National Health and Nutrition Examination Survey Data

CSC 478 Data Analysis Final Project

Kari Palmier

Table of Contents

I. Introduction

The dataset I chose is the National Health and Nutrition Examination Survey (NHANES) from California from 2013 – 2014 from Kaggle (https://www.kaggle.com/cdc/national-health-and-nutrition-examination-survey). The purpose of this survey is to assess the health and nutrition of children and adults in the state of California. Every year a new survey is taken and compared to prior years to determine how people's health has been trending. The NHANES is just one of the surveys that are performed countrywide in the United States. Each state takes their own survey and the CDC compiles the results together for a nationwide dataset. The datasets that I could find on the government website did not have information as to what the variables were, nor did it have any background information. The Kaggle dataset contained variable definitions as well as links to the codebooks that explained what each variable entry meant. The Kaggle dataset also consisted of four csv files that had been compiled from the numerous data files of the CDC (there were over 45 separate data files for just the demographic and questionnaire data).

After evaluating the variables in the Kagge NHANES dataset, I decided focus on factors that may affect whether a person is obese and their BMI. I chose several variables to use that I feel could contribute to a person's weight and that could possibly be predictive.

Variables Chosen:

- Gender
- Age
- Race
- Education Level
- Annual Household Income
- Marital Status
- Height
- Weight
- BMI (calculated from Height and Weight)
- Obesity Indicator (0 for BMI values under 30, 1 for over)
- Ratio of Income to Poverty Level
- High Blood Pressure Flag
- Diabetes Flag
- Amount Spent at Grocery Store Per Month
- Amount Spent on Non-Food Per Month
- Amount Spent Eating Out Per Month
- Amount Spent Delivery/Carryout Per Month
- Number of Meals Made at Home Per Week
- Number of Fast Food Meals Per Week
- Number of Frozen Meals Per Week
- Number of Ready Made Meals Per Week
- Number of Frozen Meals Per Week
- Doctor Said Overweight
- Doctor Said to Exercise
- Number of Sedentary Minutes Per Day
- Has Smoked 100 Cigarettes in Lifetime
- Has Had HEP B
- Has Had HEP C
- Number of Doctor Visits
- Has Stayed in Hospital Overnight
- Has Had Asthma
- Has Had Celiac
- Eats Gluten Free
- Has Had Gout
- Has Had Congestive Heart Failure
- Has Had Coronory Hear Failure
- Has Had Angina
- Has Had Heart Attack
- Has Had Stroke
- Has Had Emphysema
- Has Had Thyroid Disease
- Has Had Chronic Bronchitis
- Has Had A Liver Condition
- Has Had COPD
- Has Had Jaundice
- Has Had Cancer

II. Data Preparation

The Kaggle dataset consisted of four different csv files. I chose to use variables from the demographic and questionnaire csv files because they were of more interest and because these files contained the information for the same people. I compared the sequence numbers in the two csv files (each sequence number represents a person) and verified that all of the sequence numbers matched. I then sorted each csv by sequence number and manually copied the sorted data of each desired variable into a new csv file (basically manually merged the variables). This was all performed outside of Python, using Excel.

Once the merged csv file was imported into Python, I created exploratory plots of all of the variables and looked over the variable descriptions in the codebooks. I found that almost all variables had codes that corresponded to whether a person refused to answer a question or if they answered that they don't know. I first removed the rows that contained refused and don't know entries from all variables. I then plotted the variables again and found that there were a couple continuous variables that had a categorical entry that corresponded to if the person answered a value over 21. Because there is no knowledge as to what these people's answers were, the rows with these entries were also removed. There were a lot of NaN entries in the categorical and continuous variables due to people not answering certain questions. I removed all of the rows that were NaN for all categorical variables because there is no way to average a categorical variable. I then looked at the remaining NaN distribution. I found that one variable contained 22% NaN values. I felt this was too many NaN values to use a mean of the variable for replacement (the mean may not be representive of the true mean of all of the data because so many data point were missing). I removed all of the rows that had NaN for this variable. There were still several continuous variables that had NaN values present at this point. I used the mean of each of these variables to replace their NaN values. At this point, I reviewed the dataset codebook again and found that the income categorical variable contained two ranges (greater than $20K and less than $20K). These ranges overlapped several of the income brackets that were also given. Because it wouldn't make sense to have income brackets and a range covering the brackets, and because the number of rows of data with the range values was very low, I decided to remove the rows with the range values. At this point I created the same exploratory plots I did at the beginning and compared the variable distributions to ensure I had not changed any of them during data cleaning. The only change was to the age variable, where all entries less than 20 were removed. All other distributions still held after cleaning.

Once all of the data was properly cleaned, I then created a BMI continuous variable from the dataset height and weight variables (had to convert height to meters and weight to kg). The BMI formula is BMI = Weight (kg) / (Height (m))^2. This new BMI variable was used as the target variable for linear regression analysis. It wass also used to create a class variable that specifies if a person is obese or not. This obesity indicator is set to 1 for rows where the BMI is greater than or equal to 30 and 0 for rows with BMI values less than 30.

III. Exploratory Analysis

I chose to focus my exploratory analysis on the gender, age, race, education, marital status, income, height, weight, BMI, and obesity indicator variables because these were the most interesting to me. All exploratory plots and tables are in the Exploratory Analysis Materials Appendix at the back of this document.

I first created bar charts of all of the categorical variables of interest. The bar chart of gender shows that there was an almost equal number of women and men. The bar chart of education shows that the highest number of people had some college or associate degree, followed by college graduate and above, then high school graduate or GED. The bar chart of marital status showed that the vast majority of people were married, followed by never married. The bar chart of race shows that the large majority of people were non-Hispanic white, followed by non-Hispanic black, then Mexican-American. The bar chart of the income brackets shows that the majority of people made $100,000 or more, followed by people that made $75,000 to $99,9999. The remaining income brackets appeared to have somewhat normal distribution. The bar chart of the obesity indicator (1 for BMI >= 30, 0 for BMI < 30) shows that the vast majority of people were not obese.

Next, I created histograms for the continuous variables of interest. The histogram of age showed that the number of people in each age bracket decreased as age increased. All of the entries for people under 20 were removed during data cleaning because they did not answer most of the survey questions. The histogram of BMI shows was skewed to the right, with the majority of BMI values being less than 25. The histogram of BMI had the exact same shape as the histogram of weight. This is not surprising since BMI was calculated directly from weight. The histogram of height was fairly normally distributed.

In order to see the distribution of people who were obese versus those who were not, I created crosstabulation visualizations of the obesity indicator with the other categorical variables of interest. The crosstabulation of the obesity indicator and gender showed that there were more women who were obese (more than just the difference between the men and women populations). The crosstabulation of the obesity indicator with race showed that non-Hispanic white group had the highest number of people who were obese. This is not surprising because the population of non-Hispanic white was the highest overall. The crosstabulation of the obesity indicator with education showed that the education level with the highest number of people who were obese is some college/associate degree, which again is not surprising since this is the largest population overall. The number of people who were obese in the college graduate and above education level was the smallest of all of the levels, even though this had the next highest population. The crosstabulation of the obesity indicator with marital status showed that the distribution of obese marital status groups matched the non-obese ones (married had the highest number of obese but also was the largest population). The crosstabulation of the obesity indicator with income brackets resulted in the $35,000 to $44,999 income bracket having the highest number of people who were obese. The highest

population overall was in the $100,000 and above bracket, which had a low number of people who were obese.

The scatterplots of BMI versus height and BMI versus age showed that there was no obvious relation between the variables. The scatterplot of BMI versus weight showed a strong positive correlation, which again makes sense since BMI is calculated from weight.

Correlation analysis showed that there were fairly strong positive correlations (0.61507) between number of fast food meals and number of meals out (which makes sense because fast food restaurants are out of the house). There was also a fairly strong positive correlation (0.608568) between people being told by their doctor to lose weight and people being told by a doctor they are overweight. There was a fairly strong correlation (.608568) between people not being told by their doctor to lose weight and people not being told by their doctor they are overweight. There was a positive correlation (0.67545) between the income bracket $100,000 and above and the income to poverty guideline ratio. There was also a strong positive correlation (0.87451) between BMI and weight (as expected). None of the correlations found were surprising (all made sense).

IV. Classification

Tables containing all of the metrics referred to in this section are present in the Supplemental Analysis Material appendix.

I performed several different types of classification with the obesity indicator variable used as the class target variable. The types of classification performed were decision tree, k nearest neighbor (KNN), naïve bayes Gaussian, naïve bayes multinomial, and linear discriminant analysis (I used the sklearn library functions for all of these methods). The first step was to create a new full training dataset without the obesity indicator, and also without the BMI, height, and weight variables. The height and weight variables were removed because they were used in the calculation of the BMI variable, which was used to create the obesity indicator. It would be expected that the weight variable would be highly predictive since it is directly used in the calculation of the class variable. The second step was to split the dataset into 80% training and 20% testing randomly using the sklearn train_test_split function. Note that the 20% testing data was never used in any modelling. It was exclusively used for evaluating the performance of the final models (ones created using the full 80% training after cross-validation). I then created a min/max normalized version of the training and testing data to be used in the KNN classification. I performed the all methods of classification with all of the dataset features (variables) and performed all but the linear discriminant analysis with a reduced set of features. For the decision tree and KNN classification, I performed model selection with the sklearn grid search function with multiple parameter values in order to find the optimal model parameters (the 80% split training data was used with either the reduced feature set or all features). I did not do this with the naïve bayes and linear discriminant analysis models. The final step for all models was to perform cross-validation on the 80% training data (calculated the training and testing accuracies, classification matrix, and confusion matrix metrics), then to model with all of the 80% training data and to use this final model to classify the 20% testing data (again calculated the training and testing accuracies, the classification matrix, and confusion matrix metrics). Any tweaks to the model parameter ranges used were made based on the values of the cross-validation accuracy values, classification matrix, and confusion matrix. The number of total nodes also was used to determine if the model was overfit for the decision tree classification. To generate the reduced set of features, I used a two-stage feature selection process. I first generated cross-validation accuracy values for models that contained different percentages of the features (using the sklearn feature_selection.SelectPercentile and the cross_validation.cross_val_score functions). I also generated the weights and p values of the significance test for the variables from each percentage. I then found the percentage with the maximum accuracy and used its selected features, weights, and p values for the rest of the features selection process. Upon analysis of the chosen features, their weights, and their p values, I found that there were several features selected whose p values exceeded the 95% confidence level of 0.05 (a feature with a p value over 0.05 mean that the feature coefficient was not significantly different than 0, or that it is not significant to the model). The next stage in feature selection eliminated the features with p values over 0.05. This was done by removing

the feature with the highest p value, performing the cross_val_score with the data containing the rest of the features, then computing the p values of this new cross-validation. If any of the remaining features had p values over 0.05, this process was repeated (feature with new maximum p value is removed, then cross-validation is repeated). This process continued until all of the remaining features had p values under 0.05.

All classification results except naïve bayes multinomial had good accuracies (96%-97% all decision tree, 96%-97% all KNN, 85% feature reduced naïve gayes Gaussian, 75% all feature naïve bayes Gaussian, 96% linear discriminant analysis), but the classification and confusion matrices showed very poor performance classifying the obese instances (1 values). The precision and f1 score of the testing set of al of the models was very low (highest precision was 27% for linear discriminant analysis and highest f1 score was 23% for naïve bayes Gaussian). This poor obese value prediction was due to there being a dramatic class imbalance between the number of not obese values (0) and obese values (1). There were far more not obese than obese (96.7% of the obesity indicator was 0, 3.3% was 1). There was no way to easily handle this imbalance in the KNN, naïve bayes, or linear discriminant analysis classifications. There is the possibility of oversampling the obese cases, but since there are so few of them, this may not be a valid way to tackle this issue (not enough unique obese cases so this would result in all the obese cases being repeated a very high number of times). Undersampling the not obese cases is another possibility, but since there are so few obese cases, there would need to be dramatic undersampling (even if the obese cases were oversampled). The linear discriminant analysis classification reported that there was multicollinearity issue, so the results of this analysis may not be valid. It also did not perform well on obese instances either, so this would not be a good classifier regardless.

In order to try to improve the classification accuracy of the obese cases, I ran all of the classification code with the height and weight variables still in the dataset. I did this out of curiousity, to see exactly what effect these variables would have. There was no significant improvement to the classification of obese cases in KNN, naïve bayes Gaussian, nor naïve bayes multinomial (still had very poor classification performance for obese instances). There was a large increase in classification performance of obese cases in the decision tree models. The decision tree classifier generated from all features performed the best (92% precision, 82% recall, 87% f1 score of obese cases of 20% testing data), followed closely by the feature selection one (94% precision, 61% recall, 74% f1 score of obese cases of 20% testing data). The important feature reported by the tree classifier with all the features was only weight. The tree visual verifies this (the only feature used in the tree was weight). The important features reported by the tree classifier with the reduced features was weight and gender. The tree visual shows that most nodes were based on weight, with a couple based on gender. The linear discriminant precision was 83%, the recall was 86%, and the f1 score was 84% on the testing data. The linear discriminant model still reported the multicollinearity issue though. Although the performance did increase with adding the weight variable in particular, the results are not especially interesting since weight was used in the calculation of the obesity indicator variable.

In another attempt to improve the classification of obese cases with the original data (without the height and weight variables), I ran the decision tree models with the class_weights parameter set to balanced (weight of each class based on the number of instances, the lower the number of instances of a class, the higher the weight). The decision tree models with the classes weighted did improve the classification performance of the obese cases, but at the expense of the performance of the not obese cases. For the reduced feature model, the precision of the testing data was 17% (because several not obese cases were classified as obese), the recall was 89%, and the f1 score was 29% (due to the poor precision). The performance (accuracy, precision, recall, and f1 score was negligible) of the model with all features was very similar to the one with reduced features. The confusion matrices showed that while 89% of the reduced feature obese cases were classified correctly (78% of all feature obese cases), 16% of the not obese cases were incorrectly classified as obese (17% of all feature not obese cases).

I also ran the random forest and adaboost ensemble methods to see if they would improve the classification of the obese cases in the original data (without the height and weight variables). The overall performance (accuracy, precision, recall, and f1 score) on the training data was extremely good for both obese and not obese cases for all of these ensemble models. Unfortunately, all of them failed 100% at properly classifying the obese cases of the 20% testing data. though. This implies that all of the ensemble methods were overfit to the training data.

It is important to note that I did not see a noticeable difference between the performance (accuracy, precision, recall, and f1 score) between using the reduced feature dataset versus the dataset with all features on the decision tree (all versions), naïve bayes Gaussian, and naïve bayes multinomial classifiers. There was a slight improvement with the reduced feature data on the KNN classifier, but the classification results were still too poor to be useful.

V. Clustering

Tables containing all of the metrics referred to in this section are present in the supplemental analysis material appendix.

I first performed k-means clustering (from the sklearn library) on the full training data (the training data without the class variable before the train/test split) using several values of k to determine the optimal k value. I determined the optimal k value by finding the knee point of the plot of the sum of squared errors versus k vaue (the point where the slope changes from rapid decent to slower decent). The optimal value found was 8. I then used this optimal k value to perform clustering and calculated the sum of squared error value. The sum of squared error was calculated by taking the difference between the training instances and the cluster mean (centroid) the instances belong to, squaring these values, summing them per cluster, then summing the cluster sums. I then repeated this process using a min/max normalized version of the full training datset (optimal k was 12). I found that there was a significant improvement in the sum of squared error value when using the normalized data versus the original, which is to be expected (original data was 222817306.81 and normalized was 23907.85). Next, I created k = 2 clusters of the full training data and of the full normalized training data. I then calculated the completeness and homogeneity scores for both of the clustering results, using the obesity indicator class variable as the target. I found that again, the normalized data had much better completeness and homogeneity scores (3.02% versus 0.032% completeness and 13.2% versus 0.19% homogeneity). Note that the completeness and homogeneity scores for the normalized, while being higher than the original data, are still extremely low. This is due to that fact that 2 clusters are not adequate for the dataset (the optimal k value of the normalized data was 12). Out of curiosity, I thought I would see how using k = 2 clusters would work for classification of test cases. I performed k = 2 clustering on the 80% split training data and a normalized version of this data. I calculated the completeness and homogeneity of the clusters created and found they had similar values to those of the full training set (orig and normalized). I then used the cluster centers as the protovectors for Rocchio classification of the 20% testing data. I found that for the original (not normalized) training data clusters had 37.6% accuracy categorizing the 80% training data used for clustering and 41.15% accuracy categorizing the 20% testing data. The normalized training data clusters had 32.47% accuracy categorizing the 80% training data and 31.56% accuracy categorizing the 20% testing data. Neither the original or normalized split training data k = 2 clustering was adequate to use to classify the test cases. This again is not surprising since k = 2 had poor completeness and homogeneity. Next, I decided to see if performing principal component analysis would assist in improving the clustering of the normalized full dataset from before the train/test split (at this point I stopped looking at the non-normalized data since it was established that is had poor performance). Initially, I generated components for the same number of variables in the training set (101). I then found the number of components requird to capture over 90% of the variance of the initial data. I found that 38 components captured 90.39% of the variance. I then used these 38 new components to determine the optimal

clustering k value (as was done initially). The optimal k value returned was 11. I then performed clustering with these components and calculated the sum of squared error. The sum of squared error was 20993.11 using the PCA components, versus 23907.85 for the full normalized training used to generate the components. There was some improvement, but not enough to be significant.

The cluster centers from the optimal k = 12 clusteirng of the normalized full training data had some interesting characteristics. In order to analyze what variables contributed to which clusters, I focused on variables whose cluster mean values were over 0.5 (these would have the most affect on determining the cluster center). None of the clusters had positive responses for heart diseases, stroke, celiac, asthma, gout, emphysema, COPD, liver conditions, jaundice, chronic bronchitis, or cancer. There were cluster centers whose variables choices could be looked at as people in good health (no health issues such as weight issues, diseases, high blood pressure, diabetes, etc.), and there were clusters whose variables could be looked at as people in bad health (positive for obesity, high blood pressure, doctor saying they should lose weight and exercise, etc.). One interesting one cluster was for women that had all healthy variables (no weight issues, no diseases, no high blood pressure, no diabetes, etc), but had smoked over 100 cigarettes in their life (were considered smokers at one point even if they had quit). There were two clusters (one with men and one with women) that contained all healthy variables (no weight issues, no diseases, etc.) and also contained a race of non-Hispanic white, a high income to poverty ratio, college graduate or above, married, and an income of $100,000 or above. This implies that these other variables may be related to healthy people of both genders. There was a cluster of women with high age, high income to poverty ratio, race of non-Hispanic white, and married that had weight issues (doctor had told them they were overweight, needed to lose weight, and needed exercise), high blood pressure, and thyroid issues. It is possible that the weight issues could be related to the high age, high blood pressure, and/or thyroid issues since the rest of the variables were similar to healthy females. There was a similar cluster of men with high age, high income to poverty ratio, race of non-Hispanic white, and married that had weight issues (doctor had told them they were overweight, needed to lose weight, and needed exercise), and high blood pressure (no thyroid issues). This may mean the weight issues in men were related to age and/or high blood pressure. Note that the high blood pressure in both cases could also be caused by the weight issues.

VI. Regression

Tables containing all of the metrics referred to in this section are present in the Supplemental Analysis Material appendix.

I performed standard linear regression, ridge regression, and lasso regression (from the sklearn library) on the original training variables and normalized training variables. In order to perform linear regression, a different target variable had to be used than the obesity indicator from classification. Since linear regression requires a continuous target variable, I decided to use the BMI variable that was used to create the obesity indicator. The first step was to create a new full training dataset without BMI, and also without the obesity indicator, height, and weight variables. The height and weight variables were removed because they were used in the calculation of the BMI variable. It would be expected that the weight variable would be highly predictive since it is directly used in the calculation of the target variable. This result would be uninteresting because the weight and target variable relationship is already known. The second step was to split the dataset into 80% training and 20% testing randomly using the sklearn train_test_split function. Note that the 20% testing data was never used in any modelling. It was exclusively used for evaluating the performance of the final models (ones created using the full 80% training after cross-validation). I then created a min/max normalized version of the training and testing data to be used in each of the types of regression performed. I performed the all methods of regression with a reduced set of features. For ridge and lasso regression, I performed model selection with a grid search with different alpha parameter values in order to find the optimal model parameter (the 80% split training data was used with either the reduced feature set). The final step for all models was to perform cross-validation on the 80% training data (calculated the cross-validation training and testing root mean squared errors), then to model with all of the 80% training and use this final model to classify the 20% testing data (again calculated the training and testing root mean squared errors). Any tweaks to the model parameter ranges used were made based on the values of the cross-validation root mean squared errors values. To generate the reduced set of features, I used a similar two-stage feature selection process as was used in classification. The value being optimized was root mean squared error though instead of classification accuracy. I first generated cross-validation root mean squared error values for models that contained different percentages of the features (using the sklearn feature_selection.SelectPercentile and the cross_validation.cross_val_score functions). I also generated the weights and p values of the significance test for the variables from each percentage. I then found the percentage with the minimum root mean squared error and used its selected features, weights, and p values for the rest of the features selection process. Upon analysis of the chosen features, their weights, and their p values, I found that there were several features selected whose p values exceeded the 95% confidence level of 0.05 (a feature with a p value over 0.05 mean that the feature coefficient was not significantly different than 0, or that it Is not significant to the model). The next stage in feature selection eliminated the features with p values over 0.05. This was done by removing the feature with the highest p value, performing the cross_val_score with the data containing the rest of the features, then computing the p values of this new cross-validation. If any of the remaining

features had p values over 0.05, this process was repeated (feature with new maximum p value is removed, then cross-validation is repeated). This process continued until all of the remaining features had p values under 0.05.

The results of the standard linear regression, ridge regression, and lasso regression all resulted in similar root mean squared error values for the cross-validation model training and testing (all had both between 12 and 13.2) and the full model training and testing (all had training about 3.5 and testing about 13.5). The increase in the root mean squared error between the full training and testing means there may be some overfitting, but the testing error matched the cross-validation training and testing errors. The root mean squared error values were a bit high, but it is hard to know what a good value should be (root mean squared error is not out of a given number like accuracy, so there is no good or bad range – the closer to zero is better, but it is hard to say how far over zero is bad). The R squared valuea of all of the models (linear, ridge, and lasso) were between 0.4 and 0.41. This means that 60% of the variance in the BMI target variable is not explained by the independent variables. This is not desirable. There could be issues with the models trying to fit to the BMI variable because the distribution of the BMI variable is skewed toward low values (is not a normal distribution). Evaluation of the model coefficients generated showed that the linear regression model made with the normalized split training data had extremely large coefficients (so large they should not be used). The remaining model coefficients from either the original or normalized data were within reasonable ranges. The coefficients for the ridge and lasso regression models were very close when modelled with the original or the normalized training data. The lasso regression root mean squared errors versus the percentage of features during the feature selection stage did not follow the desired behavior. The expected behavior would be for the root mean squared error to start off high for a low percentage of variables and decrease as the percentage increases (standard linear regression and ridge regression followed this trend). The lasso model from the original training data had a step pattern to it where it was constant for several percentages at a time. The lasso model from the normalized training data was constant over all percentages. More information on why the lasso models exhibited this behavior is needed before I would be comfortable using lasso regression. The ridge regression alpha chosen during the model selection step for both the original and normalized training data was the maximum of the range given (was 4.996). This is a high alpha value but according to research I have done into acceptable ridge alpha values, the value of 4.996 should be acceptable.

Since the root mean squared error of the standard linear regression with the original training data and ridge regression with the original training data and normalized training data were all similar, the next step is to evaluate the coefficients chosen for these models. Note that lasso regression was no longer considered due to the undesirable behavior during features selection and standard linear regression with normalized training was no longer considered because of its very large coefficient values. To analyze the coefficients, I looked at coefficients that were greater than 0.3 and less than -0.3 (these weere the most predictive coefficients). All three models contained positive coefficients for non-Hispanic black, Mexican-American and no doctor visits in the past year. These coefficients contribute to a higher BMI. All three models contained negative coefficients for other race and a person was not told they are overweight by

a doctor and not told to lose weight by a doctor. These negative coefficients contribute to a lower BMI. The standard linear regression coefficients from the original training data model also contained positive coefficients for high school graduate/GED, some college/associate degree, income brackets between $15,000 and $19,999 and between $35000 and $44,999, if the person is borderline for diabetes, if a person has been told they were overweight by a doctor and and told to lose weight by their doctor, one doctor visit in the past year, and 16 and more doctor visits in the past year. It also contained a negative coefficient for the income bracket $100,000 and above. The ridge regression coefficients from the original training data model also contained negative coefficients for no high blood pressure and no asthma. The ridge regression coefficients from the normalized training data model also contained positive coefficients for the number of fastfood meals, the minutes spent sedimentary (ie sitting), and 16 and more doctor visits in the past year. It also contained negative coefficients for the income bracket $100,000 and over, no high blood pressure, a person not being told to exercise by a doctor, and no asthma. The coefficients of all three models make sense and give interesting insight into what predicts BMI. Since the models all have very similar performance, the choice between them would come down to what predictive variables are the most interesting. The ridge regression model fit from the normalized training data is the model I would choose because it has a good assortment of variables, the largest number of variables with coefficients greater than 0.5 or less than -0.5, and doesn't have extra variables. The standard linear regression had both if a doctor told a person they are over weight and to lose weight and if they did not for example.

Out of curiousity, I ran through all of the regression models with the height and weight parameters in the training dataset (as I did with classification). The models were all generated the same way (with the two-stage feature selection and model selection for the ridge and lasso regressions). There was significant improvement of the root mean squared errors of all models (cross-validation model training and testing values were all between 2.6 and 2.8 and the full model training and testing values all had training of approximately 1.6 and testing approximately 2.9). The R squared of all of the models also improved to 0.87. This means that 87% of the variance in the BMI variable was explained by the independent variables. This improvement again is not suprising given that BMI is directly calculated from weight. The lasso regression models still had the undesirable behavior in feature selection stage. The standard linear regression using the normalized training data still had extremely high coefficients. I next looked into the coefficients of the three remaining models (standard linear with original training and ridge regression with original and normal training). All models had positive coefficients for female (increase in BMI) and negative for male (decrease in BMI). This relationship was seen in the crosstabulation of the obesity indicator and gender variable. The ridge regression model from the normalized training data was the only model that contained the weight variable coefficient. Its coefficient was much higher than all of the others (was 36.6 where the rest were between -1 and 1, but this was normalized data so the weight variable being multiplied by this coefficient is a maximum of 1). The coefficient for weight was under 0.1 for the standard linear and ridge regression models from the original training data (this made sense because multiplying a weight amount over 100 by this coefficient would be around 10). The coefficients that were over 0.5 or under -0.5 for the standard linear and ridge

regression models from the original data were very similar. They both contained positive coefficients for female, Mexican-American, other race, and if a person has been told they are overweight by a doctor, and negative coefficients for male, some college/associate degree, college graduate and above, and not being told they are overweight by their doctor. The only difference is that the standard linear model also had a positive coefficient for no doctor visits. The ridge regression model from the normalized training data contained all of the variables from the standard linear regression model with the same coefficient signs plus a positive coefficient for weight and negative coefficients for the income to poverty ratio and the amount of money spent eating out. Although most of these variable coefficients make sense, the fact that the other race coefficient is positive when it has second lowest count of obese people in the obesity indicator and race crosstabulation is a little confusing to me. The ridge regression from the normalized training data having negative coefficients for the income to poverty ratio and the amount of money spent eating out also do not make sense to me. I am inclined to not use the ridge regression from normalized training data because of these variable coefficient inconsistencies. Either the standard linear regression or ridge regression model from the original training data would be acceptable since they have similar performance and coefficients. Again though, the fact that these models are all generated from a dataset containing the weight variable does make me question how interesting the results are though since the target variable BMI is calculated directly from weight.

VII. Conclusion

There were issues with classification with all of the models due to the distribution of the obesity indicator class variable containing significantly more not obese cases than obese cases (there was a large class imbalance). Most models had issues classifying the obese cases, even though they had high accuracy. The high accuracy was due to a high amount of the not obese cases being classified correctly, which is misleading since 96.7% of the class variable were not obese cases. When the decision tree models were altered to account for the class imbalance by adding balanced weighting the classes, the models classified the obese cases properly but had issues with misclassification of the not obese cases (17% were misclassified as obese). When the height and weight variables were added into the dataset, the decision tree model improved, but the only variable used in the tree was weight. This is not an interesting model since the class variable was generated from the weight. None of the classification models had good enough performance for future use because of the class imbalance. Possible ways to attempt to tackle the class imbalance issue would be to over sample the obese cases and under sample the not obese cases. Since there are so few obese cases, this would lead to a lot of oversampling of these cases. This could also cause issues. Another way to deal with the class imbalance is to change the class variable to be overweight (BMI greater than 25), underweight (BMI under 185), and normal weight (BMI 18.5 to 24.9). These classes then could be over or undersampled as needed.

The clustering results lead to many interesting relationships between variables. It was found that most healthy people were race of non-Hispanic white, had a high income to poverty ratio, were college graduates or above, were married, and had incomes of $100,000 or above for both men and women. It was also found that people with high ages, a high income to poverty ratio, races of non-Hispanic white, and were married had weight issues (doctor had told them they were overweight, needed to lose weight, and needed exercise) and high blood pressure for men and women (women also had thyroid issues). The best clustering was found when normalized training data was used. Primary component analysis did not improve the clustering significantly. Trying to cluster with a k value of two in order to calculate completeness, homogeneity, and to predict test cases did not work well because two clusters were not adequate to capture the variation in the training dataset.

All of the linear regression models for the BMI target variable resulted in similar root mean squared error values and R squared values even though their behavior during the modelling process and their coefficients were different. The lasso regression models during feature selection displayed unexpected and undesirable behavior and the standard linear regression model generated from normalized training data resulted in extremely large coefficients (too large). The R squared of all of the models was below 0.5, which meant that the variance of the BMI variable was not adequately explained by the independent variables. It is possible this is because the BMI variable distribution is heavily skewed toward low values. Adding height and weight did improve the root mean squared error and R squared of all of the models, but this is not suprising since BMI is calculated from weight.

Appendices

I. Supplemental Exploratory Materials

This section contains some of the visualizations used during the exploratoray analysis stage. Note that only visualizations of variables of interest are shown due to the large number of variables in the dataset.

.

1. Bar Charts

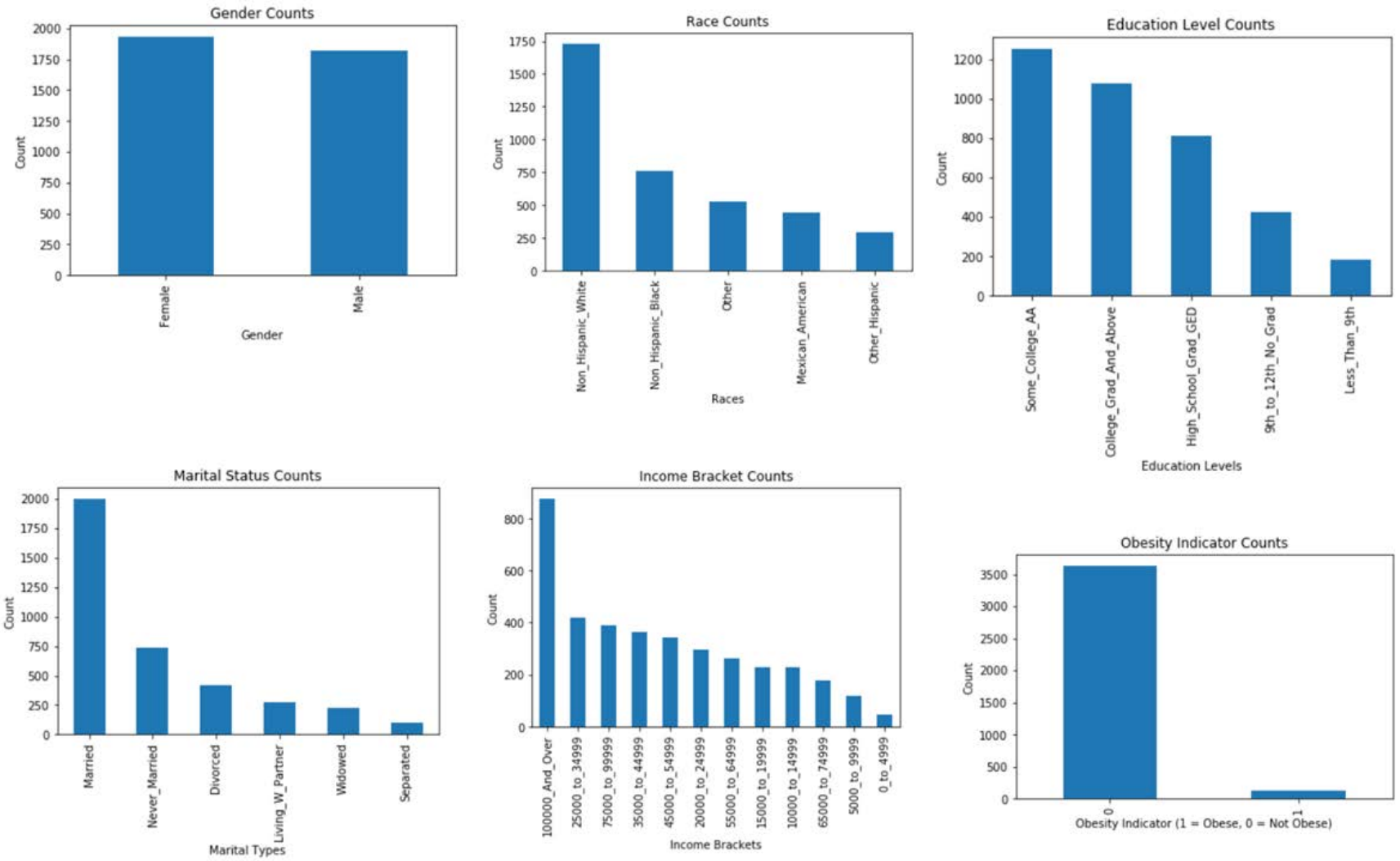


Figure 1: Exploratory Bar Charts

2. Histograms

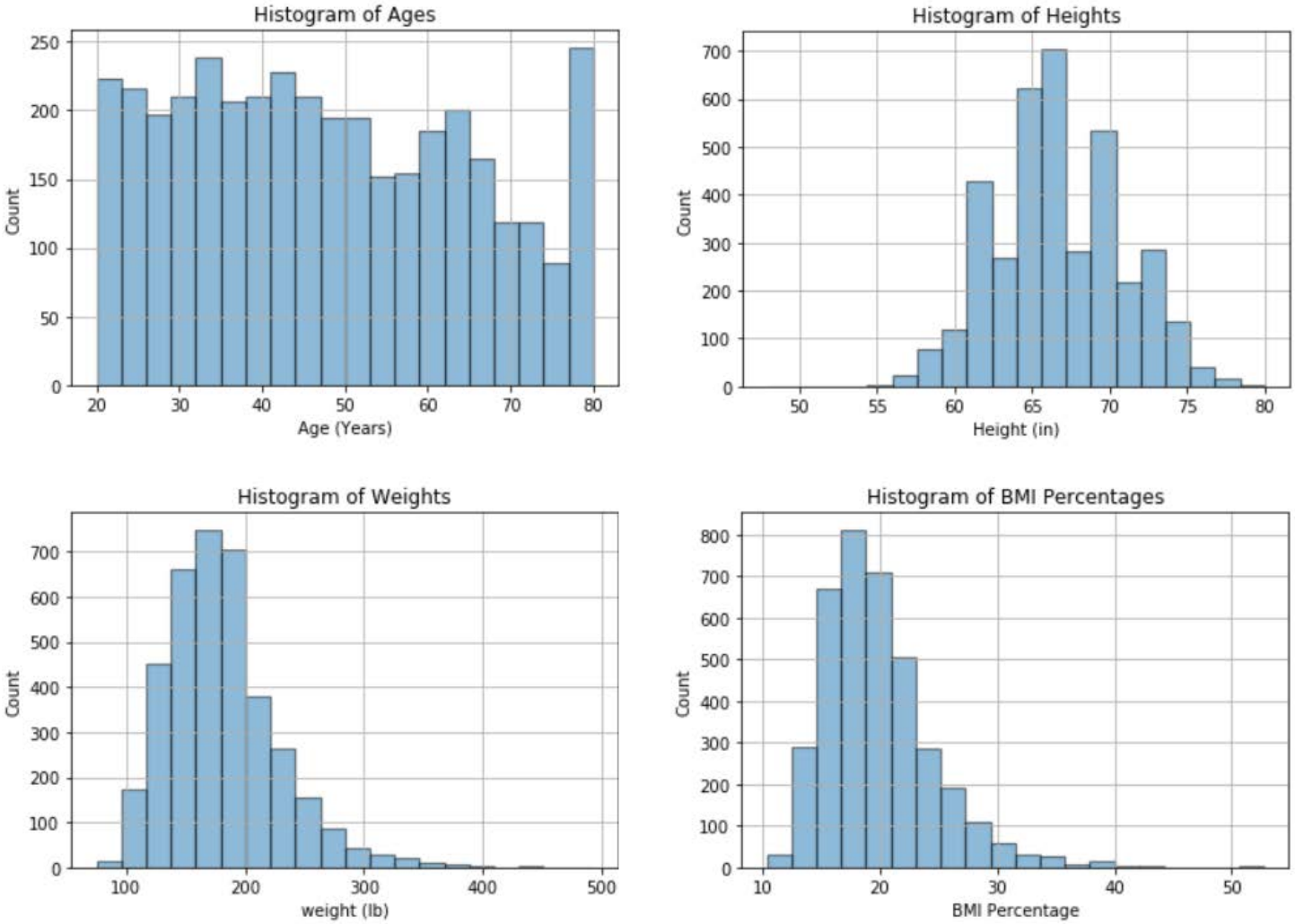


Figure 2: Exploratory Histograms

3. Crosstabulations

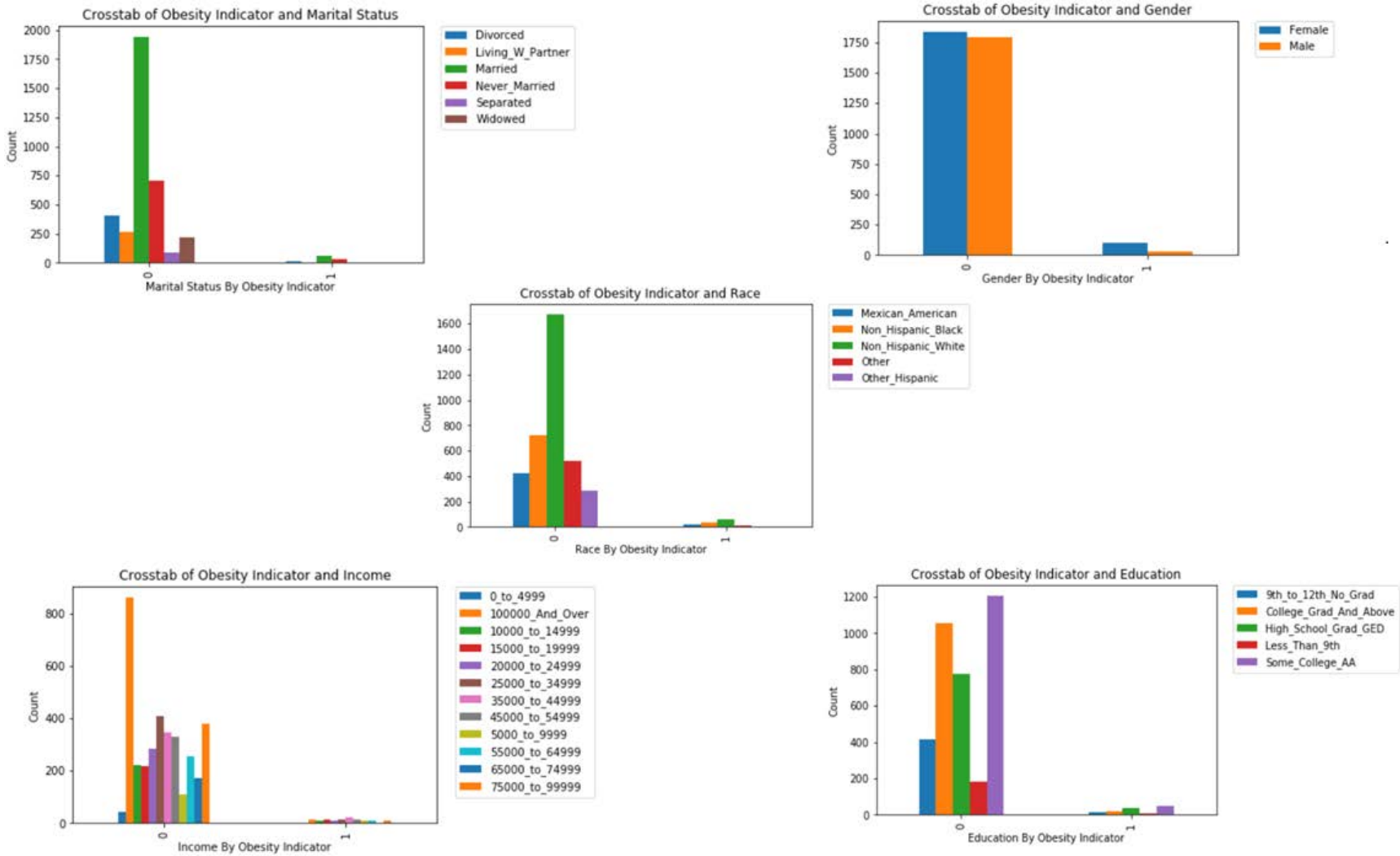


Figure 3: Exploratory Crosstabulation

4. Scatterplots

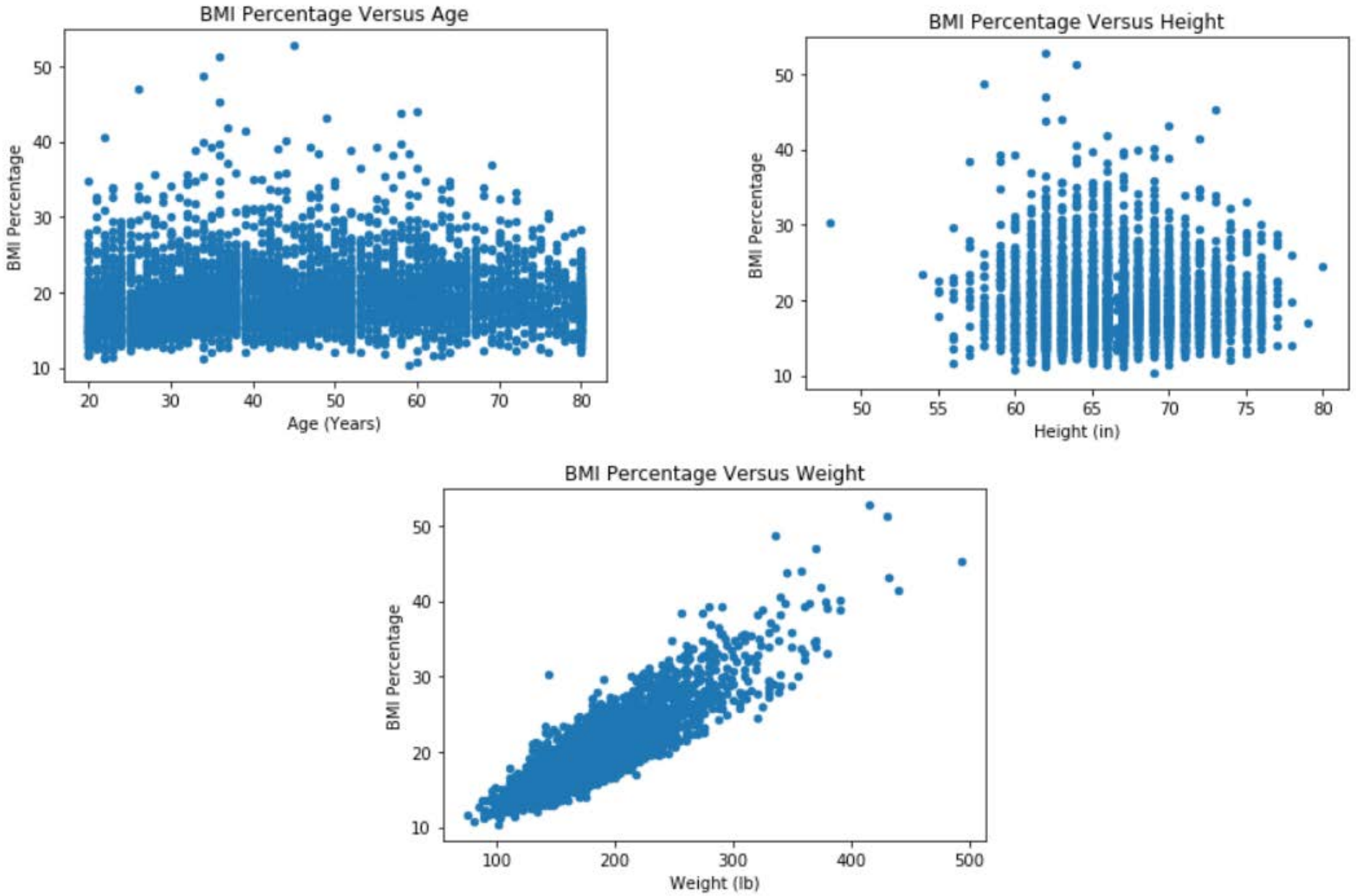
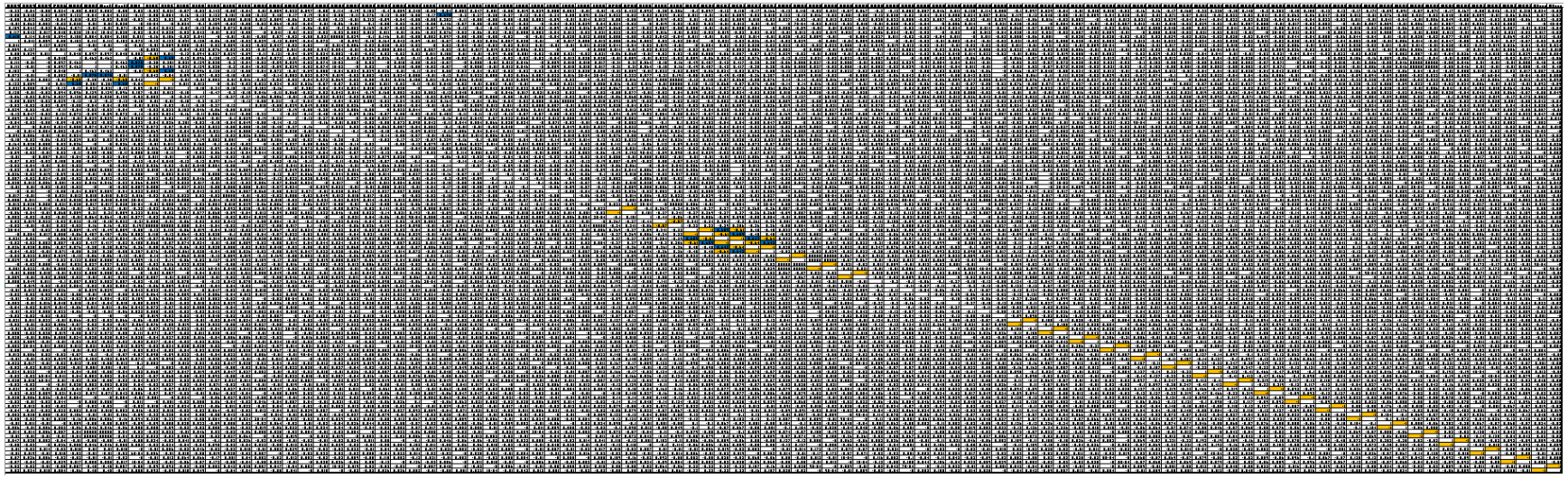


Figure 3: Exploratory Scatterplots

5. Correlation Analysis

Below is a screen capture of the Excel notebook containing the correlation of all of the variables in the dataset (there are 105 total). The yellow cells are values where the correlation is negative and is less than -0.6. The diagonal yellow cells correspond to -1 values for binary categorical variables (gender male correlated to gender female for example). The blue cells are values where the correlation is positive and greater than 0.6. All of the correlation analysis was performed in Excel on the correlation dataframe that was saved to a csv file in the Jupyter notebook code at the end of this document.



