SEIS 763: Machine Learning

Garth Mortensen, mort0052@stthomas.edu

2018.06.10

Assignment 2 - Linear Regression on Patient Dataset

Load the Data

1. Use Matlab command “load patients ” to load patient self evaluation dataset.

2. You can also load the data from “patients.csv” file.

Begin by importing packages.

Pandas is used for loading data

Numpy for ?

In [76]:
```
import pandas as pd
import numpy as np
```

In [77]:
```
# Load data
# genfromtext is a Numpy function. I prefer this explicit file path.
# Because there is header column, set header=0
patients = pd.read_csv("C:\\tmp\\patients.csv", header=0)

# Backup patients, just in case we need it later
patientsBackup = patients
```

Preview the Data

We want to preview the data to see what we'll be working with. This will display any missing values, as well.

In [78]:
```
# quick description of the data
patients.info()

# top 3 rows
patients.head(3)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
Age                       100 non-null int64
Diastolic                 100 non-null int64
Gender                    100 non-null object
Height                    100 non-null int64
LastName                  100 non-null object
Location                  100 non-null object
SelfAssessedHealthStatus  100 non-null object
Smoker                    100 non-null int64
Systolic                  100 non-null int64
Weight                    100 non-null int64
dtypes: int64(6), object(4)
memory usage: 7.9+ KB
```

Out[78]:

	Age	Diastolic	Gender	Height	LastName	Location	SelfAssessedHealthStatus	Smoker	Systolic	Weight
0	38	93	'Male'	71	'Smith'	'County General Hospital'	'Excellent'	1	124	176
1	43	77	'Male'	69	'Johnson'	'VA Hospital'	'Fair'	0	109	163
2	38	83	'Female'	64	'Williams'	'St. Mary's Medical Center'	'Good'	0	125	131

Each attribute contains 100 observations; there are no missing values. Therefore, we do not need to fill missing values with mean/mode, or drop any columns/rows.

In [79]:
```
# show a summary of the numerical attributes
patients.describe; # semi-colon to turn off echo (terminology?)
```

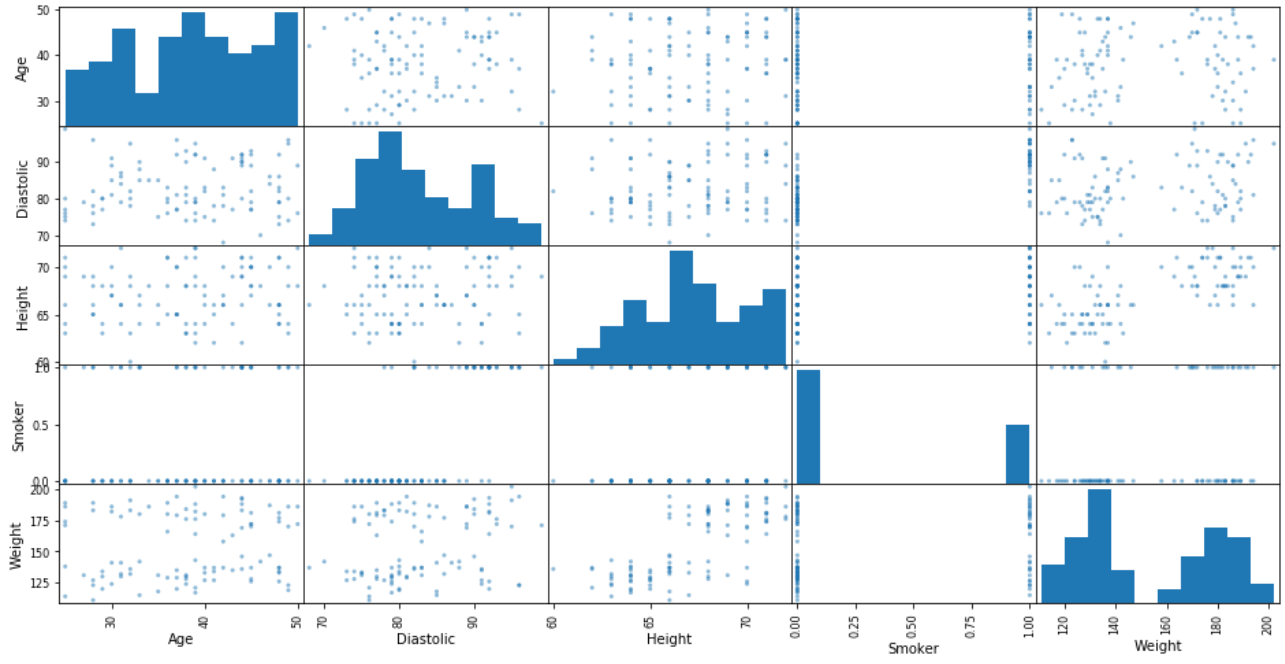
In [83]:
```
## Histogram visualization

## hist() relies on matplotlib
import matplotlib.pyplot as plot
#%matplotlib inline
#patients.hist(bins=20, figsize=(16,8))
#plot.show()
```

