1 Part One: Theoretical Problems

2.4.7 The table below provides a training data set containing six observations, three predictors, and one qualitative response variable.

<table>
<thead>
<tr>
<th>Obs</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>Red</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>Red</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>Red</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>Green</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>Green</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Red</td>
</tr>
</tbody>
</table>

Suppose we wish to use this data set to make a prediction for $Y$ when $X_1 = X_2 = X_3 = 0$ using $K$-nearest neighbors.

(a) Compute the Euclidean distance between each observation and the test point, $X_1 = X_2 = X_3 = 0$.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$\sqrt{10} \approx 3.16$</td>
</tr>
<tr>
<td>4</td>
<td>$\sqrt{2} \approx 2.23$</td>
</tr>
<tr>
<td>5</td>
<td>$\sqrt{2} \approx 1.14$</td>
</tr>
<tr>
<td>6</td>
<td>$\sqrt{3} \approx 1.73$</td>
</tr>
</tbody>
</table>

(b) What is our prediction with $K = 1$? Why?
With $K = 1$, we look only at the nearest point to determine what color the point $(0, 0, 0)$ should be. As can be seen in the chart above, in this case the nearest point is point 5, with a Euclidean distance of 1.14. Thus, our point gets classified the same as point 5, so it is Green.

(c) What is our prediction with $K = 3$? Why?
With $K = 3$, we look at the three nearest points and classify our point as the most common classification among those three. The three nearest points to $(0, 0, 0)$ are points 5, 6, and 2, which have classifications Green, Red, and Red. Thus, since $\frac{3}{3}$ of the results are Red, our point gets classified as Red.

(d) If the Bayes decision boundary in this problem is highly nonlinear, then would we expect the best value for $K$ to be large or small? Why?
The best value for $K$ would be small. The decision boundary is highly nonlinear, which means it twists and turns a lot around the points, giving lots of opportunity for points that are fairly close together to be in different categories. Thus, as $K$ increases and the points being considered get farther from the test point, it becomes much more likely for us to include points from the other section than it would be if the Bayes decision boundary were much more linear.

3.7.3 Suppose we have a data set with five predictors, $X_1 =$ GPA, $X_2 =$ IQ, $X_3 =$ Gender (1 for Female and 0 for Male), $X_4 =$ Interaction between GPA and
IQ, and $X_5$ = Interaction between GPA and Gender. The response is starting salary after graduation (in thousands of dollars). Suppose we use least squares to fit the model, and get $\beta_0 = 50, \beta_1 = 20, \beta_2 = 0.07, \beta_3 = 35, \beta_4 = 0.01, \beta_5 = -10$.

(a) Option iii. For a fixed value of IQ and GPA, males earn more on average than females provided that the GPA is high enough. The coefficient of $X_3$, the gender term, tells us that females earn an average of $35,000 more than males when GPA is ignored. However, for every point increase in GPA, this gap shrinks by $10,000. Thus, for a set GPA of 3.5, males and females will earn the same amount, and for any GPA greater than 3.5, males earn more money.

(b) A female with an IQ of 110 and a GPA of 4.0 will have a starting salary of $Y = 50 + 20(4.0) + 0.07(110) + 35(1) + 0.01(4.0)(110) - 10(4.0)(1) = 137.1$. Thus she will have a starting salary of $137,100.

(c) True or false: Since the coefficient for the GPA/IQ interaction term is very small, there is very little evidence of an interaction effect. Justify your answer. False. Because IQ is measured on the scale of hundreds as opposed to on a one-point scale (gender) or a four-point scale (GPA), the coefficient must be much smaller than the others to produce an effect that is on the same scale as the other results. Thus the small coefficient does not indicate that there is no interaction, merely that the terms being multiplied are much larger in the first place.

3.7.4 I collect a set of data ($n = 100$ observations) containing a single predictor and a quantitative response. I then fit a linear regression model to the data; as well as a separate cubic regression, i.e. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$.