II. Supplemental Analysis Materials

This section contains all of the tables containing the metrics from classification, clustering, and regression. Models with “No H and W” in their name were generated using the original dataset without the weight and height variables. Models with “With H and W” in their names were generated with the dataset with weight and height included. Models with Feature Selection were generated with a reduced number of features (reduced set was generated through the two stage features selection process discussed previously). Decision tree models that have Balanced in their names were created with the class_weight parameter set to balanced

1. Classification

Decision Tree Classification:

Decision Tree Performance Summary

Model Name	Number of Features Chosen	CV Training Accuracy	CV Testing Accuracy	Full Training Accuracy	Testing Accuracy	Number of Nodes	Min Sample Split Chosen	Criterion Chosen	Max Depth Chosen	Min Samples Leaf Chosen	N Estimator
Decision Tree, Feature Selection, No H and W	37	0.967699	0.9670277	0.968032	0.9533955	23	40	entropy	4	10	N/A
Decision Tree, All Features, No H and W	101	0.967699	0.9670277	0.968032	0.9533955	23	40	entropy	4	10	N/A
Decision Tree, Feature Selection, With H and W	38	0.983387	0.9816833	0.98335	0.9840213	15	10	gini	3	10	N/A
Decision Tree, All Features, With H and W	103	0.991342	0.9866822	0.991009	0.9906791	15	10	gini	3	10	N/A
Decision Tree, Feature Selection, Balanced, No H and W	35	0.8419361	0.8121816	0.8315018	0.8388815	65	20	gini	8	10	N/A
Decision Tree, All Features, Balanced, No H and W	101	0.8460799	0.8201816	0.8401598	0.8322237	61	10	gini	8	10	N/A
Random Forest, All Features, Balanced, No H and W	101	0.9971511	0.9666966	0.998668	0.9627164	N/A	N/A	gini	N/A	N/A	20
Adaboost Decision Tree, All Features, Not Balanced, No H and W	101	1	0.9360587	1	0.9227696	N/A	N/A	N/A	N/A	N/A	95
Adaboost Decision Tree, All Features, Balanced, No H and W	101	1	0.9390576	1	0.9467377	N/A	N/A	N/A	N/A	N/A	40

	Full Training Set										
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Decision Tree, Feature Selection, No H and W	0.97	0.59	1	0.1	0.98	0.17	2904	99	2897	89	7	10
Decision Tree, All Features, No H and W	0.97	0.59	1	0.1	0.98	0.17	2904	99	2897	89	7	10
Decision Tree, Feature Selection, With H and W	0.99	0.88	1	0.58	0.99	0.7	2904	99	2896	42	8	57
Decision Tree, All Features, With H and W	1	0.85	0.99	0.88	1	0.87	2904	99	2889	12	15	87
Decision Tree, Feature Selection, Balanced, No H and W	1	0.16	0.83	1	0.9	0.28	2904	99	2398	0	506	99
Decision Tree, All Features, Balanced, No H and W	1	0.17	0.83	1	0.91	0.29	2904	99	2424	0	480	99
Random Forest, All Features, Balanced, No H and W	1	1	1	0.96	1	0.98	2904	99	2904	4	0	95
Adaboost Decision Tree, All Features, Not Balanced, No H and W	1	1	1	1	1	1	2904	99	2904	0	0	99
Adaboost Decision Tree, All Features, Balanced, No H and W	1	1	1	1	1	1	2904	99	2904	0	0	99

	Full Testing Set										
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Decision Tree, Feature Selection, No H and W	0.96	0	0.99	0	0.98	0	723	28	716	28	7	0
Decision Tree, All Features, No H and W	0.96	0	0.99	0	0.98	0	723	28	716	28	7	0
Decision Tree, Feature Selection, With H and W	0.98	0.94	1	0.61	0.99	0.74	723	28	722	11	1	17
Decision Tree, All Features, With H and W	0.99	0.92	1	0.82	1	0.87	723	28	721	5	2	23
Decision Tree, Feature Selection, Balanced, No H and W	1	0.17	0.84	0.89	0.91	0.29	723	28	605	3	118	25
Decision Tree, All Features, Balanced, No H and W	0.99	0.15	0.83	0.79	0.91	0.26	723	28	603	6	120	22
Random Forest, All Features, Balanced, No H and W	0.96	0	1	0	0.98	0	723	28	723	28	0	0
Adaboost Decision Tree, All Features, Not Balanced, No H and W	0.96	0.03	0.96	0.04	0.96	0.03	723	28	692	27	31	1
Adaboost Decision Tree, All Features, Balanced, No H and W	0.97	0.3	0.97	0.32	0.97	0.31	723	28	702	19	21	9

These are the non-zero feature importances for the different decision tree models generated:

Chosen Features And Importances

Model Name	RIDAGEYR	INDFMPIR	CBD070	CBD120	CBD130	DBD900	DBD910	PAD680	RIAGENDR_Female	RIAGENDR_Male	RIDRETH1_Other_Hispanic	MCQ080_No	MCQ365A_Yes	HUQ051_1
	Age Years	Poverty Ratio	Money Grocery	Money Food Out	Money Delivery	Fast Food	Frozen Food	Mins Sitting	Gender F	Gender M	Race Other Hisp	Doc OverW No	Doc LoseW Yes	Doc Visit 1
Decision Tree, Feature Selection, Balanced, No H and W	0.1050494	0.1010816	0.0300721	0.0133557	0.0144044	0.0282291	0.0184951	0.011018	0.0148375	0.0057679	0.0154257	0.543294	0.0611432	0.0378263

Model Name	RIDAGEYR	INDFMPIR	CBD070	DBD895	DBD900	DBD910	RIAGENDR_Female	RIAGENDR_Male	RIDRETH1_Other_Hispanic	DMDEDUC2_9th_to_12th_No_Grad	INDHHIN2_10000_to_14999	INDHHIN2_45000_to_54999	MCQ080_No	MCQ365A_Yes	HUQ051_1	MCQ160L_No
	Age Years	Poverty Ratio	Money Grocery	Food Out	Fast Food	Frozen Food	Gender F	Gender M	Race Other Hisp	Ed 9-12 No Grad	Income 10K-15K	Income 45K-55K	Doc OverW No	Doc LoseW Yes	Doc Visit 1	Liver Cond No
Decision Tree, All Features, Balanced, No H and W	0.0897905	0.0883287	0.0272267	0.0364221	0.0178113	0.0185404	0.0148739	0.0054109	0.0154635	0.0088506	0.0129995	0.0129114	0.5446257	0.0612931	0.0374212	0.0080305

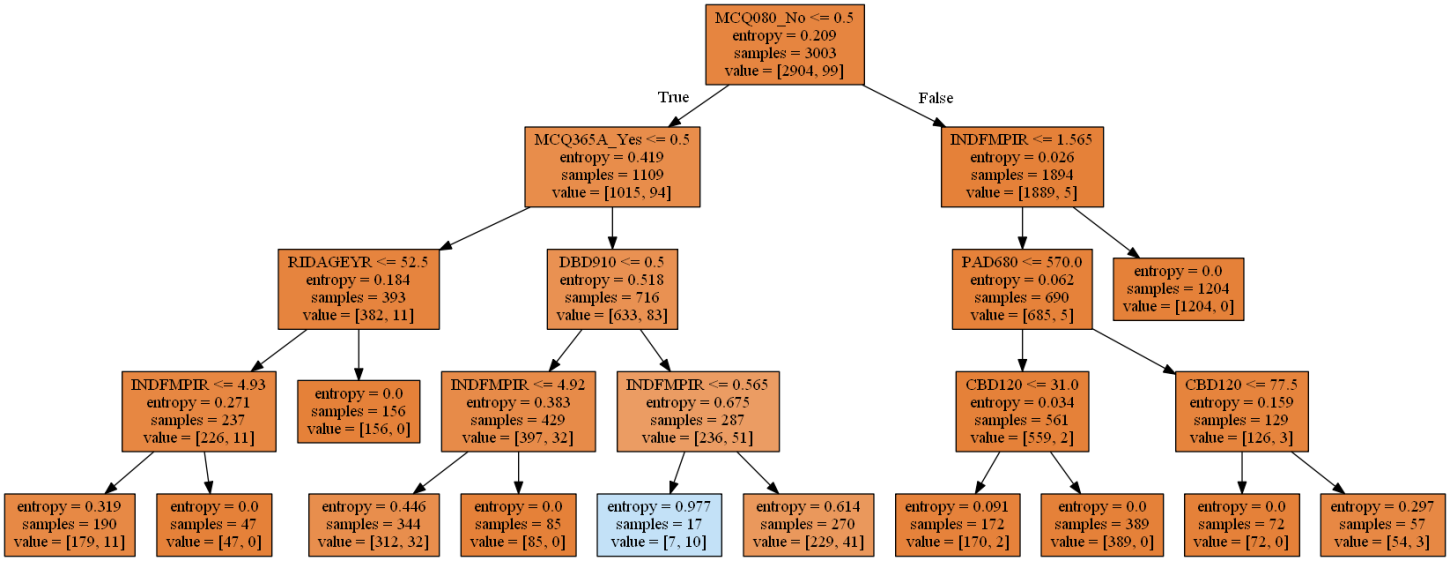
Model Name	RIDAGEYR	INDFMPIR	CBD120	DBD910	PAD680	MCQ080_No	MCQ365A_Yes
	Age Years	Poverty Ratio	Money Food Out	Frozen Food	Mins Sitting	Doc OverW No	Doc LoseW Yes
Decision Tree, Feature Selection, No H and W	0.0412085	0.1651143	0.0353978	0.0634122	0.0152707	0.571543	0.1080534

Model Name	RIDAGEYR	INDFMPIR	DBD910	PAD680	DMDMARTL_Never_Married	MCQ080_No	MCQ365A_Yes	MCQ160L_Yes
	Age Years	Poverty Ratio	Frozen Food	Mins Sitting	Marital Never	Doc OverW No	Doc LoseW Yes	Liver Cond Yes
Decision Tree, All Features, No H and W	0.0409912	0.1642435	0.0630778	0.0151902	0.0228449	0.5685289	0.1074836	0.0176398

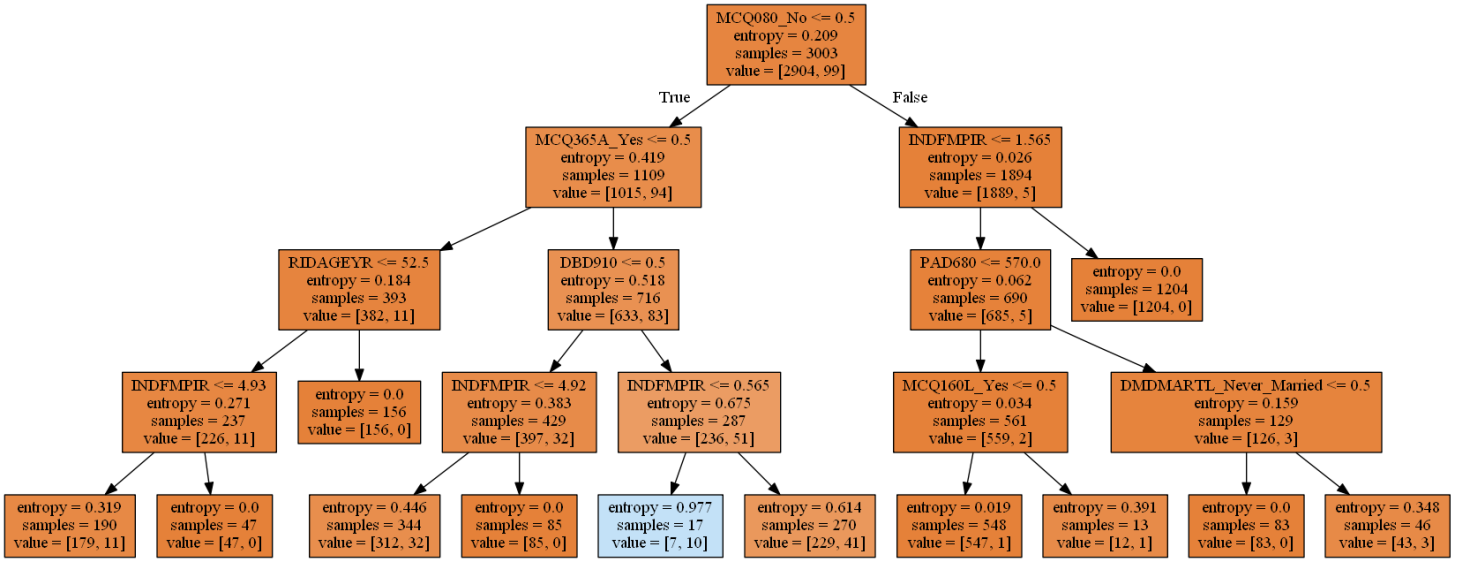
Model Name	WHD020	RIAGENDR_Female	RIAGENDR_Male
	Weight	Gender F	Gender M
Decision Tree, Feature Selection, With H and W	0.7713171	0.1660686	0.0626143

Model Name	WHD010	WHD020
	Height	Weight
Decision Tree, All Features, With H and W	0.399099	0.600901

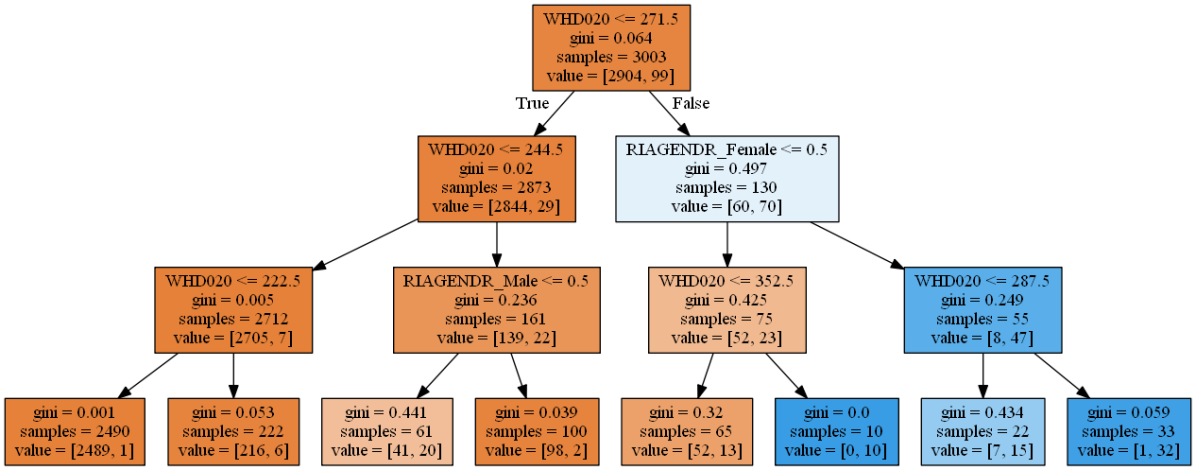
Below is the decision tree for the dataset without the height and weight with feature selection:



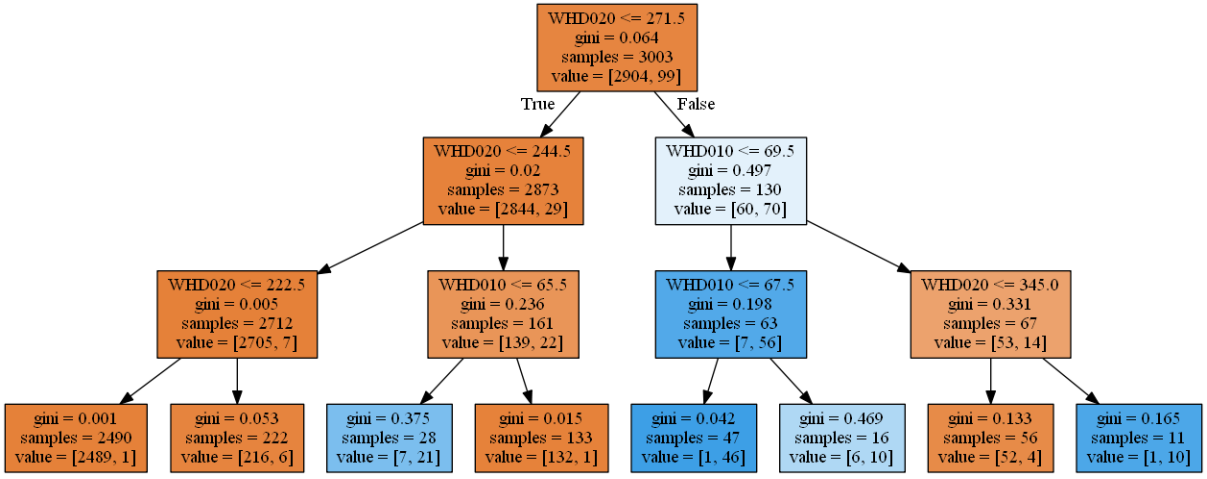
Below is the decision tree for the dataset without height and weight with all features:



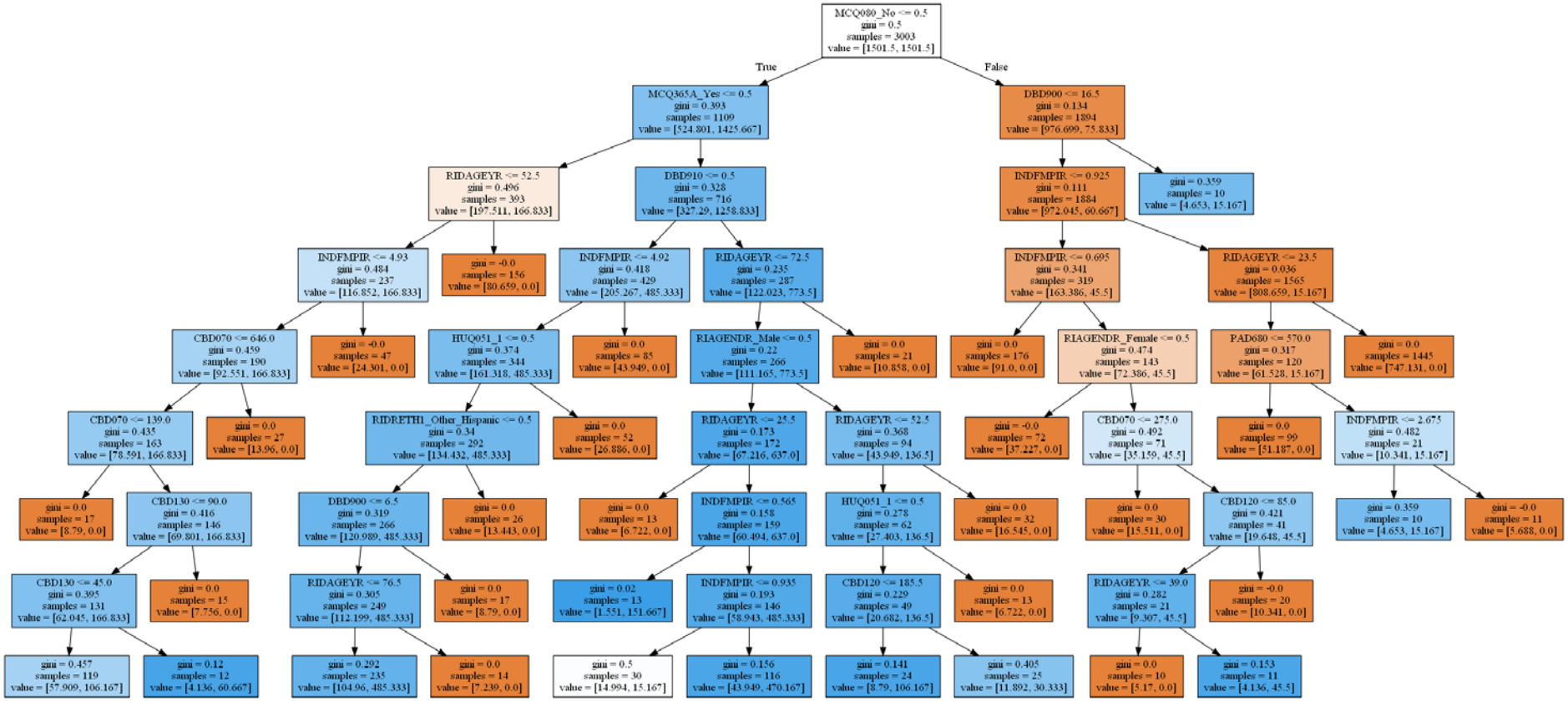
Below is the decision tree for the dataset with the height and weight with feature selection:



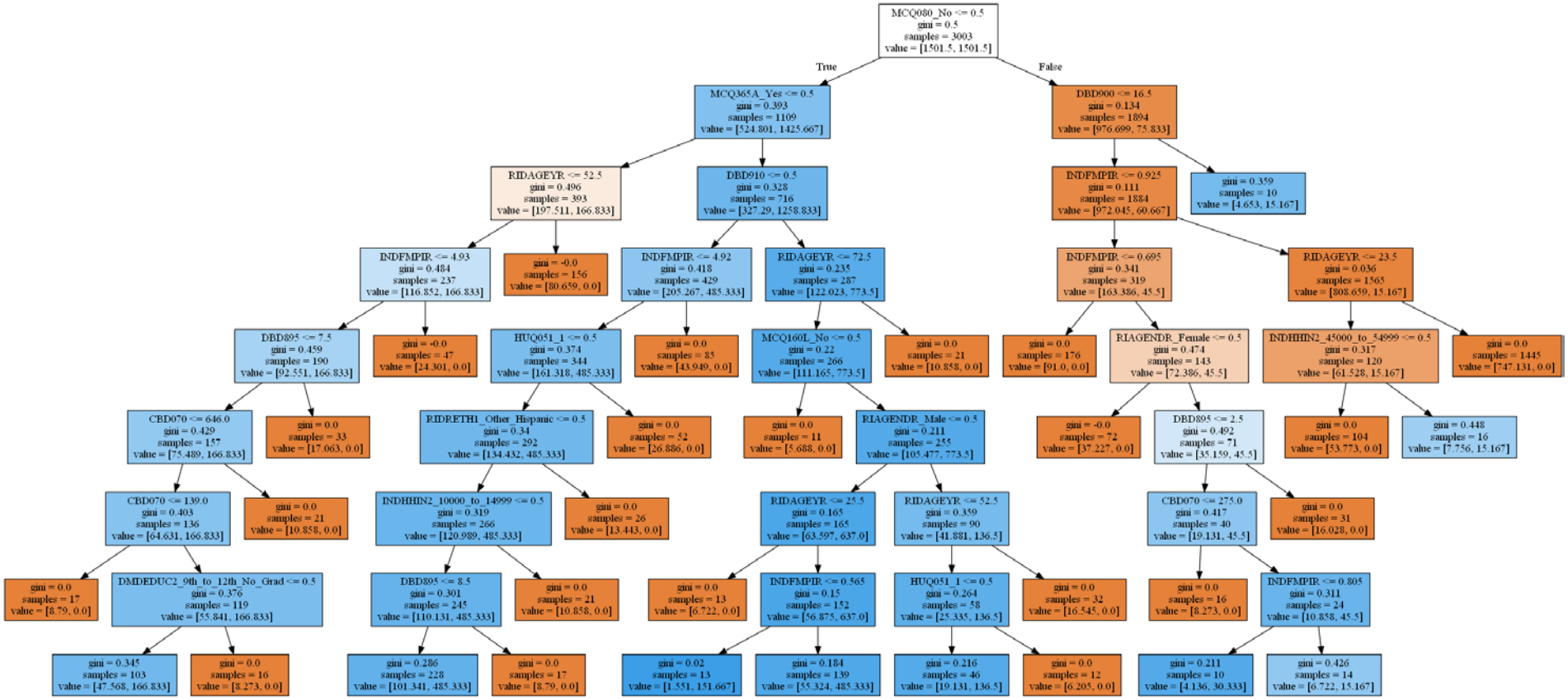
Below is the decision tree for the dataset without height and weight with all features:



Below is the decision tree for the dataset with balanced class weights and without the height and weight with feature selection:



Below is the decision tree for the dataset with balanced class weights and without the height and weight with all features:



KNN Classification:

KNN Performance Summary							
Model Name	Number of Features Chosen	CV Training Accuracy	CV Testing Accuracy	Full Training Accuracy	Testing Accuracy	N Neighbors	Weighs
KNN, Feature Selection, No H and W	24	0.96814296	0.96669878	0.96736597	0.96404794	5	uniform
KNN, All Features, No H and W	101	0.96703293	0.9670299	0.96703297	0.96271638	10	uniform
KNN, Feature Selection, With H and W	15	0.98009401	0.97569103	0.97968698	0.96804261	5	uniform
KNN, All Features, With H and W	103	0.96721792	0.9670299	0.96736597	0.96271638	10	uniform

	Full Training Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
KNN, Feature Selection, No H and W	0.97	0.57	1	0.04	0.98	0.08	2904	99	2901	95	3	4
KNN, All Features, No H and W	0.97	0	1	0	0.98	0	2904	99	2904	99	0	0
KNN, Feature Selection, With H and W	0.98	0.9	1	0.43	0.99	0.59	2904	99	2899	56	5	43
KNN, All Features, With H and W	0.97	1	1	0.01	0.98	0.02	2904	99	2904	98	0	1

	Full Testing Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
KNN, Feature Selection, No H and W	0.96	1	1	0.04	0.98	0.07	723	28	723	27	0	1
KNN, All Features, No H and W	0.96	0	1	0	0.98	0	723	28	723	28	0	0
KNN, Feature Selection, With H and W	0.97	0.75	1	0.21	0.98	0.33	723	28	721	22	2	6
KNN, All Features, With H and W	0.96	0	1	0	0.98	0	723	28	723	28	0	0

Naïve Bayes Gaussian:

Naïve Bayes Gaussian Performance Summary												
Model Name	Number of Features Chosen	CV Training Accuracy	CV Testing Accuracy	Full Training Accuracy	Testing Accuracy							
Naive Bayes Gaussian, Feature Selection, No H and W	37	0.84482173	0.84082392	0.84582085	0.83488682							
Naive Bayes Gaussian, All Features, No H and W	101	0.78255007	0.77390587	0.78687979	0.78428762							
Naive Bayes Gaussian, Feature Selection, With H and W	39	0.87142466	0.86646069	0.87179487	0.86551265							
Naive Bayes Gaussian, All Features, With H and W	103	0.80486088	0.79754153	0.80885781	0.8069241							
	Full Training Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Naive Bayes Gaussian, Feature Selection, No H and W	0.99	0.14	0.85	0.74	0.91	0.24	2904	99	2467	26	437	73
Naive Bayes Gaussian, All Features, No H and W	0.99	0.12	0.79	0.84	0.88	0.21	2904	99	2280	16	624	83
Naive Bayes Gaussian, Feature Selection, With H and W	0.99	0.19	0.87	0.86	0.93	0.31	2904	99	2533	14	371	85
Naive Bayes Gaussian, All Features, With H and W	0.99	0.13	0.81	0.88	0.89	0.23	2904	99	2342	12	562	87
	Full Testing Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Naive Bayes Gaussian, Feature Selection, No H and W	0.99	0.14	0.84	0.68	0.91	0.23	723	28	608	9	115	19
Naive Bayes Gaussian, All Features, No H and W	0.98	0.11	0.79	0.68	0.88	0.19	723	28	570	9	153	19
Naive Bayes Gaussian, Feature Selection, With H and W	0.99	0.18	0.87	0.75	0.93	0.29	723	28	629	7	94	21
Naive Bayes Gaussian, All Features, With H and W	0.99	0.13	0.81	0.75	0.89	0.22	723	28	585	7	138	21

Naïve Bayes Multinomial:

Naïve Bayes Multinomial Performance Summary												
Model Name	Number of Features Chosen	CV Training Accuracy	CV Testing Accuracy	Full Training Accuracy	Testing Accuracy							
Naive Bayes Gaussian, Feature Selection, No H and W	92	0.57542404	0.57108859	0.57708958	0.58721704							
Naive Bayes Gaussian, All Features, No H and W	101	0.57734807	0.57408527	0.57775558	0.59121172							
Naive Bayes Gaussian, Feature Selection, With H and W	103	0.62641041	0.63035327	0.62670663	0.65246338							
Naive Bayes Gaussian, All Features, With H and W	103	0.6359195	0.63434994	0.63636364	0.66045273							
	Full Training Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Naive Bayes Gaussian, Feature Selection, No H and W	0.98	0.05	0.58	0.63	0.72	0.09	2904	99	1671	37	1233	62
Naive Bayes Gaussian, All Features, No H and W	0.98	0.05	0.58	0.63	0.73	0.09	2904	99	1673	37	1231	62
Naive Bayes Gaussian, Feature Selection, With H and W	0.99	0.06	0.62	0.73	0.76	0.11	2904	99	1810	27	1094	72
Naive Bayes Gaussian, All Features, With H and W	0.99	0.06	0.63	0.73	0.77	0.12	2904	99	1839	27	1065	72
	Full Testing Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Naive Bayes Gaussian, Feature Selection, No H and W	0.97	0.05	0.59	0.54	0.73	0.09	723	28	426	13	297	15
Naive Bayes Gaussian, All Features, No H and W	0.97	0.05	0.59	0.54	0.74	0.09	723	28	429	13	294	15
Naive Bayes Gaussian, Feature Selection, With H and W	0.99	0.08	0.65	0.75	0.78	0.14	723	28	469	7	254	21
Naive Bayes Gaussian, All Features, With H and W	0.99	0.08	0.66	0.75	0.79	0.14	723	28	475	7	248	21

Linear Discriminant Analysis:

Linear Discriminant Analysis Performance Summary												
Model Name	Number of Features Chosen	CV Training Accuracy	CV Testing Accuracy	Full Training Accuracy	Testing Accuracy							
Linear Discriminant Analysis, All Features, No H and W	101	0.96562691	0.96003654	0.96503497	0.95605859							
Linear Discriminant Analysis, All Features, With H and W	103	0.991823	0.98834441	0.99134199	0.98801598							
	Full Training Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Linear Discriminant Analysis, All Features, No H and W	0.97	0.43	0.99	0.18	0.98	0.26	2904	99	2880	81	24	18
Linear Discriminant Analysis, All Features, With H and W	1	0.87	1	0.87	1	0.87	2904	99	2891	1	13	86
	Full Testing Set											
Model Name	Precision 0	Precision 1	Recall 0	Recall 1	F1 Score 0	F1 Score 1	Support 0	Support 1	Confusion Pred 0, Actual 0	Confusion Pred 0, Actual 1	Confusion Pred 1, Actual 0	Confusion Pred 1, Actual 1
Linear Discriminant Analysis, All Features, No H and W	0.97	0.27	0.99	0.11	0.98	0.15	723	28	715	25	8	3
Linear Discriminant Analysis, All Features, With H and W	0.99	0.83	0.99	0.86	0.99	0.84	723	28	718	4	5	24

2. Clustering

K Means Clustering on Full Dataset						
Model Name	**Number of Features Used**	**Number of Clusters Chosen**	**Sum Square Error**	**K = 2 Sum Square Error**	**K = 2 Completeness**	**K = 2 Homogeneity**
Full Original Dataset No H and W	101	8	222817306.8	489032287.8	0.000326378	0.001058712
Full Normalized Dataset No H and W	101	12	23907.84609	29217.20513	0.030273558	0.132366826
Full PCA from Normalized Dataset No H and W	38	11	20993.11154	.21081.7474791	0.011758271	0.185802855
Full Original Dataset With H and W	103	8	231265897.1	497551958.8	0.000326378	0.001058712
Full Normalized Dataset With H and W	103	10	24492.36709	29318.78962	0.030232918	0.132223265
Full PCA from Normalized Dataset With H and W	38	11	21040.98417	21005.13359	0.01165071	0.186242205

K Means Clustering on Training and Testing Split Datasets					
Model Name	**K = 2 Training Sum Square Error**	**K = 2 Training Completeness**	**K = 2 Training Homogeneity**	**K = 2 Training Prediction Accuracy**	**K = 2 Testing Prediction Accuracy**
Full Original Dataset No H and W	399531916.2	0.00025087	0.000824299	0.375957376	0.411451398
Full Normalized Dataset No H and W	23392.07808	0.030104816	0.134885949	0.324675325	0.315579228
Full PCA from Normalized Dataset No H and W					
Full Original Dataset With H and W	406365306.8	0.00025087	0.000824299	0.648018648	0.607190413
Full Normalized Dataset With H and W	23474.16333	0.030452393	0.136143946	0.676989677	0.687083888
Full PCA from Normalized Dataset With H and W					

Below are the variables deemed interesting of the cluster centroids of the K = 12 clustering using the full normalized training data (cluster variables whose means are greater than 0.5 and are different at least one other centroid). Note that the color of yellow indicates the mean values was between 0.5 and 0.7 (not including 0.7), and blue indicates the mean values between 0.7 and 1.

Full Normalized Dataset Interesting Cluster Center Variables With Mean Values Greater Than 0.5												
Cluster_0			Gender Female		Race White					No High BP		No Diabetes
Cluster_1			Gender Female			Some College/AA				No High BP		No Diabetes
Cluster_2		Income to Poverty		Gender Male	Race White		College Grad+	Married	Income 100K+	No High BP		No Diabetes
Cluster_3				Gender Male						No High BP		No Diabetes
Cluster_4	Age in Years	Income to Poverty	Gender Female		Race White			Married			High BP	No Diabetes
Cluster_5		Income to Poverty	Gender Female				College Grad+	Married	Income 100K+	No High BP		No Diabetes
Cluster_6	Age in Years			Gender Male	Race White						High BP	
Cluster_7	Age in Years	Income to Poverty		Gender Male	Race White			Married			High BP	No Diabetes
Cluster_8	Age in Years	Income to Poverty	Gender Female								High BP	No Diabetes
Cluster_9		Income to Poverty	Gender Female							No High BP		No Diabetes
Cluster_10				Gender Male						No High BP		No Diabetes
Cluster_11	Age in Years	Income to Poverty		Gender Male				Married			High BP	No Diabetes
Cluster_0	Not Overweight		No Weight Loss		No Exercise			Smoker	No Hospital Stay		No Thyrod Prob	
Cluster_1	Not Overweight		No Weight Loss		No Exercise		Non Smoker		No Hospital Stay		No Thyrod Prob	
Cluster_2	Not Overweight		No Weight Loss		No Exercise		Non Smoker		No Hospital Stay		No Thyrod Prob	
Cluster_3	Not Overweight		No Weight Loss		No Exercise		Non Smoker		No Hospital Stay		No Thyrod Prob	
Cluster_4		Overweight		Need to Lose Weight		Need Exercise	Non Smoker		No Hospital Stay			Has Thyroid Prob
Cluster_5	Not Overweight		No Weight Loss		No Exercise	Need Exercise	Non Smoker				No Thyrod Prob	
Cluster_6		Overweight		Need to Lose Weight		Need Exercise		Smoker		Hospital Stay	No Thyrod Prob	
Cluster_7	Not Overweight		No Weight Loss		No Exercise			Smoker	No Hospital Stay		No Thyrod Prob	
Cluster_8		Overweight		Need to Lose Weight		Need Exercise	Non Smoker		No Hospital Stay		No Thyrod Prob	
Cluster_9		Overweight		Need to Lose Weight		Need Exercise	Non Smoker		No Hospital Stay		No Thyrod Prob	
Cluster_10	Not Overweight		No Weight Loss		No Exercise		Non Smoker		No Hospital Stay		No Thyrod Prob	
Cluster_11		Overweight		Need to Lose Weight		Need Exercise	Non Smoker		No Hospital Stay		No Thyrod Prob	

3. Regression

Linear Regression Performance Summary										
Model Name	Number of Features Chosen	CV Training RMSE	CV Training R-Sq	CV Testing RMSE	CV Testing R-Sq	Full Training RMSE	Full Training R-Sq	Testing RMSE	Testing R-Sq	
Linear Regression, Orig Training, Feature Selection, No H and W	49	12.64597	0.418232	13.05129	0.398671	3.558932	0.41732	13.57838	0.403256	
Linear Regression, Normalized Training, Feature Selection, No H and W	49	12.79014	0.411653	13.21745	0.390408	3.558952	0.417313	13.57807	0.403269	
Linear Regression, Orig Training, Feature Selection, With H and W	50	2.668947	0.877208	2.740375	0.873095	1.634772	0.877057	2.906948	0.872245	
Linear Regression, Normalized Training, All Features, With H and W	50	2.679932	0.87669	2.747342	0.872831	1.634734	0.877062	2.907975	0.8722	

Ridge Linear Regression Performance Summary										
Model Name	Number of Features Chosen	CV Training RMSE	CV Training R-Sq	CV Testing RMSE	CV Testing R-Sq	Full Training RMSE	Full Training R-Sq	Testing RMSE	Testing R-Sq	Alpha
Linear Regression, Orig Training, Feature Selection, No H and W	49	12.64637	0.418214	13.04375	0.399018	3.558976	0.417305	13.57944	0.403209	4.996
Linear Regression, Normalized Training, Feature Selection, No H and W	49	12.64678	0.418195	13.03916	0.399232	3.559019	0.417291	13.5908	0.40271	4.996
Linear Regression, Orig Training, Feature Selection, With H and W	50	2.669065	0.877203	2.739114	0.873159	1.6348	0.877052	2.906226	0.872277	4.996
Linear Regression, Normalized Training, All Features, With H and W	50	2.668989	0.877206	2.740457	0.873096	1.634782	0.877055	2.907892	0.872204	0.041

Lasso Linear Regression Performance Summary										
Model Name	Number of Features Chosen	CV Training RMSE	CV Training R-Sq	CV Testing RMSE	CV Testing R-Sq	Full Training RMSE	Full Training R-Sq	Testing RMSE	Testing R-Sq	Alpha
Linear Regression, Orig Training, Feature Selection, No H and W	49	12.71386	0.415107	13.01143	0.400335	3.568251	0.414264	13.63601	0.400723	0.016
Linear Regression, Normalized Training, Feature Selection, No H and W	49	12.69498	0.415976	13.00665	0.40065	3.565614	0.41513	13.68474	0.398582	0.011
Linear Regression, Orig Training, Feature Selection, With H and W	50	2.679154	0.876739	2.728166	0.873691	1.637927	0.876582	2.914122	0.87193	0.006
Linear Regression, Normalized Training, All Features, With H and W	50	2.686734	0.87639	2.733325	0.873497	1.640171	0.876243	2.93957	0.870811	0.006

Below are the regression coefficients for the models generated with the training dataset without the weight and height variables. The color coding is: blue (0.5 and higher), green (0.3 to 0.5), yellow (-0.3 to -0.5), and pink (-0.5 and below).

