UNIVERSITY OF PENNSYLVANIA

ESE 305: Homework 2

Braden Fineberg
ESE 305: Introduction to Machine Learning
September 27, 2017

1 Conceptual

1.1 Statistical Significance

Given the statistical information in the table provided, several conclusions can be drawn. The incercept, 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 unlikley 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

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

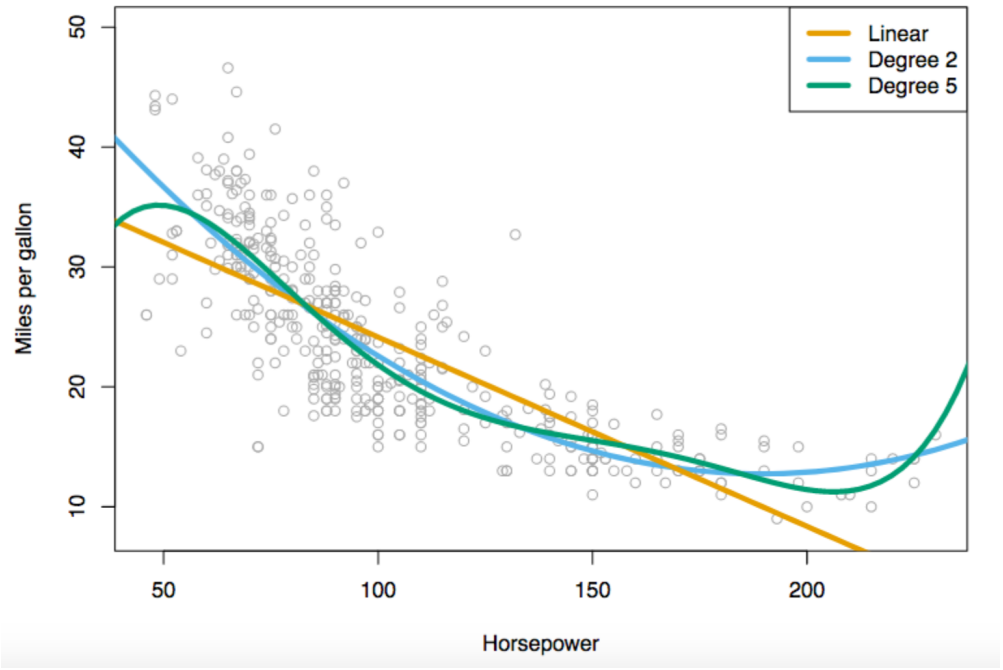
1.2.3

False. This likely indicates that because there is a such a strong correlation, the salary does not need to depend on both. For example, someone with a high GPA likely has a high IQ. There is some an effect for which the model needs to account, but it should be minimal (ie. $\hat{\beta}_4 = 0.01$). They are each contributing factors, but we don't want to double count the effect. In addition, I would need to look at the p-Value of this term.

1.3 Cubic vs. Linear Regression

1.3.1 Given Linear Training Data

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 difficutlt 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) underfits, the green line (quintic) overfits. The blue line most accurately fits the data. Because this data set is fairly small (1 regressor, 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 visulaization is not possible, the cubic model will have a lower RSS.