Cross Correlation Check

Previous function provided in Hands-On Machine Learning book was depreciated (https://stackoverflow.com/questions/44102532/problems-importing-pandas-plotting)

In [84]:
```
from pandas.plotting import scatter_matrix
#This is not the latest dataframe

attributes = ["Age", "Diastolic", "Height", "Smoker", "Weight"]
scatter_matrix(patients[attributes], figsize=(16, 8))

# the text output below is expected
# https://pandas.pydata.org/pandas-docs/stable/visualization.html
```

Out[84]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8F71CF8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8BB9358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7A56128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7A256D8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8B665F8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8B665C0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8B700B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8F39EF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8EA10B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8BEEB00>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7AB5C88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8ECE898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8EAABA8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8F16278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7B5B908>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7A8EF98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7A5D7F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7A34CF8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D99F43C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D7A1DA58>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8FA3128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8FC97B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D8FF4E48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D9025518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000073D904CBA8>]],
      dtype=object)
```



Nothing so interesting.

Data Adjustments

First split matrix into y (dependent) and x (independent)

Remember, Python is 0-offset! The "3rd" entry is at position 2.

patientsY = Diastolic

patientsX = Everything else *exluding LastName and Systolic*

Perfect, clear example on splitting y and x found here (http://www.stephacking.com/split-data-training-set-testing-set-python/) and here (http://www.stephacking.com/split-data-training-set-testing-set-python/)

Final solution on selecting multiple columns found here (https://pythonhow.com/accessing-dataframe-columns-rows-and-cells/)

The independent variables consist of numeric, categorical and binary datatypes. Each will be processed individually.

In [85]:
```
#split dependent variable and independent variables

# patientsY = patients[patients.columns[1]]
# patientsY = patients.iloc[:,1:1]
# The clearest is this:
patientsY = patients["Diastolic"]

# patientsX = patients["Age","Gender"]
# patientsX = patients.loc[:,"Age":"Gender"]
patientsX = patients[["Age", "Gender", "Height", "Location", "SelfAssessedHealthStatus", "Weigh
t"]]
patientsXNumeric = patients[["Age", "Height", "Weight"]]
# Smoker is not pulled with the other categorical data, so this next code line was added
patientsXBinary = patients[["Smoker"]]
```

Standardize the Data, or mean removal and variance scaling (http://scikit-learn.org/stable/modules/preprocessing.html)

> Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.
>
> Basically, take a matrix and change it so that its mean is equal to 0 and variance is 1

It matters in our case because Weight has values so much higher than Age. After fitting, our interpretation of the model will be influenced more by weight than age, since it has higher values.

We don't need to standardize the dependent y variable, so we split the matrix before standardizing the entire X matrix.

here (https://machinelearningmastery.com/rescaling-data-for-machine-learning-in-python-with-scikit-learn/) is the clearest example of normalizing and standardizing.

In [86]:
```
from sklearn import preprocessing

#Standardize
patientsXNumeric = patients[["Age", "Height", "Weight"]]

patientsXNumeric_scaled = preprocessing.scale(patientsXNumeric, axis=0)
```

Scaled data should have zero mean and unit variance.

In [87]:
```
#Mean
print("Mean: ", patientsXNumeric_scaled.mean())

#Std
print("Std: ", patientsXNumeric_scaled.std())
print("Length: ", len(patientsXNumeric_scaled))
```