(a) Suppose that the true relationship between $X$ and $Y$ is linear, i.e. $Y = \beta_0 + \beta_1 X + \epsilon$. Consider the training residual sum of squares (RSS) for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer. We would expect the training RSS to be lower for the cubic regression. Even though the actual relationship is linear, the cubic term allows the model to more closely approximate the training data itself, picking up on small fluctuations that may not be present in the data set as a whole (overfitting). If the linear model was more accurate than the closest cubic model, then the closest cubic model would simply become the linear model, with $\beta_2 = \beta_3 = 0$, so the linear model will never have a lower training RSS than the cubic.

(b) Answer (a) using test rather than training RSS. For the test data, we would expect the linear model to produce a lower RSS than the cubic. As mentioned before, the cubic model will be overfitted to the training data, which means that it will predict fluctuations that won’t be present in the test data. The linear model, on the other hand, because it is constrained to only linear terms, has less overfitting and thus will better approximate the entire data set, producing less error once we move away from the training data.
(c) Suppose that the true relationship between $X$ and $Y$ is not linear, but we don’t know how far it is from linear. Consider the training RSS for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer. For similar reasoning as part (a), we would expect the cubic model’s RSS to be lower for the training data. Regardless of how close to linear the data is, the cubic model will always be able to pick up on more of the variations in the training data set to minimize the RSS farther than the linear model can.

(d) Answer (c) using test rather than training RSS. For the test data, there is not enough information to tell whether the linear or cubic RSS will be lower. If the model is very close to linear, there is a good chance that the linear RSS will be lower for the same reasoning as in part (b); however, if the data is markedly not linear, the cubic model’s curves would produce a more accurate model.

3.7.5 Consider the fitted values that result from performing linear regression without an intercept. In this setting, the $i$th fitted value takes the form $\hat{y}_i = x_i \beta$, where

$$\hat{\beta} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}.$$

Show that we can write

$$\hat{y}_i = \sum_{i'=1}^{n} a_{i'} y_{i'}. $$

What is $a_{i'}$?

Substituting the given expression for $\hat{\beta}$ into the equation for $\hat{y}_i$ gives

$$\hat{y}_i = x_i \sum_{i=1}^{n} \frac{x_i y_i}{x_i^2}.$$ 

Note that $x_i$ is a constant, so it can be brought inside the sum. Additionally, $\sum_{i=1}^{n} x_i^2$, the sum of all $x_i^2$, is a constant. Thus,

$$\hat{y}_i = \sum_{i'=1}^{n} y_{i'} \frac{x_i x_{i'}}{\sum_{i''=1}^{n} x_i^2}.$$ 

Therefore $a_{i'} = \frac{x_i x_{i'}}{\sum_{i''=1}^{n} x_i^2}.$

3.7.6 Using (3.4), argue that in the case of simple linear regression, the least squares line always passes through the point $(\bar{x}, \bar{y})$. According to equation (3.4), the predicted value of $y$ will be $\hat{y}_i = \beta_0 + \beta_1 x_i$, where

$$\beta_0 = \bar{y} - \bar{\beta}_1 \bar{x}. $$
Figure 1: The linear regression of the relationship between gas mileage and horsepower.

The formula for $\hat{\beta}_1$ turns out not to be relevant in this case. If we substitute the expression for $\hat{\beta}_0$ into the equation for $\hat{y}_i$, we get

$$\hat{y}_i = \bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 x_i.$$ 

If we look at the predicted $y$ value for $\bar{x}$ using this equation, we get

$$y = \bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 \bar{x} = \bar{y}.$$ 

Thus the point ($\bar{x}, \bar{y}$) is on the least squares line.

**Linear Algebra Exercise**  See attached handwritten notes.

# Part Two: Programming

**Problem 1** Full code for this exercise can be found in the appendix labelled HW01.1.

Simple linear regression was performed on the Auto dataset, found at http://faculty.marshall.usc.edu/gareth-james/ISL/data.html. Least squares was used to estimate gas mileage as a function of horsepower for automobiles. Figure 1 shows the scatter plot of this data, along with the linear regression. It is clear that this model does not fit the data well, as the scatter plot appears to follow a curve rather than a linear relationship. This observation is confirmed by the RSS value, which is approximately 9385.9.

**Problem 2** Full code for this exercise can be found in the appendix labelled HW01.2.

Multiple linear regression was performed on the Auto data set using all numerical data available (name data was excluded).
Figure 2: The colorplot of the relationships between variables in the Auto dataset.

