1. Which of two things is larger?

**Due: Monday 17th May 2015 00:00**

---

Useful libraries for this homework:

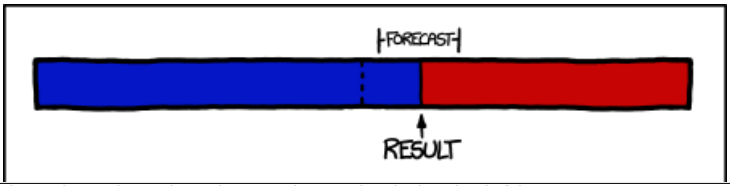
- numpy, for arrays
- pandas, for data frames
- matplotlib, for plotting
- requests, for downloading web content
- pattern, for parsing html and xml pages
- fnmatch (optional), for Unix-style string matching

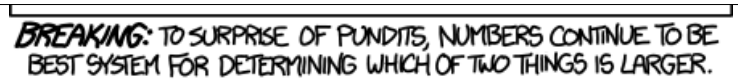
In [1]:

```
# special IPython command to prepare the notebook for matplotlib
%matplotlib inline

from fnmatch import fnmatch

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
from pattern import web

# set some nicer defaults for matplotlib
from matplotlib import rcParams

#these colors come from colorbrewer2.org. Each is an RGB triplet
dark2_colors = [(0.10588235294117647, 0.6196078431372549, 0.4666666666666667),
                (0.8509803921568627, 0.37254901960784315, 0.00784313725490196),
                (0.4588235294117647, 0.4392156862745098, 0.7019607843137254),
                (0.9058823529411765, 0.1607843137254902, 0.5411764705882353),
                (0.4, 0.6509803921568628, 0.11764705882352941),
                (0.9019607843137255, 0.6705882352941176, 0.007843137254901961),
                (0.6509803921568628, 0.4627450980392157, 0.11372549019607843),
                (0.4, 0.4, 0.4)]

rcParams['figure.figsize'] = (10, 6)
rcParams['figure.dpi'] = 150
rcParams['axes.color_cycle'] = dark2_colors
rcParams['lines.linewidth'] = 2
rcParams['axes.grid'] = True
rcParams['axes.facecolor'] = '#eeeeee'
rcParams['font.size'] = 14
rcParams['patch.edgecolor'] = 'none'
```

## Introduction

This was the XKCD comic after the 2012 Presidential election:






The comic refers to the fact that Nate Silver's statistical model (which is based mostly on combining information from pre-election polls) correctly predicted the outcome of the 2012 presidential race in all 50 states.

Polling data isn't a perfect predictor for the future, and some polls are more accurate than others. This means that election forecastors must consider prediction uncertainty when building models.

In this first assignment, you will perform a simple analysis of polling data about the upcoming Governor races. The assignment has three main parts:

1. **First** you will build some tools to download historical polling data from the web, and parse it into a more convenient format.
2. **Next** you will use these tools to aggregate and visualize several past Governor races
3. **Finally** you will run a *bootstrap analysis* to estimate the probable outcome of current Governor races, given the level of precision of historical polls. No need to worry about the details of the bootstrap right now, we will come to that when we discuss Resampling methods.

---

# Part 1: Collect and Clean

The Real Clear Politics (RCP) website archives many political polls. In addition, they combine related polls to form an "RCP average" estimate of public opinion over time. For example, the chart on this page shows historical polling data for the Obama-Romney presidential race. The chart is an average of the polling data table below the chart.

The data used to generate plots like this are stored as XML pages, with URLs like:

http://charts.realclearpolitics.com/charts/[id].xml

Here, `[id]` is a unique integer, found at the end of the URL of the page that displays the graph. The `[id]` for the Obama-Romney race is 1171:

http://charts.realclearpolitics.com/charts/1171.xml

Opening this page in Google Chrome or Firefox will show you the XML content in an easy-to-read format. Notice that XML tags are nested inside each other, hierarchically (the jargony term for this is the "Document Object Model", or "DOM"). The first step of webscraping is almost always exploring the HTML/XML source in a browser, and getting a sense of this hierarchy.

---

## Problem 0

The above XML page includes 5 distinct tags (one, for example, is `chart`). List these tags, and depict how they nest inside each other using an indented list. For example:

- Page
  - Section
    - Paragraph
  - Conclusion

*Your Answer Here*

- chart
  - series
    - value
  - graphs
    - graph

- value
- graph
  - value

---

## Problem 1

We want to download and work with poll data like this. Like most programming tasks, we will break this into many smaller, easier pieces

Fill in the code for the `get_poll_xml` function, that finds and downloads an XML page discussed above

**Hint**

`requests.get("http://www.google.com").text` downloads the text from Google's homepage

In [2]:

```
"""
Function
--------
get_poll_xml

Given a poll_id, return the XML data as a text string

Inputs
------
poll_id : int
    The ID of the poll to fetch

Returns
-------
xml : str
    The text of the XML page for that poll_id

Example
-------
>>> get_poll_xml(1044)
u'<?xml version="1.0" encoding="UTF-8"?><chart><series><value xid=\'0\'
>1/27/2009</value>
...etc...
"""
#your code here
def get_poll_xml(poll_id ):
     return requests.get("http://charts.realclearpolitics.com/charts/{i
d}.xml".format(id = poll_id ) ).text
```

Here are some other functions we'll use later. `plot_colors` contains hints about parsing XML data.

In [3]:

```
# "r"egular "e"xpressions is kind of a mini-language to
# do pattern matching on text
import re

def _strip(s):
    """This function removes non-letter characters from a word

    for example _strip('Hi there!') == 'Hi there'
    """
    return re.sub(r'[\W_]+', '', s)

def plot_colors(xml):
    """
    Given an XML document like the link above, returns a python dictionary
    that maps a graph title to a graph color.

    Both the title and color are parsed from attributes of the <graph> tag:
    <graph title="the title", color="#ff0000"> -> {'the title': '#ff0000'}

    These colors are in "hex string" format. This page explains them:
    http://coding.smashingmagazine.com/2012/10/04/the-code-side-of-color/

    Example
    -------
    >>> plot_colors(get_poll_xml(1044))
    {u'Approve': u'#000000', u'Disapprove': u'#FF0000'}
    """
    dom = web.Element(xml)
    result = {}
    for graph in dom.by_tag('graph'):
        title = _strip(graph.attributes['title'])
        result[title] = graph.attributes['color']
    return result
```

---

## Problem 2

Even though `get_poll_xml` pulls data from the web into Python, it does so as a block of text. This still isn't very useful. Use the `web` module in `pattern` to parse this text, and extract data into a pandas DataFrame.

**Hints**

- You might want create python lists for each column in the XML. Then, to turn these lists into a DataFrame, run:

  ```
  pd.DataFrame({'column_label_1': list_1, 'column_label_2':list_2, ...})
  ```

- use the pandas function `pd.to_datetime` to convert strings into dates

In [4]:

```
"""
    Function
    ---------
    rcp_poll_data

    Extract poll information from an XML string, and convert to a DataF
rame

    Parameters
    ----------
    xml : str
        A string, containing the XML data from a page like
        get_poll_xml(1044)

    Returns
    -------
    A pandas DataFrame with the following columns:
        date: The date for each entry
        title_n: The data value for the gid=n graph (take the column na
me from the `title` tag)

    This DataFrame should be sorted by date

    Example
    -------
    Consider the following simple xml page:

    <chart>
        <series>
            <value xid="0">1/27/2009</value>
            <value xid="1">1/28/2009</value>
        </series>
        <graphs>
            <graph gid="1" color="#000000" balloon_color="#000000" titl
e="Approve">
                <value xid="0">63.3</value>
                <value xid="1">63.3</value>
            </graph>
            <graph gid="2" color="#FF0000" balloon_color="#FF0000" titl
e="Disapprove">
                <value xid="0">20.0</value>
                <value xid="1">20.0</value>
            </graph>
        </graphs>
    </chart>

    Given this string, rcp_poll_data should return
    result = pd.DataFrame({'date': pd.to_datetime(['1/27/2009', '1/28/2
009']),
                           'Approve': [63.3, 63.3], 'Disapprove': [20.0
, 20.0]})
"""
#your code here

def rcp_poll_data(xml):
    result = pd.DataFrame()
    dom = web.Element(xml)
    result['date'] = pd.to_datetime([dates.content for dates in dom.by_
tag('series')[0]])

    for graph in dom.by_tag('graph'):
        title_n = graph.attributes[u'title']
        result[title_n] = [np.nan if value.content is u'' else  float(v
alue.content) for value in graph]
    return result
```

The output from `rcp_poll_data` is much more useful for analysis. For example, we can plot with it:

In [5]:

```
def poll_plot(poll_id):
    """
    Make a plot of an RCP Poll over time

    Parameters
    ----------
    poll_id : int
        An RCP poll identifier
    """

    # hey, you wrote two of these functions. Thanks for that!
    xml = get_poll_xml(poll_id)
    data = rcp_poll_data(xml)
    colors = plot_colors(xml)

    #remove characters like apostrophes
    data = data.rename(columns = {c: _strip(c) for c in data.columns})

    #normalize poll numbers so they add to 100%
    norm = data[colors.keys()].sum(axis=1) / 100
    for c in colors.keys():
        data[c] /= norm

    for label, color in colors.items():
        plt.plot(data.date, data[label], color=color, label=label)


    plt.xticks(rotation=70)
    plt.legend(loc='best')
    plt.xlabel("Date")
    plt.ylabel("Normalized Poll Percentage")
```

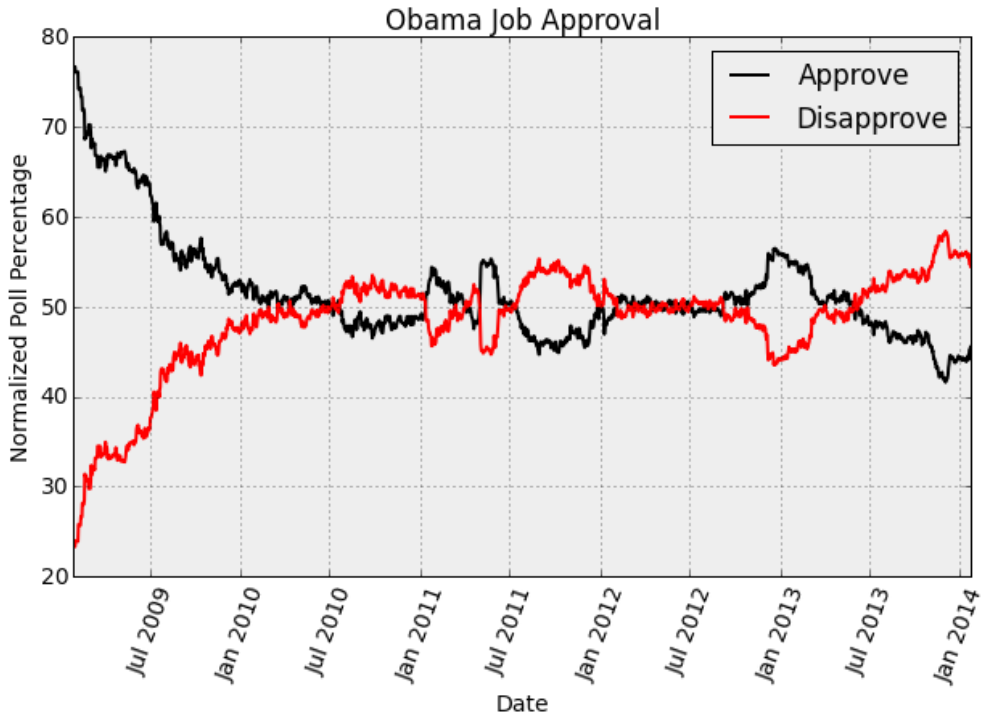
If you've done everything right so far, the following code should reproduce the graph on this page

In [6]:

```
poll_plot(1044)
plt.title("Obama Job Approval")
```

Out[6]:

```
<matplotlib.text.Text at 0x7f2b4169e410>
```



# Part 2: Aggregate and Visualize

## Problem 3

Unfortunately, these data don't have any error bars. If a candidate leads by 10% in the RCP average, is he or she certain to win? Or is this number too close to call? Does a 10% poll lead mean more a day before a race than it does a week before? Without error estimates, these questions are impossible to answer.