Linear Regression Coefficients From Models Without Weight and Height

Standard Linear Regression

Original Training	Coeffs	Normalized Training	Coeffs
Mex-American	0.6974762	Num Fast Food	1.3373248
Non-Hisp Black	0.8436818	Mins Sedentary	1.2336762
Other Race	-0.992962	Female	-4.12E+12
HS Graduate/GED	0.3968981	Male	-4.12E+12
Some College/AA	0.3114107	Mex-American	0.6953856
Income $100K and Above	-0.363587	Non-Hisp Black	0.8415339
Income $15K to $20K	0.3989362	Other Race	-0.992572
Income $35K to $45K	0.3344213	HS Graduate/GED	0.394043
Borderline Diabetes	0.3350481	Some College/AA	0.3088379
Not Told Overweight	-2.013991	Income $100K and Above	-0.366211
Told Overweight	2.0139912	Income $15K to $20K	0.4003906
Not Told to Lose Weight	-0.996151	Income $35K to $45K	0.3322754
Told to Lose Weight	0.9961505	No High BP	-1.98E+11
No Doctor Visits	1.0652733	High BP	-1.98E+11
1 Doctor Visit	0.3994295	Borderline Diabetes	2.607E+10
16+ Doctor Visits	0.8088589	No Diabetes	2.607E+10
		Diabetes	2.607E+10
		Not Told Overweight	7.67E+11
		Told Overweight	7.67E+11
		Not Told to Lose Weight	-1.05E+12
		Told to Lose Weight	-1.05E+12
		Not Told to Exercise	2.38E+11
		Told to Exercise	2.38E+11
		No Doctor Visits	1.0673828
		1 Doctor Visit	0.3993225
		16+ Doctor Visits	0.8083496
		Overnight Hospital	-1.51E+12
		No Overnight Hospital	-1.51E+12
		No Asthma	4.84E+11
		Asthma	4.84E+11
		No Gout	5.24E+11
		Gout	5.24E+11
		No Congestive Heart Failure	6.292E+10
		Congestive Heart Failure	6.292E+10
		No Angina	-3.78E+11
		Angina	-3.78E+11
		No Heart Attack	-7.94E+10
		Heart Attack	-7.94E+10
		No Thyroid Prob	-9.66E+10
		Thyroid Prob	-9.66E+10
		No Bronchitis	4.44E+11
		Bronchitis	4.44E+11
		No COPD	2.41E+11
		COPD	2.41E+11

Ridge Regression

Original Training	Coeffs	Normalized Training	Coeffs
Mex-American	0.50662	Num Fast Food	0.8001039
Non-Hisp Black	0.7348229	Mins Sedentary	0.9184151
Other Race	-0.902639	Mex-American	0.5691567
No High BP	-0.485139	Non-Hisp Black	0.8239482
Not Told Overweight	-4.059203	Other Race	-0.928409
Not Told to Lose Weight	-1.964574	Income $100K and Above	-0.386563
No Doctor Visits	0.64008	No High BP	-0.487766
No Asthma	-0.341942	Not Told Overweight	-4.059633
		Not Told to Lose Weight	-1.968986
		Not Told to Exercise	-0.309673
		No Doctor Visits	0.7209666
		16+ Doctor Visits	0.3726166
		No Asthma	-0.388321

Lasso Regression

Original Training	Coeffs	Normalized Training	Coeffs
Mex-American	0.686081	Num Fast Food	1.2166375
Non-Hisp Black	0.8342994	Mins Sedentary	1.1783303
Other Race	-0.980959	Mex-American	0.6852074
HS Graduate/GED	0.3855422	Non-Hisp Black	0.8425431
Some College/AA	0.3008175	Other Race	-0.981232
Income $100K and Ab	-0.358887	HS Graduate/GED	0.3886595
Income $15K to $20K	0.3835517	Some College/AA	0.3042891
Income $35K to $45K	0.3294492	Income $100K and Abo	-0.364812
Borderline Diabetes	0.3202781	Income $15K to $20K	0.3850604
Not Told Overweight	-2.007123	Income $35K to $45K	0.3307925
Told Overweight	2.0071235	Borderline Diabetes	0.3197221
Not Told to Lose Weig	-0.994509	Not Told Overweight	-2.007943
Told to Lose Weight	0.9945086	Told Overweight	2.0079433
No Doctor Visits	1.0237132	Not Told to Lose Weigh	-0.99433
1 Doctor Visit	0.3687046	Told to Lose Weight	0.9943296
16+ Doctor Visits	0.7602786	No Doctor Visits	1.0232046
		1 Doctor Visit	0.367894
		16+ Doctor Visits	0.7637982

Below are the regression coefficients for the models generated with the training dataset with the weight and height variables. The color coding is: blue (0.5 and higher), green (0.3 to 0.5), yellow (-0.3 to -0.5), and pink (-0.5 and below).

Linear Regression Coefficients From Models With Weight and Height Included

Standard Linear Regression

Original Training		Normalized Training	
	Coeffs		Coeffs
Female	1.3211041	Income to Poverty	-0.371949
Male	-1.321104	Money Eatin Out	-0.483415
Mex-American	0.8034953	Weight	36.706552
Other Race	0.5038661	Female	-9.10E+12
College Grad+	-0.559354	Male	-9.10E+12
Some College/AA	-0.455092	Mex-American	0.8018853
Not Told Overweight	-0.406856	Other Race	0.506134
Told Overweight	0.4068557	College Grad+	-0.558594
No Doctor Visits	0.3024802	Some College/AA	-0.456146
		No High BP	-2.15E+12
		High BP	-2.15E+12
		Borderline Diabetes	9.55E+11
		No Diabetes	9.55E+11
		Diabetes	9.55E+11
		Not Told Overweight	2.49E+12
		Told Overweight	2.49E+12
		Not Told to Lose Wei	-2.31E+11
		Told to Lose Weight	-2.31E+11
		Not Told to Exercise	-5.83E+12
		Told to Exercise	-5.83E+12
		No Doctor Visits	0.3109131
		Overnight Hospital	-5.44E+12
		No Overnight Hospita	-5.44E+12
		No Asthma	-8.21E+11
		Asthma	-8.21E+11
		No Gout	5.93E+11
		Gout	5.93E+11
		No Congestive Heart	-1.03E+11
		Congestive Heart Fail	-1.03E+11
		No Angina	-1.33E+12
		Angina	-1.33E+12
		No Heart Attack	-1.18E+12
		Heart Attack	-1.18E+12
		No Thyroid Prob	-1.43E+12
		Thyroid Prob	-1.43E+12
		No Bronchitis	4.83E+11
		Bronchitis	4.83E+11
		No COPD	1.42E+11
		COPD	1.42E+11

Ridge Regression

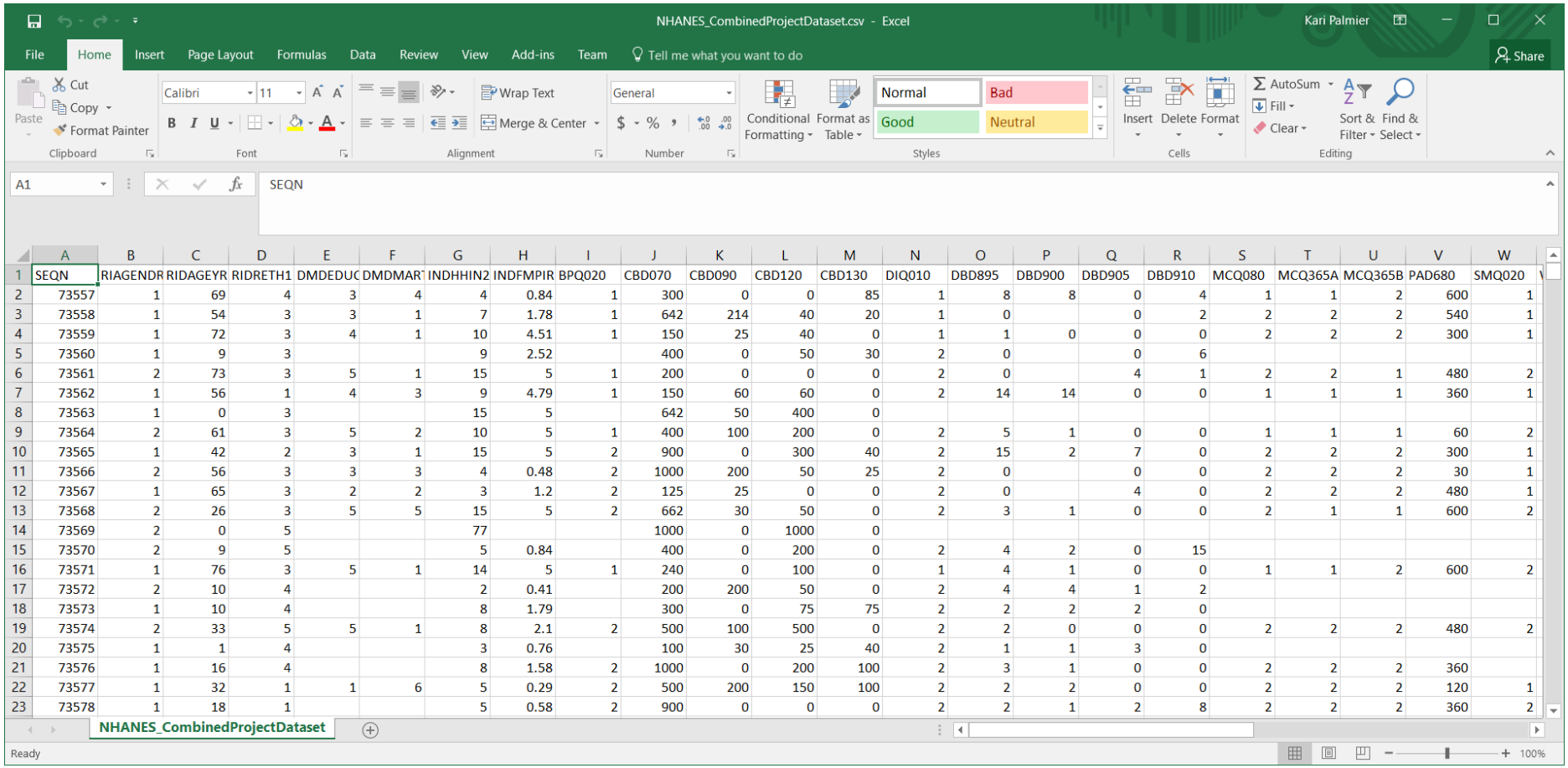
Original Training		Normalized Training	
	Coeffs		Coeffs
Female	1.3153841	Income to Poverty	-0.37606
Male	-1.315384	Money Eatin Out	-0.467547
Mex-American	0.7921091	Weight	36.650539
Other Race	0.4927967	Female	1.3185195
College Grad+	-0.541881	Male	-1.31852
Some College/AA	-0.43977	Mex-American	0.8032011
Not Told Overweight	-0.405797	Other Race	0.5010205
Told Overweight	0.4057967	College Grad+	-0.558293
		Some College/AA	-0.453562
		Not Told Overweight	-0.409802
		Told Overweight	0.4098018
		No Doctor Visits	0.3038302

Lasso Regression

Original Training		Normalized Training	
	Coeffs		Coeffs
Female	2.6314306	Income to Poverty	-0.414203
Mex-American	0.7807396	Weight	35.948884
Other Race	0.4347537	Female	2.5684566
College Grad+	-0.421147	Mex-American	0.7651123
Some College/AA	-0.326932	Other Race	0.3994522
Not Told Overweight	-0.806462	College Grad+	-0.442191
Told Overweight	-0.38028	Some College/AA	-0.323232
		Not Told Overweight	-0.876082
		Told Overweight	-0.37874

III. Dataset Example

Below is a screenshot of the csv file that I generated by combining attributes from the demographics csv and questionnaire csv files.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	SEQN	RIAGENDR	RIDAGEYR	RIDRETH1	DMDEDUC	DMDMART	INDHHIN2	INDFMPIR	BPQ020	CBD070	CBD090	CBD120	CBD130	DIQ010	DBD895	DBD900	DBD905	DBD910	MCQ080	MCQ365A	MCQ365B	PAD680	SMQ020
2	73557	1	69	4	3	4	4	0.84	1	300	0	0	85	1	8	8	0	4	1	1	2	600	1
3	73558	1	54	3	3	1	7	1.78	1	642	214	40	20	1	0		0	2	2	2	2	540	1
4	73559	1	72	3	4	1	10	4.51	1	150	25	40	0	1	1	0	0	0	2	2	2	300	1
5	73560	1	9	3			9	2.52		400	0	50	30	2	0		0	6					
6	73561	2	73	3	5	1	15	5	1	200	0	0	0	2	0		4	1	2	2	1	480	2
7	73562	1	56	1	4	3	9	4.79	1	150	60	60	0	2	14	14	0	0	1	1	1	360	1
8	73563	1	0	3			15	5		642	50	400	0										
9	73564	2	61	3	5	2	10	5	1	400	100	200	0	2	5	1	0	0	1	1	1	60	2
10	73565	1	42	2	3	1	15	5	2	900	0	300	40	2	15	2	7	0	2	2	2	300	1
11	73566	2	56	3	3	3	4	0.48	2	1000	200	50	25	2	0		0	0	2	2	2	30	1
12	73567	1	65	3	2	2	3	1.2	2	125	25	0	0	2	0		4	0	2	2	2	480	1
13	73568	2	26	3	5	5	15	5	2	662	30	50	0	2	3	1	0	0	2	1	1	600	2
14	73569	2	0	5			77			1000	0	1000	0										
15	73570	2	9	5			5	0.84		400	0	200	0	2	4	2	0	15					
16	73571	1	76	3	5	1	14	5	1	240	0	100	0	1	4	1	0	0	1	1	2	600	2
17	73572	2	10	4			2	0.41		200	200	50	0	2	4	4	1	2					
18	73573	1	10	4			8	1.79		300	0	75	75	2	2	2	2	0					
19	73574	2	33	5	5	1	8	2.1	2	500	100	500	0	2	2	0	0	0	2	2	2	480	2
20	73575	1	1	4			3	0.76		100	30	25	40	2	1	1	3	0					
21	73576	1	16	4			8	1.58	2	1000	0	200	100	2	3	1	0	0	2	2	2	360	
22	73577	1	32	1	1	6	5	0.29	2	500	200	150	100	2	2	2	0	0	2	2	2	120	1
23	73578	1	18	1			5	0.58	2	900	0	0	0	2	2	1	2	8	2	2	2	360	2

IV. Variable Descriptions

Variable Name	Label	Variable Values
SEQN	Respondent sequence number	
RIAGENDR	Gender	1 - Male 2 - Female
RIDAGEYR	Age in Years at Screening	0 to 79 - Value 80 - Anyone 80 and 0lder
RIDRETH1	Race/Hispanic Origin	1 - Mexican American 2 - Other Hispanic 3 - Non-Hispanic White 4 - Non-Hispanic Black 5 - Other Race - Including Multi-Racial
DMDEDUC2	Education Level - Adults 20+	1 - Less than 9th grade 2 - 9 to 11th grade (includes 12th with no diploma 3 - High school graduate/GED or equivalent 4 - Some college or AA degree 5 - College graduate or abovce 7 - Refused 9 - Don't Know
DMDMARTL	Marital Status	1 - Married 2 - Widowed 3 - Divorced 4 - Separated 5 - Never married 6 - Living with partner 77 - Refused 99 - Don't Know
INDHHIN2	Annual Household Income	1 - $0 to $4,999 2 - $5,000 to $9,999 3 - $10,000 to $14,999 4 - $15,000 to $19,999 5 - $20,000 to $24,999 6 - $25,000 to $34,999 7 - $35, 000 to $44,999 8 - $45,000 to $54,999 9 - $55,000 to $64,999 10 - $65,000 to $74,999 12 - $20,000 and Over 13 - Under $20,000 14 - $75,000 to $99,999 15 - $100,000 and Over 77 - Refused 99 - Don't Know

Variable Name	Label	Variable Values
INDFMPIR	Ratio of family income to poverty	0 to 4.99 - Values 5 - 5 and Above
BPQ020	Ever told you had high blood pressure	1 - Yes 2 - No 7 - Refused 9 - Don't Know
CBD070	Money spent at supermarket/ grocery store	0 to 4285 - Values 777777 - Refused 999999 - Don't Know
CBD090	Money spent on nonfood items	0 to 1542 - Values 777777 - Refused 999999 - Don't Know
CBD120	Money spent on eating out	0 to 2142 - Values 777777 - Refused 999999 - Don't Know
CBD130	Money spent on carryout/delivered foods	0 to 1028 - Values 777777 - Refused 999999 - Don't Know
DIQ010	Doctor told you have diabetes	1 - Yes 2 - No 3 - Borderline 7 - Refused 9 - Don't Know
DBD895	# of meals not home prepared	1 to 21 - Values 0 - None 5555 - More than 21 7777 - Refused 9999 - Don't Know
DBD900	# of meals from fast food or pizza place	1 to 21 - Values 0 - None 5555 - More than 21 7777 - Refused 9999 - Don't Know
DBD905	# of ready-to-eat foods in past 30 days	1 to 180 - Values 0 - None 5555 - More than 21 7777 - Refused 9999 - Don't Know
DBD910	# of frozen meals/pizza in past 30 days	1 to 180 - Values 0 - None 5555 - More than 21 7777 - Refused 9999 - Don't Know

Variable Name	Label	Variable Values
MCQ080	Doctor ever said you were overweight	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ365a	Doctor told you to lose weight	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ365b	Doctor told you to exercise	1 - Yes 2 - No 7 - Refused 9 - Don't Know
PAD680	Minutes sedentary activity	0 to 1200 - Values 7777 - Refused 9999 - Don't Know
SMQ020	Smoked at least 100 cigarettes in life	1 - Yes 2 - No 7 - Refused 9 - Don't Know
WHD010	Current self-reported height (inches)	48 to 81 - Values 7777 - Refused 9999 - Don't Know
WHD020	Current self-reported weight (pounds)	75 to 493 - Values 7777 - Refused 9999 - Don't Know
HEQ010	Ever told you have Hepatitis B?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
HEQ030	Ever told you have Hepatitis C?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
HUQ051	#times receive healthcare over past year	0 - None 1 - 1 2 - 2 to 3 3 - 4 to 5 4 - 6 to 7 5 - 8 to 9 6 - 10 to 12 7 - 13 to 15 8 - 16 or more 77 - Refused 99 - Don't Know

Variable Name	Label	Variable Values
HUQ071	Overnight hospital patient in last year	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ010	Ever been told you have asthma	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ082	Ever been told you have celiac disease?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ086	Are you on a gluten-free diet?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160n	Doctor ever told you that you had gout?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160b	Ever told had congestive heart failure	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160c	Ever told you had coronary heart disease	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160d	ver told you had angina/angina pectoris	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160e	Ever told you had heart attack	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160f	Ever told you had a stroke	1 - Yes 2 - No 7 - Refused 9 - Don't Know

Variable Name	Label	Variable Values
MCQ160g	Ever told you had emphysema	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160m	Ever told you had thyroid problem	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160k	Ever told you had chronic bronchitis	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160l	Ever told you had any liver condition	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ160o	Ever told you had COPD?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ203	Ever been told you have jaundice?	1 - Yes 2 - No 7 - Refused 9 - Don't Know
MCQ220	Ever told you had cancer or malignancy	1 - Yes 2 - No 7 - Refused 9 - Don't Know

V. Jupyter Notebook Python Code

I created one Jupyter notebook that I executed with the two different versions of the training data (the version without height and weight and the version with them). The base code was written for the dataset without height and weight. When I wanted to run the dataset with them in, I commented out the code to drop them (I have commented this section of the code accordingly). After the base classification, k-means clustering, and regression code was written and integrated, I saved off the results of each dataset into separate Jupyter notebooks. I then decided to add executing the decision tree with balanced class weighting and ensemble classification methods to the code with the dataset without the height and weight. This is the version of the code that I commented and is considered the final version of the code. This is the code that I have included in this section of this document. I have named the this final code file KariPalmier_CSC478_Final_Project_NHANESData_No_H_and_W.ipynb (and the corresponding HTML file is KariPalmier_CSC478_Final_Project_NHANESData_No_H_and_W.html). The version of the code that has the results for the dataset with height and weight is called KariPalmier_CSC478_Final_Project_NHANESData_With_H_and_W.ipynb (the HTML file is KariPalmier_CSC478_Final_Project_NHANESData_With_H_and_W.html). I have included all HTML and ipynb Jupyter notebook files in my final project submission.

Import libraries and python scripts

In [1]:
```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import pylab as pl
from sklearn import preprocessing
from sklearn import cross_validation
from sklearn.cross_validation import KFold
from sklearn.cross_validation import train_test_split
from sklearn import neighbors, tree, naive_bayes
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import feature_selection
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import completeness_score, homogeneity_score
from sklearn import decomposition
from sklearn.tree import export_graphviz
from sklearn.linear_model import LinearRegression, Lasso, Ridge
import graphviz
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline
```

```
C:\Users\Kari\Anaconda3\envs\Python27\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was
deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are m
oved. Also note that the interface of the new CV iterators are different from that of this module. This module will be remo
ved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Kari\Anaconda3\envs\Python27\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: This module was depr
ecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are move
d. This module will be removed in 0.20.
  DeprecationWarning)
```

---------- Import Dataset ----------

In [2]:
```
nhanes_data = pd.read_csv("C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\NHANES_CombinedProjectDataset.csv", del
imiter=",", index_col=0, header=0)
```

In [3]:
```
num_rows = nhanes_data.shape[0] + 1
num_cols = nhanes_data.shape[1] + 1
pd.set_option('max_rows', num_rows)
pd.set_option('max_columns', num_cols)
np.set_printoptions(threshold=np.inf)
```

In [4]:
```
nhanes_data.shape
```

Out[4]: (10175, 44)

In [5]:
```
print nhanes_data.head()
```

SEQN	RIAGENDR	RIDAGEYR	RIDRETH1	DMDEDUC2	DMDMARTL	INDHHIN2	INDFMPIR
73557	1	69	4	3.0	4.0	4.0	0.84
73558	1	54	3	3.0	1.0	7.0	1.78
73559	1	72	3	4.0	1.0	10.0	4.51
73560	1	9	3	NaN	NaN	9.0	2.52
73561	2	73	3	5.0	1.0	15.0	5.00

SEQN	BPQ020	CBD070	CBD090	CBD120	CBD130	DIQ010	DBD895	DBD900	DBD905
73557	1.0	300.0	0.0	0.0	85.0	1.0	8.0	8.0	0.0
73558	1.0	642.0	214.0	40.0	20.0	1.0	0.0	NaN	0.0
73559	1.0	150.0	25.0	40.0	0.0	1.0	1.0	0.0	0.0
73560	NaN	400.0	0.0	50.0	30.0	2.0	0.0	NaN	0.0
73561	1.0	200.0	0.0	0.0	0.0	2.0	0.0	NaN	4.0

SEQN	DBD910	MCQ080	MCQ365A	MCQ365B	PAD680	SMQ020	WHD010	WHD020
73557	4.0	1.0	1.0	2.0	600.0	1.0	69.0	180.0
73558	2.0	2.0	2.0	2.0	540.0	1.0	71.0	200.0
73559	0.0	2.0	2.0	2.0	300.0	1.0	70.0	195.0
73560	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
73561	1.0	2.0	2.0	1.0	480.0	2.0	67.0	120.0

SEQN	HEQ010	HEQ030	HUQ051	HUQ071	MCQ010	MCQ082	MCQ086	MCQ160N
73557	2.0	2.0	5	2	2.0	2.0	2.0	2.0
73558	2.0	2.0	5	2	1.0	2.0	2.0	2.0
73559	2.0	2.0	2	2	2.0	2.0	2.0	2.0
73560	2.0	2.0	1	2	2.0	2.0	2.0	NaN
73561	2.0	2.0	4	2	2.0	2.0	2.0	2.0

SEQN	MCQ160B	MCQ160C	MCQ160D	MCQ160E	MCQ160F	MCQ160G	MCQ160M	MCQ160K
73557	2.0	2.0	2.0	2.0	1.0	2.0	2.0	2.0
73558	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
73559	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
73560	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
73561	2.0	2.0	2.0	2.0	2.0	2.0	1.0	1.0

SEQN	MCQ160L	MCQ160O	MCQ203	MCQ220
73557	2.0	2.0	2.0	2.0
73558	2.0	2.0	2.0	2.0
73559	2.0	2.0	2.0	1.0
73560	NaN	NaN	2.0	NaN
73561	2.0	2.0	2.0	2.0

---------- Remove all Refused and Don't Know Answers ----------

Remove all entries where the person refused to answer or answered don't know

Create array of refused values (each entry corresponds to each variable of the nhanes_data dataframe).

In [6]:
```
features = np.array(nhanes_data.columns.values.tolist())

refused_ndx = [0, 0, 0, 7, 77, 77, 0, 7, 777777, 777777, 777777, 777777, 7, 7777, 7777, 7777, 7777, 7, 7, 7, 7777, 7, 7777,
 7777, 7, 7, 77, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]
refused_arr = np.zeros((1, len(refused_ndx)))
refused_arr[0] = refused_ndx

refused_df = pd.DataFrame(refused_arr)
refused_df.columns = features
print refused_df.shape
print refused_df
```

```
(1, 44)
   RIAGENDR  RIDAGEYR  RIDRETH1  DMDEDUC2  DMDMARTL  INDHHIN2  INDFMPIR  \
0       0.0       0.0       0.0       7.0      77.0      77.0       0.0

   BPQ020    CBD070    CBD090    CBD120    CBD130  DIQ010  DBD895  DBD900  \
0     7.0  777777.0  777777.0  777777.0  777777.0     7.0  7777.0  7777.0

   DBD905  DBD910  MCQ080  MCQ365A  MCQ365B  PAD680  SMQ020  WHD010  WHD020  \
0  7777.0  7777.0     7.0      7.0      7.0  7777.0     7.0  7777.0  7777.0

   HEQ010  HEQ030  HUQ051  HUQ071  MCQ010  MCQ082  MCQ086  MCQ160N  MCQ160B  \
0     7.0     7.0    77.0     7.0     7.0     7.0     7.0      7.0      7.0

   MCQ160C  MCQ160D  MCQ160E  MCQ160F  MCQ160G  MCQ160M  MCQ160K  MCQ160L  \
0      7.0      7.0      7.0      7.0      7.0      7.0      7.0      7.0

   MCQ160O  MCQ203  MCQ220
0      7.0     7.0     7.0
```

Create array of don't know values (each entry corresponds to each variable of the nhanes_data dataframe).

In [7]:
```
dk_ndx = [0, 0, 0, 9, 99, 99, 0, 9, 999999, 999999, 999999, 999999, 9, 9999, 9999, 9999, 9999, 9, 9, 9, 9999, 9, 9999, 9999
, 9, 9, 99, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
dk_arr = np.zeros((1, len(dk_ndx)))
dk_arr[0] = dk_ndx

dk_df = pd.DataFrame(dk_arr)
dk_df.columns = features
print dk_df.shape
print dk_df
```

```
(1, 44)
   RIAGENDR  RIDAGEYR  RIDRETH1  DMDEDUC2  DMDMARTL  INDHHIN2  INDFMPIR  \
0       0.0       0.0       0.0       9.0      99.0      99.0       0.0

   BPQ020    CBD070    CBD090    CBD120    CBD130  DIQ010  DBD895  DBD900  \
0     9.0  999999.0  999999.0  999999.0  999999.0     9.0  9999.0  9999.0

   DBD905  DBD910  MCQ080  MCQ365A  MCQ365B  PAD680  SMQ020  WHD010  WHD020  \
0  9999.0  9999.0     9.0      9.0      9.0  9999.0     9.0  9999.0  9999.0

   HEQ010  HEQ030  HUQ051  HUQ071  MCQ010  MCQ082  MCQ086  MCQ160N  MCQ160B  \
0     9.0     9.0    99.0     9.0     9.0     9.0     9.0      9.0      9.0

   MCQ160C  MCQ160D  MCQ160E  MCQ160F  MCQ160G  MCQ160M  MCQ160K  MCQ160L  \
0      9.0      9.0      9.0      9.0      9.0      9.0      9.0      9.0

   MCQ160O  MCQ203  MCQ220
0      9.0     9.0     9.0
```

Loop over the variables and find the refused and don't know entries. Remove any matches found.

In [8]:
```
for i in range(len(features)):
    ref_ndx = nhanes_data[features[i]] == float(refused_df[features[i]])
    nhanes_data = nhanes_data[~ref_ndx]

    dk_ndx = nhanes_data[features[i]] == float(dk_df[features[i]])
    nhanes_data = nhanes_data[~dk_ndx]

print nhanes_data.shape
```

```
(8841, 44)
```

---------- Create Initial Exploratory Plots ----------

Create exploratory plots of all variables and save to folder

The code below creates bar plots for all categorical variables and histograms for all continuous variables. It automatically saves off the plots to a folder specified so I could review them without cluttering up the notebook.

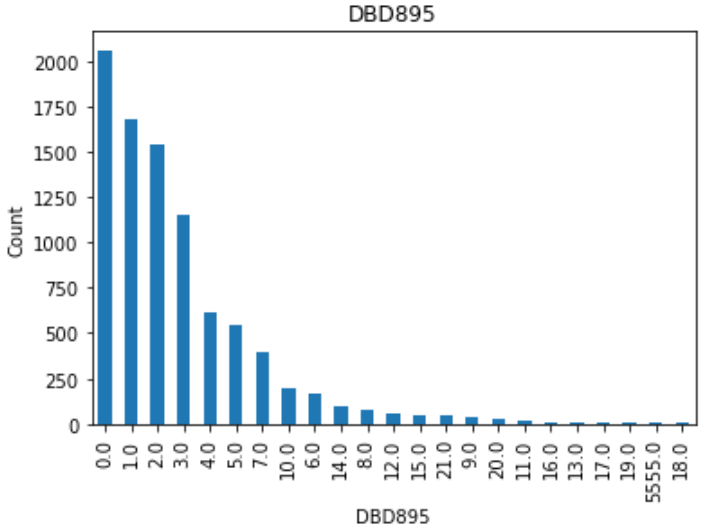
In [9]:
```
for i in range(len(features)):
    if str(nhanes_data[features[i]].dtypes) == 'category':
        nhanes_data[features[i]].value_counts().plot(kind='bar')
        plt.xlabel(features[i])
        plt.ylabel('Count')
        plt.title(features[i])
    else:
        temp_ndx = nhanes_data[features[i]].isnull()
        temp_var = nhanes_data[features[i]][~temp_ndx]

        plt.hist(temp_var, bins=20, alpha=0.5, edgecolor='black', linewidth=1.2)
        plt.xlabel(features[i])
        plt.ylabel('Count')
        plt.title(features[i])
        plt.grid(True)

    figPath = 'C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Exploratory\\Initial Plots\\' + features[i] + '.pn
g'
    plt.savefig(figPath)
    plt.close()
```

---------- Remove 5555 Entries From DBD895 and DBD900 ----------

Reviewing the histograms created above showed that two variables contained 5555 values. This corresponds to 'Over 21'. Since this is value is categorical but the rest of the values of these variables were continuous, and because there were only a couple entries, the 5555 values were deleted.

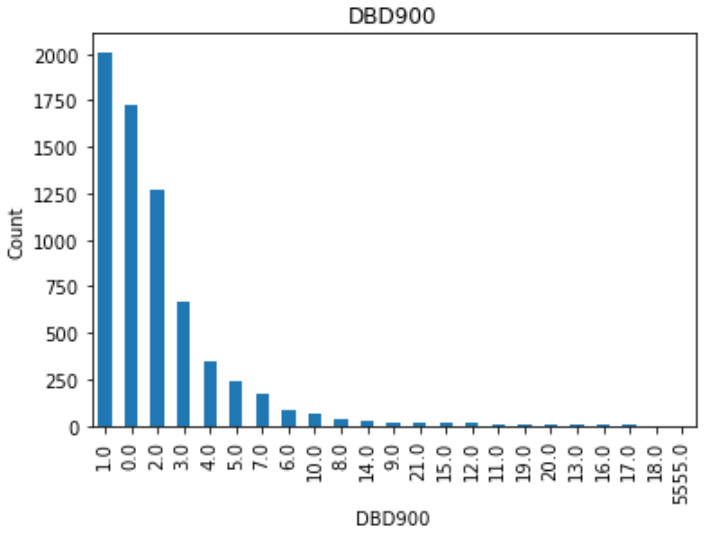
In [10]:
```
nhanes_data['DBD895'].value_counts().plot(kind='bar')
plt.xlabel('DBD895')
plt.ylabel('Count')
plt.title('DBD895')
plt.show
```

Out[10]: <function matplotlib.pyplot.show>



In [11]:
```
nhanes_data['DBD900'].value_counts().plot(kind='bar')
plt.xlabel('DBD900')
plt.ylabel('Count')
plt.title('DBD900')
plt.show
```

Out[11]: <function matplotlib.pyplot.show>



Removal of 5555 entries.

In [12]:
```
temp_ndx = nhanes_data['DBD895'] == 5555
print temp_ndx.sum()
nhanes_data = nhanes_data[~temp_ndx]

temp_ndx = nhanes_data['DBD900'] == 5555
print temp_ndx.sum()
nhanes_data = nhanes_data[~temp_ndx]
```

```
5
0
```

---------- Set Column Types to Match Dataset Types ----------

Variable data types assigned in the read_csv function were not correct for any of the categorical variables. The data types of all of these variables needs to be category, not float64 or int (what was assigned).

In [13]:
```
nhanes_data.dtypes
```

Out[13]:
```
RIAGENDR      int64
RIDAGEYR      int64
RIDRETH1      int64
DMDEDUC2    float64
DMDMARTL    float64
INDHHIN2    float64
INDFMPIR    float64
BPQ020      float64
CBD070      float64
CBD090      float64
CBD120      float64
CBD130      float64
DIQ010      float64
DBD895      float64
DBD900      float64
DBD905      float64
DBD910      float64
MCQ080      float64
MCQ365A     float64
MCQ365B     float64
PAD680      float64
SMQ020      float64
WHD010      float64
WHD020      float64
HEQ010      float64
HEQ030      float64
HUQ051        int64
HUQ071        int64
MCQ010      float64
MCQ082      float64
MCQ086      float64
MCQ160N     float64
MCQ160B     float64
MCQ160C     float64
MCQ160D     float64
MCQ160E     float64
MCQ160F     float64
MCQ160G     float64
MCQ160M     float64
MCQ160K     float64
MCQ160L     float64
MCQ160O     float64
MCQ203      float64
MCQ220      float64
dtype: object
```

Create a list of the variable names that are categorical, then loop over them, changing their data type.

In [14]:
```
cat_attrs = ['RIAGENDR', 'RIDRETH1', 'DMDEDUC2', 'DMDMARTL', 'INDHHIN2', 'BPQ020', 'DIQ010', 'MCQ080', 'MCQ365A', 'MCQ365B'
, 'SMQ020', 'HEQ010', 'HEQ030', 'HUQ051', 'HUQ071', 'MCQ010', 'MCQ082', 'MCQ086', 'MCQ160N', 'MCQ160B', 'MCQ160C', 'MCQ160
D', 'MCQ160E', 'MCQ160F', 'MCQ160G', 'MCQ160M', 'MCQ160K', 'MCQ160L', 'MCQ160O', 'MCQ203', 'MCQ220']

for i in range(len(cat_attrs)):
    nhanes_data[cat_attrs[i]] = nhanes_data[cat_attrs[i]].astype('category')
```

---------- Remove NaN Values ----------

Find all null values and replace with mean if numeric or drop rows if categorical

In [15]:
```
nhanes_data.isnull().sum().sum()
```

Out[15]: 84813

Remove all rows that contain nulls in any categorical variables because there is no way to replace these values with a mean or something similar (arithmetic operations don't apply to categorical variables).

In [16]:
```
for i in range(len(features)):
    if str(nhanes_data[features[i]].dtypes) == 'category':
        nhanes_data.drop(nhanes_data[nhanes_data[features[i]].isnull()].index, axis=0, inplace=True)
```

In [17]:
```
nhanes_data.shape
```

Out[17]: (5055, 44)

In [18]:
```
nhanes_data.isnull().sum().sum()
```

Out[18]: 1358

Find all the variables that still have null values present.

In [19]:
```
null_cnt = nhanes_data.isnull().sum()
null_ndx = null_cnt > 0
print null_cnt[null_ndx]
```

```
INDFMPIR     157
CBD070         4
CBD090         5
CBD120         4
CBD130         4
DBD900      1123
DBD905        13
DBD910        10
PAD680         5
WHD010        16
WHD020        17
dtype: int64
```

Find the percent of rows that are null for the DBD900 attribute.

In [20]:
```
print 'Percent NaN of DBD900 Attribute:'
print (nhanes_data['DBD900'].isnull().sum() / float(nhanes_data.shape[0])) * 100
```

```
Percent NaN of DBD900 Attribute:
22.215628091
```

Since a significant number of points are null in DBD900 (22%), I decided to just remove these rows instead of replace with a mean. The reason that I decided to remove them is that I believe there are too many null values present to get an accurate mean (the mean of the data present will only account for 78% of the data - it may not be representative of the value that should be in the missing 22%). A better way of dealing with this may have been to perform regression with the variables in order to predict the missing 22% (the rows with the 78% present would be training, the rows with the 22% missing would be testing).

In [21]:
```
nhanes_data.drop(nhanes_data[nhanes_data['DBD900'].isnull()].index, axis=0, inplace=True)
```

In [22]:
```
nhanes_data.shape
```

Out[22]: (3932, 44)

In [23]:
```
nhanes_data.isnull().sum().sum()
```

Out[23]: 193

Replace all of the nulls in the remaining continuous variables with the mean of the variable.

In [24]:
```
for i in range(len(features)):
    if str(nhanes_data[features[i]].dtypes) != 'category':
        nhanes_data[features[i]].fillna(nhanes_data[features[i]].mean(), inplace=True)
```

In [25]:
```
nhanes_data.shape
```

Out[25]: (3932, 44)

In [26]:
```
nhanes_data.isnull().sum().sum()
```

Out[26]: 0

When the variables were converted to category from float64, the categorical values all contain one point after the decimal. Since the category values should not have decimal points, I convert them first to integer here then back to category. This will ensure the categorical variables do not contain decimal places. This is necessary for indexing by values.

In [27]:
```
cat_attrs = ['RIAGENDR', 'RIDRETH1', 'DMDEDUC2', 'DMDMARTL', 'INDHHIN2', 'BPQ020', 'DIQ010', 'MCQ080', 'MCQ365A', 'MCQ365B'
, 'SMQ020', 'HEQ010', 'HEQ030', 'HUQ051', 'HUQ071', 'MCQ010', 'MCQ082', 'MCQ086', 'MCQ160N', 'MCQ160B', 'MCQ160C', 'MCQ160
D', 'MCQ160E', 'MCQ160F', 'MCQ160G', 'MCQ160M', 'MCQ160K', 'MCQ160L', 'MCQ160O', 'MCQ203', 'MCQ220']

for i in range(len(cat_attrs)):
    if str(nhanes_data[features[i]].dtypes) == 'category':
        nhanes_data[cat_attrs[i]] = nhanes_data[cat_attrs[i]].astype('int')
        nhanes_data[cat_attrs[i]] = nhanes_data[cat_attrs[i]].astype('category')
```

In [28]:
```
print nhanes_data.head()
```

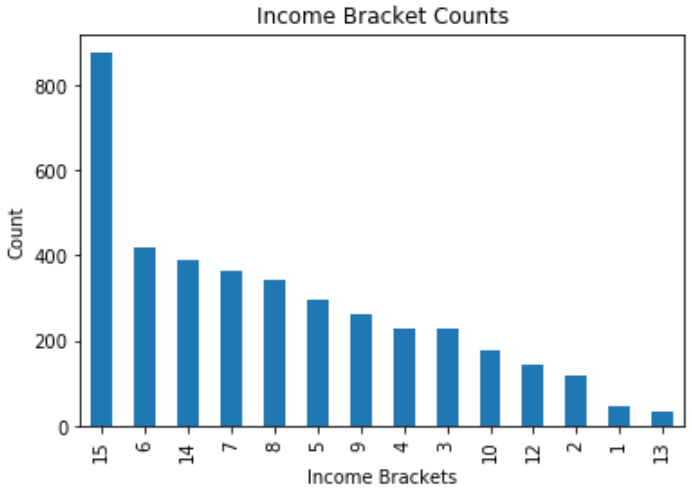
```
      RIAGENDR  RIDAGEYR RIDRETH1 DMDEDUC2 DMDMARTL INDHHIN2  INDFMPIR BPQ020  \
SEQN
73557        1        69        4        3        4        4      0.84      1
73559        1        72        3        4        1       10      4.51      1
73562        1        56        1        4        3        9      4.79      1
73564        2        61        3        5        2       10      5.00      1
73565        1        42        2        3        1       15      5.00      2

       CBD070  CBD090  CBD120  CBD130 DIQ010  DBD895  DBD900  DBD905  DBD910  \
SEQN
73557   300.0     0.0     0.0    85.0    1.0     8.0     8.0     0.0     4.0
73559   150.0    25.0    40.0     0.0    1.0     1.0     0.0     0.0     0.0
73562   150.0    60.0    60.0     0.0    2.0    14.0    14.0     0.0     0.0
73564   400.0   100.0   200.0     0.0    2.0     5.0     1.0     0.0     0.0
73565   900.0     0.0   300.0    40.0    2.0    15.0     2.0     7.0     0.0

      MCQ080 MCQ365A MCQ365B  PAD680 SMQ020  WHD010  WHD020 HEQ010 HEQ030  \
SEQN
73557      1     1.0     2.0   600.0    1.0    69.0   180.0    2.0      2
73559      2     2.0     2.0   300.0    1.0    70.0   195.0    2.0      2
73562      1     1.0     1.0   360.0    1.0    64.0   235.0    2.0      2
73564      1     1.0     1.0    60.0    2.0    64.0   212.0    2.0      2
73565      2     2.0     2.0   300.0    1.0    70.0   200.0    2.0      2

      HUQ051 HUQ071 MCQ010 MCQ082 MCQ086 MCQ160N MCQ160B MCQ160C MCQ160D  \
SEQN
73557      5      2    2.0    2.0      2       2       2     2.0       2
73559      2      2    2.0    2.0      2       2       2     2.0       2
73562      3      1    2.0    2.0      2       1       2     1.0       2
73564      2      2    1.0    2.0      2       2       2     2.0       2
73565      0      2    2.0    2.0      2       2       2     2.0       2

      MCQ160E MCQ160F MCQ160G MCQ160M MCQ160K MCQ160L MCQ160O MCQ203 MCQ220
SEQN
73557     2.0     1.0       2       2       2       2       2      2      2
73559     2.0     2.0       2       2       2       2       2      2      1
73562     1.0     2.0       2       1       2       2       2      2      2
73564     2.0     2.0       2       2       2       2       2      2      2
73565     2.0     2.0       2       2       2       2       2      2      2
```

---------- Remove Income Ranges That Overlap Other Ranges ----------

Upon analysis of the dataset, I found that the income variable contained several income brackets (specified by values 1 through 10, 14, and 15) as well as two income ranges (under 20K was value 13 and over 20K was 12). Since these ranges overlap the other brackets, they were removed. This is because it is not possible to know the exact value of these ranges so they can't be put into the correct bracket. They will also throw off analysis and modelling because there will be multiple variables expressing the same income values if they were left in.

