Part 3: Data Validation and Pre-Processing (40 points)

In this part of the assignment, we'll cover all of the data munging that's a pre-requisite for data mining. First, we'll look at missing values. Next, we'll look at data normalization and feature aggregation. Last, we'll cover some feature selection and dimensionality reduction techniques. For these exercises, we'll use some data from the Cleveland Clinic used to predict heart disesase. You can read more about the dataset here (http://archive.ics.uci.edu/ml/datasets/Heart+Disease) and even download similar data from three other clinics. To make life a little easier, I'll list the attributes and their types here.

- Age: ratio
- Sex: nominal
- ChestPainType: nominal
- RestingBP: ratio
- Cholesterol: ratio
- FastingBloodSugar: nominal
- RestingECG: nominal
- MaxHeartRate: ratio
- ExerciseInducedAngine: nominal
- STExerciseDepression: ratio
- STExercisePeakSlope: ordinal
- FlouroscopyVessels: ratio
- Thalassemia: nominal
- Heart Disease: nominal (label)

In [2]:
```
## Preliminaries

#Show plots in the notebook
%matplotlib inline

# To start we import some prerequisites
from sklearn import datasets, preprocessing
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib2

#let's load the data
heart_data = urllib2.urlopen("http://archive.ics.uci.edu/ml/machine-le
heart = pd.read_csv(heart_data, quotechar='"', skipinitialspace=True,
```

In [3]:
```
# To make things a bit more difficult, let's break the dataset even mo
# We'll replace a nominal attribute with an unknown value
heart.loc[::10,'Sex']=2
# And put some inconsistent values for blood pressure
heart.loc[::7,'RestingBP']=-100
```

Missing Values

Real data often has missing or erroneous values. The pandas documentation has a great writeup (http://pandas.pydata.org/pandas-docs/stable/missing_data.html) on dealing with missing data. In the dataset above, there are some naturally missing values, as well as erroneous values for the patient's sex and resting blood pressure. Let's try to deal with these data issues. First, we'll see how bad the data quality issues are. We'll use some tools to determine the scope of the missing and inconsistent values.

First let's look at the size of our dataset

In [4]:
```
print heart.shape
```

```
(303, 14)
```

However some of these 303 instances may have missing values. Let's see if we can find any

In [5]:
```
print heart.isnull().sum()
```

```
Age                        0
Sex                        0
ChestPainType              0
RestingBP                  0
Cholesterol                0
FastingBloodSugar          0
RestingECG                 0
MaxHeartRate               0
ExerciseInducedAngina      0
STExerciseDepression       0
STExercisePeakSlope        0
FlouroscopyVessels         4
Thalassemia                2
HeartDisease               0
dtype: int64
```

It looks like there are four missing values for FlouroscopyVessels and two missing values for Thalassemia. That's not too bad, but maybe there are some other data issues to worry about...

Let's look at the Sex field - the dataset description states that these values are coded as 0 for female and 1 for male, which means that values other than 0 or 1 are invalid. Are there any such values?

In [6]:
```
print len(heart[(heart['Sex'] > 1) | (heart['Sex'] < 0)])
```

```
31
```

There are 31 invalid values in the dataset! Let's flag these inconsistent values by setting them to the value Python uses to indicate erroneous numerical values. That value is called NotaNumber (NaN). Let's set those invalid values to NaN.

In [7]:
```
heart.loc[(heart['Sex'] > 1) | (heart['Sex'] < 0),'Sex'] = np.nan

#As a sanity check, let's make sure those values were updated
print heart[(heart['Sex'] > 1) | (heart['Sex'] < 0)].head()
```

```
Empty DataFrame
Columns: [Age, Sex, ChestPainType, RestingBP, Cholesterol, FastingBl
oodSugar, RestingECG, MaxHeartRate, ExerciseInducedAngina, STExercis
eDepression, STExercisePeakSlope, FlouroscopyVessels, Thalassemia, H
eartDisease]
Index: []
```

In [8]:
```
#Now let's remove these and any other missing values from the dataset
heart_cleaned = heart.dropna()
print heart_cleaned.shape
```

```
(266, 14)
```

Question 1: Missing Values (20 points)

After all that work, you'll still need to fix the `heart_cleaned` data a little bit.