To get a sense of how accurate the RCP polls are, you will gather data from many previous Governor races, where the outcome is known.

This url has links to many governer races.

http://www.realclearpolitics.com/epolls/2010/governor/2010_elections_governor_map.html

Notice that each link to a governor race has the following URL pattern:

http://www.realclearpolitics.com/epolls/[YEAR]/governor/[STATE]/[TITLE]-[ID].html

Write a function that scans html for links to URLs like this

**Hint** The fnmatch function is useful for simple string matching tasks.

In [7]:

```
"""
    Function
    --------
    find_governor_races

    Find and return links to RCP races on a page like
    http://www.realclearpolitics.com/epolls/2010/governor/2010_election
s_governor_map.html

    Parameters
    ----------
    html : str
        The HTML content of a page to scan

    Returns
    -------
    A list of urls for Governer race pages

    Example
    -------
    For a page like

    <html>
    <body>
    <a href="http://www.realclearpolitics.com/epolls/2010/governor/ma/m
assachusetts_governor_baker_vs_patrick_vs_cahill-1154.html"></a>
    <a href="http://www.realclearpolitics.com/epolls/2010/governor/ca/c
alifornia_governor_whitman_vs_brown-1113.html"></a>
    </body>
    </html>

    find_governor_races would return
    ['http://www.realclearpolitics.com/epolls/2010/governor/ma/massachu
setts_governor_baker_vs_patrick_vs_cahill-1154.html',
     'http://www.realclearpolitics.com/epolls/2010/governor/ca/californ
ia_governor_whitman_vs_brown-1113.html']
"""
#your code here

from fnmatch import filter

def find_governor_races(html):
    return filter(web.find_urls(html),
                  'http://www.realclearpolitics.com/epolls/*/go
vernor/*/*-*.html')
```

## Problem 4

At this point, you have functions to find a collection of governor races, download historical polling data from each one, parse them into a numerical DataFrame, and plot this data.

The main question we have about these data are how accurately they predict election outcomes. To answer this question, we need to grab the election outcome data.

Write a function that looks up and returns the election result on a page like this one.

**Remember to look at the HTML source!**

You can do this by selection `view->developer->view source` in Chrome, or `Tools -> web developer -> page source` in Firefox. Altenatively, you can right-click on a part of the page, and select "inspect element"

In [8]:

```
"""
    Function
    --------
    race_result

    Return the actual voting results on a race page

    Parameters
    ----------
    url : string
        The website to search through

    Returns
    -------
    A dictionary whose keys are candidate names,
    and whose values is the percentage of votes they received.

    If necessary, normalize these numbers so that they add up to 100%.

    Example
    --------
    >>> url = 'http://www.realclearpolitics.com/epolls/2010/governor/ca
/california_governor_whitman_vs_brown-1113.html'
    >>> race_result(url)
    {'Brown': 56.0126582278481, 'Whitman': 43.9873417721519}
"""
#your code here

def race_result(url):
    dom = web.Element(requests.get(url).text)
    candivotes = {}
    candits =[i.content.split()[0] for i in  dom.by_tag('table.data')[0
].by_tag('th')[3:-1]]
    values = [float(i.content) for i in dom.by_tag('table.data')[0].by_
tag('tr.final')[0][3:-1]]
    total =  sum(values)
    values = [v / total * 100. for v in values]
#     for i in range(len(candits)):
#         candivotes[dom.by_tag('th')[i+3].content.split()[0]] = float(d
om.by_tag('tr.final')[0][i+3].content) / total * 100.
    for i in range(len(candits)):
        candivotes[candits[i]] = values[i]

    return candivotes
race_result('http://www.realclearpolitics.com/epolls/2009/governor/nj/n
ew_jersey_governor_corzine_vs_christie-1051.html')
```

Out[8]:

```
{u'Christie': 49.24318869828456,
 u'Corzine': 44.904137235116046,
 u'Daggett': 5.852674066599395}
```

Here are some more utility functions that take advantage of what you've done so far.

In [9]:

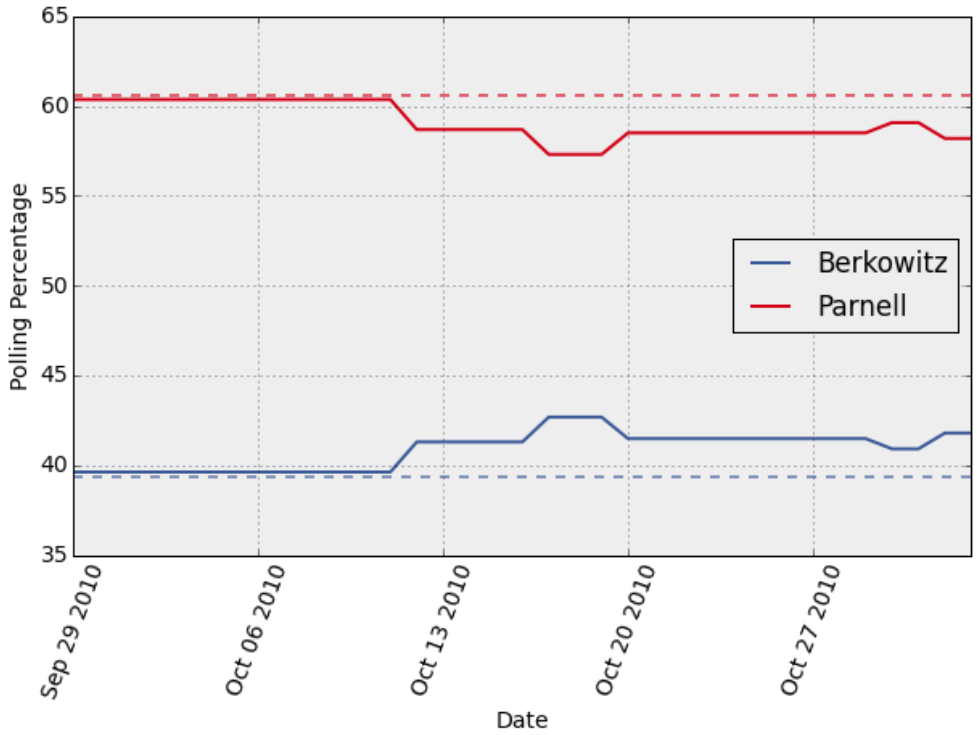
```
def id_from_url(url):
    """Given a URL, look up the RCP identifier number"""
    return url.split('-')[-1].split('.html')[0]


def plot_race(url):
    """Make a plot summarizing a senate race

    Overplots the actual race results as dashed horizontal lines
    """
    #hey, thanks again for these functions!
    id = id_from_url(url)
    xml = get_poll_xml(id)
    colors = plot_colors(xml)

    if len(colors) == 0:
        return

    #really, you shouldn't have
    result = race_result(url)

    poll_plot(id)
    plt.xlabel("Date")
    plt.ylabel("Polling Percentage")
    for r in result:
        plt.axhline(result[r], color=colors[_strip(r)], alpha=0.6, ls='
--')
```

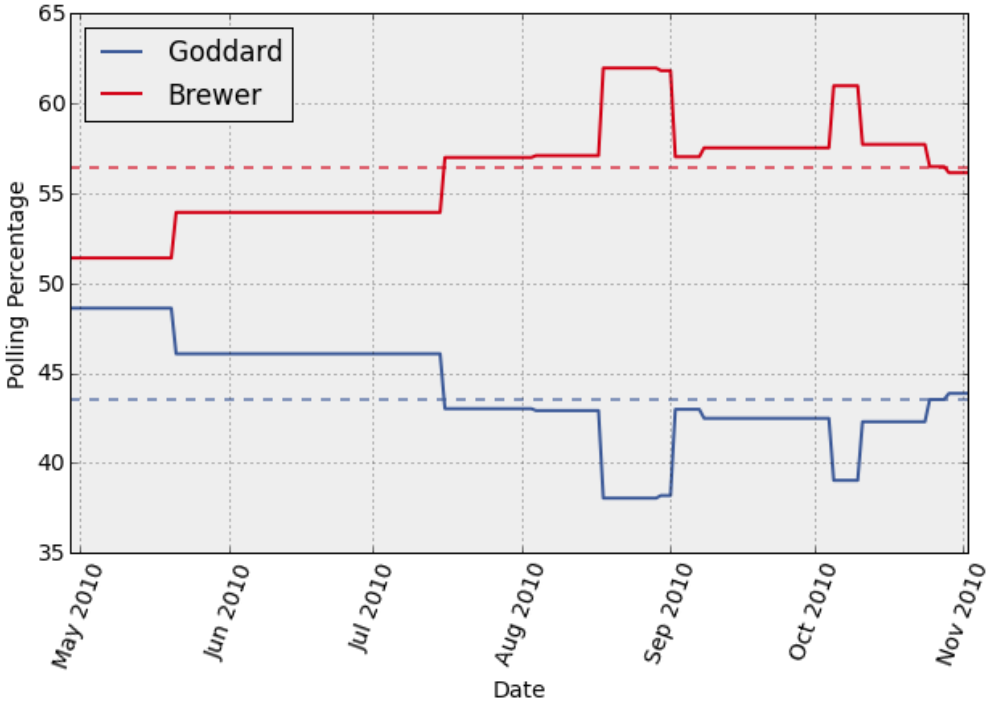
Now that this is done, we can easily visualize many historical Governer races. The solid line plots the poll history, the dotted line reports the actual result.

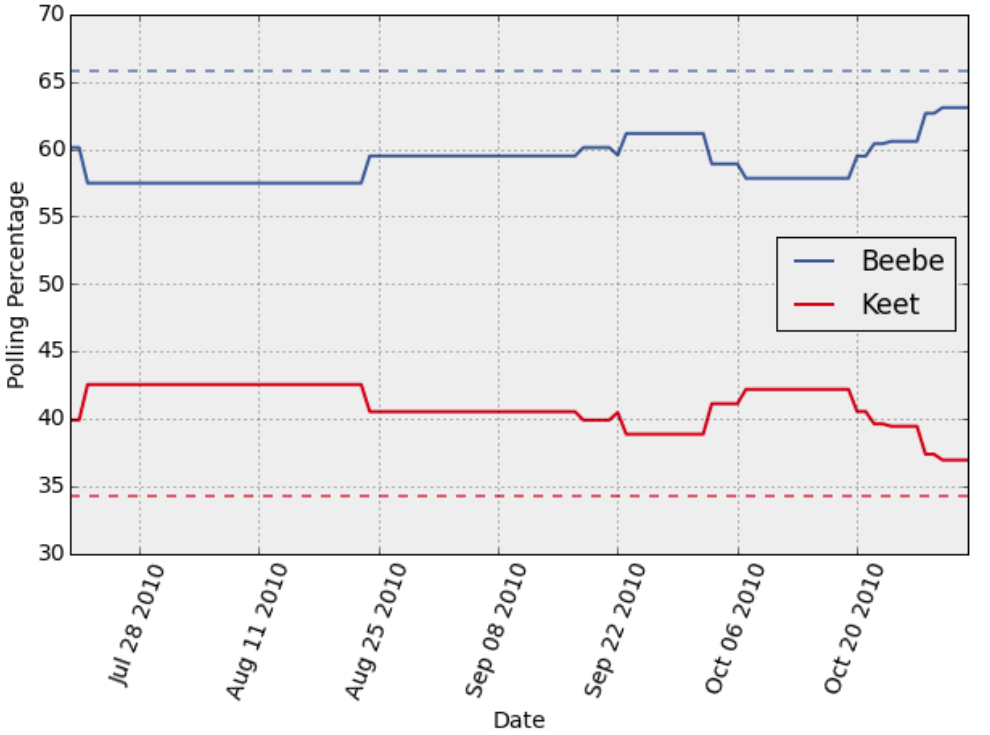
If this code block fails, you probably have a bug in one of your functions.