In [29]:
```
nhanes_data["INDHHIN2"].value_counts().plot(kind='bar')
plt.xlabel('Income Brackets')
plt.ylabel('Count')
plt.title('Income Bracket Counts')
```

Out[29]:
```
<matplotlib.text.Text at 0x1534cac8>
```



Removal of income range values 12 and 13.

In [30]:
```
ndx_12 = nhanes_data['INDHHIN2'] == 12
nhanes_data = nhanes_data[~ndx_12]

ndx_13 = nhanes_data['INDHHIN2'] == 13
nhanes_data = nhanes_data[~ndx_13]

nhanes_data.shape
```

Out[30]:
```
(3754, 44)
```

---------- Set Categorical Variables To Their String Values ----------

The categorical values in the dataset all represent actual text strings. In order to make it easier to interpret the variables after dummy variables are generated, I decided to replace the numeric category value of the variables with their actual string meanings.

In [31]:
```
for i in range(len(features)):
     if str(nhanes_data[features[i]].dtypes) == 'category':

        nhanes_data[features[i]] = nhanes_data[features[i]].astype('int')
        nhanes_data[features[i]] = nhanes_data[features[i]].astype('str')

        if features[i] == 'RIAGENDR':
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'Male')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'Female')

        elif features[i] == 'RIDRETH1':
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'Mexican_American')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'Other_Hispanic')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('3' ,'Non_Hispanic_White')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('4' ,'Non_Hispanic_Black')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('5' ,'Other')

        elif features[i] == 'DMDEDUC2':
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'Less_Than_9th')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'9th_to_12th_No_Grad')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('3' ,'High_School_Grad_GED')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('4' ,'Some_College_AA')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('5' ,'College_Grad_And_Above')

        elif features[i] == 'DMDMARTL':
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'Married')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'Widowed')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('3' ,'Divorced')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('4' ,'Separated')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('5' ,'Never_Married')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('6' ,'Living_W_Partner')

        elif features[i] == 'INDHHIN2':
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'0_to_4999')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'5000_to_9999')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('3' ,'10000_to_14999')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('4' ,'15000_to_19999')
            nhanes_data[features[i]] = nhanes_data[features[i]].replace('5' ,'20000_to_24999')
```

```
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('6' ,'25000_to_34999')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('7' ,'35000_to_44999')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('8' ,'45000_to_54999')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('9' ,'55000_to_64999')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('10' ,'65000_to_74999')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('14' ,'75000_to_99999')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('15' ,'100000_And_Over')

    elif features[i] == 'DIQ010':
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'Yes')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'No')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('3' ,'Borderline')

    elif features[i] == 'HUQ051':
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'1')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'2_to_3')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('3' ,'4_to_5')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('4' ,'6_to_7')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('5' ,'8_to_9')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('6' ,'10_to_12')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('7' ,'13_to_15')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('8' ,'16_Or_More')

    else:
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('1' ,'Yes')
        nhanes_data[features[i]] = nhanes_data[features[i]].replace('2' ,'No')

    nhanes_data[features[i]] = nhanes_data[features[i]].astype('category')
```

---------- Create Classification and Regression Variables ----------

Create BMI Percentage and Obesity Indicator variables (Obesity Indicator will become the class variable)

The BMI needs to be computed from the weight in kg and height in m. First these variables had to be converted to the proper units. Once they were in the correct units, the formula BMI = weight / (height^2) is applied.

In [32]:
```
lb_to_kg_const =  0.45359237
ft_to_m_const = 0.3048

nhanes_data['Weight_kg'] = nhanes_data['WHD020'] * lb_to_kg_const
nhanes_data['Height_m'] = nhanes_data['WHD010'] * ft_to_m_const

nhanes_data['BMI_Perc'] = (nhanes_data['Weight_kg'] / (nhanes_data['Height_m']**2)) * 100
```

Next the obesity indicator needs to be created (this is the class variable that will be used in classification and clustering comparisons). The definition of obesity is having a BMI greater than or equal to 30.

In [33]:
```
obese_arr = np.zeros(nhanes_data.shape[0])
obese_ndx = nhanes_data['BMI_Perc'] >= 30
obese_arr[obese_ndx] = 1

nhanes_data['Obese_Ind'] = obese_arr
nhanes_data['Obese_Ind'] = nhanes_data['Obese_Ind'].astype('int')
nhanes_data['Obese_Ind'] = nhanes_data['Obese_Ind'].astype('category')
```

---------- Print Descriptive Statistics ----------

In [34]:
```
num_rows = nhanes_data.shape[0] + 1
num_cols = nhanes_data.shape[1] + 1
pd.set_option('max_rows', num_rows)
pd.set_option('max_columns', num_cols)
np.set_printoptions(threshold=np.inf)
```

Generate all of the descriptive statistics for the entire dataset.

In [35]:
```
nhanes_data.describe(include = "all").T
```

Out[35]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
RIAGENDR	3754	2	Female	1931	NaN	NaN	NaN	NaN	NaN	NaN	NaN
RIDAGEYR	3754	NaN	NaN	NaN	47.3013	17.308	20	33	45	61	80
RIDRETH1	3754	5	Non_Hispanic_White	1730	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DMDEDUC2	3754	5	Some_College_AA	1251	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DMDMARTL	3754	6	Married	1999	NaN	NaN	NaN	NaN	NaN	NaN	NaN
INDHHIN2	3754	12	100000_And_Over	876	NaN	NaN	NaN	NaN	NaN	NaN	NaN
INDFMPIR	3754	NaN	NaN	NaN	2.7216	1.65501	0.01	1.2	2.46	4.55	5
BPQ020	3754	2	No	2449	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CBD070	3754	NaN	NaN	NaN	432.925	314.826	0	200	400	557	4285
CBD090	3754	NaN	NaN	NaN	38.954	71.3283	0	0	0	50	1542
CBD120	3754	NaN	NaN	NaN	178.226	207.793	0	50	100	214	2142
CBD130	3754	NaN	NaN	NaN	27.6372	63.559	0	0	0	35	1028
DIQ010	3754	3	No	3225	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DBD895	3754	NaN	NaN	NaN	4.2171	3.87082	1	2	3	5	21
DBD900	3754	NaN	NaN	NaN	2.03756	2.77602	0	0	1	3	21
DBD905	3754	NaN	NaN	NaN	2.53708	10.153	0	0	0	2	180
DBD910	3754	NaN	NaN	NaN	2.66383	8.61337	0	0	0	2	150
MCQ080	3754	2	No	2374	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ365A	3754	2	No	2695	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ365B	3754	2	No	2350	NaN	NaN	NaN	NaN	NaN	NaN	NaN
PAD680	3754	NaN	NaN	NaN	424.703	197.167	1	240	422.901	540	1200
SMQ020	3754	2	No	2145	NaN	NaN	NaN	NaN	NaN	NaN	NaN
WHD010	3754	NaN	NaN	NaN	66.5843	4.08709	48	64	66	70	80
WHD020	3754	NaN	NaN	NaN	180.49	47.4848	75	148	174	205	493
HEQ010	3754	2	No	3726	NaN	NaN	NaN	NaN	NaN	NaN	NaN
HEQ030	3754	2	No	3714	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
HUQ051	3754	9	2_to_3	1093	NaN	NaN	NaN	NaN	NaN	NaN	NaN
HUQ071	3754	2	No	3343	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ010	3754	2	No	3153	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ082	3754	2	No	3736	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ086	3754	2	No	3685	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160N	3754	2	No	3610	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160B	3754	2	No	3657	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160C	3754	2	No	3625	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160D	3754	2	No	3676	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160E	3754	2	No	3631	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160F	3754	2	No	3656	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160G	3754	2	No	3707	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160M	3754	2	No	3369	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160K	3754	2	No	3554	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160L	3754	2	No	3622	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ160O	3754	2	No	3652	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ203	3754	2	No	3668	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MCQ220	3754	2	No	3380	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Weight_kg	3754	NaN	NaN	NaN	81.8687	21.5387	34.0194	67.1317	78.9251	92.9864	223.621
Height_m	3754	NaN	NaN	NaN	20.2949	1.24574	14.6304	19.5072	20.1168	21.336	24.384
BMI_Perc	3754	NaN	NaN	NaN	19.8079	4.68477	10.3576	16.5307	19.0259	22.052	52.7109
Obese_Ind	3754	2	0	3627	NaN	NaN	NaN	NaN	NaN	NaN	NaN

---------- Exploratory Plot Creation ----------

Create exploratory plots of variables of interest

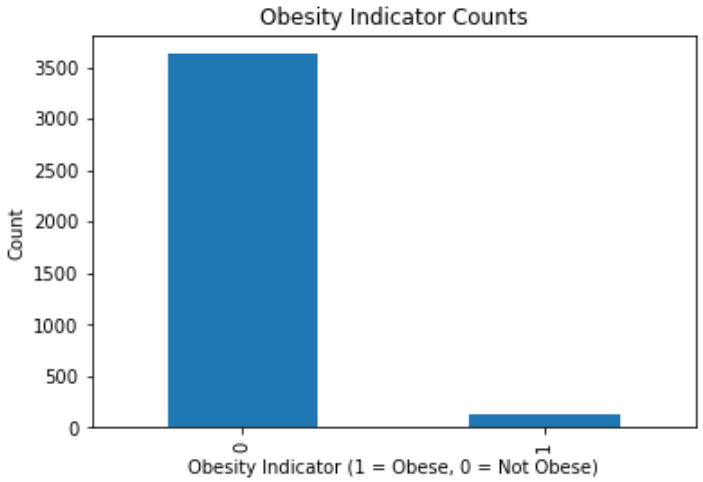
Create new bar plots and histograms as was done at the beginning of the data cleaning process and save them off into a new folder. These plots and histograms will contain the correct category strings, which will make them easier to intepret. I reviewed all of the plots generated to ensure that all outliers and bad values were properly removed from the data, and also to see what the distributions of each variables are like.

In [36]:
```
for i in range(len(features)):
    if str(nhanes_data[features[i]].dtypes) == 'category':
        nhanes_data[features[i]].value_counts().plot(kind='bar')
        plt.xlabel(features[i])
        plt.ylabel('Count')
        plt.title(features[i])
    else:
        temp_ndx = nhanes_data[features[i]].isnull()
        temp_var = nhanes_data[features[i]][~temp_ndx]

        plt.hist(temp_var, bins=20, alpha=0.5, edgecolor='black', linewidth=1.2)
        plt.xlabel(features[i])
        plt.ylabel('Count')
        plt.title(features[i])
        plt.grid(True)

    figPath = 'C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Exploratory\\Processed Plots\\' + features[i] + '.p
ng'
    plt.savefig(figPath)
    plt.close()
```

Below are bar plots and histograms of the variables that appeared to be of the most interest in the dataset.

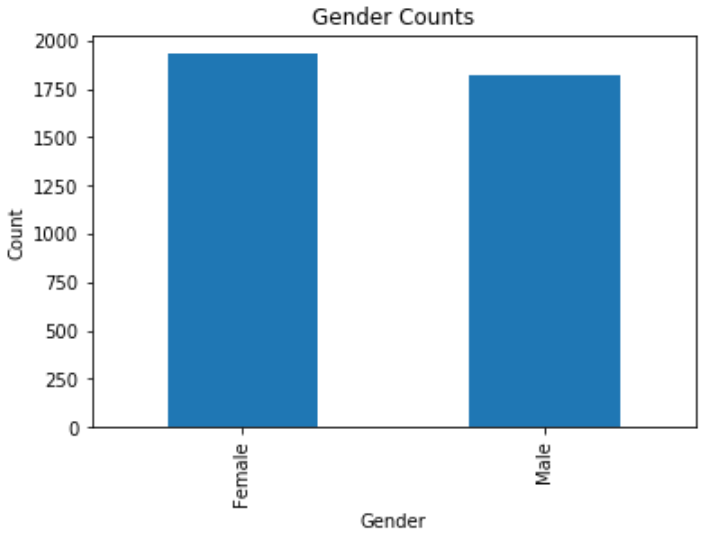
In [37]:
```
nhanes_data["Obese_Ind"].value_counts().plot(kind='bar')
plt.xlabel('Obesity Indicator (1 = Obese, 0 = Not Obese)')
plt.ylabel('Count')
plt.title('Obesity Indicator Counts')
```

Out[37]: <matplotlib.text.Text at 0xf9d3898>



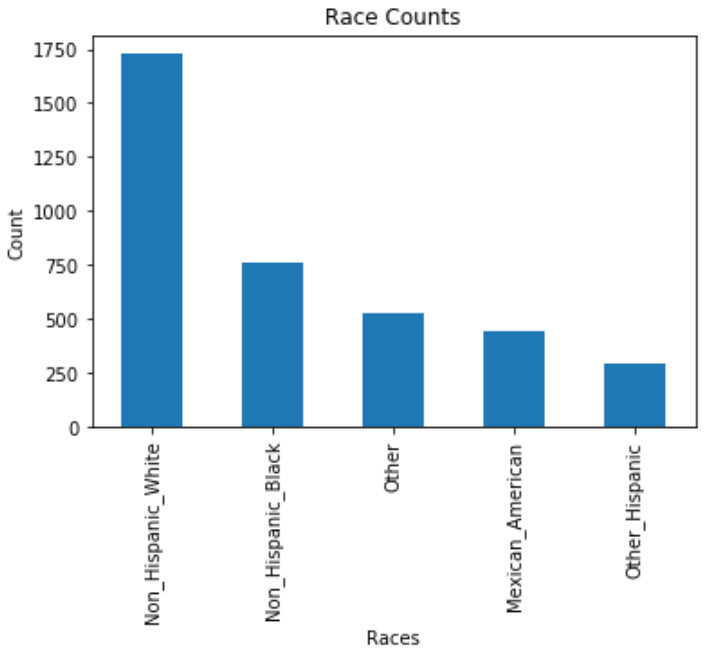
In [38]:
```
nhanes_data["RIAGENDR"].value_counts().plot(kind='bar')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Counts')
```

Out[38]: <matplotlib.text.Text at 0xfc8bfd0>



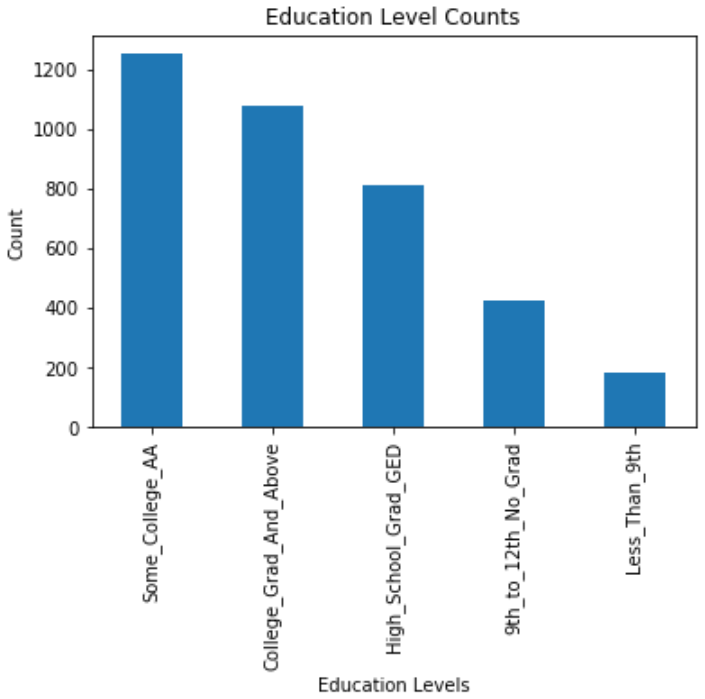
In [39]:
```
nhanes_data["RIDRETH1"].value_counts().plot(kind='bar')
plt.xlabel('Races')
plt.ylabel('Count')
plt.title('Race Counts')
```

Out[39]: <matplotlib.text.Text at 0xee05cc0>



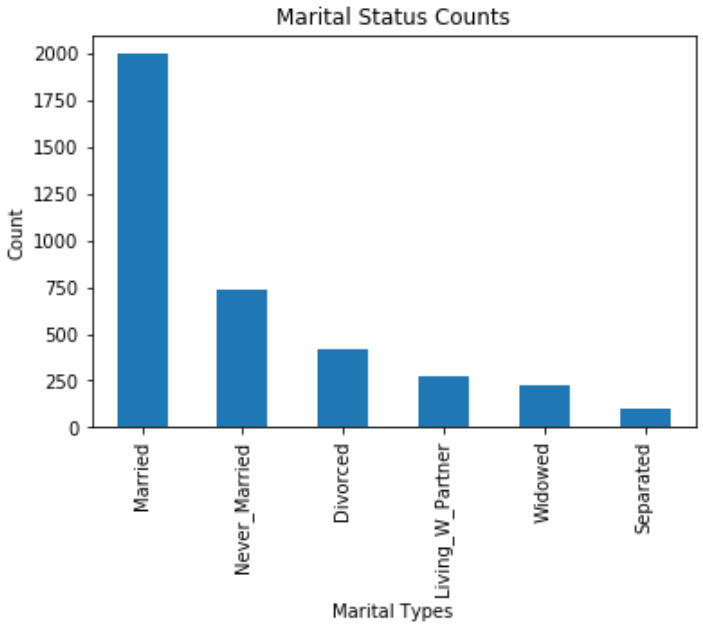
In [40]:
```
nhanes_data["DMDEDUC2"].value_counts().plot(kind='bar')
plt.xlabel('Education Levels')
plt.ylabel('Count')
plt.title('Education Level Counts')
```

Out[40]: <matplotlib.text.Text at 0x142b4b70>



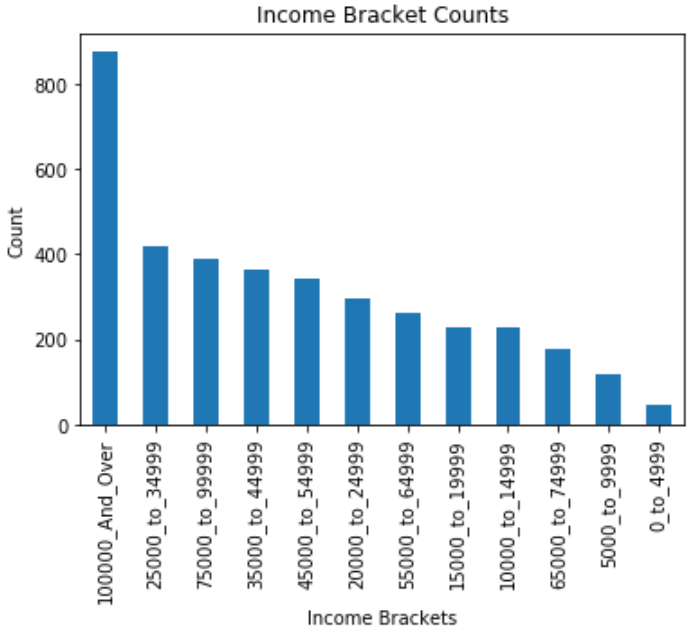
In [41]:
```
nhanes_data["DMDMARTL"].value_counts().plot(kind='bar')
plt.xlabel('Marital Types')
plt.ylabel('Count')
plt.title('Marital Status Counts')
```

Out[41]: <matplotlib.text.Text at 0x1373e470>

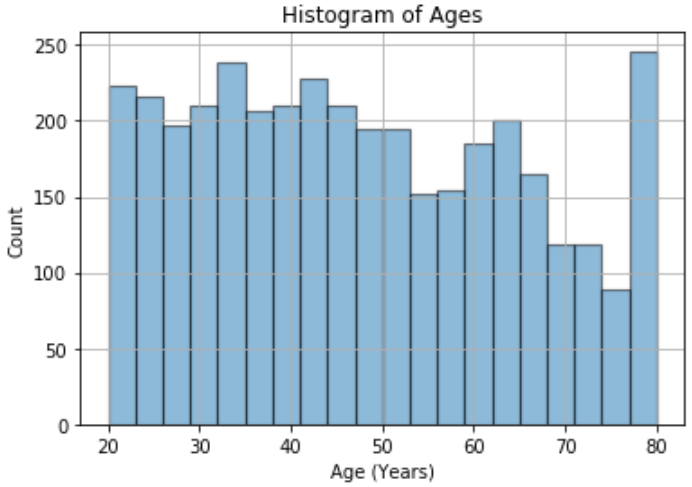


In [42]:
```
nhanes_data["INDHHIN2"].value_counts().plot(kind='bar')
plt.xlabel('Income Brackets')
plt.ylabel('Count')
plt.title('Income Bracket Counts')
```

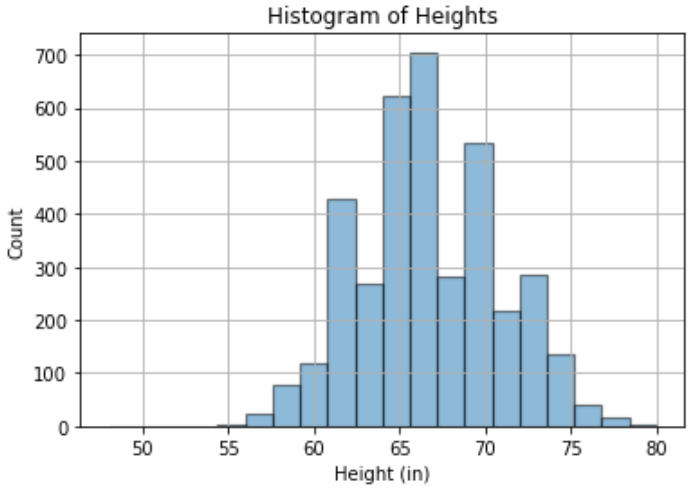
Out[42]: <matplotlib.text.Text at 0xfb80390>



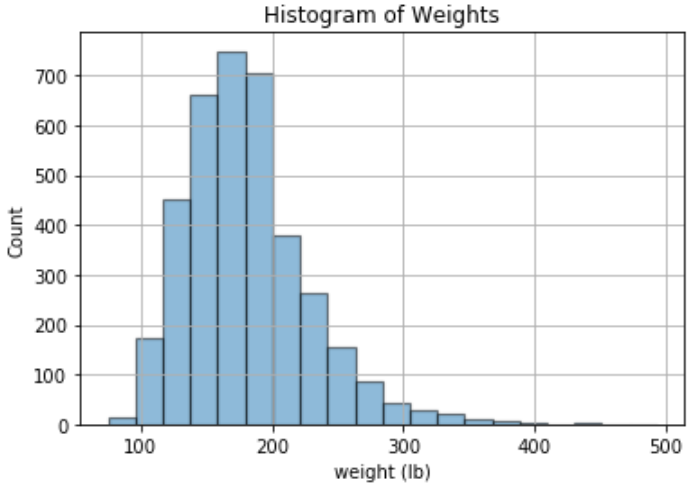
In [43]:
```
plt.hist(nhanes_data["RIDAGEYR"], bins=20, alpha=0.5, edgecolor='black', linewidth=1.2)
plt.xlabel('Age (Years)')
plt.ylabel('Count')
plt.title('Histogram of Ages')
plt.grid(True)
```



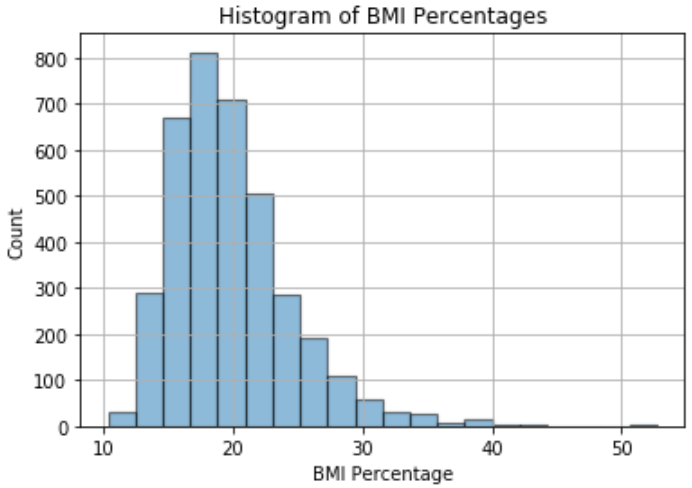
In [44]:
```
plt.hist(nhanes_data["WHD010"], bins=20, alpha=0.5, edgecolor='black', linewidth=1.2)
plt.xlabel('Height (in)')
plt.ylabel('Count')
plt.title('Histogram of Heights')
plt.grid(True)
```



In [45]:
```
plt.hist(nhanes_data["WHD020"], bins=20, alpha=0.5, edgecolor='black', linewidth=1.2)
plt.xlabel('weight (lb)')
plt.ylabel('Count')
plt.title('Histogram of Weights')
plt.grid(True)
```



In [46]:
```
plt.hist(nhanes_data["BMI_Perc"], bins=20, alpha=0.5, edgecolor='black', linewidth=1.2)
plt.xlabel('BMI Percentage')
plt.ylabel('Count')
plt.title('Histogram of BMI Percentages')
plt.grid(True)
```

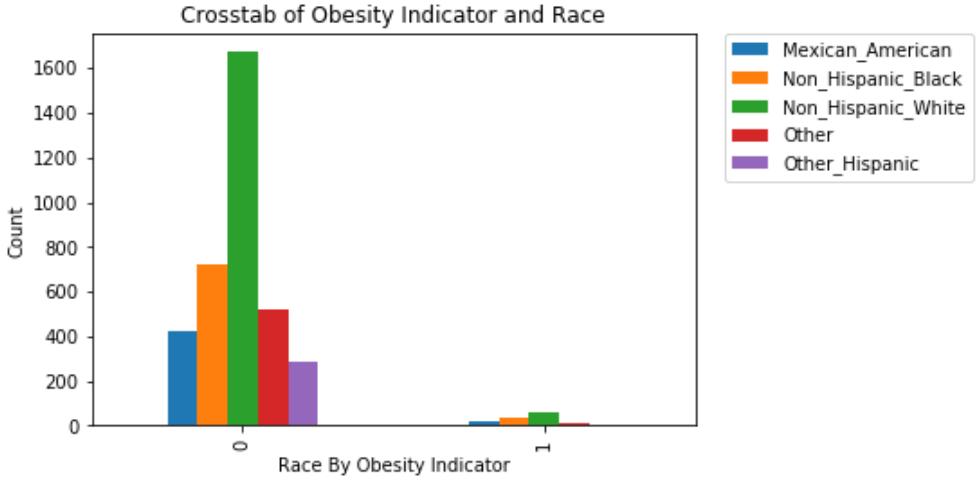


Create exploratory crosstabulation plots of variables of interest

Below are crosstabs of the categorical variables of interest with respect to the obesity indicator class variable.

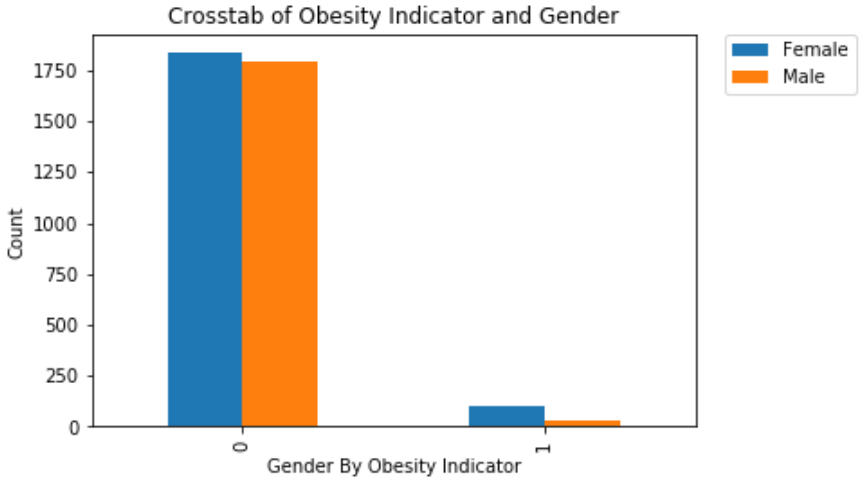
In [47]:
```
obs_race_cross = pd.crosstab(nhanes_data["Obese_Ind"], nhanes_data["RIDRETH1"])
obs_race_cross.plot(kind="bar")
plt.xlabel('Race By Obesity Indicator')
plt.ylabel('Count')
plt.title('Crosstab of Obesity Indicator and Race')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[47]: <matplotlib.legend.Legend at 0x14008ef0>



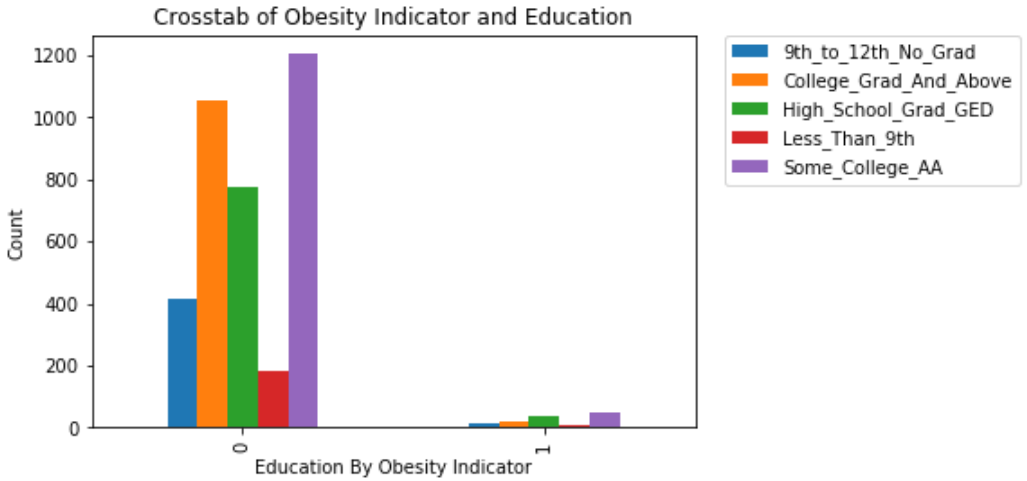
In [48]:
```
obs_race_cross = pd.crosstab(nhanes_data["Obese_Ind"], nhanes_data["RIAGENDR"])
obs_race_cross.plot(kind="bar")
plt.xlabel('Gender By Obesity Indicator')
plt.ylabel('Count')
plt.title('Crosstab of Obesity Indicator and Gender')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[48]: <matplotlib.legend.Legend at 0x10af00b8>



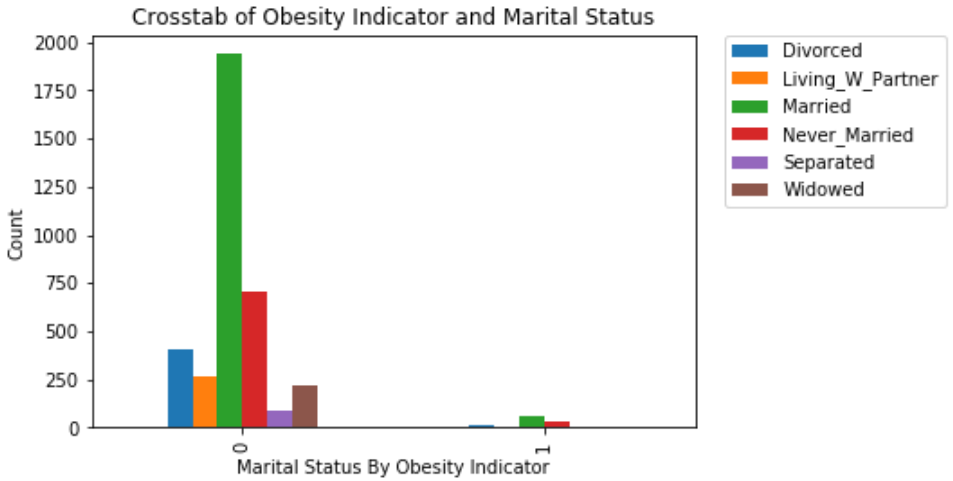
In [49]:
```
obs_ed_cross = pd.crosstab(nhanes_data["Obese_Ind"], nhanes_data["DMDEDUC2"])
obs_ed_cross.plot(kind="bar")
plt.xlabel('Education By Obesity Indicator')
plt.ylabel('Count')
plt.title('Crosstab of Obesity Indicator and Education')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[49]: <matplotlib.legend.Legend at 0xe70af60>



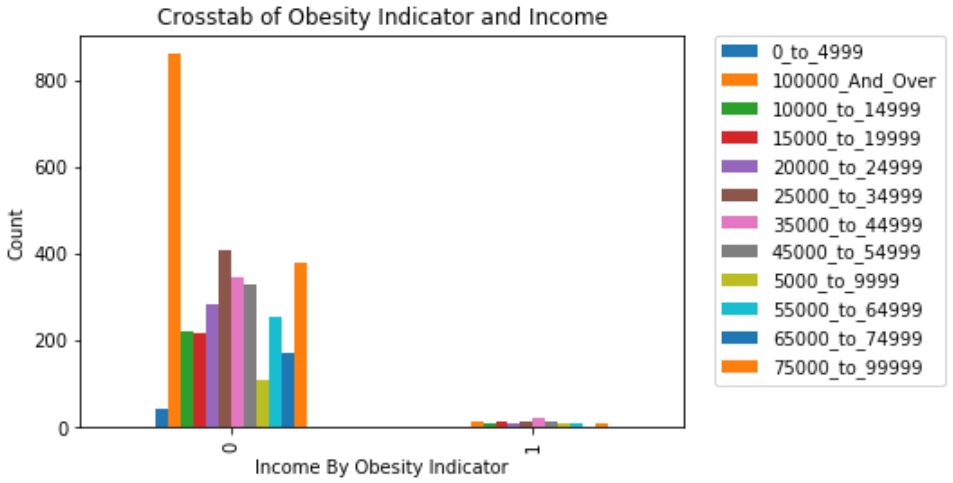
In [50]:
```
obs_mar_cross = pd.crosstab(nhanes_data["Obese_Ind"], nhanes_data["DMDMARTL"])
obs_mar_cross.plot(kind="bar")
plt.xlabel('Marital Status By Obesity Indicator')
plt.ylabel('Count')
plt.title('Crosstab of Obesity Indicator and Marital Status')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[50]: <matplotlib.legend.Legend at 0x13fce198>



