CSCI 3022 Homework

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below.

In [1]:
```
NAME = "Tyler Nevell"
COLLABORATORS = "Cody Hegwer, Stephen Kay"
```

If you referenced any web sites or solutions not of your own creation, list those references here:

- List any external references or resources here

In [2]:
```
%matplotlib inline
import numpy as np
import scipy as sp
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Set color map to have light blue background
sns.set()
import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The pandas.core.da
tetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module in
stead.
  from pandas.core import datetools
```

N.B.: I recommend that you use the `statsmodel` library to do the regression analysis as opposed to *e.g.* `sklearn`. The `sklearn` library is great for advanced topics, but it's easier to get lost in a sea of details and it's not needed for these problems.

Fitting a Regression Model [10 points]

Using the following data:

In [3]:
```
poly_x = np.array([-5.        , -4.79591837, -4.59183673, -4.3877551 , -4.18367347,
       -3.97959184, -3.7755102 , -3.57142857, -3.36734694, -3.16326531,
       -2.95918367, -2.75510204, -2.55102041, -2.34693878, -2.14285714,
       -1.93877551, -1.73469388, -1.53061224, -1.32653061, -1.12244898,
       -0.91836735, -0.71428571, -0.51020408, -0.30612245, -0.10204082,
        0.10204082,  0.30612245,  0.51020408,  0.71428571,  0.91836735,
        1.12244898,  1.32653061,  1.53061224,  1.73469388,  1.93877551,
        2.14285714,  2.34693878,  2.55102041,  2.75510204,  2.95918367,
        3.16326531,  3.36734694,  3.57142857,  3.7755102 ,  3.97959184,
        4.18367347,  4.3877551 ,  4.59183673,  4.79591837,  5.        ])
```

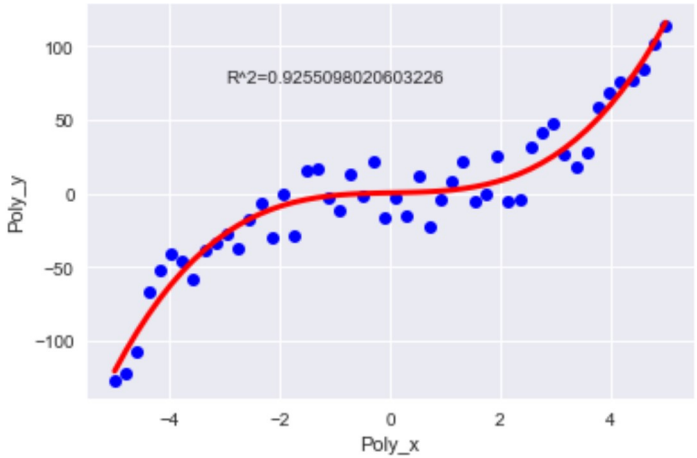
In [4]:
```
poly_y = np.array([-126.99769104, -121.76363745, -107.05551625,  -66.4135443 ,
        -51.66992718,  -40.69610161,  -45.38354202,  -58.75964619,
        -39.16974993,  -33.85796291,  -28.04111151,  -36.81766799,
        -18.28088923,   -6.45665831,  -29.47435017,   -0.75081722,
        -28.57616465,   15.16568923,   16.59302615,   -2.78764992,
        -11.75883341,   13.12494657,   -1.88394446,   22.02465669,
        -16.55482107,   -2.91427672,  -15.78789502,   11.2687875 ,
        -23.09030237,   -3.7153776 ,    7.87571976,   22.1994704 ,
         -5.909068  ,   -0.56562841,   25.03561638,   -5.8824932 ,
         -4.52039745,   31.85118924,   41.35070605,   47.13163559,
         26.12835471,   17.40423706,   27.98496168,   59.02195887,
         68.41065196,   76.08040375,   77.44338322,   83.74765871,
        101.37765463,  113.69478397])
```

In [5]:
```
poly = pd.DataFrame( {'x' : poly_x, 'y': poly_y })
```

Fit a regression model to this data that minimizes the R^2 as well as the complexity of the model. You should consider the models discussed in the book and lecture (linear, polynomial and logrithmic models). Explain, in comments using '#' how you arrived at your model.

In [6]:
```
X = PolynomialFeatures(3).fit_transform(poly.x.values.reshape(-1,1))
ss = sm.OLS(poly.y, X).fit()
plt.plot(poly.x, poly.y, 'bo');
xticks = np.linspace(-5,5)
plt.plot(xticks, np.polynomial.polynomial.polyval(xticks,ss.params), 'r', lw=3)
plt.text(-3,75, 'R^2=' + str(ss.rsquared))
plt.xlabel('Poly_x');
plt.ylabel('Poly_y');

#after plotting our scatterplot, we can see that is clearly respresents a cubic function
```



Multi-Linear Regression

In the following problem, you will construct a simple multi-linear regression model, identify interaction terms and use diagnostic plots to identify outliers in the data. The original problem is as described by John Verzani in the excellent tutorial 'SimplR' on the R statistics language (https://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf) and uses data from the 2000 presidential election in Florida. The problem is interesting because it contains a small number of highly leveraged points that influence the model.