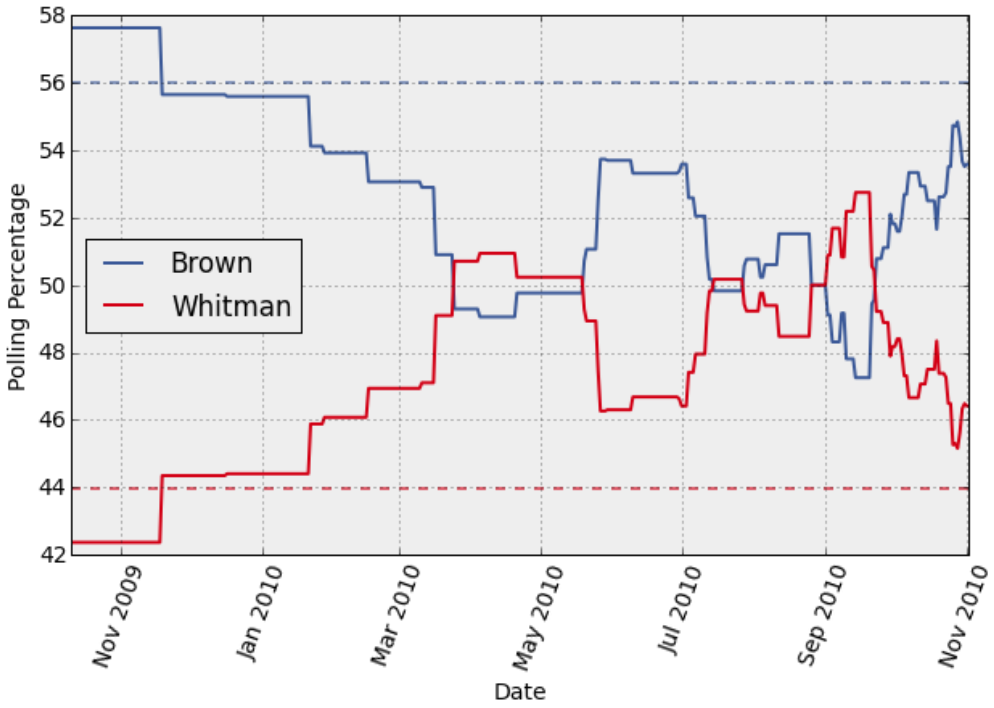
In [10]:

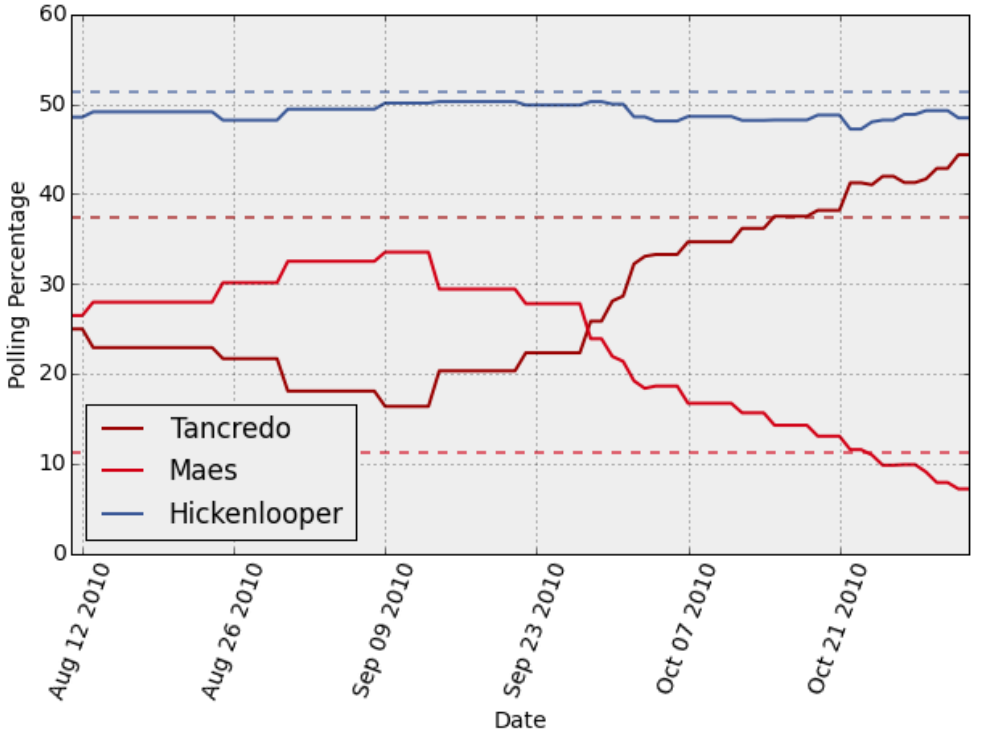
```
page = requests.get('http://www.realclearpolitics.com/epolls/2010/gover
nor/2010_elections_governor_map.html').text.encode('ascii', 'ignore')

for race in find_governor_races(page):
    plot_race(race)
    plt.show()
```

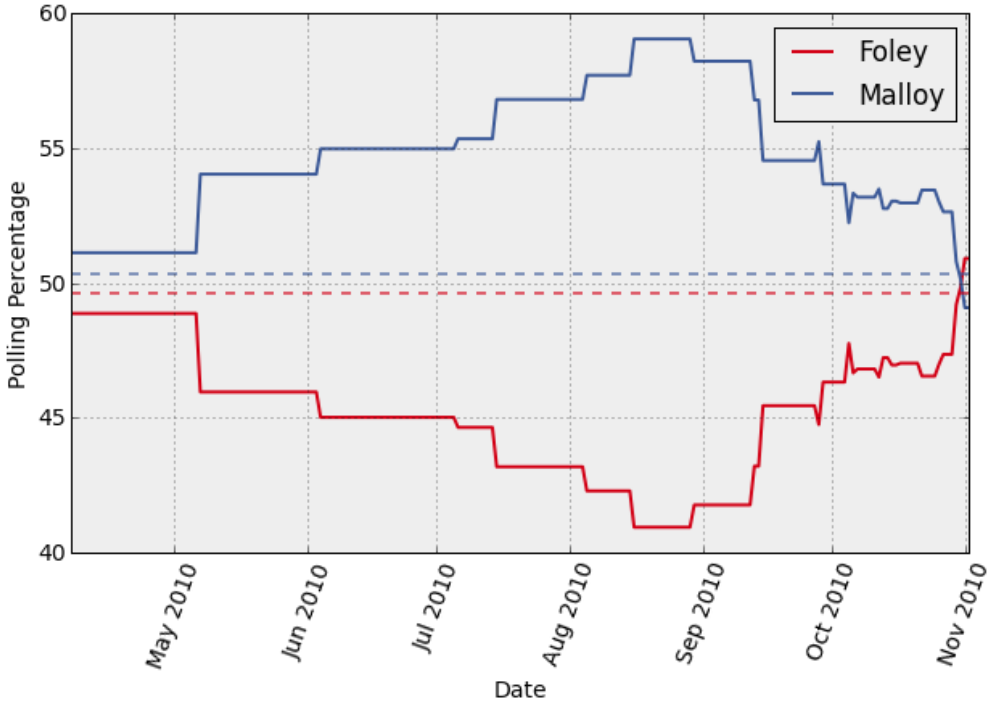


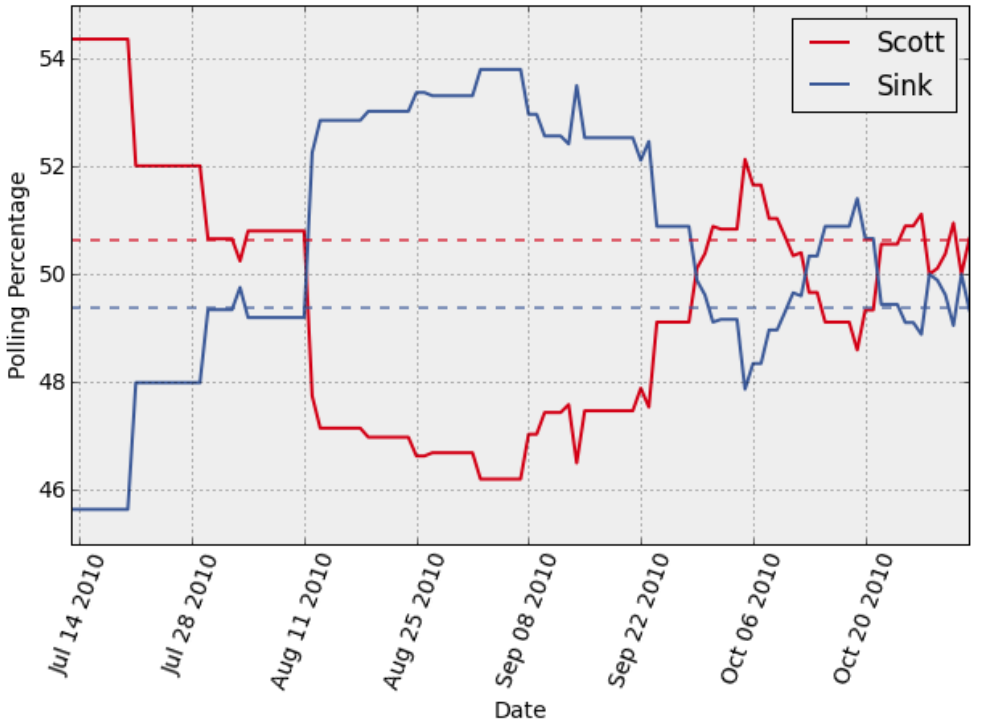

Goddard
Brewer
Polling Percentage
65
60
55
50
45
40
35
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010
Date

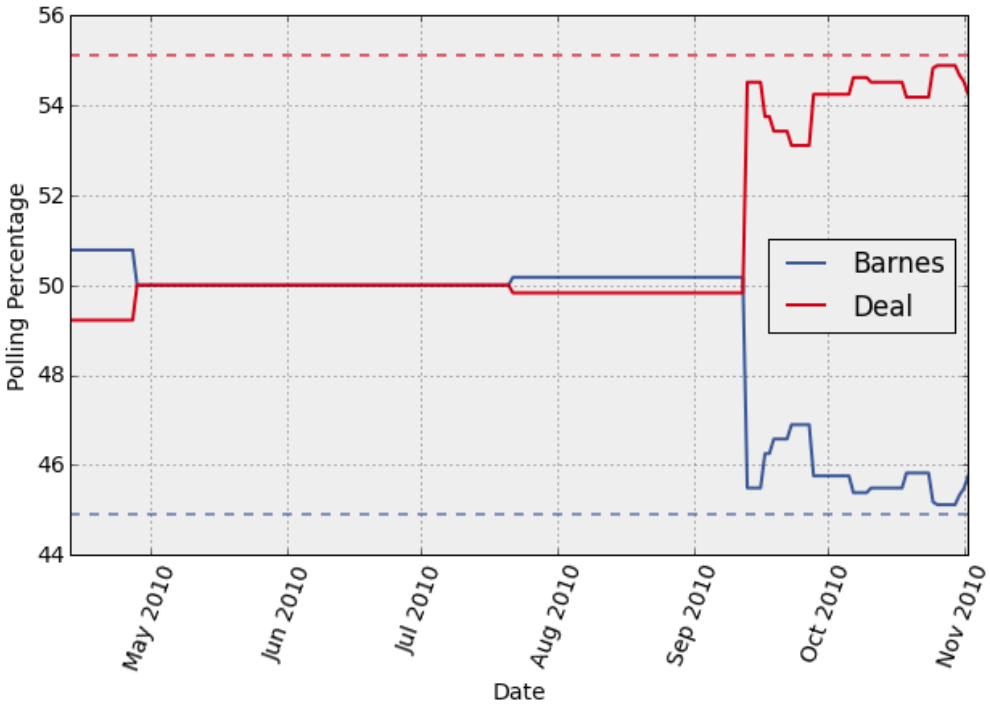

Beebe
Keet
Polling Percentage
70
65
60
55
50
45
40
35
30
Jul 28 2010
Aug 11 2010
Aug 25 2010
Sep 08 2010
Sep 22 2010
Oct 06 2010
Oct 20 2010
Date