Figure 3: The correlation matrix of the variables in the Auto dataset.
Figure 2 shows the colorplot matrix of the relationships between the variables. This data is also given in numerical form in Figure 3, the correlation matrix for the variables in the Auto dataset. Least squares was used to perform multiple linear regression on this dataset to predict gas mileage. The resulting vector of coefficients was \([-0.49337632, 0.01989564, -0.01695114, 0.00647404, 0.68057584, 0.75077268, 1.4261405]\), and the RSS was 4252.2. This result is approximately half of the residual sum of squares for the simple linear regression model, meaning the multiple linear regression model performs much better.

**Problem 3** A feature vector \(\mathbf{X} \) was created with 100 samples with mean 0 and standard deviation of 1. A noise vector \(\mathbf{eps} \) was generated with mean 0 and standard deviation 0.5. These vectors were used to generate the response vector \(\mathbf{y} = \mathbf{\beta}_0 + 0.5\mathbf{X} + \mathbf{eps} \). The length of the vector \(\mathbf{y} \) is 100. In this model, \(\mathbf{\beta}_0 = 0\) and \(\beta_1 = 0.5\). The scatterplot showing the relationship between \(\mathbf{x} \) and \(\mathbf{y} \) is shown in Figure 4. Notice that the data is spread out around a roughly linear relationship between \(\mathbf{y} \) and \(\mathbf{x} \), with more data near the center of the plot, which is centered around the point \((0, -1)\). \(\mathbf{y} \) increases approximately half as fast as \(\mathbf{x} \).

A least squares linear model was fitted to predict \(\mathbf{y} \) using \(\mathbf{x} \). The obtained coefficients were \(\hat{\mathbf{\beta}} = [-1.04445963, 0.50514424] \). Notice that these are both within approximately ten percent of the actual values \(\mathbf{\beta}_0 = 0\) and \(\beta_1 = 0.5\). The plot of the linear regression, as well as the line originally used to populate the dataset, is shown in Figure 5. For this model, the rmse was 0.24 and \(r^2 \) was 0.81.

A polynomial model was used to model the data based on \(\mathbf{x} \) and \(\mathbf{x}^2 \). This model is shown in Figure 6. For the quadratic model, rmse was 0.056 and \(r^2 \) was 0.99, which is a much better performance than the linear model. This is likely due to overfitting of this specific data, since we know the original model was linear.
The above procedure was repeated with a data set that contained less noise. The resulting model is shown in Figure 7. It is clear that the model is much more accurate when there is less noise, as rmse was 0.10 and $r^2$ was 0.97, much better than the original model. However, the estimates for the coefficients were closer, but not by much: $\hat{\beta}_0$ was $-0.986$, and $\hat{\beta}_1$ was 0.503.

The procedure was repeated once again, this time with data that contained more noise. This plot is shown in Figure 8. The rmse and $r^2$ terms were much worse for this model, with values of 0.41 and 0.59, respectively. However, the coefficient estimates, while not as close as the low-noise model, were actually closer than the original model, with $\beta_0 = -1.04$ and $\beta_1 = 0.478$. 
Figure 7: A linear model of data with less noise.

Figure 8: A linear model of data with more noise.
Linear Algebra Exercise: Use the least square formula (pseudo-inverse $\beta = (A^T A)^{-1} A^T y$) to derive Equation (3.4) in the textbook.

Equation (3.4) describes the least-squares coefficients of a simple linear regression for the equation $y = A \beta$, with $A = \begin{bmatrix} 1 & x_i \\ 1 & x_n \end{bmatrix}$. Thus,

$$A^T A = \begin{bmatrix} 1 & \cdots & 1 \\ x_i & \cdots & x_n \end{bmatrix} \begin{bmatrix} n & \Sigma x_i \\ \Sigma x_i & \Sigma x_i x^2 \end{bmatrix} = \begin{bmatrix} n & \Sigma x_i \\ \Sigma x_i & \Sigma x_i x^2 \end{bmatrix} = \begin{bmatrix} n & \Sigma x_i \\ \Sigma x_i & \Sigma x_i x^2 \end{bmatrix}.$$ 