In [7]:
```
votes = pd.read_csv('http://users.stat.ufl.edu/~presnell/Shared/Data/fl2000.txt', delim_whitespace=True, comment='#')
votes = votes[['county', 'Bush', 'Gore', 'Nader', 'Buchanan']]
votes.describe(include='all')
```

Out[7]:

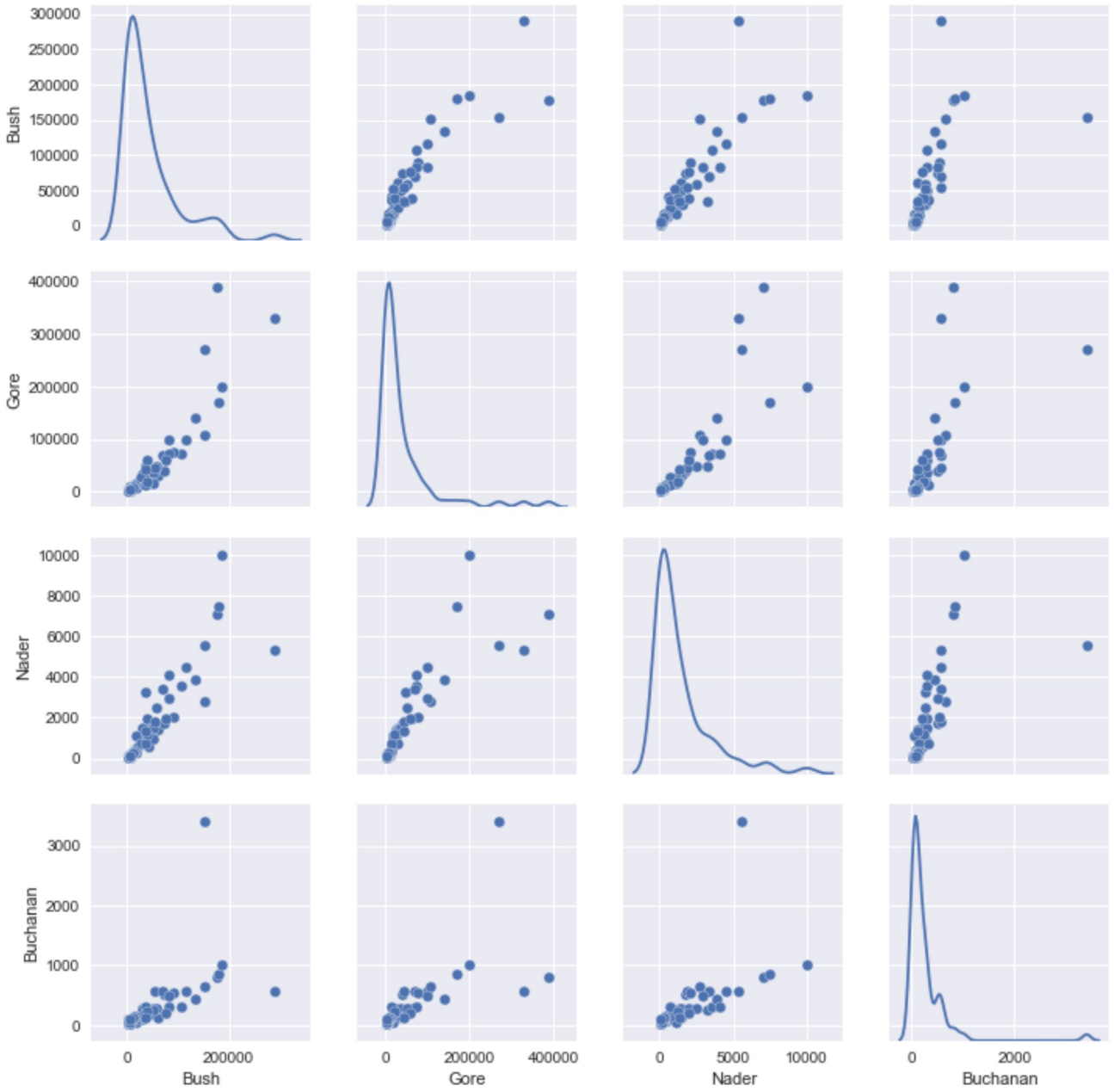
	county	Bush	Gore	Nader	Buchanan
count	67	67.000000	67.000000	67.000000	67.000000
unique	67	NaN	NaN	NaN	NaN
top	Manatee	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN
mean	NaN	43450.970149	43453.985075	1454.119403	260.880597
std	NaN	57182.620266	75070.435056	2033.620972	450.498092
min	NaN	1317.000000	789.000000	19.000000	9.000000
25%	NaN	4757.000000	3058.000000	95.500000	46.500000
50%	NaN	20206.000000	14167.000000	562.000000	120.000000
75%	NaN	56546.500000	46015.000000	1870.500000	285.500000
max	NaN	289533.000000	387703.000000	10022.000000	3411.000000

Plot a pair plot of the data using the `seaborn` library.

In [8]:
```
sns.pairplot(votes[['Bush', 'Gore', 'Nader', 'Buchanan' ]], diag_kind='kde');
votes.corr()
```

Out[8]:

	Bush	Gore	Nader	Buchanan
Bush	1.000000	0.912652	0.892249	0.625185
Gore	0.912652	1.000000	0.864378	0.691314
Nader	0.892249	0.864378	1.000000	0.654215
Buchanan	0.625185	0.691314	0.654215	1.000000



Comment on the relationship between the quantiative datasets. Are they correlated? Colinear?

Correlated, but not colinear.