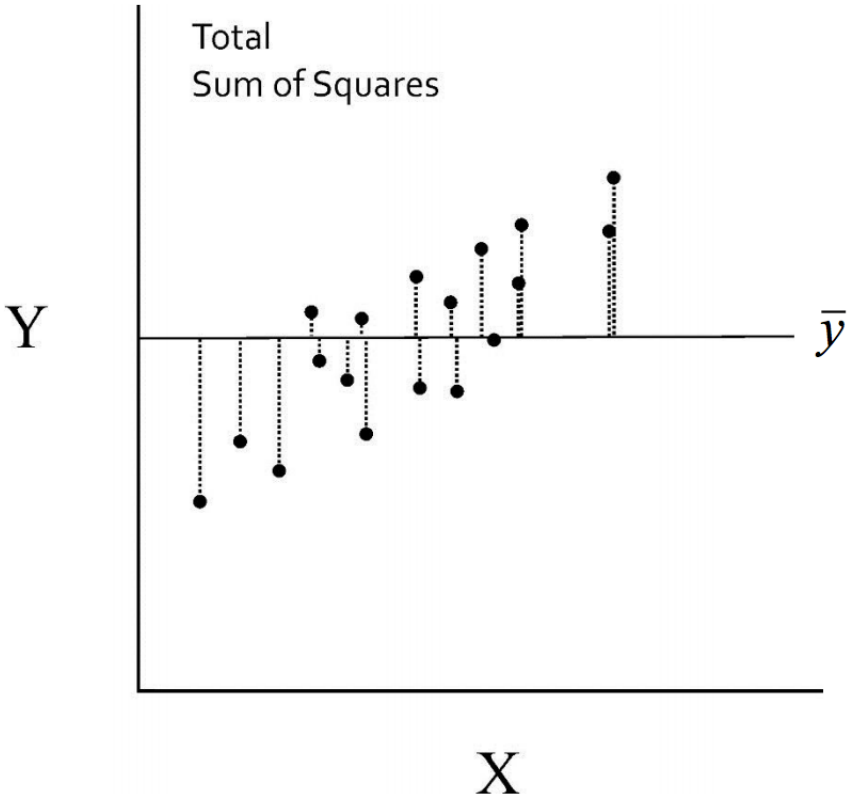
1.3.4 Given Unknown Data

Without viusalizing the underlying data it is hard to tell how this will perform. If the data truely 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 covarience of x and y and normalizing by the inital two 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 mode, 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

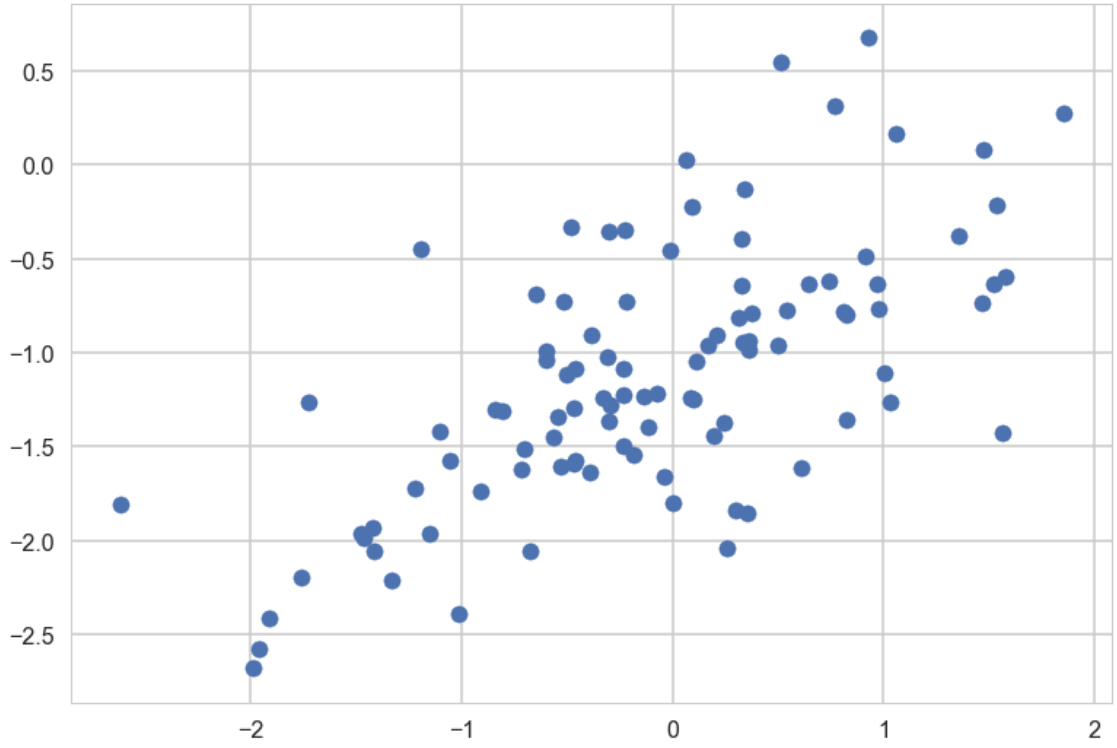
1.6 Fitting linear regression to data

```
In [2]: 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 [3]: np.random.seed(seed=42)
        mu, sigma = 0, 1
        x = np.random.normal(mu, sigma, 100)
```

```
In [4]: np.random.seed(seed=31)
        eps = np.random.normal(0,np.sqrt(0.25), 100)

In [5]: Y = -1 + 0.5*x + eps

In [6]: np.size(Y)

Out[6]: 100
```

