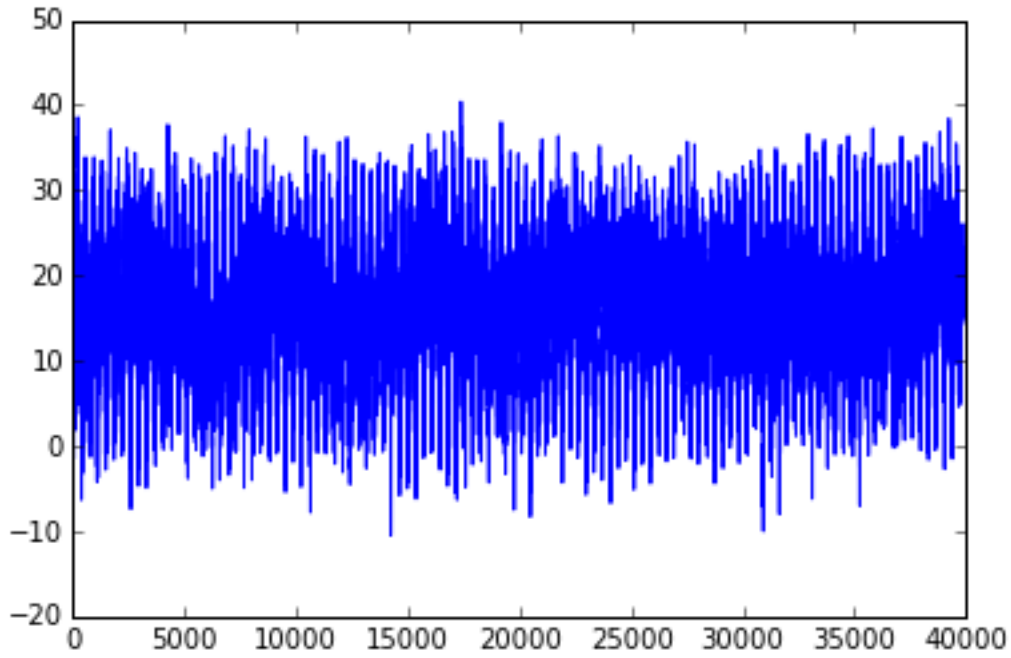
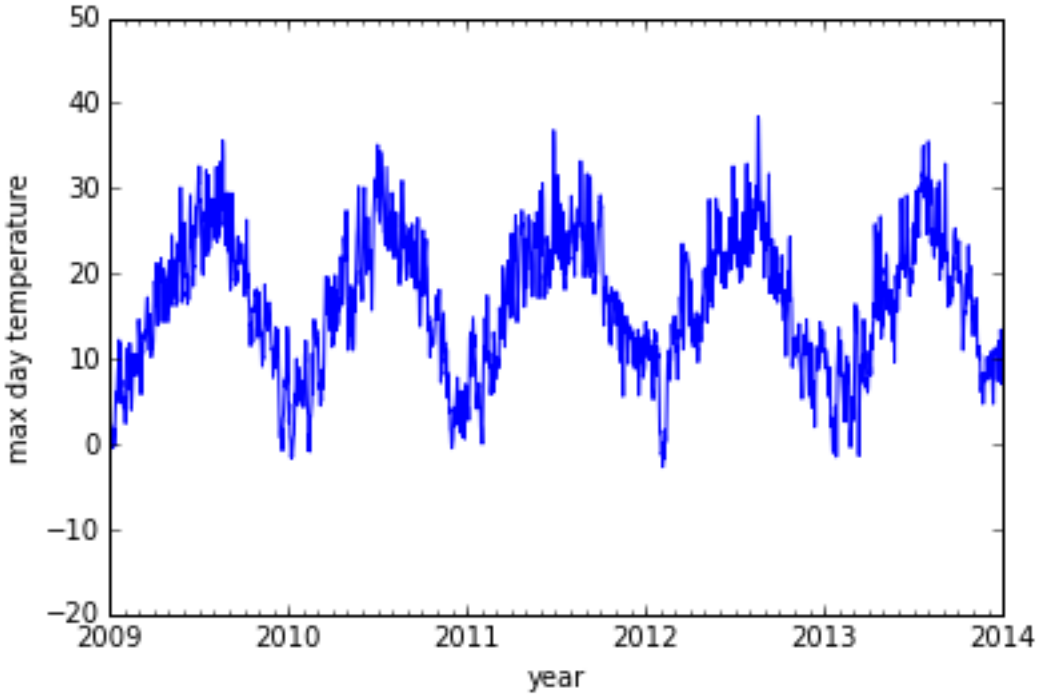
01_visualising_patterns_over_time

July 22, 2014

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import matplotlib.dates as mdates
        import numpy as np

In [2]: !head '../data/paris_weather.dat'

# Searching for GHCND series nr FR000007150
# coordinates:  48.82N,    2.34E,     75.0m; GHCN-D station code: FR000007150 PARIS-14E_PARC_MONTSOURIS
# WMO station  7156
# TMAX GHCN-D V2.0 data with QC in [Celsius]
# The non-U.S. data cannot be redistributed within or outside of the U.S. for any commercial activities
 1900  1  1     10.10
 1900  1  2     12.00
 1900  1  3      9.30
 1900  1  4      8.30
 1900  1  5      6.00

In [3]: temperatures = np.loadtxt('../data/paris_weather.dat',
                                  dtype=[('year', 'i8'),
                                         ('month', 'i8'),
                                         ('day', 'i8'),
                                         ('temp', 'f8')])

In [4]: from goodies import as_table

In [5]: as_table(temperatures)

Out[5]: <IPython.core.display.HTML at 0x10fc868d0>

In [6]: plt.plot(temperatures['temp'])

Out[6]: [<matplotlib.lines.Line2D at 0x11015be50>]
```

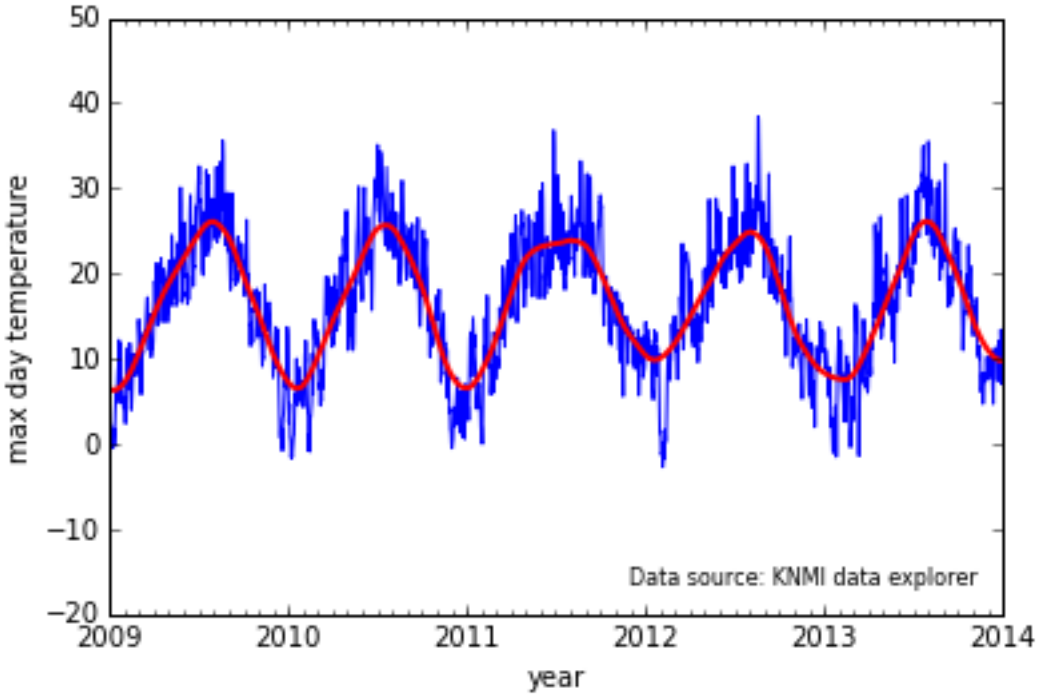