Using the formula for the inverse of a $2 \times 2$ matrix gives

$$(A^T A)^{-1} = \frac{1}{n (\Sigma x_i x^2 - n \Sigma x_i \bar{x})} \begin{bmatrix} \Sigma x_i^2 & -n \Sigma x_i \\ -n \Sigma x_i & n \end{bmatrix}.$$ 

Thus

$$(A^T A)^{-1} A^T = \frac{1}{n (\Sigma x_i x^2 - n \Sigma x_i \bar{x})} \begin{bmatrix} \Sigma x_i^2 & -n \Sigma x_i \\ -n \Sigma x_i & n \end{bmatrix} \begin{bmatrix} \Sigma x_i x^2 & \Sigma x_i x \bar{x} \\ \Sigma x_i x \bar{x} & \Sigma x_i x^2 \end{bmatrix} n \begin{bmatrix} \Sigma x_i x^2 - n \Sigma x_i \bar{x} \\ n (x_i - \bar{x}) \\ n (x_n - \bar{x}) \end{bmatrix}.$$ 

Therefore $\beta = (A^T A)^{-1} A^T y$ is the product of the above matrix and the vector $[y_i \ldots y_n]$, so $\beta = \frac{1}{n (\Sigma x_i x^2 - n \Sigma x_i \bar{x})} \begin{bmatrix} \Sigma y_i (\Sigma x_i x^2 - n x_i \bar{x}) \\ \Sigma y_i (n (x_i - \bar{x})) \end{bmatrix}$

Thus $\beta = \frac{\Sigma y_i (\Sigma x_i (\Sigma x_i x^2 - n x_i \bar{x}))}{n (\Sigma x_i x^2 - n \Sigma x_i \bar{x})}$. 

And $\beta_1 = \frac{\Sigma y_i (x_i - \bar{x}) n}{n (\Sigma x_i x^2 - n \Sigma x_i \bar{x})} = \frac{\Sigma y_i x_i - \Sigma y_i \bar{x}}{\Sigma x_i x^2 - n \Sigma x_i \bar{x}} = \frac{\Sigma y_i x_i - n \Sigma x_i \bar{x}}{\Sigma x_i x^2 - n \Sigma x_i \bar{x}} = \frac{\Sigma y_i x_i - n \Sigma x_i \bar{x}}{\Sigma x_i x^2 - n \Sigma x_i \bar{x}} \frac{\Sigma x_i x^2 - n \Sigma x_i \bar{x}}{\Sigma x_i x^2 - n \Sigma x_i \bar{x}} = \frac{\Sigma (x_i x - x \bar{y} - y \bar{x})}{\Sigma (x_i ^2 - 2 x_i \bar{x} + \bar{x}^2)}$.
\[ \beta_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}, \text{ as desired.} \]

Additionally,
\[ \beta_0 = \frac{\left(\frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i} \left(\bar{x}^2 - n\bar{x}\bar{y}\right)\right)}{n \sum_{i=1}^{n} (x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n} y_i \left(\bar{x}^2 - x_i \bar{x}\right)}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \]
\[ = \frac{\sum_{i=1}^{n} y_i \bar{x}^2 - \sum_{i=1}^{n} y_i x_i \bar{x}}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \]
\[ = \bar{y} \frac{\sum_{i=1}^{n} x_i^2 - \bar{x} \sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} (x_i - \bar{x})^2} = \bar{y} \frac{\sum_{i=1}^{n} x_i^2 - n \bar{x} \bar{y} \bar{x}^2 - \bar{x} \sum_{i=1}^{n} y_i x_i + n \bar{y} \bar{x}^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \]
\[ = \bar{y} \left(\frac{\sum_{i=1}^{n} x_i^2 - n \bar{x}^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}\right) - \bar{x} \left(\frac{\sum_{i=1}^{n} y_i x_i - n \bar{y} \bar{x}}{\sum_{i=1}^{n} (x_i - \bar{x})^2}\right) \]
\[ = \bar{y} \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2 - \bar{x} \sum_{i=1}^{n} (x_i - \bar{x}) (y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \]
\[ = \bar{y} - \bar{\beta_1} \bar{x}, \text{ as desired.} \]
In [1]:
   : import numpy as np
   :
       : import matplotlib.pyplot as plt
       : import pandas as pd
       : from sklearn.linear_model import LinearRegression