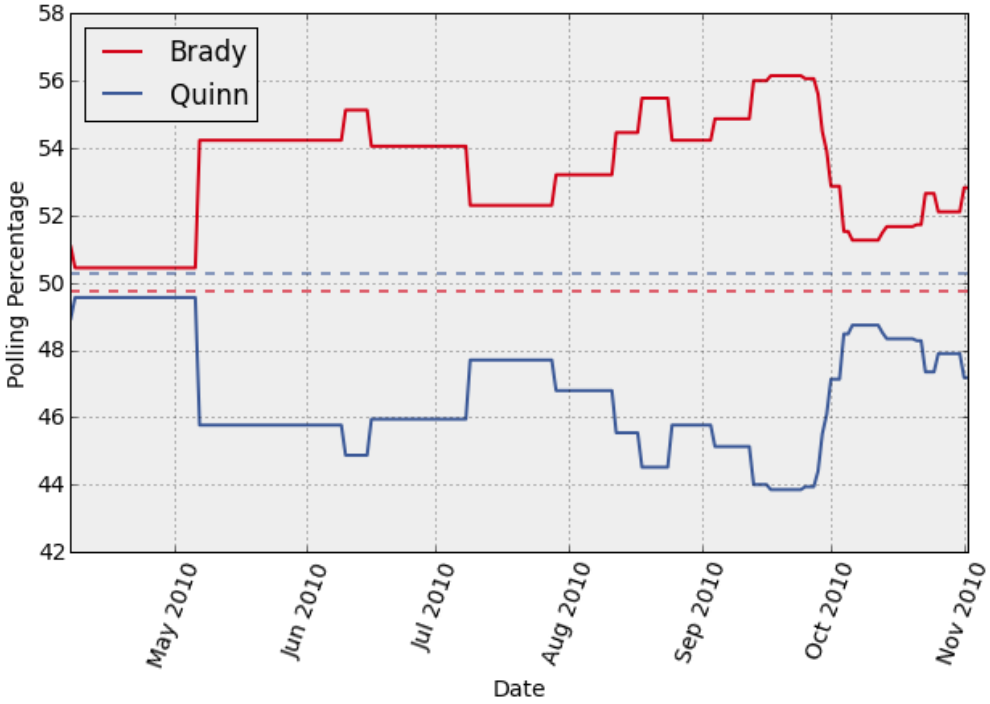

Polling Percentage
Date
Brown
Whitman
58
56
54
52
50
48
46
44
42
Nov 2009
Jan 2010
Mar 2010
May 2010
Jul 2010
Sep 2010
Nov 2010

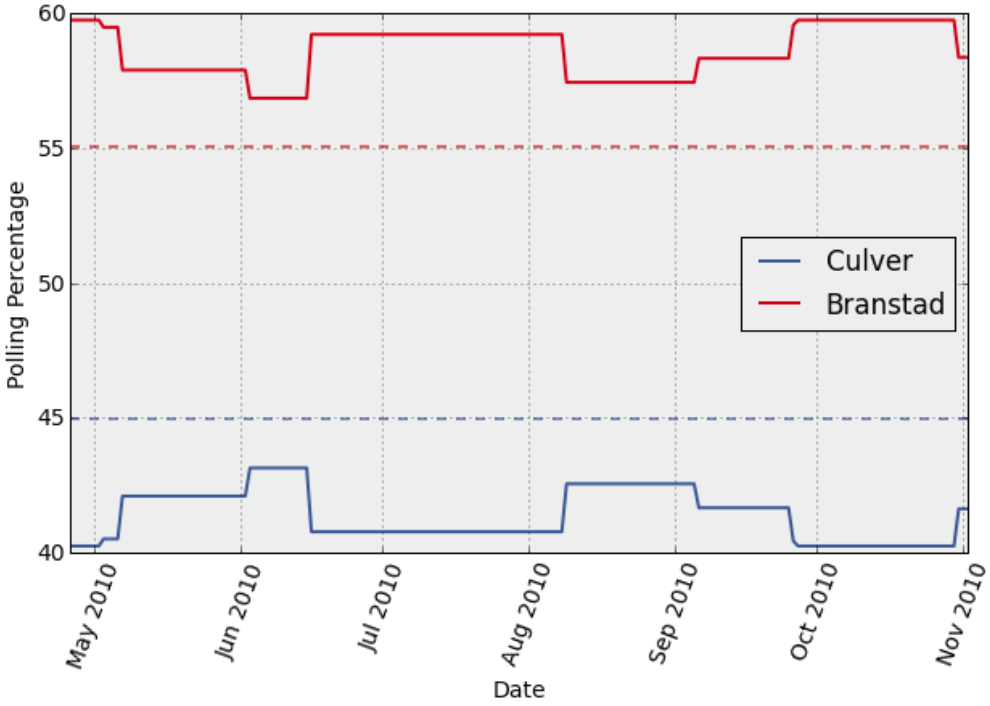

Polling Percentage
Date
Tancredo
Maes
Hickenlooper
60
50
40
30
20
10
0
Aug 12 2010
Aug 26 2010
Sep 09 2010
Sep 23 2010
Oct 07 2010
Oct 21 2010

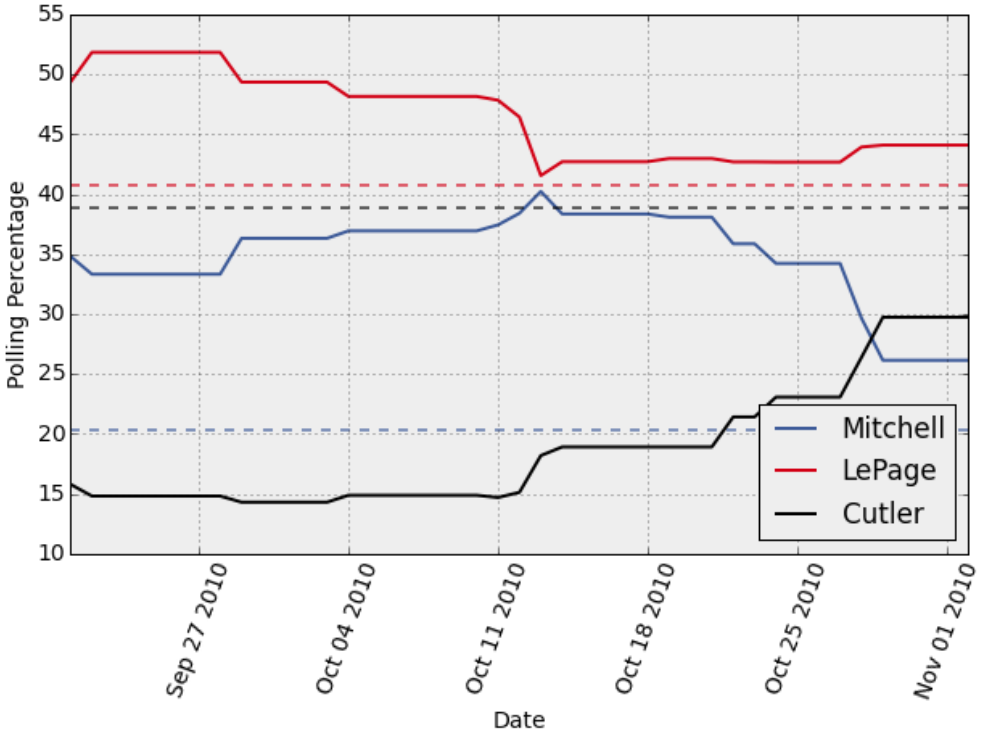

Foley
Malloy
Polling Percentage
Date
60
55
50
45
40
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

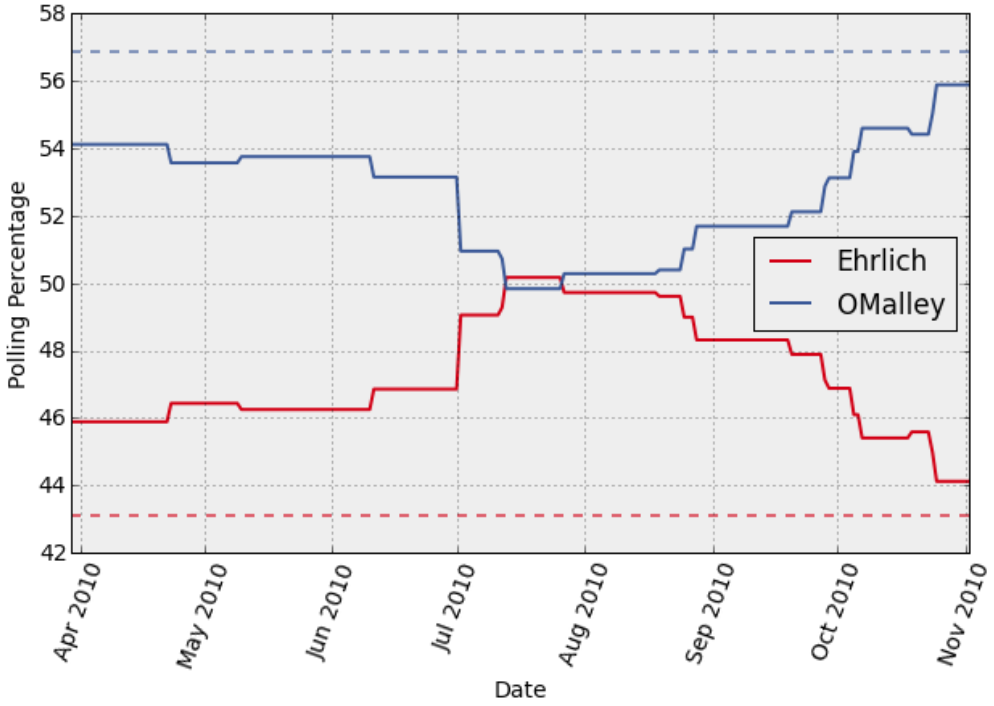

Scott
Sink
Polling Percentage
Date
54
52
50
48
46
Jul 14 2010
Jul 28 2010
Aug 11 2010
Aug 25 2010
Sep 08 2010
Sep 22 2010
Oct 06 2010
Oct 20 2010

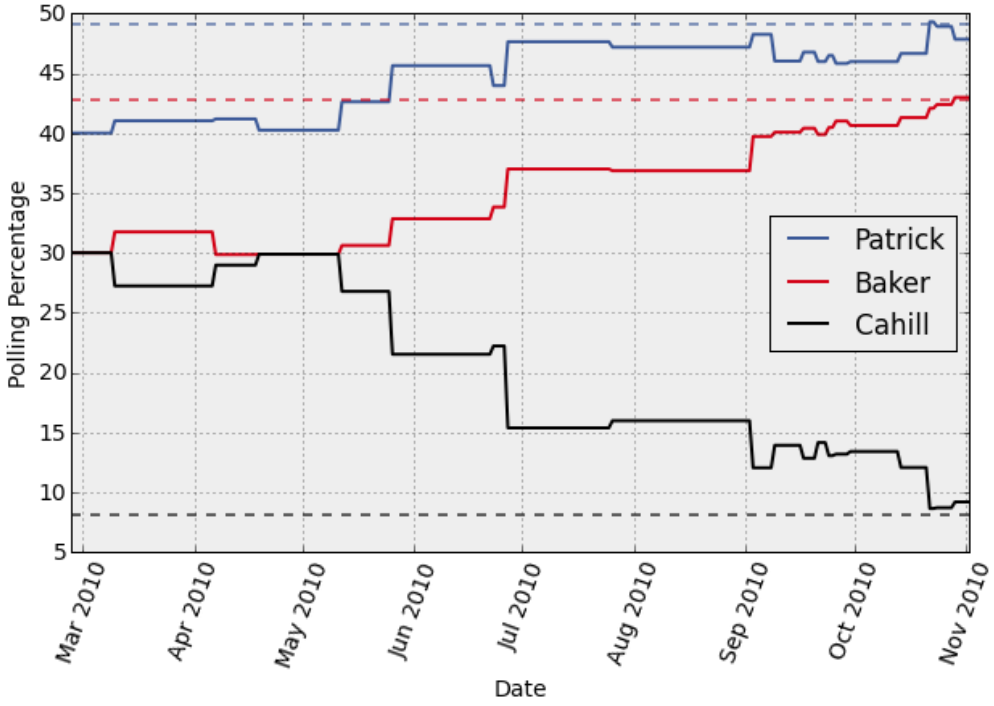

Barnes
Deal
Polling Percentage
Date
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