Multi-linear

Construct a mulit-linear model without interaction terms predicting the Bush column on the other columns and print out the summary table

In [28]:
```
model_votes = smf.ols(formula='Bush~Gore+Nader+Buchanan', data=votes).fit()
model_votes.summary()
```

Out[28]:

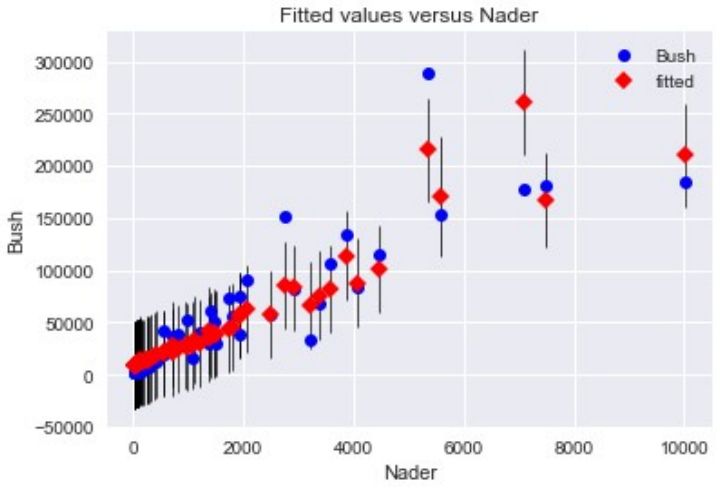
OLS Regression Results

Dep. Variable:	Bush	**R-squared:**	0.877
Model:	OLS	**Adj. R-squared:**	0.871
Method:	Least Squares	**F-statistic:**	149.5
Date:	Tue, 10 Apr 2018	**Prob (F-statistic):**	1.35e-28
Time:	11:43:30	**Log-Likelihood:**	-758.33
No. Observations:	67	**AIC:**	1525.
Df Residuals:	63	**BIC:**	1533.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	8647.6837	3133.545	2.760	0.008	2385.793	1.49e+04
Gore	0.4475	0.071	6.305	0.000	0.306	0.589
Nader	11.8533	2.503	4.735	0.000	6.851	16.855
Buchanan	-7.2033	7.864	-0.916	0.363	-22.917	8.511

Omnibus:	20.698	**Durbin-Watson:**	1.969
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	128.017
Skew:	0.383	**Prob(JB):**	1.59e-28
Kurtosis:	9.728	**Cond. No.**	1.08e+05

In [32]:
```
sm.graphics.plot_fit(model_votes, 2);
```



Multi-linear with interactions

Construct a multi-linear model with interactions that are statistically significant at the $p = 0.05$ level. You can start with full interactions and then eliminate interactions are do not meet the $p = 0.05$ threshold.

In [33]:
```
print("Buchanan:", model_votes.f_test('Buchanan = 0'))
```

```
Buchanan: <F test: F=array([[0.83912398]]), p=0.3631406396613793, df_denom=63, df_num=1>
```

In [29]:
```
model_votes_2 = smf.ols(formula='Bush~Gore+Nader', data=votes).fit()
model_votes_2.summary()
```

Out[29]:

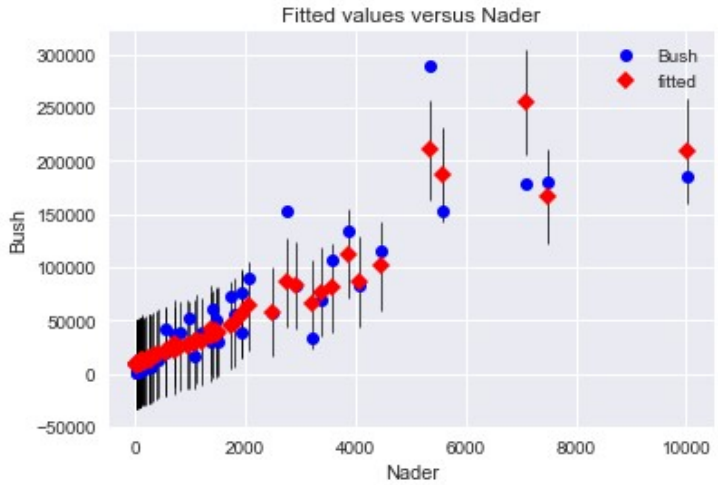
OLS Regression Results

Dep. Variable:	Bush	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.871
Method:	Least Squares	**F-statistic:**	224.4
Date:	Tue, 10 Apr 2018	**Prob (F-statistic):**	1.20e-29
Time:	11:43:33	**Log-Likelihood:**	-758.77
No. Observations:	67	**AIC:**	1524.
Df Residuals:	64	**BIC:**	1530.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	8223.1597	3095.188	2.657	0.010	2039.810	1.44e+04
Gore	0.4260	0.067	6.368	0.000	0.292	0.560
Nader	11.4957	2.469	4.655	0.000	6.563	16.429

Omnibus:	23.156	**Durbin-Watson:**	1.952
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	116.158
Skew:	0.668	**Prob(JB):**	5.98e-26
Kurtosis:	9.311	**Cond. No.**	1.07e+05