In [2]:
   : data = pd.read_csv('Auto.csv')
   : data = data[data.horsepower != '?']
   : X = data.iloc[:, 3].values.reshape(-1, 1)
   : for i in X:
   :     i[0] = float(i[0])
   : Y = data.iloc[:, 0].values.reshape(-1, 1)
   : linear_regressor = LinearRegression()
   : linear_regressor.fit(X, Y)
   : Y_pred = linear_regressor.predict(X)

In [3]:
   : fig, ax = plt.subplots()
   : plt.scatter(X, Y)
   : plt.plot(X, Y_pred, color='red')
   : ax.set_title("Effect of Horsepower on Gas Milage for Automobiles")
   : ax.set_xlabel("Horsepower")
   : ax.set_ylabel("MPG")
   : plt.show()
   : fig.savefig('mpg_vs_horsepower.jpeg')
In [4]: ssr = np.sum((Y_pred - Y)**2)
   print("Residual Sum of Squares: ", ssr)

Residual Sum of Squares: 9385.915871932419

In [ ]:
In [1]: # library & dataset
    import seaborn as sns
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np
    from sklearn.linear_model import LinearRegression

In [2]: df = pd.read_csv('Auto.csv')
    print(df[0:5])

    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 [3]: df = df[df.horsepower != '?']
    df = df[['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year']
    df = df.astype('float64')

In [4]: plt.matshow(df.corr())
    plt.xticks(ticks=range(8), labels=df.columns, rotation=40, horizontalalignment='left')
    plt.yticks(ticks=range(8), labels=df.columns)
    plt.colorbar()
    plt.title("Colorplot of Vehicle Trait Correlations", pad=60)
    plt.show()

display(df.corr())
Colorplot of Vehicle Trait Correlations

```
mpg    cylinders  displacement    horsepower  weight
mpg     1.000000  -0.777618    -0.805127    -0.778427    -0.832244
cylinders -0.777618   1.000000    0.950823    0.842983    0.897527
displacement -0.805127    0.950823    1.000000    0.897257    0.932994
horsepower -0.778427    0.842983    0.897257    1.000000    0.864538
weight     -0.832244    0.897527    0.932994    0.864538    1.000000
acceleration 0.423329   -0.504683    -0.543800    -0.689196   -0.416839
year        0.580541   -0.345647    -0.369855    -0.416361   -0.309120
origin      0.565209   -0.568932    -0.614535    -0.455171   -0.585005
```

```
acceleration   year    origin
mpg             0.423329  0.580541  0.565209
cylinders      -0.504683  -0.345647  -0.568932
displacement   -0.543800  -0.369855  -0.614535
horsepower     -0.689196  -0.416361  -0.455171
weight         -0.416839  -0.309120  -0.585005
acceleration   1.000000  0.290316  0.212746
year           0.290316  1.000000  0.181528
origin         0.212746  0.181528  1.000000
```