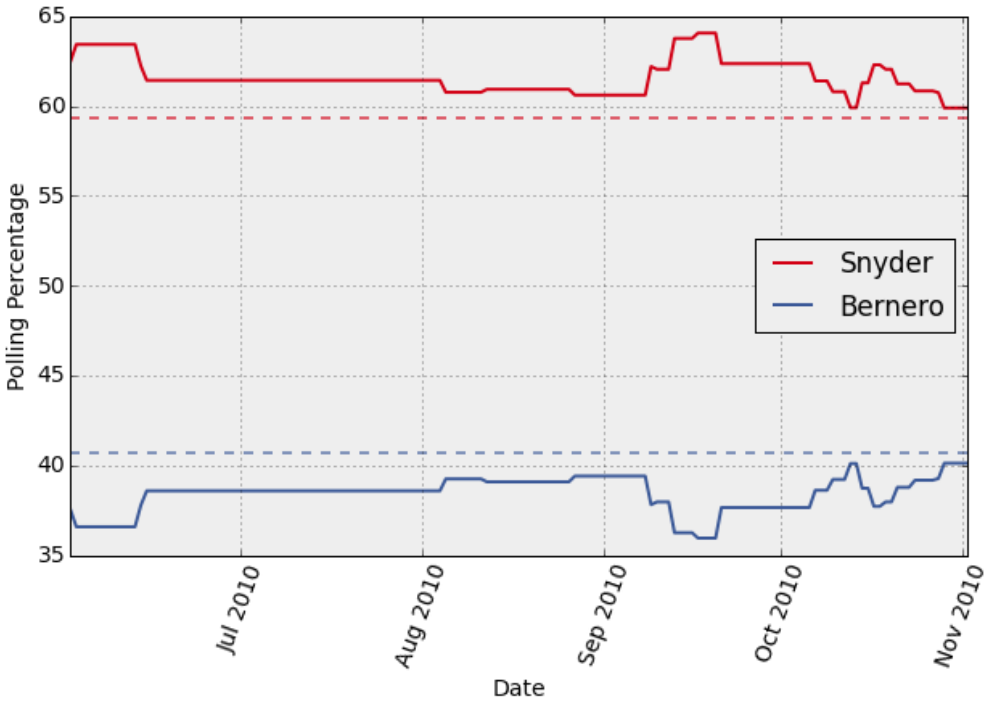

Brady
Quinn
Polling Percentage
Date
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

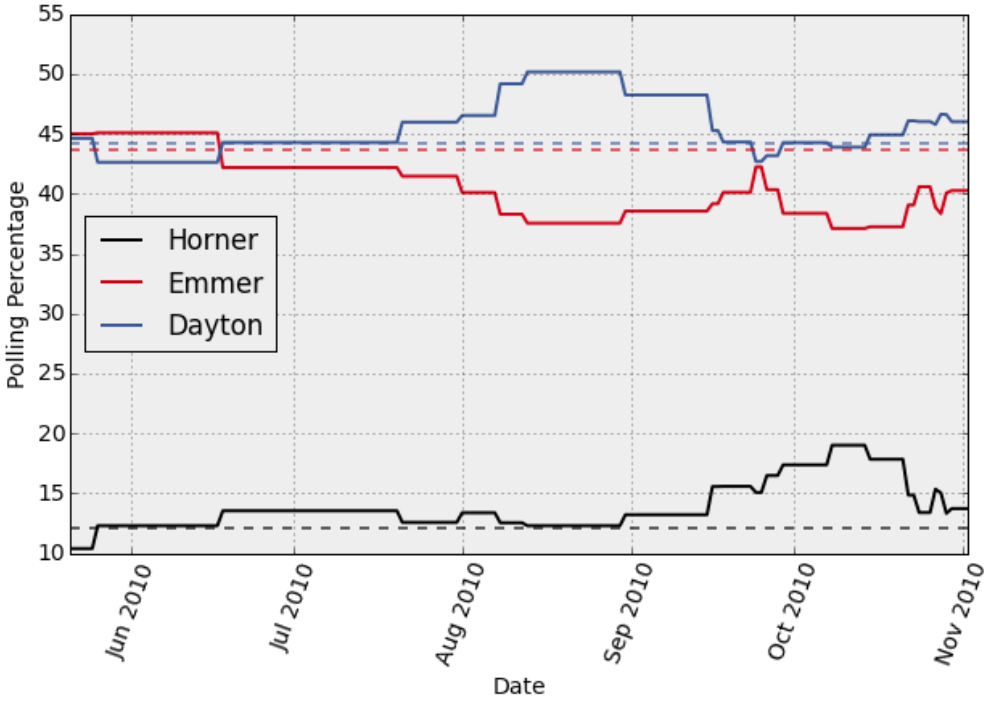

60
55
50
45
40
Polling Percentage
Culver
Branstad
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010
Date

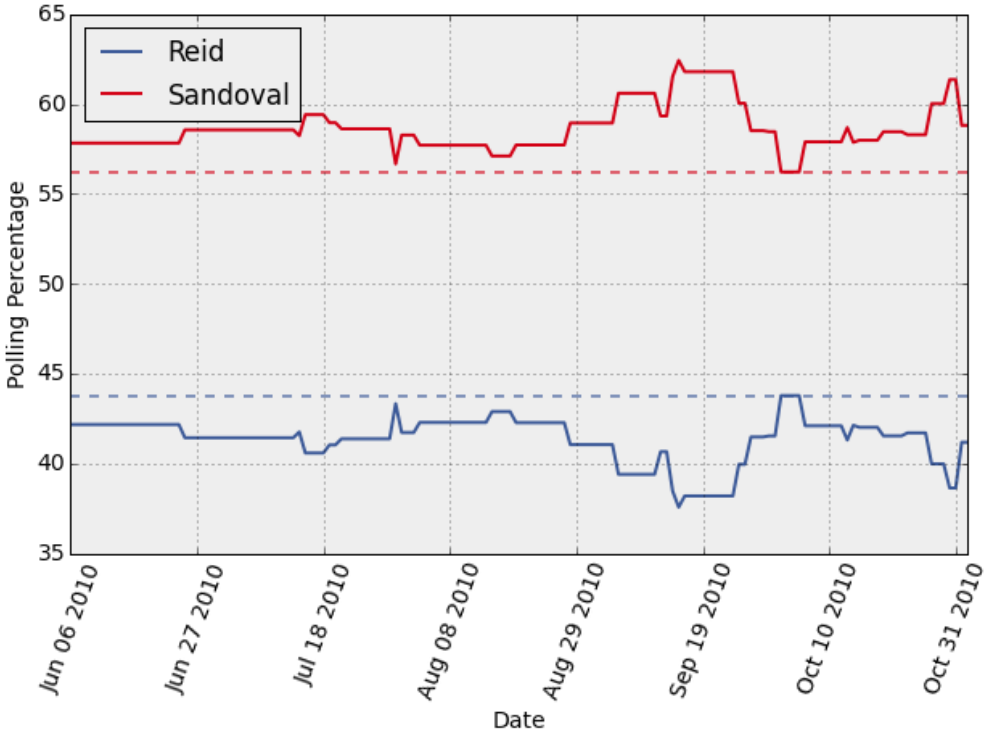

55
50
45
40
35
30
25
20
15
10
Polling Percentage
Mitchell
LePage
Cutler
Sep 27 2010
Oct 04 2010
Oct 11 2010
Oct 18 2010
Oct 25 2010
Nov 01 2010
Date

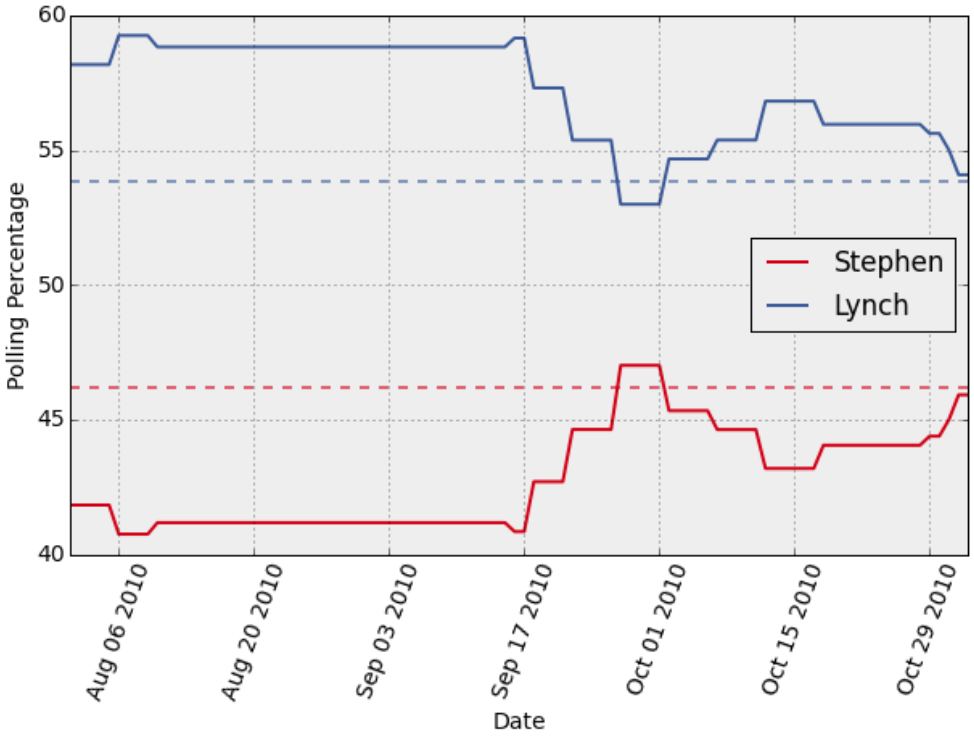

Polling Percentage
Date
Ehrlich
OMalley
58
56
54
52
50
48
46
44
42
Apr 2010
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

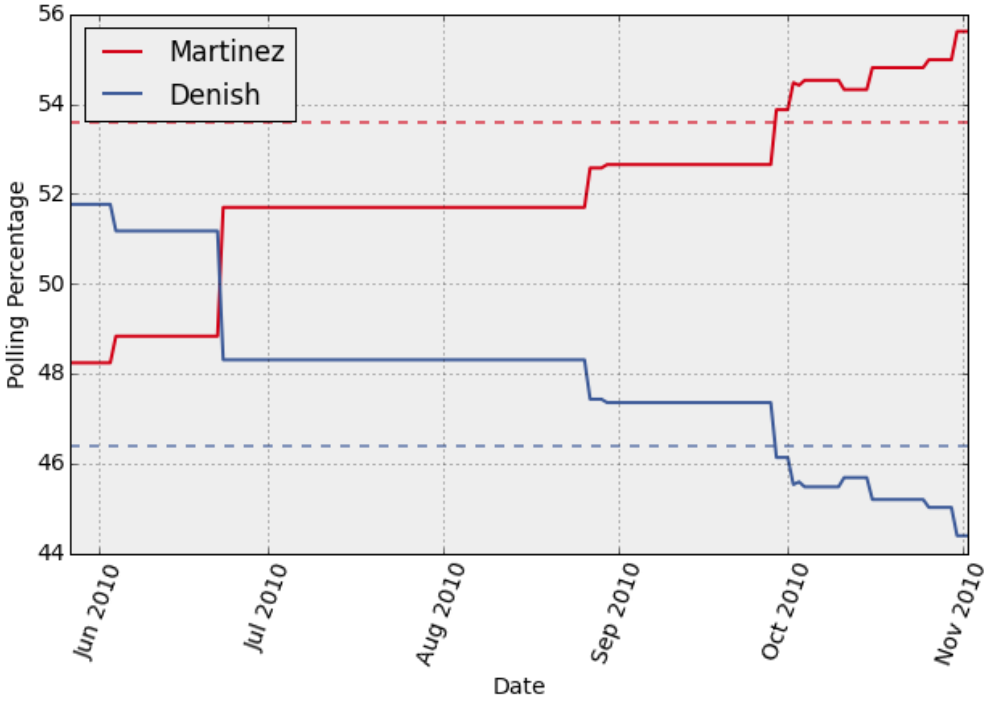

Polling Percentage
Date
Patrick
Baker
Cahill
50
45
40
35
30
25
20
15
10
5
Mar 2010
Apr 2010
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

