University of Texas at Austin, Cockrell School of Engineering
Data Mining – EE 380L

Problem Set # 2 (2b)

March 09, 2016

Gabrielson Eapen
EID: EAPENGP

Discussed Homework with Following Students:
1. Mudra Gandhi

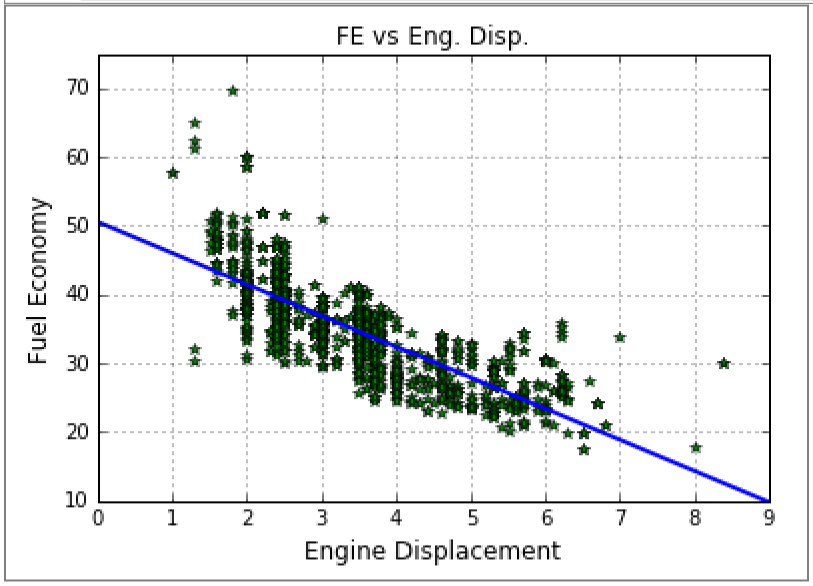
Q1] Part a]

FE (Fuel Efficiency) is a good candidate for dependent variable.
EngDispl, NumCyl, Transmission, NumGears, IntakeValvePerCyl, ExhaustValvesPerCyl are continuous variables.
AirAspirationMethod, TransLockup, TransCreeperGear, DriveDesc, CarlineClassDesc, VarValveTiming, VarValveLift are categorical variables.

Part b]

```
In [5]: #Part b
        # http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.linalg.lstsq.html
        # B1 = m; B0 = c
        # y = B1x + B0
        # y = Ap, where A = [[x 1]] and p = [[B1], [B0]]
        x = df["EngDispl"]
        A = np.vstack([x, np.ones(len(x))]).T
        #print A
        y = np.array(df["FE"])
        #m, c = np.linalg.lstsq(A, y)[0]
        #B1, B0 = np.linalg.lstsq(A, y)[0]
        model, resid = np.linalg.lstsq(A, y)[:2]
        B1, B0 = model
        #np.linalg.lstsq(A, y)
        print("m(B1) is: {}.".format(round(B1,8)))
        print("c(B0) is: {}.".format(round(B0,8)))
        r2 = 1 - resid / (y.size * y.var())
        print("R2 is: {}.".format(round(r2[0],8)))

        m(B1) is: -4.52092928.
        c(B0) is: 50.56322991.
        R2 is: 0.61998904.
```

```
In [6]: #Part b
        # Add bestfit line to ScatterPlot using B1 and B0 values
        # y = B1x + B0

        part_b, = plt.plot(df["EngDispl"],df["FE"],'g*')
        plt.grid(True)
        plt.ylabel("Fuel Economy",fontsize=12)
        plt.xlabel("Engine Displacement",fontsize=12)
        plt.title("FE vs Eng. Disp.")
        plt.xlim(0, 9)
        plt.ylim(10, 75)
        xVals = np.arange(0, 325)
        yVals = (B1*xVals) + B0
        plt.plot(xVals,yVals,'b-',linewidth=2.0)
        plt.show()
```



