Lumosity Case Study

September 29, 2014

1 Part 1 Landing Page Assessment

Given the following data regarding 3 different landing pages, including the number of visitors, number of free trial registrations, and the number of purchased accounts associated with those landing pages:

Landing Page #	# of Visitors	# of Registrations
1	998832	331912
2	1012285	349643
3	995750	320432

the goal is to answer the following questions: 1. Are any differences between the landing pages meaningful? 2. What are the estimated size of the differences between groups in terms of - Registration rate - Purchase rate 3. Which landing page should be made the new default based on the results and why? 4. Describe and justify the analyses you use to come to these conclusions.

1.1 Analysis motivation

As far as I can tell, this is a pretty standard A/B/n testing problem. The idea is to determine if the conversion rates for two separate landing pages are statistically different from one another. Conversion rates are based on one of two goals: free trial registration or purchased account. To do so, I'll perform a number of one-versus-other statistical tests where I assume one of the landing pages as the null hypothesis/default page and perform statistical significance test against each of the others in turn. For this problem, I'll assume that Landing Page #1 is the default page.

1.2 Analysis description

Under each goal, each of n_{page} visitors to a page can either convert (e.g. register for a free account), or not. Thus each visitor represents a binary trial, resulting in a binomial distribution with mean np, where p is the probability of conversion. To compare landing pages we estimate estimate p for each landing page using the sample mean $\hat{p}_{\text{page}} = c_{\text{page}}/n_{\text{page}}$, where c_{page} is the number of conversions for a landing page. However, we also need to estimate the confidence interval for this estimate and, in addition, the statistical significance in differences between estimated conversion rates.

Due to the large number of visitors for each of the pages, we can approximate the error distribution as a normal distribution, leading to the following definition of the confidence interval

$\hat{p} \pm z\sqrt{\frac{1}{n}\hat{p}(1-\hat{p})} = z\text{SE}$,

where z is the critical value for a normal distribution at a significance level of $p \leq \alpha$, and SE is the standard error for the normal approximation. Given that we are using the normal approximation for the error, then we can also use the standard error to determine statistically significant differences between two conversion rates by calculating the z-score:

$\text{Z} = \frac{\hat{p}_0 - \hat{p}_{\text{test}}}{\sqrt{\text{SE}_0^2 + \text{SE}_{\text{test}}^2}}$

The z-score is the number of standard deviations that need to exist between a Null and test hypotheses error distributions. From Z we can determine the p-value of rejecting the null hypothesis that two conversion rates are the same (i.e. $p < \alpha$) as the inverse cumulative distribution for the standard normal.

All of these analyses are in the function

```
In [71]: # FIGURES FOR EACH LANDING PAGE
         page1 = [998832., 331912., 18225.]
         page2 =  [1012285., 349643., 18531.]
         page3 = [995750., 320432., 18585.]

         # COMPARING THE FREE TRIAL CONVERSION RATE FOR PAGES 1 AND 2
         ab_test(page1[:2],page2[:2],['Landing Page 1', 'Landing Page 2'])
         title('Figure ' + str(fig_cnt) + ': Comparing free trial conversions (Pages 1 and 2)'); fig_cn
```

```
Condition 1: conversion rate: 33.23 +/- 0.12% (99.0% confidence interval)
Condition 2: conversion rate: 34.54 +/- 0.12% (99.0% confidence interval)
Percent difference: 3.9421%
abs(Z-score): 19.63 | p-value: 0.000000000
```

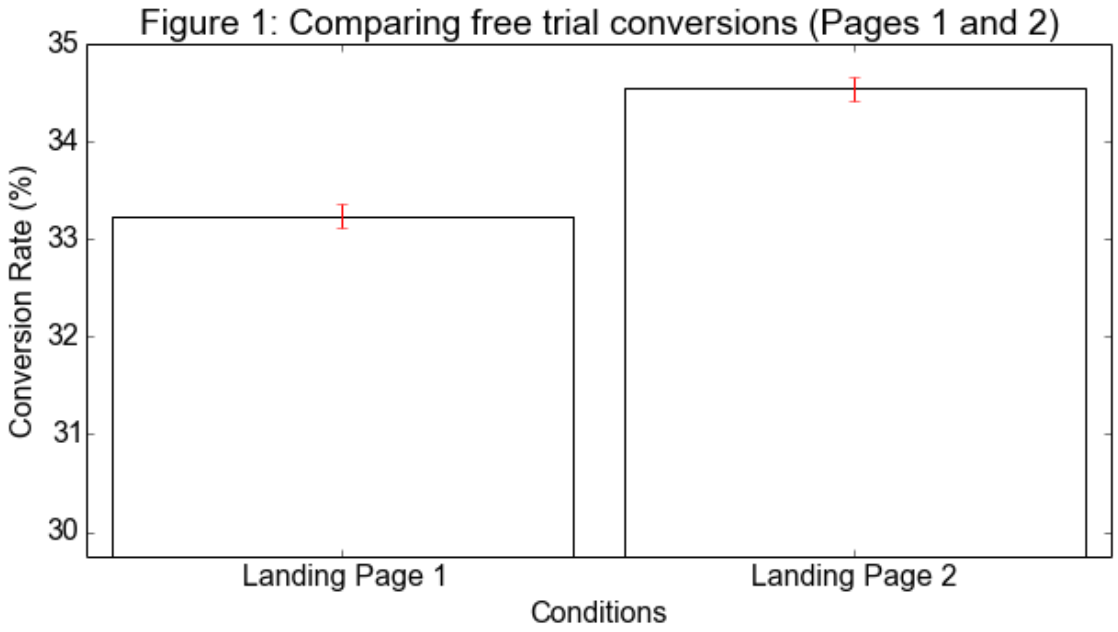


Figure 1 and associated std output display the results for comparing conversion rates on free trial registrations between Landing Page 1 and Landing Page 2. The conversion rates for page 1 and 3 are 33.23% and 34.54%, respectively. The red error bars plot the confidence interval on those conversion rate estimates at 99% confidence. We see that Landing Page 2 has a higher conversion rate for free trial registrations than Landing Page 1 (3.94% higher). The confidence intervals are nowhere near overlapping, suggesting these two values converion rates are significantly different. This notion is further supported by the near-zero p-value.

```
In [72]: # COMPARING THE FREE TRIAL CONVERSION RATE FOR PAGES 1 AND 3
         ab_test(page1[:2],page3[:2],['Landing Page 1', 'Landing Page 3'])
         title('Figure ' + str(fig_cnt) + ': Comparing free trial conversions (Pages 1 and 3)'); fig_cn
```

```
Condition 1: conversion rate: 33.23 +/- 0.12% (99.0% confidence interval)
Condition 2: conversion rate: 32.18 +/- 0.12% (99.0% confidence interval)
```

```
Percent difference: -3.1599%
abs(Z-score): 15.81 | p-value: 0.000000000
```

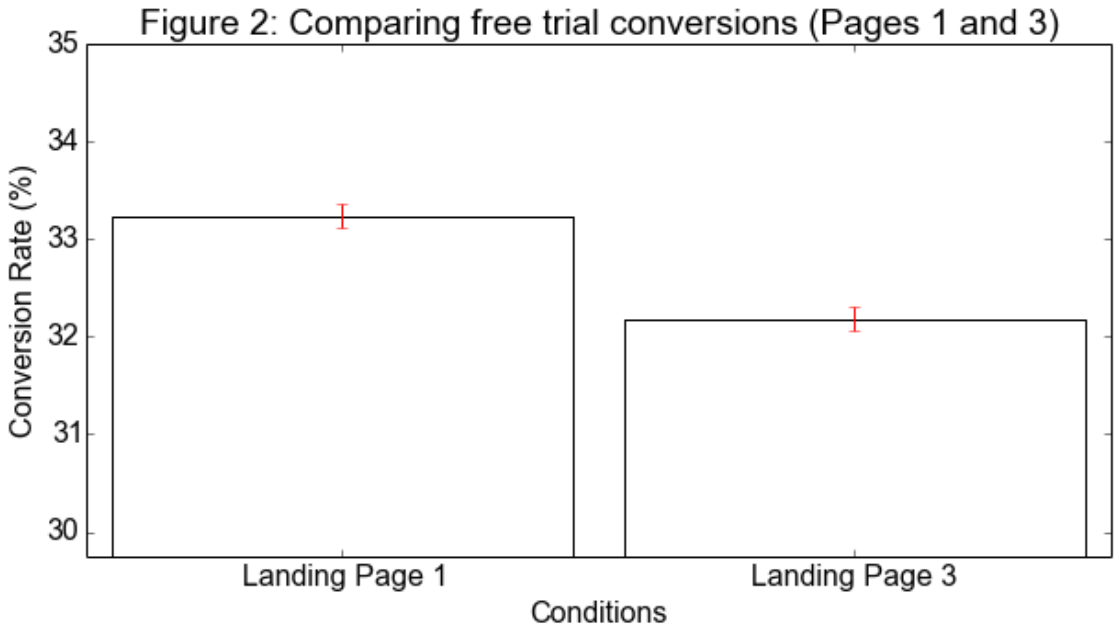


Figure 2 and associated std output display the results for comparing conversion rates on free trial registrations between Landing Page 1 and Landing Page 3. We see that Landing Page 3 has a lower conversion rate for free trial registrations (32.18%) than Landing Page 1 (3.16% lower). The confidence intervals are again nowhere near overlapping, suggesting these two values converion rates are significantly different. Similarly, the near-zero p-value gives further evidence that Landing Page 3 has significanly lower conversion rate for free trial registrations.

Take-home: For free trial registrations, Landing Page 2 has a higher conversion rate than Landing Page 1, while Landing Page 3 has has the smallest conversion rate.

```
In [73]: # COMPARING THE PAID ACCOUNT CONVERSION RATE FOR PAGES 1 AND 2
         ab_test(page1[::2],page2[::2],['Landing Page 1', 'Landing Page 2'])
         title('Figure ' + str(fig_cnt) + ': Comparing free to paid conversions (Pages 1 and 2)'); fig_

Condition 1: conversion rate: 1.82 +/- 0.03% (99.0% confidence interval)
Condition 2: conversion rate: 1.83 +/- 0.03% (99.0% confidence interval)
Percent difference: 0.3277%
abs(Z-score): 0.32 | p-value: 0.375796380
```

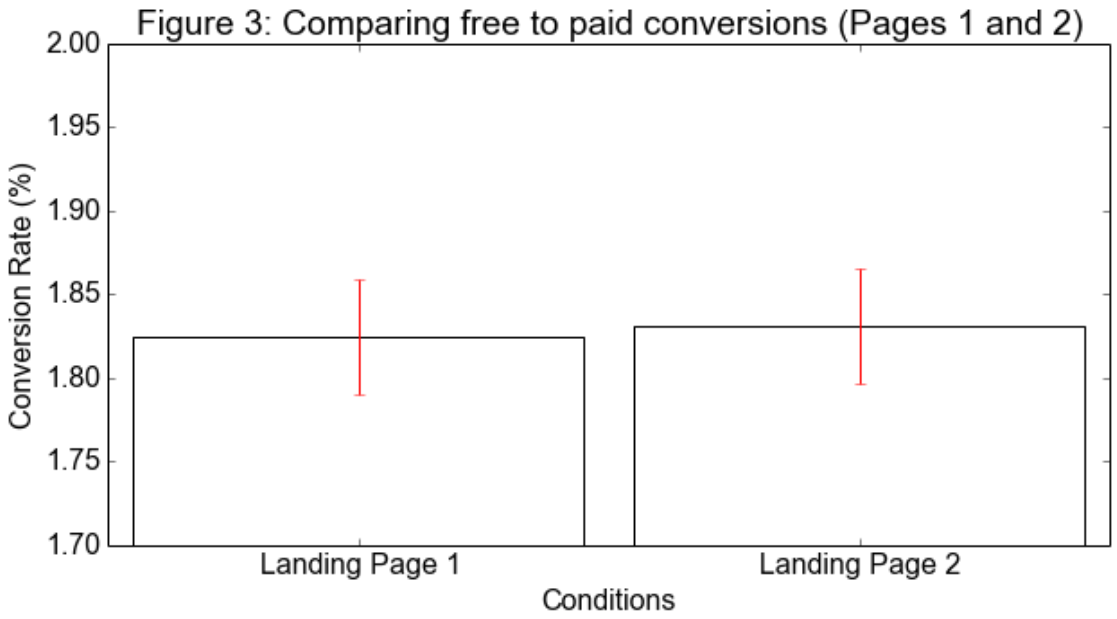


Figure 3 and associated std output display the results for comparing conversion rates for purchased accounts between Landing Page 1 and Landing Page 2. We see that Landing Page 2 has only slightly higher conversion rate (at 1.83%) than Landing Page 1 (1.82%), with a difference of 0.33 (of Landing Page 1's conversion rate). The convidence intervals show a large degree of overlap, suggesting these two values converion rates are not significantly different. This notion is further supported by the the large p-value (p=0.38).

```
In [74]: # COMPARING THE PAID ACCOUNT CONVERSION RATE FOR PAGES 1 AND 3
         ab_test(page1[::2],page3[::2],['Landing Page 1', 'Landing Page 3'])
         title('Figure ' + str(fig_cnt) + ': Comparing free to paid conversions (Pages 1 and 3)'); fig_
```

```
Condition 1: conversion rate: 1.82 +/- 0.03% (99.0% confidence interval)
Condition 2: conversion rate: 1.87 +/- 0.03% (99.0% confidence interval)
Percent difference: 2.2909%
abs(Z-score): 2.19 | p-value: 0.014148921
```