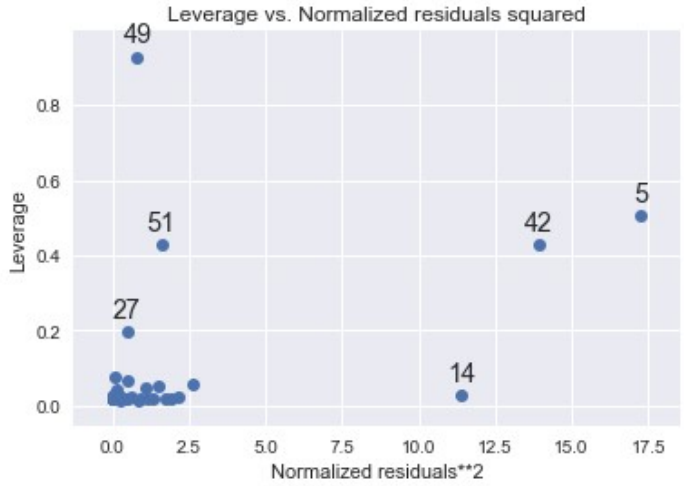
In [30]:
```
sm.graphics.plot_fit(model_votes_2, 2);
```



Leverage

Plot the *leverage* vs. the square of the residual as described in the diagnostic plots.

In [34]:
```
sm.graphics.plot_leverage_resid2(model_votes, alpha=.05);
```



Identify and Clean

The leverage *vs* residual plot indicates that some rows have high leverage but small residuals and others have high residual. The R^2 of the model is determined by the residual. The data is from the disputed 2000 election where one county (https://en.wikipedia.org/wiki/2000_United_States_presidential_election_recount_in_Florida) caused significant issues.

Plot the *4* rows for the points indicated having high leverage and/or high residual squared. You will use this to improve the model R^2.

In [35]:
```
votes_final = votes.drop([5, 42, 49, 14])
votes_final
```

Out[35]:

	county	Bush	Gore	Nader	Buchanan
0	Alachua	34124	47365	3226	263
1	Baker	5610	2392	53	73
2	Bay	38637	18850	828	248
3	Bradford	5414	3075	84	65
4	Brevard	115185	97318	4470	570
6	Calhoun	2873	2155	39	90
7	Charlotte	35426	29645	1462	182
8	Citrus	29767	25525	1379	270
9	Clay	41736	14632	562	186
10	Collier	60450	29921	1400	122
11	Columbia	10964	7047	258	89
12	DeSoto	4256	3320	157	36
13	Dixie	2697	1826	75	29
15	Escambia	73017	40943	1727	502
16	Flagler	12613	13897	435	83
17	Franklin	2454	2046	85	33
18	Gadsden	4767	9735	139	38
19	Gilchrist	3300	1910	97	29
20	Glades	1841	1442	56	9
21	Gulf	3550	2397	86	71
22	Hamilton	2146	1722	37	23
23	Hardee	3765	2339	75	30
24	Hendry	4747	3240	104	22
25	Hernando	30646	32644	1501	242
26	Highlands	20206	14167	545	127
27	Hillsborough	180760	169557	7490	847

Final model

Develop your final model by dropping *one* of the troublesome data points indicated in the leverage *vs* residual plot and insuring any interactions in your model are still significant at $p = 0.05$. Your model should have an R^2 great than 0.95.

In [36]:
```
model_votes_final = smf.ols(formula='Bush~Gore+Nader+Buchanan', data=votes_final).fit()
model_votes_final.summary()
```

Out[36]:

OLS Regression Results

Dep. Variable:	Bush	**R-squared:**	0.939
Model:	OLS	**Adj. R-squared:**	0.936
Method:	Least Squares	**F-statistic:**	303.2
Date:	Tue, 10 Apr 2018	**Prob (F-statistic):**	8.49e-36
Time:	12:26:19	**Log-Likelihood:**	-670.92
No. Observations:	63	**AIC:**	1350.
Df Residuals:	59	**BIC:**	1358.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	2679.0773	1887.637	1.419	0.161	-1098.075	6456.229
Gore	0.7167	0.130	5.525	0.000	0.457	0.976
Nader	1.0226	2.866	0.357	0.723	-4.713	6.758
Buchanan	48.8488	15.698	3.112	0.003	17.436	80.261

Omnibus:	11.691	**Durbin-Watson:**	1.825
Prob(Omnibus):	0.003	**Jarque-Bera (JB):**	13.984
Skew:	0.773	**Prob(JB):**	0.000919
Kurtosis:	4.714	**Cond. No.**	7.07e+04

Body Mass Index Model

In this problem, you will first clean a data set and create a model to estimate body fat based on the common BMI measure. Then, you will use the **forward stepwise selection** and **backware stepwise selection** methods to create more accurate predictors for body fat.

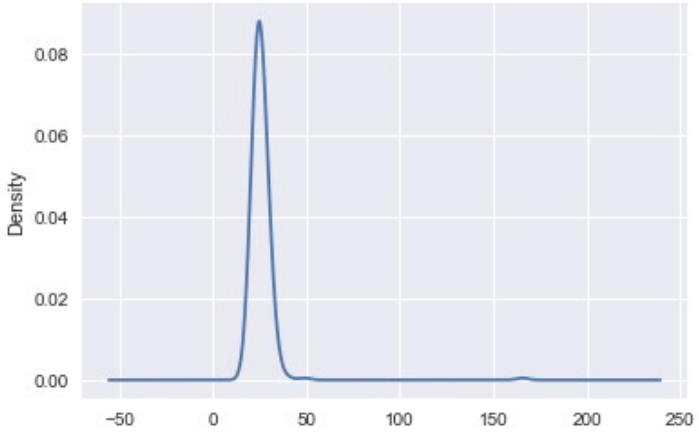
The body density dataset in file `bodyfat` includes the following 15 variables listed from left to right:

- Density : Density determined from underwater weighing
- Fat : Percent body fat from Siri's (1956) equation
- Age : Age (years)
- Weight : Weight (kg)
- Height : Height (cm)
- Neck : Neck circumference (cm)
- Chest: Chest circumference (cm)
- Abdomen : Abdomen circumference (cm)
- Hip : Hip circumference (cm)
- Thigh : Thigh circumference (cm)
- Knee : Knee circumference (cm)
- Ankle : Ankle circumference (cm)
- Biceps : Biceps (extended) circumference (cm)
- Forearm : Forearm circumference (cm)
- Wrist : Wrist circumference (cm)

The `Density` column is the "gold standard" -- it is a measure of body density obtained by dunking people in water and measuring the displacement. The `Fat` column is a prediction using another statistical model. The body mass index (BMI) is calculated as Kg/m^2 (https://en.wikipedia.org/wiki/Body_mass_index) and is used to classify people into different weight categories with a BMI over 30 being 'obese' (https://www.medicalnewstoday.com/info/obesity). You will find that BMI is a poor predictor of the `Density` information it purports to predict. You will try to find better models using measurements and regression.

Unfortunately for us, the dataset we have has imperial units for weight and height, so we will convert those to metric and then calculate the BMI and plot the KDE of the data.

In [87]:
```
fat = pd.read_csv('bodyfat.csv')
fat = fat.drop('Unnamed: 0', axis=1)
fat.Weight = fat.Weight * 0.453592 # Convert to Kg
fat.Height = fat.Height * 0.0254 # convert inches to m
fat['BMI'] = fat.Weight / (fat.Height**2)
fat.BMI.plot.kde();
```



The BMI has at least one outlier since it's unlikely anyone has a BMI of 165, even Arnold Schwarzenegger (http://www.health.com/health/gallery/0,,20460621,00.html).

Form a new table `cfat` (cleaned fat) that removes any rows with a BMI greater than 40 and calculate the regression model predicting the `Density` from the `BMI`. Display the summary of the regression model. You should achieve an R^2 of at least 0.53.

In [88]:
```
cfat = fat.drop(fat[fat['BMI'] > 40].index)
cfat
```

Out[88]:

	Density	BodyFat	Age	Weight	Height	Neck	Chest	Abdomen	Hip	Thigh	Knee	Ankle	Biceps	Forearm	Wrist	BMI
0	1.0708	12.3	23	69.966566	1.72085	36.2	93.1	85.2	94.5	59.0	37.3	21.9	32.0	27.4	17.1	23.626779
1	1.0853	6.1	22	78.584814	1.83515	38.5	93.6	83.0	98.7	58.7	37.3	23.4	30.5	28.9	18.2	23.334338
2	1.0414	25.3	22	69.853168	1.68275	34.0	95.8	87.9	99.2	59.6	38.9	24.0	28.8	25.2	16.6	24.668737
3	1.0751	10.4	26	83.801122	1.83515	37.4	101.8	86.4	101.2	60.1	37.3	22.8	32.4	29.4	18.2	24.883227
4	1.0340	28.7	24	83.574326	1.80975	34.4	97.3	100.0	101.9	63.2	42.2	24.0	32.2	27.7	17.7	25.517358
5	1.0502	20.9	24	95.367718	1.89865	39.0	104.5	94.4	107.8	66.0	42.0	25.6	35.7	30.6	18.8	26.455231
6	1.0549	19.2	26	82.100152	1.77165	36.4	105.1	90.7	100.3	58.4	38.3	22.9	31.9	27.8	17.7	26.157010
7	1.0704	12.4	25	79.832192	1.84150	37.8	99.6	88.5	97.1	60.0	39.4	23.2	30.5	29.0	18.8	23.541526
8	1.0900	4.1	25	86.636072	1.87960	38.1	100.9	82.5	99.9	62.9	38.3	23.8	35.9	31.1	18.2	24.522677
9	1.0722	11.7	23	89.924614	1.86690	42.1	99.6	88.6	104.1	63.1	41.7	25.0	35.6	30.0	19.2	25.800996
10	1.0830	7.1	26	84.481510	1.89230	38.5	101.5	83.6	98.2	59.7	39.7	25.2	32.8	29.4	18.5	23.592920
11	1.0812	7.8	27	97.975872	1.93040	39.4	103.6	90.9	107.7	66.2	39.2	25.9	37.2	30.2	19.0	26.292054
12	1.0513	20.8	32	81.873356	1.76530	38.4	102.0	91.6	103.9	63.4	38.3	21.5	32.5	28.6	17.7	26.272751
13	1.0505	21.2	30	93.099758	1.80975	39.4	104.1	101.8	108.6	66.0	41.5	23.7	36.9	31.6	18.8	28.425714
14	1.0484	22.1	35	85.161898	1.76530	40.5	101.3	96.4	100.1	69.0	39.0	23.1	36.1	30.5	18.2	27.328028
15	1.0512	20.9	35	73.822098	1.67640	36.4	99.1	92.8	99.2	63.1	38.7	21.7	31.1	26.4	16.9	26.268246
16	1.0333	29.0	34	88.790634	1.80340	38.9	101.9	96.4	105.2	64.8	40.8	23.1	36.2	30.8	17.3	27.301281
17	1.0468	22.9	32	94.914126	1.80340	42.1	107.6	97.5	107.0	66.9	40.0	24.4	38.2	31.6	19.3	29.184128
18	1.0622	16.0	28	83.347530	1.72085	38.0	106.8	89.6	102.4	64.2	38.7	22.9	37.2	30.5	18.5	28.145353
19	1.0610	16.5	33	96.048106	1.86690	40.0	106.2	100.5	109.0	65.8	40.6	24.0	37.1	30.1	18.2	27.557936
20	1.0551	19.1	28	81.192968	1.72720	39.1	103.3	95.9	104.9	63.5	38.0	22.1	32.5	30.3	18.4	27.216555
21	1.0640	15.2	28	90.945196	1.77165	41.3	111.4	98.8	104.8	63.4	40.6	24.6	33.0	32.8	19.9	28.975031
22	1.0631	15.6	31	63.616278	1.73355	33.9	86.0	76.4	94.6	57.4	35.3	22.2	27.9	25.9	16.7	21.168765
23	1.0584	17.7	32	67.471810	1.77800	35.5	86.7	80.0	93.4	54.9	36.2	22.1	29.8	26.7	17.1	21.343166
24	1.0668	14.0	28	68.605790	1.72085	34.5	90.2	76.3	95.8	58.4	35.5	22.9	31.1	28.0	17.6	23.167263
25	1.0911	3.7	27	72.234526	1.81610	35.7	89.6	79.7	96.5	55.0	36.7	22.5	29.9	28.2	17.7	21.901069

