UNIVERSITY OF PENNSYLVANIA

ESE 305: Homework 2

Braden Fineberg
ESE 305: Introduction to Machine Learning

September 29, 2017

1 Conceptual

1.1 Statistical Significance

Given the statistical information in the table provided, several conclusions can be drawn. The intercept, TV,\ and radio are all statistically significant. This is indicated by the especially low p-value. This translates to the impact that each category of ads have on the gross sales of the company. TV and radio advertisements have a statistically significant impact on sales. I am unable to make that same conclusion with newspaper. \ Null Hypothesis H_0 = There is no correlation between advertisements run on various mediums (TV, Radio, Newspaper) and the gross sales of a company.

1.1.1 Intercept

With a high degree of certainty, the intercept coefficient is not zero. The intercept of with a coefficient of 2.939 is a highly unlikely realization of the data if the true impact that these factors (TV, Radio, and Newspaper) have is 0. Therefore, this is a characteristic of the data.

1.1.2 TV

TV advertisements have a definitive impact on gross sales. The likelihood that this scenario of sales would not occur naturally without the addition of TV advertising dollars. If the spend in other mediums are fixed, then for every addition dollar spent on TV advertising will increase gross sales by 0.046. This is unlikely to occur naturally (extremely low P-value), therefore, TV spend has an impact on gross sales.

1.1.3 Radio

Radio also has a significant impact on sales numbers. If all other spend is held constant, for every additional dollar spent on radio will increase gross sales by 0.189. This is extremely unlikely to occur naturally. Therefore, a conclusion can be drawn that increase radio spend does increase gross sales, leading to a correlation.

1.1.4 Newspaper

In contrast to the other advertising mediums, I am unable to draw a conclusion from newspaper spend. I cannot reject the null hypothesis. In common language this indicates that the increase in newspaper spend cannot definitively impact gross sales. For newspaper, this realization of the data could have occurred naturally and is within the bounds of reason; therefore, no conclusion can be drawn.

1.1.5 General Conclusion

I would recommend that the company spends advertising dollars on TV and Radio. If extra money is found, devote it to radio spend as this will maximize the return. Newspaper has not statistically increased sales; therefore, I would not continue spending money on this form of advertising.

1.2 Least Squares Analysis

1.2.1 Which is True?

3 is TRUE. As a result of the increasing value of GPA (1.0-4.0), a fixed GPA of greater than 3.5 will compensate for the additional salary from being a women. A salary of above 3.5 combined with the positive correlation of being male, guys will tend to make more than females. This is compounded by the fact that gender is binary. The additional salary is only added once to the total; in the case of GPAs, a 1 point increase from 2.0 to 3.0 will increase salary by 20k, then again as the student moves from 3.0 to 4.0.

1.2.2 Proof

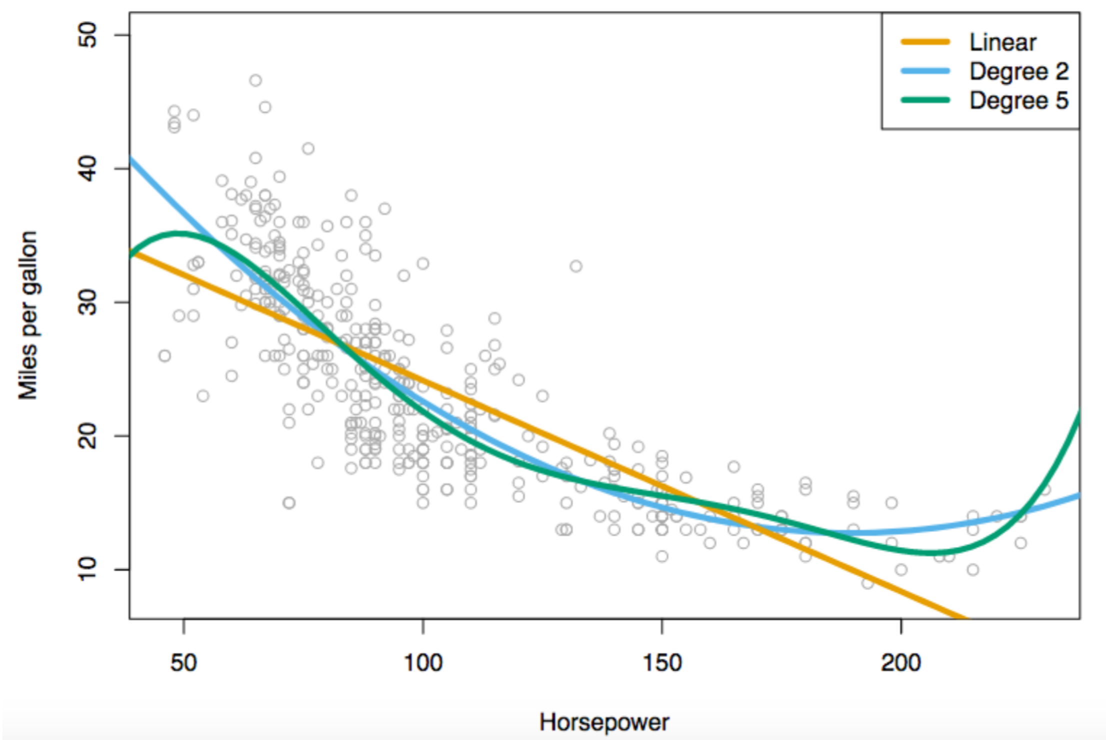
$$\begin{aligned}
E(salary) =& 50 + 20X_1 + 0.07X_2 + 35X_3 + 0.01X_4 - 10X_5 && (1)\\
=& 50 + 20(GPA) + 0.07(IQ) + 35(Gender) + 0.01(GPA * IQ) - 10(GPA * Gender)) && (2)\\
=& 50 + 20(4.0) + 0.07(110) + 35(1) + 0.01(440) - 10(4) && (3)\\
=& 172.7 + 0.01(440) - 40 && (4)\\
=& 137.1 \quad \square && (5)
\end{aligned}$$

Fasle. No conlcusion can be drawn from purely a coefficient, the standard error and p-value must be considered. For example, this counld indicate that because there is a such a strong correlation, the salary does not need to depend on both (ie. someone with a high GPA likely has a high IQ). However, without the p-value, there is not way to know wheter this is statistically significant.

1.3 Cubic vs. Linear Regression

1.3.1 Linear Training

I would expect the cubic regression to be lower. While the true relationship might be linear, it is unlikely that this realization of the scenario is perfectly linear. Given this fact, allowing the model the bend slightly will decrease its distance to the point, decreasing the RSS. The image from the lecture slides below does a good job illustrating this behavior of more flexible models to fit training data.



1.3.2 Given Linear Test Data

Knowing that the data represents a linear relationship, a linear fit of the training data will better predict the test data, decreasing the RSS. A more flexible model will adapt better to the training data, but the flexibility will make it difficult to predict new data accurately (test data). this is illustrated by the green line above. The cubic model will be swayed by noise in the training data and attempt to fit to it. Therefore, linear model will be much accurate when new data is presented, especially if the true relationship is linear.

1.3.3 Given non-linear Training Data

Again, a more flexible model will preform better (lower RSS) in training. Because the model is more flexible, it will pass through more points minimizing RSS until it passes through every point (RSS =0); however this can be very dangerous, as illustrated by the image above, the yellow line (linear) under fits, the green line (quintic) overfits. The blue line most accurately fits the data. Because this data set is fairly small (1 regressors, 1 predictor, 100 samples), it will be most beneficial to visualize the data on a 2d plot and pick the regression that appears to follow the underlying trend of the data. If visualization is not possible, the cubic model will have a lower RSS.

1.3.4 Given Unknown Data

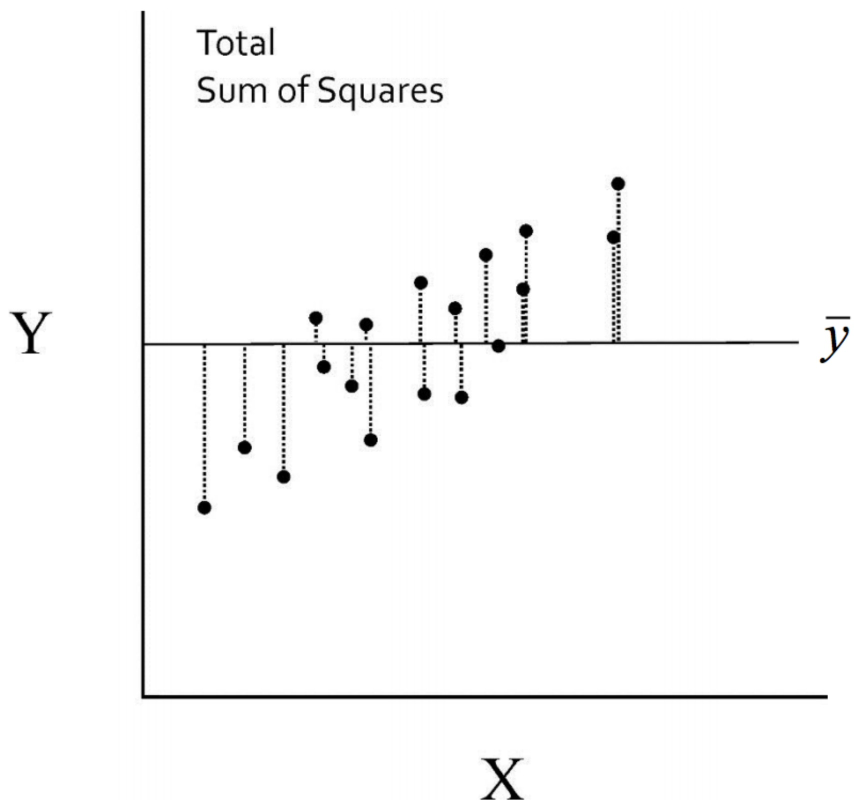
Without visualizing the underlying data it is hard to tell how this will perform. If the data truly has a cubic relationship, then the cubic fit will have a lower RSS. For this scenario, I would want to visualize the data before making a final decision.