1. The RestingBP attribute also has some data quality issues - sometimes it's negative. Replace these values with NaN
2. Replace the missing values (NaN) with:

   - the mean value
   - the median value

   In each case, compute descriptive statistics and report whether the effect on mean, median, and standard deviation.
3. Replace the missing values with means/medians computed for each sex (0 or 1) instead of the global mean/median. Compute the descriptive statistics for the dataset. How do the global mean, median and standard deviation change?

See cell bellow, the formatting was bad in markup langauge so I left it as a comment

See cell bellow, the formatting was bad in markup langauge so I left it as a comment

In [9]:

```
# 1
print len(heart_cleaned[heart_cleaned['RestingBP'] < 0])
heart_cleaned.loc[heart_cleaned['RestingBP'] < 0, 'RestingBP'] = np.na
print len(heart_cleaned[heart_cleaned['RestingBP'] < 0])

# 2
# print heart_cleaned.mean() # 131.34
# heart_cleaned.fillna(heart_cleaned.mean(), inplace=True)
# print heart_cleaned.describe()
# #              Age         Sex  ChestPainType   RestingBP  Cholester
# count  266.000000  266.000000     266.000000  266.000000  266.00000
# mean    54.571429    0.680451       3.169173  131.340611  248.30451
# std      9.033093    0.467181       0.954408   16.064426   52.84531
# min     29.000000    0.000000       1.000000   94.000000  126.00000
# 25%     48.000000    0.000000       3.000000  120.000000  212.00000
# 50%     56.000000    1.000000       3.000000  130.670306  243.00000
# 75%     61.000000    1.000000       4.000000  140.000000  277.75000
# max     77.000000    1.000000       4.000000  192.000000  564.00000

#         FastingBloodSugar  RestingECG  MaxHeartRate  ExerciseInducedA
# count          266.000000  266.000000    266.000000             266.0
# mean             0.146617    1.000000    149.458647               0.3
# std              0.354390    0.994324     23.517159               0.4
# min              0.000000    0.000000     71.000000               0.0
# 25%              0.000000    0.000000    132.000000               0.0
# 50%              0.000000    1.000000    154.000000               0.0
# 75%              0.000000    2.000000    166.000000               1.0
# max              1.000000    2.000000    202.000000               1.0

#         STExerciseDepression  STExercisePeakSlope  FlouroscopyVessels
# count             266.000000           266.000000          266.000000
# mean                1.075188             1.601504            0.706767
# std                 1.187126             0.625735            0.953895
# min                 0.000000             1.000000            0.000000
# 25%                 0.000000             1.000000            0.000000
# 50%                 0.800000             2.000000            0.000000
# 75%                 1.750000             2.000000            1.000000
# max                 6.200000             3.000000            3.000000

#         Thalassemia  HeartDisease
# count    266.000000    266.000000
# mean       4.703008      0.969925
# std        1.938233      1.234350
# min        3.000000      0.000000
# 25%        3.000000      0.000000
# 50%        3.000000      0.000000
# 75%        7.000000      2.000000
# max        7.000000      4.000000

# The effect is:
# mean - stayed the same at 131.34
# median - increased from 130 -> 130.670306
# std - decreased from 17.318917 -> 16.064426
# Part 2 b) Replace the missing values with the median
# # print heart_cleaned.median() # 130
# heart_cleaned.fillna(heart_cleaned.median(), inplace=True)
# print heart_cleaned.describe()
#               Age         Sex  ChestPainType   RestingBP  Cholesterol
# count  266.000000  266.000000     266.000000  266.000000   266.00000
# mean    54.571429    0.680451       3.169173  131.154135   248.30451
# std      9.033093    0.467181       0.954408   16.071148    52.84531
# min     29.000000    0.000000       1.000000   94.000000   126.00000
# 25%     48.000000    0.000000       3.000000  120.000000   212.00000
# 50%     56.000000    1.000000       3.000000  130.000000   243.00000
# 75%     61.000000    1.000000       4.000000  140.000000   277.75000
# max     77.000000    1.000000       4.000000  192.000000   564.00000

#        FastingBloodSugar  RestingECG  MaxHeartRate  ExerciseInducedA
# count         266.000000  266.000000    266.000000             266.0
# mean            0.146617    1.000000    149.458647               0.3
# std             0.354390    0.994324     23.517159               0.4
# min             0.000000    0.000000     71.000000               0.0
# 25%             0.000000    0.000000    132.000000               0.0
# 50%             0.000000    1.000000    154.000000               0.0
# 75%             0.000000    2.000000    166.000000               1.0
# max             1.000000    2.000000    202.000000               1.0

#        STExerciseDepression  STExercisePeakSlope  FlouroscopyVessels
# count            266.000000           