In [51]:
```
obs_inc_cross = pd.crosstab(nhanes_data["Obese_Ind"], nhanes_data["INDHHIN2"])
obs_inc_cross.plot(kind="bar")
plt.xlabel('Income By Obesity Indicator')
plt.ylabel('Count')
plt.title('Crosstab of Obesity Indicator and Income')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[51]: <matplotlib.legend.Legend at 0x11cd8cc0>

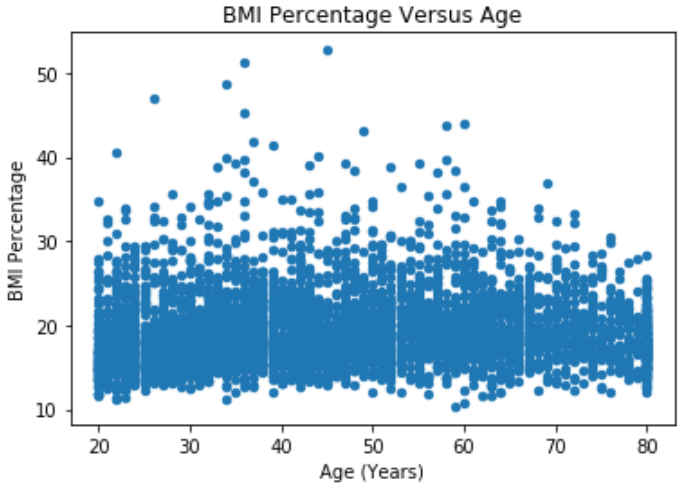


Create exploratory scatter plots of variables of interest

Below are scatterplots of the continuous variables of interest with BMI.

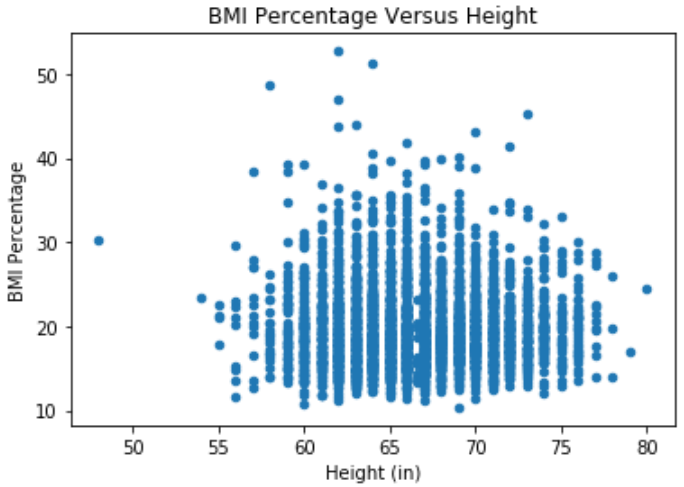
In [52]:
```
nhanes_data.plot(x="RIDAGEYR", y="BMI_Perc", kind="scatter")
plt.xlabel('Age (Years)')
plt.ylabel('BMI Percentage')
plt.title('BMI Percentage Versus Age')
```

Out[52]: <matplotlib.text.Text at 0x1302efd0>



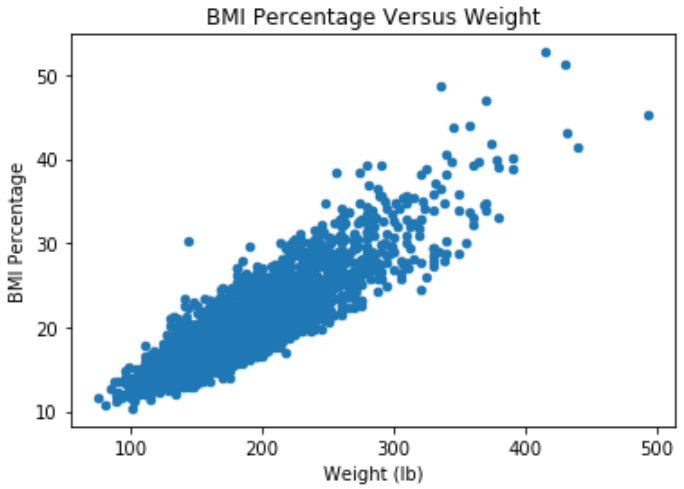
In [53]:
```
nhanes_data.plot(x="WHD010", y="BMI_Perc", kind="scatter")
plt.xlabel('Height (in)')
plt.ylabel('BMI Percentage')
plt.title('BMI Percentage Versus Height')
```

Out[53]: <matplotlib.text.Text at 0x143e9d68>



In [54]:
```
nhanes_data.plot(x="WHD020", y="BMI_Perc", kind="scatter")
plt.xlabel('Weight (lb)')
plt.ylabel('BMI Percentage')
plt.title('BMI Percentage Versus Weight')
```

Out[54]: <matplotlib.text.Text at 0x14b3b4e0>



---------- Create All Variable Correlation ----------

Create dummy variables of all categorical variables for correlation

In [55]:
```
nhanes_spdsht_all = pd.get_dummies(nhanes_data)
```

In [56]:
```
num_rows = nhanes_spdsht_all.shape[0] + 1
num_cols = nhanes_spdsht_all.shape[1] + 1
pd.set_option('max_rows', num_rows)
pd.set_option('max_columns', num_cols)
np.set_printoptions(threshold=np.inf)
```

Create correlation matrix of entire dataframe

In [57]:
```
nhanes_corr = nhanes_spdsht_all.corr(method="pearson")
nhanes_corr.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NHANES Correlation Output.csv', sep=
",")
nhanes_corr
```

Out[57]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	
RIDAGEYR	1.000000	0.099393	-0.089679	-0.007419	-0.042008	-0.078684	-0.177204	-0.227756	-0.042923	-
INDFMPIR	0.099393	1.000000	0.153555	0.145180	0.363103	0.056779	0.110298	-0.080414	0.047889	-
CBD070	-0.089679	0.153555	1.000000	0.374941	0.249199	0.095908	-0.034514	-0.052953	0.013243	-
CBD090	-0.007419	0.145180	0.374941	1.000000	0.152694	0.045916	0.021800	-0.012895	0.061957	0
CBD120	-0.042008	0.363103	0.249199	0.152694	1.000000	0.183396	0.298335	0.109256	0.043551	-
CBD130	-0.078684	0.056779	0.095908	0.045916	0.183396	1.000000	0.125116	0.116527	0.042642	0
DBD895	-0.177204	0.110298	-0.034514	0.021800	0.298335	0.125116	1.000000	0.615070	0.113556	0
DBD900	-0.227756	-0.080414	-0.052953	-0.012895	0.109256	0.116527	0.615070	1.000000	0.040568	0
DBD905	-0.042923	0.047889	0.013243	0.061957	0.043551	0.042642	0.113556	0.040568	1.000000	0
DBD910	-0.075917	-0.025653	-0.026869	0.001910	-0.026863	0.016920	0.048415	0.088500	0.136684	1
PAD680	0.033973	0.158692	-0.040123	-0.026734	0.066463	0.047901	0.093967	0.049787	0.029428	0
WHD010	-0.048542	0.122049	0.014750	0.024510	0.071488	0.037525	0.158090	0.101896	-0.003574	-
WHD020	0.001725	-0.035297	-0.023013	0.009708	-0.009764	0.000120	0.084329	0.110726	-0.009853	-
Weight_kg	0.001725	-0.035297	-0.023013	0.009708	-0.009764	0.000120	0.084329	0.110726	-0.009853	-
Height_m	-0.048542	0.122049	0.014750	0.024510	0.071488	0.037525	0.158090	0.101896	-0.003574	-
BMI_Perc	0.029551	-0.108154	-0.034706	-0.001992	-0.050769	-0.019116	0.011235	0.073385	-0.006745	-
RIAGENDR_Female	-0.020874	-0.054245	-0.029668	0.000388	-0.042587	-0.035254	-0.156471	-0.122995	-0.009016	0
RIAGENDR_Male	0.020874	0.054245	0.029668	-0.000388	0.042587	0.035254	0.156471	0.122995	0.009016	-
RIDRETH1_Mexican_American	-0.066194	-0.155170	0.069636	0.034104	-0.009133	-0.032737	-0.025468	0.030665	0.004673	-
RIDRETH1_Non_Hispanic_Black	-0.027693	-0.124824	-0.151398	-0.102273	-0.124117	0.042686	0.041052	0.137841	-0.011759	-
RIDRETH1_Non_Hispanic_White	0.143494	0.121916	0.028924	0.108906	0.045517	-0.007470	-0.001600	-0.057756	-0.026332	0
RIDRETH1_Other	-0.084658	0.151172	0.055185	-0.030047	0.072643	0.029763	-0.011805	-0.083320	0.059140	-
RIDRETH1_Other_Hispanic	-0.035824	-0.048641	0.017803	-0.051491	0.018376	-0.049451	-0.012621	-0.028358	-0.015799	-
DMDEDUC2_9th_to_12th_No_Grad	0.009640	-0.232457	0.012549	-0.032369	-0.099915	-0.003241	-0.049791	0.028664	-0.022152	-
DMDEDUC2_College_Grad_And_Above	0.008845	0.449516	0.088721	0.064201	0.210059	0.040781	0.069925	-0.130827	0.047076	-
DMDEDUC2_High_School_Grad_GED	0.021527	-0.166539	-0.061621	-0.032949	-0.097556	-0.007460	-0.034506	0.059055	-0.019300	0

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	
DMDEDUC2_Less_Than_9th	0.097458	-0.194793	0.020442	-0.044724	-0.060772	-0.033253	-0.052770	-0.040681	-0.022720	-
DMDEDUC2_Some_College_AA	-0.078663	-0.039277	-0.049134	0.009619	-0.020914	-0.015109	0.020939	0.073298	-0.002899	0
DMDMARTL_Divorced	0.113703	-0.078043	-0.132586	-0.056849	-0.096092	-0.025819	-0.000040	-0.002367	0.014245	0
DMDMARTL_Living_W_Partner	-0.190165	-0.109432	-0.013320	-0.007157	0.001701	0.052830	0.039297	0.090655	0.006440	-
DMDMARTL_Married	0.182042	0.275869	0.213177	0.100029	0.169314	0.012628	-0.102630	-0.145624	-0.018127	-
DMDMARTL_Never_Married	-0.404484	-0.134454	-0.090167	-0.048240	-0.068640	0.010625	0.145794	0.164482	0.025420	0
DMDMARTL_Separated	0.008979	-0.084050	-0.036538	-0.033708	-0.030569	-0.008210	0.009661	0.042189	-0.017547	0
DMDMARTL_Widowed	0.347756	-0.075114	-0.082101	-0.023664	-0.094921	-0.062638	-0.078345	-0.094164	-0.018866	-
INDHHIN2_0_to_4999	-0.046152	-0.174874	-0.054410	-0.048478	-0.065535	-0.027063	0.003136	-0.007614	-0.001122	0
INDHHIN2_100000_And_Over	-0.038537	0.675450	0.252671	0.145922	0.359529	0.076455	0.097281	-0.073267	0.029792	-
INDHHIN2_10000_to_14999	0.053254	-0.299000	-0.106712	-0.074733	-0.121861	-0.032380	-0.049918	-0.021465	-0.036541	-
INDHHIN2_15000_to_19999	0.046707	-0.268907	-0.075998	-0.051695	-0.095662	-0.057214	-0.052214	-0.015863	-0.009796	0
INDHHIN2_20000_to_24999	0.001037	-0.283887	-0.051824	-0.054224	-0.105638	-0.024051	-0.045029	0.025651	-0.016816	0
INDHHIN2_25000_to_34999	-0.046612	-0.270033	-0.049207	-0.038518	-0.111943	-0.005633	-0.018818	0.057002	-0.023190	0
INDHHIN2_35000_to_44999	0.018875	-0.119561	-0.052031	-0.016457	-0.076732	0.007726	-0.004283	0.035347	0.011549	0
INDHHIN2_45000_to_54999	0.029097	-0.037334	0.001758	-0.001585	-0.023515	0.002436	-0.007277	-0.011952	0.010271	-
INDHHIN2_5000_to_9999	0.004539	-0.241133	-0.093502	-0.042268	-0.085639	-0.037817	-0.036928	-0.001338	-0.014409	-
INDHHIN2_55000_to_64999	0.002277	0.080248	-0.012059	-0.019490	-0.020175	0.013095	0.025321	0.016590	-0.012966	-
INDHHIN2_65000_to_74999	-0.007649	0.091134	0.014111	0.029166	0.015933	-0.027899	0.012850	0.004687	0.010515	-
INDHHIN2_75000_to_99999	-0.006281	0.261381	0.012187	0.039894	0.056611	0.021465	0.005488	0.012692	0.025170	0
BPQ020_No	-0.432572	0.023384	0.079921	0.023781	0.067802	0.022019	0.054678	0.057640	0.043693	0
BPQ020_Yes	0.432572	-0.023384	-0.079921	-0.023781	-0.067802	-0.022019	-0.054678	-0.057640	-0.043693	-
DIQ010_Borderline	0.093108	-0.015751	0.010134	-0.003809	-0.020937	-0.016037	-0.018359	0.002607	-0.015715	-
DIQ010_No	-0.276485	0.039185	0.019127	0.018515	0.057233	0.015650	0.041709	0.029754	0.019037	0
DIQ010_Yes	0.255886	-0.034860	-0.026821	-0.018470	-0.052054	-0.008562	-0.036228	-0.034483	-0.012501	-
MCQ080_No	-0.097347	0.011238	0.009731	-0.020394	0.008067	0.031039	0.011363	0.000166	-0.004086	-
MCQ080_Yes	0.097347	-0.011238	-0.009731	0.020394	-0.008067	-0.031039	-0.011363	-0.000166	0.004086	0

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	
MCQ365A_No	-0.115472	-0.008706	-0.008939	-0.011976	-0.005493	0.018729	0.009775	0.020638	0.029742	-
MCQ365A_Yes	0.115472	0.008706	0.008939	0.011976	0.005493	-0.018729	-0.009775	-0.020638	-0.029742	0
MCQ365B_No	-0.166346	-0.017910	0.018381	0.010261	0.004067	0.029693	0.013913	0.035847	0.025153	0
MCQ365B_Yes	0.166346	0.017910	-0.018381	-0.010261	-0.004067	-0.029693	-0.013913	-0.035847	-0.025153	-
SMQ020_No	-0.118390	0.132252	0.026582	-0.000728	0.087512	-0.012230	0.024800	-0.028617	0.022660	0
SMQ020_Yes	0.118390	-0.132252	-0.026582	0.000728	-0.087512	0.012230	-0.024800	0.028617	-0.022660	-
HEQ010_No	-0.030872	0.015762	-0.011685	-0.021631	-0.010426	-0.019007	0.008862	-0.001058	-0.031706	0
HEQ010_Yes	0.030872	-0.015762	0.011685	0.021631	0.010426	0.019007	-0.008862	0.001058	0.031706	-
HEQ030_No	-0.067006	0.051322	0.032490	-0.018402	0.038150	0.021004	-0.004234	-0.008878	0.005347	0
HEQ030_Yes	0.067006	-0.051322	-0.032490	0.018402	-0.038150	-0.021004	0.004234	0.008878	-0.005347	-
HUQ051_0	-0.198799	-0.115231	0.024809	-0.011630	-0.023490	0.005722	0.037325	0.084737	0.048887	0
HUQ051_1	-0.120781	0.028031	0.031616	0.008356	0.023071	0.034466	0.010699	-0.003389	0.018900	-
HUQ051_10_to_12	0.082637	-0.030875	0.008005	-0.020501	-0.030113	-0.006180	-0.032224	-0.013734	0.011557	0
HUQ051_13_to_15	0.007207	-0.042917	-0.021314	-0.043910	-0.024385	-0.017211	-0.020213	-0.015357	-0.024211	-
HUQ051_16_Or_More	0.045884	-0.036956	0.007047	-0.004092	-0.035866	-0.001189	-0.036644	-0.013296	-0.005205	-
HUQ051_2_to_3	0.010595	0.085068	-0.031551	0.006382	0.023849	-0.015976	0.010712	0.002101	-0.006059	0
HUQ051_4_to_5	0.125183	0.023877	-0.029063	0.014031	0.020469	-0.002323	-0.002650	-0.036529	-0.026967	0
HUQ051_6_to_7	0.113504	0.007251	-0.007338	-0.002131	0.013541	0.012373	0.011221	-0.013856	-0.022030	0
HUQ051_8_to_9	0.055231	0.015782	0.028797	0.031167	-0.021703	-0.035405	-0.034346	-0.030083	-0.027898	-
HUQ071_No	-0.096927	0.118641	0.060961	0.028160	0.065234	0.024740	0.057802	0.021035	0.010940	0
HUQ071_Yes	0.096927	-0.118641	-0.060961	-0.028160	-0.065234	-0.024740	-0.057802	-0.021035	-0.010940	-
MCQ010_No	0.038832	0.045027	0.015928	-0.004020	0.052512	0.021490	0.001216	-0.008225	0.020840	-
MCQ010_Yes	-0.038832	-0.045027	-0.015928	0.004020	-0.052512	-0.021490	-0.001216	0.008225	-0.020840	0
MCQ082_No	-0.014612	-0.009487	0.002887	0.006876	-0.008499	0.007128	0.024817	0.024557	-0.008863	0
MCQ082_Yes	0.014612	0.009487	-0.002887	-0.006876	0.008499	-0.007128	-0.024817	-0.024557	0.008863	-
MCQ086_No	0.018311	-0.038098	-0.012097	-0.007012	-0.016202	0.008518	0.025098	0.051865	0.000292	0
MCQ086_Yes	-0.018311	0.038098	0.012097	0.007012	0.016202	-0.008518	-0.025098	-0.051865	-0.000292	-

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	
MCQ160N_No	-0.160419	-0.009940	0.031451	0.012318	-0.005237	-0.004763	-0.019974	0.011697	0.005631	0
MCQ160N_Yes	0.160419	0.009940	-0.031451	-0.012318	0.005237	0.004763	0.019974	-0.011697	-0.005631	-
MCQ160B_No	-0.167532	0.070847	0.038759	0.018281	0.045028	0.017855	0.028657	0.033054	0.017382	-
MCQ160B_Yes	0.167532	-0.070847	-0.038759	-0.018281	-0.045028	-0.017855	-0.028657	-0.033054	-0.017382	0
MCQ160C_No	-0.224615	0.013270	0.044474	0.028810	0.029577	0.021336	0.025695	0.023627	0.023296	-
MCQ160C_Yes	0.224615	-0.013270	-0.044474	-0.028810	-0.029577	-0.021336	-0.025695	-0.023627	-0.023296	0
MCQ160D_No	-0.148219	0.032846	0.022925	-0.007478	0.041245	0.013274	0.027955	0.024847	0.015433	0
MCQ160D_Yes	0.148219	-0.032846	-0.022925	0.007478	-0.041245	-0.013274	-0.027955	-0.024847	-0.015433	-
MCQ160E_No	-0.202498	0.041041	0.038494	0.005378	0.051947	-0.000203	0.048600	0.031063	0.019825	-
MCQ160E_Yes	0.202498	-0.041041	-0.038494	-0.005378	-0.051947	0.000203	-0.048600	-0.031063	-0.019825	0
MCQ160F_No	-0.184914	0.062056	0.043985	0.017252	0.044042	0.031348	0.043717	0.028097	0.005535	0
MCQ160F_Yes	0.184914	-0.062056	-0.043985	-0.017252	-0.044042	-0.031348	-0.043717	-0.028097	-0.005535	-
MCQ160G_No	-0.120417	0.047176	0.032737	0.001708	0.049198	-0.002829	0.057694	0.036912	0.013037	0
MCQ160G_Yes	0.120417	-0.047176	-0.032737	-0.001708	-0.049198	0.002829	-0.057694	-0.036912	-0.013037	-
MCQ160M_No	-0.170884	0.012780	0.011236	-0.006842	0.018202	0.015852	0.044145	0.052658	0.028167	0
MCQ160M_Yes	0.170884	-0.012780	-0.011236	0.006842	-0.018202	-0.015852	-0.044145	-0.052658	-0.028167	-
MCQ160K_No	-0.090865	0.075505	0.040585	0.009393	0.057689	0.008444	0.009936	-0.005764	0.011236	-
MCQ160K_Yes	0.090865	-0.075505	-0.040585	-0.009393	-0.057689	-0.008444	-0.009936	0.005764	-0.011236	0
MCQ160L_No	-0.071137	0.031385	-0.006427	0.005900	0.011637	-0.005095	0.028271	0.013525	0.013724	0
MCQ160L_Yes	0.071137	-0.031385	0.006427	-0.005900	-0.011637	0.005095	-0.028271	-0.013525	-0.013724	-
MCQ160O_No	-0.167320	0.048321	0.048903	0.007244	0.069429	0.013303	0.077109	0.047124	0.028048	0
MCQ160O_Yes	0.167320	-0.048321	-0.048903	-0.007244	-0.069429	-0.013303	-0.077109	-0.047124	-0.028048	-
MCQ203_No	-0.046410	0.009358	0.020026	0.026339	0.011298	0.022408	0.010429	0.009770	-0.018909	0
MCQ203_Yes	0.046410	-0.009358	-0.020026	-0.026339	-0.011298	-0.022408	-0.010429	-0.009770	0.018909	-
MCQ220_No	-0.315372	-0.063117	0.057866	0.003041	0.006915	0.021264	0.069217	0.069870	0.036348	0
MCQ220_Yes	0.315372	0.063117	-0.057866	-0.003041	-0.006915	-0.021264	-0.069217	-0.069870	-0.036348	-
Obese_Ind_0	0.033993	0.069961	0.020387	-0.000369	0.038002	0.016970	0.012400	-0.029849	-0.034223	-

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905
Obese_Ind_1	-0.033993	-0.069961	-0.020387	0.000369	-0.038002	-0.016970	-0.012400	0.029849	0.034223

-------- Create variables for Classification and Clustering ----------

Remove create train and target dataframes

Create the class target variable dataframe of the obesity indicator.

In [58]:
```
target_df = nhanes_data['Obese_Ind']
target_df.head()
```

Out[58]:
```
SEQN
73557    0
73559    0
73562    0
73564    0
73565    0
Name: Obese_Ind, dtype: category
Categories (2, int64): [0, 1]
```

Create a training dataframe then remove the obesity indicator from the training dataframe (this is the target and shouldn't be in the training).

In [59]:
```
train_df = nhanes_data[:]
train_df = train_df.drop('Obese_Ind', 1)
```

Remove all of the variables that were used to create the obesity indicator variable. This was done because these variables would have a strong relation to the obesity indicator due to its calculation. These variable could take over in any modelling and prediction because of their strong relation. Since we already know they are related because they were used in the calculation, they are also of no interest.

Note that this code was run 2 different ways. The first way was with WHD010 and WHD020 removed as is shown below (these are the original height and weight variables used to calculate BMI). The second way was to leave these variables in the dataset to see what impact they have. I found that there was fairly poor performance with modelling and prediction without these variables and wanted to see how performance was affected by them.

In [60]:
```
train_df = train_df.drop('BMI_Perc', 1)
train_df = train_df.drop('Height_m', 1)
train_df = train_df.drop('Weight_kg', 1)
train_df = train_df.drop('WHD010', 1)
train_df = train_df.drop('WHD020', 1)
train_df.head()
```

Out[60]:

	RIAGENDR	RIDAGEYR	RIDRETH1	DMDEDUC2	DMDMARTL	INDHHIN2	INDFMPIR	BPQ020	CBD070
SEQN									
73557	Male	69	Non_Hispanic_Black	High_School_Grad_GED	Separated	15000_to_19999	0.84	Yes	300.0
73559	Male	72	Non_Hispanic_White	Some_College_AA	Married	65000_to_74999	4.51	Yes	150.0
73562	Male	56	Mexican_American	Some_College_AA	Divorced	55000_to_64999	4.79	Yes	150.0
73564	Female	61	Non_Hispanic_White	College_Grad_And_Above	Widowed	65000_to_74999	5.00	Yes	400.0
73565	Male	42	Other_Hispanic	High_School_Grad_GED	Married	100000_And_Over	5.00	No	900.0

Create dummy variables of all categorical variables for classification

Create dummy variables for all of the categorical variables in training dataset.

In [61]:
```
train_df_spdsht = pd.get_dummies(train_df)
```

Split data into testing and training for classification and clustering

Split the dataset into 80% training and 20% testing randomly using a given random state to ensure repeatability.

In [62]:
```
data_train, data_test, target_train, target_test = train_test_split(train_df_spdsht, target_df, test_size = 0.2, random_sta
te = 45)
```

Apply max/min normalization to data

Create normalized target and training datasets using min/max normalization. These normalized datasets will be used in the KNN classification and K-Means clustering.

In [63]:
```
min_max_scaler = preprocessing.MinMaxScaler().fit(data_train)
data_train_norm_np = min_max_scaler.transform(data_train)
data_test_norm_np = min_max_scaler.transform(data_test)
```

Convert target variables into numpy arrays for sklearn functions

In [64]:
```
data_train_np = np.array(data_train)
data_test_np = np.array(data_test)
target_train_np = np.array(target_train)
target_test_np = np.array(target_test)

print data_train_np.shape
print target_train_np.shape
print data_test_np.shape
print target_test_np.shape
```

```
(3003L, 101L)
(3003L,)
(751L, 101L)
(751L,)
```

---------- Classification ----------

Create Functions Needed For Classification

The find_percent function calculates the accuracy for a range of percentage of features passed in. The optimal percent is then given as the percent with the maximum accuracy value. If more than one of the percentages have the same accuracy value, the last it taken. I use the feature_selection.SelectPercentile function with chi2 selection to select the i percent of variables in the dataset (i is given in the loop containing the function call). Cross validation modelling is then done with the cross_validation.cross_val_score function to calculate the accuracy of each fold. The mean of these accuracy values is then used as the score for the i percentage.

In [65]:
```
def find_percent(train_arr, target_arr, model, percent_list, cv_num):

    print '---------------------------------- Feature Selection ----------------------------------'

    results = []
    feature_select = []
    feature_scores = []
    features_pvals = []
    for i in percent_list: # loop over the list of percentages passed in

        # extract the ith percentage of variables for use in cross validation
        fs = feature_selection.SelectPercentile(feature_selection.chi2, percentile = i)
        train_arr_fs = fs.fit_transform(train_arr, target_arr)

        # calculate cross validation accuracy and save them, the mean accuracy, the weights, and the p values into lists
        scores = cross_validation.cross_val_score(model, train_arr_fs, target_arr, cv = cv_num, scoring = "accuracy")
        results = np.append(results, scores.mean())
        feature_select.append(fs.get_support())
        feature_scores.append(fs.scores_)
        features_pvals.append(fs.pvalues_)

    # find the maximum mean accuracy and use as the optimal index - if there more than 1, take the highest percentage value
    optimal_ndx = np.where(results == results.max())[0]
    if len(optimal_ndx) > 1:
        optimal_ndx = optimal_ndx[-1]

    # get the optimal percentage and the corresponding number of features
    optimal_percentile = int(percent_list[optimal_ndx])
    optimal_num_features = int(optimal_percentile * (train_arr.shape[1])/100)

    # get the optimal weights, scores, and p values and store into final variables
    chosen_features = feature_select[int(optimal_ndx)]
    chosen_weights = feature_scores[int(optimal_ndx)]
    chosen_pvals = features_pvals[int(optimal_ndx)]

    # plot the percentages vs accuracy and mark the optimal value chosen with an X
    plt.figure()
    plt.xlabel("Percentage of features selected")
    plt.ylabel("Cross validation Accuracy")
    plt.title('Feature Selection Cross Val Accuracy Versus Percentage of Features')
    plt.plot(percent_list, results)
    plt.plot(optimal_percentile, results[optimal_ndx], 'x', c='k')
    plt.show()

    return optimal_percentile, optimal_num_features, chosen_features, chosen_weights, chosen_pvals
```

The optimize_features function below narrows down the list of features used in the final model by removing all features with p values over 0.05 (95% confidence level). P values over 0.05 correspond to features that fail the test of significance to the model. By removing features greater than 0.05, the number of features is further reduced and will help with preventing overfitting. The features are removed one by one, with the feature with the highest p value removed each time. Each time a feature is removed, the cross validation is performed and new p values are computed. This is repeated until none of the remaining features have p values over 0.05.

In [66]:
```
def optimize_features(train_arr, target_arr, model, all_attrs, chosen_features, chosen_weights, chosen_pvals, cv_num):

    print 'Feature Optimization (Removal of All Attributes with P Values >= 0.05)'
    print '-------------------------------------------------------------------------'

    train_all_df = pd.DataFrame(train_arr, columns = all_attrs)

    # create a dataframe with 1 row containing the weights from feature selection and create dataframe with selected fea
tures
    fs_features = all_attrs[chosen_features]
    train_fs_df = train_all_df[fs_features]

    # extract weights and p values corresponding to selected features
    fs_weights = chosen_weights[chosen_features]
    fs_p_vals = chosen_pvals[chosen_features]

    # create a dataframe with 1 row containing the weights from feature selection
    fs_w_arr = np.zeros((1, len(fs_weights)))
    fs_w_arr[0] = fs_weights
    fs_w_df = pd.DataFrame(fs_w_arr, columns = fs_features)

    # create a dataframe with 1 row containing the p values from feature selection
    fs_p_arr = np.zeros((1, len(fs_p_vals)))
    fs_p_arr[0] = fs_p_vals
    fs_p_df = pd.DataFrame(fs_p_arr, columns = fs_features)

    result = 0
    feature_select = []
    feature_scores = []
    features_pvals = []

    # create model to fit with all current features
    fs = feature_selection.SelectPercentile(feature_selection.chi2, percentile = 100)

    while True:

        # find the index of the maximum p value.  if two have the same value, take the index of the first.
        p_vals = np.array(fs_p_df)
        max_ndx = np.where(p_vals == p_vals.max())[1]
        if len(max_ndx) > 1:
```

```
            max_ndx = max_ndx[0]
        max_p = p_vals[0, int(max_ndx)]

        # if the max p value is less than 0.05, create the list of new features and exit the loop
        if max_p < 0.05:
            new_fs_features = train_fs_df.columns.values.tolist()
            break

        # Remove attribute with max p value over 0.05
        col_names = train_fs_df.columns.values.tolist()
        max_attr = col_names[int(max_ndx)]

        train_fs_df = train_fs_df.drop(max_attr, 1)

        # create new array of training data and list of new features
        train_fs_np = np.array(train_fs_df)
        new_fs_features = train_fs_df.columns.values.tolist()

        # fit the model with the new training data (without the max feature)
        train_arr_fs = fs.fit_transform(train_fs_np, target_arr)

        # get the accuracy, weights, p values, and mean accuracy of the new model
        scores = cross_validation.cross_val_score(model, train_arr_fs, target_arr, cv = cv_num, scoring = "accuracy")
        result = scores.mean()
        feature_select = fs.get_support()
        feature_scores = fs.scores_
        features_pvals = fs.pvalues_

        # populate a dataframe with the new feature weights
        fs_w_arr = np.zeros((1, len(feature_scores)))
        fs_w_arr[0] = feature_scores
        fs_w_df = pd.DataFrame(fs_w_arr, columns = new_fs_features)

        # populate a dataframe with the new feature p values
        fs_p_arr = np.zeros((1, len(features_pvals)))
        fs_p_arr[0] = features_pvals
        fs_p_df = pd.DataFrame(fs_p_arr, columns = new_fs_features)

    # populate a dataframe with the p values and weights of all of the variables
    all_info_df = pd.DataFrame(all_attrs)
    all_info_df["weights"] = chosen_weights
    all_info_df["p_values"] = chosen_pvals

    # populate a dataframe with the p values and weights of all the newly reduced features
    fs_info_df = pd.DataFrame(new_fs_features)
    fs_info_df["weights"] = np.array(fs_w_df)[0]
    fs_info_df["p_values"] = np.array(fs_p_df)[0]
```

```
    # print the results of the optimization
    print 'Final Number of Training Set Attributes:', train_fs_df.shape[1]
    print 'Final Training Set Features Selection Accuracy:', result
    print 'Final Training Set Attributes:'
    print new_fs_features
    print '\n'

    return train_fs_df, all_info_df, fs_info_df
```

The meausre_performance function below prints the model accuracy, the classificaiton report, and the confusion matrix of a model passed in.

In [67]:
```
def measure_performance(x_arr, y_arr, model, show_accuracy=True, show_classification_report=True, show_confussion_matrix=Tr
ue):
    y_pred_arr = model.predict(x_arr)
    if show_accuracy:
        print "Accuracy:"
        print metrics.accuracy_score(y_arr, y_pred_arr), "\n"

    if show_classification_report:
        print "Classification Report:"
        print metrics.classification_report(y_arr, y_pred_arr), "\n"

    if show_confussion_matrix:
        print "Confussion Matrix:"
        cm = metrics.confusion_matrix(y_arr, y_pred_arr)
        print cm, "\n"
```

The perform_grid_search function below takes in a dictionary with the parameters and their ranges for usage in grid search model selection and performs the grid search with all of the parameter options.

In [68]:
```
def perform_grid_search(model, data_train, target_train, data_test, target_test, param_dict, cv_num):

    print '---------------------------------- Parameter Grid Search ----------------------------------'

    num_params = len(param_dict.keys())
    gs = GridSearchCV(model, param_dict, verbose=1, cv=cv_num)
    %time _ = gs.fit(data_train, target_train)
    opt_params = gs.best_params_
    opt_score = gs.best_score_
    print '\n'
    print 'Grid Search Optimal Parameters:', opt_params
    print 'Grid Search Optimal Parameter Score:', opt_score
    print '\n'

    # Set the optimal grid search parameters
    for key, value in opt_params.items():
        model.set_params(**{key: value})
    print 'Final Model Parameter Settings:'
    print(model)
    print '\n'

    return model
```

The model_data function below performs both the cross validation modelling and the final modelling with the full training set and the testing set. The cross validation modelling is what is used to refine the model parameter ranges to be used. The testing set is used only as an evaluation dataset to judge the final model performance.

In [69]:
```
def model_data(model, data_train, target_train, data_test, target_test, cv_num):

    # Get the performance of the training and testing data sets
    print '---------------------------------- Training Data Performance ----------------------------------'
    print '\n'
    print 'Final Model Training Set Cross Validation Results'
    print '-------------------------------------------------'

    model_cv_data(model, data_train, target_train, cv_num)
    print '\n'

    # Fit the training data to the optimal model, using the features from feature selection
    model.fit(data_train, target_train)

    print 'Final Model Full Training Set Results'
    print '-------------------------------------'
    measure_performance(data_train, target_train, model, show_accuracy=True, show_confussion_matrix=True,
                        show_classification_report=True)

    print '\n'
    print '---------------------------------- Testing Data Performance ----------------------------------'
    print '\n'
    print 'Final Model Testing Set Results'
    print '-------------------------------'

    measure_performance(data_test, target_test, model, show_accuracy=True, show_confussion_matrix=True,
                        show_classification_report=True)

    return model
```

The model_cv_data function below performs cross validation on a training and target dataset passed in. The mean accuracy of the training and testing datasets are printed at the end.

In [70]:
```
def model_cv_data(model, data_train, target_train, cv_num):

    # generate the folds for use in each iteration of the cross validation
    kf = KFold(len(data_train), n_folds = cv_num, shuffle=True, random_state=0)
    cross_acc_train = 0
    cross_acc_test = 0
    for trainNdx, testNdx in kf:

        # fit a model to the current cross validation training fold and calculate the training accuracy
        model.fit(data_train[trainNdx,:], target_train[trainNdx])
        train_pred = model.predict(data_train[trainNdx,:])
        cross_acc_train += metrics.accuracy_score(target_train[trainNdx], train_pred)

        # calculate the testing accuracy of the current fold
        test_pred = model.predict(data_train[testNdx,:])
        cross_acc_test += metrics.accuracy_score(target_train[testNdx], test_pred)

    acc_cv_train = cross_acc_train/cv_num
    acc_cv_test = cross_acc_test/cv_num

    print cv_num, 'Fold Cross Validation Training Accuracy:'
    print acc_cv_train
    print cv_num, 'Fold Cross Validation Testing Accuracy:'
    print acc_cv_test
```

The create_opt_model function below is a wrapper function that calls the find_percent function, the optimize features function, creates the datasets that are the result of the two feature selection steps, call the grid search function, then calls the model data function.

In [71]:
```
def create_opt_model(model, data_train, target_train, data_test, target_test, percent_list, param_dict, cv_num, attrs):

    # Find the optimal list of features to use
    opt_percent, opt_num, chosen_attr, chosen_w, chosen_p = find_percent(data_train, target_train, model, percent_list, cv_
num)
    print 'Optimal Percent:', opt_percent
    print 'Optimal Number of Features:', opt_num
    print 'Features Chosen:'
    print attrs[chosen_attr]
    print '\n'

    train_fs_df, all_info_df, fs_info_df = optimize_features(data_train, target_train, model, attrs, chosen_attr, chosen_w,
 chosen_p, cv_num)

    # Get the training and testing data with chosen features
    selected_features = train_fs_df.columns.values.tolist()
    train_arr_fs = np.array(train_fs_df)

    test_df = pd.DataFrame(data_test, columns = attrs)
    test_fs_df = test_df[selected_features]
    test_arr_fs = np.array(test_fs_df)

    # Perform grid search with parameter dictionary passed in
    opt_model = perform_grid_search(model, train_arr_fs, target_train, test_arr_fs, target_test, param_dict, cv_num)

    # Perform grid search with parameter dictionary passed in
    final_model = model_data(opt_model, train_arr_fs, target_train, test_arr_fs, target_test, cv_num)

    return final_model, all_info_df, fs_info_df, selected_features
```

Create feature names array used in feature selection

In [72]:
```
spdsht_features = np.array(data_train.columns.values.tolist())
print spdsht_features.shape

spdsht_features_df = pd.DataFrame(spdsht_features)
spdsht_features_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\Dataset Training Features.csv'
)
```

```
(101L,)
```

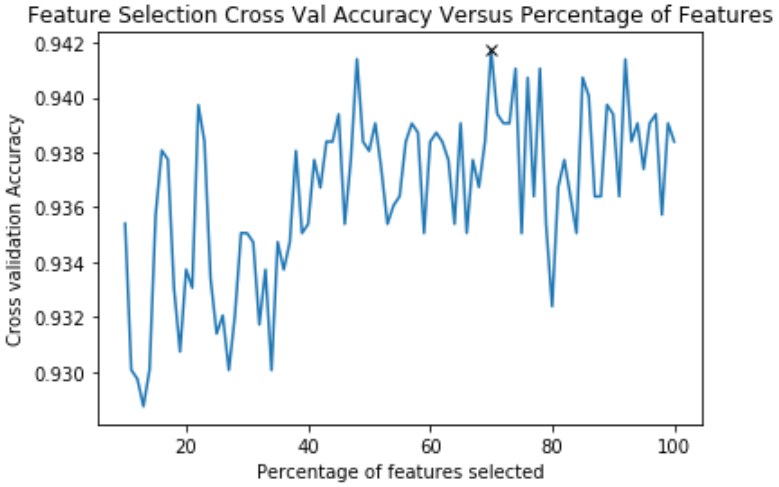
Decision Tree Classification With Feature Selection

Call the create_opt_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [73]:
```
cv = 10
dt = tree.DecisionTreeClassifier()
percentages = a = np.arange(10, 101, 1)

temp_leaf = np.arange(10, 201, 10)
temp_depth = np.arange(1, 11, 1)

parameters = {
    'criterion': ['entropy','gini'],
    'max_depth': temp_depth,
    'min_samples_leaf': temp_leaf,
    'min_samples_split': temp_leaf
}

print 'Percentage List Used:'
print percentages
print '\n'
print 'Parameters Used:'
print parameters
print '\n'

dt_model, dt_all_info_df, dt_fs_info_df, dt_features = create_opt_model(dt, data_train_np, target_train_np, data_test_np, t
arget_test_np, percentages, parameters, cv, spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]


Parameters Used:
{'min_samples_split': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200]), 'criterion': ['entropy', 'gini'], 'max_depth': array([ 1,  2,  3,  4,  5,  6,
7,  8,  9, 10]), 'min_samples_leaf': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200])}


---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 70
Optimal Number of Features: 70
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD090' 'CBD120' 'CBD130' 'DBD895'
 'DBD900' 'DBD905' 'DBD910' 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male'
 'RIDRETH1_Mexican_American' 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Other'
 'RIDRETH1_Other_Hispanic' 'DMDEDUC2_9th_to_12th_No_Grad'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Living_W_Partner'
 'DMDMARTL_Never_Married' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_10000_to_14999' 'INDHHIN2_15000_to_19999'
 'INDHHIN2_20000_to_24999' 'INDHHIN2_35000_to_44999'
 'INDHHIN2_45000_to_54999' 'INDHHIN2_5000_to_9999'
 'INDHHIN2_65000_to_74999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_No' 'SMQ020_Yes' 'HEQ010_Yes' 'HEQ030_Yes' 'HUQ051_0' 'HUQ051_1'
 'HUQ051_16_Or_More' 'HUQ051_4_to_5' 'HUQ051_6_to_7' 'HUQ071_No'
 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes' 'MCQ082_Yes' 'MCQ160N_Yes'
 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160D_Yes' 'MCQ160E_Yes' 'MCQ160F_Yes'
 'MCQ160G_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_Yes' 'MCQ160L_Yes'
 'MCQ160O_No' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 37
Final Training Set Features Selection Accuracy: 0.934392963626
Final Training Set Attributes:
['RIDAGEYR', 'INDFMPIR', 'CBD070', 'CBD090', 'CBD120', 'CBD130', 'DBD900', 'DBD905', 'DBD910', 'PAD680', 'RIAGENDR_Female',
'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_Hispanic_Black', 'RIDRETH1_Other', 'RIDRETH1_Other_Hispanic',
'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'INDHHIN2_15000_to_19999', 'INDHHIN2_5000_to_9999', 'BP
Q020_No', 'BPQ020_Yes', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Yes',
'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ071_Yes', 'MCQ010_No', 'MCQ010_Yes', 'MCQ160B_Yes', 'MCQ160M_Yes', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 8000 candidates, totalling 80000 fits

[Parallel(n_jobs=1)]: Done 80000 out of 80000 | elapsed:  7.8min finished
```

```
Wall time: 7min 48s


Grid Search Optimal Parameters: {'min_samples_split': 40, 'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 10}
Grid Search Optimal Parameter Score: 0.968697968698


Final Model Parameter Settings:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=10, min_samples_split=40,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.967698951709
10 Fold Cross Validation Testing Accuracy:
0.967027685493


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.968031968032

Classification Report:
             precision    recall  f1-score   support

          0       0.97      1.00      0.98      2904
          1       0.59      0.10      0.17        99

avg / total       0.96      0.97      0.96      3003


Confussion Matrix:
[[2897    7]
 [  89   10]]



---------------------------------- Testing Data Performance ----------------------------------
```

```
Final Model Testing Set Results
-------------------------------
Accuracy:
0.953395472703

Classification Report:
             precision    recall  f1-score   support

          0       0.96      0.99      0.98       723
          1       0.00      0.00      0.00        28

avg / total       0.93      0.95      0.94       751


Confussion Matrix:
[[716   7]
 [ 28   0]]
```

Print out the number of nodes in the tree.

In [74]:
```
treeObj = dt_model.tree_
print 'Number of nodes in the tree:'
print treeObj.node_count
```

```
Number of nodes in the tree:
23
```

Save csv files with the feature weights and p values of the feature selection features and all of the original features.

In [75]:
```
dt_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Chosen Features.csv')
```

In [76]:
```
dt_all_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT All Features.csv')
```

Compute the feature importances of the model and save them into a csv file.

In [77]:
```
feature_imp = dt_model.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = dt_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Feature Selection Importances.csv')
feature_imp_df
```

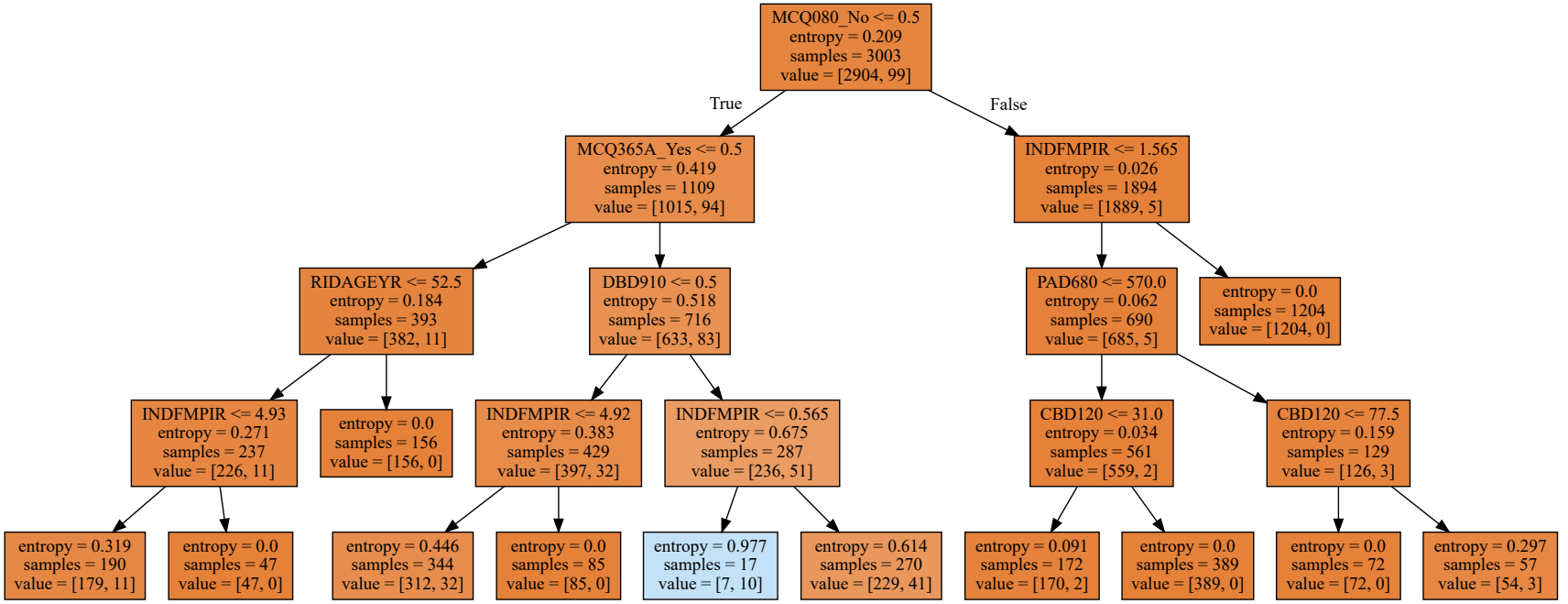
Out[77]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIAGENDR_Male
0	0.041209	0.165114	0.0	0.0	0.035398	0.0	0.0	0.0	0.063412	0.015271	0.0	0.0

Create a visualization of the model tree.

In [78]:
```
export_graphviz(dt_model, out_file='tree.dot', feature_names=dt_features, filled=True)

with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[78]:



Save the tree visualization to a jpeg file.

In [79]:
```
system(dot -Tpng tree.dot -o dtree.jpeg)
```

Out[79]: []

Decision Tree Classification With All Features

Call the grid search and model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [80]:
```
cv = 10
dt_all = tree.DecisionTreeClassifier()

temp_leaf = np.arange(10, 201, 10)
temp_depth = np.arange(1, 11, 1)

parameters = {
    'criterion': ['entropy','gini'],
    'max_depth': temp_depth,
    'min_samples_leaf': temp_leaf,
    'min_samples_split': temp_leaf
}

print 'Parameters Used:'
print parameters
print '\n'

# Perform grid search with parameter dictionary passed in
opt_model = perform_grid_search(dt_all, data_train_np, target_train_np, data_test_np, target_test_np, parameters, cv)

# Perform grid search with parameter dictionary passed in
dt_model_all = model_data(opt_model, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
Parameters Used:
{'min_samples_split': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200]), 'criterion': ['entropy', 'gini'], 'max_depth': array([ 1,  2,  3,  4,  5,  6,
7,  8,  9, 10]), 'min_samples_leaf': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200])}


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 8000 candidates, totalling 80000 fits

[Parallel(n_jobs=1)]: Done 80000 out of 80000 | elapsed: 16.7min finished
```

```
Wall time: 16min 42s


Grid Search Optimal Parameters: {'min_samples_split': 40, 'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 10}
Grid Search Optimal Parameter Score: 0.968697968698


Final Model Parameter Settings:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=10, min_samples_split=40,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.967698951709
10 Fold Cross Validation Testing Accuracy:
0.967027685493


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.968031968032

Classification Report:
             precision    recall  f1-score   support

          0       0.97      1.00      0.98      2904
          1       0.59      0.10      0.17        99

avg / total       0.96      0.97      0.96      3003


Confussion Matrix:
[[2897    7]
 [  89   10]]



---------------------------------- Testing Data Performance ----------------------------------
```

```
Final Model Testing Set Results
-------------------------------
Accuracy:
0.953395472703

Classification Report:
             precision    recall  f1-score   support

          0       0.96      0.99      0.98       723
          1       0.00      0.00      0.00        28

avg / total       0.93      0.95      0.94       751


Confussion Matrix:
[[716   7]
 [ 28   0]]
```

Print out the number of nodes in the tree.

In [81]:
```
treeObjAll = dt_model_all.tree_
print 'Number of nodes in the tree:'
print treeObjAll.node_count
```

```
Number of nodes in the tree:
23
```

Compute feature importances and save them to a csv file.

In [82]:
```
feature_imp = dt_model_all.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = spdsht_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Full Training Importances.csv')
feature_imp_df
```

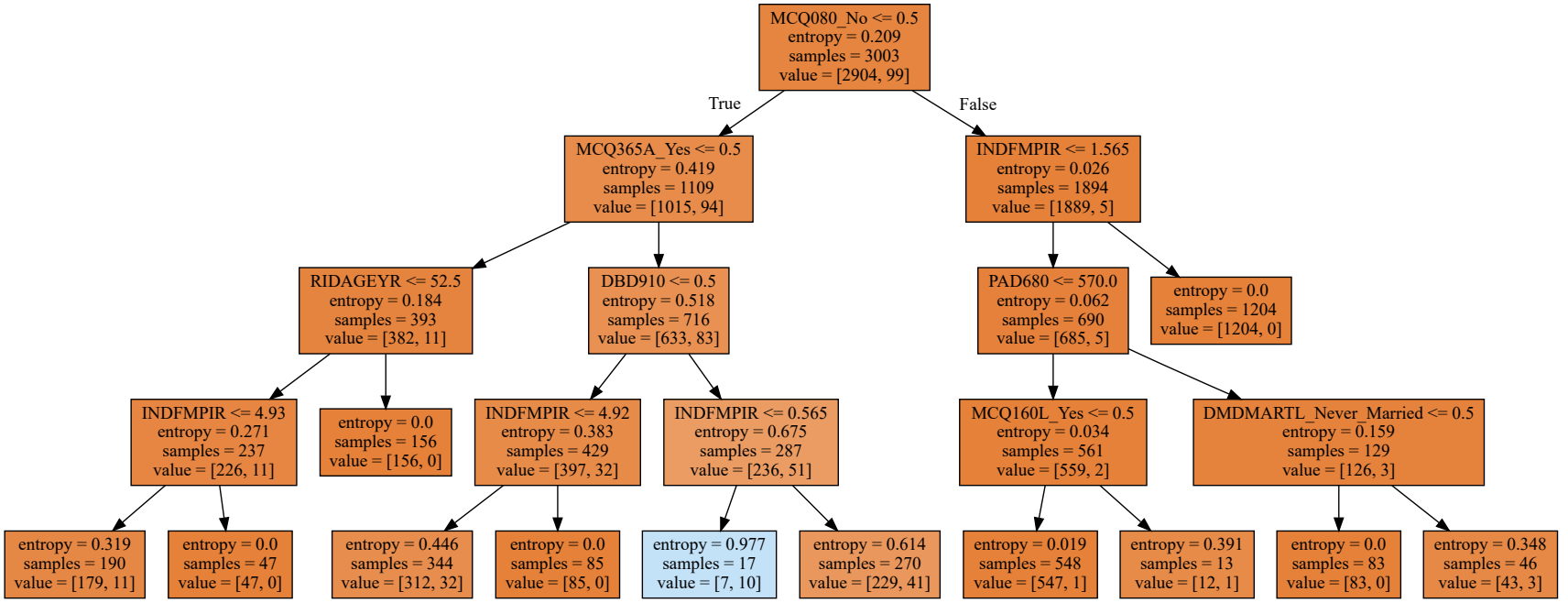
Out[82]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIAGENI
0	0.040991	0.164244	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.063078	0.01519	0.0	0.0

Create the model tree visualization.