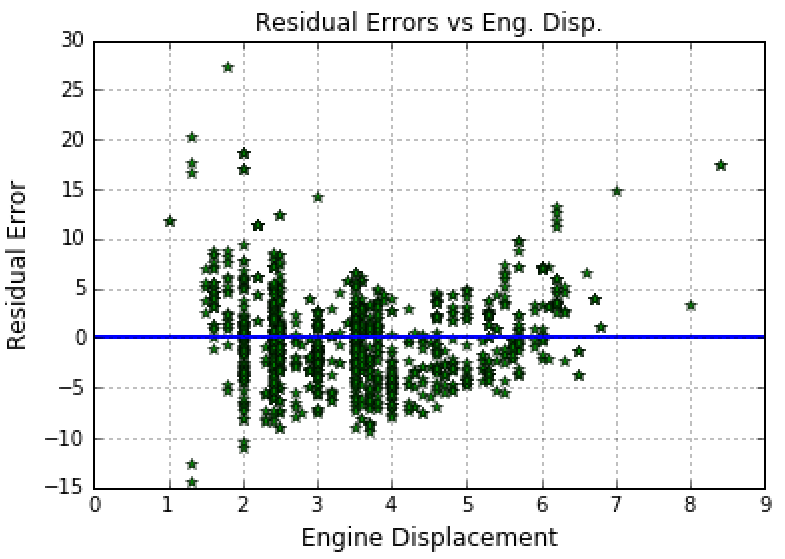
Part c]

```
In [7]: # Part C
        # Plot the residual errors. Now sum the residual errors. What do you find?
        # compute the predicted value for EngDispl
        def predictedVals(coeff,intercept,engDisp):
            y_vals=list()
            for x in np.nditer(engDisp):
                y_vals.append((coeff * x )+ intercept)
            return y_vals
```

```
In [8]: yPredict=predictedVals(B1,B0,df.as_matrix(["EngDispl"]))
```