```
Mean:  7.460698725481052e-16
Std:  1.0
Length:  100
```

Standardizing removed the header from the row. I need to fix this.

One hot encoding preperation

Perform one hot encoding, where 1 = hot, 0 = cold. Each feature value gets its own binary column.

Excellent tutorial here (https://www.ritchieng.com/machinelearning-one-hot-encoding/)

Another one here (https://www.datacamp.com/community/tutorials/categorical-data)

In [88]:
```
#Return only object datatypes (non-numeric here)
# categories = patientsX.select_dtypes(include=[object])

# As you will only be dealing with categorical features in this tutorial, it's better to filter
 them out.
# You can create a separate DataFrame consisting of only these features by running the following
 command.
# The method .copy() is used here so that any changes made in new DataFrame don't get reflected
 in the original one.
categoriesX = patientsX.select_dtypes(include=[object]).copy()

categoriesX.head()
```

Out[88]:

	Gender	Location	SelfAssessedHealthStatus
0	'Male'	'County General Hospital'	'Excellent'
1	'Male'	'VA Hospital'	'Fair'
2	'Female'	'St. Mary's Medical Center'	'Good'
3	'Female'	'VA Hospital'	'Fair'
4	'Female'	'County General Hospital'	'Good'

Let's also check the column-wise distribution of null values:

In [89]:
```
print(categoriesX.isnull().sum())
print(patientsXBinary.isnull().sum())
```

```
Gender                    0
Location                  0
SelfAssessedHealthStatus  0
dtype: int64
Smoker    0
dtype: int64
```

No missing values. Good!

Next, count distint cases of each category

In [90]:
```
# print(categoriesX["Location"].value_counts().count())
print("Gender: ", categoriesX["Gender"].value_counts().count())
print("Location: ", categoriesX["Location"].value_counts().count())
print("SelfAssessedHealthStatus: ", categoriesX["SelfAssessedHealthStatus"].value_counts().count
())
print("Smoker: ", patientsXBinary["Smoker"].value_counts().count())
```

```
Gender:  2
Location:  3
SelfAssessedHealthStatus:  4
Smoker:  2
```

There are not too many unique values that would complicate linear regression as a result of one-hot encoding.

One-Hot Encoding

As said **in this terrific one-hot tutorial (https://www.datacamp.com/community/tutorials/categorical-data)**:

> There are many libraries out there that support one-hot encoding but the simplest one is using pandas' .get_dummies() method.
>
> There are mainly three arguments important here, the first one is the DataFrame you want to encode on, second being the columns argument which lets you specify the columns you want to do encoding on, and third, the prefix argument which lets you specify the prefix for the new columns that will be created after encoding.

LastName is not to be included in the linear regression.

In [91]:
```
categoriesX_onehot = categoriesX.copy()
categoriesX_onehot = pd.get_dummies(categoriesX, columns=["Gender", "Location", "SelfAssessedHea
lthStatus"], prefix = ["Gender", "Location", "SelfAssessedHealthStatus"])
categoriesXBinary_onehot = pd.get_dummies(patientsXBinary, columns=["Smoker"], prefix = ["Smoke
r"])

# Return results
print(categoriesX_onehot.head());
print(categoriesXBinary_onehot.head());
```

```
   Gender_'Female'  Gender_'Male'  Location_'County General Hospital'  \
0                0              1                                   1
1                0              1                                   0
2                1              0                                   0
3                1              0                                   0
4                1              0                                   1

   Location_'St. Mary's Medical Center'  Location_'VA Hospital'  \
0                                     0                      0
1                                     0                      1
2                                     1                      0
3                                     0                      1
4                                     0                      0

   SelfAssessedHealthStatus_'Excellent'  SelfAssessedHealthStatus_'Fair'  \
0                                     1                                0
1                                     0                                1
2                                     0                                0
3                                     0                                1
4                                     0                                0