266.000000          266.000000
# mean               1.075188             1.601504            0.706767
# std                1.187126             0.625735            0.953895
# min                0.000000             1.000000            0.000000
# 25%                0.000000             1.000000            0.000000
# 50%                0.800000             2.000000            0.000000
# 75%                1.750000             2.000000            1.000000
# max                6.200000             3.000000            3.000000

#        Thalassemia  HeartDisease
# count   266.000000    266.000000
# mean      4.703008      0.969925
# std       1.938233      1.234350
# min       3.000000      0.000000
# 25%       3.000000      0.000000
# 50%       3.000000      0.000000
# 75%       7.000000      2.000000
# max       7.000000      4.000000

# The effect is:
# mean - decreased from 131.340611 -> 131.154135
# median - stayed the same at 130
# std - decreased from 17.318917 -> 16.071148



# 3
heart_cleaned_female = heart_cleaned[heart_cleaned.Sex == 0]
heart_cleaned_male = heart_cleaned[heart_cleaned.Sex == 1]
```

```
# print heart_cleaned_female.mean() # 132.2
# print heart_cleaned_male.mean() # 130.96 ~ 131
# print heart_cleaned.describe()
# heart_cleaned.loc[(heart_cleaned['Sex'] == 0) & (heart_cleaned['Rest
# heart_cleaned.loc[(heart_cleaned['Sex'] == 1) & (heart_cleaned['Rest
# print heart_cleaned.describe()
# Descriptive stats:
#          RestingBP
# count  266.000000
# mean   131.360902
# std    16.066014
# min    94.000000
# 25%    120.000000
# 50%    130.500000
# 75%    140.000000
# max    192.000000

# # The effect is:
# # mean - increased 131.34 -> 131.36
# # median - increased from 130 -> 130.5
# # std - decreased from 17.32 -> 16.1


# b) Replace the missing values with means/medians computed for each s
# # print heart_cleaned_female.median() # 130
# # print heart_cleaned_male.median() # 130
# print heart_cleaned.describe()
heart_cleaned.loc[(heart_cleaned['Sex'] == 0) & (heart_cleaned['Restin
heart_cleaned.loc[(heart_cleaned['Sex'] == 1) & (heart_cleaned['Restin
print heart_cleaned.describe()
# Descriptive stats:
#         RestingBP
# count   266.000000
# mean    131.154135
# std     16.071148
# min     94.000000
# 25%     120.000000
# 50%     130.000000
# 75%     140.000000
# max     192.000000

# # The effect is:
# # mean - decreased from 131.34 -> 131.15
# # median - stayed the same 130 -> 130
# # std - decreased from 17.32 -> 16.07
```

```
37
0
              Age         Sex  ChestPainType   RestingBP  Cholestero
l  \
count  266.000000  266.000000     266.000000  266.000000   266.00000
0
```

```
mean     54.571429    0.680451       3.169173  131.154135   248.30451
1
std       9.033093    0.467181       0.954408   16.071148    52.84531
8
min      29.000000    0.000000       1.000000   94.000000   126.00000
0
25%      48.000000    0.000000       3.000000  120.000000   212.00000
0
50%      56.000000    1.000000       3.000000  130.000000   243.00000
0
75%      61.000000    1.000000       4.000000  140.000000   277.75000
0
max      77.000000    1.000000       4.000000  192.000000   564.00000
0

       FastingBloodSugar  RestingECG  MaxHeartRate  ExerciseInducedA
ngina  \
count         266.000000  266.000000    266.000000             266.0
00000
mean            0.146617    1.000000    149.458647               0.3
19549
std             0.354390    0.994324     23.517159               0.4
67181
min             0.000000    0.000000     71.000000               0.0
00000
25%             0.000000    0.000000    132.000000               0.0
00000
50%             0.000000    1.000000    154.000000               0.0
00000
75%             0.000000    2.000000    166.000000               1.0
00000
max             1.000000    2.000000    202.000000               1.0
00000

       STExerciseDepression  STExercisePeakSlope  FlouroscopyVessels
\
count            266.000000           266.000000          266.000000
mean               1.075188             1.601504            0.706767
std                1.187126             0.625735            0.953895
min                0.000000             1.000000            0.000000
25%                0.000000             1.000000            0.000000
50%                0.800000             2.000000            0.000000
75%                1.750000             2.000000            1.000000
max                6.200000             3.000000            3.000000

       Thalassemia  HeartDisease
count   266.000000    266.000000

mean      4.703008      0.969925
std       1.938233      1.234350
min       3.000000      0.000000
25%       3.000000      0.000000
50%       3.000000      0.000000
75%       7.000000      2.000000
max       7.000000      4.000000
max       7.000000      4.000000

/Users/viktorjankov/anaconda/lib/python2.7/site-packages/pandas/cor
e/indexing.py:426: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy (http://panda
s.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-c
opy)
  self.obj[item] = s
```