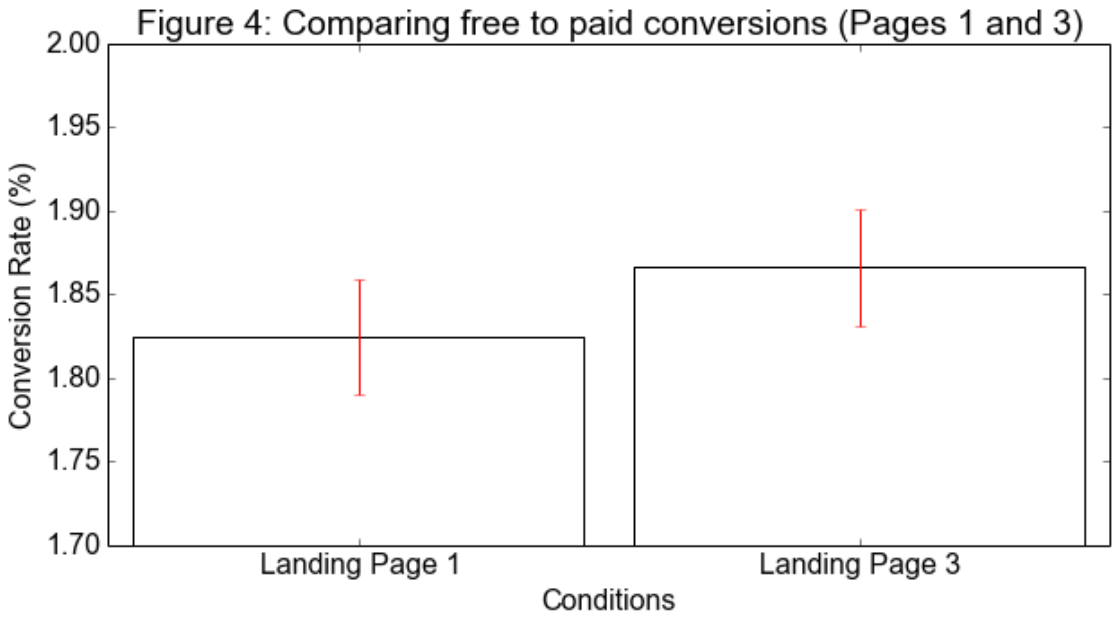


Figure 4 and associated std output display the results for comparing conversion rates for purchased accounts between Landing Page 1 and Landing Page 3. We see that Landing Page 3 has a higher conversion rate (1.87%) than Landing Page 1, an increase of 2.30% compared to Landing Page 1. The confidence intervals overlap somwhat, but not completely, suggesting these two values converion rates may or may not be significantly different. The p-value for the hypothesis test is p=0.014, indicating fairly strong statistical significance in the differences between the two conversion rates.

Take home: For paid accounts, Landing Page 3 has a higher conversion rate than Landing Page 1, which has a neglibly different conversion rate than Landing Page 2. All of the results are summarized in the table below.

Hypothesis Testing H_0 = Landing Page 1

	Landing Page #	# of Registrations	# of Purchases
2		Larger (+3.94%), $p = 0$	˜Same, $p = 0.34$
3		Smaller (-3.16%), $p = 0$	Larger (+2.29%), $p = 0.014$

1.3 Which landing page should we use?

In terms of which Landing Page should be used, it depends on the goal of the landing page. If the goal is obtain additional user data that can be used to improve products, perhaps more free trials are desired. In this case Landing Page 2 would be the preferred page. However, generally the bottom line is to generate revenue, which does not come from free trials, but from paid accounts. In this case, Landing Page 3 should be adopted. Either way, Landing Page 1 should not be used.

2 Part 2: Improving Training

The goal if this exercise is to brainstorm potential variables that could be introduced into the Lumosity training program that increase the probability of a user playing games that they find engaging and/or more effective.

During the training system, Lumosity uses a pre-training check-in, where the user is asked to report their current mood and the amount of sleep they've had the previous night. This point offers the obvious opportunity to introduce new variables that may me helpful for determining cognitive and physiological states. For example one could ask the length of time since the user had eaten their last meal. However, from my experience with the product, there does not appear to be a post-training check-out (other than ranking games). Measuring cognitive factors after training would also be helpful for determining the degree to which a user was engaged or the effectiveness of the training session. As an example, it is reasonable to assert that games which make a user feel less confident about their performance (perhaps a game is frustrating or too difficult) are less engaging, or less effective. Therefore each user could be asked to report their post-training confidence level during a training check-out. The confidence level would be reported on a ordinal scale such as a 5-interval Likert axis:

Not Confident < Somewhat Unconfident < Undecided < Somewhat confident < Very Confident ,

These confidence reports would then be associated with all games that were presented during the training session and then used to determine the probability that those games are presented to the user: more confidence-inducing games should be shown more often.

In order to test the effectiveness of this type of confidence-based scheduling (CBS), we could run the following experiment. Users are randomly selected to be in one of two groups: the experimental group, who will experience CBS, and the control group that will experience standard game scheduling. Neither group will know that they are part of the experimental or control groups. Before the test period in which CBS is used by the experimental group, the performance metrics for users in each group are assessed and used as a baselines. After the test period in which CBS is used by the experimental group and standard scheduling is used for the control group, the performance metrics are then re-assessed. We can then run statistical tests to determine if there is a significant difference in the change in performance score from baseline between the experimental and control groups during the testing period. If the difference in performance for the experimental group is significantly larger than the for the control group, it would support adopting confidence-based scheduling into the training system.

3 Part 3: Modeling Smartest Cities Data

In this exercise, the goal is to build a model that predicts median overall Brain Areas score for each city (Metro/Micro-politan areas) in Lumosity's Smartest Cities dataset from socioeconomic attributes sampled from the American Community Survey dataset. To do so, I'll perform a number of tasks including: loading, cleaning, and joining the two datasets, selecting attributes to include as features in the model, feature exploration and dimensionality reduction, model fitting, and model evaluation. For the data munging and joining, I'll use python's pandas package, and for model fitting, I'll use python's scikits-learn package. Below I perform some intiial data wrangling using pandas.

```
In [75]: import numpy as np
         import pandas as pd
         import os

         # CONFIGURE PANDAS DISPLAY WIDTH
         pd.options.display.max_colwidth = 100

         ## IMPORT DATA
         # SMARTEST CITIES SCORES
         data_home = "/home/dustin/jobs/lumosity/data_scientist_case"
         scores = pd.read_csv(os.path.join(data_home,'scs_2013.csv'))
         scores = scores.drop(scores.columns[9:],axis=1) # REMOVE RANKING DATA

         # AMERICAN COMMUNITY SURVEY DATA
         survey = pd.read_csv(os.path.join(data_home,'ACS_12_3YR_S0201_with_ann.csv'))
         survey_meta = survey.ix[0] # COLUMN TRANSLATIONS IN FIRST ROW
```

```
        # RENAME 'GEOID.ID2' TO 'FIPS' FOR JOIN
        survey = survey.drop(survey.index[0])
        survey = survey.rename(columns={'GEO.id2':'fips'})
        survey['fips'] = survey['fips'].astype(int)

        # GET RID OF NANS IN GENERAL
        survey = survey.dropna(axis=0,how='any')

        # SANITY CHECKS ON DATA SIZES
        print "Size of scores dataset: " + str(scores.shape)
        print "Size of survey dataset: " + str(survey.shape)
        print "Number of overlapping observations: " + str(len(np.intersect1d(scores.fips,survey.fips))
```

```
Size of scores dataset: (478, 9)
Size of survey dataset: (102, 613)
Number of overlapping observations: 99
```

4 Exploring and selecting survey data

From the output above, it appears that we have 99 training observation and over 600 attributes. This means that I'm going to need to trim down the attributes significantly and/or incorporate regularization during the model estimation. First, I'll see if I can remove or collapse any of the attributes.

At a first glance, I see that half of the attributes in the survey data are margin of error fields. Outside of possibly being used for a weigthing factor in a reweighted regression, I can't think of an obvious way to use these MOE attributes. Thus, I'll drop them.

```
In [76]: # FIRST, REMOVE MARGIN OF ERROR FIELDS
         import re
         cols = survey.columns.values
         moes = [c for c in cols if re.match('MOE',c)];
         survey = survey.drop(moes,axis=1)
         survey_meta = survey_meta.drop(moes)
```

We'll also do some additional cleaning of the survey data: getting rid of *** and '(X)' fields (these will be replaced with median value along columns), converting strings to floating point values.

```
In [77]: # REPLACE BAD ENTRIES WITH NANS
         survey = survey.replace("*****", NaN)
         survey = survey.replace("(X)", NaN)
```

```
In [78]: # CONVERT NUMERICAL VALUES TO FLOAT
         for c in survey.columns[5:]:
             survey[c] = survey[c].astype(float)
```

```
In [79]: # REPLACE NaN WITH MEDIAN COLUMN / FIELD
         survey = survey.fillna(survey.median())

         # SANITY CHECK TO SEE IF ANY NANS REMAIN
         null_idx = pd.isnull(survey).any(1).nonzero()[0]
         print null_idx
         print survey.shape
```

```
[]
(102, 309)
```

Ah, that's a little better, now let's take a look a the remaining 300 or so attributes...

```
In [80]: # TAKE A LOOK AT THE REMAINING SURVEY VALUES
         #for i, c in enumerate(survey.columns):
         #    print '-'*50
         #    print 'Attribute: ' + survey_meta[i]
         #    print (survey[c].describe())
         #    print survey[c]
```

Looking at the individual attribute values (above, commented out), it appears that the survey data table contains a combination of population counts (integers), percentages (floating points in 0-1), and income values (integers). Some of these attributes are repeated, indicating (I assume) measurements from multiple years in 2010-2012. Lets' take a closer look. First, let's see what the repeated values are all about...

```
In [81]: # IDENTIFY REPEATED FIELDS
         from collections import defaultdict

         duplicates = defaultdict(list)
         for i, item in enumerate(list(survey_meta)):
             duplicates[item].append(i)

         # GET DUPLICATES AND THEIR INDICES
         duplicates = {k:v for k,v in duplicates.items() if len(v)>1}
         duplicates
```

```
Out[81]: {'Estimate; DISABILITY STATUS - With a disability': [116, 118, 120, 122],
          'Estimate; EMPLOYMENT STATUS - In labor force': [159, 167],
          'Estimate; EMPLOYMENT STATUS - In labor force - Civilian labor force': [160,
           168],
          'Estimate; EMPLOYMENT STATUS - In labor force - Civilian labor force - Employed': [161,
           169],
          'Estimate; EMPLOYMENT STATUS - In labor force - Civilian labor force - Unemployed': [162,
           170],
          'Estimate; EMPLOYMENT STATUS - In labor force - Civilian labor force - Unemployed - Percent o
           171],
          'Estimate; INCOME IN THE PAST 12 MONTHS (IN 2012 INFLATION-ADJUSTED DOLLARS) - Median income
           235,
           237],
          'Estimate; MARITAL STATUS - Divorced': [67, 73, 79],
          'Estimate; MARITAL STATUS - Never married': [69, 75, 81],
          'Estimate; MARITAL STATUS - Now married, except separated': [65, 71, 77],
          'Estimate; MARITAL STATUS - Separated': [68, 74, 80],
          'Estimate; MARITAL STATUS - Widowed': [66, 72, 78],
          'Estimate; OCCUPATION - Management, business, science, and arts occupations': [181,
           187,
           193],
          'Estimate; OCCUPATION - Natural resources, construction, and maintenance occupations': [184,
           190,
           196],
          'Estimate; OCCUPATION - Production, transportation, and material moving occupations': [185,
           191,
           197],
          'Estimate; OCCUPATION - Sales and office occupations': [183, 189, 195],
          'Estimate; OCCUPATION - Service occupations': [182, 188, 194],
          'Estimate; PLACE OF BIRTH, CITIZENSHIP STATUS AND YEAR OF ENTRY - Female': [133,
```

```
 136,
 139,
 142],
'Estimate; PLACE OF BIRTH, CITIZENSHIP STATUS AND YEAR OF ENTRY - Male': [132,
 135,
 138,
 141],
'Estimate; SCHOOL ENROLLMENT - Percent enrolled in college or graduate school': [90,
 93],
'Estimate; SCHOOL ENROLLMENT - Percent enrolled in kindergarten to grade 12': [89,
 92],
'Estimate; SEX AND AGE - 18 years and over': [23, 30],
'Estimate; SEX AND AGE - 65 years and over': [26, 39],
'Estimate; SEX AND AGE - Female': [12, 29, 32, 35, 38, 41],
'Estimate; SEX AND AGE - Male': [11, 28, 31, 34, 37, 40],
'Id': [0, 3]}
```