1.4 RSS proof

Let $\bar{y}$ = the average of all y values

Let $\bar{x}$ = the average of all x values

Therefore the line y = $\bar{y}$ minimizes the RSS in the y axis as shown by the image below:



The same logic can be applied for the line x = $\bar{x}$.

$\hat{\beta}_1$ computes the slope by determining the covariance of x and y and normalizing by the x values. This will determine how best to fit the slope to the set of points.

$\hat{\beta}_0$ computes the ideal intercept by beginning at the $(\bar{x}, \bar{y})$ and stepping back until x = 0. This will determine the best intercept.

Therefore, if $\bar{x}$ and $\bar{y}$ minimize error on the respective axis, then to minimize the error for this linear model, the line should pass through the point $(\bar{x}, \bar{y})$. To finish minimizing error, the line should consider the slope and intercept of the line as denoted by $\hat{\beta}_0$ and $\hat{\beta}_1$

1.5 Applied

1.6 Fitting linear regression to data

```
In [84]: import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         %matplotlib inline
         sns.set(context='poster', style='whitegrid')
```

```
In [85]: np.random.seed(seed=42)
         mu, sigma = 0, 1
         x = np.random.normal(mu, sigma, 100)
```

```
In [86]: np.random.seed(seed=31)
         eps = np.random.normal(0,np.sqrt(0.25), 100)
```

```
In [87]: Y = -1 + 0.5*x + eps
```

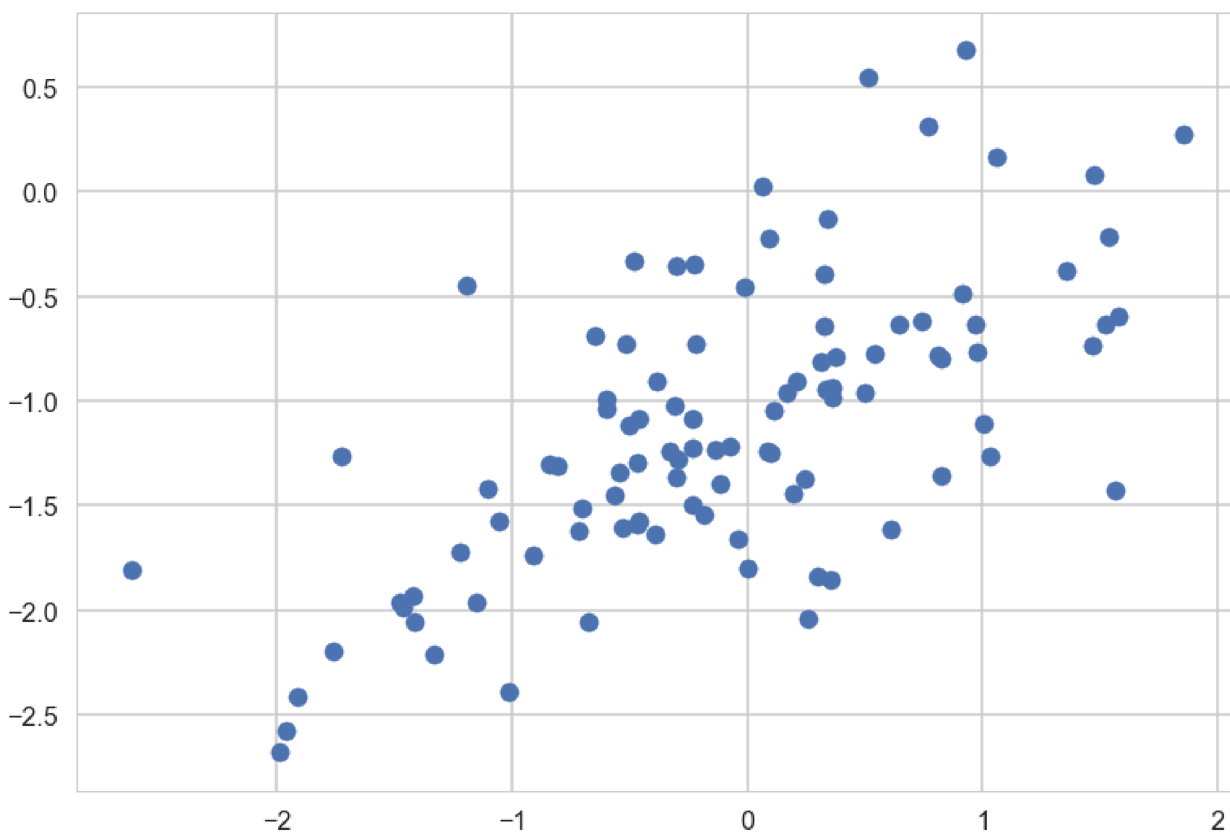
```
In [88]: np.size(Y)
```

```
Out[88]: 100
```

As indicated above, Y is 100 values long, as expected.\
β_0 = -1\
β_1 = 0.5

```
In [89]: plt.scatter(x, Y)
```

```
Out[89]: <matplotlib.collections.PathCollection at 0x116c1f390>
```



I see a relatively linear correlation, as expected, but there is variation due to the addition of "noise" which is generated by ϵ.

```
In [90]: from sklearn.linear_model import LinearRegression
         from matplotlib.collections import LineCollection
         from sklearn.metrics import mean_squared_error, classification_report
         import statsmodels.api as sm
         import statsmodels.formula.api as smf
         import scipy.stats as stats
```

```
In [91]: lr = LinearRegression()
         lr.fit(x[:, np.newaxis], Y)  # x needs to be 2d for LinearRegression
         predsLr = lr.predict(x[:, np.newaxis])
```

```
In [92]: print("Best Fit: y = " + str(np.round(lr.coef_[0], 3))+"x " +  str(np.round(
         print("beta_0 = "+ str(np.round(lr.intercept_, 3)))
         print("beta_1 = "+ str(np.round(lr.coef_[0], 3)))
         RSS = np.round(np.sum((Y-predsLr)**2), 4)
         RSE = np.round(np.sqrt(1/len(Y-2)*RSS), 4)
         MSE = np.round(mean_squared_error(Y, predsLr), 4)
         print('RSS = ' + str(RSS))
         print('RSE = ' + str(RSE))
         print("MSE = " +  str(MSE))
```

```
Best Fit: y = 0.491x -1.074
beta_0 = -1.074
beta_1 = 0.491
RSS = 26.0699
RSE = 0.5106
MSE = 0.2607
```