In [6]: mlr = LinearRegression()
mpg = df.mpg.values.reshape(-1,1)
```python
x_matrix = df.iloc[:, 1:].values
print(df[0:5])
print(x_matrix[0:5])

<table>
<thead>
<tr>
<th>mpg</th>
<th>cylinders</th>
<th>displacement</th>
<th>horsepower</th>
<th>weight</th>
<th>acceleration</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>8.0</td>
<td>307.0</td>
<td>130.0</td>
<td>3504.0</td>
<td>12.0</td>
<td>70.0</td>
</tr>
<tr>
<td>15.0</td>
<td>8.0</td>
<td>350.0</td>
<td>165.0</td>
<td>3693.0</td>
<td>11.5</td>
<td>70.0</td>
</tr>
<tr>
<td>18.0</td>
<td>8.0</td>
<td>318.0</td>
<td>150.0</td>
<td>3436.0</td>
<td>11.0</td>
<td>70.0</td>
</tr>
<tr>
<td>16.0</td>
<td>8.0</td>
<td>304.0</td>
<td>150.0</td>
<td>3433.0</td>
<td>12.0</td>
<td>70.0</td>
</tr>
<tr>
<td>17.0</td>
<td>8.0</td>
<td>302.0</td>
<td>140.0</td>
<td>3494.0</td>
<td>10.5</td>
<td>70.0</td>
</tr>
</tbody>
</table>
```

```python
origin
0 1.0
1 1.0
2 1.0
3 1.0
4 1.0
```

```
[[8.000e+00  3.070e+02  1.300e+02  3.504e+03  1.200e+01  7.000e+01  1.000e+00]
 [8.000e+00  3.500e+02  1.650e+02  3.693e+03  1.150e+01  7.000e+01  1.000e+00]
 [8.000e+00  3.180e+02  1.500e+02  3.436e+03  1.100e+01  7.000e+01  1.000e+00]
 [8.000e+00  3.040e+02  1.500e+02  3.433e+03  1.200e+01  7.000e+01  1.000e+00]
 [8.000e+00  3.020e+02  1.400e+02  3.449e+03  1.050e+01  7.000e+01  1.000e+00]]
```

```
In [6]: mlr.fit(x_matrix, mpg)
    mpg_pred = mlr.predict(x_matrix)
    coef = mlr.coef_
    intercept = mlr.intercept_
    print("Coefficients: ", coef)
    print("Residual Sum of Squares: ", ssr)
```

```
Coefficients: [-0.49337632  0.01989564 -0.01695114 -0.00647404  0.08057584  0.75077268
  1.4261405 ]
Residual Sum of Squares:  4252.212530440178
```

```
In [ ]:
```
HW01_3

September 4, 2019

In this exercise you will create some simulated data and will fit simple linear regression models to it. Make sure to use set.seed(1) prior to starting part (a) to ensure consistent results.

In [2]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

(a) Using the rnorm() function, create a vector, x, containing 100 observations drawn from a N(0, 1) distribution. This represents a feature, X.

In [3]: #mean 0, variance 1
x = np.random.normal(size=(100,1))
print(x.shape)
print(x.size)

(100, 1)
100

(b) Using the rnorm() function, create a vector, eps, containing 100 observations drawn from a N(0, 0.25) distribution i.e. a normal distribution with mean zero and variance 0.25.

In [4]: #mean 0, variance 0.25
eps = np.random.normal(scale = 0.25, size=(100,1))
print(eps.shape)

(100, 1)

(c) Using x and eps, generate a vector y according to the model $Y = 1 + 0.5X + \epsilon$. What is the length of the vector y? What are the values of 0 and 1 in this linear model?

In [5]: y = np.full(shape=(100,1), fill_value=(-1)) + 0.5 * x + eps
print(y.shape)
print(y.size)
(100, 1)
100

(d) Create a scatterplot displaying the relationship between x and y. Comment on what you observe.

In [6]: fig, ax = plt.subplots()
   ...: plt.scatter(x, y)
   ...: ax.set_title("Scatter Plot of y vs. x")
   ...: ax.set_xlabel("x")
   ...: ax.set_ylabel("y")
   ...: plt.show()
   ...: fig.savefig('Q3_dscatter.jpg')

![Scatter Plot of y vs. x](Q3_dscatter.jpg)

(e) Fit a least squares linear model to predict y using x. Comment on the model obtained. How do 0 and 1 compare to 0 and 1?

In [7]: regr = LinearRegression()
   ...: regr.fit(x, y)
   ...: y_pred = regr.predict(x)
   ...: b_0 = regr.intercept_
   ...: b_1 = regr.coef_
   ...: print(b_0, b_1)
(f) Display the least squares line on the scatterplot obtained in (d). Draw the population regression line on the plot, in a different color. Use the legend() command to create an appropriate legend.

In [8]: fig, ax = plt.subplots()
   plt.scatter(x, y)
   ax.set_title("Regressions of y vs. x")
   ax.set_xlabel("x")
   ax.set_ylabel("y")
   plt.plot(x, y_pred, color='red', label='Least Squares')
   plt.plot(x, y - eps, color='green', label='Population')
   ax.legend()
   plt.show()
   fig.savefig('03_fRegressions.jpg')
   plt.close('all')

Regressions of y vs. x

In [9]: rmse_lin = np.sqrt(mean_squared_error(y, y_pred))
   r2_lin = r2_score(y, y_pred)
   print("rmse linear = ", rmse_lin)
   print("r2 linear = ", r2_lin)
(g) Now fit a polynomial regression model that predicts y using x and x2. Is there evidence that the quadratic term improves the model fit? Explain your answer.