Ok, my assumption wasn't completely correct. Some fields are repeated three times, while others are repeated twice, and others six times. It appears that there are sometimes subclasses of distinctions like, for example men and women based on the total population (EST_VC12/13), under the age of 18 (EST_VC33/34), and over the age of 65 (EST_VC49/50). Below I dive into this a little further...

```
In [82]: def display_duplicates_by_meta(meta):
             '''Helper function'''
             print meta
             print survey[survey.columns[duplicates[meta]]][:10]

In [83]: display_duplicates_by_meta('Estimate; SEX AND AGE - Female')
         display_duplicates_by_meta('Estimate; SEX AND AGE - Male')
```

```
Estimate; SEX AND AGE - Female
    EST_VC13  EST_VC34  EST_VC38  EST_VC42  EST_VC46  EST_VC50
1       51.6      49.2      52.2      50.5      51.3      57.4
2       51.2      49.1      51.8      49.4      51.0      57.6
3       50.8      49.0      51.4      49.3      51.2      55.8
4       51.3      48.7      52.0      49.7      51.0      57.7
5       51.3      49.0      52.2      50.4      52.0      57.4
6       51.3      48.8      52.1      49.3      52.2      56.8
7       49.9      48.9      50.2      49.1      49.8      55.9
8       48.5      49.0      48.3      45.6      48.6      54.8
9       51.8      49.1      52.7      50.6      52.1      58.0
10      50.9      48.9      51.6      49.8      51.2      57.1
Estimate; SEX AND AGE - Male
    EST_VC12  EST_VC33  EST_VC37  EST_VC41  EST_VC45  EST_VC49
1       48.4      50.8      47.8      49.5      48.7      42.6
2       48.8      50.9      48.2      50.6      49.0      42.4
3       49.2      51.0      48.6      50.7      48.8      44.2
4       48.7      51.3      48.0      50.3      49.0      42.3
5       48.7      51.0      47.8      49.6      48.0      42.6
6       48.7      51.2      47.9      50.7      47.8      43.2
7       50.1      51.1      49.8      50.9      50.2      44.1
8       51.5      51.0      51.7      54.4      51.4      45.2
9       48.2      50.9      47.3      49.4      47.9      42.0
10      49.1      51.1      48.4      50.2      48.8      42.9
```

Here I've displayed each of the duplicate values for SEX AND AGE - Female/Male fields. Interestingly, you can see the effect that women live longer than men (compare EST_VC50 to EST_VC49). This information is redundant however, as you can get one set of data by subtracting the other from one.

There are also other redundant fields that are not associated with repeats. For example, the proportion of the population that do no finish high school is redundant in light of the proportion do finish high school (i.e. 1 minus the proportion that finish high school).

Rather than trying to remove all of these redundant attributes by hand, I'll take a more automated approach. First I'll determine those attricbutes that are most correlated with the target value (median overall Brain Areas score), then select those sub attributes to be included in the feature set. Then, given these selected features, I'll remove redundancy by orthogonalization via statistical whitening (a la PCA).

5 Feature selection

Here I join on the two datasets (i.e. Luminosity scores and Survey information) to line up the compatible observations and determine the median overall score (MOS) for these observations.

```
In [84]: # JOIN TWO DATA SETS WHERE THEY OVERLAP ('fips' FIELD)
         data = pd.merge(scores,survey)
         all_brain_areas = ['Attention','Flexibility','Memory','Problem Solving','Speed']
         median_overall = data[all_brain_areas].median(axis=1)

         survey = data[data.columns[10:]]
         data = data[data.columns[:10]]
         survey_meta = survey_meta[2:]
```

In order to determine relevant attributes to include in the model, I calculate the correlation between each attribute and the MOS.

```
In [85]: # DETERMINE THE DEGREE OF CORRELATION
         # AMONGST ATTRIBUTES AND MEDIAN OVERALL SCORE
         ccs = []
         for col in survey.columns:
             try:
                 ccs.append(np.corrcoef(median_overall,survey[col].values)[0,1])
             except:
                 ccs.append(np.NaN)
         ccs = np.array(ccs)
```

Due to the fact that correlation coefficients are random variables (because the survey attributes and MOS's are random variables), we must determine a statistically significant level of correlation. This significance level will be the threshold for which we select our features.

It turns out that correlation coefficients are distributed according to a t-distribution with $N-2$ degrees of freedom, where N is the length of the vector used to calculate the correlation (in our case the number of observations: $N = 99$). Thus at a p-value $p < \alpha$, we can determine the significant level of correlation as $r_{\text{significant}} = \frac{t}{\sqrt{(N-2)+t^2}}$,

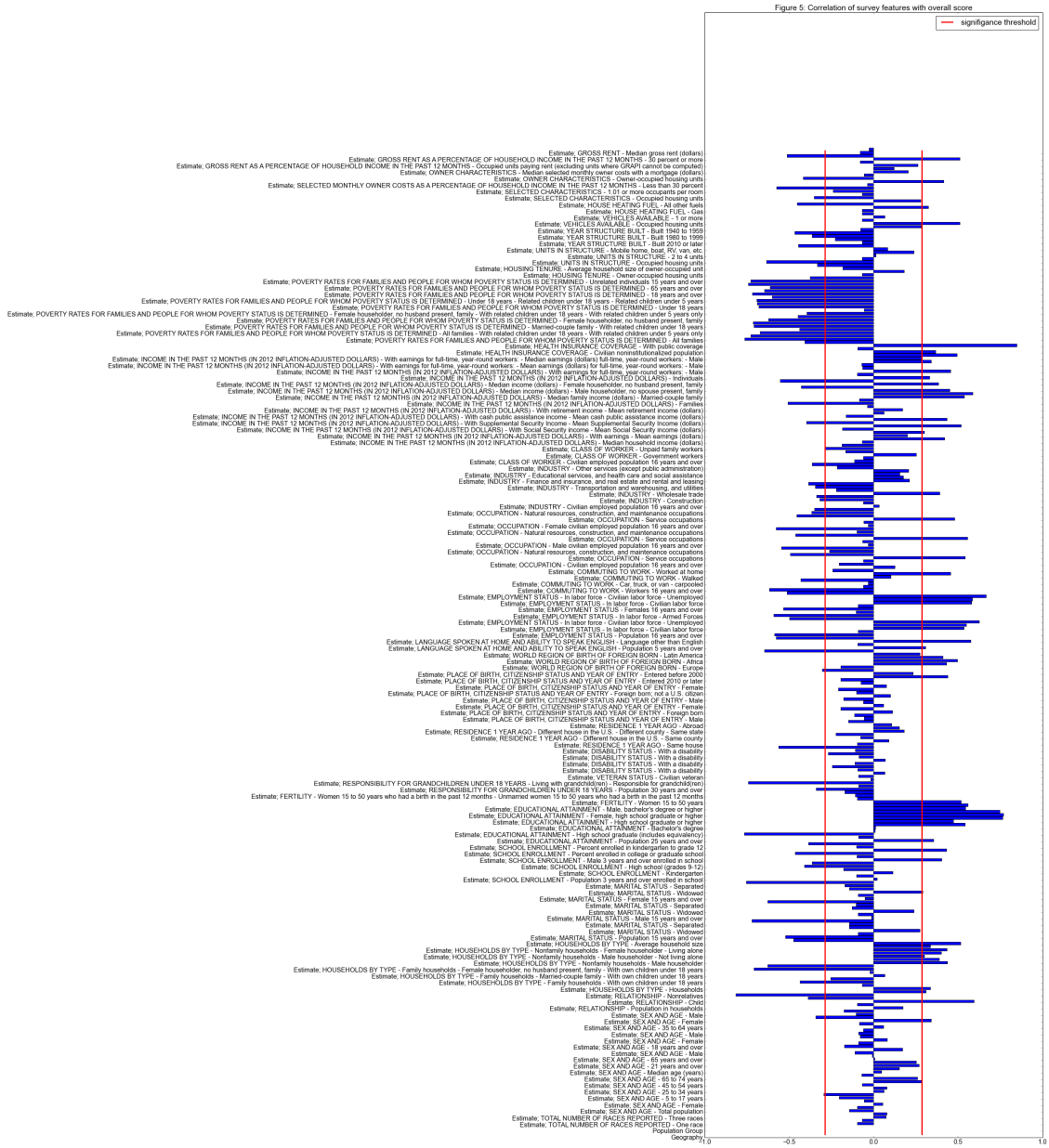
where t is the critical value for a student's t-distribution at $N-2$ degrees of freedom at a given α.

Below, I plot the degree of correlation for each attribute, and the statistical significance cuttoff $\pm r_{\text{significant}}$

```
In [86]: from matplotlib.pylab import find

         # DETERMINE SIGNIFICANT CORRELATION COEFFICIENT
         from scipy.stats import t

         # CORRCOEFF FOLLOWS T-DISTRIBUTION
```

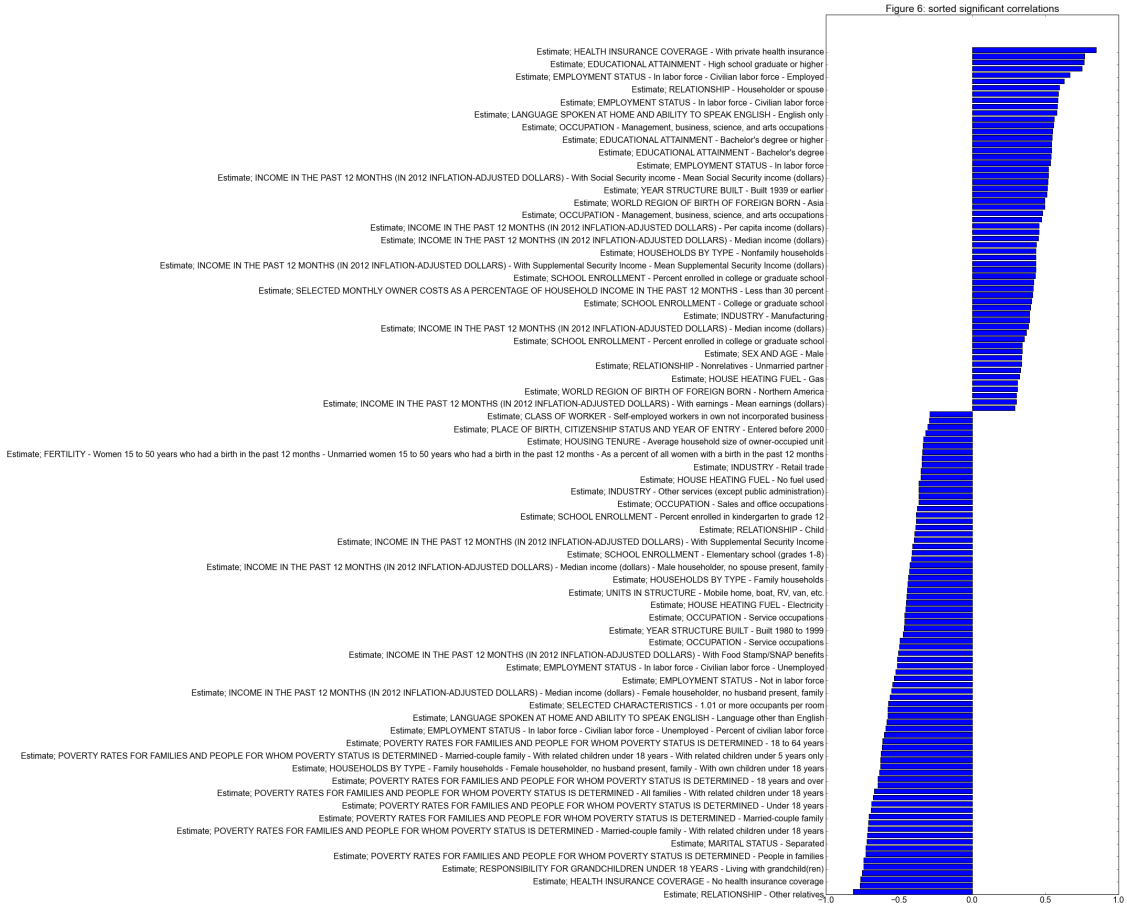
```
        # WITH N-2 DEGREES OF FREEDOM
        p_val = 0.001 # AMOUNT OF SIGNIFICANCE OF CORRELATION
        n_obs = 99
        df = n_obs - 2
        pval = p_val*2 # TWO-TAILED
        tval = t.ppf(1-pval,df)
        sigcc = tval / sqrt(df + tval**2)

        # CHOOSE FEATURES *ATLEAST* THIS CORRELATED
        keep_idx = find(abs(ccs) >= sigcc)

        # PLOT DISTRIBUTION OF CORRELATIONS
        xx = range(1,len(ccs)+1)
        plt.figure(figsize=(15,50))
        plt.barh(xx,ccs)
        plt.plot(np.ones_like(xx)*sigcc,xx,'r',linewidth=3)
        plt.plot(np.ones_like(xx)*-sigcc,xx,'r',linewidth=3)
        plt.yticks(range(0,len(ccs),2),survey_meta[survey.columns][::2])
        plt.title('Figure %d: Correlation of survey features with overall score' % fig_cnt); fig_cnt +=
        plt.legend(['signifigance threshold']);

        print "significant correlation: " + str(sigcc)
        print "# of selected features: " + str(len(keep_idx))
```

```
significant correlation: 0.28680549201
# of selected features: 135
```