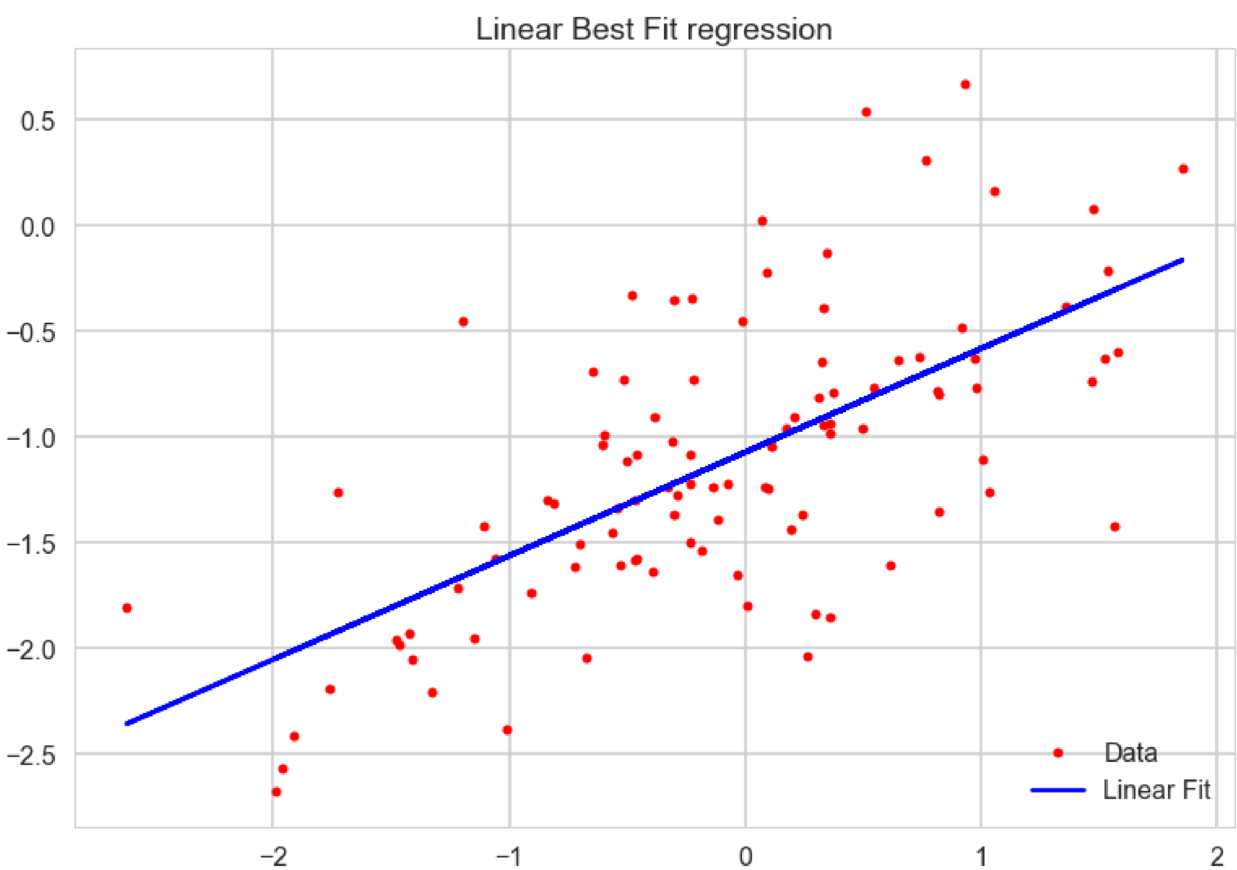
```
In [93]: results = sm.OLS(Y, sm.add_constant(x)).fit()
         print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.430
Model:                            OLS   Adj. R-squared:                  0.424
Method:                 Least Squares   F-statistic:                     74.02
Date:                Fri, 29 Sep 2017   Prob (F-statistic):           1.29e-13
Time:                        11:24:09   Log-Likelihood:                -74.674
No. Observations:                 100   AIC:                             153.3
Df Residuals:                      98   BIC:                             158.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.0742      0.052    -20.691      0.000      -1.177      -0.971
x1             0.4911      0.057      8.603      0.000       0.378       0.604
```

```
==============================================================================
Omnibus:                        3.815   Durbin-Watson:                   2.002
Prob(Omnibus):                  0.148   Jarque-Bera (JB):                3.267
Skew:                           0.431   Prob(JB):                        0.195
Kurtosis:                       3.206   Cond. No.                         1.16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

Using both linear regression and statsmodels, the coefficients are the same. $\hat{\beta}_0 = -1.0742$ and $\hat{\beta}_1 = 0.4911$. These vary slightly from the values provided, but not substantially.

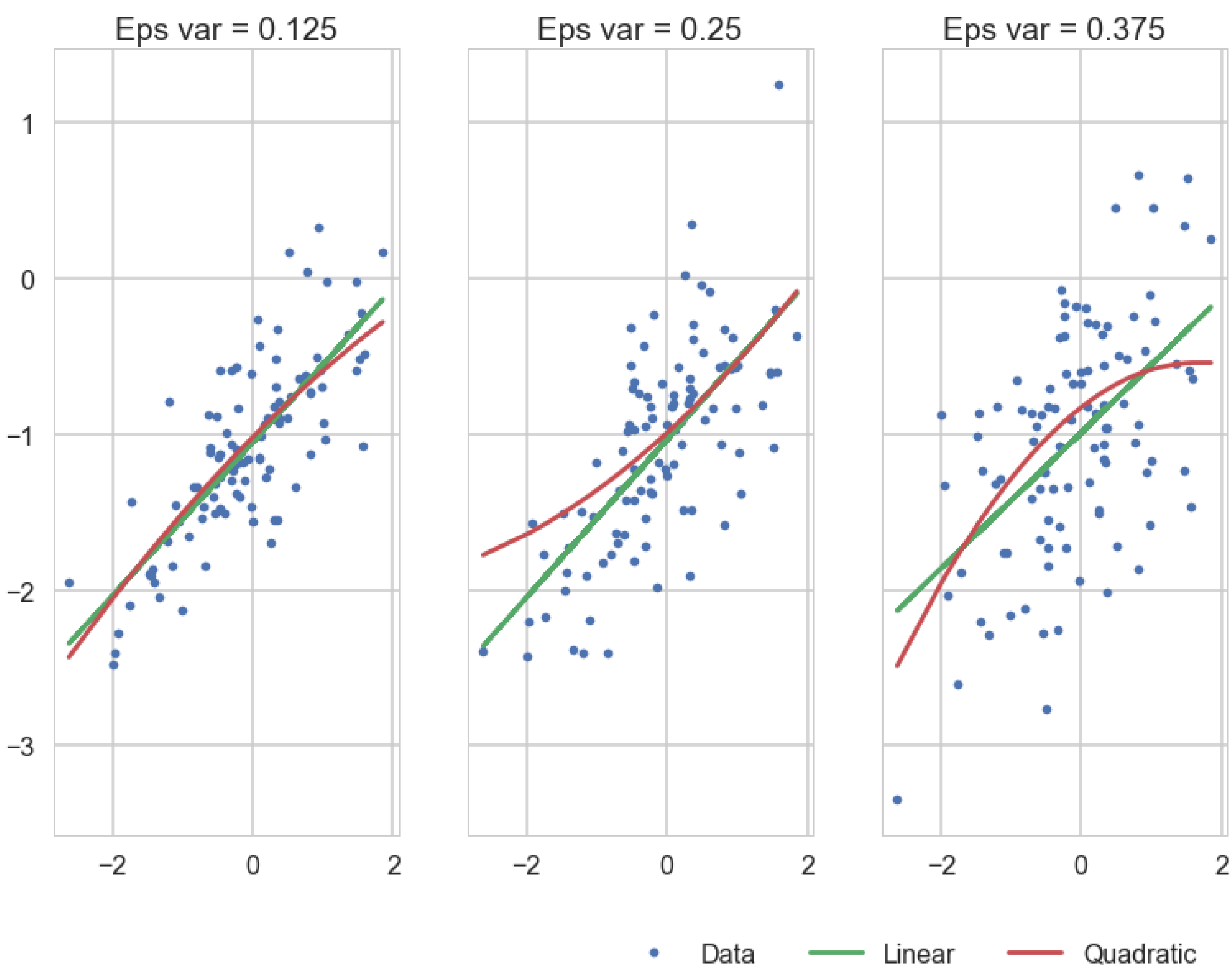
```
In [94]: n = len(Y)

         segments = [[[i, Y[i]]] for i in range(n)]
         lc = LineCollection(segments, zorder=0)
         lc.set_array(np.ones(len(Y)))
         lc.set_linewidths(0.5 * np.ones(n))

         fig = plt.figure()
         plt.plot(x, Y, 'r.', markersize=12)
         plt.plot(x, results.fittedvalues, 'b-')
         plt.gca().add_collection(lc)
         plt.legend(('Data', 'Linear Fit'), loc='lower right')
         plt.title('Linear Best Fit regression')
         plt.show()
```



```
In [95]: var = [.125, .25, .375, .125, .25, .375]
         model = ['Linear', 'Linear', 'Linear', 'Quad', 'Quad', 'Quad']
         df = pd.DataFrame(np.zeros([6, 12]), columns =
                           ['Eps Var', 'Model', 'RSS', 'MSE', 'RSE', 'b1',
                            'b2', 'Int', 'ci [b2,b1,int]',
                           'Std Err b2', 'Std Err b1', 'Std Err b0'])
         df['Eps Var'] = var
         df['Model'] = model
         np.random.seed(seed=31)

         X = np.column_stack(((x)**2, x, np.ones([100,1])))
         xp = np.linspace(min(x), max(x), 100)

         f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, sharex=True)

         for i in df.index:
             v = df['Eps Var'][i]
             eps = np.random.normal(0,np.sqrt(v), 100)
             Y = -1 + 0.5*x + eps
             if 'Linear' in df['Model'][i]:
                 m = sm.OLS(Y,sm.add_constant(x)).fit()
                 df.set_value(i, ['b1', 'Int'], m.params)
                 df.set_value(i, ['RSS'], np.round(m.ssr, 4))
                 df.set_value(i, ['MSE'], np.round(m.mse_resid, 4))
                 df.set_value(i, ['RSE'], np.round(np.sqrt(1/len(Y-2)*m.ssr), 4))
                 df.set_value(i, ['Std Err b1', 'Std Err b0'], np.round(m.bse, 4 ))
                 df.set_value(i, ['ci [b2,b1,int]'], str(np.round(m.conf_int(), 4)))

                 if v == .375:
                     if i < 3:
                         ax3.plot(x, Y, '.', label='Data')
                     ax3.plot(x, m.fittedvalues, '-', label='Linear')
                 elif v == .25 and i < 3:
                     if i < 3:
                         ax2.plot(x, Y, '.', label='Data')
                     ax2.plot(x, m.fittedvalues, '-', label='Linear')
                 else:
                     if i < 3:
                         ax1.plot(x, Y, '.', label='Data')
                     ax1.plot(x, m.fittedvalues, '-', label='Linear')

             else:
                 t = pd.DataFrame([x, Y], index=['x', 'Y'])
                 t = t.transpose()
                 m = smf.ols('Y~x+np.power(x,2)', data = t).fit()
                 df.set_value(i, ['b2','b1', 'Int'], np.round(np.array(m.params), 4))
                 df.set_value(i, ['RSS'], np.round(m.ssr, 4))
                 df.set_value(i, ['MSE'], np.round(m.mse_resid, 4))
                 df.set_value(i, ['RSE'], np.round(np.sqrt(1/len(Y-2)*m.ssr), 4))
```

```
            df.set_value(i, ['Std Err b2', 'Std Err b1', 'Std Err b0'], np.round
            df.set_value(i, ['ci [b2,b1,int]'], str(np.array(m.conf_int())))
            y_pred = m.params[0]*(x**2) + m.params[1]*x + m.params[2]
            if v == .375:
                l3 = ax3.plot(np.sort(x), np.sort(m.fittedvalues), label='Quadrat
            elif v == .25:
                l3 = ax2.plot(np.sort(x), np.sort(m.fittedvalues), label='Quadrat
            else:
                l3 = ax1.plot(np.sort(x), np.sort(m.fittedvalues), label='Quadrat

    handles, labels = ax3.get_legend_handles_labels()
    f.legend(handles, labels, loc = (0.5, 0), ncol=5 )
    ax1.set_title('Eps var = 0.125')
    ax2.set_title('Eps var = 0.25')
    ax3.set_title('Eps var = 0.375')
```