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

```
In [7]: plt.scatter(x, Y)

Out[7]: <matplotlib.collections.PathCollection at 0x11a0f7278>
```



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

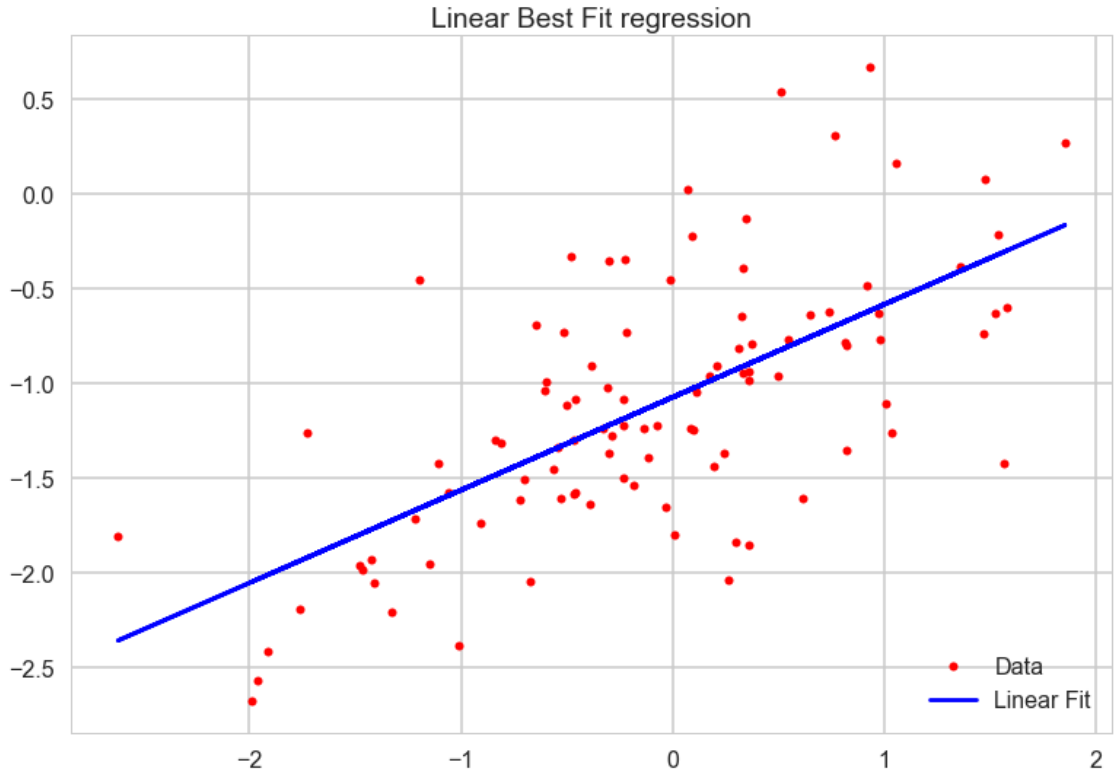
```
In [8]: from sklearn.linear_model import LinearRegression
        from matplotlib.collections import LineCollection
        from sklearn.metrics import mean_squared_error, classification_report
        import statsmodels.api as sm
        import scipy.stats as stats

/Users/Braden/anaconda/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56:
  from pandas.core import datetools
```

```
In [9]: lr = LinearRegression()
        lr.fit(x[:, np.newaxis], Y)  # x needs to be 2d for LinearRegression
        predsLr = lr.predict(x[:, np.newaxis])

In [10]: print("Best Fit: y = " + str(np.round(lr.coef_[0], 3))+"x " +  str(np.rour
         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 [11]: 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:                Mon, 25 Sep 2017   Prob (F-statistic):           1.29e-13
Time:                        12:14:35   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 sp
```