Transforming and Generating Features

In class, we discussed various ways to transform features. There are simple approaches like normalizing features and more complex approaches that involve generating new features from the ones we already have. Let's look at an example of each.

First, let's take the ratio features and normalize them into a function of the data range. To do this, we'll use some tools from scikit-learn. Now would be a great time to familiarize yourself with the data preprocessing tools (http://scikit-learn.org/stable/modules/preprocessing.html) available in scikit-learn.

In [10]:
```
#Let's re-load a clean dataset, so our results don't depend on your an
heart_data = urllib2.urlopen("http://archive.ics.uci.edu/ml/machine-le
heart = pd.read_csv(heart_data, quotechar='"', skipinitialspace=True,
heart=heart.dropna()

#Now let's look at a snippet of the original data
heart_ratio =  heart.loc[:,['Age','RestingBP','Cholesterol','MaxHeartR
print heart_ratio.head()

#The MinMaxScaler scales each value by subtracting the minimum and the
# or scaled_value = (value - min)/(max-min)
scaler = preprocessing.MinMaxScaler()

#Scale the heart data
heart_ratio_scaled_values = scaler.fit_transform(heart_ratio.values);

#Put the result back into a dataframe
heart_ratio_scaled = pd.DataFrame(heart_ratio_scaled_values, columns =
print heart_ratio_scaled.head()

#heart_ratio_scaled = pd.DataFrame(preprocessing.MinMaxScaler().fit_tr
```

```
   Age  RestingBP  Cholesterol  MaxHeartRate  STExercisePeakSlope  \
0   63        145          233           150                    3
1   67        160          286           108                    2
2   67        120          229           129                    2
3   37        130          250           187                    3
4   41        130          204           172                    1

   FlouroscopyVessels
0                   0
1                   3
2                   2
3                   0
4                   0
        Age  RestingBP  Cholesterol  MaxHeartRate  STExercisePeakSlo
pe  \
0  0.708333   0.481132     0.244292      0.603053
1.0
1  0.791667   0.622642     0.365297      0.282443
0.5
2  0.791667   0.245283     0.235160      0.442748
0.5
3  0.166667   0.339623     0.283105      0.885496
1.0
4  0.250000   0.339623     0.178082      0.770992
0.0

   FlouroscopyVessels
0            0.000000
1            1.000000
2            0.666667
3            0.000000
```

```
4            0.000000
```