Using this correlation criterion I've further reduced the number of parameters to around 130, another factor of nearly 2.3. Below I diplay the significantly-correlated features, sorted according to the degree of correlation:

```
In [87]: # SORT CORRELATIONS FOR DISPLAY
         sorted_idx = sorted(range(len(keep_idx)), key=lambda k: ccs[keep_idx][k])

         # PLOT SORTED CORRELATIONS AND ASSOCIATED FEATURES
         plt.figure(figsize=(10,30))
         plt.barh(range(len(sorted_idx)), ccs[keep_idx[sorted_idx]]);
         plt.yticks(range(0,len(sorted_idx),2),survey_meta[survey.columns[keep_idx[sorted_idx]]][::2])
         plt.title('Figure %d: sorted significant correlations' % fig_cnt); fig_cnt +=1
```



In the Figure 5 above, we see the features that are most correlated and anticorrelated with MOS. In general the most postively correlated features are associated with higher education, employment status, health insurance coverage. Those variables that are most anti-correlated with MOS are associated with poverty rate, divorce or separated marriages, single-mother households, and lack of health insurance coverage.

5.1 Feature extraction and preprocessing

Here, I extract the significantly-correlated features and transform the variables using the negative logarithm so that the strictly positive variables will be closer to normally-distributed. I also standardize each feature by centring it and rescaling it to have unit variance. This will will make the variables more ammenable to PCA and model fitting in later analyses.

In [88]:
```
# EXTRACT FEATURES (SORT THEM ACCORDING TO CORRELATION)
selected_features = survey[survey.columns[keep_idx[sorted_idx]]].values.astype(float)

# GAUSSIANIZE FEATURES
selected_features = -log(selected_features)

# STANDARDIZE EACH VARIABLE
selected_features -= selected_features.mean(0)
selected_features /= selected_features.std(0)

# DISPLAY THE MOST CORRELATED / SELECTED FEATURES
feature_names = survey_meta[survey.columns[keep_idx[sorted_idx[::-1]]]]
```

```
In [89]: # DISPLAY SELECTED, TRANSFORMED FEATURES
         new_figure()
         plt.set_cmap(cm.seismic)
         plt.imshow(selected_features, interpolation='none')
         plt.axis("tight")
         plt.colorbar()
         plt.title('Figure %d: Selected and Transformed Feature Matrix' % fig_cnt); fig_cnt+=1
```

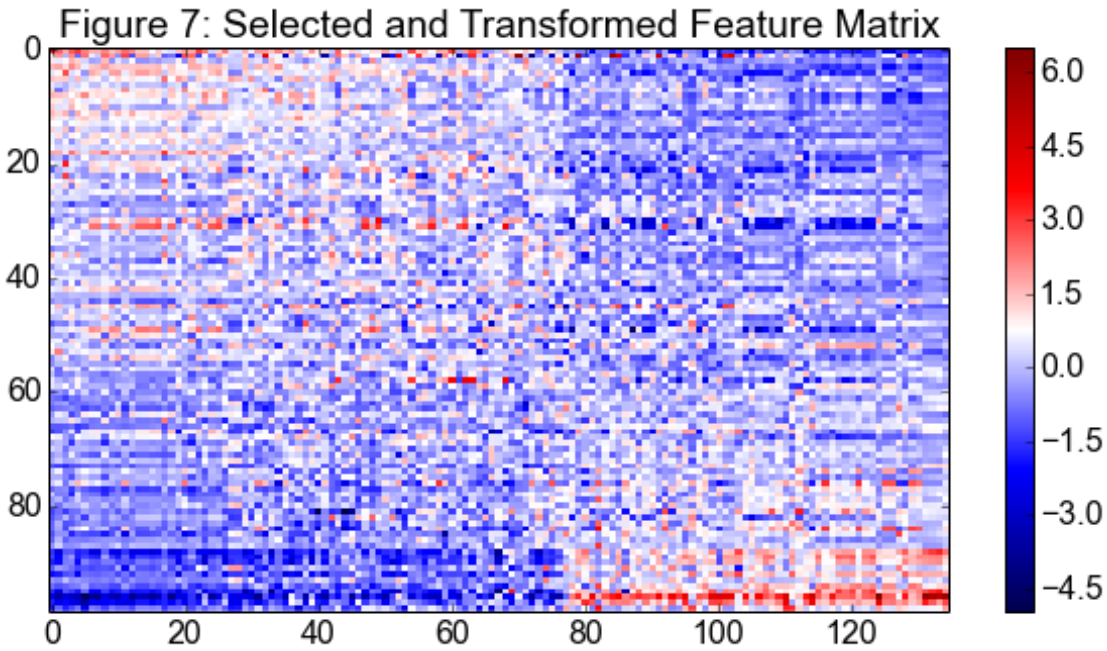
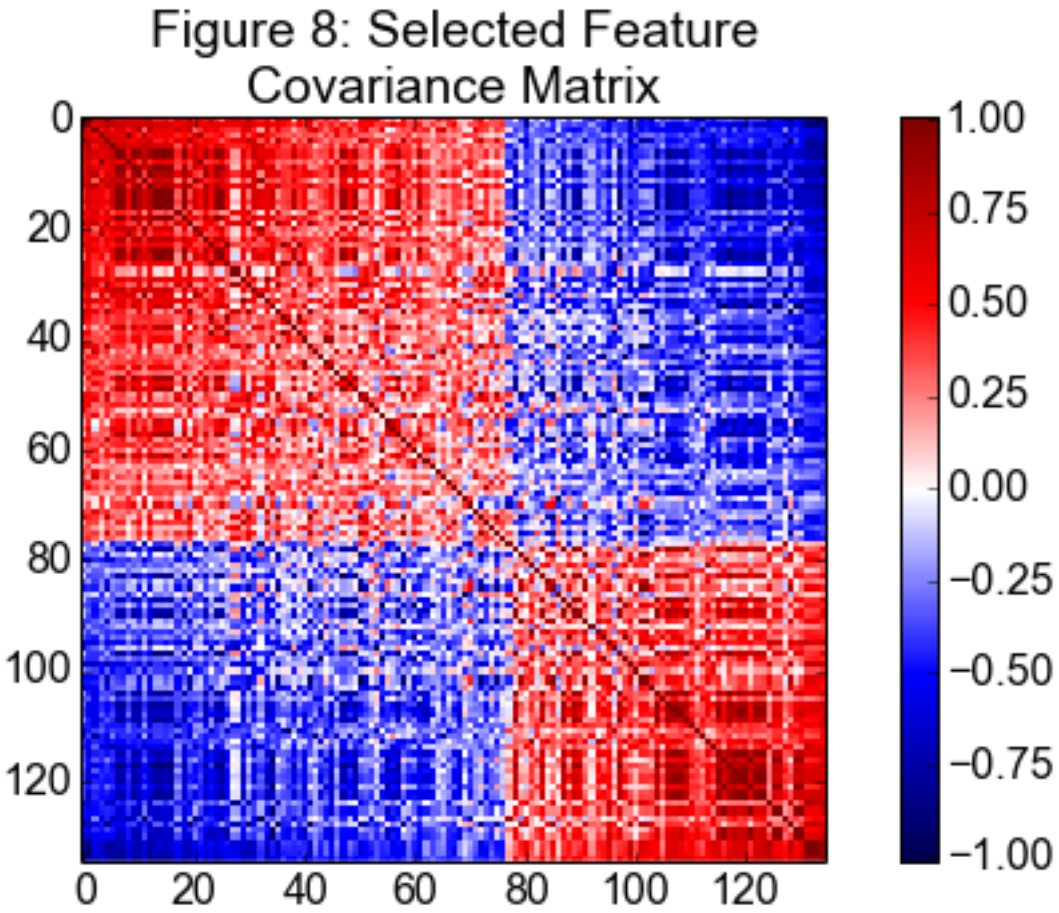


Figure 7 shows the transformed features for the observations. Note that the observations are currently sorted by their ranking in the Smartest Cities dataset. We can see the clear progression that the first 70 or so features have strong loadings for higher-ranking cities. These features have strong negative loading for low-ranking cities. Conversely the remaining features have strong positive loadings for low-ranking cities, but negative loadings for hihg-ranking cities.

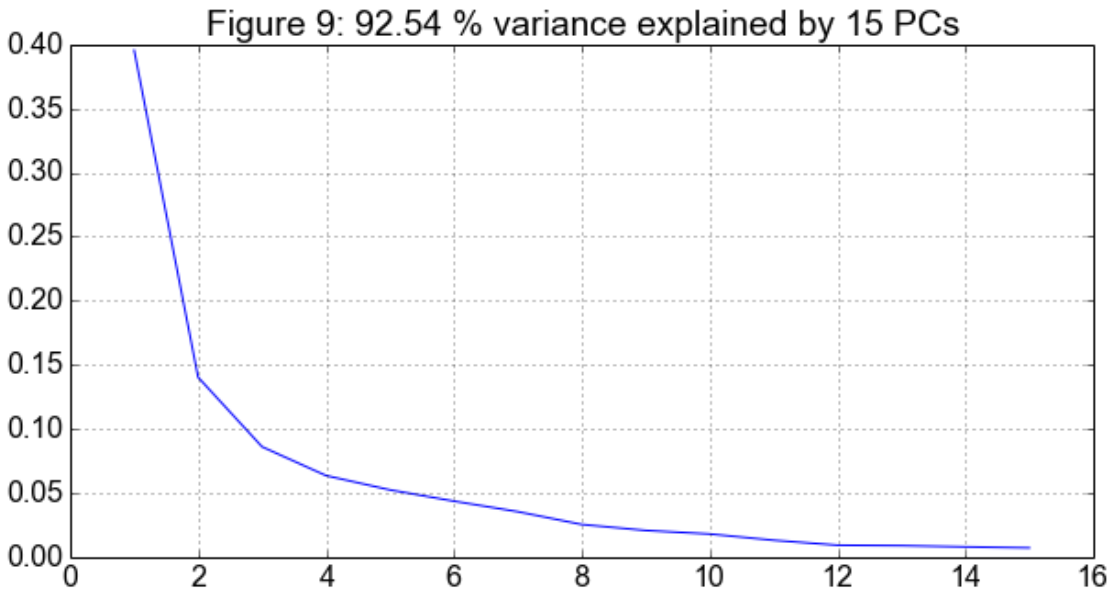
As I mentioned earlier, there appears to be a lot of redundancy in these features. Let's check to see the degree of covariance between the selected features...

```
In [90]: # FEATURE COVARIANCE MATRIX
         C = np.cov(selected_features.T)
         new_figure()
         plt.set_cmap(cm.seismic)
         plt.imshow(C, interpolation='none')
         plt.axis("image")
         plt.colorbar()
         plt.title('Figure %d: Selected Feature\nCovariance Matrix' % fig_cnt); fig_cnt+=1
```