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 [12]: 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()
```

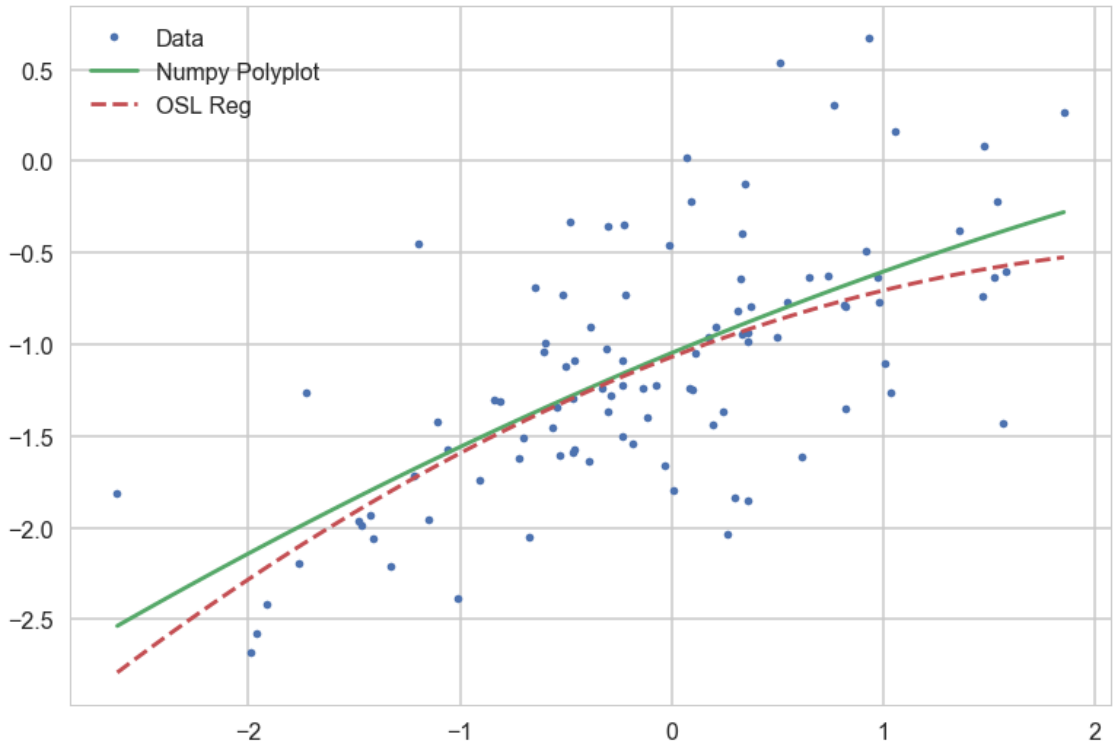



```
In [13]: X = np.column_stack(((x)**2, x))
         X = sm.add_constant(X)
```

```
Beta = np.polyfit(x, Y, 2)
y = np.dot(X, Beta) + eps
res = sm.OLS(y,X).fit()
print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.851
Model:                            OLS   Adj. R-squared:                  0.848
Method:                 Least Squares   F-statistic:                     276.1
Date:                Mon, 25 Sep 2017   Prob (F-statistic):           9.06e-41
Time:                        12:14:35   Log-Likelihood:                -74.399
No. Observations:                 100   AIC:                             154.8
Df Residuals:                      97   BIC:                             162.6
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0814      0.064     -1.274      0.206      -0.208       0.045
x1             0.4441      0.047      9.445      0.000       0.351       0.537
x2            -1.0686      0.060    -17.885      0.000      -1.187      -0.950
==============================================================================
Omnibus:                        4.045   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.132   Jarque-Bera (JB):                3.508
Skew:                           0.448   Prob(JB):                        0.173
Kurtosis:                       3.200   Cond. No.                         2.27
==============================================================================

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

```
In [14]: q = np.poly1d(Beta)
         xp = np.linspace(min(x), max(x), 100)

In [15]: y_pred = res.params[0]*(xp**2) + res.params[1]*xp + res.params[2]

In [16]: plt.plot(x, Y, '.', xp, q(xp), '-', xp, y_pred, '--')
         plt.legend(['Data', 'Numpy Polyplot', 'OSL Reg'])

Out[16]: <matplotlib.legend.Legend at 0x11ea71ef0>
```