   SelfAssessedHealthStatus_'Good'  SelfAssessedHealthStatus_'Poor'
0                                0                               0
1                                0                               0
2                                1                               0
3                                0                               0
4                                1                               0
   Smoker_0  Smoker_1
0         0         1
1         1         0
2         1         0
3         1         0
4         1         0
```

Now that one-hot encoding has split the categorical attributes into many dummy attributes, they must be concatenated back together. This can be done via pandas' .concat() method. The axis argument is set to 1 as you want to merge on columns.

In [92]:
```
print("categoriesX_onehot is: ", type(categoriesX_onehot))

print(categoriesX_onehot.shape)

print("categoriesXBinary_onehot is: ", type(categoriesXBinary_onehot))

print(categoriesXBinary_onehot.shape)

print("patientsXNumeric_scaled is: ", type(patientsXNumeric_scaled))

print(patientsXNumeric_scaled.shape)
```

```
categoriesX_onehot is:  <class 'pandas.core.frame.DataFrame'>
(100, 9)
categoriesXBinary_onehot is:  <class 'pandas.core.frame.DataFrame'>
(100, 2)
patientsXNumeric_scaled is:  <class 'numpy.ndarray'>
(100, 3)
```

patientsXNumeric_scaled is an array. I used this SO post (https://stackoverflow.com/questions/20763012/creating-a-pandas-dataframe-from-a-numpy-array-how-do-i-specify-the-index-colum) to convert it to a dataframe.

In [93]:
```
patientsXNumeric_scaleddf = pd.DataFrame(patientsXNumeric_scaled)
```

In [94]:
```
print("patientsXNumeric_scaleddf is: ", type(patientsXNumeric_scaleddf))
```

```
patientsXNumeric_scaleddf is:  <class 'pandas.core.frame.DataFrame'>
```

In [95]:
```
print(patientsXNumeric_scaleddf.head())
```

```
          0         1         2
0 -0.039001  1.392506  0.832128
1  0.657450  0.683851  0.340416
2 -0.039001 -1.087784 -0.869952
3  0.239579 -0.024803 -0.794304
4  1.493193 -1.087784 -1.323841
```

Looks better, but it still needs column names.

In [96]:
```
patientsXNumeric_scaleddf.columns = ["Age", "Height", "Weight"]
```

Now we bring all the columns back together as one dataframe.

In [97]:
```
#Bring them back together
#patientsXAll = pd.concat([categoriesX_onehot, patientsXNumeric, categoriesXBinary_onehot], axis=1)
patientsXAll = pd.concat([categoriesX_onehot, patientsXNumeric_scaleddf, categoriesXBinary_onehot], axis=1)

#top 3 rows
print(patientsXAll.head(3))
```

```
   Gender_'Female'  Gender_'Male'  Location_'County General Hospital'  \
0                0              1                                   1
1                0              1                                   0
2                1              0                                   0

   Location_'St. Mary's Medical Center'  Location_'VA Hospital'  \
0                                     0                       0
1                                     0                       1
2                                     1                       0

   SelfAssessedHealthStatus_'Excellent'  SelfAssessedHealthStatus_'Fair'  \
0                                     1                               0
1                                     0                               1
2                                     0                               0

   SelfAssessedHealthStatus_'Good'  SelfAssessedHealthStatus_'Poor'       Age  \
0                                0                               0 -0.039001
1                                0                               0  0.657450
2                                1                               0 -0.039001

     Height    Weight  Smoker_0  Smoker_1
0  1.392506  0.832128         0         1
1  0.683851  0.340416         1         0
2 -1.087784 -0.869952         1         0
```

Above, the numeric, one-hot encoded categorical and binary columns have been concatenated into one dataframe.

Binning/Aggregating

None of the features (e.g. Age) require binning/aggregating.

Build a Linear Regression Model

3. Use variables Age, Gender, Height, Weight, Smoker, Location, SelfAssessedHealthStatus to build a linear regression model to predict the systolic blood pressure.

That does not include Diastolic or LastName in the prediction.

In this assignment, there is no need to split the dataset into training and testing, or training, validation and testing, but if you wanted to, this (http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py) is an incredibly clear example on that.

In [98]:
```
#mdl = fitlm(patientsXAll, patientsY)

import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
trainedmodel = regr.fit(patientsXAll, patientsY)
```

Interpretation

4. What are the regression coefficients (thetas)?

In [99]:
```
# yint (theta0)
yint = regr.intercept_
print("Y intercept: \n", yint, "\n")