```
Out[95]: <matplotlib.text.Text at 0x1165795f8>
```

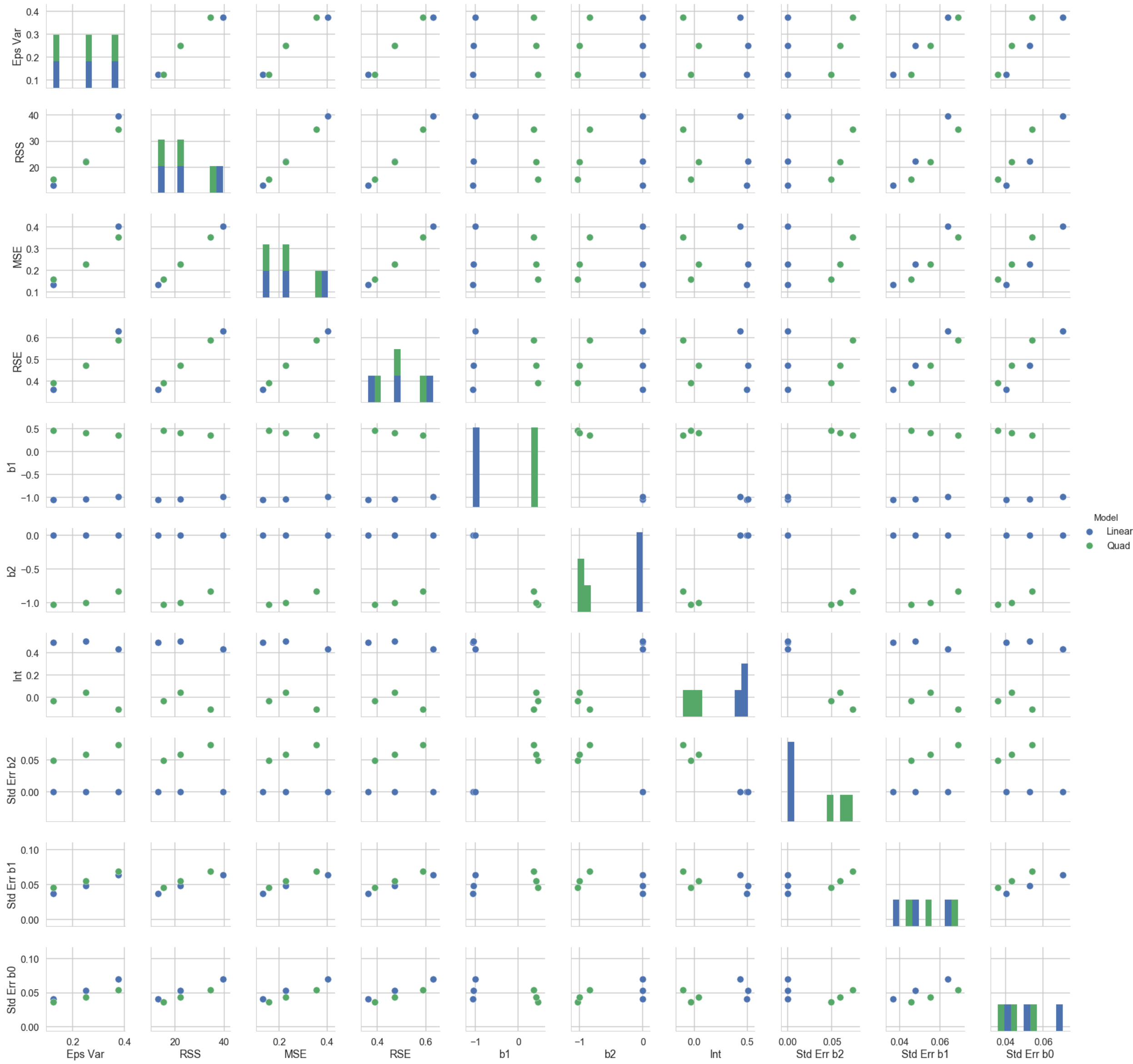



```
In [96]: print(df.head())
```

```
   Eps Var   Model      RSS     MSE     RSE        b1      b2       Int  \
0    0.125  Linear  13.0350  0.1330  0.3610 -1.052466  0.0000  0.493682
```

```
1    0.250  Linear  22.1979  0.2265  0.4711 -1.038366  0.0000  0.505778
2    0.375  Linear  39.5935  0.4040  0.6292 -0.994353  0.0000  0.435533
3    0.125    Quad  15.1892  0.1566  0.3897  0.456200 -1.0220 -0.031400
4    0.250    Quad  22.1001  0.2278  0.4701  0.410800 -0.9949  0.042700

                                    ci [b2,b1,int]  Std Err b2  Std Err b1  \
0            [[-1.1253 -0.9796]\n [ 0.4136  0.5738]]      0.0000      0.0367
1            [[-1.1334 -0.9433]\n [ 0.4013  0.6103]]      0.0000      0.0479
2            [[-1.1213 -0.8674]\n [ 0.2959  0.5751]]      0.0000      0.0640
3  [[-1.11911219 -0.92492341]\n [ 0.36543969  0.5...      0.0489      0.0457
4  [[-1.11204493 -0.87780916]\n [ 0.30127729  0.5...      0.0590      0.0552

   Std Err b0
0      0.0404
1      0.0527
2      0.0703
3      0.0360
4      0.0434
```

```
In [97]: sns.pairplot(df, hue='Model')

Out[97]: <seaborn.axisgrid.PairGrid at 0x116c3d860>
```



This has generated fairly expected resaults. When the varience of eplison decreased, linear models tended to preform better. This is indicated by the RSS, MSE, and RSE. Interestingly, the beta values seem to cluser based on algorithm chosen as to standard errors. Unsurpsisingly, the linear model performs better wiht a lower variance.

```
In [98]: print(df[['Eps Var', 'Model', 'ci [b2,b1,int]', 'Std Err b2', 'Std Err b1',

   Eps Var   Model                                         ci [b2,b1,int]  \
0    0.125  Linear           [[-1.1253 -0.9796]\n [ 0.4136  0.5738]]
1    0.250  Linear           [[-1.1334 -0.9433]\n [ 0.4013  0.6103]]
2    0.375  Linear           [[-1.1213 -0.8674]\n [ 0.2959  0.5751]]
3    0.125    Quad  [[-1.11911219 -0.92492341]\n [ 0.36543969  0.5...
4    0.250    Quad  [[-1.11204493 -0.87780916]\n [ 0.30127729  0.5...
5    0.375    Quad  [[ -9.78837453e-01  -6.86487584e-01]\n [  2.14...

   Std Err b2  Std Err b1  Std Err b0
0      0.0000      0.0367      0.0404
```

```
1       0.0000      0.0479      0.0527
2       0.0000      0.0640      0.0703
3       0.0489      0.0457      0.0360
4       0.0590      0.0552      0.0434
5       0.0737      0.0688      0.0542
```

1.6.1 Confidence intervals:

The confidence intervals appears smaller on models with lower error. This is expected as the model is more likely to contain the true value of the data if there is less varience. To minimize confidence intervals, our sample size would need to increase substantially. This is indicated by the Central Limit Theory Proof.

1.7 Question 6

```
In [99]: df = pd.read_csv('Auto.csv')
```

```
In [100]: print(df.head())
```

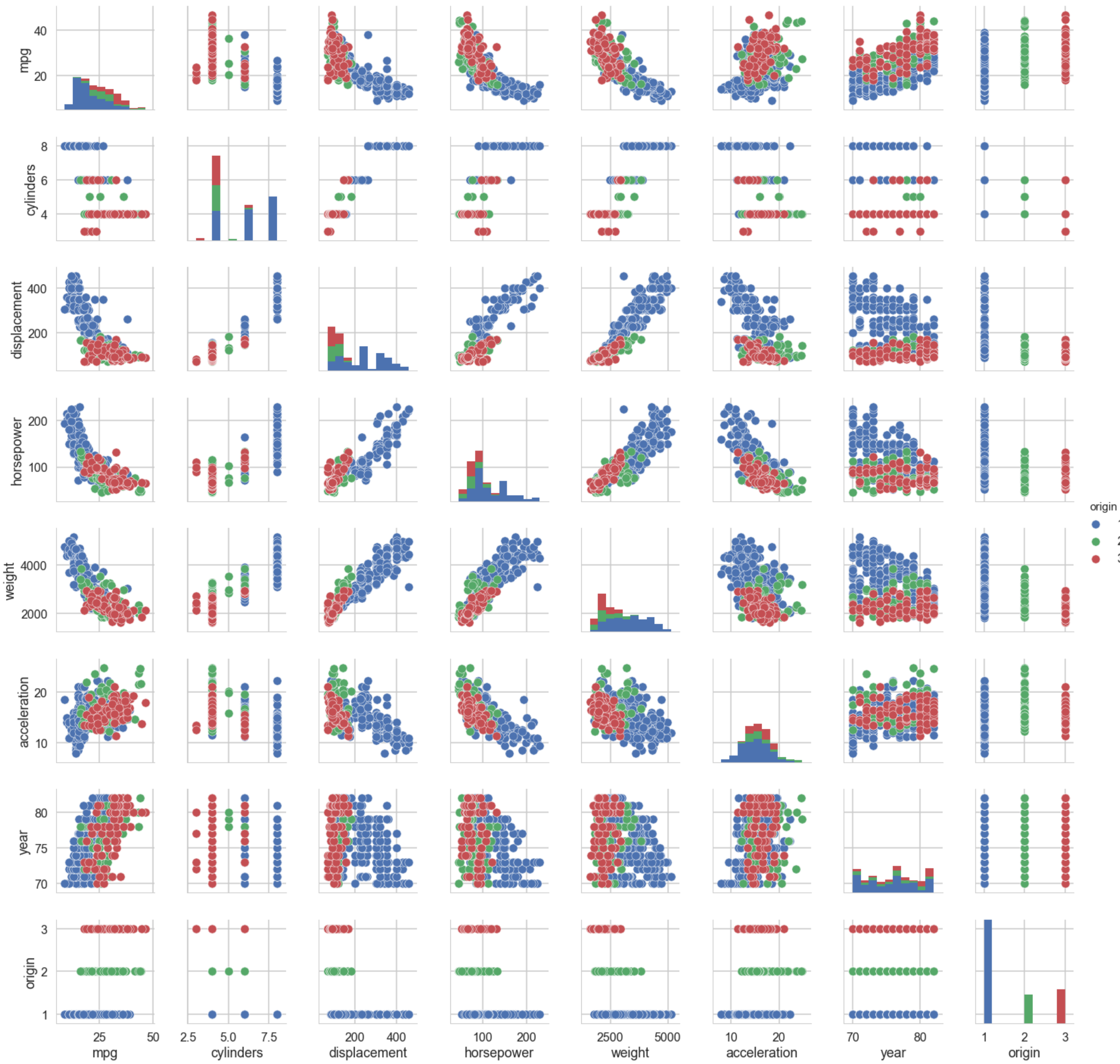
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0    70
1  15.0          8         350.0         165    3693          11.5    70
2  18.0          8         318.0         150    3436          11.0    70
3  16.0          8         304.0         150    3433          12.0    70
4  17.0          8         302.0         140    3449          10.5    70

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