Indeed, there is substantial structure in the covariance matrix, as an orthogonal set of features would look much more like the identity. We'll try to account for this redundancy as well as further reduce the dimensionality of the features by performing Principal Components Analysis (PCA) on the selected features. We PCA in order to keep the number of features that accound for at least 95 % of the variance in the feature matrix.

```
In [91]: # RUN PCA AND WHITENING TO ORTHOGONALIZE SELECTED FEATURES
         from sklearn.decomposition import PCA
         prct_explained = 0.925
         pca = PCA(n_components = prct_explained, whiten=True);
         pca.fit(selected_features);
         new_figure();
         plt.plot(arange(pca.n_components)+1, pca.explained_variance_ratio_);
         plt.grid("on")
         title('Figure %d: %1.2f %% variance explained by %d PCs' % (fig_cnt, sum(pca.explained_varianc
```



He we see that only 15 components are needed to explain a majority of the variance in the features. This greatly reduces the number of model parameters from the 135. Now, let's take a look at the features to ensure that they are indeed independent.

```
In [92]: # WHITEN SELECTED FEATURES
         w_features = pca.transform(selected_features)

         # VISUALIZE WHITENED FEATURE STATISTICS...
         new_figure()
         plt.set_cmap(cm.seismic)
         plt.imshow(w_features, interpolation='none')
         plt.axis("tight")
         plt.colorbar()
         plt.title('Figure %d: Whitened Feature Matrix' % fig_cnt); fig_cnt+=1

         C = np.cov(w_features.T)
         new_figure()
         plt.set_cmap(cm.seismic)
         plt.imshow(C, interpolation='none')
         plt.axis("image")
         plt.colorbar()
         plt.title('Figure %d: Whitened Feature\nCovariance Matrix' % fig_cnt); fig_cnt+=1
```

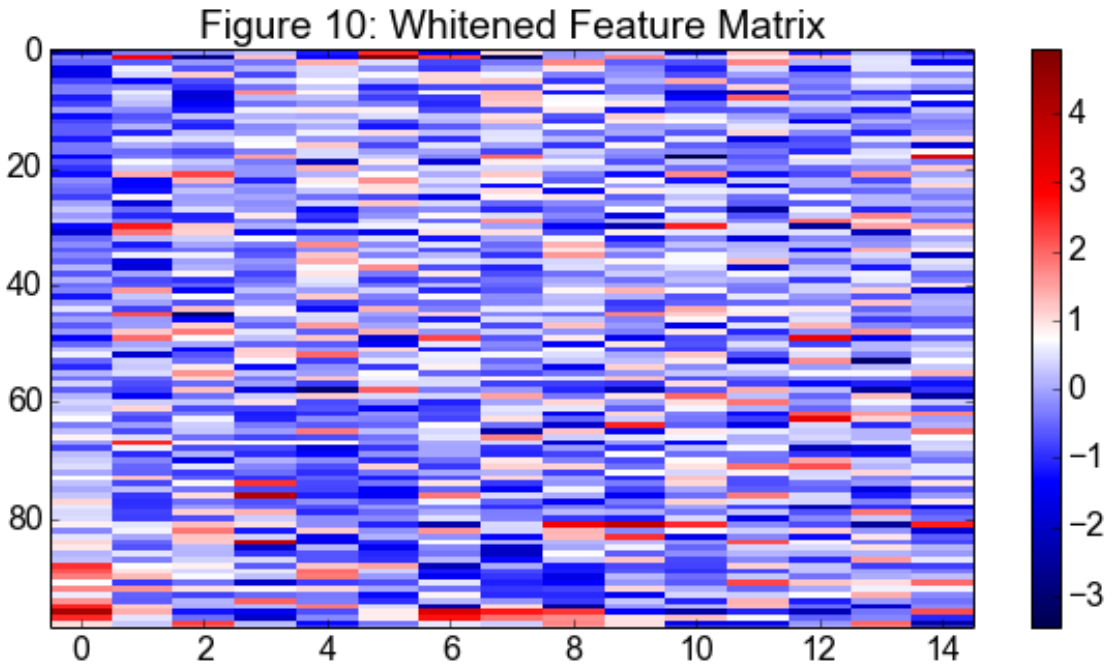


Figure 10: Whitened Feature Matrix
0
20
40
60
80
0
2
4
6
8
10
12
14
4
3
2
1
0
−1
−2
−3

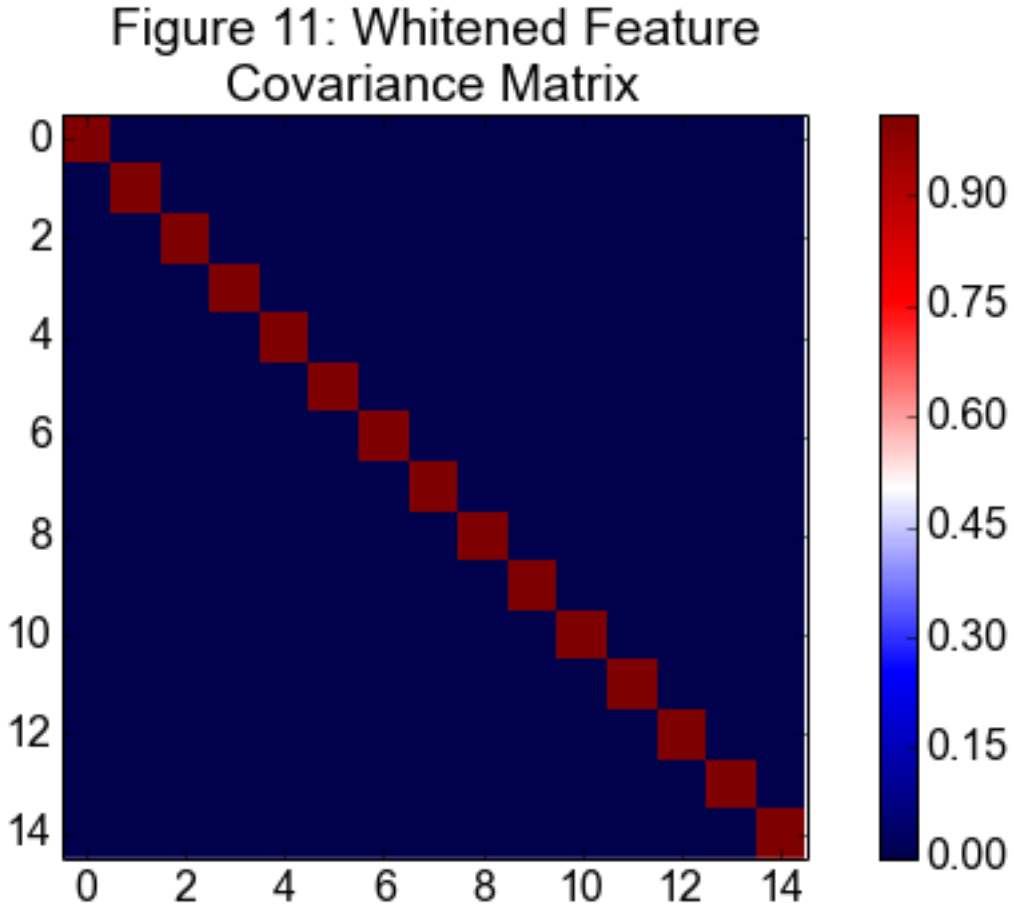
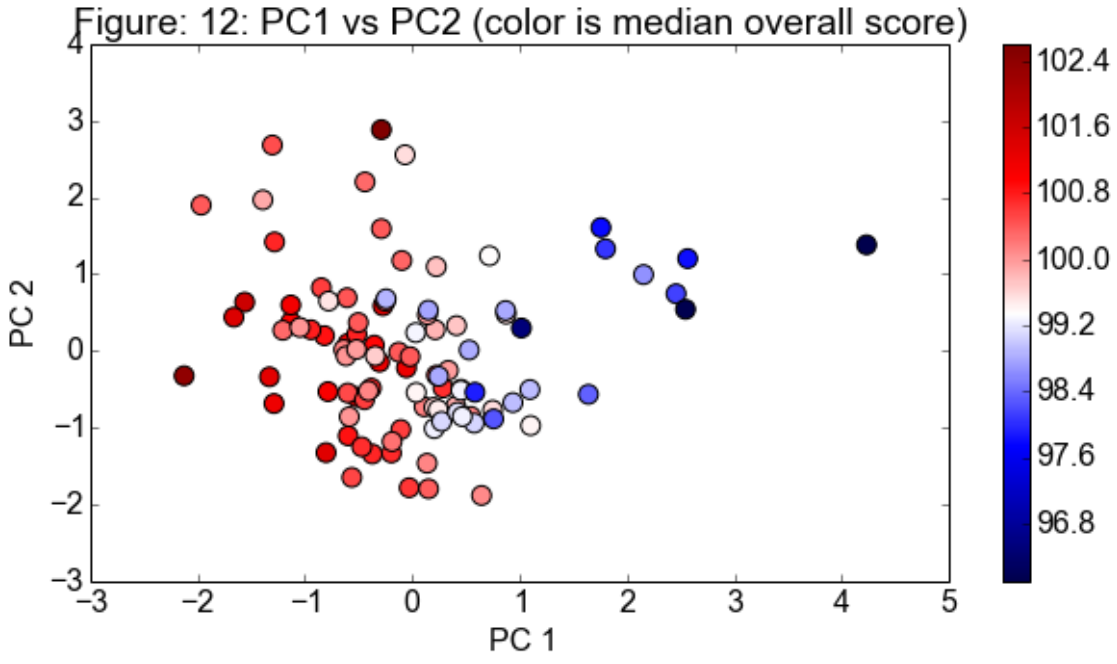


Figure 11: Whitened Feature Covariance Matrix
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14
0.90
0.75
0.60
0.45
0.30
0.15
0.00

Great! It looks like we've removed all the redundant information, as the covarianc of the principal features is essentially an identity matrix. Now, let's take a look at what information these orthogonal dimensions capture about the input features. We'll do this by analyzing how the observed features project onto the first few principal components (PCs).

```
In [93]: # VISUALIZE OBSERVATIONS IN PC SPACE
         new_figure()
         plt.set_cmap(plt.cm.seismic)
         plt.scatter(w_features[:,0],w_features[:,1],100,median_overall)
         plt.xlabel('PC 1'); plt.ylabel('PC 2')
         plt.colorbar()
         plt.title('Figure: %d: PC1 vs PC2 (color is median overall score)' % fig_cnt); fig_cnt += 1

         new_figure()
         plt.scatter(w_features[:,0],w_features[:,2],100,median_overall)
         plt.xlabel('PC 1'); plt.ylabel('PC 3')
         plt.colorbar()
         plt.title('Figure: %d: PC1 vs PC3 (color is median overall score)' % fig_cnt); fig_cnt += 1

         new_figure()
         plt.scatter(w_features[:,1],w_features[:,2],100,median_overall)
         plt.xlabel('PC 2'); plt.ylabel('PC 3')
         plt.colorbar()
         plt.title('Figure: %d: PC2 vs PC3 (color is median overall score)' % fig_cnt); fig_cnt += 1
```



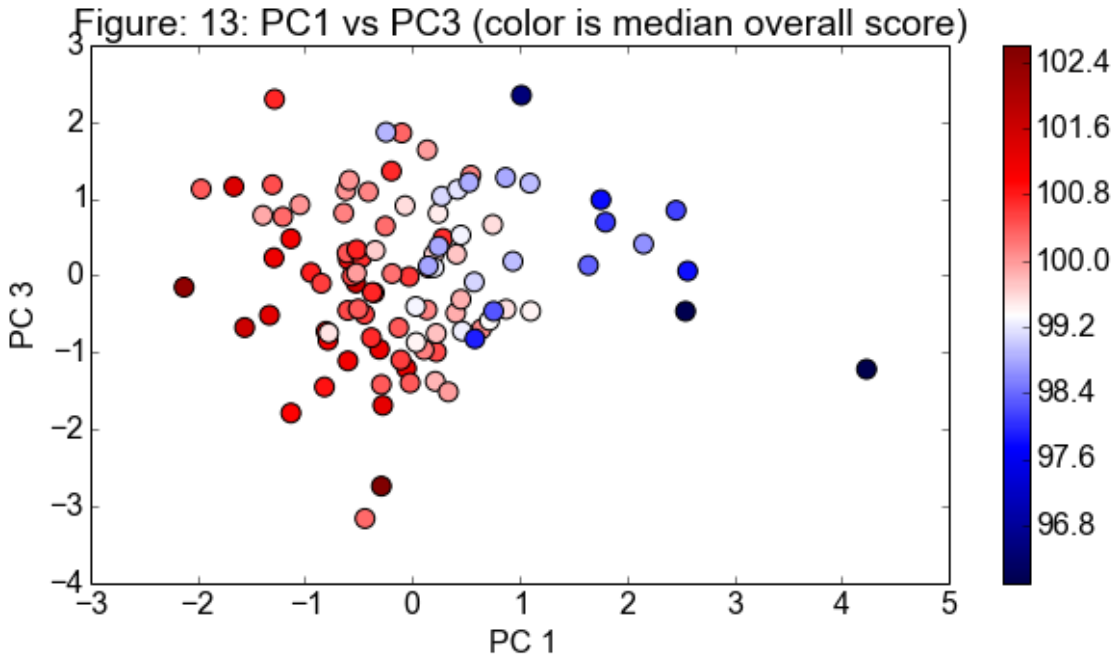


Figure: 13: PC1 vs PC3 (color is median overall score)