Normalizing features is a good idea, since some algorithms won't understand that age and cholesterol are equally important when the cholesterol can be an order of magnitude larger than the age. Transforming features to the [0,1] range makes them all equally important. Note that we did not normalize the nominal features, but there would be no harm in doing so since the nominal properties (each value has a distinct identity) would be preserved.

Another common task is generating new features from your existing data. Let's look at a simple way of generating new features: polynomial features.

In many cases, we may want to look at how one feature interacts with another. Scikit-learn provides a function to automatically generate all of these features, `PolynomialFeatures`. Read the documentation (http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.p to understand what it does. We'll apply it to our dataset below.

In [11]:
```
the label from the features
labeled = heart[['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholester

he values, then generate polynomial features
lynomial = pd.DataFrame(preprocessing.PolynomialFeatures().fit_transfor
lynomial_values = heart_polynomial.values
```

Question 2: Feature Transformations (10 points)

1. The MinMaxScaler is pretty limited. The StandardScaler is a bit more powerful. Apply it to the ratio features in the heart dataset `heart_ratio` above. Compute descriptive statistics for the data. What are the mean and standard deviation for each column?
2. How many polynomial features are generated? Explain why this number of features was generated, giving a breakdown of features of each type.

See cell bellow, the formatting was bad in markup langauge so I left it as a comment

In [12]:
```
# 1
better_scaler = preprocessing.StandardScaler()
heart_ratio_better_scaler_values = better_scaler.fit_transform(heart_r
heart_ratio_better_scaled = pd.DataFrame(heart_ratio_better_scaler_val
print heart_ratio_better_scaled.describe()
#                  Age     RestingBP   Cholesterol  MaxHeartRate  \
# count  2.970000e+02  2.970000e+02  2.970000e+02  2.970000e+02
# mean  -1.237319e-16  4.810966e-16 -1.911116e-16  5.143660e-16
# std    1.001688e+00  1.001688e+00  1.001688e+00  1.001688e+00
# min   -2.827176e+00 -2.125634e+00 -2.337704e+00 -3.431849e+00
# 25%   -7.241238e-01 -6.594306e-01 -7.002541e-01 -7.247694e-01
# 50%    1.613719e-01 -9.550637e-02 -8.380217e-02  1.484822e-01
# 75%    7.148067e-01  4.684179e-01  5.519138e-01  7.160957e-01
# max    2.485798e+00  3.851964e+00  6.099981e+00  2.287949e+00

#        STExercisePeakSlope  FlouroscopyVessels
# count         2.970000e+02        2.970000e+02
# mean         -1.278439e-16        6.653862e-17
# std           1.001688e+00        1.001688e+00
# min          -9.765832e-01       -7.219761e-01
# 25%          -9.765832e-01       -7.219761e-01
# 50%           6.437811e-01       -7.219761e-01
# 75%           6.437811e-01        3.448244e-01
# max           2.264145e+00        2.478425e+00

# Mean and Std for each column
#                 Age     RestingBP   Cholesterol  MaxHeartRate   STExe
# mean  -1.237319e-16  4.810966e-16 -1.911116e-16  5.143660e-16   -1.2
# std    1.001688e+00  1.001688e+00  1.001688e+00  1.001688e+00   1.00

# 2
# There are 105 polynomial features created
# The number is generated using the formula
#    f(n) = 1 + 2n + nChoose2
# where n is the number of features
# the 1 is there by default.
# The 2n comes from the fact that each feature including it's root pow
# and the last term is all the possible combinations between the n fea
```

```
                 Age     RestingBP   Cholesterol  MaxHeartRate  \
count  2.970000e+02  2.970000e+02  2.970000e+02  2.970000e+02
mean  -1.237319e-16  4.810966e-16 -1.911116e-16  5.143660e-16
std    1.001688e+00  1.001688e+00  1.001688e+00  1.001688e+00
min   -2.827176e+00 -2.125634e+00 -2.337704e+00 -3.431849e+00
25%   -7.241238e-01 -6.594306e-01 -7.002541e-01 -7.247694e-01
50%    1.613719e-01 -9.550637e-02 -8.380217e-02  1.484822e-01
75%    7.148067e-01  4.684179e-01  5.519138e-01  7.160957e-01
max    2.485798e+00  3.851964e+00  6.099981e+00  2.287949e+00

       STExercisePeakSlope  FlouroscopyVessels
count         2.970000e+02        2.970000e+02
mean         -1.278439e-16        6.653862e-17
std           1.001688e+00        1.001688e+00
min          -9.765832e-01       -7.219761e-01
min          9.765832e-01        7.219761e-01
25%         -9.765832e-01       -7.219761e-01
50%          6.437811e-01       -7.219761e-01
75%          6.437811e-01        3.448244e-01
max          2.264145e+00        2.478425e+00
```