```
In [8]: import datetime
        dates = [datetime.date(year,month,day) for year, month, day, T in temperatures]

In [9]: major = mdates.YearLocator()
        minor = mdates.MonthLocator()

In [10]: plt.plot(dates, temperatures['temp'])
         plt.xlabel('year')
         plt.ylabel('max day temperature')
         plt.xlim(datetime.datetime(2009,1,1), datetime.datetime(2014,1,1))
         ax = plt.gca()
         ax.xaxis.set_major_locator(major)
         ax.xaxis.set_minor_locator(minor)
```



In [19]:
```
from statsmodels.nonparametric.smoothers_lowess import lowess
plt.plot(dates, temperatures['temp'])
x = np.array(dates, dtype=np.datetime64)
smoothed_temp = lowess(temperatures['temp'], x,
                       return_sorted=False,
                       frac=0.003,
                       delta=10)
plt.plot(dates, smoothed_temp, color='r', lw=2)
plt.xlabel('year')
plt.ylabel('max day temperature')
plt.xlim(datetime.datetime(2009,1,1), datetime.datetime(2014,1,1))
ax = plt.gca()
ax.xaxis.set_major_locator(major)
ax.xaxis.set_minor_locator(minor)
plt.text(0.58, 0.05, 'Data source: KNMI Climate Explorer',
         transform=ax.transAxes, size=8)
```

Out[19]:
```
<matplotlib.text.Text at 0x10ff05190>
```


50
40
30
20
10
0
−10
−20
max day temperature
2009
2010
2011
2012
2013
2014
year
Data source: KNMI data explorer

In []:

02_visualising_proportions

July 22, 2014

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from goodies import as_table

%matplotlib inline
```

In [2]:
```
import sqlite3

sql_query = """
SELECT country1, teams.title as country2, winner
FROM ( SELECT teams.title as country1, team2_id, winner
       FROM (SELECT team1_id, team2_id, winner
             FROM games
             INNER JOIN rounds on games.round_id=rounds.id
             WHERE rounds.title='Final') t1
       INNER JOIN teams on teams.id=t1.team1_id) t2
INNER JOIN teams on teams.id=t2.team2_id;
"""

db = sqlite3.connect('../data/worldcup.db')
cur = db.execute(sql_query)

arr = np.fromiter(cur, dtype=[('country1', 'S30'),
                              ('country2', 'S30'),
                              ('winner', 'i2')])

arr['country1'] = np.char.replace(arr['country1'], 'West Germany (-1989)', 'Germany')
arr['country2'] = np.char.replace(arr['country2'], 'West Germany (-1989)', 'Germany')
```

In [3]:
```
as_table(arr, maxrows=100)
```

Out[3]:
```
<IPython.core.display.HTML at 0x10cfabc90>
```

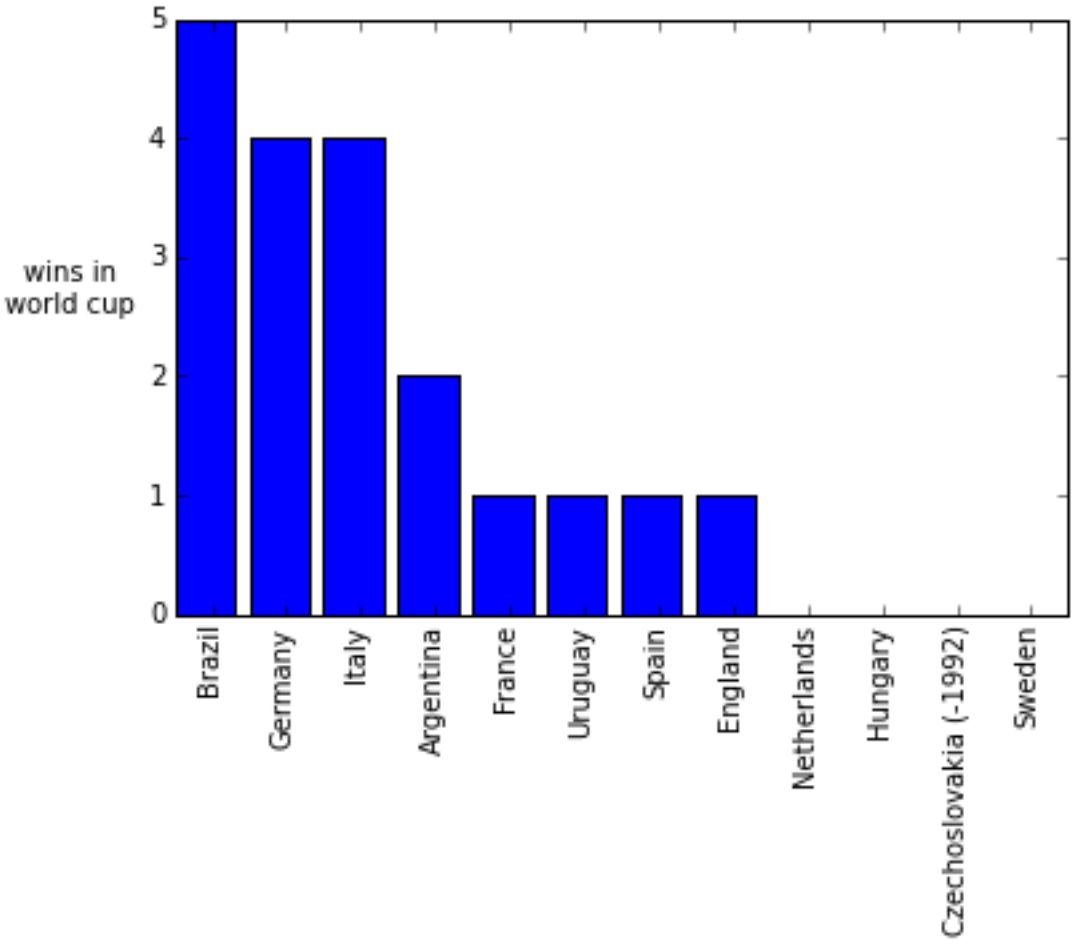
In [4]:
```
countries = np.unique(np.concatenate([arr['country1'], arr['country2']]))
winners = np.where(arr['winner']==1, arr['country1'], arr['country2'])
loosers = np.where(arr['winner']==2, arr['country1'], arr['country2'])
```

In [5]:
```
wins = [np.sum(winners==c) for c in countries]
losses = [np.sum(loosers==c) for c in countries]

stats = np.rec.fromarrays((countries, wins, losses),
                          names='country,wins,losses')
```

In [6]:
```
i = np.argsort(stats, order=('wins', 'losses', 'country'))[::-1]
stats = stats[i]
```