In [83]:
```
export_graphviz(dt_model_all, out_file='tree_all.dot', feature_names=spdsht_features, filled=True)

with open("tree_all.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[83]:



Save the model tree to a jpeg file.

In [84]:
```
system(dot -Tpng tree_all.dot -o dtree_all.jpeg)
```

Out[84]: []

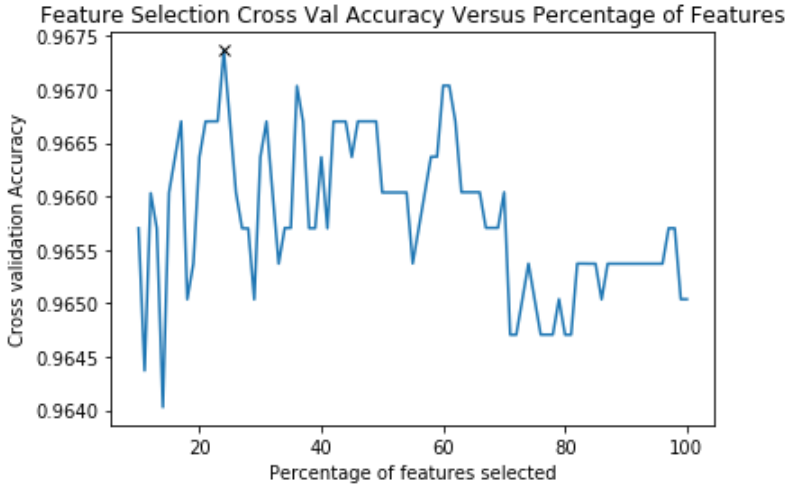
K Nearest Neighbor Classification With Feature Selection

Call the create_opt_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [85]:
```
cv = 10
knn = neighbors.KNeighborsClassifier(p = 2)
percentages = a = np.arange(10, 101, 1)

temp_k = np.arange(5, 201, 5)

parameters = {
    'weights': ['uniform','distance'],
    'n_neighbors': temp_k
}

print 'Percentage List Used:'
print percentages
print '\n'
print 'Parameters Used:'
print parameters
print '\n'

knn_model, knn_all_info_df, knn_fs_info_df, knn_features = create_opt_model(knn, data_train_norm_np, target_train_np, data_
test_norm_np, target_test_np, percentages, parameters, cv, spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]


Parameters Used:
{'n_neighbors': array([  5,  10,  15,  20,  25,  30,  35,  40,  45,  50,  55,  60,  65,
        70,  75,  80,  85,  90,  95, 100, 105, 110, 115, 120, 125, 130,
       135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200]), 'weights': ['uniform', 'distance']}


---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 24
Optimal Number of Features: 24
Features Chosen:
['RIAGENDR_Female' 'RIAGENDR_Male' 'RIDRETH1_Non_Hispanic_Black'
 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic'
 'DMDEDUC2_College_Grad_And_Above' 'INDHHIN2_15000_to_19999'
 'INDHHIN2_5000_to_9999' 'BPQ020_No' 'BPQ020_Yes' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'HUQ051_1' 'HUQ051_16_Or_More' 'HUQ071_Yes' 'MCQ010_Yes' 'MCQ160B_Yes'
 'MCQ160M_Yes' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
------------------------------------------------------------------------
Final Number of Training Set Attributes: 24
Final Training Set Features Selection Accuracy: 0
Final Training Set Attributes:
['RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Non_Hispanic_Black', 'RIDRETH1_Other', 'RIDRETH1_Other_Hispanic', 'DMDEDUC2_
College_Grad_And_Above', 'INDHHIN2_15000_to_19999', 'INDHHIN2_5000_to_9999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010_Yes', 'MCQ0
80_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Yes', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ071_Ye
s', 'MCQ010_Yes', 'MCQ160B_Yes', 'MCQ160M_Yes', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 80 candidates, totalling 800 fits

[Parallel(n_jobs=1)]: Done 800 out of 800 | elapsed:   33.0s finished
```

```
Wall time: 33.1 s


Grid Search Optimal Parameters: {'n_neighbors': 5, 'weights': 'uniform'}
Grid Search Optimal Parameter Score: 0.967365967366


Final Model Parameter Settings:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.968142957643
10 Fold Cross Validation Testing Accuracy:
0.966698781838


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.967365967366

Classification Report:
             precision    recall  f1-score   support

          0       0.97      1.00      0.98      2904
          1       0.57      0.04      0.08        99

avg / total       0.96      0.97      0.95      3003


Confussion Matrix:
[[2901    3]
 [  95    4]]



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
```

```
------------------------------
Accuracy:
0.964047936085

Classification Report:
             precision    recall  f1-score   support

          0       0.96      1.00      0.98       723
          1       1.00      0.04      0.07        28

avg / total       0.97      0.96      0.95       751


Confussion Matrix:
[[723   0]
 [ 27   1]]
```

Save csv file with the weights and p values of the feature selection features and of all of the original features.

In [86]:
```
knn_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\KNN Chosen Features.csv')
```

In [87]:
```
knn_all_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\KNN All Features.csv')
```

K Nearest Neighbor Classification With All Features

Call the grid search and model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [88]:
```
cv = 10
knn_all = neighbors.KNeighborsClassifier(p = 2)

temp_k = np.arange(5, 201, 5)
#temp_k = np.arange(20, 101, 20)

parameters = {
    'weights': ['uniform','distance'],
    'n_neighbors': temp_k
}

print 'Parameters Used:'
print parameters
print '\n'

# Perform grid search with parameter dictionary passed in
opt_model = perform_grid_search(knn_all, data_train_norm_np, target_train_np, data_test_norm_np, target_test_np, parameters
, cv)

# Perform grid search with parameter dictionary passed in
knn_model_all = model_data(opt_model, data_train_norm_np, target_train_np, data_test_norm_np, target_test_np, cv)
```

```
Parameters Used:
{'n_neighbors': array([  5,  10,  15,  20,  25,  30,  35,  40,  45,  50,  55,  60,  65,
        70,  75,  80,  85,  90,  95, 100, 105, 110, 115, 120, 125, 130,
       135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200]), 'weights': ['uniform', 'distance']}


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 80 candidates, totalling 800 fits

[Parallel(n_jobs=1)]: Done 800 out of 800 | elapsed:  2.0min finished
```

```
Wall time: 1min 59s


Grid Search Optimal Parameters: {'n_neighbors': 10, 'weights': 'uniform'}
Grid Search Optimal Parameter Score: 0.967032967033


Final Model Parameter Settings:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=10, p=2,
           weights='uniform')


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.967032929117
10 Fold Cross Validation Testing Accuracy:
0.967029900332


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.967032967033

Classification Report:
             precision    recall  f1-score   support

          0       0.97      1.00      0.98      2904
          1       0.00      0.00      0.00        99

avg / total       0.94      0.97      0.95      3003


Confussion Matrix:
[[2904    0]
 [  99    0]]



---------------------------------- Testing Data Performance ----------------------------------
```

```
Final Model Testing Set Results
-------------------------------

C:\Users\Kari\Anaconda3\envs\Python27\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)

Accuracy:
0.962716378162

Classification Report:
             precision    recall  f1-score   support

          0       0.96      1.00      0.98       723
          1       0.00      0.00      0.00        28

avg / total       0.93      0.96      0.94       751


Confussion Matrix:
[[723   0]
 [ 28   0]]
```

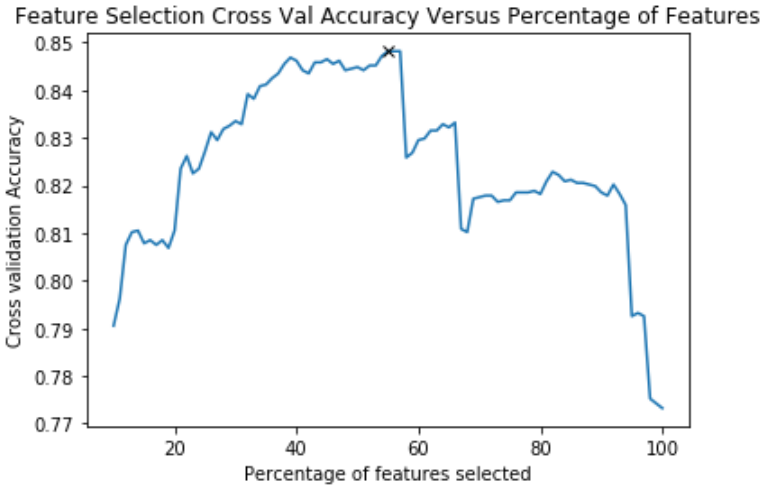
Naive Bayes Gaussian Classification With Feature Selection

Call the find percent and optimize functions to perform feature selection on the training set. Then call the model data functions with the reduced feature training set. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [89]:
```
cv = 10
nbg = naive_bayes.GaussianNB()
percentages = a = np.arange(10, 101, 1)

opt_percent, opt_num, attr_ndx, attr_w, attr_p = find_percent(data_train_np, target_train_np, nbg, percentages, cv)
print 'Optimal Percent:', opt_percent
print 'Optimal Number of Features:', opt_num
print 'Features Chosen:'
print spdsht_features[attr_ndx]
print '\n'

train_fs_df, nbg_all_info_df, nbg_fs_info_df = optimize_features(data_train_np, target_train_np, nbg, spdsht_features, attr
_ndx, attr_w, attr_p, cv)

# Get the training and testing data with chosen features
selected_features = train_fs_df.columns.values.tolist()
train_arr_fs = np.array(train_fs_df)

test_df = pd.DataFrame(data_test_np, columns = spdsht_features)
test_fs_df = test_df[selected_features]
test_arr_fs = np.array(test_fs_df)

nbg_model = model_data(nbg, train_arr_fs, target_train_np, test_arr_fs, target_test_np, cv)
```

---------- Feature Selection ----------



```
Optimal Percent: 55
Optimal Number of Features: 55
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD090' 'CBD120' 'CBD130' 'DBD900'
 'DBD905' 'DBD910' 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male'
 'RIDRETH1_Mexican_American' 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Other'
 'RIDRETH1_Other_Hispanic' 'DMDEDUC2_9th_to_12th_No_Grad'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDMARTL_Living_W_Partner' 'DMDMARTL_Never_Married'
 'INDHHIN2_100000_And_Over' 'INDHHIN2_15000_to_19999'
 'INDHHIN2_35000_to_44999' 'INDHHIN2_45000_to_54999'
 'INDHHIN2_5000_to_9999' 'BPQ020_No' 'BPQ020_Yes' 'DIQ010_Borderline'
 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No' 'MCQ080_Yes' 'MCQ365A_No'
 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes' 'SMQ020_No' 'SMQ020_Yes'
 'HEQ030_Yes' 'HUQ051_0' 'HUQ051_1' 'HUQ051_16_Or_More' 'HUQ051_6_to_7'
 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes' 'MCQ160B_Yes' 'MCQ160D_Yes'
 'MCQ160E_Yes' 'MCQ160G_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_Yes'
 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 37
Final Training Set Features Selection Accuracy: 0.843492431397
Final Training Set Attributes:
['RIDAGEYR', 'INDFMPIR', 'CBD070', 'CBD090', 'CBD120', 'CBD130', 'DBD900', 'DBD905', 'DBD910', 'PAD680', 'RIAGENDR_Female',
'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_Hispanic_Black', 'RIDRETH1_Other', 'RIDRETH1_Other_Hispanic',
'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'INDHHIN2_15000_to_19999', 'INDHHIN2_5000_to_9999', 'BP
Q020_No', 'BPQ020_Yes', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Yes',
'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ071_Yes', 'MCQ010_No', 'MCQ010_Yes', 'MCQ160B_Yes', 'MCQ160M_Yes', 'MCQ160O_Yes']


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.844821733562
10 Fold Cross Validation Testing Accuracy:
0.840823920266


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.845820845821
```

```
Classification Report:
             precision    recall  f1-score   support

          0       0.99      0.85      0.91      2904
          1       0.14      0.74      0.24        99

avg / total       0.96      0.85      0.89      3003


Confussion Matrix:
[[2467  437]
 [  26   73]]



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.834886817577

Classification Report:
             precision    recall  f1-score   support

          0       0.99      0.84      0.91       723
          1       0.14      0.68      0.23        28

avg / total       0.95      0.83      0.88       751


Confussion Matrix:
[[608 115]
 [  9  19]]
```

Save csv files with weights and p values with feature selection features and with all of the original features.

In [90]:
```
nbg_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NBG ChosenFeatures.csv')
```

In [91]:
```
nbg_all_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NBG All Features.csv')
```

Naive Bayes Gaussian Classification With All Features

Call the model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

In [92]:
```
cv = 10
nbg_all = naive_bayes.GaussianNB()

nbg_model = model_data(nbg_all, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.782550065681
10 Fold Cross Validation Testing Accuracy:
0.773905869324


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.78687978688

Classification Report:
             precision    recall  f1-score   support

          0       0.99      0.79      0.88      2904
          1       0.12      0.84      0.21        99

avg / total       0.96      0.79      0.85      3003


Confussion Matrix:
[[2280  624]
 [  16   83]]



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.784287616511

Classification Report:
             precision    recall  f1-score   support

          0       0.98      0.79      0.88       723
          1       0.11      0.68      0.19        28

avg / total       0.95      0.78      0.85       751
```

```
Confussion Matrix:
[[570 153]
 [  9  19]]
```

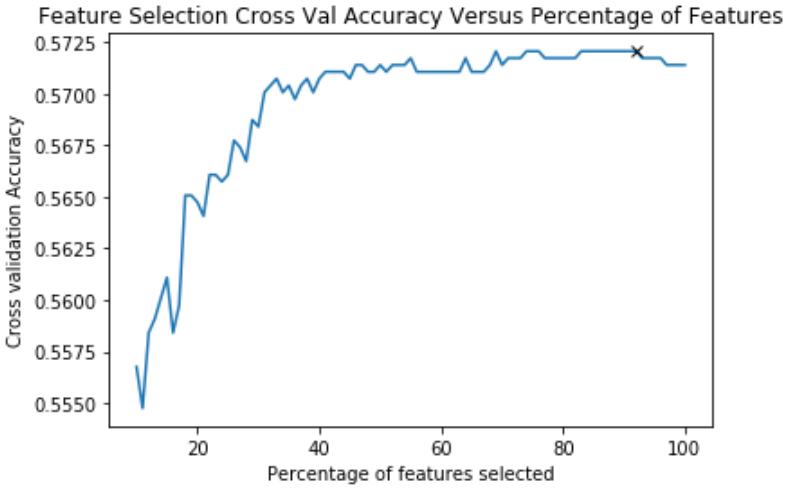
Naive Bayes Multinomial Classification With Feature Selection

Call the find percent and optimize functions to perform feature selection on the training set. Then call the model data functions with the reduced feature training set. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [93]:
```
cv = 10
nbm = naive_bayes.MultinomialNB()
percentages = a = np.arange(10, 101, 1)

opt_percent, opt_num, attr_ndx, attr_w, attr_p = find_percent(data_train_np, target_train_np, nbm, percentages, cv)
print 'Optimal Percent:', opt_percent
print 'Optimal Number of Features:', opt_num
print 'Features Chosen:'
print spdsht_features[attr_ndx]
print '\n'

train_fs_df, nbm_all_info_df, nbm_fs_info_df = optimize_features(data_train_np, target_train_np, nbm, spdsht_features, attr
_ndx, attr_w, attr_p, cv)

# Get the training and testing data with chosen features
selected_features = train_fs_df.columns.values.tolist()
train_arr_fs = np.array(train_fs_df)

test_df = pd.DataFrame(data_test_np, columns = spdsht_features)
test_fs_df = test_df[selected_features]
test_arr_fs = np.array(test_fs_df)


nbm_model = model_data(nbm, train_arr_fs, target_train_np, test_arr_fs, target_test_np, cv)
```

---------- Feature Selection ----------



```
Optimal Percent: 92
Optimal Number of Features: 92
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD090' 'CBD120' 'CBD130' 'DBD895'
 'DBD900' 'DBD905' 'DBD910' 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male'
 'RIDRETH1_Mexican_American' 'RIDRETH1_Non_Hispanic_Black'
 'RIDRETH1_Non_Hispanic_White' 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic'
 'DMDEDUC2_9th_to_12th_No_Grad' 'DMDEDUC2_College_Grad_And_Above'
 'DMDEDUC2_High_School_Grad_GED' 'DMDEDUC2_Less_Than_9th'
 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Divorced' 'DMDMARTL_Living_W_Partner'
 'DMDMARTL_Married' 'DMDMARTL_Never_Married' 'DMDMARTL_Separated'
 'DMDMARTL_Widowed' 'INDHHIN2_100000_And_Over' 'INDHHIN2_10000_to_14999'
 'INDHHIN2_15000_to_19999' 'INDHHIN2_20000_to_24999'
 'INDHHIN2_25000_to_34999' 'INDHHIN2_35000_to_44999'
 'INDHHIN2_45000_to_54999' 'INDHHIN2_5000_to_9999'
 'INDHHIN2_65000_to_74999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_No' 'SMQ020_Yes' 'HEQ010_Yes' 'HEQ030_No' 'HEQ030_Yes' 'HUQ051_0'
 'HUQ051_1' 'HUQ051_10_to_12' 'HUQ051_16_Or_More' 'HUQ051_2_to_3'
 'HUQ051_4_to_5' 'HUQ051_6_to_7' 'HUQ051_8_to_9' 'HUQ071_No' 'HUQ071_Yes'
 'MCQ010_No' 'MCQ010_Yes' 'MCQ082_Yes' 'MCQ160N_No' 'MCQ160N_Yes'
 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160C_Yes' 'MCQ160D_No' 'MCQ160D_Yes'
 'MCQ160E_No' 'MCQ160E_Yes' 'MCQ160F_No' 'MCQ160F_Yes' 'MCQ160G_No'
 'MCQ160G_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_No' 'MCQ160K_Yes'
 'MCQ160L_No' 'MCQ160L_Yes' 'MCQ160O_No' 'MCQ160O_Yes' 'MCQ203_Yes'
 'MCQ220_No' 'MCQ220_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 37
Final Training Set Features Selection Accuracy: 0.570393204369
Final Training Set Attributes:
['RIDAGEYR', 'INDFMPIR', 'CBD070', 'CBD090', 'CBD120', 'CBD130', 'DBD900', 'DBD905', 'DBD910', 'PAD680', 'RIAGENDR_Female',
'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_Hispanic_Black', 'RIDRETH1_Other', 'RIDRETH1_Other_Hispanic',
'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'INDHHIN2_15000_to_19999', 'INDHHIN2_5000_to_9999', 'BP
Q020_No', 'BPQ020_Yes', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Yes',
'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ071_Yes', 'MCQ010_No', 'MCQ010_Yes', 'MCQ160B_Yes', 'MCQ160M_Yes', 'MCQ160O_Yes']


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.575424036073
```

```
10 Fold Cross Validation Testing Accuracy:
0.571088593577


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.57708957709

Classification Report:
             precision    recall  f1-score   support

          0       0.98      0.58      0.72      2904
          1       0.05      0.63      0.09        99

avg / total       0.95      0.58      0.70      3003


Confussion Matrix:
[[1671 1233]
 [  37   62]]



--------------------------------- Testing Data Performance ---------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.587217043941

Classification Report:
             precision    recall  f1-score   support

          0       0.97      0.59      0.73       723
          1       0.05      0.54      0.09        28

avg / total       0.94      0.59      0.71       751


Confussion Matrix:
[[426 297]
 [ 13  15]]
```

Save csv files with the weights and p values of the feature selection features and of all the original features.

In [94]:
```
nbm_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NBM Chosen Features.csv')
```

In [95]:
```
nbm_all_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NBM All Features.csv')
```

Naive Bayes Multinomial Classification With All Features

Call the model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

In [96]:
```
cv = 10
nbm_all = naive_bayes.MultinomialNB()

nbm_model = model_data(nbm_all, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.577348070913
10 Fold Cross Validation Testing Accuracy:
0.574085271318


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.577755577756

Classification Report:
             precision    recall  f1-score   support

          0       0.98      0.58      0.73      2904
          1       0.05      0.63      0.09        99

avg / total       0.95      0.58      0.70      3003


Confussion Matrix:
[[1673 1231]
 [  37   62]]



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.59121171771

Classification Report:
             precision    recall  f1-score   support

          0       0.97      0.59      0.74       723
          1       0.05      0.54      0.09        28

avg / total       0.94      0.59      0.71       751
```

```
Confussion Matrix:
[[429 294]
 [ 13  15]]
```

Linear Discriminant Analysis Classification With All Features

Call the model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

In [97]:
```
cv = 10
lda_all = LinearDiscriminantAnalysis()

lda_model = model_data(lda_all, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------

C:\Users\Kari\Anaconda3\envs\Python27\lib\site-packages\sklearn\discriminant_analysis.py:388: UserWarning: Variables are co
llinear.
  warnings.warn("Variables are collinear.")
```

```
10 Fold Cross Validation Training Accuracy:
0.965626905763
10 Fold Cross Validation Testing Accuracy:
0.96003654485


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.965034965035

Classification Report:
             precision    recall  f1-score   support

          0       0.97      0.99      0.98      2904
          1       0.43      0.18      0.26        99

avg / total       0.95      0.97      0.96      3003


Confussion Matrix:
[[2880   24]
 [  81   18]]



---------------------------------- Testing Data Performance ---------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.956058588549

Classification Report:
             precision    recall  f1-score   support

          0       0.97      0.99      0.98       723
          1       0.27      0.11      0.15        28

avg / total       0.94      0.96      0.95       751


Confussion Matrix:
[[715   8]
 [ 25   3]]
```

---------- K Means Clustering ----------

Create Function Needed For K Means Clustering

The find_opt_k function below performs clustering with several different values of k and computes the sum square error of the clustering for each k value. The optimal k value is chosen to be the knee of the sum square error versus k plot (where the sum square error values stop decreasing rapidly and start to decrease slowly). The knee is found by finding the slopes between each sum square error point and determining when the slope begins to decrease. A tolerance is used to determine when the slope values are levelling off. The tolerance value chosen is determined by trying multiple values until the correct one has been found.

In [98]:
```
def find_opt_k(data_train, k_vals, tol):

    all_sse_dist = []
    for i in k_vals:

        # perform k means clustering with i number of clusters
        kmeans = KMeans(n_clusters = i, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
        kmeans.fit(data_train)
        clusters = kmeans.predict(data_train)
        centers = kmeans.cluster_centers_

        # calculate the sum square error for the current cluster
        sum_sse_dist = calc_cluster_error(centers, clusters, data_train)
        all_sse_dist.append(sum_sse_dist)

    all_sse_dist = np.array(all_sse_dist)
    # find the elementwise difference in the all_sse_dist array (is the slope of the sse points)
    temp_diff = np.diff(all_sse_dist)
    # multiply this diff slope array by -1 to make all neg slopes pos and vis versa (for use with tolerance)
    #find the indices where the diff slope is greater than the tolerance
    temp_ndx = np.where(temp_diff > tol)[0]
    # increment the first index found by 1 to set it to the corresponding point in the values array
    opt_ndx = temp_ndx[0] + 1
    # extract the optimal param value using the index above
    optimal_k = k[opt_ndx]

    print 'Optimal K Value:', optimal_k

    # plot the sum square errors versus k
    plt.figure()
    plt.xlabel("K Values")
    plt.ylabel("Cluster SSE")
    plt.title('Cluster SSE Versus K Values')
    plt.plot(k, all_sse_dist)
    plt.plot(optimal_k, all_sse_dist[opt_ndx], 'x', c='k')
    plt.show()

    # plot the slope between the sum square error points used to determine the knee point
    plt.figure()
    plt.xlabel("K Values")
    plt.ylabel("Cluster SSE Diff")
    plt.title('Cluster SSE Diff Versus K Values')
    plt.plot(k[1:len(k)], temp_diff)
    plt.plot(optimal_k, temp_diff[opt_ndx-1], 'x', c='k')
    plt.show()

    return optimal_k
```

The calc_cluster_error function below computes the total sum square error for all of the clusters based on the cluster centers and the training data instances. The sum square error is defined as the sum of the squares of the differences between the training instances and the cluster center of the cluster they are assigned to.

In [99]:
```
def calc_cluster_error(centers, clusters, data_train):

    uniq_clusters = np.array(np.unique(clusters))

    sse_dist_cent = []
    for j in uniq_clusters:

        # get the cluster center and the training data that has been assigned to the current cluster center
        cluster_center = centers[j]
        cluster_ndx = clusters == j
        cluster_pts = data_train[cluster_ndx]

        # create an array the size of the cluster training data with the cluster center (for used in subtraction)
        center_arr = np.ones((cluster_pts.shape[0], cluster_pts.shape[1])) * cluster_center

        # calculate the sum square error of the training instances to the cluster center
        dist_to_center = cluster_pts - center_arr
        sq_dist_center = np.square(dist_to_center)
        sum_sq_dist = sq_dist_center.sum(axis=1)
        sum_sse_dist = sum_sq_dist.sum(axis=0)
        sse_dist_cent.append(sum_sse_dist)

    # sum the sum square error from all the clusters
    sse_dist_cent = np.array(sse_dist_cent)
    sum_sse_dist = sse_dist_cent.sum()

    return sum_sse_dist
```

Create full dataset numpy array

In [100]:
```
data_train_all = np.array(train_df_spdsht)
target_all = np.array(target_df)
```

Create normalized full training data

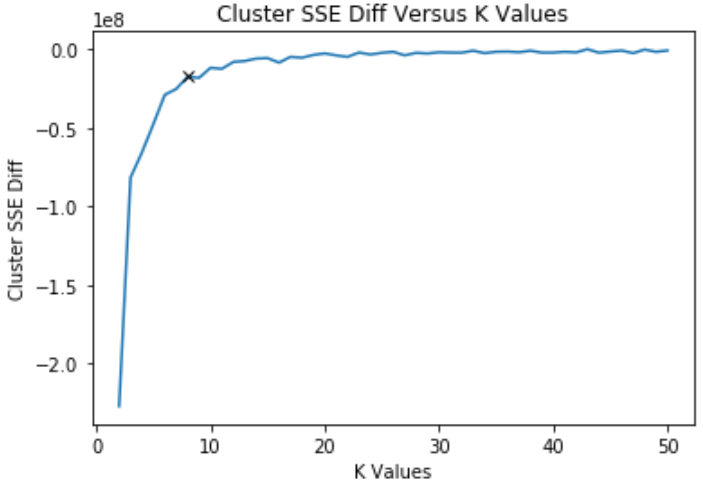
In [101]:
```
min_max_scaler = preprocessing.MinMaxScaler().fit(train_df_spdsht)
data_all_norm_np = min_max_scaler.transform(train_df_spdsht)
```

K Means Clustering With Full Training Set

Call the find_opt_k function to find the optimal k value for the clustering of the full original (non-normalized) training data (before train/test split).

In [102]:
```
k = np.arange(1, 51, 1)
tol = -0.2 * (10**8)

optimal_k = find_opt_k(data_train_all, k, tol)
```

```
Optimal K Value: 8
```



Perform k means clustering with the optimal k values calculated above.

In [103]:
```
kmeans = KMeans(n_clusters = optimal_k, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_train_all)
clusters = kmeans.predict(data_train_all)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the final clustering with the optimal k value.

In [104]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_train_all)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
222817306.81
```

Create names to be used for each row of the cluster center dataframe created below.

In [105]:
```
cluster_names = []
uniq_cluster = np.unique(clusters)
for i in uniq_cluster:
    temp_str = 'Cluster_' + str(i)
    cluster_names.append(temp_str)
```

Create a dataframe with the cluster center values.

In [106]:
```
spdsht_features = np.array(train_df_spdsht.columns.values.tolist())
centroid_df = pd.DataFrame(centers, index = cluster_names)
centroid_df.columns = spdsht_features
centroid_df
```

Out[106]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIA
Cluster_0	45.143491	2.382637	498.196746	44.721893	125.362426	21.134615	3.377219	1.727811	2.074150	2.674556	257.150742	0.5
Cluster_1	50.844037	2.438589	201.141055	16.025476	92.181193	18.975917	4.119266	2.113532	2.299957	2.810643	570.221217	0.5
Cluster_2	48.456241	2.214304	197.018651	17.946915	92.044476	18.873745	3.845050	2.007174	2.087251	3.074434	245.274032	0.5
Cluster_3	47.775510	3.435306	1842.102041	118.285714	371.408163	48.775510	5.000000	2.673469	2.795918	0.734694	377.142857	0.3
Cluster_4	46.411765	4.291176	515.847059	60.976471	1067.529412	67.611765	9.082353	3.352941	2.835294	2.858824	470.823529	0.4
Cluster_5	46.811881	3.698680	378.775578	34.079208	454.346535	45.960396	6.590759	2.732673	3.087666	1.669967	464.950495	0.4
Cluster_6	45.638400	3.002219	494.485179	41.321944	138.695139	30.547415	3.961600	2.001600	3.142601	2.821803	598.368000	0.5
Cluster_7	44.626398	3.145951	934.608501	94.825503	238.225951	41.624161	3.995526	1.664430	3.096197	2.348159	400.342061	0.5

Save the cluster center values to a csv file.

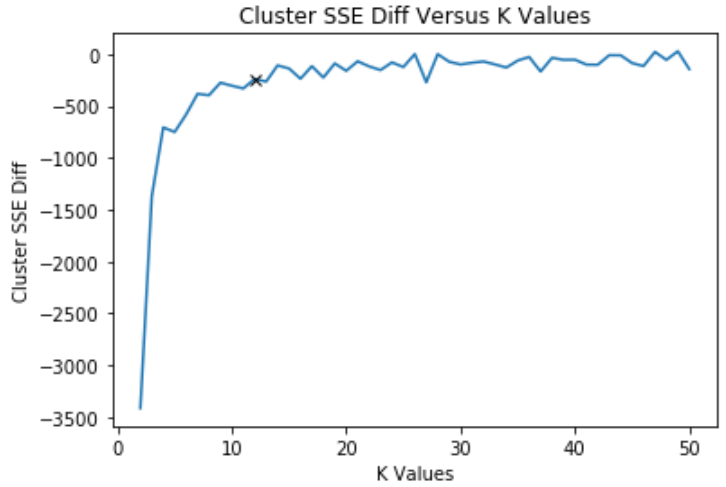
In [107]:
```
centroid_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NHANES Optimal Clusters.csv')
```

K Means Clustering With Normalized Full Training Set

Call the find_opt_k function to find the optimal k value for the clustering of the full normalized training dataset (before train/test split).

In [108]:
```
k = np.arange(1, 51, 1)
tol = -250

optimal_k = find_opt_k(data_all_norm_np, k, tol)
```

```
Optimal K Value: 12
```



Perform k means clustering with the optimal k values calculated above.

In [109]:
```
kmeans = KMeans(n_clusters = optimal_k, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_all_norm_np)
clusters = kmeans.predict(data_all_norm_np)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the final clustering with the optimal k value.

In [110]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_all_norm_np)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
23907.8460861
```

Create names to be used for each row of the cluster center dataframe created below.

In [111]:
```
cluster_names = []
uniq_cluster = np.unique(clusters)
for i in uniq_cluster:
    temp_str = 'Cluster_' + str(i)
    cluster_names.append(temp_str)
```

Create a dataframe with the cluster center values.

In [112]:
```
centroid_df = pd.DataFrame(centers, index = cluster_names)
centroid_df.columns = spdsht_features
centroid_df
```

Out[112]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Fe
Cluster_0	0.376740	0.489477	0.104198	0.027197	0.079011	0.025883	0.143223	0.095936	0.017420	0.019756	0.345792	1.000000e+00
Cluster_1	0.312461	0.386441	0.094686	0.019769	0.064955	0.025169	0.121729	0.094682	0.012871	0.019870	0.327263	1.000000e+00
Cluster_2	0.444655	0.899758	0.124960	0.033684	0.134056	0.033614	0.216724	0.080788	0.021946	0.011333	0.412096	2.220446e-16
Cluster_3	0.278338	0.458451	0.094406	0.020987	0.076507	0.029551	0.237466	0.154275	0.015652	0.022275	0.337527	2.664535e-15
Cluster_4	0.661024	0.506962	0.093523	0.022667	0.077527	0.025353	0.140104	0.077629	0.006890	0.015764	0.384387	9.062500e-01
Cluster_5	0.402879	0.843834	0.120254	0.033296	0.134409	0.026216	0.130731	0.047144	0.020653	0.013001	0.351869	1.000000e+00
Cluster_6	0.718619	0.364152	0.075739	0.021185	0.042795	0.018036	0.112162	0.087516	0.009159	0.015135	0.386426	4.324324e-01
Cluster_7	0.697814	0.508844	0.091417	0.024646	0.066286	0.025399	0.134192	0.084644	0.007561	0.015254	0.355895	3.911007e-01
Cluster_8	0.577841	0.521074	0.092332	0.024071	0.066523	0.023413	0.125568	0.084776	0.012322	0.017667	0.369361	1.000000e+00
Cluster_9	0.332930	0.507139	0.100327	0.030063	0.079678	0.024006	0.164879	0.095732	0.011880	0.021237	0.357293	9.550173e-01
Cluster_10	0.384696	0.413833	0.098696	0.022060	0.065894	0.031515	0.179089	0.130285	0.017104	0.021194	0.313663	3.885781e-15
Cluster_11	0.523785	0.628825	0.109927	0.025371	0.101144	0.026769	0.180208	0.095610	0.012363	0.016411	0.362366	2.997602e-15

Save the cluster center values to a csv file.

In [113]:
```
centroid_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NHANES Optimal Clusters Norm.csv')
```

K Means Clustering With K = 2 For Scoring on Full Training

Create 2 clusters on the full original (non-normalized) training data (before the train/test split) for use in comparison to the class variable.

In [114]:
```
kmeans = KMeans(n_clusters = 2, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_train_all)
clusters = kmeans.predict(data_train_all)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the above clustering.

In [115]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_train_all)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
489032287.811
```

Compute the completeness of the k = 2 clustering assignments and the class variable.

In [116]:
```
print 'Clustering Completeness Score:', completeness_score(target_all, clusters)
```

```
Clustering Completeness Score: 0.000326377661238
```

Compute the homogeneity of the k = 2 clustering assignments and the class variable.

In [117]:
```
print 'Clustering Homogeneity Score:', homogeneity_score(target_all, clusters)
```

```
Clustering Homogeneity Score: 0.00105871216439
```

K Means Clustering With K = 2 For Scoring on Normalized Full Training

Create 2 clusters on the full normalized training data (before train/test split) for use in comparison to the class variable.

In [118]:
```
kmeans = KMeans(n_clusters = 2, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_all_norm_np)
clusters = kmeans.predict(data_all_norm_np)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the above clustering.

In [119]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_all_norm_np)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
29217.2051317
```

Compute the completeness of the k = 2 clustering assignments and the class variable.

In [120]:
```
print 'Clustering Completeness Score:', completeness_score(target_all, clusters)
```

```
Clustering Completeness Score: 0.0302735584693
```

Compute the homogeneity of the k = 2 clustering assignments and the class variable.

In [121]:
```
print 'Clustering Homogeneity Score:', homogeneity_score(target_all, clusters)
```

```
Clustering Homogeneity Score: 0.132366825554
```

Create Functions Needed For K Means Clustering Prediction On Split Data

The classify_rocchio function below calculates the cosine distance between a prototype vector and a test instance passed in. The class that is passed back is the class corresponding to the closest distance.

In [122]:
```
def classify_rocchio(proto_vectors, training_classes, test_inst):

    proto_norm = np.array([np.linalg.norm(proto_vectors[i]) for i in range(len(proto_vectors))])
    test_case_norm = np.linalg.norm(test_inst)
    sims = np.dot(proto_vectors, test_inst) / (proto_norm * test_case_norm)
    dists = 1 - sims

    idx = np.argsort(dists) # sorting

    return training_classes[idx[0]], dists[idx]
```

The perform_rocchio function below classifies several testing instances using the classify_rocchio function. It also calculates the accuracy (the number of correctly classified instances divided by the total number of instances).

In [123]:
```
def perform_rocchio(proto_vectors, train_classes, test_data, test_classes, verbose = 0):

    num_tests = test_data.shape[0]
    uniq_classes = np.unique(train_classes)

    if verbose == 1:
        print 'Doc Number\tPred Class\tActual Class\tEuclidean Distance'

    num_correct = 0
    for x in range(num_tests):
        pred_class, distances = classify_rocchio(proto_vectors, uniq_classes, test_data[x,:])
        if pred_class == test_classes[x]:
            num_correct += 1

        if verbose == 1:
            print '{}\t\t{}\t\t{}\t\t{}'.format(x, int(pred_class), test_classes[x], distances[0])
    calc_accuracy = float(num_correct) / num_tests

    return calc_accuracy
```

K Means With K = 2 For Scoring And Prediction on Split Data

Perform clustering on the training data that was split with 80% of the original dataset.

In [124]:
```
kmeans = KMeans(n_clusters = 2, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_train_np)
clusters = kmeans.predict(data_train_np)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the clustering above.

In [125]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_train_np)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
399531916.162
```

Compute the completeness of the k = 2 clustering assignments and the class variable.

In [126]:
```
print 'Clustering Completeness Score:', completeness_score(target_train_np, clusters)
```

```
Clustering Completeness Score: 0.000250870251492
```

Compute the homogeneity of the k = 2 clustering assignments and the class variable.

In [127]:
```
print 'Clustering Homogeneity Score:', homogeneity_score(target_train_np, clusters)
```

```
Clustering Homogeneity Score: 0.000824299250945
```

Call the perform_rocchio function to perform the prediction of the training instances to see how well they are classified.

In [128]:
```
acc = perform_rocchio(centers, clusters, data_train_np, target_train_np)
print 'Clustering Prediction Accuracy of Training Data:', acc
```

```
Clustering Prediction Accuracy of Training Data: 0.375957375957
```

Call the perform_rocchio function to perform the prediction of the testing instances (the 20% from the train/test split).

In [129]:
```
acc = perform_rocchio(centers, clusters, data_test_np, target_test_np)
print 'Clustering Prediction Accuracy of Testing Data:', acc
```

```
Clustering Prediction Accuracy of Testing Data: 0.411451398136
```

K Means With K = 2 For Scoring And Prediction on Normalized Split Data

Perform clustering on the normalized training data that was split with 80% of the original dataset.

In [130]:
```
kmeans = KMeans(n_clusters = 2, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_train_norm_np)
clusters = kmeans.predict(data_train_norm_np)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the clustering above.

In [131]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_train_norm_np)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
23392.0780824
```

Compute the completeness of the k = 2 clustering assignments and the class variable.

In [132]:
```
print 'Clustering Completeness Score:', completeness_score(target_train_np, clusters)
```

```
Clustering Completeness Score: 0.0301048159726
```

Compute the homogeniety of the k = 2 clustering assignments and the class variable.

In [133]:
```
print 'Clustering Homogeneity Score:', homogeneity_score(target_train_np, clusters)
```

```
Clustering Homogeneity Score: 0.134885949373
```

Call the perform_rocchio function to perform the prediction of the normalized training instances to see how well they are classified.

In [134]:
```
acc = perform_rocchio(centers, clusters, data_train_norm_np, target_train_np)
print 'Clustering Prediction Accuracy of Training Data:', acc
```

```
Clustering Prediction Accuracy of Training Data: 0.324675324675
```

Call the perform_rocchio function to perform the prediction of the normalized testing instances (the 20% from the train/test split).

In [135]:
```
acc = perform_rocchio(centers, clusters, data_test_norm_np, target_test_np)
print 'Clustering Prediction Accuracy of Testing Data:', acc
```

```
Clustering Prediction Accuracy of Testing Data: 0.315579227696
```

Perform PCA on the Normalized Full Training Set

Perform PCA on the full normalized training data (before the train/test split) with the number of features set to the number of variables in the dataset. This is done so the variance explained by each variable can be used to determine the number of components needed to capture 90% of the total variance.

In [136]:
```
num_features = data_all_norm_np.shape[1]
pca_norm = decomposition.PCA(n_components = num_features)
data_full_norm_trans = pca_norm.fit(data_all_norm_np).transform(data_all_norm_np)
```

In [137]:
```
np.set_printoptions(precision=2, suppress=True)
```

In [138]:
```
var_explained = pca_norm.explained_variance_ratio_
print 'Variance Explained By PCA Components = ', var_explained.sum()
print var_explained
```

```
Variance Explained By PCA Components =  1.0
[ 0.13  0.07  0.06  0.05  0.05  0.04  0.03  0.03  0.03  0.03  0.03  0.02
  0.02  0.02  0.02  0.02  0.02  0.02  0.02  0.02  0.01  0.01  0.01  0.01
  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01
  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
```

Find the number of components needed to cover 90% of the variance.

In [139]:
```
val = np.arange(1,len(var_explained), 1)
desired_var = 0.90

for i in val:
    temp_sum = var_explained[0:i].sum()
    if temp_sum > desired_var:
        break

num_pca_features = i
print 'Variance Explained By', num_pca_features, 'PCA Components = ', var_explained[0:num_pca_features].sum()
```

```
Variance Explained By 38 PCA Components =  0.903940786395
```

Perform PCA with the number of components required for 90% of the variance.

In [140]:
```
pca_norm = decomposition.PCA(n_components = num_pca_features)
data_full_norm_trans = pca_norm.fit(data_all_norm_np).transform(data_all_norm_np)
```

In [141]:
```
var_explained = pca_norm.explained_variance_ratio_
print 'Variance Explained By First', num_pca_features, '=', var_explained.sum()
print var_explained
```

```
Variance Explained By First 38 = 0.903815542161
[ 0.13  0.07  0.06  0.05  0.05  0.04  0.03  0.03  0.03  0.03  0.03  0.02
  0.02  0.02  0.02  0.02  0.02  0.02  0.02  0.02  0.01  0.01  0.01  0.01
  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01
  0.01  0.01]
```

Get and print the PCA components created.

In [142]:
```
pca_comps_norm = pca_norm.components_
pca_comp_norm_df = pd.DataFrame(pca_comps_norm, columns = spdsht_features)
pca_comp_norm_df.head()
```

Out[142]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Fema
0	0.087045	-0.014325	-0.003396	-0.000193	-0.004972	-0.002587	-0.011775	-0.006976	-0.002178	-0.000090	0.011763	0.085884
1	0.074975	-0.111405	-0.007730	-0.002289	-0.018634	0.000462	0.005691	0.016960	-0.002409	0.000415	-0.009993	-0.377495
2	0.063128	0.263650	0.020652	0.007222	0.039607	0.004334	0.026990	-0.014244	0.001821	-0.003299	0.022038	-0.410983
3	0.187056	0.116022	0.003370	0.005173	0.008300	-0.001605	-0.045154	-0.043647	-0.003083	-0.001424	0.009253	0.286939
4	0.117083	-0.093705	-0.015223	-0.007628	-0.018604	-0.001999	-0.005737	0.004354	-0.001862	-0.002845	0.002827	-0.146434

Save the PCA components to a csv file.

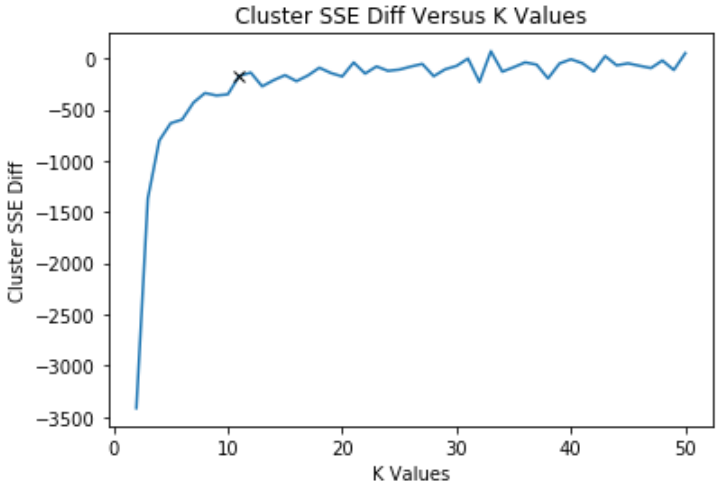
In [143]:
```
pca_comp_norm_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NHANES Full Normalized Training
 PCA Components.csv')
```

Find K Value of Clustering with Normalized Full Training PCA Variables

Call the find_opt_k function to find the optimal k value for the clustering of the PCA components.

In [144]:
```
k = np.arange(1, 51, 1)
tol = -200

optimal_k = find_opt_k(data_full_norm_trans, k, tol)
```

```
Optimal K Value: 11
```



Perform k means clustering with the optimal k values calculated above.

In [145]:
```
kmeans = KMeans(n_clusters = optimal_k, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_full_norm_trans)
clusters = kmeans.predict(data_full_norm_trans)
centers = kmeans.cluster_centers_
```

In [146]:
```
print clusters.shape
```

```
(3754L,)
```

Calculate the sum square error of the final clustering with the optimal k value.

In [147]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_full_norm_trans)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
20993.1115436
```

Create names to be used for each row of the cluster center dataframe created below.