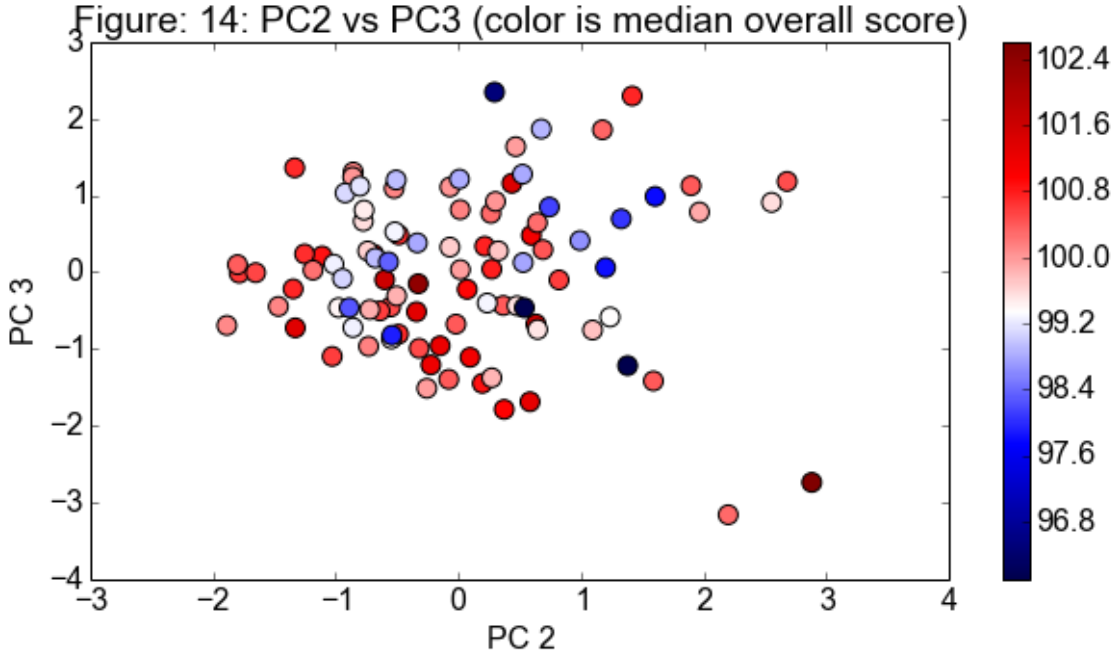


Figure: 14: PC2 vs PC3 (color is median overall score)

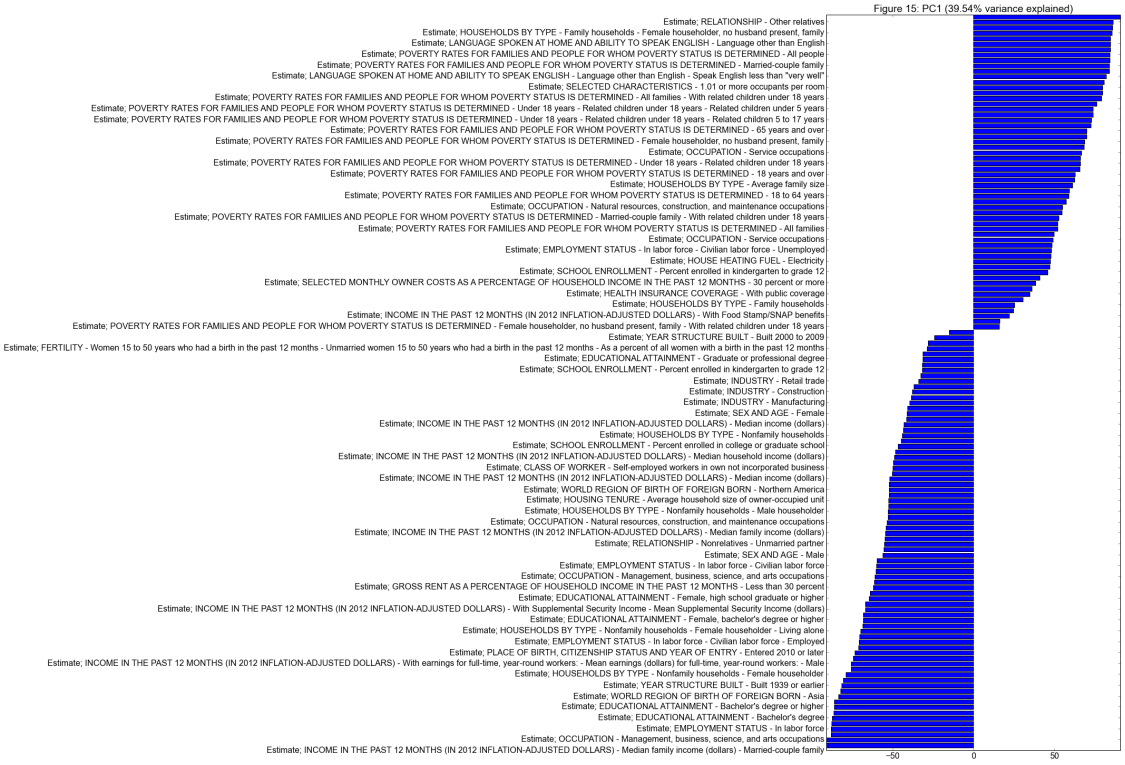
Each plot above depicts how the feature vector for each observation projects into the space spanned by two of the first three PCs. Each dot represents an observation (city) and the color of each dot corresponding to the observation's MOS (i.e. target variable).

From from these visualizations we can see that the MOS varies smoothly along the first principal dimension: low projections onto PC1 correspond to high MOS values while high projections onto PC1 correspond

with low MOS values. However, PC's 2 and 3 do not appear to be particularly correlated MOS, as can be seen by the random, overlapping organization of the colors in Figure 14. However, that is not to say that there are not other PCs that might better explain MOS.

There is obviously structured dependence between MOS and (at least) the first principal dimensions of the selected features. What does this tell us about the factors that are important for predicting MOS? To address this question, I rank the selected features in terms of their projection onto the first PC.

```
In [94]: # VISUALIZE REPRESENTATION ON FIRST FEW PCS IN TERMS OF INPUT FEATURES
         n_obs, n_features =  selected_features.shape

         ## PROJECT FEATURES ONTO 1ST PC AND VISUALIZE...
         pc1 = selected_features.T.dot(w_features[:,0:1])
         pc1_sort_idx = sorted(range(len(pc1)), key=lambda k: pc1[k])

         plt.figure(figsize=(10,25))
         plt.barh(arange(len(pc1_sort_idx))+.5,pc1[pc1_sort_idx])
         plt.axis("tight");
         plt.yticks(arange(0,n_features,2),feature_names.values[pc1_sort_idx[::2]])
         plt.title('Figure %d: PC1 (%1.2f%% variance explained)' % (fig_cnt,pca.explained_variance_ratio
         #print feature_names.values[pc1_sort_idx]
```



5.2 PC1 interpretation

Above, the projection of the features onto PC1 suggests that there is a postive relationships with MOS and education level, employment, income, having a relatively high-paying occupation (e.g. sciences, arts, or business), and being single without family or children. Conversely, there appears to be a negative relationship between MOS and poverty rate, non-English speaking households, service occupations, and single parents.

6 Model Fitting

Now that we've done some initial exploratory analysis, let's fit some models to our features in order to predict MOS. In order to ensure that our model generalizes to new data, we would like to have a testing set of cities to asses model accuracy outside of the fitting procedure. However, given that we do not have any testing data, we will have to sample it from the training data. Therefore, I'll use cross-validation to estimate many models, then average their weights to get a final model.

Also, given that we only have 99 observations, ensuring that problem is well constrained (i.e ensuring that the number of features we use is much smaller than the number of observations used to train the model) becomes tricky. Luckily, I was able to reduce the number of parameters down to 15 using the PCA. Otherwise, using some form of regularization would be necessary to constrain the ill-conditioned problem.

6.1 Linear regression

Here, I perform simple linear regression to predict MOS from the PC features, using 5-fold cross validation. I'll also include a LASSO penalty. The LASSO penalty prevents the model from overfitting to noise while also performing feature selection by encouranging a sparse distribution of model coefficients.

```
In [95]: from sklearn import linear_model
         from sklearn import cross_validation
         from sklearn.utils import shuffle

         # REMOVE MEAN FROM TARGET
         targets = median_overall - median_overall.mean()

         # MAKE SURE TO SHUFFLE
         features, targets = shuffle(w_features, targets)

         # CORSS-VALIDATE MODELS
         k_folds = 5
         kf = cross_validation.KFold(len(median_overall),n_folds=k_folds)

         coeffs = []
         r2 = []
         cnt = 1
         for train_idx, xval_idx in kf:
             print 'Fold %d / %d' % (cnt,k_folds)

             # CURRENT SAMPLES
             X_train, X_xval = features[train_idx], features[xval_idx]
             y_train, y_xval = targets[train_idx], targets[xval_idx]

             # INITIALIZE THE MODEL
             regr = linear_model.Lasso(alpha=0.1) # LASSO
             #regr = linear_model.LinearRegression()

             # TRAIN
             regr.fit(X_train, y_train)

             # X-VALIDATE
             r2_tmp = regr.score(X_xval, y_xval)

             # EXPLAINED VARIANCE
             print('Variance explained: %.2f' % r2_tmp)
             r2.append(r2_tmp)
```

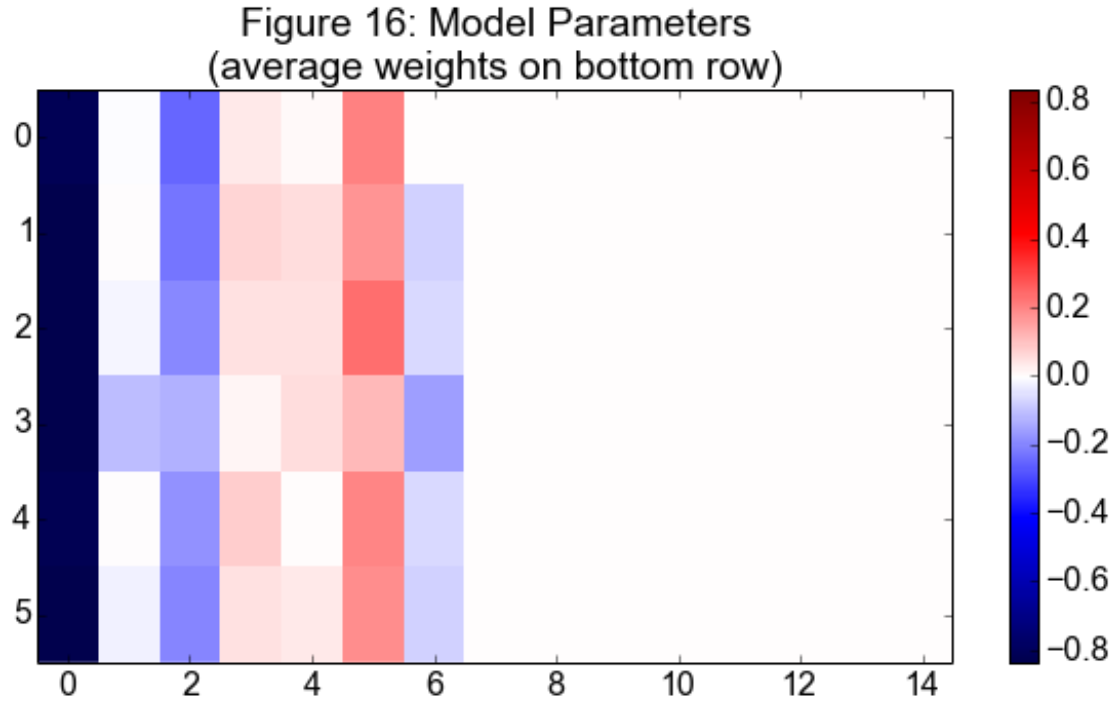
```
            # MODEL PARAMETERS
            coeffs.append(regr.coef_)
            cnt += 1

        coeffs = np.vstack(coeffs)
        avg_r2 = np.array(r2).mean()
        print '-'*50
        print ('Average explained variance: %1.2f (correlation: %1.2f)' % (avg_r2, np.sqrt(avg_r2)));
        maxx = abs(coeffs).max()
```