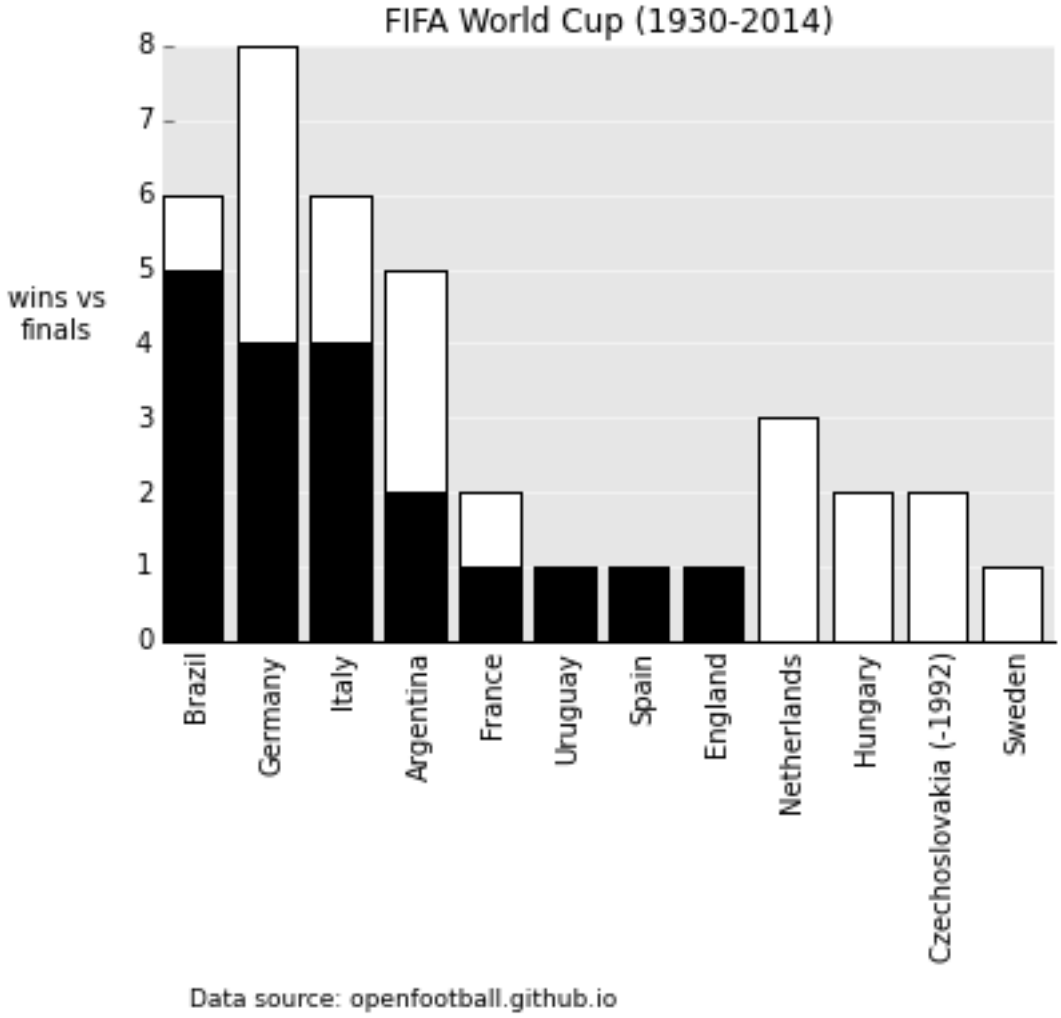
```
x = np.arange(len(stats))
plt.bar(x, stats['wins'])
plt.xticks(x+0.5, stats['country'], rotation=90)
plt.ylabel('wins in\nworld cup', rotation=0, labelpad=30)
```

Out[6]: <matplotlib.text.Text at 0x10d07cf90>



In [7]:
```
plt.bar(x, stats['wins']+stats['losses'], fc='w', ec='k')
plt.bar(x, stats['wins'], ec='none', fc='k')
plt.xticks(x+0.5, stats['country'], rotation=90)
ax = plt.gca()
[ax.spines[s].set_visible(False) for s in ['top', 'left', 'right']]
#ax.spines['left'].set_position(('outward',20))
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('left')
plt.ylabel('wins vs\nfinals', rotation=0, labelpad=30)
ax.set_axis_bgcolor('0.9')
ax.set_axisbelow(True)
plt.grid(axis='y', ls='-', color='w',zorder=200)
plt.title('FIFA World Cup (1930-2014)')
plt.figtext(0.15,-0.35, 'Data source: openfootball.github.io', size=9)
```

Out[7]: <matplotlib.text.Text at 0x10d1d3b10>



In [7]:

03_visualising_distributions

July 22, 2014

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import HTML

%matplotlib inline
```

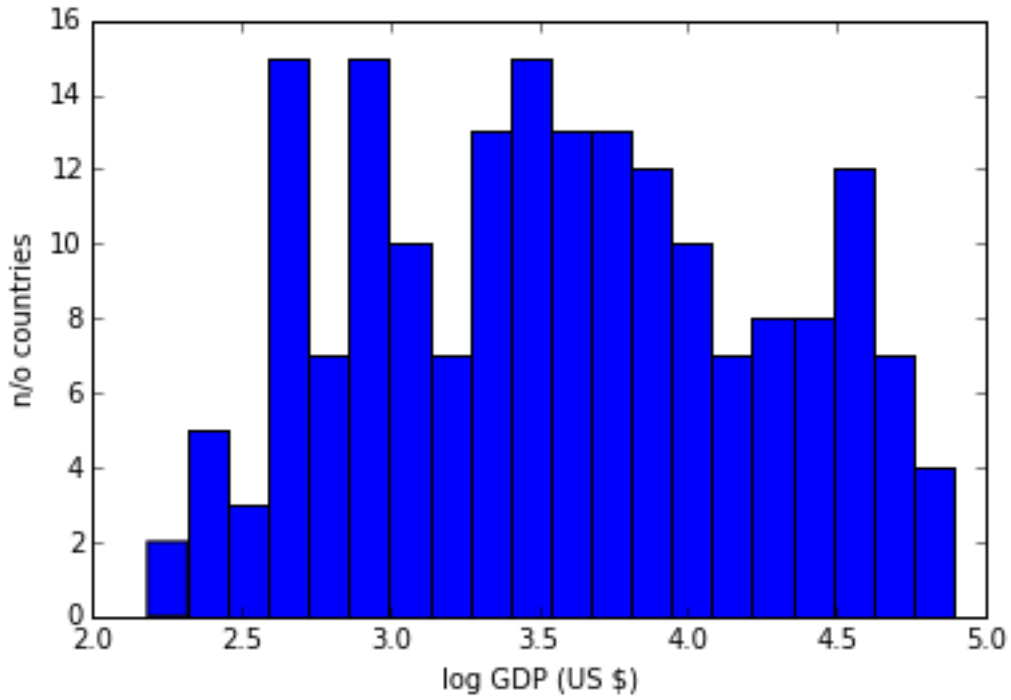
In [2]:
```
df = pd.read_csv('../data/gdp_wordbank.csv',
                 index_col=[1,2], na_values=['..'])
df.sortlevel(inplace=True)