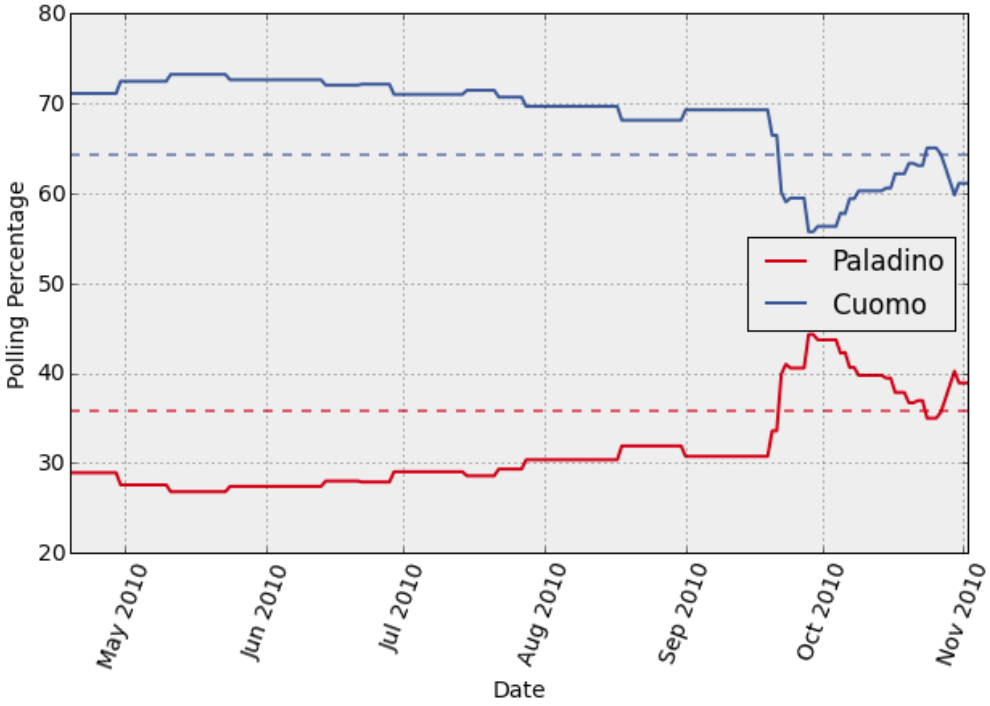

Polling Percentage
Date
Snyder
Bernero
65
60
55
50
45
40
35
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

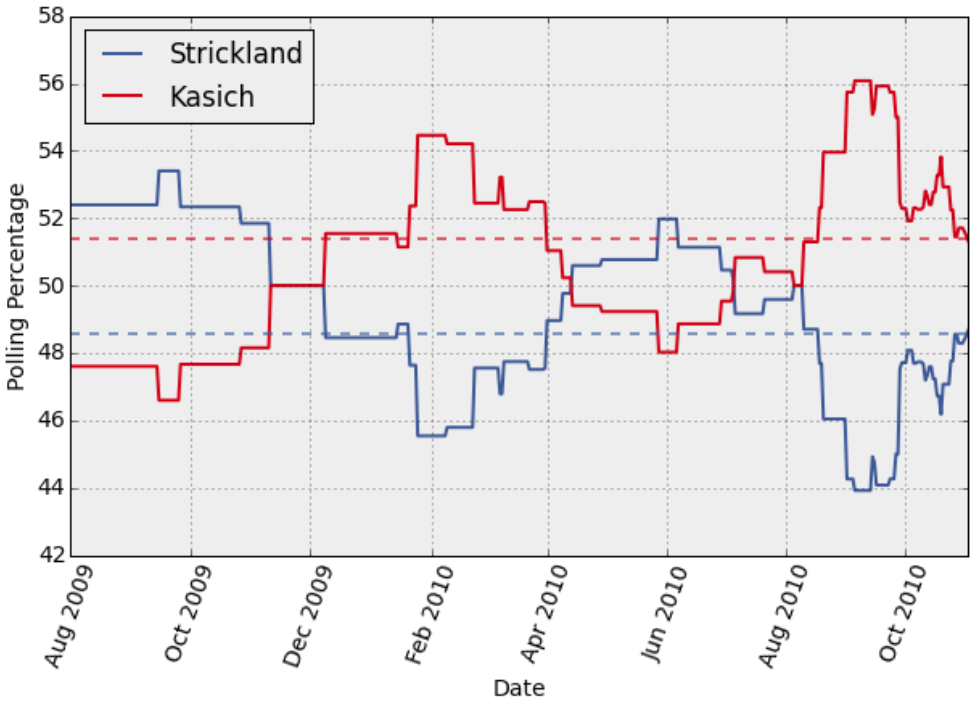

Polling Percentage
Date
Horner
Emmer
Dayton
55
50
45
40
35
30
25
20
15
10
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

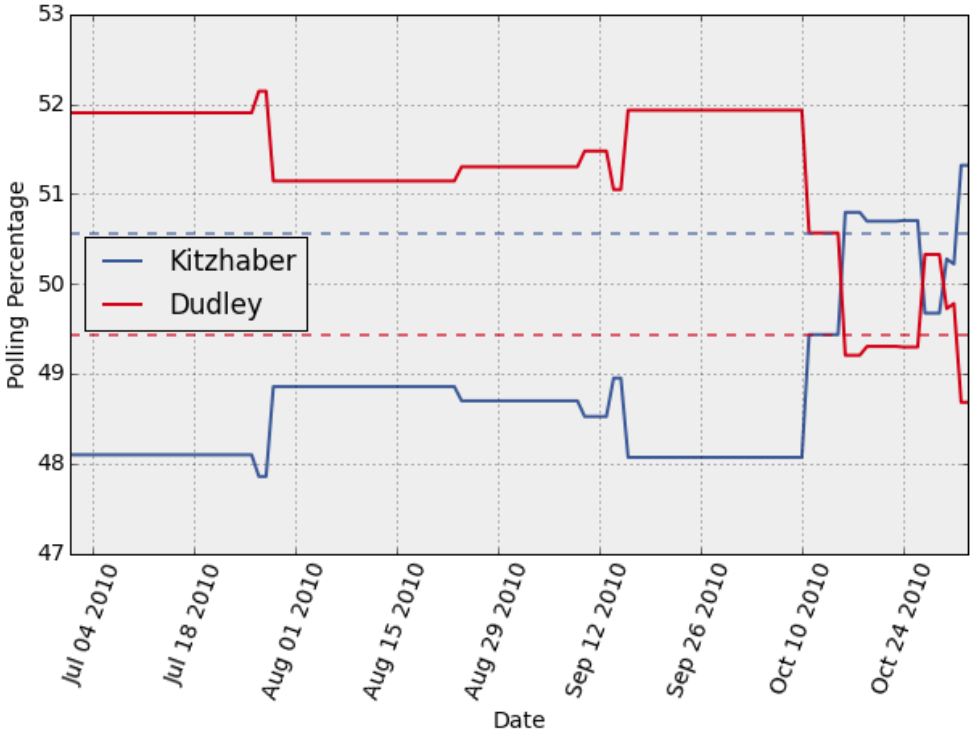

Reid
Sandoval
Polling Percentage
65
60
55
50
45
40
35
Jun 06 2010
Jun 27 2010
Jul 18 2010
Aug 08 2010
Aug 29 2010
Sep 19 2010
Oct 10 2010
Oct 31 2010
Date

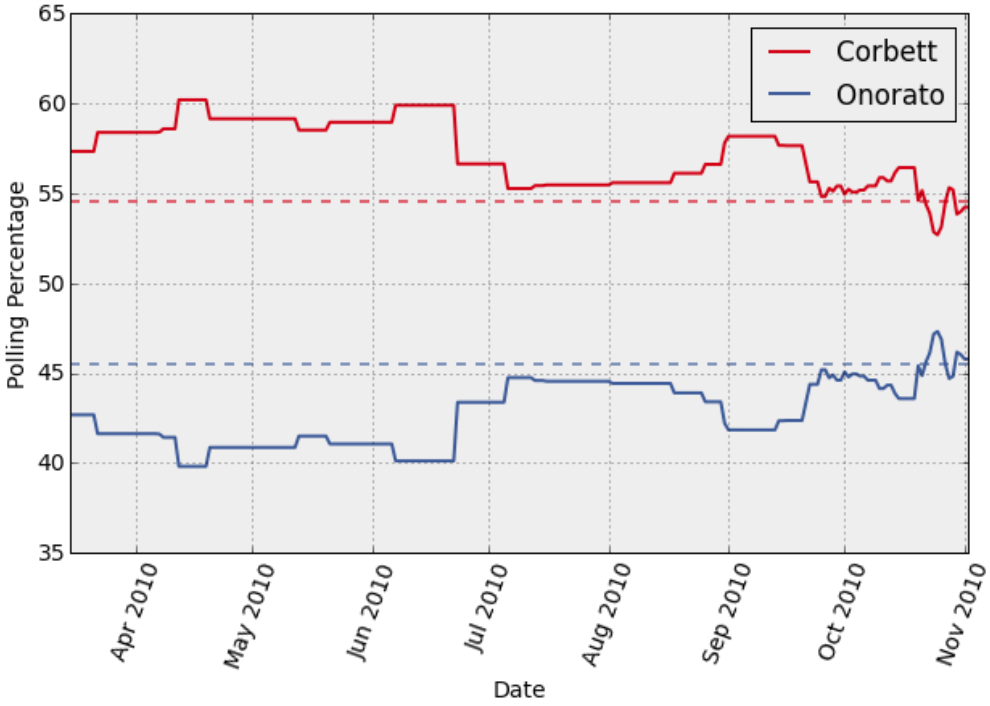

Stephen
Lynch
Polling Percentage
60
55
50
45
40
Aug 06 2010
Aug 20 2010
Sep 03 2010
Sep 17 2010
Oct 01 2010
Oct 15 2010
Oct 29 2010
Date

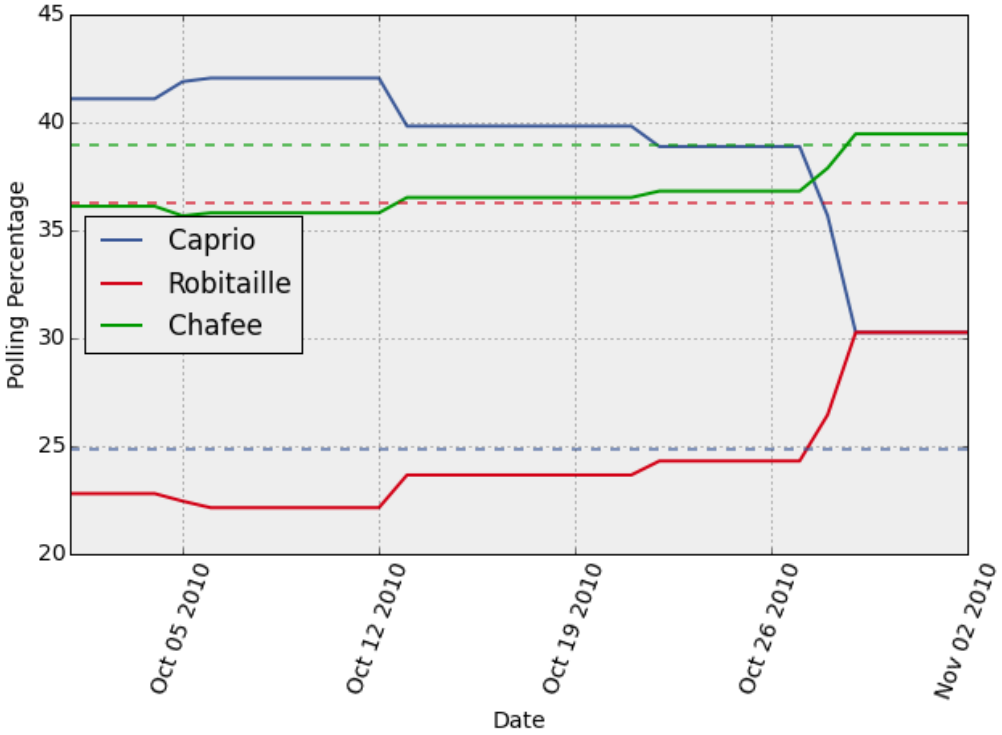

Martinez
Denish
Polling Percentage
Date
56
54
52
50
48
46
44
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