# The coefficients (theta1)
coefficients = regr.coef_
print("Coefficients: \n", coefficients)
```

```
Y intercept:
 84.6055749682606

Coefficients:
 [-7.35647214e-01  7.35647214e-01  1.64575586e-01 -1.67794419e-01
  3.21883325e-03  6.29458102e-01  2.31924718e-01 -4.15324079e-01
 -4.46058740e-01  2.15729927e-01 -5.92449588e-01  1.76272116e-01
 -5.18796653e+00  5.18796653e+00]
```

5. How do you interpret those numbers?

In [100]:
```
print("Number of coefficients: ", len(coefficients))
print("Number of columns: ", len(patientsXAll.columns))
```

```
Number of coefficients:  14
Number of columns:  14
```

Just wanted to quickly check to see if the coefficient count is the same as my attribute count.

In [101]:
```
print(patientsXAll.columns)
print(coefficients)
```

```
Index(['Gender_'Female'', 'Gender_'Male'',
       'Location_'County General Hospital'',
       'Location_'St. Mary's Medical Center'', 'Location_'VA Hospital'',
       'SelfAssessedHealthStatus_'Excellent'',
       'SelfAssessedHealthStatus_'Fair'', 'SelfAssessedHealthStatus_'Good'',
       'SelfAssessedHealthStatus_'Poor'', 'Age', 'Height', 'Weight',
       'Smoker_0', 'Smoker_1'],
      dtype='object')
[-7.35647214e-01  7.35647214e-01  1.64575586e-01 -1.67794419e-01
  3.21883325e-03  6.29458102e-01  2.31924718e-01 -4.15324079e-01
 -4.46058740e-01  2.15729927e-01 -5.92449588e-01  1.76272116e-01
 -5.18796653e+00  5.18796653e+00]
```

A coefficient of 10 for a numeric, non-dummy attribute indicates that for every +1 standard deviation in the independent variable (exogenous), the dependent variable (endogenous) increases by 10 variable. That is, when **Weight** increases by 1 standard deviation, then predicted diastolic increases by 1.76e-1 or **0.176**.

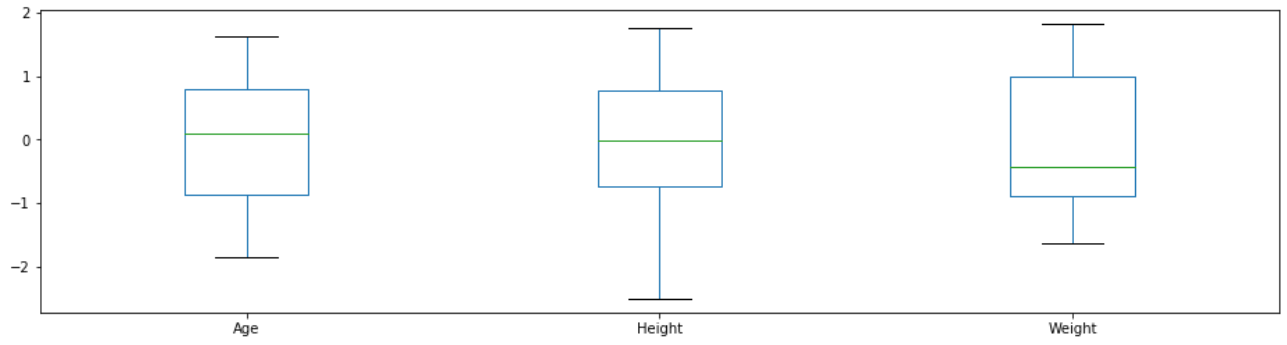
A coefficient of 10 for a categorical, dummy attribute indicates that when the independent variable is 1 (TRUE), the dependent variable increases by 10 variable, relative to the base assumption. That is, when you **Smoke**, predicted diastolic increases by **5.188**, relative to the baseline of not-Smoking.

6. If you need to identify one outlier record, which record is a potential outlier? How do you reach this conclusion?

There are no outliers for categorical (dummy)/binary attributes. Gender, LastName, Location, SelfAssessment and Smoker are all irrelevant in the search for outliers. Hence, we are only interested in the remaining three numeric attributes Age, Height and Weight. Let's begin the search with a box plot.

In [102]:
```
import matplotlib.pyplot as plot