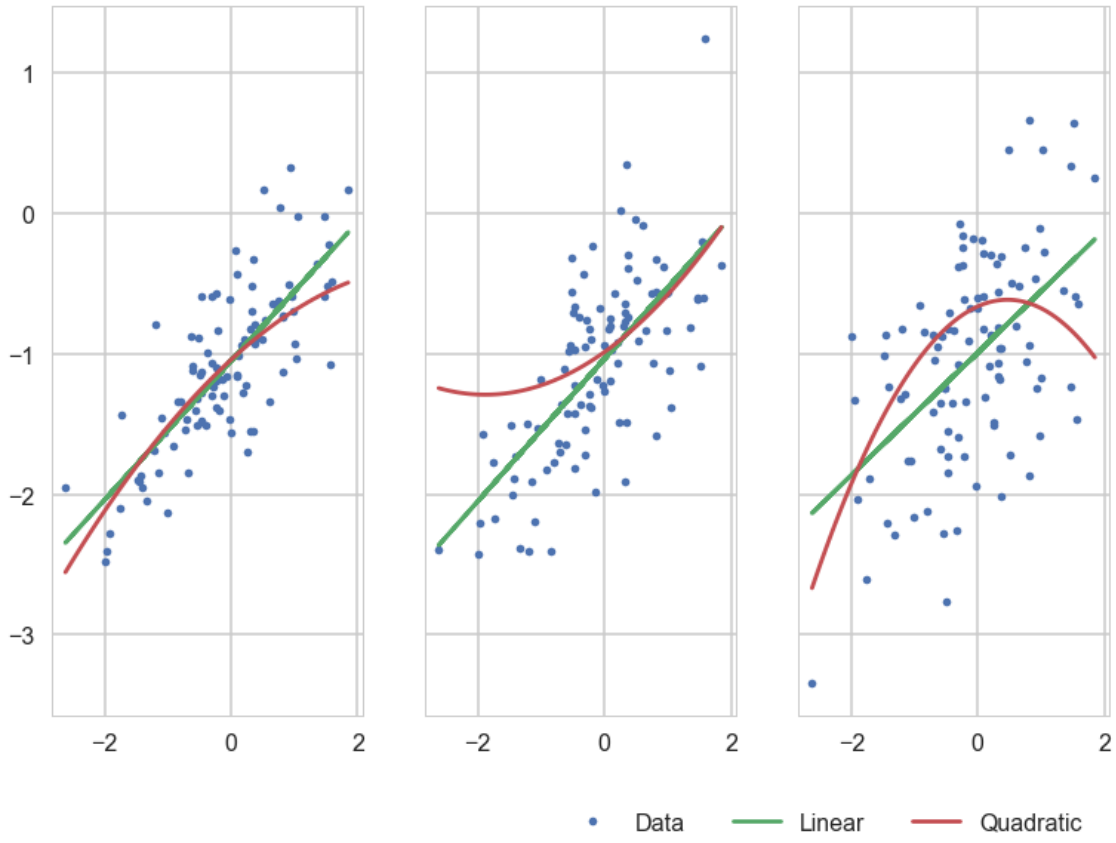


****Why are these values diffrent?

```
In [17]: 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', 'I
                           '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:
                Beta = np.polyfit(x, Y, 2)
                y = np.dot(X, Beta) + eps
                m = sm.OLS(y,X).fit()
                df.set_value(i, ['b2','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 b2', 'Std Err b1', 'Std Err b0'], np.rou
                df.set_value(i, ['ci [b2,b1,int]'], str(np.round(m.conf_int(), 4))
                y_pred = m.params[0]*(xp**2) + m.params[1]*xp + m.params[2]
                if v == .375:
                    l3 = ax3.plot(xp, y_pred, label='Quadratic')
                elif v == .25:
                    l3 = ax2.plot(xp, y_pred, label='Quadratic')
                else:
                    l3 = ax1.plot(xp, y_pred, label='Quadratic')

        handles, labels = ax3.get_legend_handles_labels()
        f.legend(handles, labels, loc = (0.5, 0), ncol=5 )
```