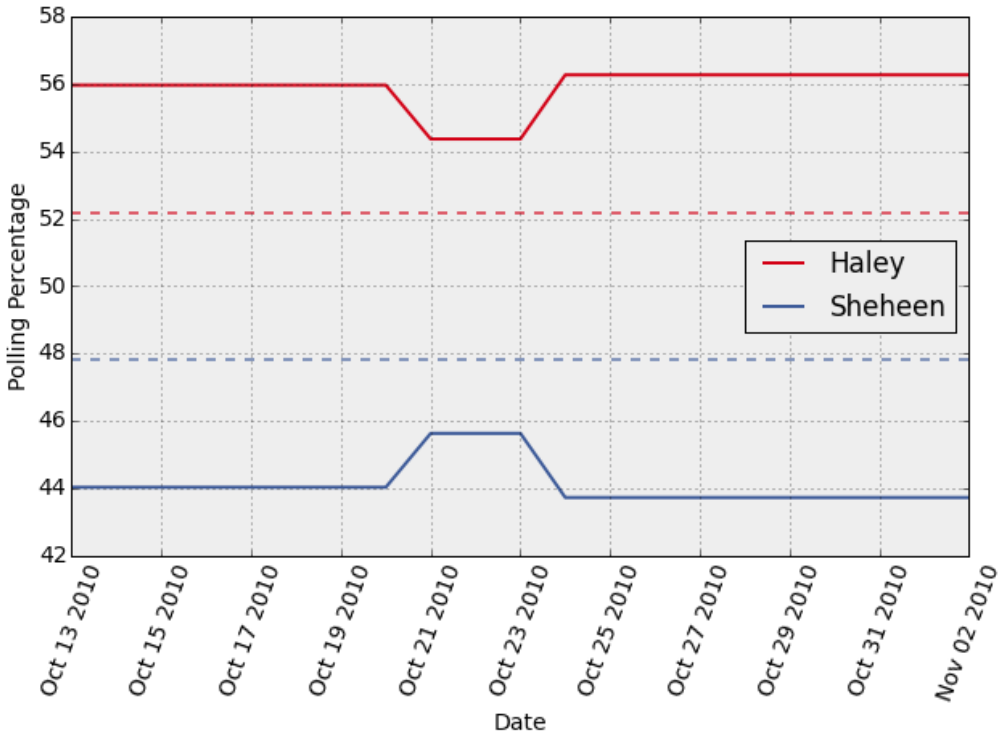

Paladino
Cuomo
Polling Percentage
Date
80
70
60
50
40
30
20
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010

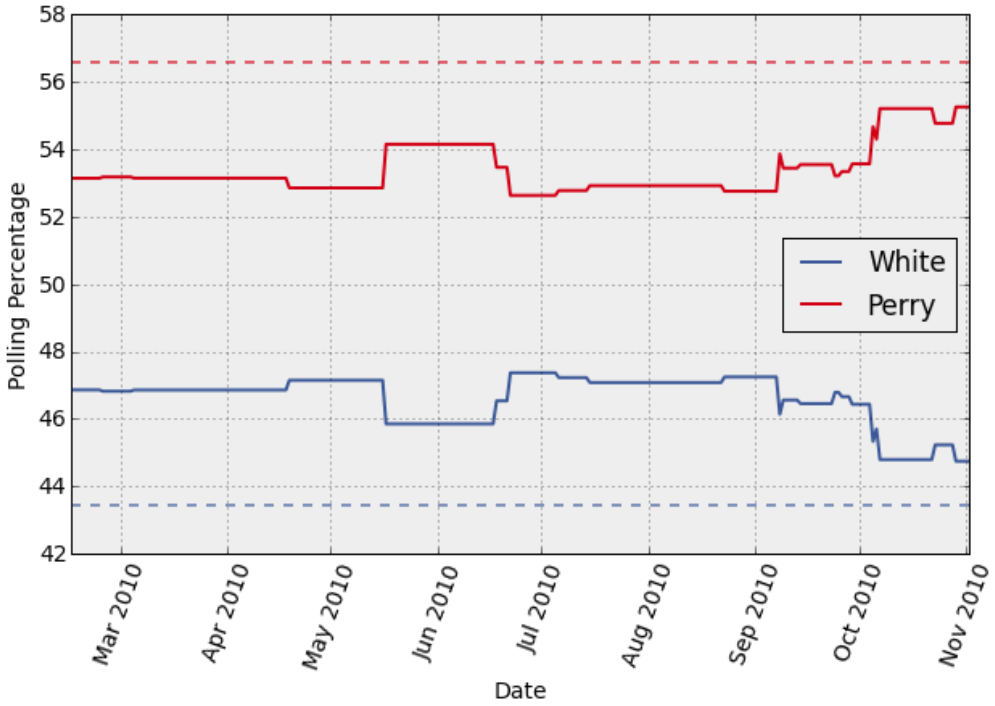

Strickland
Kasich
Polling Percentage
Date
58
56
54
52
50
48
46
44
42
Aug 2009
Oct 2009
Dec 2009
Feb 2010
Apr 2010
Jun 2010
Aug 2010
Oct 2010

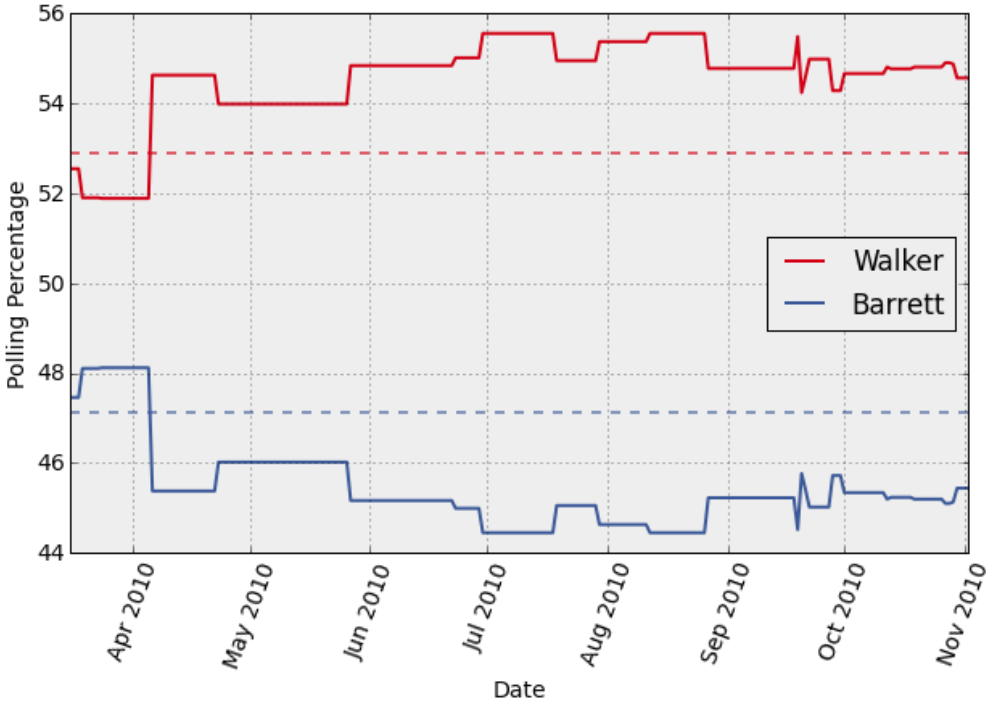

Kitzhaber
Dudley
Polling Percentage
Date
53
52
51
50
49
48
47
Jul 04 2010
Jul 18 2010
Aug 01 2010
Aug 15 2010
Aug 29 2010
Sep 12 2010
Sep 26 2010
Oct 10 2010
Oct 24 2010

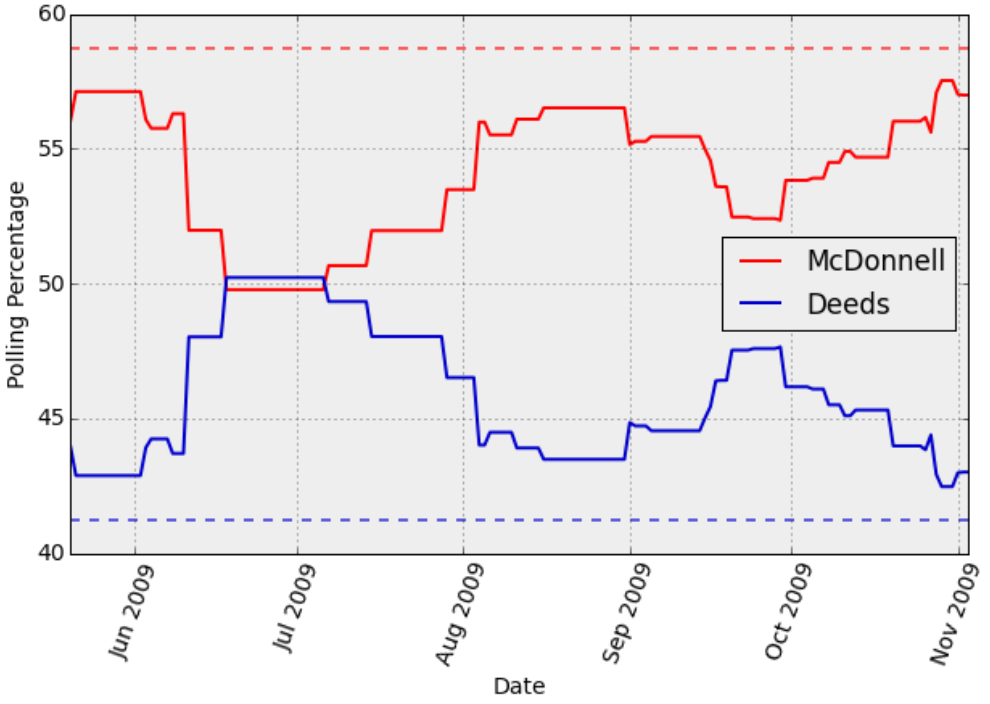

Corbett
Onorato
Polling Percentage
65
60
55
50
45
40
35
Apr 2010
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010
Date

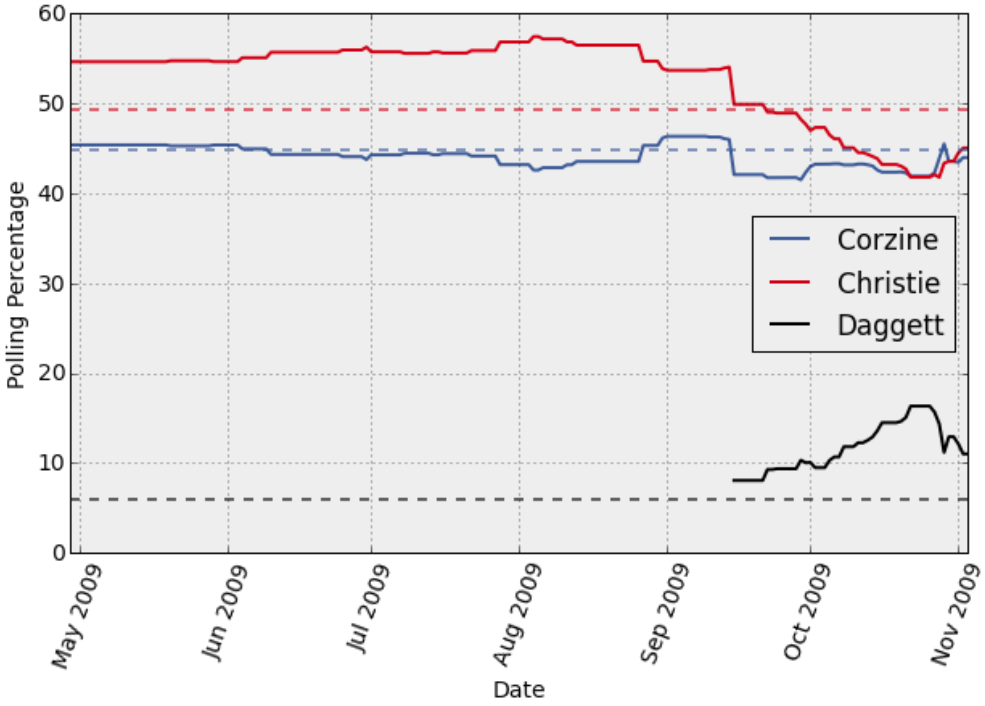

Caprio
Robitaille
Chafee
Polling Percentage
45
40
35
30
25
20
Oct 05 2010
Oct 12 2010
Oct 19 2010
Oct 26 2010
Nov 02 2010
Date


Polling Percentage
Date
Haley
Sheheen
58
56
54
52
50
48
46
44
42
Oct 13 2010
Oct 15 2010
Oct 17 2010
Oct 19 2010
Oct 21 2010
Oct 23 2010
Oct 25 2010
Oct 27 2010
Oct 29 2010
Oct 31 2010
Nov 02 2010


Polling Percentage
Date
White
Perry
58
56
54
52
50
48
46
44
42
Mar 2010
Apr 2010
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010


Walker
Barrett
Polling Percentage
Date
56
54
52
50
48
46
44
Apr 2010
May 2010
Jun 2010
Jul 2010
Aug 2010
Sep 2010
Oct 2010
Nov 2010


McDonnell
Deeds
Polling Percentage
Date
60
55
50
45
40
Jun 2009
Jul 2009
Aug 2009
Sep 2009
Oct 2009
Nov 2009




Briefly summarize these graphs -- how accurate is the typical poll a day before the election? How often does a prediction one month before the election mispredict the actual winner?