%matplotlib inline
patientsXNumeric_scaleddf.plot.box(figsize=(16,4))
```

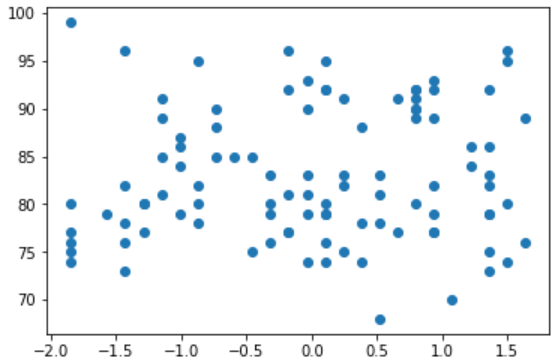
Out[102]: <matplotlib.axes._subplots.AxesSubplot at 0x73d9690198>



This suggests Height has the largest absolute outlier, which is a minimum. We can now examine this with scatterplots.

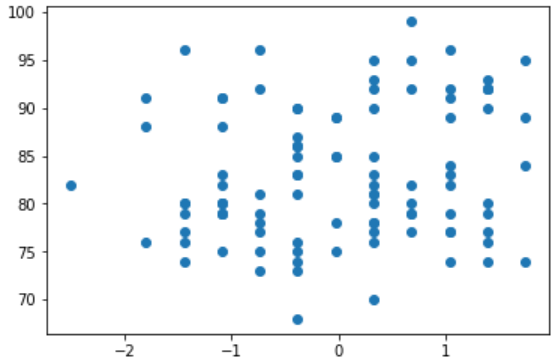
In [107]:
```
import matplotlib.pyplot
#import pylab

matplotlib.pyplot.scatter(patientsXNumeric_scaleddf["Age"], patientsY)
```

Out[107]: <matplotlib.collections.PathCollection at 0x73cf2d64a8>



In [108]:
```
matplotlib.pyplot.scatter(patientsXNumeric_scaleddf["Height"], patientsY)
```

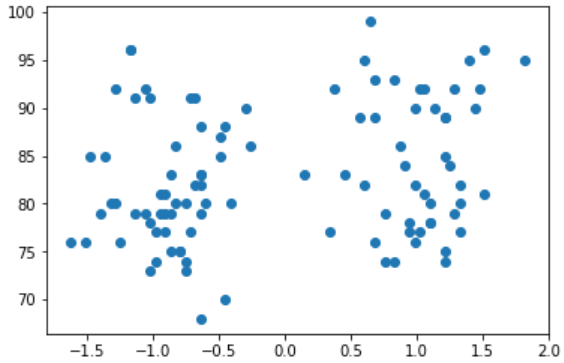
Out[108]: <matplotlib.collections.PathCollection at 0x73cf338358>



Here we see that same minimum outlier in Height.

In [109]:
```
matplotlib.pyplot.scatter(patientsXNumeric_scaleddf["Weight"], patientsY)
```

Out[109]: <matplotlib.collections.PathCollection at 0x73d9706e10>



Find the numbers for those minumums and maximums.

In [110]:
```
patientsXNumeric_scaleddf.min()
```

Out[110]:
```
Age      -1.849776
Height   -2.505093
Weight   -1.626433
dtype: float64
```

In [111]:
```
patientsXNumeric_scaleddf.max()
```

Out[111]:
```
Age       1.632483
Height    1.746833
Weight    1.815553
dtype: float64
```

So far, the single outlier record I identify using a boxplot is the lowest Height value, -2.505

In [112]:
```
patientsXNumeric_scaleddf["Height"].min()
```

Out[112]:
```
-2.505092677361049
```

But what about Cook or Leverage?

Professor Lai points us to statsmodels.stats.outliers_influence.OLSInfluence (statsmodels.stats.outliers_influence.OLSInfluence).

But I could not get this working. The above is all I can manage. Think next time I need to try matlab...

7. If you need to identify one or few useless features (independent variables or predictors), which one(s) will you choose? Why do you reach this conclusion?

The feature I'd remove first is *LastName*, since through one-hot encoding this attribute renders 100 columns. For our dataset consisting of only 100 rows, this is far too high and might conflict againt maximum degress of freedom.

Assignment Complete!