In [148]:
```
cluster_names = []
uniq_cluster = np.unique(clusters)
for i in uniq_cluster:
    temp_str = 'Cluster_' + str(i)
    cluster_names.append(temp_str)
```

Create a dataframe with the cluster center values.

In [149]:
```
centroid_df = pd.DataFrame(centers, index = cluster_names)
centroid_df
```

Out[149]:

	0	1	2	3	4	5	6	7	8	9	10	11
Cluster_0	0.239095	0.058346	-0.599115	1.007538	0.523839	-0.073311	-0.064639	0.095184	-0.080797	-0.129234	-0.023698	-0.136282
Cluster_1	-0.987256	-0.016612	0.198918	-0.661052	0.418071	0.391549	0.087089	-0.121072	0.012615	0.192366	0.025533	0.036975
Cluster_2	-0.710055	-0.886410	-0.207223	0.275228	-0.038128	-0.037585	-0.421928	-0.372934	0.281728	0.055423	0.125447	-0.044331
Cluster_3	-0.837941	-0.554270	0.956090	0.341017	-0.234736	-0.176421	0.246569	0.140189	-0.022571	-0.171441	-0.093032	0.081543
Cluster_4	1.489718	0.521954	0.760062	-0.263386	0.295540	0.113044	0.086337	0.057153	-0.013060	0.016463	-0.029952	0.083284
Cluster_5	1.817754	-0.283308	-0.344322	0.224854	0.197248	-0.081906	-0.000390	0.047369	-0.047802	-0.184219	0.026944	-0.102265
Cluster_6	-0.065452	0.970753	0.352580	0.419165	0.751281	0.101945	-0.041121	0.124022	-0.017062	-0.114950	0.031184	-0.108305
Cluster_7	1.014296	-0.534932	0.062333	-0.655415	-0.638162	-0.034070	-0.080556	-0.086550	-0.002290	0.194185	-0.007214	-0.001649
Cluster_8	-0.434793	0.235318	-0.824209	0.345566	-0.778811	-0.209530	-0.087146	0.050175	-0.013455	-0.074691	0.006564	0.060010
Cluster_9	-0.785903	-0.769892	-0.825931	-0.124052	0.154289	0.048639	0.209997	0.139289	-0.121333	0.107127	-0.048311	0.041282
Cluster_10	-0.794252	1.029662	0.003474	-0.355744	-0.365688	-0.063018	-0.053838	-0.080864	0.057088	0.046134	0.019557	0.047125

Save the cluster centers to a csv file.

In [150]:
```
centroid_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\NHANES Optimal Clusters PCA Norm.csv'
)
```

Cluster Scoring with PCA Data From Normalized Full Training and K = 2

Create 2 clusters on the PCA data for use in comparison to the class variable.

In [151]:
```
kmeans = KMeans(n_clusters = optimal_k, max_iter = 500, verbose = 0, n_init = 5, init = "k-means++")
kmeans.fit(data_full_norm_trans)
clusters = kmeans.predict(data_full_norm_trans)
centers = kmeans.cluster_centers_
```

Calculate the sum square error of the above clustering.

In [152]:
```
sse_center_dist = calc_cluster_error(centers, clusters, data_full_norm_trans)
print 'Total Clustering Sum Squared Error:'
print sse_center_dist
```

```
Total Clustering Sum Squared Error:
21081.7474791
```

Compute the completeness of the k = 2 clustering assignments and the class variable.

In [153]:
```
print 'Clustering Completeness Score:', completeness_score(target_all, clusters)
```

```
Clustering Completeness Score: 0.0117582709867
```

Compute the homogeneity of the k = 2 clustering assignments and the class variable.

In [154]:
```
print 'Clustering Homogeneity Score:', homogeneity_score(target_all, clusters)
```

```
Clustering Homogeneity Score: 0.185802855329
```

---------- Linear Regression ----------

Create Target and Training Dataset For Linear Regression (BMI is the Target)

Create a target data set for use in regression with the BMI variable (is a continuous variable which will work for linear regression).

In [155]:
```
target_reg_df = nhanes_data['BMI_Perc']
target_reg_df.head()
```

Out[155]:
```
SEQN
73557    18.459084
73559    19.430069
73562    28.011975
73564    25.270377
73565    19.928276
Name: BMI_Perc, dtype: float64
```

Create a training data set for use in regression with the BMI variable removed.

In [156]:
```
train_reg_df = nhanes_data[:]
train_reg_df = train_reg_df.drop('BMI_Perc', 1)
```

Remove all of the variables that were used to create the BMI variable, as well as the original obesity indicator variable. This was done because these variables would have a strong relation to the BMI due to its calculation. These variable could take over in any modelling and prediction because of their strong relation. Since we already know they are related because they were used in the calculation, they are also of no interest.

Note that this code was run 2 different ways. The first way was with WHD010 and WHD020 removed as is shown below (these are the original height and weight variables used to calculate BMI). The second way was to leave these variables in the dataset to see what impact they have. I found that there was fairly poor performance with modelling and prediction without these variables and wanted to see how performance was affected by them.

In [157]:
```
train_reg_df = train_reg_df.drop('Obese_Ind', 1)
train_reg_df = train_reg_df.drop('Height_m', 1)
train_reg_df = train_reg_df.drop('Weight_kg', 1)
train_reg_df = train_reg_df.drop('WHD010', 1)
train_reg_df = train_reg_df.drop('WHD020', 1)
```

In [158]:
```
print train_reg_df.head()
```

SEQN	RIAGENDR	RIDAGEYR	RIDRETH1	DMDEDUC2
73557	Male	69	Non_Hispanic_Black	High_School_Grad_GED
73559	Male	72	Non_Hispanic_White	Some_College_AA
73562	Male	56	Mexican_American	Some_College_AA
73564	Female	61	Non_Hispanic_White	College_Grad_And_Above
73565	Male	42	Other_Hispanic	High_School_Grad_GED

SEQN	DMDMARTL	INDHHIN2	INDFMPIR	BPQ020	CBD070	CBD090	CBD120
73557	Separated	15000_to_19999	0.84	Yes	300.0	0.0	0.0
73559	Married	65000_to_74999	4.51	Yes	150.0	25.0	40.0
73562	Divorced	55000_to_64999	4.79	Yes	150.0	60.0	60.0
73564	Widowed	65000_to_74999	5.00	Yes	400.0	100.0	200.0
73565	Married	100000_And_Over	5.00	No	900.0	0.0	300.0

SEQN	CBD130	DIQ010	DBD895	DBD900	DBD905	DBD910	MCQ080	MCQ365A	MCQ365B
73557	85.0	Yes	8.0	8.0	0.0	4.0	Yes	Yes	No
73559	0.0	Yes	1.0	0.0	0.0	0.0	No	No	No
73562	0.0	No	14.0	14.0	0.0	0.0	Yes	Yes	Yes
73564	0.0	No	5.0	1.0	0.0	0.0	Yes	Yes	Yes
73565	40.0	No	15.0	2.0	7.0	0.0	No	No	No

SEQN	PAD680	SMQ020	HEQ010	HEQ030	HUQ051	HUQ071	MCQ010	MCQ082	MCQ086
73557	600.0	Yes	No	No	8_to_9	No	No	No	No
73559	300.0	Yes	No	No	2_to_3	No	No	No	No
73562	360.0	Yes	No	No	4_to_5	Yes	No	No	No
73564	60.0	No	No	No	2_to_3	No	Yes	No	No
73565	300.0	Yes	No	No	0	No	No	No	No

SEQN	MCQ160N	MCQ160B	MCQ160C	MCQ160D	MCQ160E	MCQ160F	MCQ160G	MCQ160M	MCQ160K
73557	No	No	No	No	No	Yes	No	No	No
73559	No	No	No	No	No	No	No	No	No
73562	Yes	No	Yes	No	Yes	No	No	Yes	No
73564	No	No	No	No	No	No	No	No	No
73565	No	No	No	No	No	No	No	No	No

SEQN	MCQ160L	MCQ160O	MCQ203	MCQ220
73557	No	No	No	No
73559	No	No	No	Yes
73562	No	No	No	No
73564	No	No	No	No
73565	No	No	No	No

Create dummy variable for all of the categorical variables in the regression training data set.

In [159]:
```
train_reg_df_spdsht = pd.get_dummies(train_reg_df)
```

Split the regression training dataset into 80% training and 20% testing.

In [160]:
```
data_reg_train, data_reg_test, target_reg_train, target_reg_test = train_test_split(train_reg_df_spdsht, target_reg_df, tes
t_size = 0.2, random_state = 45)
```

Perform min/max normalization on the split training and testing data sets.

In [161]:
```
min_max_scaler = preprocessing.MinMaxScaler().fit(data_reg_train)
data_train_reg_norm_np = min_max_scaler.transform(data_reg_train)
data_test_reg_norm_np = min_max_scaler.transform(data_reg_test)
```

Convert the training and testing dataframes to numpy arrays for use in sklearn functions.

In [162]:
```
data_reg_train_np = np.array(data_reg_train)
data_reg_test_np = np.array(data_reg_test)
target_reg_train_np = np.array(target_reg_train)
target_reg_test_np = np.array(target_reg_test)

print data_reg_train_np.shape
print target_reg_train_np.shape
print data_reg_test_np.shape
print target_reg_test_np.shape
```

```
(3003L, 101L)
(3003L,)
(751L, 101L)
(751L,)
```

Create functions Needed For Linear Regression

The find_reg_percent function calculates the mean squared error for a range of percentage of features passed in. The optimal percent is then given as the percent with the minimum mean squared error value. If more than one of the percentages have the same error value, the last it taken. I use the feature_selection.SelectPercentile function with f_regression selection to select the i percent of variables in the dataset (i is given in the loop containing the function call). Cross validation modelling is then done with the cross_validation.cross_val_score function to calculate the accuracy of each fold. The mean of these accuracy values is then used as the score for the i percentage.

In [163]:
```
def find_reg_percent(x_arr, y_arr, model, percent_list, cv_num):

    print '---------------------------------- Feature Selection ----------------------------------'

    results = []
    feature_select = []
    feature_scores = []
    features_pvals = []
    for i in percent_list:

        # select i percent of features from the training dataset
        fs = feature_selection.SelectPercentile(feature_selection.f_regression, percentile = i)
        x_arr_fs = fs.fit_transform(x_arr, y_arr)

        # calculate cross validation rmse and save them, the mean rmse, the weights, and the p values into lists
        scores = cross_validation.cross_val_score(model, x_arr_fs, y_arr, cv = cv_num, scoring = "neg_mean_squared_error")
        scores = np.absolute(scores)
        score_rmse = np.sqrt(scores.mean())
        results = np.append(results, score_rmse)
        feature_select.append(fs.get_support())
        feature_scores.append(fs.scores_)
        features_pvals.append(fs.pvalues_)


    # find the minimum mean error and use as the optimal index - if there more than 1, take the highest percentage value
    optimal_ndx = np.where(results == results.min())[0]
    if len(optimal_ndx) > 1:
        optimal_ndx = optimal_ndx[-1]
    optimal_percentile = int(percent_list[optimal_ndx])
    optimal_num_features = int(optimal_percentile*(x_arr.shape[1])/100)

    # get the optimal percentage and the corresponding number of features
    chosen_features = feature_select[int(optimal_ndx)]
    chosen_weights = feature_scores[int(optimal_ndx)]
    chosen_pvals = features_pvals[int(optimal_ndx)]

    # plot the root mean squared error versus the percentage of features
    pl.figure()
    pl.xlabel("Percentage of features selected")
    pl.ylabel("Cross Validation Root Mean Squared Error")
    pl.plot(percent_list, results)
    plt.plot(optimal_percentile, results[optimal_ndx], 'x', c='k')
    plt.show()

    return optimal_percentile, optimal_num_features, chosen_features, chosen_weights, chosen_pvals
```

The optimize_reg_features function below narrows down the list of features used in the final model by removing all features with p values over 0.05 (95% confidence level). P values over 0.05 correspond to features that fail the test of significance to the model. By removing features greater than 0.05, the number of features is further reduced and will help with preventing overfitting. The features are removed one by one, with the feature with the highest p value removed each time. Each time a feature is removed, the cross validation is performed and new p values are computed. This is repeated until none of the remaining features have p values over 0.05.

In [164]:
```
def optimize_reg_features(train_arr, target_arr, model, all_attrs, chosen_features, chosen_weights, chosen_pvals, cv_num
):

    print 'Feature Optimization (Removal of All Attributes with P Values >= 0.05)'
    print '-----------------------------------------------------------------------'

    train_all_df = pd.DataFrame(train_arr, columns = all_attrs)

    # create a dataframe with 1 row containing the weights from feature selection and create dataframe with selected fea
tures
    fs_features = all_attrs[chosen_features]
    train_fs_df = train_all_df[fs_features]

    # extract weights and p values corresponding to selected features
    fs_weights = chosen_weights[chosen_features]
    fs_p_vals = chosen_pvals[chosen_features]

    # create a dataframe with 1 row containing the weights from feature selection
    fs_w_arr = np.zeros((1, len(fs_weights)))
    fs_w_arr[0] = fs_weights
    fs_w_df = pd.DataFrame(fs_w_arr, columns = fs_features)

    # create a dataframe with 1 row containing the p values from feature selection
    fs_p_arr = np.zeros((1, len(fs_p_vals)))
    fs_p_arr[0] = fs_p_vals
    fs_p_df = pd.DataFrame(fs_p_arr, columns = fs_features)

    result = 0
    feature_select = []
    feature_scores = []
    features_pvals = []

    # create model to fit with all current features
    fs = feature_selection.SelectPercentile(feature_selection.f_regression, percentile = 100)

    while True:

        # find the index of the maximum p value.  if two have the same value, take the index of the first.
        p_vals = np.array(fs_p_df)
        max_ndx = np.where(p_vals == p_vals.max())
```

```
        max_ndx = np.where(p_vals == p_vals.max())[1]
        if len(max_ndx) > 1:
            max_ndx = max_ndx[0]
        max_p = p_vals[0, int(max_ndx)]

        # if the max p value is less than 0.05, create the list of new features and exit the loop
        if max_p < 0.05:
            new_fs_features = train_fs_df.columns.values.tolist()
            break

        # Remove attribute with max p value over 0.05
        col_names = train_fs_df.columns.values.tolist()
        max_attr = col_names[int(max_ndx)]

        train_fs_df = train_fs_df.drop(max_attr, 1)

        # create new array of training data and list of new features
        train_fs_np = np.array(train_fs_df)
        new_fs_features = train_fs_df.columns.values.tolist()

        # fit the model with the new training data (without the max feature)
        train_arr_fs = fs.fit_transform(train_fs_np, target_arr)

        # get the rmse, weights, p values, and mean rmse of the new model
        scores = cross_validation.cross_val_score(model, train_arr_fs, target_arr, cv = cv_num, scoring = "neg_mean_squa
red_error")
        scores = np.absolute(scores)
        result = np.sqrt(scores.mean())
        feature_select = fs.get_support()
        feature_scores = fs.scores_
        features_pvals = fs.pvalues_

        # populate a dataframe with the new feature weights
        fs_w_arr = np.zeros((1, len(feature_scores)))
        fs_w_arr[0] = feature_scores
        fs_w_df = pd.DataFrame(fs_w_arr, columns = new_fs_features)

        # populate a dataframe with the new feature p values
        fs_p_arr = np.zeros((1, len(features_pvals)))
        fs_p_arr[0] = features_pvals
        fs_p_df = pd.DataFrame(fs_p_arr, columns = new_fs_features)


    # populate a dataframe with the p values and weights of all of the variables
    all_info_df = pd.DataFrame(all_attrs)
    all_info_df["weights"] = chosen_weights
    all_info_df["p_values"] = chosen_pvals
```

```
    # populate a dataframe with the p values and weights of all newly reduced features
    fs_info_df = pd.DataFrame(new_fs_features)
    fs_info_df["weights"] = np.array(fs_w_df)[0]
    fs_info_df["p_values"] = np.array(fs_p_df)[0]

    # print the result of the optimization
    print 'Final Number of Training Set Attributes:', train_fs_df.shape[1]
    print 'Final Training Set Features Selection Root Mean Squared Error:', result
    print 'Final Training Set Attributes:'
    print new_fs_features
    print '\n'

    return train_fs_df, all_info_df, fs_info_df
```

The perform_reg_grid_search function below takes in a dictionary with the parameters and their ranges for usage in grid search model selection and performs the grid search with all of the parameter options.

In [165]:
```
def perform_reg_grid_search(model, data_train, target_train, data_test, target_test, param_dict, cv_num):

    print '---------------------------------- Parameter Grid Search ----------------------------------'

    num_params = len(param_dict.keys())
    gs = GridSearchCV(model, param_dict, verbose=1, cv=cv_num)
    %time _ = gs.fit(data_train, target_train)
    opt_params = gs.best_params_
    opt_score = gs.best_score_
    print '\n'
    print 'Grid Search Optimal Parameters:', opt_params
    print 'Grid Search Optimal Parameter Score:', opt_score
    print '\n'

    # Set the optimal grid search parameters
    for key, value in opt_params.items():
        model.set_params(**{key: value})
    print 'Final Model Parameter Settings:'
    print(model)
    print '\n'

    return model
```

The model_reg_data function below performs both the cross validation modelling and the final modelling with the full training set and the testing set. The cross validation modelling is what is used to refine the model parameter ranges to be used. The testing set is used only as an evaluation dataset to judge the final model performance.

In [166]:
```
def model_reg_data(model, data_train, target_train, data_test, target_test, cv_num):

    # Get the performance of the training and testing data sets
    print '--------------------------------- Training Data Performance ---------------------------------'
    print '\n'
    print 'Final Model Training Set Cross Validation Results'
    print '-------------------------------------------------'

    model_reg_cv_data(model, data_train, target_train, cv_num)
    print '\n'

    # Fit the training data to the optimal model, using the features from feature selection
    model.fit(data_train, target_train)

    print 'Final Model Full Training Set Results'
    print '-------------------------------------'
    train_pred = model.predict(data_train)
    train_rmse = np.sqrt(metrics.mean_squared_error(target_train, train_pred))
    train_r2 = metrics.r2_score(target_train, train_pred)
    print "Root Mean Squared Error:"
    print train_rmse, "\n"
    print "R-Squared:"
    print train_r2, "\n"

    # Return the model coefficients
    model_coefs = model.coef_

    print '\n'
    print '--------------------------------- Testing Data Performance ---------------------------------'
    print '\n'
    print 'Final Model Testing Set Results'
    print '-------------------------------'

    test_pred = model.predict(data_test)
    test_rmse = metrics.mean_squared_error(target_test, test_pred)
    test_r2 = metrics.r2_score(target_test, test_pred)
    print "Root Mean Squared Error:"
    print test_rmse, "\n"
    print "R-Squared:"
    print test_r2, "\n"


    return model, model_coefs
```

The model_reg_cv_data function below performs cross validation on a training and target dataset passed in. The mean accuracy of the training and testing datasets are printed at the end.

In [167]:
```
def model_reg_cv_data(model, data_train, target_train, cv_num):

    # generate the folds for use in each iteration of the cross validation
    kf = KFold(len(data_train), n_folds = cv_num, shuffle=True, random_state=0)
    cross_rmse_train = 0
    cross_rmse_test = 0
    cross_r2_train = 0
    cross_r2_test = 0
    for trainNdx, testNdx in kf:

        # fit a model to the current cross validation training fold and calculate the training accuracy
        model.fit(data_train[trainNdx,:], target_train[trainNdx])
        train_pred = model.predict(data_train[trainNdx,:])
        cross_rmse_train += metrics.mean_squared_error(target_train[trainNdx], train_pred)
        cross_r2_train += metrics.r2_score(target_train[trainNdx], train_pred)

        # calculate the testing accuracy of the current fold
        test_pred = model.predict(data_train[testNdx,:])
        cross_rmse_test += metrics.mean_squared_error(target_train[testNdx], test_pred)
        cross_r2_test += metrics.r2_score(target_train[testNdx], test_pred)

    rmse_cv_train = cross_rmse_train/cv_num
    rmse_cv_test = cross_rmse_test/cv_num
    r2_cv_train = cross_r2_train/cv_num
    r2_cv_test = cross_r2_test/cv_num

    print cv_num, 'Fold Cross Validation Training Average Root Mean Squared Error:'
    print rmse_cv_train
    print cv_num, 'Fold Cross Validation Training Average R-Squared:'
    print r2_cv_train
    print '\n'
    print cv_num, 'Fold Cross Validation Testing Average Root Mean Squared Error:'
    print rmse_cv_test
    print cv_num, 'Fold Cross Validation Testing Average R-Squared:'
    print r2_cv_test
```

The create_opt_reg_model function below is a wrapper function that calls the find_percent function, the optimize features function, creates the datasets that are the result of the two feature selection steps, call the grid search function, then calls the model data function.

In [168]:
```
def create_opt_reg_model(model, data_train, target_train, data_test, target_test, percent_list, param_dict, cv_num, attrs):

    # Find the optimal list of features to use
    opt_percent, opt_num, chosen_attr, chosen_w, chosen_p = find_reg_percent(data_train, target_train, model, percent_list,
 cv_num)
    print 'Optimal Percent:', opt_percent
    print 'Optimal Number of Features:', opt_num
    print 'Features Chosen:'
    print attrs[chosen_attr]
    print '\n'

    train_fs_df, all_info_df, fs_info_df = optimize_reg_features(data_train, target_train, model, attrs, chosen_attr, chose
n_w, chosen_p, cv_num)

    # Get the training and testing data with chosen features
    selected_features = train_fs_df.columns.values.tolist()
    train_arr_fs = np.array(train_fs_df)

    test_df = pd.DataFrame(data_test, columns = attrs)
    test_fs_df = test_df[selected_features]
    test_arr_fs = np.array(test_fs_df)

    # Perform grid search with parameter dictionary passed in
    opt_model = perform_reg_grid_search(model, train_arr_fs, target_train, test_arr_fs, target_test, param_dict, cv_num)

    # Perform grid search with parameter dictionary passed in
    final_model, model_coefs = model_reg_data(opt_model, train_arr_fs, target_train, test_arr_fs, target_test, cv_num)

    # add model coef values into the feature selection info dataframe (with the feature selection weights and p values)
    fs_info_df["coeffs"] = model_coefs

    return final_model, all_info_df, fs_info_df, selected_features
```

Create array of feature names used for linear regression modelling

In [169]:
```
reg_spdsht_features = np.array(data_reg_train.columns.values.tolist())
print reg_spdsht_features.shape

reg_spdsht_features_df = pd.DataFrame(reg_spdsht_features)
reg_spdsht_features_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\Reg Dataset Training Featu
res.csv')
```

```
(101L,)
```

Linear Regression On Original Training

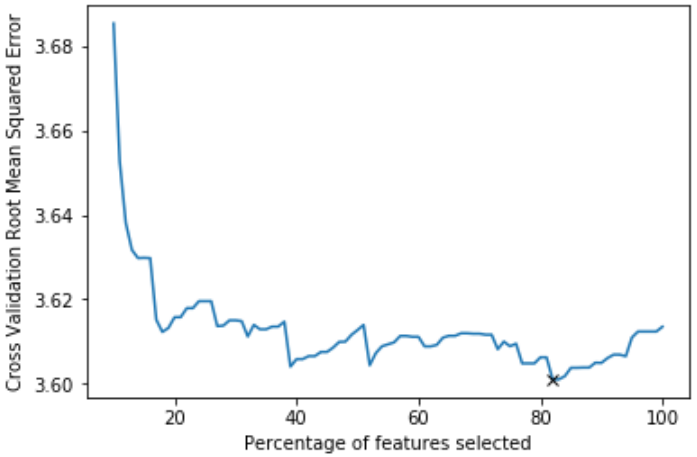
Call the find percent and optimize feature functions to perform feature selection on the original (non-normalized) 80% split training data. Create new training and testing dataframes and numpy arrays that contain only the features from feature selection. Call the model data function with these new training and testing arrays. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best RMSE and model complexity trade-off, is not overfit).

In [170]:
```
cv = 10
lin_reg = LinearRegression(fit_intercept=True)
percentages = a = np.arange(10, 101, 1)

print 'Percentage List Used:'
print percentages
print '\n'

# Find the optimal list of features to use
opt_percent, opt_num, chosen_attr, chosen_w, chosen_p = find_reg_percent(data_reg_train_np, target_reg_train_np, lin_reg, p
ercentages, cv)
print 'Optimal Percent:', opt_percent
print 'Optimal Number of Features:', opt_num
print 'Features Chosen:'
print reg_spdsht_features[chosen_attr]
print '\n'

train_fs_df, lin_all_info_df, lin_fs_info_df = optimize_reg_features(data_reg_train_np, target_reg_train_np, lin_reg, reg_s
pdsht_features, chosen_attr, chosen_w, chosen_p, cv)

# Get the training and testing data with chosen features
selected_features = train_fs_df.columns.values.tolist()
train_arr_fs = np.array(train_fs_df)

test_df = pd.DataFrame(data_reg_test_np, columns = reg_spdsht_features)
test_fs_df = test_df[selected_features]
test_arr_fs = np.array(test_fs_df)

# Perform grid search with parameter dictionary passed in
lin_model, lin_model_coefs = model_reg_data(lin_reg, train_arr_fs, target_reg_train_np, test_arr_fs, target_reg_test_np, cv
)

# add model coef values into the feature selection info dataframe (with the feature selection weights and p values)
lin_fs_info_df["coeffs"] = lin_model_coefs
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]
```

```
---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 82
Optimal Number of Features: 82
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD120' 'CBD130' 'DBD895' 'DBD900'
 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male' 'RIDRETH1_Mexican_American'
 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Non_Hispanic_White'
 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic' 'DMDEDUC2_9th_to_12th_No_Grad'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDEDUC2_Less_Than_9th' 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Divorced'
 'DMDMARTL_Living_W_Partner' 'DMDMARTL_Married' 'DMDMARTL_Never_Married'
 'DMDMARTL_Widowed' 'INDHHIN2_0_to_4999' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_15000_to_19999' 'INDHHIN2_25000_to_34999'
 'INDHHIN2_35000_to_44999' 'INDHHIN2_5000_to_9999'
 'INDHHIN2_55000_to_64999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_Yes' 'HEQ010_No' 'HEQ010_Yes' 'HUQ051_0' 'HUQ051_1'
 'HUQ051_10_to_12' 'HUQ051_16_Or_More' 'HUQ051_2_to_3' 'HUQ051_4_to_5'
 'HUQ051_6_to_7' 'HUQ071_No' 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes'
 'MCQ082_No' 'MCQ082_Yes' 'MCQ086_No' 'MCQ086_Yes' 'MCQ160N_No'
 'MCQ160N_Yes' 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160C_No' 'MCQ160C_Yes'
 'MCQ160D_No' 'MCQ160D_Yes' 'MCQ160E_No' 'MCQ160E_Yes' 'MCQ160F_No'
 'MCQ160F_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_No' 'MCQ160K_Yes'
 'MCQ160L_No' 'MCQ160L_Yes' 'MCQ160O_No' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 49
Final Training Set Features Selection Root Mean Squared Error: 3.61156016967
Final Training Set Attributes:
['INDFMPIR', 'CBD120', 'DBD900', 'PAD680', 'RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_H
ispanic_Black', 'RIDRETH1_Other', 'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'DMDEDUC2_Some_Colleg
e_AA', 'INDHHIN2_100000_And_Over', 'INDHHIN2_15000_to_19999', 'INDHHIN2_35000_to_44999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010
_Borderline', 'DIQ010_No', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Ye
s', 'HUQ051_0', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ051_2_to_3', 'HUQ051_4_to_5', 'HUQ071_No', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160N_No', 'MCQ160N_Yes', 'MCQ160B_No', 'MCQ160B_Yes', 'MCQ160D_No', 'MCQ160D_Yes', 'MCQ160E_No', 'MCQ160E
_Yes', 'MCQ160M_No', 'MCQ160M_Yes', 'MCQ160K_No', 'MCQ160K_Yes', 'MCQ160O_No', 'MCQ160O_Yes']


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Average Root Mean Squared Error:
12.6459667069
10 Fold Cross Validation Training Average R-Squared:
```

```
0.418232104947


10 Fold Cross Validation Testing Average Root Mean Squared Error:
13.0512921608
10 Fold Cross Validation Testing Average R-Squared:
0.398670661663


Final Model Full Training Set Results
-------------------------------------
Root Mean Squared Error:
3.55893167682

R-Squared:
0.417319960556



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Root Mean Squared Error:
13.5783805429

R-Squared:
0.403255814878
```

Save the feature selection info with weights, p values, and coefs to a csv file.

In [171]:
```
lin_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\LinReg Feature Info.csv')
```

Linear Regression On Normalized Training

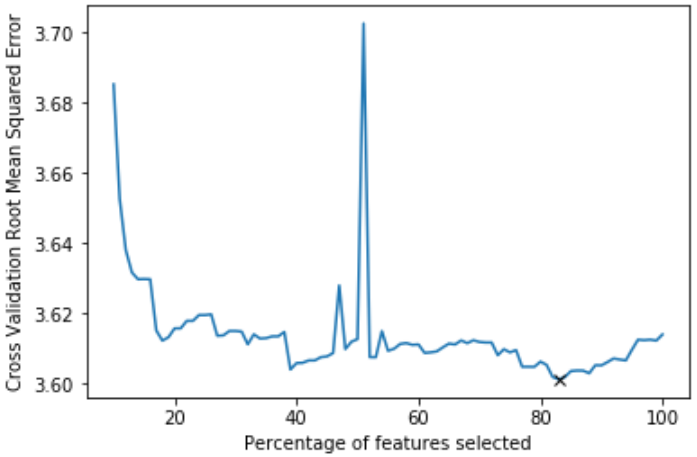
Call the find percent and optimize feature functions to perform feature selection on the normalized 80% split training data. Create new training and testing dataframes and numpy arrays that contain only the features from feature selection. Call the model data function with these new training and testing arrays.I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best RMSE and model complexity trade-off, is not overfit).

In [172]:
```
cv = 10
lin_reg = LinearRegression(fit_intercept=True)
percentages = a = np.arange(10, 101, 1)

print 'Percentage List Used:'
print percentages
print '\n'

# Find the optimal list of features to use
opt_percent, opt_num, chosen_attr, chosen_w, chosen_p = find_reg_percent(data_train_reg_norm_np, target_reg_train_np, lin_r
eg, percentages, cv)
print 'Optimal Percent:', opt_percent
print 'Optimal Number of Features:', opt_num
print 'Features Chosen:'
print reg_spdsht_features[chosen_attr]
print '\n'

train_fs_df, lin_norm_all_info_df, lin_norm_fs_info_df = optimize_reg_features(data_train_reg_norm_np, target_reg_train_np,
 lin_reg, reg_spdsht_features, chosen_attr, chosen_w, chosen_p, cv)

# Get the training and testing data with chosen features
selected_features = train_fs_df.columns.values.tolist()
train_arr_fs = np.array(train_fs_df)

test_df = pd.DataFrame(data_test_reg_norm_np, columns = reg_spdsht_features)
test_fs_df = test_df[selected_features]
test_arr_fs = np.array(test_fs_df)

# Perform grid search with parameter dictionary passed in
lin_model, lin_norm_model_coefs = model_reg_data(lin_reg, train_arr_fs, target_reg_train_np, test_arr_fs, target_reg_test_n
p, cv)

lin_norm_fs_info_df["coeffs"] = lin_norm_model_coefs
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]
```

---------- Feature Selection ----------



```
Optimal Percent: 83
Optimal Number of Features: 83
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD120' 'CBD130' 'DBD895' 'DBD900'
 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male' 'RIDRETH1_Mexican_American'
 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Non_Hispanic_White'
 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic' 'DMDEDUC2_9th_to_12th_No_Grad'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDEDUC2_Less_Than_9th' 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Divorced'
 'DMDMARTL_Living_W_Partner' 'DMDMARTL_Married' 'DMDMARTL_Never_Married'
 'DMDMARTL_Widowed' 'INDHHIN2_0_to_4999' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_15000_to_19999' 'INDHHIN2_25000_to_34999'
 'INDHHIN2_35000_to_44999' 'INDHHIN2_5000_to_9999'
 'INDHHIN2_55000_to_64999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_No' 'SMQ020_Yes' 'HEQ010_No' 'HEQ010_Yes' 'HUQ051_0' 'HUQ051_1'
 'HUQ051_10_to_12' 'HUQ051_16_Or_More' 'HUQ051_2_to_3' 'HUQ051_4_to_5'
 'HUQ051_6_to_7' 'HUQ071_No' 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes'
 'MCQ082_No' 'MCQ082_Yes' 'MCQ086_No' 'MCQ086_Yes' 'MCQ160N_No'
 'MCQ160N_Yes' 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160C_No' 'MCQ160C_Yes'
 'MCQ160D_No' 'MCQ160D_Yes' 'MCQ160E_No' 'MCQ160E_Yes' 'MCQ160F_No'
 'MCQ160F_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_No' 'MCQ160K_Yes'
 'MCQ160L_No' 'MCQ160L_Yes' 'MCQ160O_No' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 49
Final Training Set Features Selection Root Mean Squared Error: 3.61193049012
Final Training Set Attributes:
['INDFMPIR', 'CBD120', 'DBD900', 'PAD680', 'RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_H
ispanic_Black', 'RIDRETH1_Other', 'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'DMDEDUC2_Some_Colleg
e_AA', 'INDHHIN2_100000_And_Over', 'INDHHIN2_15000_to_19999', 'INDHHIN2_35000_to_44999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010
_Borderline', 'DIQ010_No', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Ye
s', 'HUQ051_0', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ051_2_to_3', 'HUQ051_4_to_5', 'HUQ071_No', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160N_No', 'MCQ160N_Yes', 'MCQ160B_No', 'MCQ160B_Yes', 'MCQ160D_No', 'MCQ160D_Yes', 'MCQ160E_No', 'MCQ160E
_Yes', 'MCQ160M_No', 'MCQ160M_Yes', 'MCQ160K_No', 'MCQ160K_Yes', 'MCQ160O_No', 'MCQ160O_Yes']


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Average Root Mean Squared Error:
12.7901375459
10 Fold Cross Validation Training Average R-Squared:
```

```
0.411653061486


10 Fold Cross Validation Testing Average Root Mean Squared Error:
13.2174542566
10 Fold Cross Validation Testing Average R-Squared:
0.390408189198


Final Model Full Training Set Results
-------------------------------------
Root Mean Squared Error:
3.55895182661

R-Squared:
0.417313362557



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Root Mean Squared Error:
13.5780742088

R-Squared:
0.403269277683
```

Save the feature selection info with weights, p values, and coefs to a csv file.

In [173]:
```
lin_norm_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\LinReg Norm Feature Info.csv'
)
```

Ridge Regression On Original Training

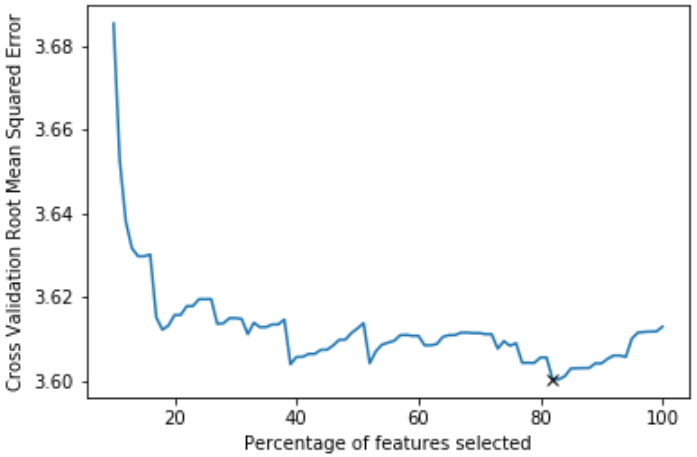
Call the create_opt_reg_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection with the 80% split training data (not normalized). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best RMSE and model complexity trade-off, is not overfit).

In [174]:
```
cv = 10
ridge_reg = Ridge(fit_intercept=True)
percentages = a = np.arange(10, 101, 1)

values = np.arange(0.001, 5, 0.005)

parameters = {
    'alpha': values
}

print 'Percentage List Used:'
print percentages
print '\n'

rid_model, rid_all_info_df, rid_fs_info_df, rid_features = create_opt_reg_model(ridge_reg, data_reg_train_np, target_reg_tr
ain_np, data_reg_test_np, target_reg_test_np, percentages, parameters, cv, reg_spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]
```

```
---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 82
Optimal Number of Features: 82
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD120' 'CBD130' 'DBD895' 'DBD900'
 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male' 'RIDRETH1_Mexican_American'
 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Non_Hispanic_White'
 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic' 'DMDEDUC2_9th_to_12th_No_Grad'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDEDUC2_Less_Than_9th' 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Divorced'
 'DMDMARTL_Living_W_Partner' 'DMDMARTL_Married' 'DMDMARTL_Never_Married'
 'DMDMARTL_Widowed' 'INDHHIN2_0_to_4999' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_15000_to_19999' 'INDHHIN2_25000_to_34999'
 'INDHHIN2_35000_to_44999' 'INDHHIN2_5000_to_9999'
 'INDHHIN2_55000_to_64999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_Yes' 'HEQ010_No' 'HEQ010_Yes' 'HUQ051_0' 'HUQ051_1'
 'HUQ051_10_to_12' 'HUQ051_16_Or_More' 'HUQ051_2_to_3' 'HUQ051_4_to_5'
 'HUQ051_6_to_7' 'HUQ071_No' 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes'
 'MCQ082_No' 'MCQ082_Yes' 'MCQ086_No' 'MCQ086_Yes' 'MCQ160N_No'
 'MCQ160N_Yes' 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160C_No' 'MCQ160C_Yes'
 'MCQ160D_No' 'MCQ160D_Yes' 'MCQ160E_No' 'MCQ160E_Yes' 'MCQ160F_No'
 'MCQ160F_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_No' 'MCQ160K_Yes'
 'MCQ160L_No' 'MCQ160L_Yes' 'MCQ160O_No' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 49
Final Training Set Features Selection Root Mean Squared Error: 3.61134016695
Final Training Set Attributes:
['INDFMPIR', 'CBD120', 'DBD900', 'PAD680', 'RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_H
ispanic_Black', 'RIDRETH1_Other', 'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'DMDEDUC2_Some_Colleg
e_AA', 'INDHHIN2_100000_And_Over', 'INDHHIN2_15000_to_19999', 'INDHHIN2_35000_to_44999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010
_Borderline', 'DIQ010_No', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Ye
s', 'HUQ051_0', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ051_2_to_3', 'HUQ051_4_to_5', 'HUQ071_No', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160N_No', 'MCQ160N_Yes', 'MCQ160B_No', 'MCQ160B_Yes', 'MCQ160D_No', 'MCQ160D_Yes', 'MCQ160E_No', 'MCQ160E
_Yes', 'MCQ160M_No', 'MCQ160M_Yes', 'MCQ160K_No', 'MCQ160K_Yes', 'MCQ160O_No', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
Wall time: 51.3 s


Grid Search Optimal Parameters: {'alpha': 4.9960000000000004}
Grid Search Optimal Parameter Score: 0.396547685307
```

```
Final Model Parameter Settings:
Ridge(alpha=4.9960000000000004, copy_X=True, fit_intercept=True,
   max_iter=None, normalize=False, random_state=None, solver='auto',
   tol=0.001)


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Average Root Mean Squared Error:
12.6463688949
10 Fold Cross Validation Training Average R-Squared:
0.418213593954


10 Fold Cross Validation Testing Average Root Mean Squared Error:
13.0437485464
10 Fold Cross Validation Testing Average R-Squared:
0.399018009808


Final Model Full Training Set Results
-------------------------------------
Root Mean Squared Error:
3.55897628071

R-Squared:
0.417305355071



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Root Mean Squared Error:
13.579439283

R-Squared:
0.403209285248


[Parallel(n_jobs=1)]: Done 10000 out of 10000 | elapsed:   51.1s finished
```

Save the feature selection info with weights, p values, and coefs to a csv file.

In [175]:
```
rid_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\Ridge Feature Info.csv')
```

Ridge Regression On Normalized Training

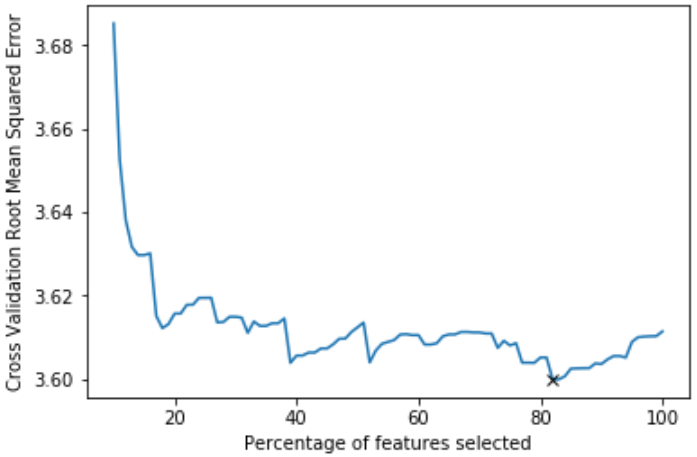
Call the create_opt_reg_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection with the normalized 80% split training data. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best RMSE and model complexity trade-off, is not overfit).