```
In [101]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
mpg             392 non-null float64
cylinders       392 non-null int64
displacement    392 non-null float64
horsepower      392 non-null int64
weight          392 non-null int64
acceleration    392 non-null float64
year            392 non-null int64
origin          392 non-null int64
name            392 non-null object
dtypes: float64(3), int64(5), object(1)
memory usage: 27.6+ KB
```

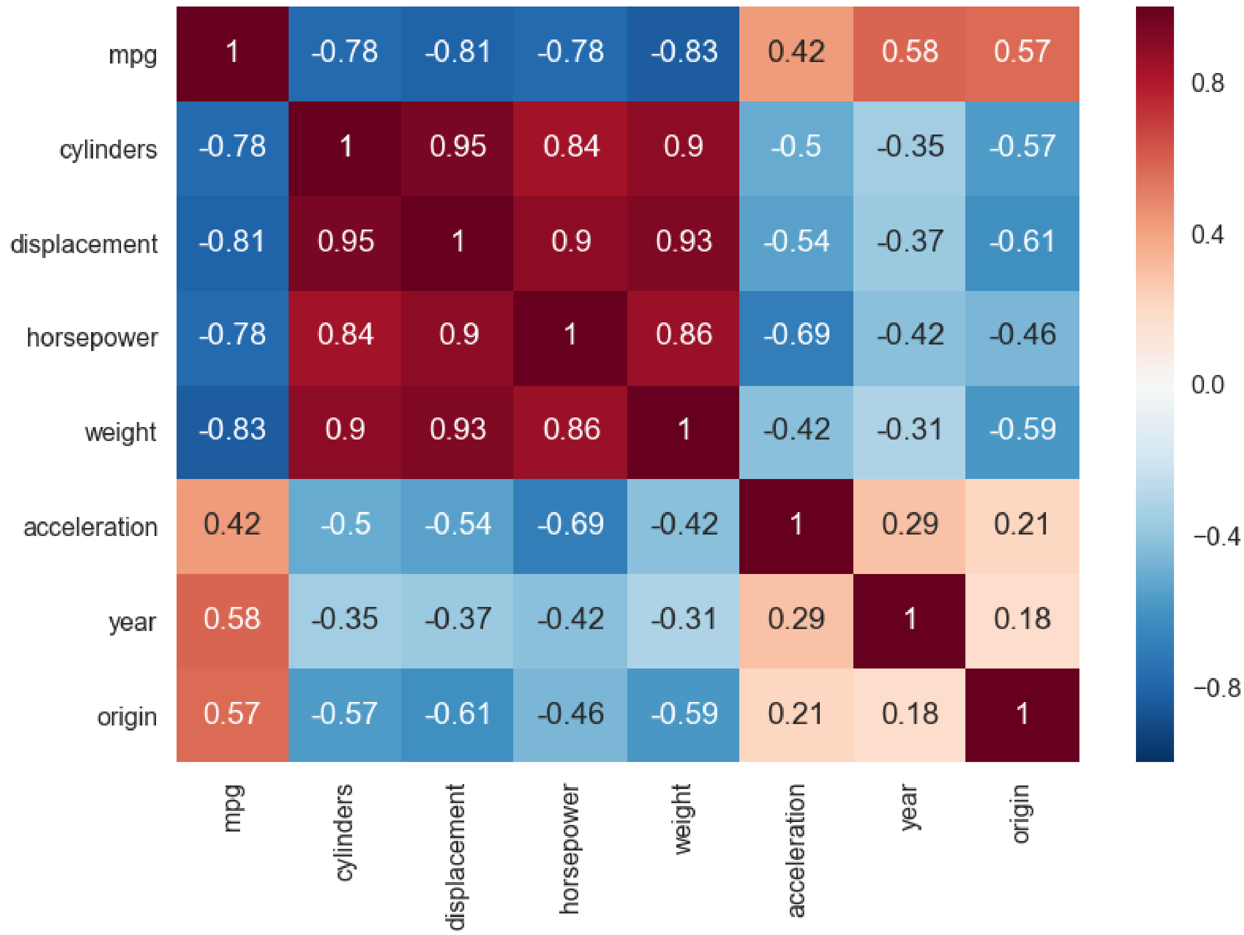
```
In [102]: sns.pairplot(df, hue='origin')
```

Out[102]: <seaborn.axisgrid.PairGrid at 0x11bda7ef0>




```
In [103]: p = sns.heatmap(df.corr(), annot=True)
          plt.xticks(rotation=90)
```

```
Out[103]: (array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5]),
           <a list of 8 Text xticklabel objects>)
```