Feature Selection

Now that we've generated polynomial features, we may have too much data, some of it not very useful. There are two approaches to trimming this dataset into a more manageable size.

The first approach is feature selection, a method to choose the features in the data that are the most useful for analysis. Of course, when feature selection occurs before analysis, the selection approaches can't use the label information and rely on statistical information as a criteria for "useful" information. Many of the feature selection approaches in scikit-learn use some sort of model that fits the data to decide which features are most useful.

Question 3: Feature Selection (5 points)

Read the documentation on feature selection (http://scikit-learn.org/stable/modules/feature_selection.html#feature-selection) in scikit-learn before completing the exercises.

1. Apply the `VarianceThreshold` feature selection algorithm (with threshold 0.2) to the `heart_polynomial` dataset. How many features remain after feature selection is applied?

1. There remain 17 features

In [13]:
```
# 1
from sklearn.feature_selection import VarianceThreshold
heart_polynomial_prunned = VarianceThreshold(threshold=(.2 * (1 - .2)))
print heart_polynomial_prunned.shape
```

```
(297, 17)
```

Dimensionality Reduction

The other approach to reducing the number of features is to find a good way to squash many, many numbers into very few numbers. One very popular method to do this is principal components analysis (PCA) (https://en.wikipedia.org/wiki/Principal_component_analysis). The key idea is to combine the many, many attribute values together in a way that maximizes the differences between instances. The outcome is taking data in a high-dimensional space and projecting it into a low-dimensional space while keeping the points spread out.

The scikit-learn documentation (http://scikit-learn.org/stable/modules/decomposition.html#pca) shows how PCA works on the Iris dataset you were exploring in the previous part. We'll try using PCA with the heart data.