In [89]:
```
cfat.BMI.plot.kde();
model_cfat = smf.ols(formula='Density~BMI', data=cfat).fit()
model_cfat.summary()
```

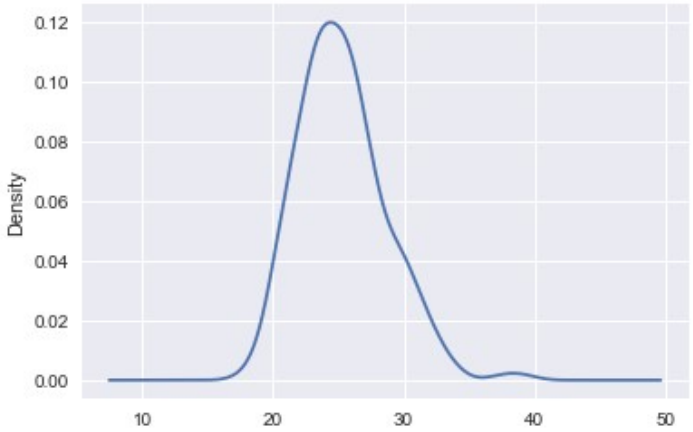
Out[89]:

OLS Regression Results

Dep. Variable:	Density	**R-squared:**	0.536
Model:	OLS	**Adj. R-squared:**	0.534
Method:	Least Squares	**F-statistic:**	286.2
Date:	Tue, 10 Apr 2018	**Prob (F-statistic):**	3.25e-43
Time:	15:22:34	**Log-Likelihood:**	734.17
No. Observations:	250	**AIC:**	-1464.
Df Residuals:	248	**BIC:**	-1457.
Df Model:	1		
Covariance Type:	nonrobust		

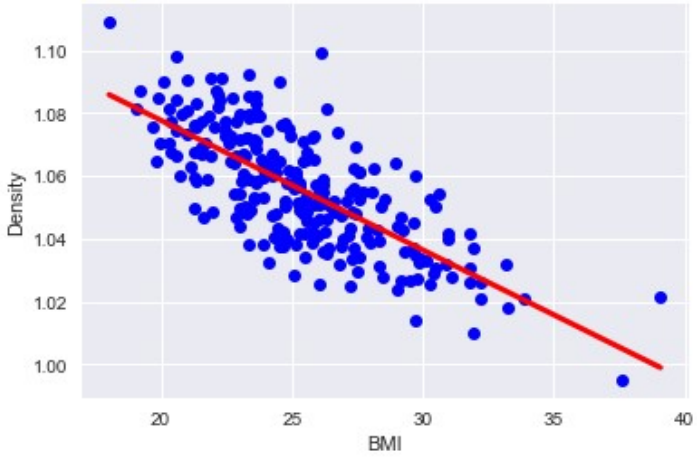
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.1602	0.006	186.410	0.000	1.148	1.172
BMI	-0.0041	0.000	-16.918	0.000	-0.005	-0.004

Omnibus:	2.262	**Durbin-Watson:**	1.576
Prob(Omnibus):	0.323	**Jarque-Bera (JB):**	2.259
Skew:	0.229	**Prob(JB):**	0.323
Kurtosis:	2.916	**Cond. No.**	195.