HTML(df.loc[( slice('NY.GDP.PCAP.KD', 'NY.GDP.PCAP.KN'),
              slice('Andorra','Angola')), :]
        .ix[:,:5]
        .to_html())
```

Out[2]: <IPython.core.display.HTML at 0x10cdffd10>

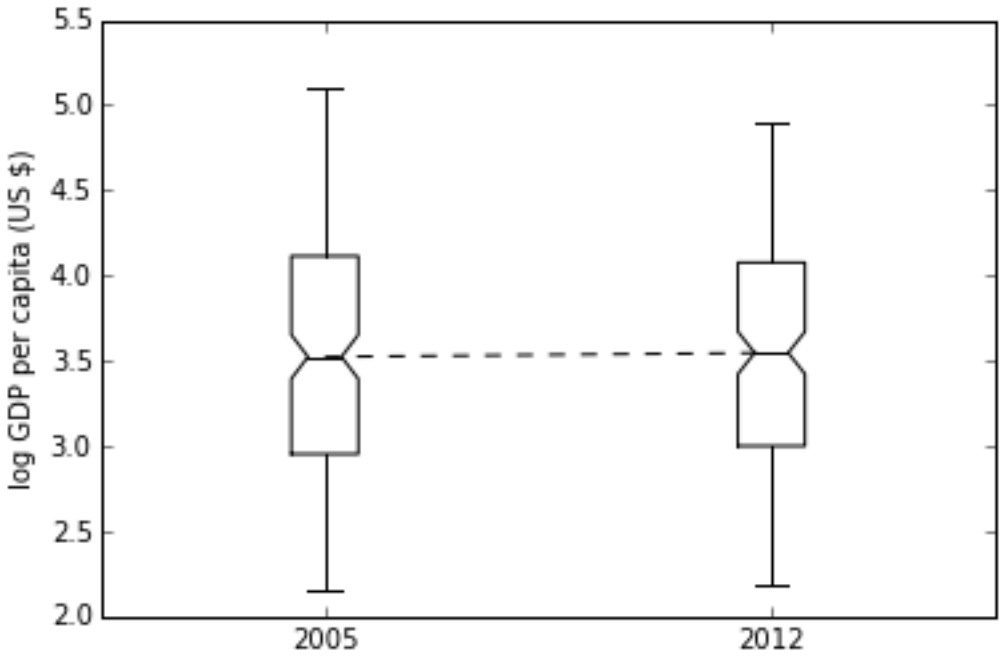
In [3]:
```
df_selected = df.ix[u'NY.GDP.PCAP.KD']
```

In [4]:
```
gdp2012 = np.log10(np.array(df_selected['2012 [YR2012]'].dropna(), dtype=np.float32))
plt.hist(gdp2012, 20)
plt.xlabel('log GDP (US $)')
plt.ylabel('n/o countries')
```

Out[4]: <matplotlib.text.Text at 0x1111e9c10>

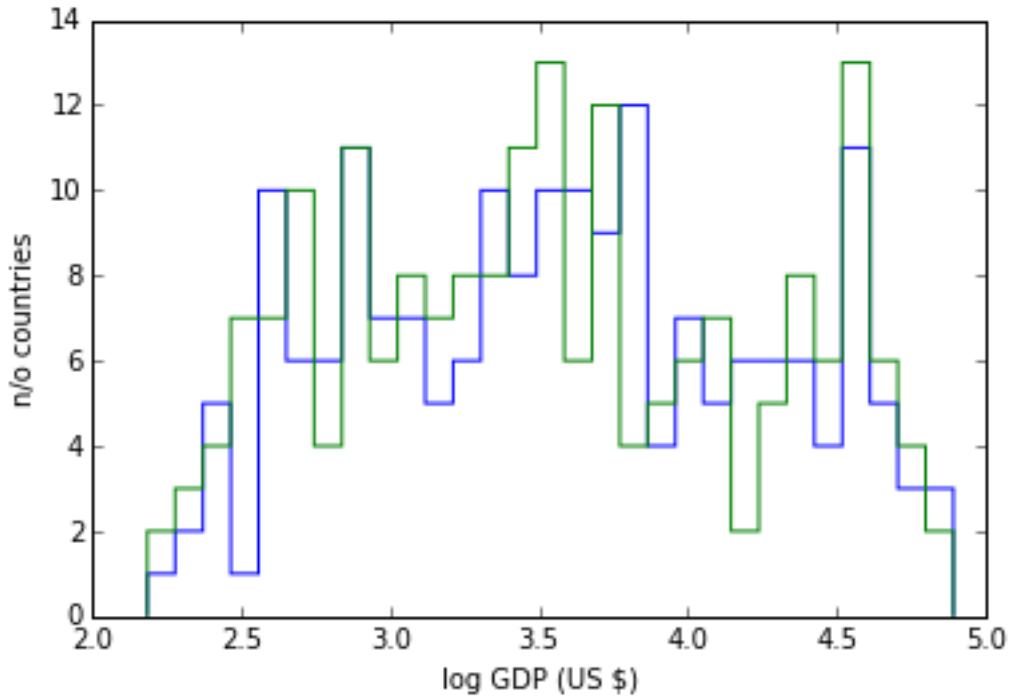



```
In [5]: gdp2005 = np.log10(np.array(df_selected['2005 [YR2005]'].dropna(), dtype=np.float32))
        plt.plot([1,2], [np.median(gdp2005), np.median(gdp2012)], 'k--')
        elements = plt.boxplot([gdp2005, gdp2012],
                               labels=['2005', '2012'],
                               notch=True);
        plt.ylabel('log GDP per capita (US $)')
        plt.setp(elements['medians'], color='k')
        plt.setp(elements['whiskers'], color='k', ls='solid')
        plt.setp(elements['boxes'], color='k')

Out[5]: [None, None]
```



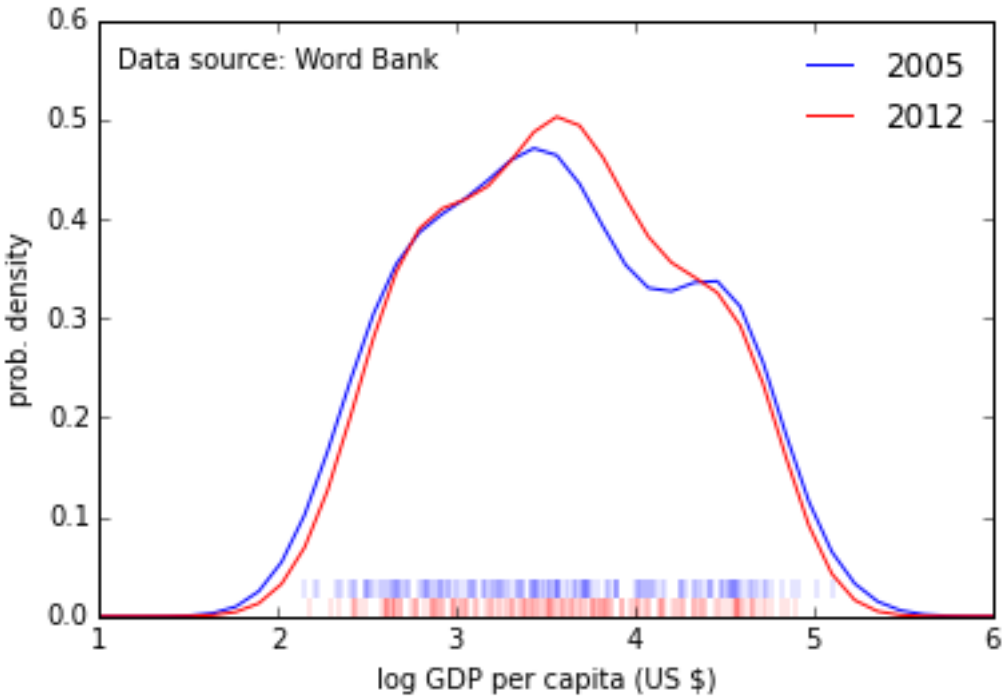

```
In [6]: bins = np.linspace(gdp2012.min(), gdp2012.max(), 30)
        n2012, bins2012, patches = plt.hist(gdp2012, bins, histtype='step',
                                            label='2012')
        n2005, bins2005, patches = plt.hist(gdp2005, bins, histtype='step',
                                            label='2005')
        plt.xlabel('log GDP (US $)')
        plt.ylabel('n/o countries')
```

```
Out[6]: <matplotlib.text.Text at 0x11128d190>
```



In [7]:
```
from scipy import stats
from matplotlib import transforms


kde2005 = stats.kde.gaussian_kde(gdp2005)
kde2012 = stats.kde.gaussian_kde(gdp2012)

x = np.linspace(1, 6, 40)

ax = plt.subplot(111)

trans = transforms.blended_transform_factory(ax.transData, ax.transAxes)

l1, = plt.plot(x, kde2005(x))
l2, = plt.plot(x, kde2012(x), color='r')
plt.legend([l1, l2], ['2005', '2012'], frameon=False)
plt.vlines(gdp2012, 0.0, 0.03, color='r', lw=0.2,
           transform=trans)
plt.vlines(gdp2005, 0.03, 0.06, color='b', lw=0.2,
           transform=trans)
plt.xlabel('log GDP per capita (US $)')
plt.ylabel('prob. density')

plt.text(0.02, 0.92, "Data source: Word Bank", transform=ax.transAxes)
```

Out[7]:
```
<matplotlib.text.Text at 0x11357a6d0>
```