```
In [108]: res = smf.ols(formula='mpg ~ cylinders + displacement + horsepower'
                        + '+ weight + acceleration + year + origin',
                       data=df).fit()
          print(res.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     252.4
Date:                Fri, 29 Sep 2017   Prob (F-statistic):          2.04e-139
Time:                        11:25:51   Log-Likelihood:                -1023.5
No. Observations:                 392   AIC:                             2063.
Df Residuals:                     384   BIC:                             2095.
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept      -17.2184      4.644     -3.707      0.000     -26.350      -8.087
cylinders       -0.4934      0.323     -1.526      0.128      -1.129       0.142
displacement     0.0199      0.008      2.647      0.008       0.005       0.035
```

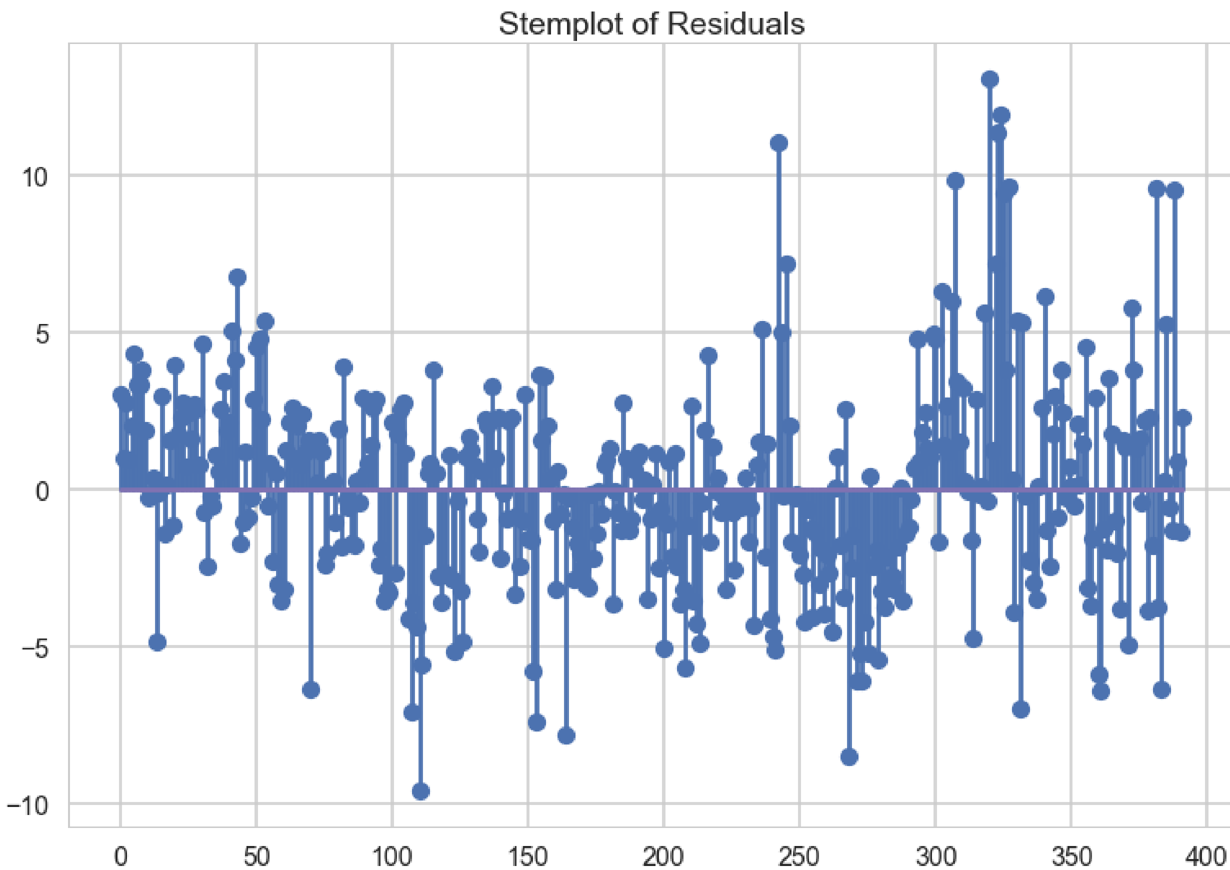
```
horsepower      -0.0170      0.014     -1.230      0.220      -0.044       0.010
weight          -0.0065      0.001     -9.929      0.000      -0.008      -0.005
acceleration     0.0806      0.099      0.815      0.415      -0.114       0.275
year             0.7508      0.051     14.729      0.000       0.651       0.851
origin           1.4261      0.278      5.127      0.000       0.879       1.973
==============================================================================
Omnibus:                       31.906   Durbin-Watson:                   1.309
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.100
Skew:                           0.529   Prob(JB):                     2.95e-12
Kurtosis:                       4.460   Cond. No.                     8.59e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
[2] The condition number is large, 8.59e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [109]: RSS = np.round(res.ssr, 4)
          RSE = np.round(np.sqrt(1/res.df_resid*RSS), 4)
          MSE = np.round(res.mse_resid, 4)
          print('RSS: ' + str(RSS))
          print('RSE: ' + str(RSE))
          print("MSE: " +  str(MSE))
```

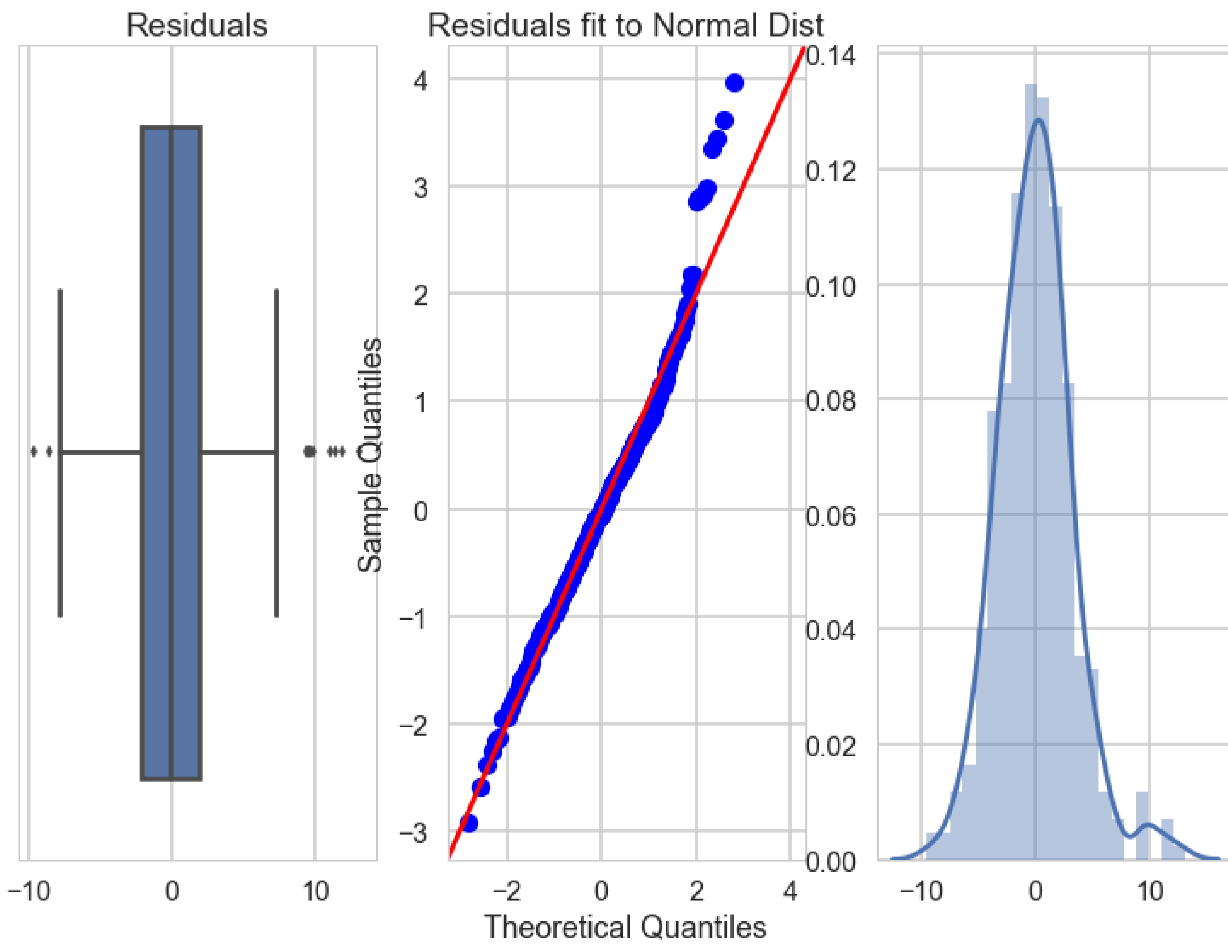
```
RSS: 4252.2125
RSE: 3.3277
MSE: 11.0735
```

There does appear to be a statistical correlation between the factors and mpg. More specifically, the intercept, weight, displacement, year, and origin have the largest impact. This data is highly unlikely to have ocurred randomly. Logically, as regulations have required car companies to increase their average MPG, year and has a large impact on MPG. Holding everything else constant, MPG increases by 0.75 for year. I find it interesting that origin also has a significant impact. Given the key, it shows that the rest of the world (not America) is much more efficient than the US. Japan and the EU have much higher MPG requirements.

```
In [110]: plt.stem(res.resid)
          plt.title('Stemplot of Residuals')

Out[110]: <matplotlib.text.Text at 0x12093c898>
```




```
In [111]: fig, [ax1, ax2, ax3] = plt.subplots(1, 3)
          sns.boxplot(res.resid, ax=ax1)
          ax1.set_title('Residuals')
          ax2.set_title('Residuals fit to Normal Dist')
          sm.qqplot(res.resid, line='45', fit=True, ax=ax2);
          sns.distplot(res.resid, ax=ax3)

Out[111]: <matplotlib.axes._subplots.AxesSubplot at 0x121744ba8>
```

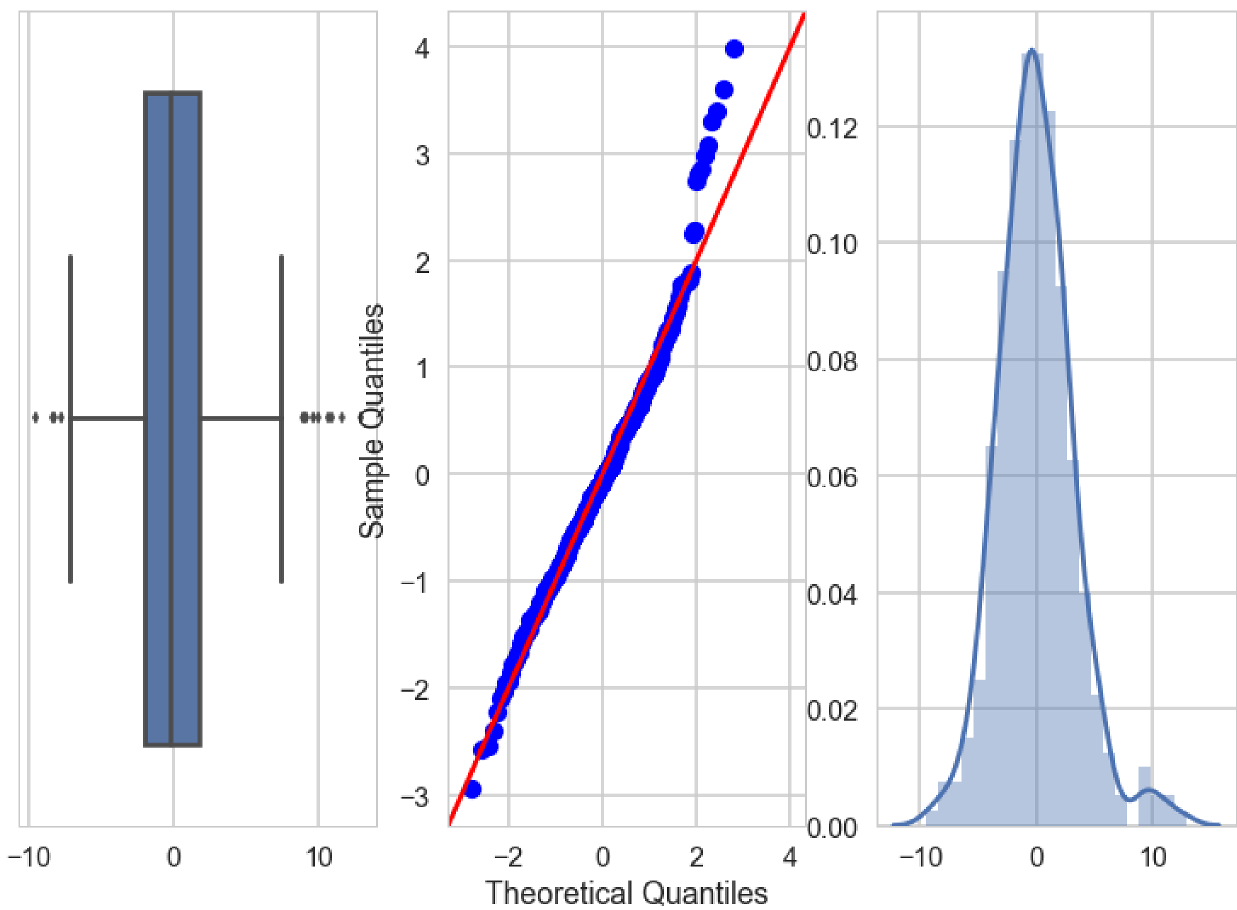


It appears that there are outliers. The majority of the model's residuals appear to fit well to a normal model (as expected) until we reach the higher and lower quartiles. The Q-Q plot is fitted to a normal model.