Out[17]: <matplotlib.legend.Legend at 0x11f094860>



In [18]: df

Out[18]:

	Eps Var	Model	RSS	MSE	RSE	b1	b2	Int
0	0.125	Linear	13.0350	0.1330	0.3610	-1.052466	0.000000	0.493682
1	0.250	Linear	22.1979	0.2265	0.4711	-1.038366	0.000000	0.505778
2	0.375	Linear	39.5935	0.4040	0.6292	-0.994353	0.000000	0.435533
3	0.125	Quad	15.1892	0.1566	0.3897	0.412405	-0.062746	-1.044036
4	0.250	Quad	22.1001	0.2278	0.4701	0.321515	0.085363	-0.989854
5	0.375	Quad	34.4266	0.3549	0.5867	0.202061	-0.214669	-0.665325

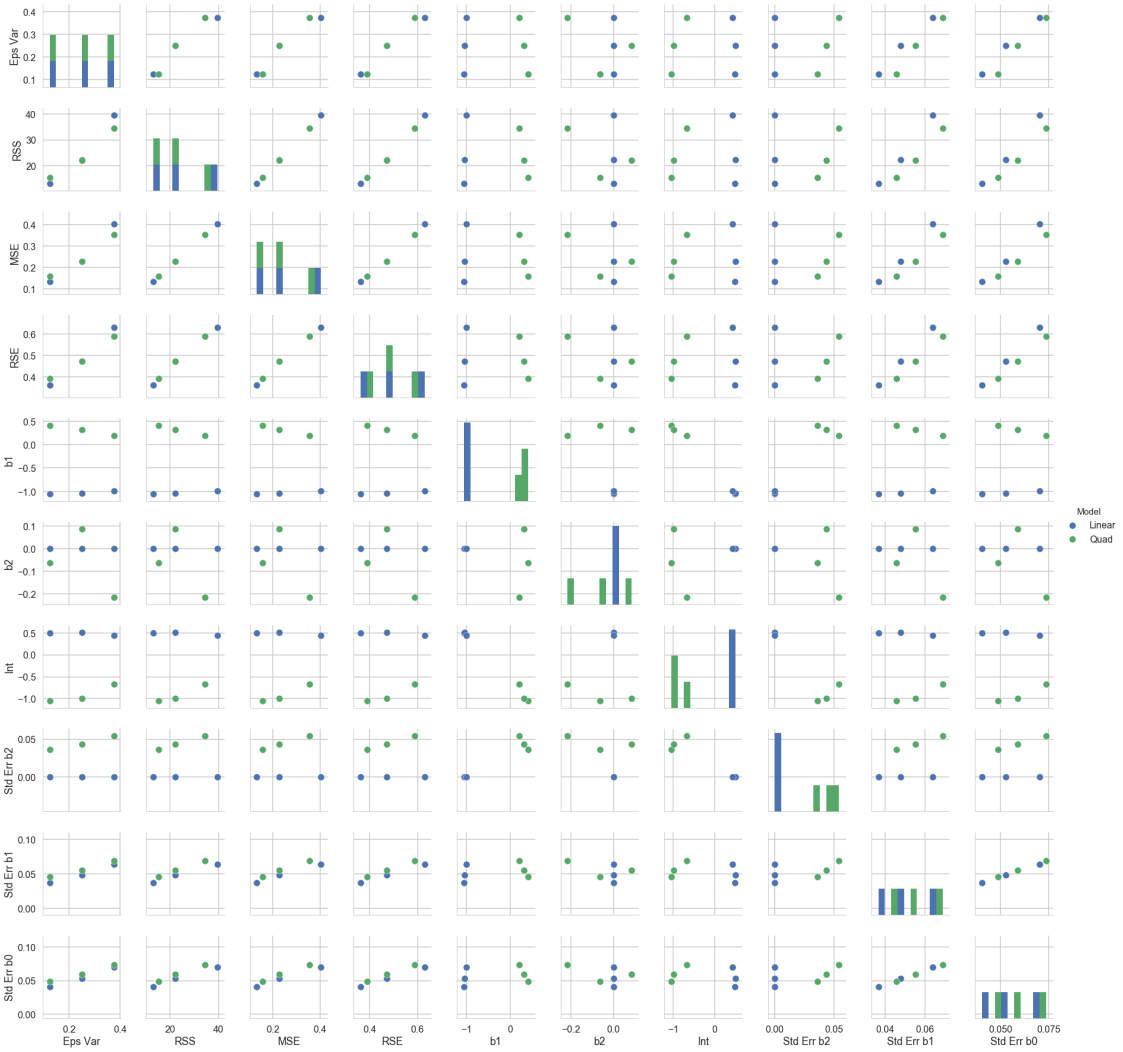
	ci [b2,b1,int]	Std Err b2	Std Err
0	[[-1.1253 -0.9796]\n [0.4136 0.5738]]	0.0000	0.03
1	[[-1.1334 -0.9433]\n [0.4013 0.6103]]	0.0000	0.04
2	[[-1.1213 -0.8674]\n [0.2959 0.5751]]	0.0000	0.06
3	[[-0.1342 0.0087]\n [0.3216 0.5032]\n [-1.1...	0.0360	0.04
4	[[-8.00000000e-04 1.71500000e-01]\n [2.12...	0.0434	0.05
5	[[-0.3222 -0.1071]\n [0.0654 0.3387]\n [-0.8...	0.0542	0.06