```
In [8]: print "K-S stat: {:.2f}, p-value: {:.2f}".format(*stats.ks_2samp(gdp2005, gdp2012))
```

```
K-S stat: 0.05, p-value: 0.98
```

```
In [8]:
```

04_visualising_correlations

July 22, 2014

In [1]:
```
import matplotlib

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

%matplotlib inline
matplotlib.rcParams['savefig.dpi'] = 300
#%config InlineBackend.figure_format = 'svg'
```

In [2]:
```
api_url = "http://api.brain-map.org/api/v2/data/"
query_expression = "query.json?criteria=service::human_microarray_expression[probes$eq{probes}]
query_probes = "query.xml?criteria=model::Probe,rma::criteria,[probe_type$eq'DNA'],products[abb

donor = "'H035.2001'"
gene_ids="'SLC6A2','SCN1A'"
```

In [3]:
```
from urllib2 import urlopen
from contextlib import closing
import json
from lxml import etree

request_url = api_url + query_probes.format(geneid=gene_ids)

with closing(urlopen(request_url)) as response:
    xml_data = response.read()
    tree = etree.fromstring(xml_data)

probes = ','.join([t.text for t in tree.xpath('//probe/id')])

request_url = api_url + query_expression.format(probes=probes, donor=donor)
with closing(urlopen(request_url)) as response:
    probe_data = json.load(response)['msg']
```

In [4]:
```
expr_lvls    = {prb['name']: map(float, prb['expression_level'])
                for prb in probe_data['probes']}
structures   = [s['top_level_structure']['abbreviation']
                for s in probe_data['samples']]
structure_id = [s['structure']['id']
                for s in probe_data['samples']]
genes        = {prb['name']: prb['gene-symbol']
                for prb in probe_data['probes']}
expr_lvls.update({'top_level_structure' : structures,
                  'structure_id' : structure_id})
```

```
df = pd.DataFrame(expr_lvls)
df.to_csv('../data/allen_brain_atlas.csv', index=False)
```

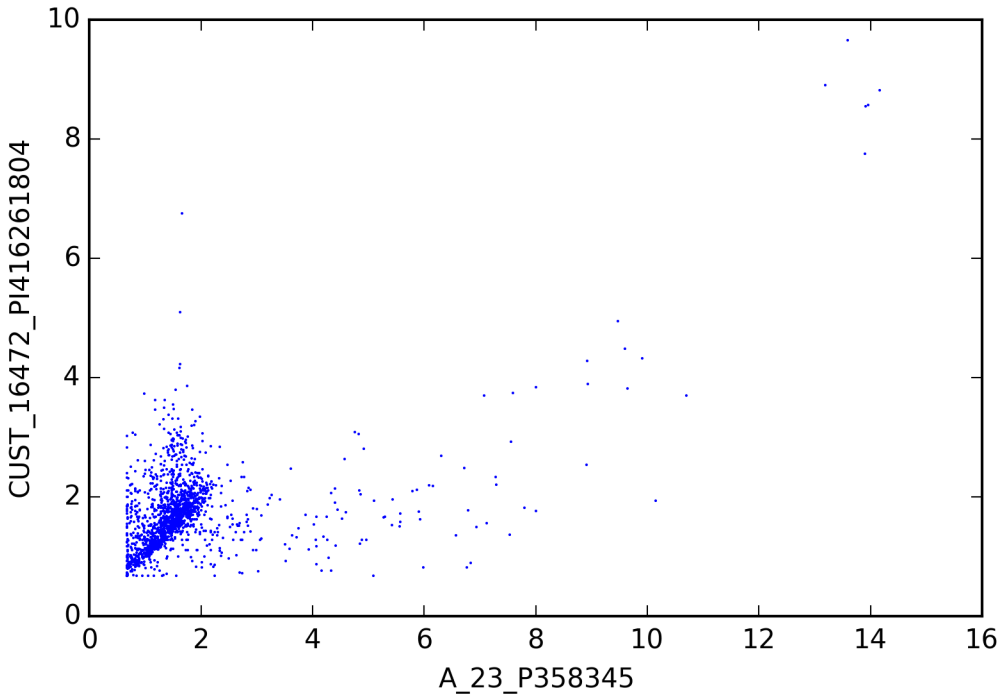
In [5]:
```
df = pd.read_csv('../data/allen_brain_atlas.csv')
df[:10]
```

Out[5]:

	A_23_P28224	A_23_P358345	CUST_16472_PI416261804	CUST_17139_PI416261804	CUST_546_PI416408490	structure_id	top_level_structure
0	8.8037	1.6248	2.3772	5.9888	6.8569	4055	FL
1	8.9927	1.5055	1.7143	5.8315	6.5077	4079	FL
2	8.6063	1.5778	2.8595	6.0884	6.5789	4079	FL
3	8.5581	2.0678	1.9892	6.3405	6.6196	4079	FL
4	8.7339	1.4767	1.4731	5.8716	6.4472	4080	FL
5	8.4503	1.8223	2.2234	5.7434	6.4456	4080	FL
6	8.7355	1.5104	1.6819	5.9797	6.8184	4890	FL
7	8.9026	1.5253	1.7685	6.2411	6.6980	4890	FL
8	8.7765	2.9994	1.7939	6.2621	6.8061	4048	FL
9	8.9004	1.7628	1.7252	6.3940	6.9415	4048	FL

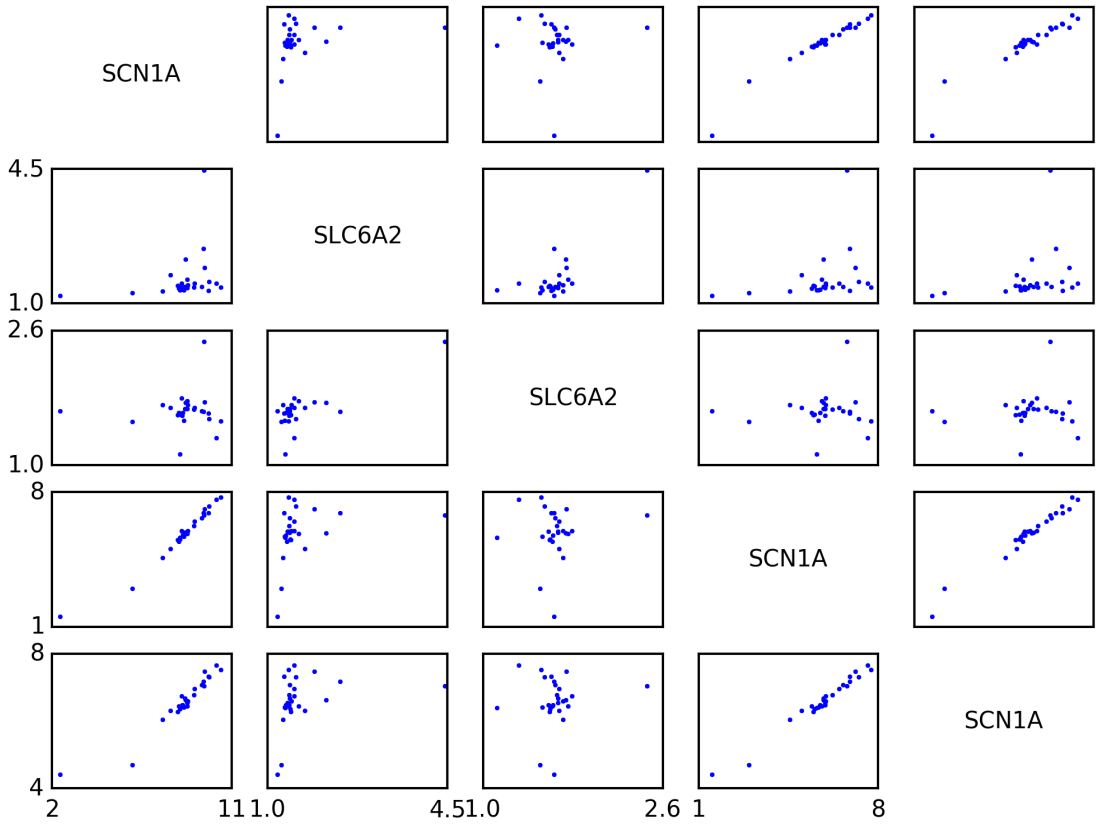
In [6]:
```
plt.plot(df.ix[::2,1], df.ix[::2,2], '.', ms=1)

plt.xlabel(df.columns[1])
plt.ylabel(df.columns[2])
```

Out[6]:
```
<matplotlib.text.Text at 0x10edf6750>
```