```
In [112]: resid = res.outlier_test()
          resid[resid['unadj_p']<0.05].count()

Out[112]: student_resid    18
          unadj_p          18
          bonf(p)          18
          dtype: int64
```

1.7.1 Alter Variables by sqrt

```
In [113]: sqr = smf.ols(formula='mpg ~ cylinders + displacement + np.sqrt(horsepower)
                           '+ weight + acceleration + year + origin',
                          data=df).fit()
          fig, [ax1, ax2, ax3] = plt.subplots(1, 3)
          sns.boxplot(sqr.resid, ax=ax1)
          sm.qqplot(sqr.resid, line='45', fit=True, ax=ax2);
          sns.distplot(sqr.resid, ax=ax3)

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x121a9dd68>
```




```
In [114]: print(sqr.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.827
Model:                            OLS   Adj. R-squared:                  0.824
Method:                 Least Squares   F-statistic:                     262.0
Date:                Fri, 29 Sep 2017   Prob (F-statistic):          5.83e-142
Time:                        11:25:55   Log-Likelihood:                -1017.5
No. Observations:                 392   AIC:                             2051.
Df Residuals:                     384   BIC:                             2083.
Df Model:                           7
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.97
---------------------------------------------------------------------------------------
Intercept              -6.0374      5.546     -1.089      0.277     -16.942       4.8
cylinders              -0.5223      0.317     -1.649      0.100      -1.145       0.1
displacement            0.0221      0.007      3.064      0.002       0.008       0.0
np.sqrt(horsepower)    -1.1435      0.311     -3.672      0.000      -1.756      -0.5
weight                 -0.0055      0.001     -7.979      0.000      -0.007      -0.0
acceleration           -0.1021      0.104     -0.983      0.326      -0.306       0.1
year                    0.7240      0.050     14.429      0.000       0.625       0.8
```

```
origin                     1.5173      0.270      5.612      0.000      0.986      2.0
==============================================================================
Omnibus:                       32.516   Durbin-Watson:                   1.343
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.556
Skew:                           0.542   Prob(JB):                     2.35e-12
Kurtosis:                       4.451   Cond. No.                     1.04e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
[2] The condition number is large, 1.04e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Sqrt does appear to group residuals well, but does not minimize the number of outliers as evident by the distplot. The transformed variable remains significant and the scale of the coefficient changes as the units shrink. This is fairly expected. I would be interested in understanding what changes the outliers cause.

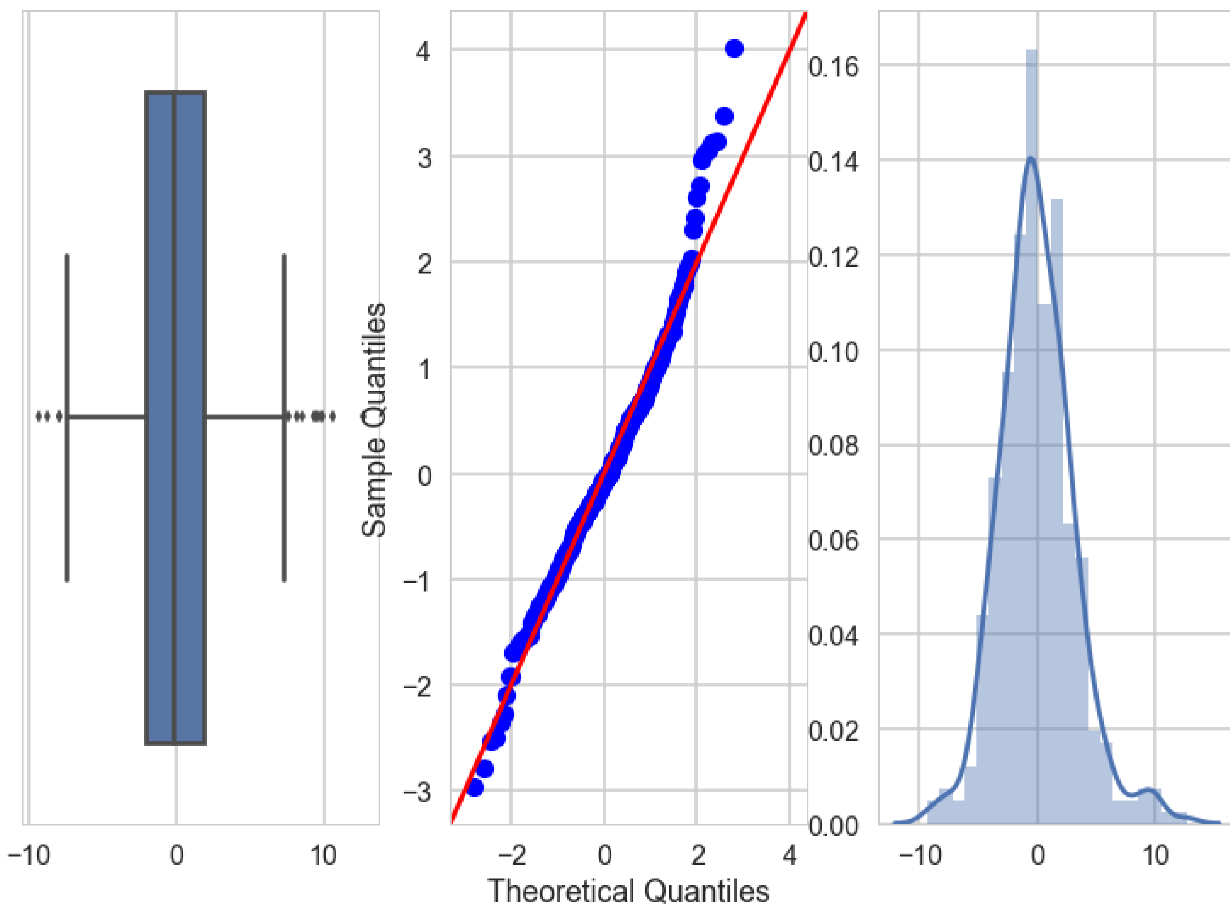
1.7.2 Alter Variables by log

```
In [115]: df.columns
```

```
Out[115]: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                 'acceleration', 'year', 'origin', 'name'],
                dtype='object')
```

```
In [116]: log = smf.ols(formula='mpg ~ cylinders + displacement + np.log(horsepower)'
                        '+ weight + acceleration + year + origin',
                       data=df).fit()
          fig, [ax1, ax2, ax3] = plt.subplots(1, 3)
          sns.boxplot(log.resid, ax=ax1)
          sm.qqplot(log.resid, line='45', fit=True, ax=ax2);
          sns.distplot(log.resid, ax=ax3)
```

```
Out[116]: <matplotlib.axes._subplots.AxesSubplot at 0x121c9b7f0>
```




```
In [117]: print(log.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.837
Model:                            OLS   Adj. R-squared:                  0.834
Method:                 Least Squares   F-statistic:                     281.6
Date:                Fri, 29 Sep 2017   Prob (F-statistic):          5.80e-147
Time:                        11:25:56   Log-Likelihood:                -1005.7
No. Observations:                 392   AIC:                             2027.
Df Residuals:                     384   BIC:                             2059.
Df Model:                           7
Covariance Type:            nonrobust
=================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975
---------------------------------------------------------------------------------
Intercept            27.2540      8.590      3.173      0.002      10.365      44.14
cylinders            -0.4862      0.307     -1.585      0.114      -1.089       0.11
displacement          0.0195      0.007      2.830      0.005       0.006       0.03
np.log(horsepower)   -9.5064      1.540     -6.175      0.000     -12.534      -6.47
weight               -0.0043      0.001     -6.148      0.000      -0.006      -0.00
acceleration         -0.2921      0.104     -2.814      0.005      -0.496      -0.08
year                  0.7053      0.048     14.556      0.000       0.610       0.80
```

```
origin                     1.4824      0.259      5.716      0.000       0.973       1.992
==============================================================================
Omnibus:                       29.129   Durbin-Watson:                   1.420
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               46.738
Skew:                           0.501   Prob(JB):                     7.10e-11
Kurtosis:                       4.362   Cond. No.                     1.68e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
[2] The condition number is large, 1.68e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
In [118]: resid = pd.DataFrame(log.outlier_test())

In [119]: outlier = resid[resid['unadj_p']<0.05]

In [120]: outlier.count()

Out[120]: student_resid    21
          unadj_p          21
          bonf(p)          21
          dtype: int64
```

Log improves the shape of the distplot, but does not minimize the number of outliers. Log acually increased the number of outliers in the residual by 3. Unfortunatly, I have not been able to find anything that accurately reduces residuals into a normal distribution. I think that there are several large outliers on many fronts that cause this, instead of one feature in all cars. For example a Tesla (Which is not in the set) would cause a huge outlier due to its features. The residual on that car would be substantial and I could not fix it with a transformation.