In [10]:

```python
poly = PolynomialFeatures(degree=2)
data_array = np.ndarray(shape=(100, 2))
    # print(data_array[0:5])
for i in range(100):
    data_array[i][0] = x[i]
    data_array[i][1] = y[i]
data_array.sort(axis=0)
x_sorted = np.ndarray(shape=(100, 1))
y_sorted = np.ndarray(shape=(100, 1))
for i in range(100):
    x_sorted[i] = data_array[i][0]
    y_sorted[i] = data_array[i][1]
x_poly = poly.fit_transform(x_sorted)
regr2 = LinearRegression()
regr2.fit(x_poly, y_sorted)
y_pred2 = regr2.predict(x_poly)

coef = regr2.coef_
intercept = regr2.intercept_
print(intercept, coef)

rmse_quad = np.sqrt(mean_squared_error(y_sorted, y_pred2))
r2_quad = r2_score(y_sorted, y_pred2)
print("rmse quadratic = ", rmse_quad)
print("r2 quadratic = ", r2_quad)
```

```
fig, ax = plt.subplots()
plt.scatter(x, y)
ax.set_title("Degree 2 Model of y vs. x")
ax.set_xlabel("x")
ax.set_ylabel("y")
plt.plot(x_sorted, y_pred2, color='red', label='Polynomial Regression')
plt.plot(x, y - eps, color='green', label='Population')
ax.legend()
plt.show()
fig.savefig('Q3_gDeg2.jpg')
plt.close('all')
```

[-1.0825575] [[0. 0.55381705 0.03144781]]
rmse quadratic = 0.05555832691467338
r2 quadratic = 0.9900625808630914
(h) Repeat (a)–(f) after modifying the data generation process in such a way that there is less noise in the data. The model (3.39) should remain the same. You can do this by decreasing the variance of the normal distribution used to generate the error term in (b). Describe your results.

In [13]:
```python
x = np.random.normal(size=(100, 1))
eps = np.random.normal(scale=0.1, size=(100, 1))
y = np.full(shape=(100, 1), fill_value=(-1)) + 0.5 * x + eps
regr = LinearRegression()
regr.fit(x, y)
y_pred = regr.predict(x)
b_0 = regr.intercept_
b_1 = regr.coef_
fig, ax = plt.subplots()
plt.scatter(x, y)
ax.set_title("Regression of y vs. x")
ax.set_xlabel("x")
ax.set_ylabel("y")
plt.plot(x, y_pred, color='red', label='Least Squares')
plt.plot(x, y - eps, color='green', label='Population')
ax.legend()
plt.show()
fig.savefig('Q3_hLessNoise.jpg')
plt.close('all')```
(i) Repeat (a)–(f) after modifying the data generation process in such a way that there is more noise in the data. The model (3.39) should remain the same. You can do this by increasing the variance of the normal distribution used to generate the error term in (b). Describe your results.

In [14]: x = np.random.normal(size=(100,1))
eps = np.random.normal(scale = 0.4, size=(100,1))
y = np.full(shape=(100,1), fill_value=(-1)) + 0.5 * x + eps
regr = LinearRegression()
regr.fit(x, y)
y_pred = regr.predict(x)
b_0 = regr.intercept_
b_1 = regr.coef_
fig, ax = plt.subplots()
plt.scatter(x, y)
ax.set_title("Regressions of y vs. x")
ax.set_xlabel("x")
ax.set_ylabel("y")
plt.plot(x, y_pred, color='red', label='Least Squares')
plt.plot(x, y - eps, color='green', label='Population')
ax.legend()
plt.show()
fig.savefig('Q3_iMoreNoise.jpg')
plt.close('all')
rmse_lin = np.sqrt(mean_squared_error(y, y_pred))
r2_lin = r2_score(y, y_pred)
b_0 = regr.intercept_
b_1 = regr.coef_
print(b_0, b_1)
print("rmse linear = ", rmse_lin)
print("r2 linear = ", r2_lin)
\[ r^2 \text{ linear} = 0.5856827761556463 \]

\[ \text{In [ ]:} \]