```
In [7]: df_aggregated = df.groupby('top_level_structure').mean().drop('structure_id', 1)

In [8]: nx = ny = df_aggregated.shape[1]
        fig, axes = plt.subplots(nx, ny,
                                 sharex='col', sharey='row',
                                 figsize=(8,6))
        for i in range(nx):
            for j in range(ny):
                if j != i:
                    axes[j, i].plot(df_aggregated.ix[:, i],
                                    df_aggregated.ix[:, j], '.', ms=3)

                else:
                    axes[j,i].set_axis_off()
                    axes[j,i].text(0.5, 0.5, genes[df_aggregated.columns[i]],
                                   transform=axes[j,i].transAxes,
                                   va='center',
                                   ha='center')
        [ax.set_yticks(ax.get_ylim()) for ax in axes[:,0]]
        [ax.set_xticks(ax.get_xlim()) for ax in axes[0,:]]

Out[8]: [[<matplotlib.axis.XTick at 0x10f72e810>,
          <matplotlib.axis.XTick at 0x10ee51f90>],
         [<matplotlib.axis.XTick at 0x10fea75d0>,
          <matplotlib.axis.XTick at 0x10fea7ed0>],
         [<matplotlib.axis.XTick at 0x10f73d750>,
          <matplotlib.axis.XTick at 0x10f7476d0>],
         [<matplotlib.axis.XTick at 0x10ff91b90>,
```

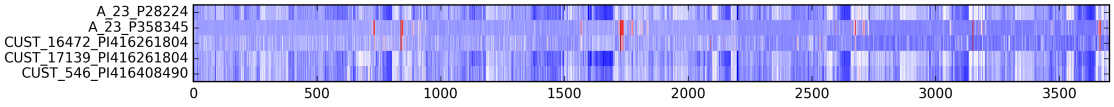
```
 <matplotlib.axis.XTick at 0x10ff7e350>],
[<matplotlib.axis.XTick at 0x10fffe250>,
 <matplotlib.axis.XTick at 0x10fffead0>]]
```



In [9]:
```
df_pure = df.ix[:,:-2]
data = (np.array(df_pure))
zscore = (data - data.mean(0))/data.std(0)
```

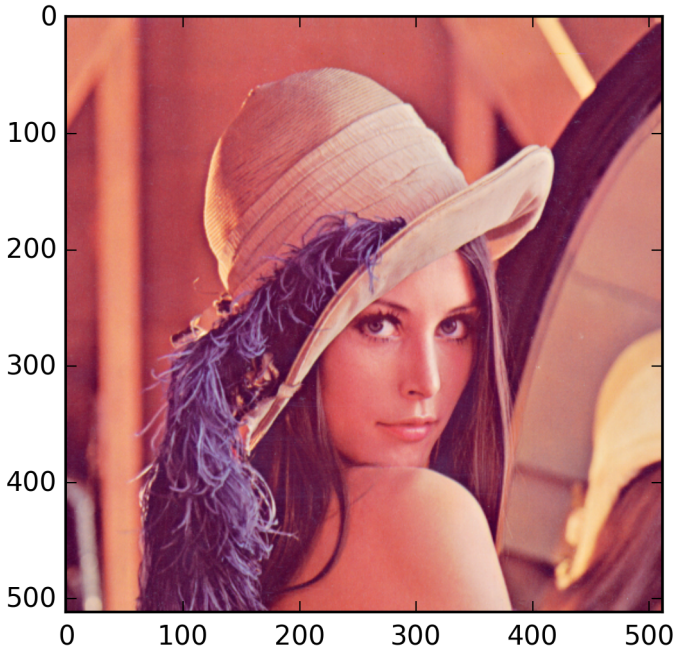
In [10]:
```
plt.figure(figsize=(12,1))
plt.imshow(zscore.T, aspect='auto', interpolation='nearest', cmap='seismic')
plt.yticks(np.arange(len(df_pure.columns)), df_pure.columns)
```

Out[10]:
```
([<matplotlib.axis.YTick at 0x110562f50>,
  <matplotlib.axis.YTick at 0x1107fdd50>,
  <matplotlib.axis.YTick at 0x10ff9e510>,
  <matplotlib.axis.YTick at 0x10fe93ed0>,
  <matplotlib.axis.YTick at 0x10f72efd0>],
 <a list of 5 Text yticklabel objects>)
```



In [11]:
```
from skimage import data
plt.imshow(data.lena())
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x1134fd310>
```



In []:

05_finding_patterns

July 22, 2014

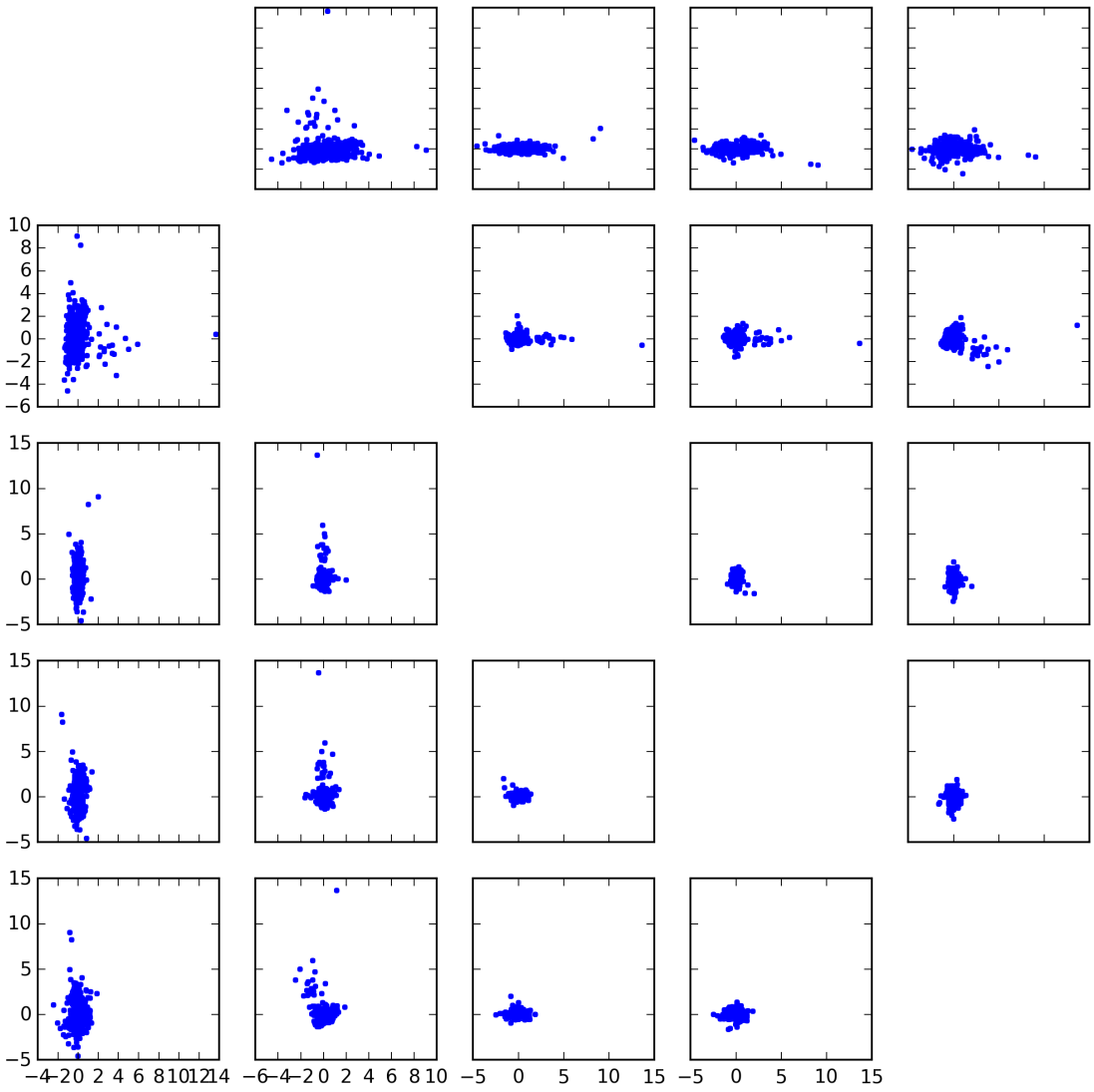
In [1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd

%matplotlib inline
matplotlib.rcParams['savefig.dpi'] = 300
```