	Std Err b0
0	0.0404
1	0.0527

```
2       0.0703
3       0.0489
4       0.0590
5       0.0737
```

```
In [19]: sns.pairplot(df, hue='Model')
```

```
Out[19]: <seaborn.axisgrid.PairGrid at 0x11ed909b0>
```



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.

```
In [20]: df[['Eps Var', 'Model', 'ci [b2,b1,int]', 'Std Err b2', 'Std Err b1', 'Std
```

```
Out[20]:    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  [[-0.1342  0.0087]\n [ 0.3216  0.5032]\n [-1.1...
4    0.250    Quad  [[ -8.00000000e-04   1.71500000e-01]\n [  2.12...
5    0.375    Quad  [[-0.3222 -0.1071]\n [ 0.0654  0.3387]\n [-0.8...

   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      0.0360      0.0457      0.0489
4      0.0434      0.0552      0.0590
5      0.0542      0.0688      0.0737
```

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 [21]: df = pd.read_csv('Auto.csv')

In [22]: df.head()
```

Out[22]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

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

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

```
In [ ]: from statsmodels.formula.api import ols

        res = ols(formula='mpg ~ cylinders + displacement + horsepower + weight + a
                    data=df).fit()
        print(res.summary())
```

```
In [ ]: RSS = np.round(res.ssr, 4)
        RSE = np.round(np.sqrt(1/len(Y-2)*RSS), 4)
        MSE = np.round(res.mse_resid, 4)
        print('RSS = ' + str(RSS))
        print('RSE = ' + str(RSE))
        print("MSE = " +  str(MSE))
```

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

```
In [ ]: sm.qqplot(res.resid, line='r', fit=True)
```

```
In [ ]: sns.boxplot(res.resid)
        plt.title('Boxplot of Residuals')
```

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.

1.7.1 Alter Variables by sqrt

```
In [ ]: sqr = ols(formula='mpg ~ cylinders + displacement + horsepower + weight + a
                  data=df).fit()
        fig, [ax1, ax2] = plt.subplots(1, 2)
        sns.boxplot(sqr.resid, ax=ax1)
        sm.qqplot(sqr.resid, line='r', fit=True, ax=ax2)

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

Sqrt does not appear to deal with the outliers well. I was hoping that it would minimize the extreme values, but it only enlarged them.

1.7.2 Alter Variables by log

```
In [ ]: dflog = df.drop(['name', 'mpg'], axis=1).apply(np.log)
        log = ols(formula='mpg ~ cylinders + displacement + horsepower + weight + a
                  data=pd.concat([dflog, df.mpg], axis=1)).fit()
        fig, [ax1, ax2] = plt.subplots(1, 2)
        sns.boxplot(log.resid, ax=ax1)
        sm.qqplot(log.resid, line='s', fit=True, ax=ax2)

In [ ]:
```