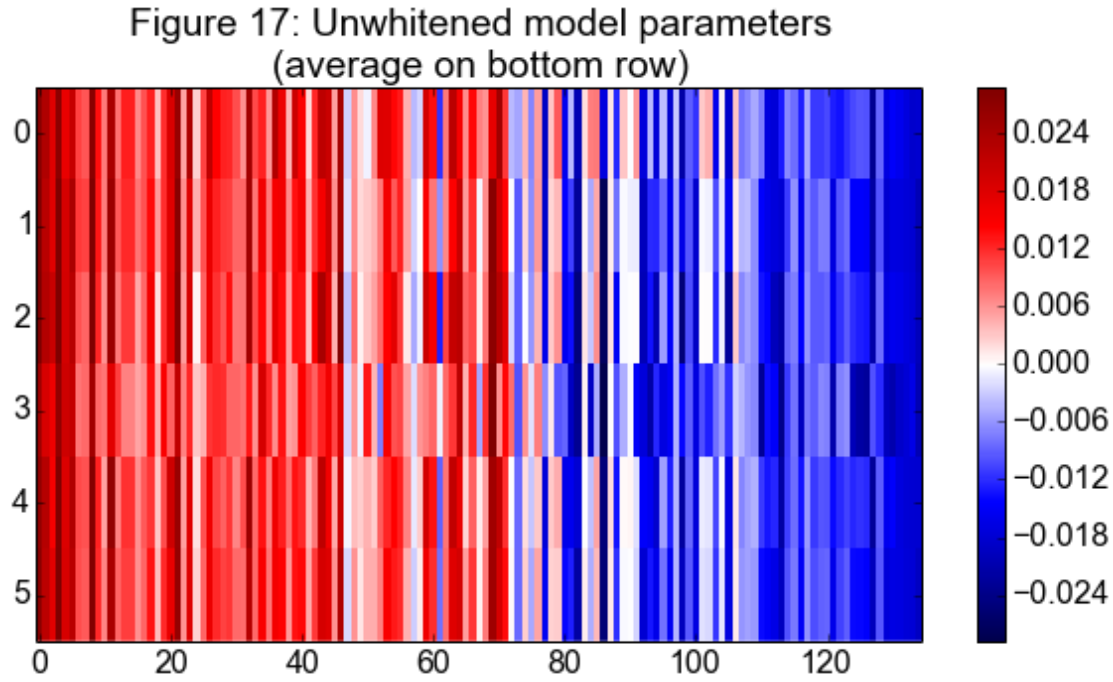
```
Fold 1 / 5
Variance explained: 0.70
Fold 2 / 5
Variance explained: 0.77
Fold 3 / 5
Variance explained: 0.76
Fold 4 / 5
Variance explained: 0.78
Fold 5 / 5
Variance explained: 0.65
--------------------------------------------------
Average explained variance: 0.73 (correlation: 0.86)
```

We see above that by using a very simple linear model and a reduced set of features (i.e. orthogonalized), we obtain accurate average predictions of held-out MOS data (on the order of correlation around 0.85).

Below we investigate the average model weights across the cross-validation folds. We also look at the model weight after inversing the PCA/whitening transform.

In [96]:
```
# VISUALIZE REGRESION MODEL WEIGHTS
new_figure();
plt.set_cmap(plt.cm.seismic)
plt.imshow(np.vstack([coeffs, coeffs.mean(0)]),interpolation='none');
plt.axis("tight");
maxx = max(abs(coeffs.mean(0)))
plt.clim([-maxx,maxx]); colorbar();
plt.title('Figure %d: Model Parameters\n(average weights on bottom row)' % fig_cnt); fig_cnt +=

# VISUALIZE WEIGHTS IN ORIGINAL FEATURE SPACE
unwhitened_coeffs = pca.inverse_transform(coeffs)
avg_coeffs = unwhitened_coeffs.mean(0)
maxx = max(abs(avg_coeffs))
new_figure();
plt.imshow(np.vstack([unwhitened_coeffs, avg_coeffs]),interpolation='none');
plt.axis("tight");
plt.clim([-maxx,maxx]); colorbar();
plt.title('Figure %d: Unwhitened model parameters\n(average on bottom row)' % fig_cnt); fig_cn
```





Looking at the fit model weights the 1st PC gets strong negative weighting.This is reasonable given the inverse relationship between the first PC and MOS discussed earlier. We can also see that a majority of the coefficients are zero, this is due to the effect of the LASSO penalty.

In order to analyze the model representation in terms of the ACS attributes, I've applied the inverse of the PC/whitening transform to the model weights to give the plot above. To see what the model is doing in terms of the actual ACS attributes, I visualize below the features ranked according to the average unwhitened weights:

```
In [97]: # LOOK AT THE AVERAGE UNWHITENED MODEL
         ac_sort_idx = sorted(range(len(avg_coeffs)), key=lambda k: avg_coeffs[k])

         # VISUALIZE
         plt.figure(figsize=(10,30))
         plt.barh(range(len(ac_sort_idx)),avg_coeffs[ac_sort_idx])
         plt.axis("tight");
         plt.yticks(arange(0,n_features,2),feature_names.values[ac_sort_idx[::2]])
         plt.title('Figure %d: Average Model Weights' % fig_cnt); fig_cnt += 1
```

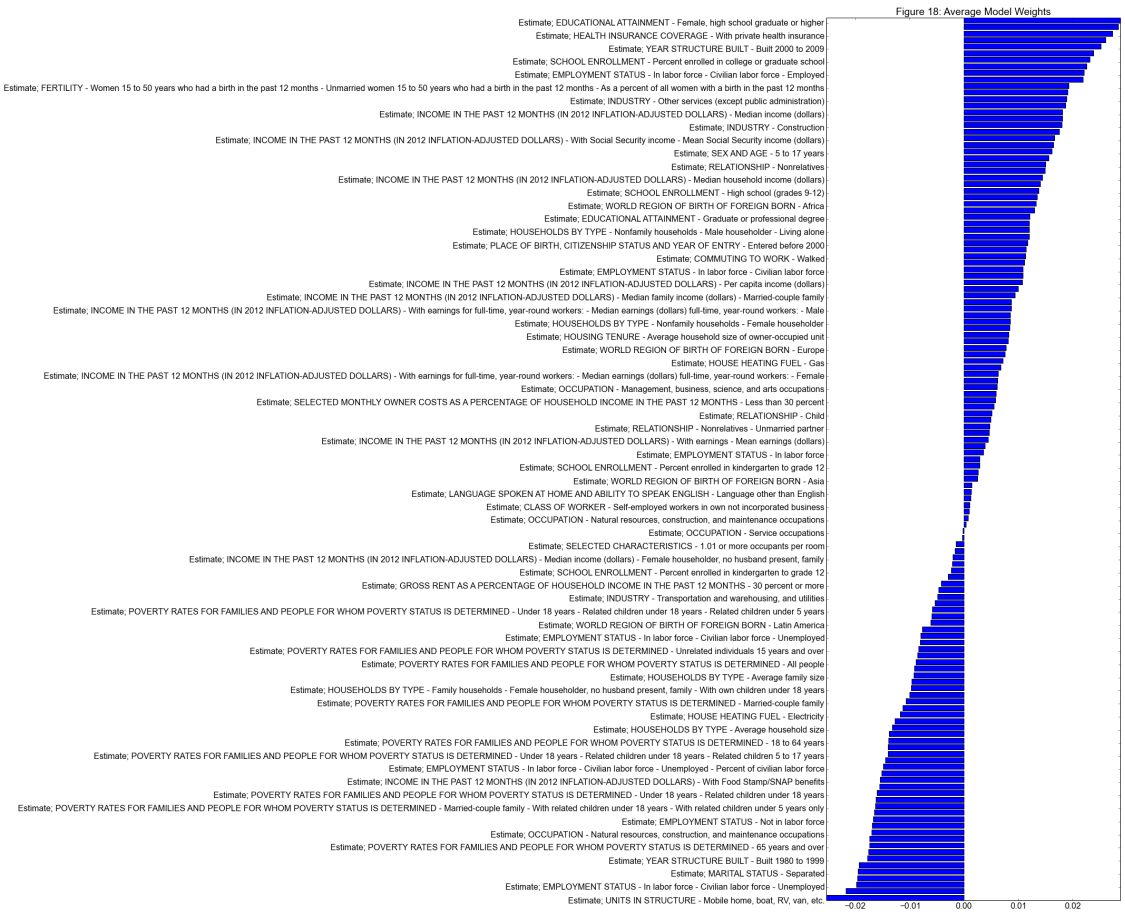


Figure 18 displays the average model weights after being de-whitened and sorted; this procedure tells us the magnitude and direction that each of selected features is related to MOS. We see that large model weights are associated with features that are indicate highly-educated, employed, and insured populations. Low model weights are associated with features that are indicative of low education, poverty rate, separated marriages, and responsibility for grandchildren, amongst others.

6.2 Gradient Boosted Trees Model

An alternateive approach to linear regression with a set number of features is to use an ensemble method which can simultaneosly select features during model fitting (and rank them). Below I train a Gradient Boosted Trees (GBT) model and display the resulting feature rankings.

In [98]:
```
from sklearn import ensemble
from sklearn.metrics import r2_score

# REMOVE MEAN FROM TARGET
targets = median_overall - median_overall.mean()

# USE ALL SELECTED FEATURES (MAKE SURE TO SHUFFLE)
features, targets = shuffle(selected_features, targets)

# CORSS-VALIDATE MODELS
k_folds = 5
kf = cross_validation.KFold(len(median_overall),n_folds=k_folds)

imp = []
r2 = []
cnt = 1
for train_idx, xval_idx in kf:
    print 'Fold %d / %d' % (cnt,k_folds)

    # CURRENT SAMPLES
    X_train, X_xval = features[train_idx], features[xval_idx]
    y_train, y_xval = targets[train_idx], targets[xval_idx]

    # INITIALIZE THE MODEL
    params = {'n_estimators': 50, 'max_depth': 3, 'min_samples_split': 1,
          'learning_rate': 0.1, 'loss': 'ls'}
    regr = ensemble.GradientBoostingRegressor(**params)

    # TRAIN
    regr.fit(X_train, y_train)

    # X-VALIDATE
    r2_tmp = r2_score(y_xval, regr.predict(X_xval))

    # EXPLAINED VARIANCE
    print('Variance explained: %.2f' % r2_tmp)
    r2.append(r2_tmp)

    # FEATURE IMPORTANCES
    imp.append(regr.feature_importances_)
    cnt += 1

importances = np.vstack(imp)
avg_r2 = np.array(r2).mean()
print '-'*50
print ('Average explained variance: %1.2f (correlation: %1.2f)' % (avg_r2, np.sqrt(avg_r2)));
maxx = abs(importances).max()
```

```
Fold 1 / 5
Variance explained: 0.89
Fold 2 / 5
Variance explained: 0.65
Fold 3 / 5
Variance explained: 0.65
Fold 4 / 5
Variance explained: 0.19
Fold 5 / 5
Variance explained: 0.49
-------------------------------------------------
Average explained variance: 0.57 (correlation: 0.76)
```

Here we see that the model predicts fairly well, albeight not as well as the linear regression model. This is likely due to the fact that the model hyperparameters (e.g. learning rate, tree depth, #of weak learners) have not been fully optimized and that there are not enough observations to adequately select the useful features. Both of these problems can be alleviated by introducing more training data and using grid search to determine the optimal hyperparameters.