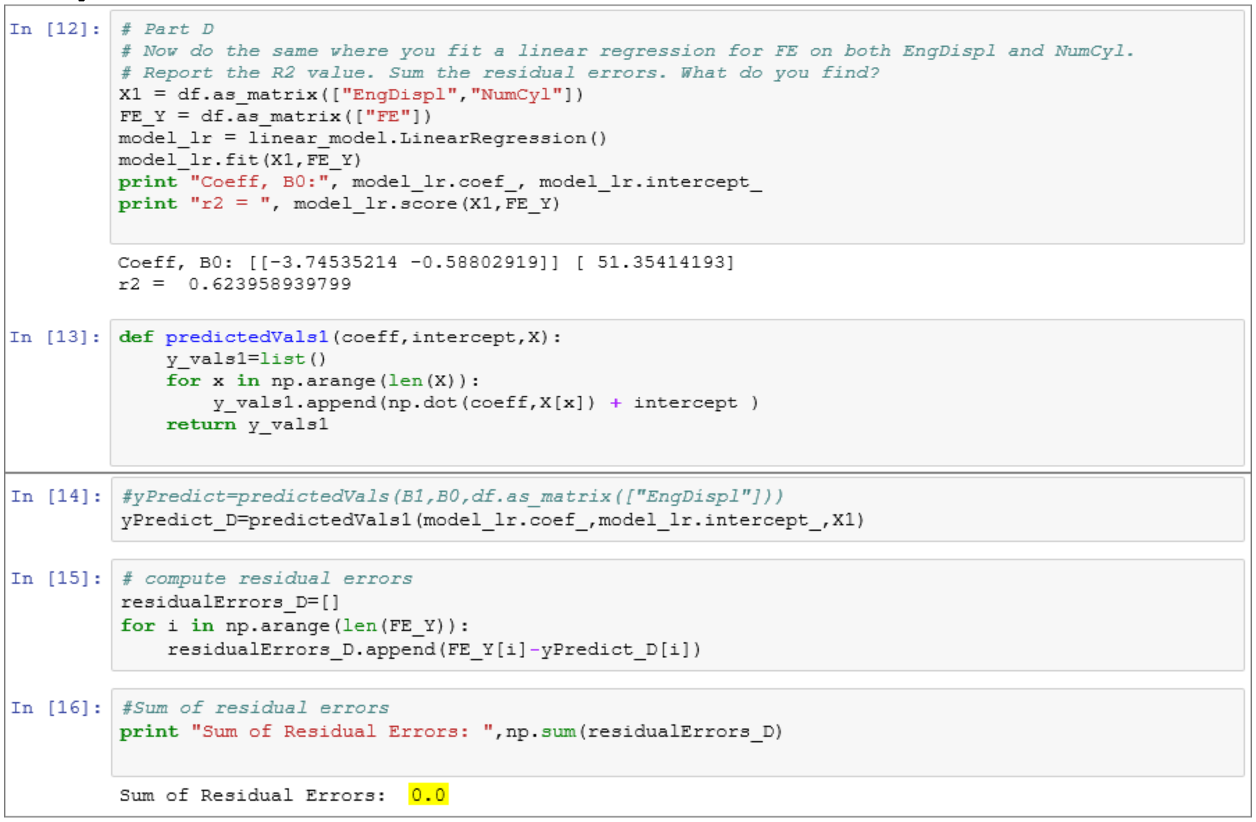
```
In [9]: # compute residual errors
        residualErrors=[]
        Y_FE = df.as_matrix(["FE"])
        for i in np.arange(len(Y_FE)):
            residualErrors.append(Y_FE[i]-yPredict[i])
```

```
In [10]: part_c, = plt.plot(df["EngDispl"],residualErrors,'g*')
         plt.grid(True)
         plt.ylabel("Residual Error",fontsize=12)
         plt.xlabel("Engine Displacement",fontsize=12)
         plt.title("Residual Errors vs Eng. Disp.")
         plt.xlim(0, 9)
         xVals1 = np.arange(0, 325)
         yVals1 = (0*xVals)
         plt.plot(xVals1,yVals1,'b-',linewidth=2.0)
         plt.show()
```



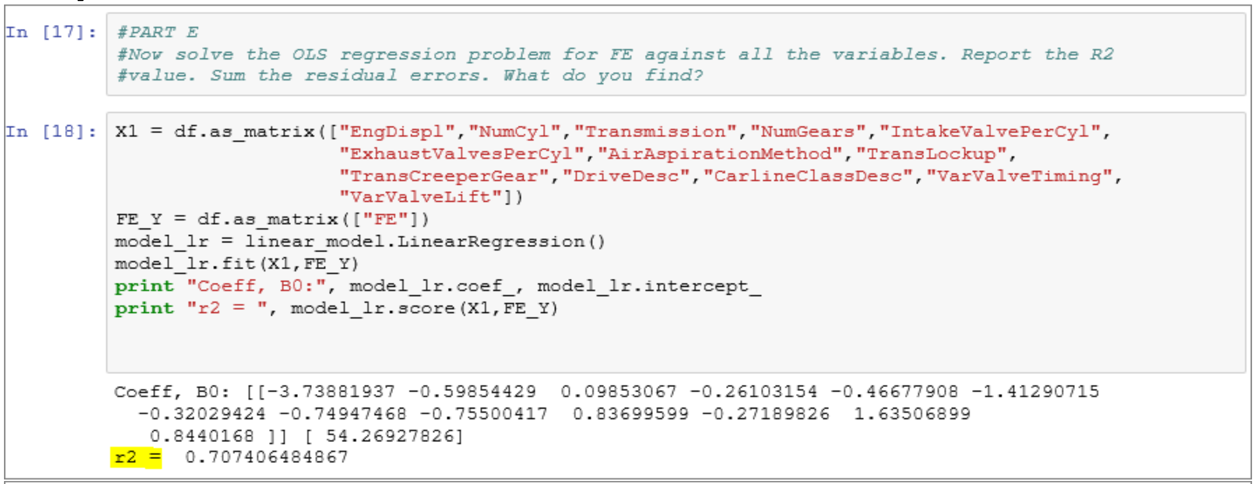
```
In [11]: #Sum of residual errors
         print "Sum of Residual Errors: ",np.sum(residualErrors)

         Sum of Residual Errors:  -2.2055246518e-11
```