1.7.3 What Effect Do the ouliers cause?

```
In [121]: df.columns

Out[121]: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                 'acceleration', 'year', 'origin', 'name'],
                dtype='object')

In [122]: df.iloc[outlier.index]['name']

Out[122]: 53                            datsun 1200
          110                             maxda rx3
          152                       mercury monarch
          153                          ford maverick
          164                       ford mustang ii
          242      volkswagen rabbit custom diesel
          245                         datsun b210 gx
          268                toyota celica gt liftback
          307                              vw rabbit
          318                    datsun 510 hatchback
          320                              mazda glc
```

```
322                           datsun 210
323                   vw rabbit c (diesel)
324                     vw dasher (diesel)
325                    audi 5000s (diesel)
327                    honda civic 1500 gl
330                          datsun 280-zx
331                          mazda rx-7 gs
361                  chrysler lebaron salon
381     oldsmobile cutlass ciera (diesel)
388                              vw pickup
Name: name, dtype: object
```

In [123]:
```
df.iloc[outlier.index].describe()
```

Out[123]:

	mpg	cylinders	displacement	horsepower	weight
count	21.000000	21.000000	21.000000	21.000000	21.000000
mean	32.866667	4.619048	135.952381	77.476190	2477.333333
std	11.719571	1.244033	74.409325	23.344419	542.422652
min	13.000000	3.000000	70.000000	48.000000	1613.000000
25%	21.100000	4.000000	86.000000	65.000000	2110.000000
50%	37.000000	4.000000	97.000000	72.000000	2335.000000
75%	43.100000	6.000000	168.000000	90.000000	2950.000000
max	46.600000	8.000000	302.000000	132.000000	3465.000000

	acceleration	year	origin
count	21.000000	21.000000	21.000000
mean	17.471429	78.476190	2.238095
std	3.831337	2.976895	0.830949
min	11.400000	71.000000	1.000000
25%	14.700000	78.000000	2.000000
50%	17.900000	80.000000	2.000000
75%	19.900000	80.000000	3.000000
max	24.600000	82.000000	3.000000

In [124]:
```
df[~df.isin(outlier)].describe()
```

Out[124]:

	mpg	cylinders	displacement	horsepower	weight
count	392.000000	392.000000	392.000000	392.000000	392.000000
mean	23.445918	5.471939	194.411990	104.469388	2977.584184
std	7.805007	1.705783	104.644004	38.491160	849.402560
min	9.000000	3.000000	68.000000	46.000000	1613.000000
25%	17.000000	4.000000	105.000000	75.000000	2225.250000
50%	22.750000	4.000000	151.000000	93.500000	2803.500000
75%	29.000000	8.000000	275.750000	126.000000	3614.750000
max	46.600000	8.000000	455.000000	230.000000	5140.000000

	acceleration	year	origin
count	392.000000	392.000000	392.000000
mean	15.541327	75.979592	1.576531
std	2.758864	3.683737	0.805518
min	8.000000	70.000000	1.000000
25%	13.775000	73.000000	1.000000
50%	15.500000	76.000000	1.000000

```
75%        17.025000   79.000000    2.000000
max        24.800000   82.000000    3.000000
```

This result is truely surprising. The outliers acutally have a much better fuel efficiency than the rest of the cars - almost 40% more. This tells me that the outliers are the cars that we should all be driving. The majority of them are foreign made, and more environmentally concious. The accelerations, displacement, and horsepower is lower in the rest of the cars in comparison to outliers. This indicates that these cars are meant for fuel efficiency. Surprisingly, this is not the case. Instead, the outliers are mostly diesel cars; diesel has higher energy density than gasoline, which results in less fuel being burned to go the same distance. To accurately run regressions, I would want to control for fuel type; therfore, I will create a new column to control for gas type. 1 = diesel, 0 = gas

```
In [125]: df['fuel'] = df['name'].apply(lambda x : int('diesel' in x))
```

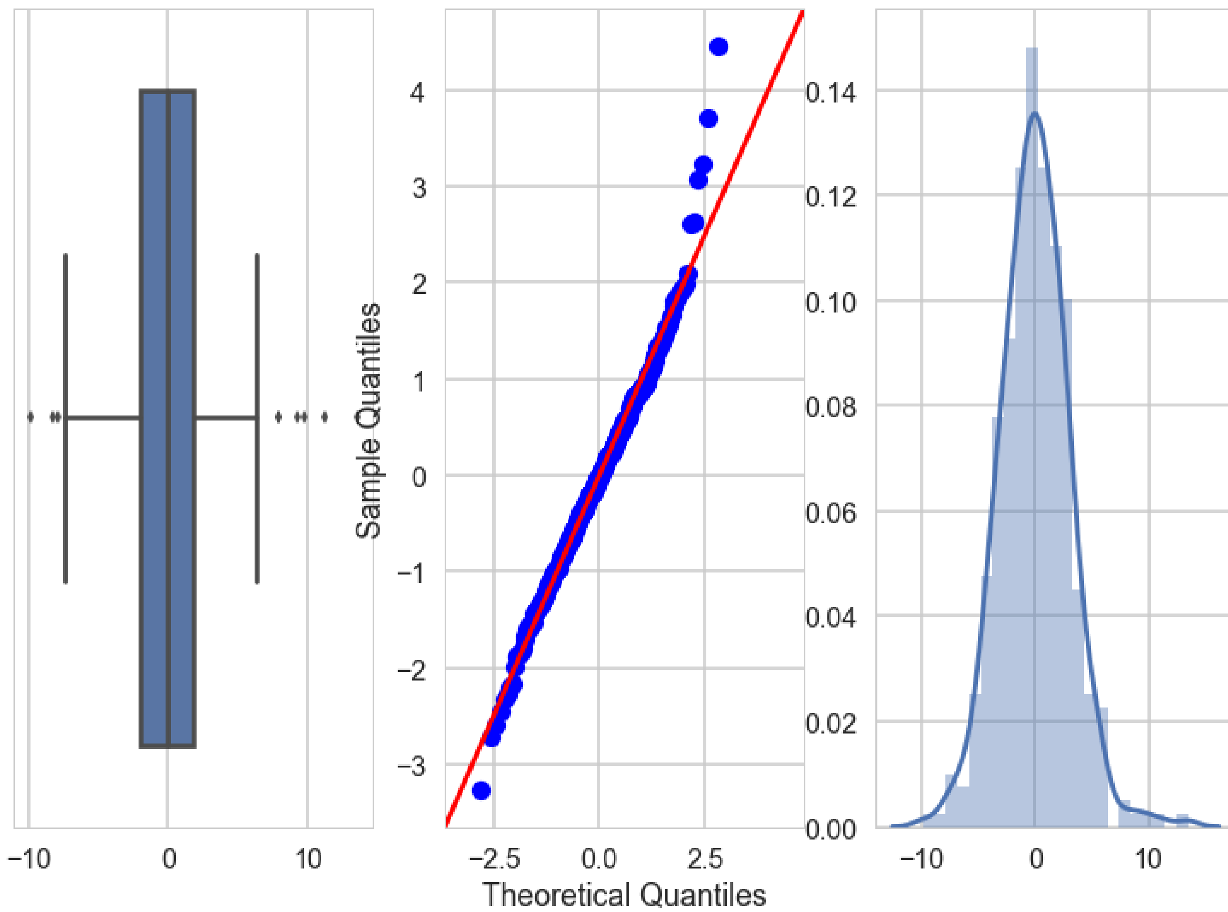
```
In [127]: gas = smf.ols(formula='mpg ~ cylinders + displacement + horsepower' +
                        '+ weight + acceleration + year + origin+fuel',
                       data=df).fit()
          print(gas.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.845
Model:                            OLS   Adj. R-squared:                  0.841
Method:                 Least Squares   F-statistic:                     260.5
Date:                Fri, 29 Sep 2017   Prob (F-statistic):         8.93e-150
Time:                        11:26:06   Log-Likelihood:                -996.11
No. Observations:                 392   AIC:                             2010.
Df Residuals:                     383   BIC:                             2046.
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept      -11.8865      4.394     -2.705      0.007     -20.525      -3.248
cylinders       -0.5577      0.302     -1.847      0.066      -1.151       0.036
displacement     0.0183      0.007      2.607      0.009       0.005       0.032
horsepower      -0.0151      0.013     -1.171      0.242      -0.040       0.010
weight          -0.0065      0.001    -10.599      0.000      -0.008      -0.005
acceleration    -0.0397      0.094     -0.424      0.672      -0.224       0.144
year             0.7104      0.048     14.833      0.000       0.616       0.805
origin           1.3239      0.260      5.091      0.000       0.813       1.835
fuel             9.3625      1.236      7.575      0.000       6.932      11.793
==============================================================================
Omnibus:                       18.032   Durbin-Watson:                   1.354
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               33.944
Skew:                           0.260   Prob(JB):                     4.26e-08
Kurtosis:                       4.344   Cond. No.                     8.71e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
[2] The condition number is large, 8.71e+04. This might indicate that there are
```

```
strong multicollinearity or other numerical problems.
```

My thesis was correct. MPG is highly dependent on fuel type. Transforming this column:

```
In [131]: fuel = smf.ols(formula='mpg ~ cylinders + displacement + np.sqrt(horsepower
                      '+ weight + acceleration + year + origin + np.sqrt(fuel)',
                    data=df).fit()
          fig, [ax1, ax2, ax3] = plt.subplots(1, 3)
          sns.boxplot(fuel.resid, ax=ax1)
          sm.qqplot(fuel.resid, line='45', fit=True, ax=ax2);
          sns.distplot(fuel.resid, ax=ax3)

Out[131]: <matplotlib.axes._subplots.AxesSubplot at 0x1220b2c50>
```




```
In [132]: resid = fuel.outlier_test()

In [133]: resid[resid['unadj_p']<0.05].count()

Out[133]: student_resid    17
          unadj_p          17
          bonf(p)          17
          dtype: int64
```

It appears that fuel type is generating some of the outliers. By controlling for this, I was able to decrease the number of outliers by 4 from log transformation and 1 from the control regression. This distplot appears to have a softer tail on the right side, which indicates that the controlling for fuel is critical for normalizing outlisers. Ultimately, I don't believe that transformations of the initial variables are the best way to normalize residuals. I believe that there are other confounding characteristics that cause these non-normal residual plots such as fuel type (possibly hybrids, electric as well).

```
In [ ]:
```