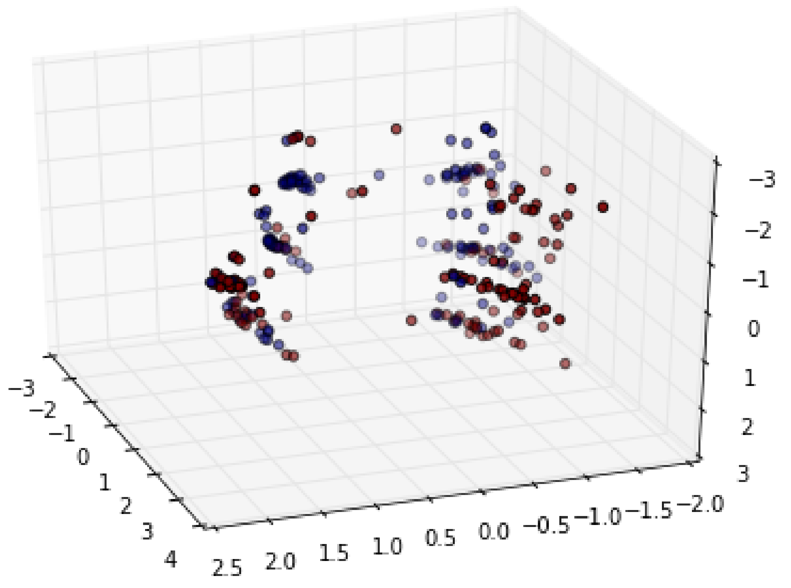
In [14]:
```
#load some modules to help
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA

#extract labels
heart_labels = heart['HeartDisease'].values
heart_labels[heart_labels > 0] = 1

#perform PCA
heart_polynomial_pca = PCA(n_components=3).fit_transform(heart_polynom

#Plot
ax = Axes3D(plt.figure(), elev=-150, azim=200)
ax.scatter(heart_polynomial_pca[:, 0], heart_polynomial_pca[:, 1], hea
```

Out[14]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x10ab85910>
```



Question 4: Dimensionality Reduction (5 points)

1. Repeat PCA using the `heart_unlabeled` data, projecting into three dimensions as in the above example. Does the projection do a better job of separating the points? Does the PCA projection produce a clearer delineation of the patients with normal heart function and those with heart disease?

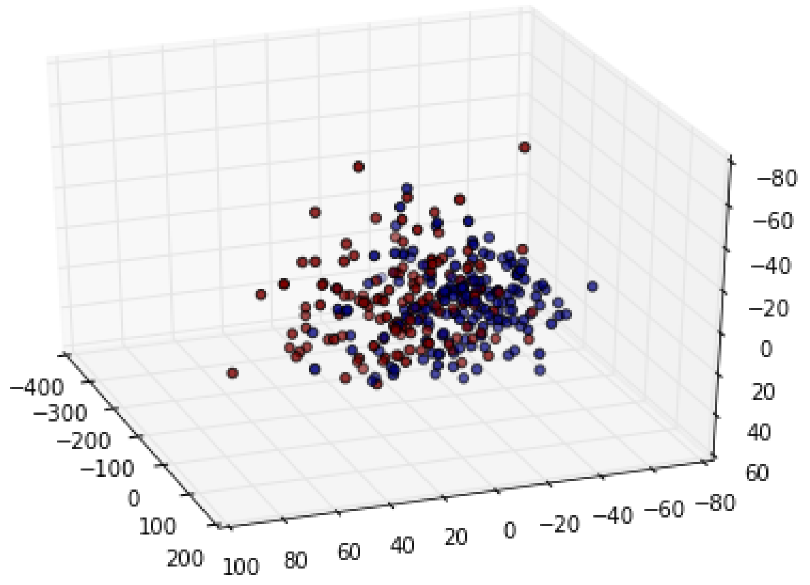
a) Does the projection do a better job of separating the points?
A: no the poits are much closely packed together

b) Does the PCA projection produce a clearer delineation of the patients with normal heart function and those with heart disease?
A: no, the PCA projection of the heart_polynomial data produced a much clearer delineation of the patients with normal heart function and those with the disease

In [15]:
```
# perform PCA
heart_unlabeled_pca = PCA(n_components=3).fit_transform(heart_unlabele

# Plot
ax = Axes3D(plt.figure(), elev=-150, azim=200)
ax.scatter(heart_unlabeled_pca[:,0], heart_unlabeled_pca[:,1], heart_u
```

Out[15]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x10adc4bd0>
```


−400
−300
−200
−100
0
100
200
100
80
60
40
20
0
−20
−40
−60
−80
−80
−60
−40
−20
0
20
40
60