Part d]

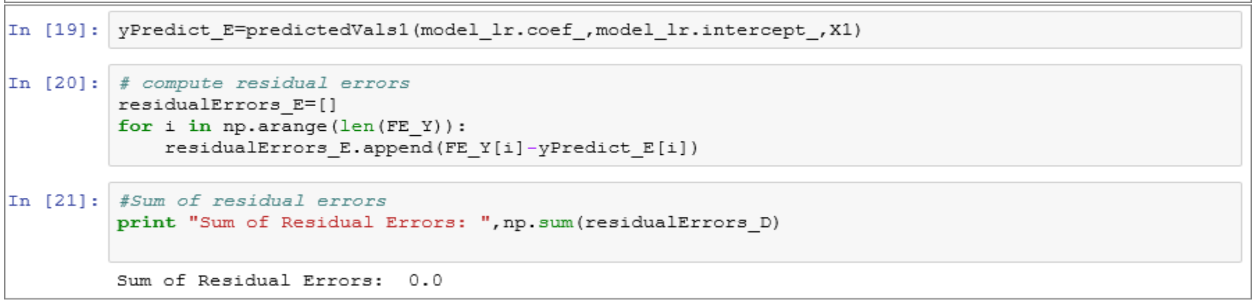


```
In [12]: # Part D
         # Now do the same where you fit a linear regression for FE on both EngDispl and NumCyl.
         # Report the R2 value. Sum the residual errors. What do you find?
         X1 = df.as_matrix(["EngDispl","NumCyl"])
         FE_Y = df.as_matrix(["FE"])
         model_lr = linear_model.LinearRegression()
         model_lr.fit(X1,FE_Y)
         print "Coeff, B0:", model_lr.coef_, model_lr.intercept_
         print "r2 = ", model_lr.score(X1,FE_Y)

         Coeff, B0: [[-3.74535214 -0.58802919]] [ 51.35414193]
         r2 =  0.623958939799

In [13]: def predictedVals1(coeff,intercept,X):
             y_vals1=list()
             for x in np.arange(len(X)):
                 y_vals1.append(np.dot(coeff,X[x]) + intercept )
             return y_vals1

In [14]: #yPredict=predictedVals(B1,B0,df.as_matrix(["EngDispl"]))
         yPredict_D=predictedVals1(model_lr.coef_,model_lr.intercept_,X1)

In [15]: # compute residual errors
         residualErrors_D=[]
         for i in np.arange(len(FE_Y)):
             residualErrors_D.append(FE_Y[i]-yPredict_D[i])

In [16]: #Sum of residual errors
         print "Sum of Residual Errors: ",np.sum(residualErrors_D)

         Sum of Residual Errors:  0.0
```

Part e]



```
In [17]: #PART E
         #Now solve the OLS regression problem for FE against all the variables. Report the R2
         #value. Sum the residual errors. What do you find?

In [18]: X1 = df.as_matrix(["EngDispl","NumCyl","Transmission","NumGears","IntakeValvePerCyl",
                            "ExhaustValvesPerCyl","AirAspirationMethod","TransLockup",
                            "TransCreeperGear","DriveDesc","CarlineClassDesc","VarValveTiming",
                            "VarValveLift"])
         FE_Y = df.as_matrix(["FE"])
         model_lr = linear_model.LinearRegression()
         model_lr.fit(X1,FE_Y)
         print "Coeff, B0:", model_lr.coef_, model_lr.intercept_
         print "r2 = ", model_lr.score(X1,FE_Y)

         Coeff, B0: [[-3.73881937 -0.59854429  0.09853067 -0.26103154 -0.46677908 -1.41290715
           -0.32029424 -0.74947468 -0.75500417  0.83699599 -0.27189826  1.63506899
            0.8440168 ]] [ 54.26927826]
         r2 =  0.707406484867
```