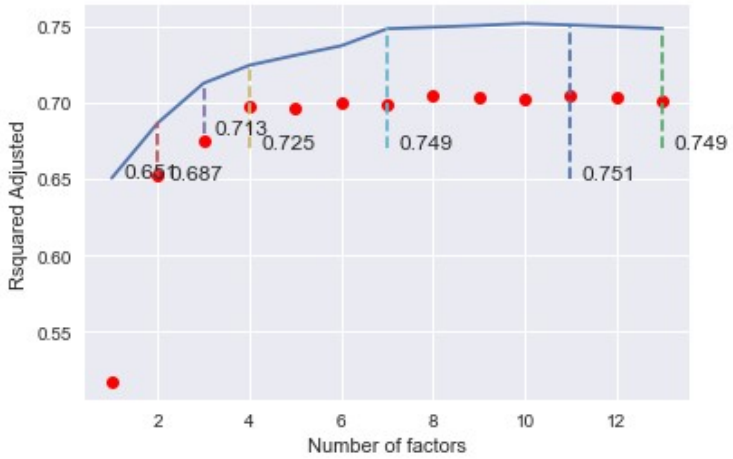
Plot your regression model against the BMI measurement, properly labeling the scatterplot axes and showing the regression line. In subsequent models, you will not be able to plot the Density *vs* your predictors because you will have too many predictors, but it's useful to visually understand the relationship between the BMI predictor and the `Density` because you should find that the regression line goes through the data but there is too much variability in the data to achieve a good R^2.

In [90]:
```
plt.plot(cfat.BMI, cfat.Density, 'bo');
xticks = np.linspace(cfat['BMI'].min(), cfat['BMI'].max())
plt.plot(xticks, model_cfat.params[0] + xticks * model_cfat.params[1], 'r', lw=3);
plt.xlabel('BMI');
plt.ylabel('Density');
```



Constructing Better Predictors for Bodyfat

The `BMI` model uses easy-to-measure predictors, but has a poor $R^2 \sim 0.54$. We will use structured subset selection methods from ISLR Chapter 6.1 to derive two better predictors. That chapter covers *best subset*, *forward stepwise* and *backware stepwise* selection. I have implemented the *best subset* selection which searches across all combinations of $1, 2, \ldots, p$ predictors and selects the best predictor based on the $adjusted\ R^2$ metric. This method involved analyzing $2^{13} = 8192$ regression models (programming and computers for the win). The resulting $adjusted\ R^2$ plot is shown below:



In this plot, `test_fat` and `train_fat` datasets each containing 200 randomly selected samples were derived from the `cfat` dataset using `np.random.choice` over the `cfat.index` and selected using the Pandas `loc` method. Then, following the algorithm of ISLR Algorithm 6.1 *Best Subset Selection*, all $\binom{p}{k}$ models with k predictors were evaluated on the training data and the model returning the best $Adjusted\ R^2$ was selected. These models are indicated by the data points for the solid blue line. As the text indicates, other measures (AIC, BIC, C_p) would be better than the $Adjusted\ R^2$, but we use it becuase because you've already seen the R^2 and should have an understanding of what it means.

Then, the best models for each k were evaluated for the `test_fat` data. These results are shown as the red dots below the blue line. Note that because the test and train datasets are randomly selected subsets, the results vary from run-to-run and it may that your test data produces better R^2 than your training data.

This splitting of the data into a test and train set is similar to the *validation set approach* that you can read about in Chapter 5 of ISLR if you choose, but differs in that the two sets may contain duplicate data. The chief reason to use a test and training set is to drill into your skull the limitation of models on unseen data.

In the following exercises, you can not use the `Density`, `Fat` or `BMI` columns in your predictive models. You can only use the 13 predictors in the `allowed_factors` list.

In [91]:
```
allowed_factors = ['Age', 'Weight', 'Height', 'Neck', 'Chest',
       'Abdomen', 'Hip', 'Thigh', 'Knee', 'Ankle', 'Biceps', 'Forearm',
       'Wrist']
```

Forward Stepwise Refinement

You will manuall perform the steps of the *forward stepwise selection* method for four parameters. You will do this following Algorithm 6.2 from ISLR. For $k = 1 \ldots 4$:

- Set up a regression model with k factors that involves the fixed predictors from the previous step $k-1$
- Try all p predictors in the new kth position
- Select the best parameter using $Adjusted - R^2$ (e.g. `model.rsquared_adj`) given your training data
- Fix the new parameter and continue the process for $k+1$

Then, you will construct a plot similar to the one above, plotting the $Adjusted - R^2$ for each of your k steps and plotting the $Adjusted - R^2$ from the test set using that model.

First, construct your training and test sets. These should each contain 200 (possibly duplicated) samples from your `cfat` dataset:

In [92]:
```
train_fat = cfat.loc[np.random.choice(cfat.index,200)]
test_fat = cfat.loc[np.random.choice(cfat.index,200)]


#model_cfat1 = smf.ols(formula = 'Density~BMI+Age', data=cfat).fit()
#model_cfat1.rsquared
```

Conduct the algorithm above for $k = 1$, leaving your best solution as the answer

In [93]:
```
#model_cfat_age = smf.ols(formula = 'Density~Age', data=train_fat).fit()
#print("Age: ", model_cfat_age.rsquared_adj - model_cfat_age.rsquared)
#model_cfat_weight = smf.ols(formula = 'Density~Weight', data=train_fat).fit()
#print("Weight: ", model_cfat_weight.rsquared_adj - model_cfat_weight.rsquared)
#model_cfat_height = smf.ols(formula = 'Density~Height', data=train_fat).fit()
#print("Height: ", model_cfat_height.rsquared_adj - model_cfat_height.rsquared)
model_cfat_abdomen = smf.ols(formula = 'Density~Abdomen', data=train_fat).fit()
print("Abdomen: ", model_cfat_abdomen.rsquared_adj - model_cfat_abdomen.rsquared)
allowed_factors.remove('Abdomen')
```

```
Abdomen:  -0.0016735580176558962
```