In [2]:
```
df = pd.read_csv('../data/allen_brain_atlas.csv')
```

In [3]:
```
data = np.array(df.ix[:, :-2])
data = data - data.mean(0)
```

In [4]:
```
corr = np.dot(data.T, data)/data.shape[0]
evals, evecs = np.linalg.eig(corr)
```

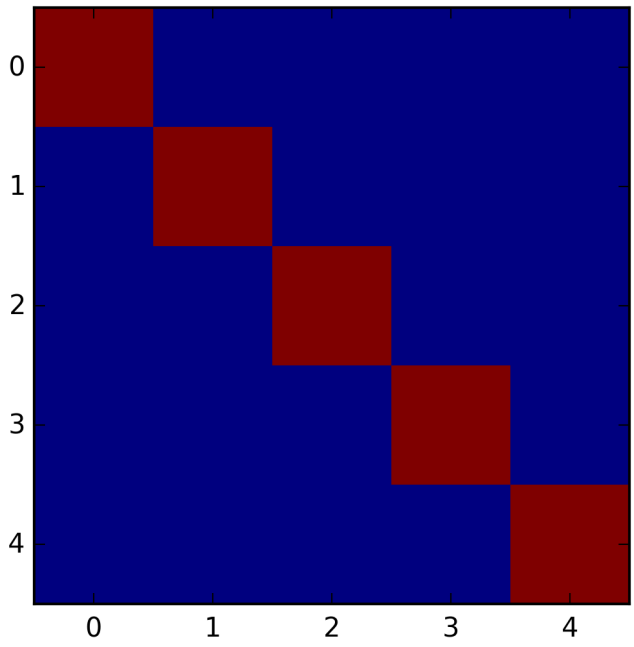
In [5]:
```
pcs = np.dot(data, evecs[:, :])
fig, axes = plt.subplots(5,5, sharex='col', sharey='row', figsize=(10,10))
for i in range(5):
    for j in range(5):
        ax = axes[i,j]
        if i==j:
            ax.set_axis_off()
        else:
            ax.plot(pcs[::10,i], pcs[::10,j], '.', ms=5)
```




```
In [6]: from sklearn import cluster
```

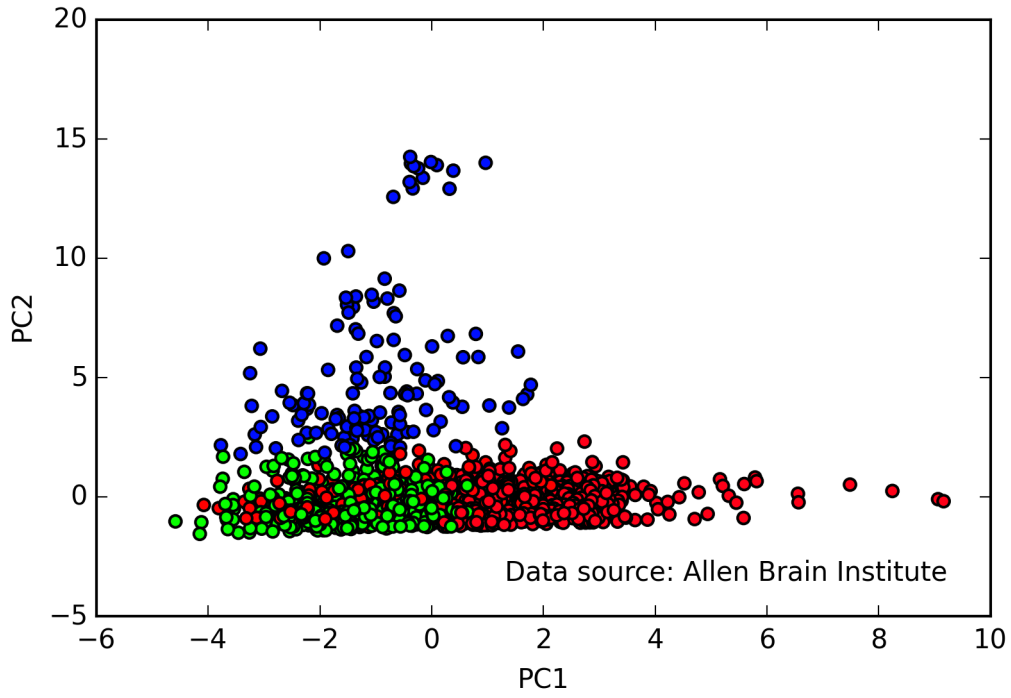
```
In [7]: W = evecs.dot(np.diag(1./np.sqrt(evals))).dot(evecs.T)
        whitened = np.dot(W, data.T)
        C_W = np.dot(whitened, whitened.T)/whitened.shape[1]
        plt.imshow(C_W, interpolation='nearest')
```

```
Out[7]: <matplotlib.image.AxesImage at 0x10b41a8d0>
```




```
In [8]: centers, labels, _ = cluster.k_means(whitened.T, 4)
        plt.scatter(pcs[:,0], pcs[:,1], c=labels, cmap='hsv')
        plt.xlabel("PC1")
        plt.ylabel("PC2")
        ax = plt.gca()
        plt.text(0.95, 0.05, 'Data source: Allen Brain Institute',
                 transform=ax.transAxes, va='bottom', ha='right')
```

```
Out[8]: <matplotlib.text.Text at 0x10c81b490>
```