**Your summary here**

The day before the election the poll is quite accurate to the actual election, while with fluctuating polls the prediction one month before can be significant diffrent from the ground truth.

---

# Part 3: Analysis

## Problem 5

You are (finally!) in a position to do some quantitative analysis.

We have provided an `error_data` function that builds upon the functions you have written. It computes a new DataFrame with information about polling errors.

Use `error_data`, `find_governer_races`, and `pd.concat` to construct a Data Frame summarizing the forecast errors from all the Governor races.

**Hint**

It's best to set `ignore_index=True` in `pd.concat`

In [11]:

```
def party_from_color(color):
    if color in ['#0000CC', '#3B5998']:
        return 'democrat'
    if color in ['#FF0000', '#D30015']:
        return 'republican'
    return 'other'


def error_data(url):
    """
    Given a Governor race URL, download the poll data and race result,
    and construct a DataFrame with the following columns:

    candidate: Name of the candidate
    forecast_length: Number of days before the election
    percentage: The percent of poll votes a candidate has.
                Normalized to that the canddidate percentages add to 10
0%
    error: Difference between percentage and actual race reulst
    party: Political party of the candidate

    The data are resampled as necessary, to provide one data point per
day
    """

    id = id_from_url(url)
    xml = get_poll_xml(id)

    colors = plot_colors(xml)
    if len(colors) == 0:
        return pd.DataFrame()

    df = rcp_poll_data(xml)
    result = race_result(url)
    #remove non-letter characters from columns
    df = df.rename(columns={c: _strip(c) for c in df.columns})
    for k, v in result.items():
        result[_strip(k)] = v

    candidates = [c for c in df.columns if c is not 'date']

    #turn into a timeseries...
    df.index = df.date

    #...so that we can resample at regular, daily intervals
    df = df.resample('D')
    df = df.dropna()

    #compute forecast length in days
    #(assuming that last forecast happens on the day of the election, f
or simplicity)
    forecast_length = (df.date.max() - df.date).values
    forecast_length = forecast_length / np.timedelta64(1, 'D')  # conve
rt to number of days

    #compute forecast error
    errors = {}
    normalized = {}
    poll_lead = {}

    for c in candidates:
        #turn raw percentage into percentage of poll votes
        corr = df[c].values / df[candidates].sum(axis=1).values * 100.
        err = corr - result[_strip(c)]

        normalized[c] = corr
        errors[c] = err

    n = forecast_length.size

    result = {}
    result['percentage'] = np.hstack(normalized[c] for c in candidates)
    result['error'] = np.hstack(errors[c] for c in candidates)
```

```
    result['candidate'] = np.hstack(np.repeat(c, n) for c in candidates
)
    result['party'] = np.hstack(np.repeat(party_from_color(colors[_stri
p(c)]), n) for c in candidates)
    result['forecast_length'] = np.hstack(forecast_length for _ in cand
idates)

    result = pd.DataFrame(result)
    return result
```

In [12]:

```
"""
function
---------
all_error_data

Calls error_data on all races from find_governer_races(page),
and concatenates into a single DataFrame

Parameters
-----------
None

Examples
--------
df = all_error_data()
"""
#your code here

def all_error_data():
    page = requests.get(
        'http://www.realclearpolitics.com/epolls/2010/governor/2010_ele
ctions_governor_map.html').text.encode('ascii', 'ignore')
    return pd.concat([error_data(data) for data in find_governor_races(
page)], ignore_index=True)
```

In [13]:

```
errors = all_error_data()
```

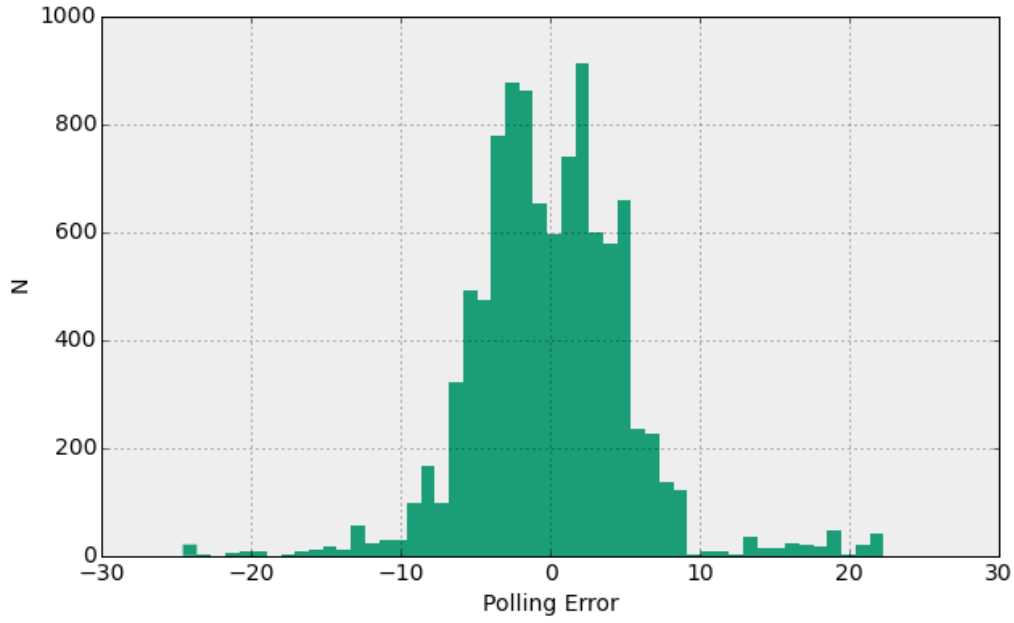
Here's a histogram of the error of every polling measurement in the data:

In [14]:

```
errors.error.hist(bins=50)
plt.xlabel("Polling Error")
plt.ylabel('N')
```

Out[14]:

```
<matplotlib.text.Text at 0x7f2b4297e490>
```



## Problem 6

Compute the standard deviation of the polling errors. How much uncertainty is there in the typical RCP poll?

In [15]:

```
#your code here
errors.error.std()
```

Out[15]:

```
5.2626670137897129
```

## Problem 7

Repeat this calculation for the data where `errors.forecast_length < 7` (i.e. the polls within a week of an election). How much more/less accurate are they? How about the data where `errors.forecast_length > 30`?

**Comment on this in 1 or 2 sentences**. Does this make sense?

In [16]:

```
#your code here
print "Errors within a week of election: ", errors[errors.forecast_leng
th < 7].error.std()
print "Errors within a month of election: ", errors[errors.forecast_len
gth > 30].error.std()

#As expected, the polls get more accurate as you get closer to the elec
tion day.
```

```
Errors within a week of election:  3.29443241142
Errors within a month of election:  5.32641507063
```

## Problem 8

**Bootstrap resampling** is a general purpose way to use empirical data like the `errors` DataFrame to estimate uncertainties. For example, consider the Viriginia Governor Race. If we wanted to estimate how likey it is that McAuliffe will win given the current RCP data, the approch would be:

1. Pick a large number N of experiments to run (say N=1000).
2. For each experiment, randomly select a value from `errors.error`. We are assuming that these numbers represent a reasonable error distribution for the current poll data.
3. Assume that the error on McAullife's current polling score is given by this number (and, by extension, the error on Cuccinelli's poll score is the opposite). Calculate who actually wins the election in this simulation.
4. Repeat N times, and calculate the percentage of simulations where either candidate wins.

Bootstrapping isn't foolproof: it makes the assumption that the previous Governor race errors are representative of the Virginia race, and it does a bad job at estimating very rare events (with only ~30 races in the errors DataFrame, it would be hard to accurately predict probabilities for 1-in-a-million scenarios). Nevertheless, it's a versatile technique.

Use bootstrap resampling to estimate how likely it is that each candidate could win the following races.

- Virginia Governor
- New Jersey Governor

**Summarize your results in a paragraph. What conclusions do you draw from the bootstrap analysis, and what assumptions did you make in reaching this conclusion. What are some limitations of this analysis?**

In [17]:

```
#your code here

def bootstrap_resampling(score1, score2):

    totalscore = score1 + score2
    score1 = score1 / totalscore * 100.
    score2 = score2 / totalscore * 100.

    rndErrors     = errors.error.irow(np.random.randint(0,len(errors),
1000)).values
    score1,score2 = score1 + rndErrors, score2 - rndErrors

    prediction1 = (score1 > score2).mean()
    prediction2 = 1- prediction1

    return prediction1,prediction2

mc, cu = bootstrap_resampling(49.6,39.7)
print "One week before election day:"
print 'McAullife: ', mc,'Cuccinelli', cu,'\n'

mc, cu = bootstrap_resampling(43.9,38.6)
print "One month before election day:"
print 'McAullife: ', mc,'Cuccinelli', cu,'\n'

mc, cu = bootstrap_resampling(42.4,42.4)
print "4 month before election day:"
print 'McAullife: ', mc,'Cuccinelli', cu,'\n'



ch, bu = bootstrap_resampling(59.6,33.2)
print "One week before election day:"
print 'Christie: ', ch,'Buono', bu,'\n'

ch, bu = bootstrap_resampling(57.5,32.8)
print "One month before election day:"
print 'Christie: ', ch,'Buono', bu,'\n'
```

```
One week before election day:
McAullife:  0.896 Cuccinelli 0.104

One month before election day:
McAullife:  0.737 Cuccinelli 0.263

4 month before election day:
McAullife:  0.498 Cuccinelli 0.502

One week before election day:
Christie:  0.997 Buono 0.003

One month before election day:
Christie:  0.991 Buono 0.009
```

**Your summary here**

The bootstrap simulation predicts the McAullife-Cuccinelli-race with a probability of 75% for the actual winning candidate. The simulation was done one month before the election day and gets more accurate one week prior to election day. The race for NewJersey Governor gets predicted with 99% accuracy. Having a look on the graph shows that the NewJersey Election was quite predictable already.

While as in the Virginia election the head-to-head race causes the prediction to be less reliable. The less difference there is between the candidates, the more uncertain is the outcome, the more unreliable is our prediction, thus only 75%

---

# Parting Thoughts

For comparison, most of the predictions in Nate Silver's presidental forecast had confidences of >95%. This is more precise than what we can estimate from the RCP poll alone. His approach, however, is the same basic idea (albeit he used many more polls, and carefully calibrated each based on demographic and other information). Homework 2 will dive into some of his techniques further.

---

# How to submit

To submit your homework keep checking in this notebook in your branch as you progress through the problems. In the end check in a PDF export of the evaluated notebook by going to File -> Print Preview and printing to a file.

---

*css tweaks in this cell*