In [176]:
```
cv = 10
ridge_reg = Ridge(fit_intercept=True)
percentages = a = np.arange(10, 101, 1)

values = np.arange(0.001, 5, 0.005)

parameters = {
    'alpha': values
}

print 'Percentage List Used:'
print percentages
print '\n'

rid_model, rid_norm_all_info_df, rid_norm_fs_info_df, rid_features = create_opt_reg_model(ridge_reg, data_train_reg_norm_np
, target_reg_train_np, data_test_reg_norm_np, target_reg_test_np, percentages, parameters, cv, reg_spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]
```

---------- Feature Selection ----------



```
Optimal Percent: 82
Optimal Number of Features: 82
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD120' 'CBD130' 'DBD895' 'DBD900'
 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male' 'RIDRETH1_Mexican_American'
 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Non_Hispanic_White'
 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic' 'DMDEDUC2_9th_to_12th_No_Grad'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDEDUC2_Less_Than_9th' 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Divorced'
 'DMDMARTL_Living_W_Partner' 'DMDMARTL_Married' 'DMDMARTL_Never_Married'
 'DMDMARTL_Widowed' 'INDHHIN2_0_to_4999' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_15000_to_19999' 'INDHHIN2_25000_to_34999'
 'INDHHIN2_35000_to_44999' 'INDHHIN2_5000_to_9999'
 'INDHHIN2_55000_to_64999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_Yes' 'HEQ010_No' 'HEQ010_Yes' 'HUQ051_0' 'HUQ051_1'
 'HUQ051_10_to_12' 'HUQ051_16_Or_More' 'HUQ051_2_to_3' 'HUQ051_4_to_5'
 'HUQ051_6_to_7' 'HUQ071_No' 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes'
 'MCQ082_No' 'MCQ082_Yes' 'MCQ086_No' 'MCQ086_Yes' 'MCQ160N_No'
 'MCQ160N_Yes' 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160C_No' 'MCQ160C_Yes'
 'MCQ160D_No' 'MCQ160D_Yes' 'MCQ160E_No' 'MCQ160E_Yes' 'MCQ160F_No'
 'MCQ160F_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_No' 'MCQ160K_Yes'
 'MCQ160L_No' 'MCQ160L_Yes' 'MCQ160O_No' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 49
Final Training Set Features Selection Root Mean Squared Error: 3.61122946171
Final Training Set Attributes:
['INDFMPIR', 'CBD120', 'DBD900', 'PAD680', 'RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_H
ispanic_Black', 'RIDRETH1_Other', 'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'DMDEDUC2_Some_Colleg
e_AA', 'INDHHIN2_100000_And_Over', 'INDHHIN2_15000_to_19999', 'INDHHIN2_35000_to_44999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010
_Borderline', 'DIQ010_No', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Ye
s', 'HUQ051_0', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ051_2_to_3', 'HUQ051_4_to_5', 'HUQ071_No', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160N_No', 'MCQ160N_Yes', 'MCQ160B_No', 'MCQ160B_Yes', 'MCQ160D_No', 'MCQ160D_Yes', 'MCQ160E_No', 'MCQ160E
_Yes', 'MCQ160M_No', 'MCQ160M_Yes', 'MCQ160K_No', 'MCQ160K_Yes', 'MCQ160O_No', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
Wall time: 51.7 s


Grid Search Optimal Parameters: {'alpha': 4.9960000000000004}
Grid Search Optimal Parameter Score: 0.39670866644
```

```
Final Model Parameter Settings:
Ridge(alpha=4.9960000000000004, copy_X=True, fit_intercept=True,
   max_iter=None, normalize=False, random_state=None, solver='auto',
   tol=0.001)


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Average Root Mean Squared Error:
12.6467768134
10 Fold Cross Validation Training Average R-Squared:
0.418194828206


10 Fold Cross Validation Testing Average Root Mean Squared Error:
13.0391562356
10 Fold Cross Validation Testing Average R-Squared:
0.399231868716


Final Model Full Training Set Results
-------------------------------------
Root Mean Squared Error:
3.55901916543

R-Squared:
0.417291312354



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Root Mean Squared Error:
13.5907988654

R-Squared:
0.402710053052


[Parallel(n_jobs=1)]: Done 10000 out of 10000 | elapsed:   51.6s finished
```

Save the feature selection info with weights, p values, and coefs to a csv file.

In [177]:
```
rid_norm_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\Ridge Norm Feature Info.csv')
```

Lasso Regression On Original Training

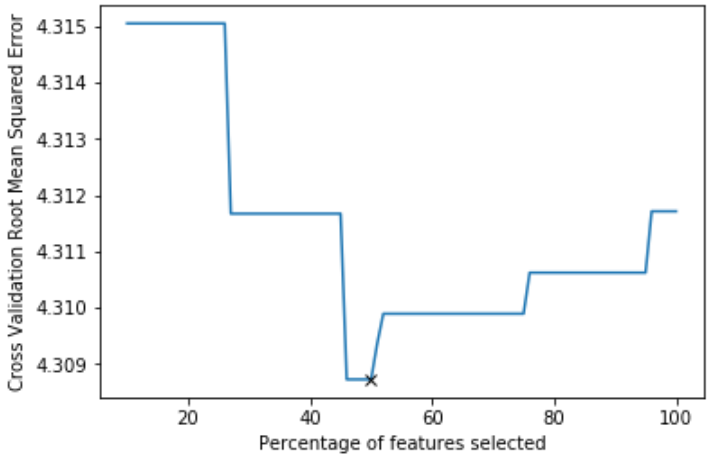
Call the create_opt_reg_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection with the 80% split training data (not normalized). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best RMSE and model complexity trade-off, is not overfit).

In [178]:
```
cv = 10
lasso_reg = Lasso(fit_intercept=True)
percentages = a = np.arange(10, 101, 1)

values = np.arange(0.001, 5, 0.005)

parameters = {
    'alpha': values
}

print 'Percentage List Used:'
print percentages
print '\n'

las_model, las_all_info_df, las_fs_info_df, las_features = create_opt_reg_model(lasso_reg, data_reg_train_np, target_reg_tr
ain_np, data_reg_test_np, target_reg_test_np, percentages, parameters, cv, reg_spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]
```

```
---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 50
Optimal Number of Features: 50
Features Chosen:
['INDFMPIR' 'CBD120' 'DBD900' 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male'
 'RIDRETH1_Mexican_American' 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Other'
 'DMDEDUC2_College_Grad_And_Above' 'DMDEDUC2_High_School_Grad_GED'
 'DMDEDUC2_Some_College_AA' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_15000_to_19999' 'INDHHIN2_35000_to_44999'
 'INDHHIN2_55000_to_64999' 'BPQ020_No' 'BPQ020_Yes' 'DIQ010_Borderline'
 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No' 'MCQ080_Yes' 'MCQ365A_No'
 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes' 'HUQ051_0' 'HUQ051_1'
 'HUQ051_16_Or_More' 'HUQ051_2_to_3' 'HUQ051_4_to_5' 'HUQ071_No'
 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes' 'MCQ160N_No' 'MCQ160N_Yes'
 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160D_No' 'MCQ160D_Yes' 'MCQ160E_No'
 'MCQ160E_Yes' 'MCQ160M_No' 'MCQ160M_Yes' 'MCQ160K_No' 'MCQ160K_Yes'
 'MCQ160O_No' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------
Final Number of Training Set Attributes: 49
Final Training Set Features Selection Root Mean Squared Error: 4.30872559269
Final Training Set Attributes:
['INDFMPIR', 'CBD120', 'DBD900', 'PAD680', 'RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_H
ispanic_Black', 'RIDRETH1_Other', 'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'DMDEDUC2_Some_Colleg
e_AA', 'INDHHIN2_100000_And_Over', 'INDHHIN2_15000_to_19999', 'INDHHIN2_35000_to_44999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010
_Borderline', 'DIQ010_No', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Ye
s', 'HUQ051_0', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ051_2_to_3', 'HUQ051_4_to_5', 'HUQ071_No', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160N_No', 'MCQ160N_Yes', 'MCQ160B_No', 'MCQ160B_Yes', 'MCQ160D_No', 'MCQ160D_Yes', 'MCQ160E_No', 'MCQ160E
_Yes', 'MCQ160M_No', 'MCQ160M_Yes', 'MCQ160K_No', 'MCQ160K_Yes', 'MCQ160O_No', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
Wall time: 53.9 s


Grid Search Optimal Parameters: {'alpha': 0.016}
Grid Search Optimal Parameter Score: 0.398899258521


Final Model Parameter Settings:
Lasso(alpha=0.016, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)


---------------------------------- Training Data Performance ----------------------------------
```

```
Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Average Root Mean Squared Error:
12.713858807
10 Fold Cross Validation Training Average R-Squared:
0.415107203357


10 Fold Cross Validation Testing Average Root Mean Squared Error:
13.0114283198
10 Fold Cross Validation Testing Average R-Squared:
0.400334721575


Final Model Full Training Set Results
-------------------------------------
Root Mean Squared Error:
3.56825117166

R-Squared:
0.41426432811



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Root Mean Squared Error:
13.6360051606

R-Squared:
0.40072332174


[Parallel(n_jobs=1)]: Done 10000 out of 10000 | elapsed:   53.8s finished
```

Save the feature selection info with weights, p values, and coefs to a csv file.

In [179]:
```
las_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\Lasso Feature Info.csv')
```

Lasso Regression On Normalized Training

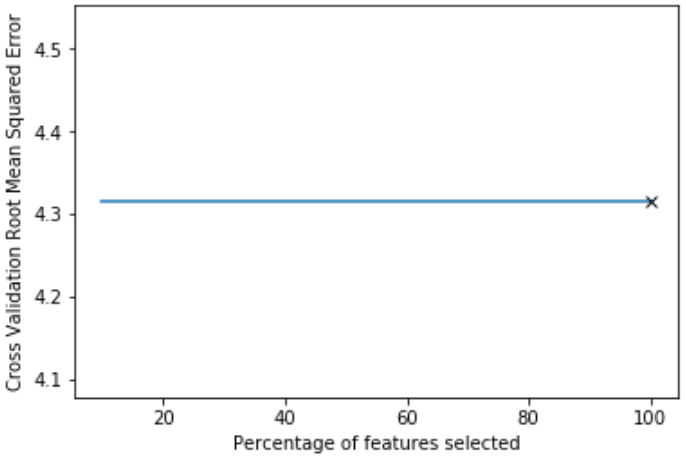
Call the create_opt_reg_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection with the normalized 80% split training data. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best RMSE and model complexity trade-off, is not overfit).

In [180]:
```
cv = 10
lasso_reg = Lasso(fit_intercept=True)
percentages = a = np.arange(10, 101, 1)

values = np.arange(0.001, 5, 0.005)

parameters = {
    'alpha': values
}

print 'Percentage List Used:'
print percentages
print '\n'

las_model, las_norm_all_info_df, las_norm_fs_info_df, las_features = create_opt_reg_model(lasso_reg, data_train_reg_norm_np
, target_reg_train_np, data_test_reg_norm_np, target_reg_test_np, percentages, parameters, cv, reg_spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]
```

```
---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 100
Optimal Number of Features: 101
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD090' 'CBD120' 'CBD130' 'DBD895'
 'DBD900' 'DBD905' 'DBD910' 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male'
 'RIDRETH1_Mexican_American' 'RIDRETH1_Non_Hispanic_Black'
 'RIDRETH1_Non_Hispanic_White' 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic'
 'DMDEDUC2_9th_to_12th_No_Grad' 'DMDEDUC2_College_Grad_And_Above'
 'DMDEDUC2_High_School_Grad_GED' 'DMDEDUC2_Less_Than_9th'
 'DMDEDUC2_Some_College_AA' 'DMDMARTL_Divorced' 'DMDMARTL_Living_W_Partner'
 'DMDMARTL_Married' 'DMDMARTL_Never_Married' 'DMDMARTL_Separated'
 'DMDMARTL_Widowed' 'INDHHIN2_0_to_4999' 'INDHHIN2_100000_And_Over'
 'INDHHIN2_10000_to_14999' 'INDHHIN2_15000_to_19999'
 'INDHHIN2_20000_to_24999' 'INDHHIN2_25000_to_34999'
 'INDHHIN2_35000_to_44999' 'INDHHIN2_45000_to_54999'
 'INDHHIN2_5000_to_9999' 'INDHHIN2_55000_to_64999'
 'INDHHIN2_65000_to_74999' 'INDHHIN2_75000_to_99999' 'BPQ020_No'
 'BPQ020_Yes' 'DIQ010_Borderline' 'DIQ010_No' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'SMQ020_No' 'SMQ020_Yes' 'HEQ010_No' 'HEQ010_Yes' 'HEQ030_No' 'HEQ030_Yes'
 'HUQ051_0' 'HUQ051_1' 'HUQ051_10_to_12' 'HUQ051_13_to_15'
 'HUQ051_16_Or_More' 'HUQ051_2_to_3' 'HUQ051_4_to_5' 'HUQ051_6_to_7'
 'HUQ051_8_to_9' 'HUQ071_No' 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes'
 'MCQ082_No' 'MCQ082_Yes' 'MCQ086_No' 'MCQ086_Yes' 'MCQ160N_No'
 'MCQ160N_Yes' 'MCQ160B_No' 'MCQ160B_Yes' 'MCQ160C_No' 'MCQ160C_Yes'
 'MCQ160D_No' 'MCQ160D_Yes' 'MCQ160E_No' 'MCQ160E_Yes' 'MCQ160F_No'
 'MCQ160F_Yes' 'MCQ160G_No' 'MCQ160G_Yes' 'MCQ160M_No' 'MCQ160M_Yes'
 'MCQ160K_No' 'MCQ160K_Yes' 'MCQ160L_No' 'MCQ160L_Yes' 'MCQ160O_No'
 'MCQ160O_Yes' 'MCQ203_No' 'MCQ203_Yes' 'MCQ220_No' 'MCQ220_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 49
Final Training Set Features Selection Root Mean Squared Error: 4.31505025066
Final Training Set Attributes:
['INDFMPIR', 'CBD120', 'DBD900', 'PAD680', 'RIAGENDR_Female', 'RIAGENDR_Male', 'RIDRETH1_Mexican_American', 'RIDRETH1_Non_H
ispanic_Black', 'RIDRETH1_Other', 'DMDEDUC2_College_Grad_And_Above', 'DMDEDUC2_High_School_Grad_GED', 'DMDEDUC2_Some_Colleg
e_AA', 'INDHHIN2_100000_And_Over', 'INDHHIN2_15000_to_19999', 'INDHHIN2_35000_to_44999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010
_Borderline', 'DIQ010_No', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Yes', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Ye
s', 'HUQ051_0', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ051_2_to_3', 'HUQ051_4_to_5', 'HUQ071_No', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160N_No', 'MCQ160N_Yes', 'MCQ160B_No', 'MCQ160B_Yes', 'MCQ160D_No', 'MCQ160D_Yes', 'MCQ160E_No', 'MCQ160E
_Yes', 'MCQ160M_No', 'MCQ160M_Yes', 'MCQ160K_No', 'MCQ160K_Yes', 'MCQ160O_No', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
Wall time: 53.4 s
```

```
Grid Search Optimal Parameters: {'alpha': 0.010999999999999999}
Grid Search Optimal Parameter Score: 0.398906476153


Final Model Parameter Settings:
Lasso(alpha=0.010999999999999999, copy_X=True, fit_intercept=True,
   max_iter=1000, normalize=False, positive=False, precompute=False,
   random_state=None, selection='cyclic', tol=0.0001, warm_start=False)


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Average Root Mean Squared Error:
12.6949770657
10 Fold Cross Validation Training Average R-Squared:
0.41597639595


10 Fold Cross Validation Testing Average Root Mean Squared Error:
13.0066534512
10 Fold Cross Validation Testing Average R-Squared:
0.400650028203


Final Model Full Training Set Results
-------------------------------------
Root Mean Squared Error:
3.56561444625

R-Squared:
0.415129655935



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Root Mean Squared Error:
13.6847387177

R-Squared:
```

```
0.398581573926

[Parallel(n_jobs=1)]: Done 10000 out of 10000 | elapsed:   53.3s finished
```

Save the feature selection info with weights, p values, and coefs to a csv file.

In [181]:
```
las_norm_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\Lasso Norm Feature Info.csv')
```

---------- Classification Revisited ----------

---------- Classification With Class Weights ----------

Decision Tree Classification With Feature Selection and Balanced Weighting

Call the create_opt_model function with the percentage list for feature selection and model parameters and their ranges for grid search model selection. I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

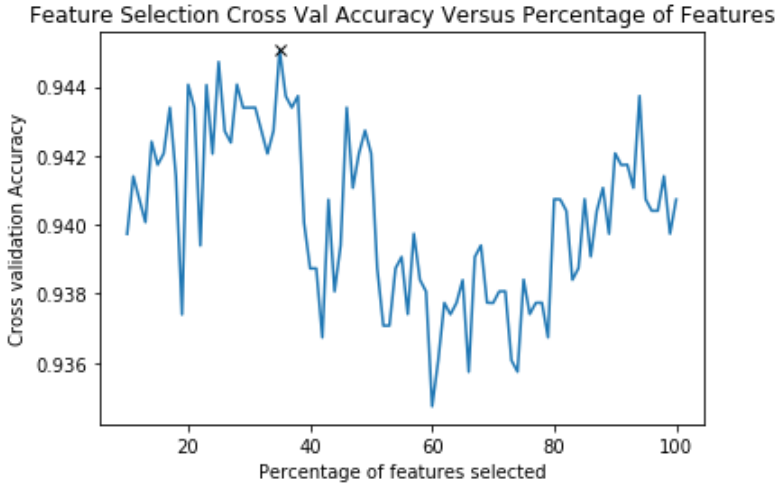
The percentage and parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [182]:
```
spdsht_features = np.array(data_train.columns.values.tolist())
print spdsht_features.shape

spdsht_features_df = pd.DataFrame(spdsht_features)
```

```
(101L,)
```

In [183]:
```
cv = 10
dt = tree.DecisionTreeClassifier(class_weight  = "balanced")
percentages = a = np.arange(10, 101, 1)

temp_leaf = np.arange(10, 201, 10)
temp_depth = np.arange(1, 9, 1)

parameters = {
    'criterion': ['entropy','gini'],
    'max_depth': temp_depth,
    'min_samples_leaf': temp_leaf,
    'min_samples_split': temp_leaf
}

print 'Percentage List Used:'
print percentages
print '\n'
print 'Parameters Used:'
print parameters
print '\n'

dt_model, dt_all_info_df, dt_fs_info_df, dt_features = create_opt_model(dt, data_train_np, target_train_np, data_test_np, t
arget_test_np, percentages, parameters, cv, spdsht_features)
```

```
Percentage List Used:
[ 10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100]


Parameters Used:
{'min_samples_split': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200]), 'criterion': ['entropy', 'gini'], 'max_depth': array([1, 2, 3, 4, 5, 6, 7, 8]),
'min_samples_leaf': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200])}


---------------------------------- Feature Selection ----------------------------------
```



```
Optimal Percent: 35
Optimal Number of Features: 35
Features Chosen:
['RIDAGEYR' 'INDFMPIR' 'CBD070' 'CBD090' 'CBD120' 'CBD130' 'DBD900'
 'DBD905' 'DBD910' 'PAD680' 'RIAGENDR_Female' 'RIAGENDR_Male'
 'RIDRETH1_Non_Hispanic_Black' 'RIDRETH1_Other' 'RIDRETH1_Other_Hispanic'
 'DMDEDUC2_College_Grad_And_Above' 'INDHHIN2_15000_to_19999'
 'INDHHIN2_5000_to_9999' 'BPQ020_No' 'BPQ020_Yes' 'DIQ010_Yes' 'MCQ080_No'
 'MCQ080_Yes' 'MCQ365A_No' 'MCQ365A_Yes' 'MCQ365B_No' 'MCQ365B_Yes'
 'HUQ051_1' 'HUQ051_16_Or_More' 'HUQ071_Yes' 'MCQ010_No' 'MCQ010_Yes'
 'MCQ160B_Yes' 'MCQ160M_Yes' 'MCQ160O_Yes']


Feature Optimization (Removal of All Attributes with P Values >= 0.05)
-----------------------------------------------------------------------
Final Number of Training Set Attributes: 35
Final Training Set Features Selection Accuracy: 0
Final Training Set Attributes:
['RIDAGEYR', 'INDFMPIR', 'CBD070', 'CBD090', 'CBD120', 'CBD130', 'DBD900', 'DBD905', 'DBD910', 'PAD680', 'RIAGENDR_Female',
'RIAGENDR_Male', 'RIDRETH1_Non_Hispanic_Black', 'RIDRETH1_Other', 'RIDRETH1_Other_Hispanic', 'DMDEDUC2_College_Grad_And_Abo
ve', 'INDHHIN2_15000_to_19999', 'INDHHIN2_5000_to_9999', 'BPQ020_No', 'BPQ020_Yes', 'DIQ010_Yes', 'MCQ080_No', 'MCQ080_Ye
s', 'MCQ365A_No', 'MCQ365A_Yes', 'MCQ365B_No', 'MCQ365B_Yes', 'HUQ051_1', 'HUQ051_16_Or_More', 'HUQ071_Yes', 'MCQ010_No',
'MCQ010_Yes', 'MCQ160B_Yes', 'MCQ160M_Yes', 'MCQ160O_Yes']


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 6400 candidates, totalling 64000 fits
Wall time: 6min 39s


Grid Search Optimal Parameters: {'min_samples_split': 20, 'criterion': 'gini', 'max_depth': 8, 'min_samples_leaf': 10}
Grid Search Optimal Parameter Score: 0.811188811189


Final Model Parameter Settings:
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=8,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=10, min_samples_split=20,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
```

```
10 Fold Cross Validation Training Accuracy:
0.841936146831
10 Fold Cross Validation Testing Accuracy:
0.812181616833


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.831501831502

Classification Report:
             precision    recall  f1-score   support

          0       1.00      0.83      0.90      2904
          1       0.16      1.00      0.28        99

avg / total       0.97      0.83      0.88      3003


Confussion Matrix:
[[2398  506]
 [   0   99]]



---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.838881491345

Classification Report:
             precision    recall  f1-score   support

          0       1.00      0.84      0.91       723
          1       0.17      0.89      0.29        28

avg / total       0.96      0.84      0.89       751


Confussion Matrix:
[[605 118]
 [  3  25]]
```

```
[Parallel(n_jobs=1)]: Done 64000 out of 64000 | elapsed:  6.7min finished
```

Compute the number of nodes present in the tree.

In [184]:
```
treeObj = dt_model.tree_
print 'Number of nodes in the tree:'
print treeObj.node_count
```

```
Number of nodes in the tree:
65
```

Save the chosen features and all features with their weights and p values to csv files.

In [185]:
```
dt_fs_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Balanced Chosen Features.csv')
```

In [186]:
```
dt_all_info_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Balanced All Features.csv')
```

Retreive and save the feature importances of the final model to a csv file.

In [187]:
```
feature_imp = dt_model.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = dt_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Balanced Feature Selection Impo
rtances.csv')
feature_imp_df
```

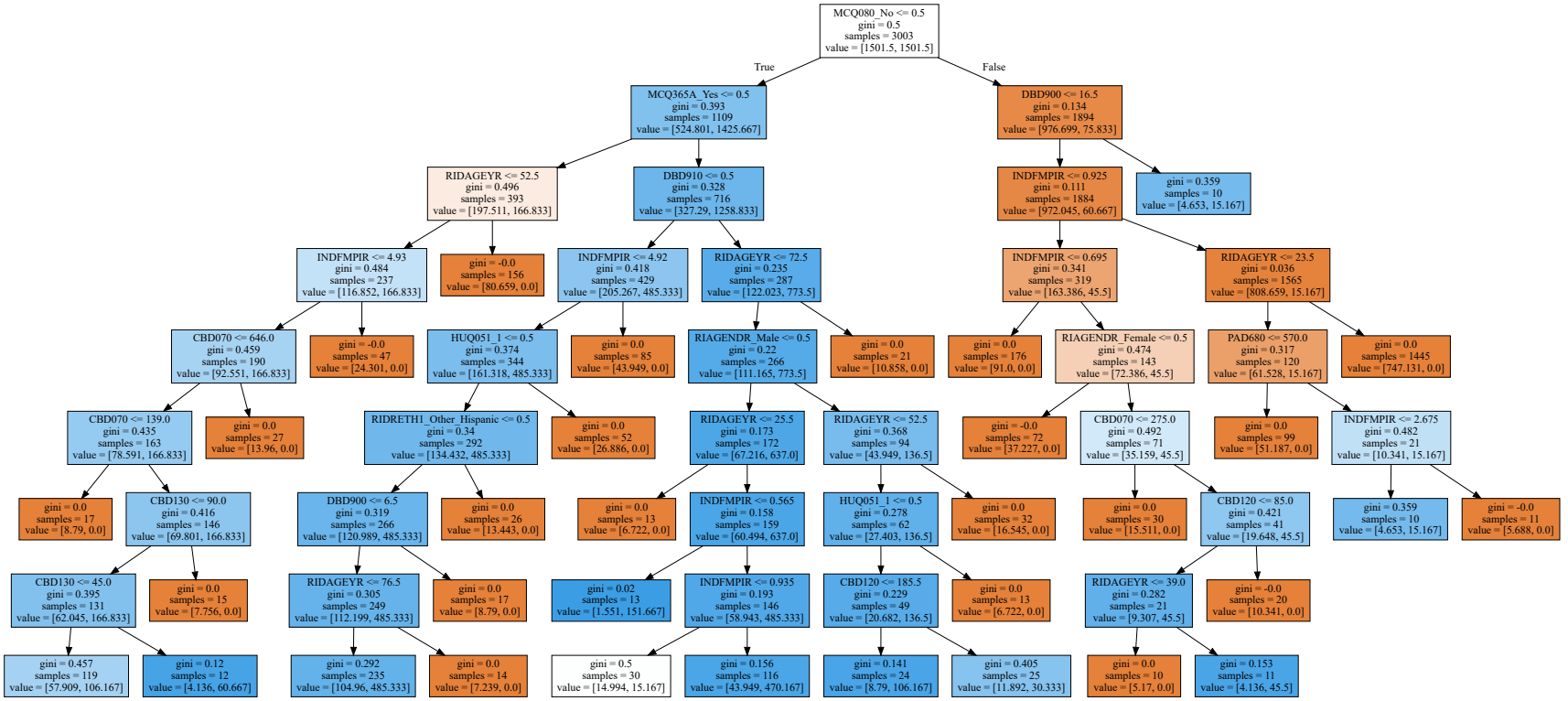
Out[187]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIAGENDR_Ma
0	0.105049	0.101082	0.030072	0.0	0.013356	0.014404	0.028229	0.0	0.018495	0.011018	0.014838	0.005768

Visualize the final tree.

In [188]:
```
export_graphviz(dt_model, out_file='tree_balanced.dot', feature_names=dt_features, filled=True)

with open("tree_balanced.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[188]:



Save the final tree visualization to a jpeg file.

In [189]:
```
system(dot -Tpng tree_balanced.dot -o tree_balanced.jpeg)
```

Out[189]: []

Decision Tree Classification With All Features and Balanced Weighting

Call the grid search and model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting). The parameter values that are used below are the result of several modelling attempts with different value ranges. The values below resulted in the best model (has the best accuracy and model complexity trade-off, is not overfit).

In [190]:
```
cv = 10
dt_all = tree.DecisionTreeClassifier(class_weight  = "balanced")

temp_leaf = np.arange(10, 201, 10)
temp_depth = np.arange(1, 9, 1)

parameters = {
    'criterion': ['entropy','gini'],
    'max_depth': temp_depth,
    'min_samples_leaf': temp_leaf,
    'min_samples_split': temp_leaf
}

print 'Parameters Used:'
print parameters
print '\n'

# Perform grid search with parameter dictionary passed in
opt_model = perform_grid_search(dt_all, data_train_np, target_train_np, data_test_np, target_test_np, parameters, cv)

# Perform grid search with parameter dictionary passed in
dt_model_all = model_data(opt_model, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
Parameters Used:
{'min_samples_split': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200]), 'criterion': ['entropy', 'gini'], 'max_depth': array([1, 2, 3, 4, 5, 6, 7, 8]),
'min_samples_leaf': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200])}


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 6400 candidates, totalling 64000 fits

[Parallel(n_jobs=1)]: Done 64000 out of 64000 | elapsed: 13.1min finished
```

```
Wall time: 13min 8s


Grid Search Optimal Parameters: {'min_samples_split': 10, 'criterion': 'gini', 'max_depth': 8, 'min_samples_leaf': 10}
Grid Search Optimal Parameter Score: 0.815517815518


Final Model Parameter Settings:
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=8,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=10, min_samples_split=10,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.846079869038
10 Fold Cross Validation Testing Accuracy:
0.820181616833


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.84015984016

Classification Report:
             precision    recall  f1-score   support

          0       1.00      0.83      0.91      2904
          1       0.17      1.00      0.29        99

avg / total       0.97      0.84      0.89      3003


Confussion Matrix:
[[2424  480]
 [   0   99]]



---------------------------------- Testing Data Performance ----------------------------------
```

```
Final Model Testing Set Results
-------------------------------
Accuracy:
0.832223701731

Classification Report:
             precision    recall  f1-score   support

          0       0.99      0.83      0.91       723
          1       0.15      0.79      0.26        28

avg / total       0.96      0.83      0.88       751


Confussion Matrix:
[[603 120]
 [  6  22]]
```

Compute the number of nodes present in the tree.

In [191]:
```
treeObjAll = dt_model_all.tree_
print 'Number of nodes in the tree:'
print treeObjAll.node_count
```

```
Number of nodes in the tree:
61
```

Retreive and save the feature importances of the final model to a csv file.

In [192]:
```
feature_imp = dt_model_all.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = spdsht_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\DT Balanced Full Training Importan
ces.csv')
feature_imp_df
```

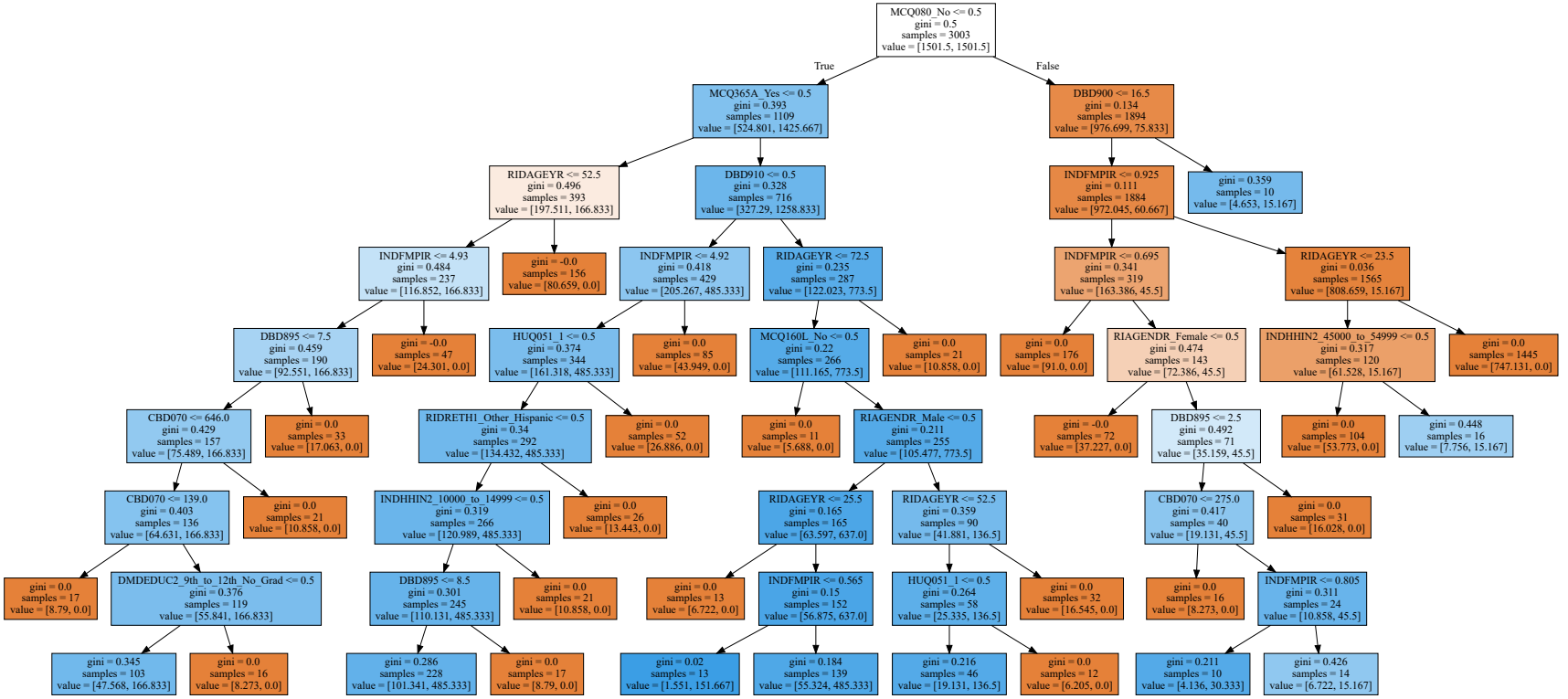
Out[192]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIAGEN
0	0.089791	0.088329	0.027227	0.0	0.0	0.0	0.036422	0.017811	0.0	0.01854	0.0	0.014874	0.00541

Visualize the final tree.

In [193]:
```
export_graphviz(dt_model_all, out_file='tree_all_balanced.dot', feature_names=spdsht_features, filled=True)

with open("tree_all_balanced.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[193]:



Save the final tree visualization to a jpeg file.

In [194]:
```
system(dot -Tpng tree_all_balanced.dot -o tree_all_balanced.jpeg)
```

Out[194]: []

---------- Ensemble Classification ----------

Random Forest Classification

Call the grid search and model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

Note that this model was run with all of the features (no feature selection) because the ensemble method will randomly choose subsets of features at each node. Since there was not a large difference in performance between feature selection and none with decision trees above, I opted for sending in all the features and allowing the algorithm to choose which to use.

I originally ran this model with max_depth, min_samples_leaf and min_samples_split specified similarly to the rest of the decision tree models, but the code took an extremely long time to run (had to kill it while it was still running over 20 hrs. because my laptop was having issues). I then opted for just leaving the criterion option and the n_estimators option. I removed the max depth because of the explanation of how the random forest algorithm already tends toward modelling several short trees.

In [195]:
```
cv = 10
rf_all = RandomForestClassifier(class_weight  = "balanced")

temp_n_estimators = np.arange(5, 101, 5)

parameters = {
    'criterion': ['entropy','gini'],
    'n_estimators': temp_n_estimators
}

print 'Parameters Used:'
print parameters
print '\n'

# Perform grid search with parameter dictionary passed in
opt_model = perform_grid_search(rf_all, data_train_np, target_train_np, data_test_np, target_test_np, parameters, cv)

# Perform grid search with parameter dictionary passed in
rf_model_all = model_data(opt_model, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
Parameters Used:
{'n_estimators': array([  5,  10,  15,  20,  25,  30,  35,  40,  45,  50,  55,  60,  65,
        70,  75,  80,  85,  90,  95, 100]), 'criterion': ['entropy', 'gini']}


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 40 candidates, totalling 400 fits

[Parallel(n_jobs=1)]: Done 400 out of 400 | elapsed:  1.3min finished
```

```
Wall time: 1min 17s


Grid Search Optimal Parameters: {'n_estimators': 20, 'criterion': 'gini'}
Grid Search Optimal Parameter Score: 0.968031968032


Final Model Parameter Settings:
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=20, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
0.997151121667
10 Fold Cross Validation Testing Accuracy:
0.966696566999


Final Model Full Training Set Results
-------------------------------------
Accuracy:
0.998667998668

Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      2904
          1       1.00      0.96      0.98        99

avg / total       1.00      1.00      1.00      3003


Confussion Matrix:
[[2904    0]
 [   4   95]]
```

```
---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.962716378162

Classification Report:
             precision    recall  f1-score   support

          0       0.96      1.00      0.98       723
          1       0.00      0.00      0.00        28

avg / total       0.93      0.96      0.94       751


Confussion Matrix:
[[723   0]
 [ 28   0]]


C:\Users\Kari\Anaconda3\envs\Python27\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Retrieve and save the feature importances to a csv file.

In [196]:
```
feature_imp = rf_model_all.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = spdsht_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\RF Balanced Full Training Importan
ces.csv')
feature_imp_df
```

Out[196]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIA
0	0.043795	0.053493	0.031891	0.015559	0.030237	0.028535	0.02247	0.024459	0.015684	0.030845	0.033186	0.012401	0.01

Adaboost Decision Tree Classification Without Balanced Weights

Call the grid search and model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

Note that this model was run with all of the features (no feature selection) because the ensemble method will randomly choose subsets of features at each node. Since there was not a large difference in performance between feature selection and none with decision trees above, I opted for sending in all the features and allowing the algorithm to choose which to use.

There was no easy way to have the grid search algorithm change the decision tree parameters because they are imbedded within the model passed into the adaboost model (grid search will only change the adaboost model parameters).

In [197]:
```
cv = 10
ab_all = AdaBoostClassifier(DecisionTreeClassifier())

temp_n_estimators = np.arange(5, 101, 5)

parameters = {
    'n_estimators': temp_n_estimators
}

print 'Parameters Used:'
print parameters
print '\n'

# Perform grid search with parameter dictionary passed in
opt_model = perform_grid_search(ab_all, data_train_np, target_train_np, data_test_np, target_test_np, parameters, cv)

# Perform grid search with parameter dictionary passed in
ab_model_all = model_data(opt_model, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
Parameters Used:
{'n_estimators': array([  5,  10,  15,  20,  25,  30,  35,  40,  45,  50,  55,  60,  65,
        70,  75,  80,  85,  90,  95, 100])}


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 20 candidates, totalling 200 fits

[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed:    5.4s finished
```

```
Wall time: 5.56 s


Grid Search Optimal Parameters: {'n_estimators': 95}
Grid Search Optimal Parameter Score: 0.942390942391


Final Model Parameter Settings:
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=1.0, n_estimators=95, random_state=None)


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
1.0
10 Fold Cross Validation Testing Accuracy:
0.936058693245


Final Model Full Training Set Results
-------------------------------------
Accuracy:
1.0

Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      2904
          1       1.00      1.00      1.00        99

avg / total       1.00      1.00      1.00      3003


Confussion Matrix:
[[2904    0]
 [   0   99]]
```

```
--------------------------------- Testing Data Performance ---------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.922769640479

Classification Report:
             precision    recall  f1-score   support

          0       0.96      0.96      0.96       723
          1       0.03      0.04      0.03        28

avg / total       0.93      0.92      0.93       751


Confussion Matrix:
[[692  31]
 [ 27   1]]
```

Retrieve and save the feature importances to a csv file.

In [198]:
```
feature_imp = ab_model_all.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = spdsht_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\AB DT Full Training Importances.csv')
feature_imp_df
```

Out[198]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIA
0	0.056895	0.063975	0.086657	0.040262	0.077533	0.014295	0.03908	0.048147	0.046395	0.032698	0.058175	0.006964	0.01

Adaboost Decision Tree Classification With Balanced Weights

Call the grid search and model data functions with the full training set (with no feature selection performed). I chose to use a cross validation fold number of 10 because the training set has 3,003 instances (which is enough to support 10 fold splitting).

Note that this model was run with all of the features (no feature selection) because the ensemble method will randomly choose subsets of features at each node. Since there was not a large difference in performance between feature selection and none with decision trees above, I opted for sending in all the features and allowing the algorithm to choose which to use.

There was no easy way to have the grid search algorithm change the decision tree parameters because they are imbedded within the model passed into the adaboost model (grid search will only change the adaboost model parameters).

In [199]:
```
cv = 10
ab_all = AdaBoostClassifier(DecisionTreeClassifier(class_weight  = "balanced"))

temp_n_estimators = np.arange(5, 101, 5)

parameters = {
    'n_estimators': temp_n_estimators
}

print 'Parameters Used:'
print parameters
print '\n'

# Perform grid search with parameter dictionary passed in
opt_model = perform_grid_search(ab_all, data_train_np, target_train_np, data_test_np, target_test_np, parameters, cv)

# Perform grid search with parameter dictionary passed in
ab_model_all = model_data(opt_model, data_train_np, target_train_np, data_test_np, target_test_np, cv)
```

```
Parameters Used:
{'n_estimators': array([  5,  10,  15,  20,  25,  30,  35,  40,  45,  50,  55,  60,  65,
        70,  75,  80,  85,  90,  95, 100])}


---------------------------------- Parameter Grid Search ----------------------------------
Fitting 10 folds for each of 20 candidates, totalling 200 fits

[Parallel(n_jobs=1)]: Done 200 out of 200 | elapsed:    5.6s finished
```

```
Wall time: 5.74 s


Grid Search Optimal Parameters: {'n_estimators': 40}
Grid Search Optimal Parameter Score: 0.942723942724


Final Model Parameter Settings:
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=1.0, n_estimators=40, random_state=None)


---------------------------------- Training Data Performance ----------------------------------


Final Model Training Set Cross Validation Results
-------------------------------------------------
10 Fold Cross Validation Training Accuracy:
1.0
10 Fold Cross Validation Testing Accuracy:
0.939057585825


Final Model Full Training Set Results
-------------------------------------
Accuracy:
1.0

Classification Report:
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      2904
          1       1.00      1.00      1.00        99

avg / total       1.00      1.00      1.00      3003


Confussion Matrix:
[[2904    0]
 [   0   99]]
```

```
---------------------------------- Testing Data Performance ----------------------------------


Final Model Testing Set Results
-------------------------------
Accuracy:
0.946737683089

Classification Report:
             precision    recall  f1-score   support

          0       0.97      0.97      0.97       723
          1       0.30      0.32      0.31        28

avg / total       0.95      0.95      0.95       751


Confussion Matrix:
[[702  21]
 [ 19   9]]
```

Retrieve and save the feature importances to a csv file.

In [200]:
```
feature_imp = ab_model_all.feature_importances_
feature_imp_arr = np.zeros((1, len(feature_imp)))
feature_imp_arr[0] = feature_imp

feature_imp_df = pd.DataFrame(feature_imp_arr, columns = spdsht_features)
feature_imp_df.to_csv('C:\\DePaul Coursework\\Fall 2017 CSC 478\\Final Project\\Outputs\\AB DT Balanced Full Training Impor
tances.csv')
feature_imp_df
```

Out[200]:

	RIDAGEYR	INDFMPIR	CBD070	CBD090	CBD120	CBD130	DBD895	DBD900	DBD905	DBD910	PAD680	RIAGENDR_Female	RIA
0	0.088829	0.094439	0.024067	0.003315	0.005687	0.033323	0.029648	0.021162	0.008489	0.017526	0.009787	0.004605	0.0