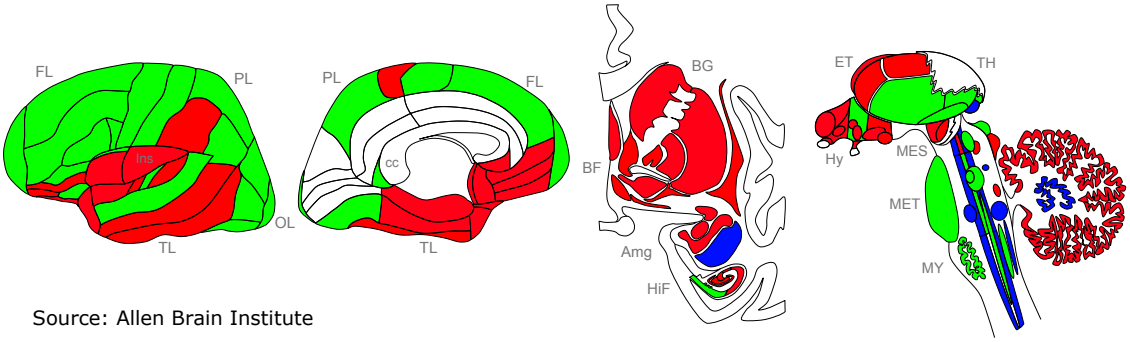

```
In [9]: from matplotlib import colors, cm
        def get_colors(labels, cmap):
            norm_labels = colors.Normalize()(labels)
            cmap_func = cm.get_cmap(cmap)
            class_colors = cmap_func(norm_labels)
            return class_colors
        class_colors = get_colors(np.unique(labels), 'hsv')
```

```
In [10]: from lxml import etree
         with file('../data/brain_atlas.svg') as fid:
             svg = etree.fromstring(fid.read())
         with file('../data/ontology.xml') as fid:
             ontology = etree.fromstring(fid.read())
         def color_svg(svg, ontology, data_ids, labels, colors):
             show = []
             unique_data_ids = np.unique(data_ids)
             paths = svg.xpath('//*[@structure_id]')
             ulabels = np.unique(labels)
             class_members = [data_ids[labels==l] for l in ulabels]
             n_members = np.array([len(c) for c in class_members])
             for p in paths:
                 sid = p.get('structure_id')
                 el = ontology.xpath('//structure[id={}]'.format(sid))[0]
                 descendants = []
                 for i in unique_data_ids:
                     if i==sid or el.xpath('descendant::structure[id={}]'.format(i)):
                         descendants.append(i)
                 if descendants:
```

```
                descendants = np.array(descendants)
                n_probes = np.array([np.sum(np.in1d(cl, descendants))
                                     for cl in class_members])
                frac = n_probes*1./n_members
                dominating = np.argmax(frac)
                r, g, b = (np.array(colors[dominating])*255).astype(int)[:3]
                p.set('style', 'fill: rgb({},{},{}); stroke: none'.format(r,g,b))
            else:
                p.set('style', 'fill: rgb(255,255,255); stroke: none')
        return svg

    from IPython.display import SVG
    new_svg = color_svg(svg, ontology, df.structure_id, labels, class_colors)
```

In [11]:
```
SVG(etree.tostring(new_svg))
```

Out[11]:



Source: Allen Brain Institute

06_making_maps_with_matplotlib

July 22, 2014

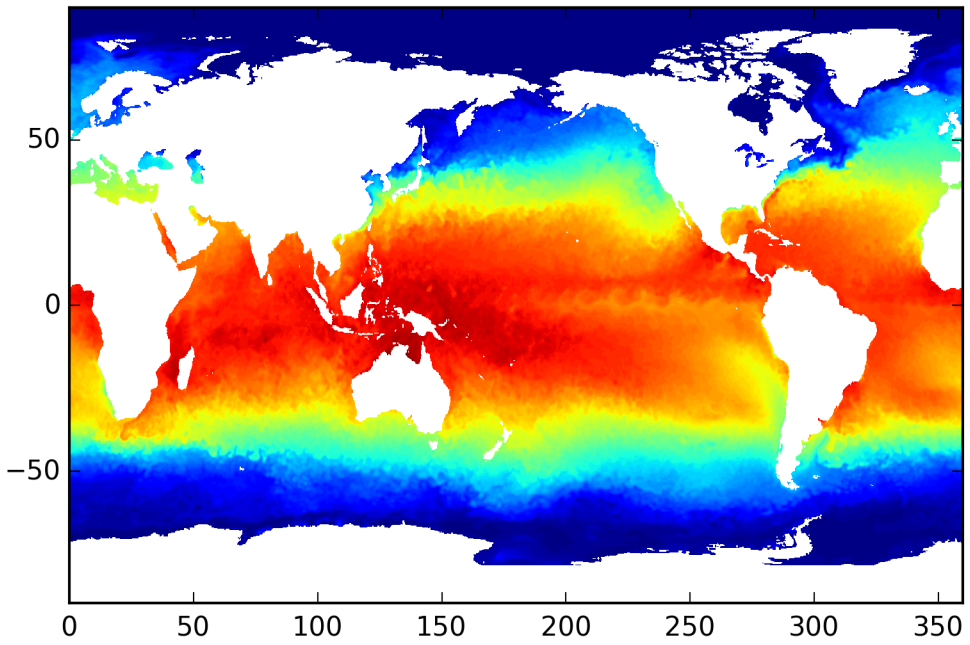
In [1]:
```
import cartopy
import matplotlib.pyplot as plt
%matplotlib inline

import matplotlib
matplotlib.rcParams['savefig.dpi'] = 300

import cartopy.crs as ccrs
import numpy as np
from matplotlib import mlab
from goodies import as_table
```

In [2]:
```
import netCDF4
dataset = netCDF4.Dataset('../data/maps/avhrr-only-v2.20140101.nc')
```

In [3]:
```
lon = dataset.variables['lon'][:]
lat = dataset.variables['lat'][:]
temp = dataset.variables['sst'][:].squeeze()
```

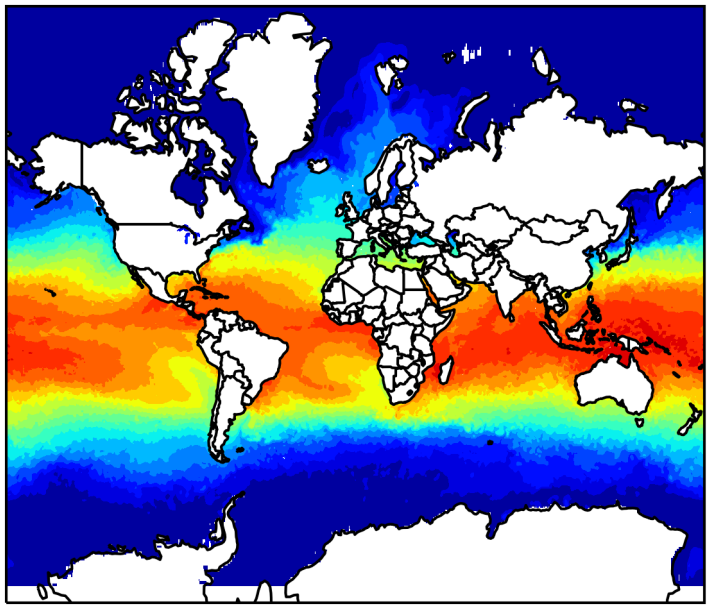
In [4]:
```
ax = plt.axes()
ax.contourf(lon, lat, temp, 100)
```

Out[4]: <matplotlib.contour.QuadContourSet instance at 0x10d619ea8>



In [5]:
```
ax = plt.axes(projection=ccrs.Mercator())
ax.coastlines()
ax.add_feature(cartopy.feature.BORDERS)
ax.set_global()
ax.contourf(lon, lat, temp, 20, transform=ccrs.PlateCarree())
```

Out[5]: <matplotlib.contour.QuadContourSet instance at 0x112cdb998>



In [6]:
```
shapes = cartopy.io.shapereader.Reader(
                '../data/maps/rgp_population/rgp_population')
pop = np.array([r.attributes['POPMUN2011']/r.attributes['AREA']
                for r in shapes.records()])
```

In [7]:
```
gl = cartopy.crs.Globe(ellipse='GRS80')
prj = cartopy.crs.LambertConformal(central_latitude=46.5,
                                   central_longitude=3.,
                                   secant_latitudes=(49, 44),
                                   false_easting=700000,
                                   false_northing=6600000)
```

In [10]:
```
from matplotlib import colors, cm
from cartopy.io.img_tiles import MapQuestOSM

imaging = MapQuestOSM()
ax = plt.axes(projection=imaging.crs)

ax.set_extent((2.0277,2.6662,48.7200,48.9987))
ax.add_image(imaging, 12)
```

```
pop_colors = cm.YlOrRd(colors.Normalize()(pop))

for c,g in zip(pop_colors, shapes.geometries()):
    ax.add_geometries(g, prj, facecolor=c, edgecolor='none', alpha=0.5)
```