Conduct the algorithm above for $k = 2$, leaving your best solution as the answer

In [102]:
```
test = 100
winner = "none"
for predict in allowed_factors:
    tempFormula = 'Density~Abdomen+' + predict
    model_cfat_temp = smf.ols(formula = tempFormula, data=train_fat).fit()
    if np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared) < test:
        test = np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared)
        winner = predict
        model_cfat_2 = model_cfat_temp


print("Winner is:", winner, "R^2:", test)
```

```
Winner is: Height R^2: 0.0029929631726018924
```

In [104]:
```
allowed_factors.remove(winner)
```

Conduct the algorithm above for $k = 3$, leaving your best solution as the answer

In [105]:
```
test = 100
winner = "none"
for predict in allowed_factors:
    tempFormula = 'Density~Abdomen+Height+' + predict
    model_cfat_temp = smf.ols(formula = tempFormula, data=train_fat).fit()
    if np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared) < test:
        test = np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared)
        winner = predict
        model_cfat_3 = model_cfat_temp


print("Winner is:", winner, "R^2:", test)
```

```
Winner is: Wrist R^2: 0.004334946201302814
```

In [106]:
```
allowed_factors.remove(winner)
```

Conduct the algorithm above for $k = 4$, leaving your best solution as the answer

In [107]:
```
test = 100
winner = "none"
for predict in allowed_factors:
    tempFormula = 'Density~Abdomen+Height+Wrist+' + predict
    model_cfat_temp = smf.ols(formula = tempFormula, data=train_fat).fit()
    if np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared) < test:
        test = np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared)
        winner = predict
        model_cfat_4 = model_cfat_temp


print("Winner is:", winner, "R^2:", test)
```

```
Winner is: Chest R^2: 0.005680006573548946
```

In [108]:
```
allowed_factors.remove(winner)
```

Conduct the algorithm above for $k = 5$, leaving your best solution as the answer

In [109]:
```
test = 100
winner = "none"
for predict in allowed_factors:
    tempFormula = 'Density~Abdomen+Height+Wrist+Chest+' + predict
    model_cfat_temp = smf.ols(formula = tempFormula, data=train_fat).fit()
    if np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared) < test:
        test = np.absolute(model_cfat_temp.rsquared_adj - model_cfat_temp.rsquared)
        winner = predict
        model_cfat_5 = model_cfat_temp


print("Winner is:", winner, "R^2:", test)
```

```
Winner is: Ankle R^2: 0.007048529987485508
```

Plot

Plot your resulting $adjusted\ R^2$ *vs* number of predictors and overlay the $adjusted\ R^2$ for the test data.

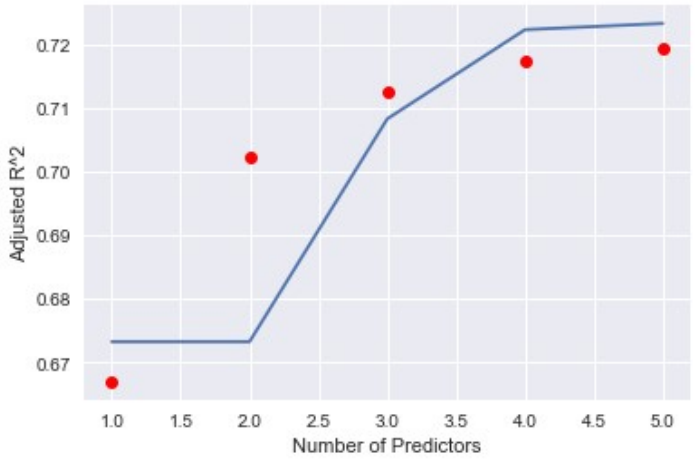
In [112]:
```
trainList = [model_cfat_abdomen.rsquared_adj, model_cfat_2.rsquared_adj, model_cfat_3.rsquared_adj, model_cfat_
4.rsquared_adj, model_cfat_5.rsquared_adj]


model_cfat_test1 = smf.ols(formula='Density~Abdomen', data=test_fat).fit()
model_cfat_test2 = smf.ols(formula='Density~Abdomen', data=test_fat).fit()
model_cfat_test3 = smf.ols(formula='Density~Abdomen+Weight', data=test_fat).fit()
model_cfat_test4 = smf.ols(formula='Density~Abdomen+Weight+Thigh', data=test_fat).fit()
model_cfat_test5 = smf.ols(formula='Density~Abdomen+Weight+Thigh+Biceps', data=test_fat).fit()

testList = [model_cfat_test1.rsquared_adj, model_cfat_test2.rsquared_adj, model_cfat_test3.rsquared_adj, model_
cfat_test4.rsquared_adj, model_cfat_test5.rsquared_adj]


predictors = [1,2,3,4,5]
plt.plot(predictors, trainList, 'ro')
plt.plot(predictors, testList)

plt.xlabel('Number of Predictors');
plt.ylabel('Adjusted R^2');
```



Critique

The BMI model has the benefit being simple (two measurements, height and wright). Looking at your resulting regression model, how many parameters would you suggest to use for your enhanced BMI model? Justify your answer using your models.

4-5 predictors. After that, it peters out after that and doesn't make as much of a difference.