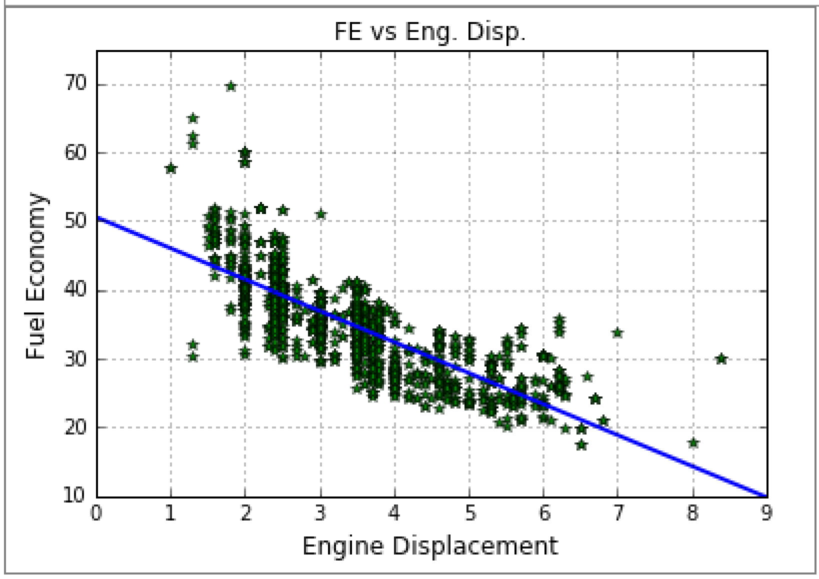


```
In [19]: yPredict_E=predictedVals1(model_lr.coef_,model_lr.intercept_,X1)

In [20]: # compute residual errors
         residualErrors_E=[]
         for i in np.arange(len(FE_Y)):
             residualErrors_E.append(FE_Y[i]-yPredict_E[i])

In [21]: #Sum of residual errors
         print "Sum of Residual Errors: ",np.sum(residualErrors_D)

         Sum of Residual Errors:  0.0
```

Q2. Part A]

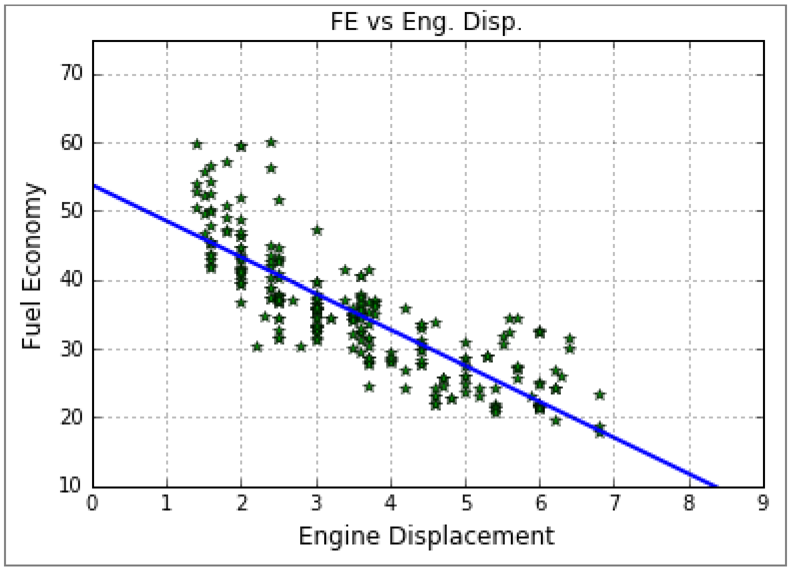
```
In [4]: df=pd.read_stata("cars2010.dta")
```

```
In [5]: X = df["EngDispl"]
        A = np.vstack([X, np.ones(len(X))]).T
        #print A
        y = np.array(df["FE"])
        #m, c = np.linalg.lstsq(A, y)[0]
        #B1, B0 = np.linalg.lstsq(A, y)[0]
        model, resid = np.linalg.lstsq(A, y)[:2]
        B1, B0 = model
        #np.linalg.lstsq(A, y)
        print("m(B1) is: {}.".format(round(B1,8)))
        print("c(B0) is: {}.".format(round(B0,8)))
        r2 = 1 - resid / (y.size * y.var())
        print("R2 is: {}.".format(round(r2[0],8)))

        m(B1) is: -4.52092928.
        c(B0) is: 50.56322991.
        R2 is: 0.61998904.
```



```
In [7]: df2=pd.read_stata("cars2011.dta")
```