Thought the GBT model doesn't work as well as the linear regression model, it can be used as a sanity check on the features that are important for predicting MOS. If the features identified by both modeling methods correspond, it provides further evidence for those features/attributes being important. Below we take a look at the feature importances in the GBT model (essentially the number of time a feature is included in the model).

```
In [99]: # VISUALIZE FEATURE IMPORTANCES
         avg_imps = importances.mean(0)[:,None]
         ai_sort_idx = sorted(range(len(avg_imps)), key=lambda k: avg_imps[k])[::-1]
         maxx = abs(avg_imps).max()

         # VISUALIZE
         n_show = 30
         plt.figure(figsize=(10,15))
         plt.barh(arange(n_show),avg_imps[ai_sort_idx[:n_show][::-1]])
         plt.axis("tight");
         plt.yticks(arange(n_show)+.5,feature_names.values[ai_sort_idx[:n_show][::-1]])
         plt.title('Figure %d: Average Feature Importances' % fig_cnt); fig_cnt += 1
```

We see that the GBT ranks a mix of features that vary both postively and negatively with MOS (given what we've gained the results from feature selection and the linear regression analyses). However, the overall findings support the linear regression findings. Namely MOS is predicted by education, employment status, and poverty rates.

One thing that I find interesting is that seemingly unrelated attributes are predictive of MOS. For example having private health insurance is often identified as an important feature for prediction MOS. This is likely because the ability to obtain private health insurance is correlated with a number of middle-to-upper class attributes (such as income and profession) that are associated with obtaining high MOS.

7 Predicting other Brain Areas

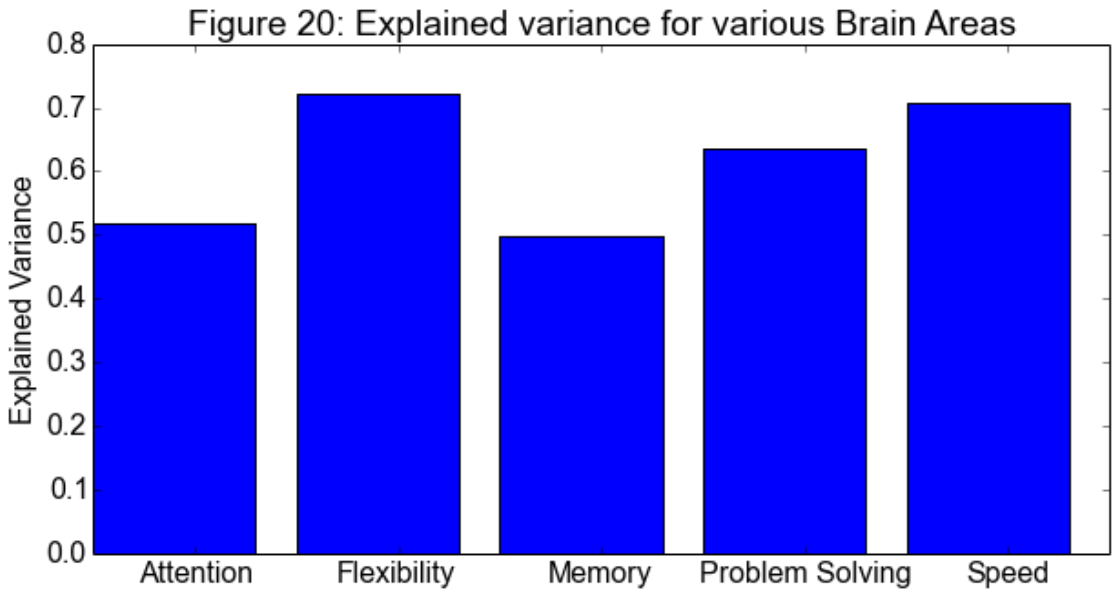
I also wondered whether there was any distinction in how the individual brain areas included in the MOS (e.g. Attention, Speed, Flexibility, etc.) were predicted using the PC-based linear regression analyses from above (I chose the linear regression only because it was the more accurate model out of those that I tried). Below we repeat the model fitting procedure, but with each individual brain areas scores as the target values.

In [100]:
```
# LOOP OVER INDIVIDUAL BRAIN AREAS
avg_accuracy = []
for area in all_brain_areas:

    # REMOVE MEAN FROM TARGET
    targets = data[area] - data[area].mean()

    # MAKE SURE TO SHUFFLE
    features, targets = shuffle(w_features, targets)

    # CROSS-VALIDATE MODELS
    k_folds = 5
    kf = cross_validation.KFold(len(median_overall),n_folds=k_folds)

    coeffs = []
    r2 = []
    cnt = 1
    print '\n'+'*'*50
    print 'Results for %s models' % area
    for train_idx, xval_idx in kf:
        print 'Fold %d / %d' % (cnt,k_folds)

        # CURRENT SAMPLES
        X_train, X_xval = features[train_idx], features[xval_idx]
        y_train, y_xval = targets[train_idx], targets[xval_idx]

        # INITIALIZE THE MODEL
        regr = linear_model.Lasso(alpha=0.1)
        #regr = linear_model.LinearRegression()

        # TRAIN
        regr.fit(X_train, y_train)

        # X-VALIDATE
        r2_tmp = regr.score(X_xval, y_xval)

        # EXPLAINED VARIANCE
        print('Variance explained: %.2f' % r2_tmp)
```

```
                r2.append(r2_tmp)

                # MODEL PARAMETERS
                coeffs.append(regr.coef_)
                cnt += 1

            coeffs = np.vstack(coeffs)
            avg_r2 = np.array(r2).mean()
            print '-'*50
            print ('Average explained variance: %1.2f (correlation: %1.2f)' % (avg_r2, np.sqrt(avg_r2)
            avg_accuracy.append(avg_r2)

        new_figure();
        xx = np.arange(len(all_brain_areas));
        plt.bar(xx, avg_accuracy);
        plt.xticks(xx+0.5,all_brain_areas);
        plt.ylabel('Explained Variance');
        plt.title('Figure %d: Explained variance for various Brain Areas' % fig_cnt); fig_cnt += 1
```

```
**************************************************
Results for Attention models
Fold 1 / 5
Variance explained: 0.50
Fold 2 / 5
Variance explained: 0.55
Fold 3 / 5
Variance explained: 0.62
Fold 4 / 5
Variance explained: 0.51
Fold 5 / 5
Variance explained: 0.42
--------------------------------------------------
Average explained variance: 0.52 (correlation: 0.72)

**************************************************
Results for Flexibility models
Fold 1 / 5
Variance explained: 0.84
Fold 2 / 5
Variance explained: 0.65
Fold 3 / 5
Variance explained: 0.73
Fold 4 / 5
Variance explained: 0.76
Fold 5 / 5
Variance explained: 0.62
--------------------------------------------------
Average explained variance: 0.72 (correlation: 0.85)

**************************************************
Results for Memory models
Fold 1 / 5
Variance explained: 0.58
Fold 2 / 5
```

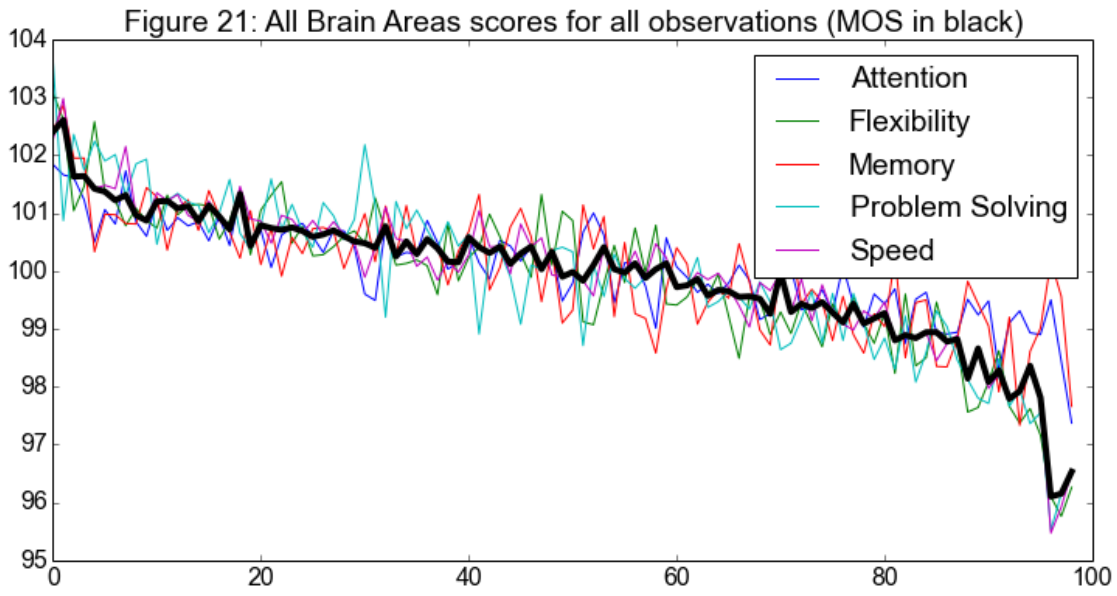
```
Variance explained: 0.49
Fold 3 / 5
Variance explained: 0.42
Fold 4 / 5
Variance explained: 0.57
Fold 5 / 5
Variance explained: 0.43
-------------------------------------------------
Average explained variance: 0.50 (correlation: 0.71)

*************************************************
Results for Problem Solving models
Fold 1 / 5
Variance explained: 0.62
Fold 2 / 5
Variance explained: 0.72
Fold 3 / 5
Variance explained: 0.81
Fold 4 / 5
Variance explained: 0.50
Fold 5 / 5
Variance explained: 0.53
-------------------------------------------------
Average explained variance: 0.63 (correlation: 0.80)

*************************************************
Results for Speed models
Fold 1 / 5
Variance explained: 0.46
Fold 2 / 5
Variance explained: 0.67
Fold 3 / 5
Variance explained: 0.82
Fold 4 / 5
Variance explained: 0.78
Fold 5 / 5
Variance explained: 0.81
-------------------------------------------------
Average explained variance: 0.71 (correlation: 0.84)
```



It appears that Speed and Flexibility are consistently better predicted than other brain areas such as Memory or Attention. This could be due to any number of factors. Perhaps Memory and Attention responses are inherently more variable and thus have less signal-to-noise. Or perhaps Attention and Memory are less indicative of the median overall score for some observations. If this is the case, then choosing features based on the correlation with the MOS would provide sup-opotimal features for predicting Attention or Memory. Or perhaps the socioeconomic factors that are predictive of Flexibility and Speed are more stable than those that are predictive of Memory and Attention.

In order to see if there are any discrpancies between the individual brain area scores and the median overall score, I plot each them below:

```
In [101]: plt.figure(figsize=(12,6))
          plt.plot(data[all_brain_areas].values);
          plt.plot(median_overall,'k',linewidth=4)
          legend(all_brain_areas);
          title('Figure %d: All Brain Areas scores for all observations (MOS in black)' % fig_cnt);
```



Taking a look at the traces of each predictor, it does appear that both Memory and Attention deviate from the MOS (in black) for lower-ranked cities. Given that this is around 10 percent of the observations, this could potentially lower model predictions. Obviously there is lots of room for future research! With that in mind, I've included a short list of possible ways to extend and improve the results of this case study.

8 Improvements on analyses

- Further explore the relationship between socioeconomic features and individual brain areas scores
- More in-depth work on hand-engineered features. Perhaps include interaction terms.
- Grid search over model parameters. For any reasonably-sized model, this will require more than 100 training observations.
- Use higher/lower p-value for selecting features. Perhaps using a p-value of 0.001 for determining the correlation threshold was too conservative, perhaps not conservative enough.
- Cluster and subspace analyses. Can we find underlying sub-populations of communities and/or individuals based on their input features?
- Other models: perhaps nonlinear models (e.g. neural networks) are more fitting (pun intended) for these data?
- More sophisticated model aggregation (other than mean of predictions)

9 Concerns regarding data collection

- **Geolocation**
- My main concern with the geolocation is that only the *last* IP address associated with a Lumosity login was used. This would perhaps be appropriate for users that only log in once, for multiple log-ins it would be more appropriate to use the mode of the distribution. Ties become weird here, however, as there is no way to interpolate. In that case, one could default to the original strategy of using the last IP address.
- **Scoring and normalization**
- My primary concern with the scoring is the choice of the $\mathcal{N}(100, \sigma^2 = 15)$ as the comparison distribution (I'm assuming there was a typo in the white page regarding the normal distribution's variance,

otherwise the variance would be 255). Where do these model parameters come from? Perhaps using a nonparametric / emprical distribution would be more appropriate.

10 Additional data

There are a number of axes that are not included in the ACS data that might also be useful for predicting MOS. For example variables like - Political affilation - Religious tendencies - The number of hours spent watching television or on the internet

are likely correlated with the way people absorb and synthesize information, and thus could be predictive of overall mental performance in a region.

In addition, indicators of overall health are also likely linked to mental performance in the region. Attributes that would be indicative of overall health include (but are obviously not limited to): - diabetes, heart disease, suicide, and infant mortality rates - food consumption trends (i.e. "do you buy organic or buy McDonalds") - the number of nearby parks and recreational areas.