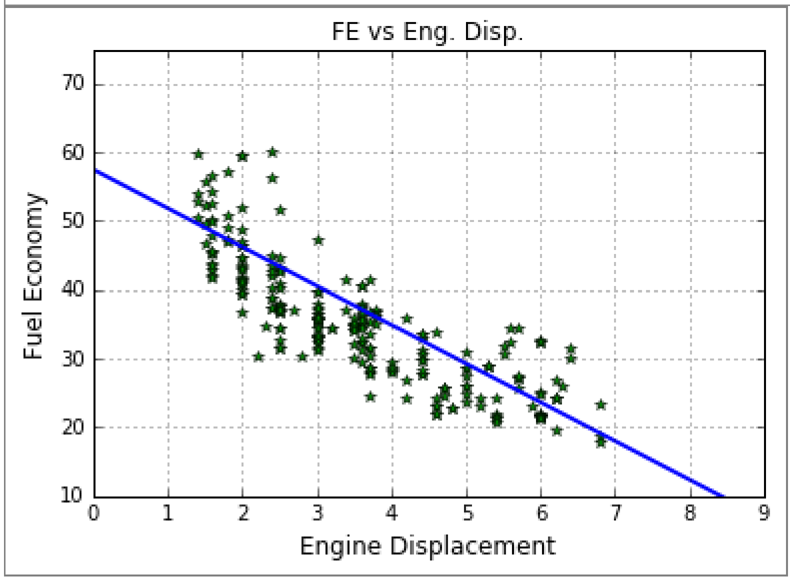
```
In [9]: X = df2["EngDispl"]
        A = np.vstack([X, np.ones(len(X))]).T
        #print A
        y = np.array(df2["FE"])
        #m, c = np.linalg.lstsq(A, y)[0]
        #B1, B0 = np.linalg.lstsq(A, y)[0]
        model, resid = np.linalg.lstsq(A, y)[:2]
        B1, B0 = model
        #np.linalg.lstsq(A, y)
        print("m(B1) is: {}.".format(round(B1,8)))
        print("c(B0) is: {}.".format(round(B0,8)))
        r2 = 1 - resid / (y.size * y.var())
        print("R2 is: {}.".format(round(r2[0],8)))

        m(B1) is: -5.24210076.
        c(B0) is: 53.78837489.
        R2 is: 0.70186418.
```



```
In [11]: df3=pd.read_stata("cars2012.dta")
```

```
In [12]: X = df3["EngDispl"]
         A = np.vstack([X, np.ones(len(X))]).T
         #print A
         y = np.array(df3["FE"])
         #m, c = np.linalg.lstsq(A, y)[0]
         #B1, B0 = np.linalg.lstsq(A, y)[0]
         model, resid = np.linalg.lstsq(A, y)[:2]
         B1, B0 = model
         #np.linalg.lstsq(A, y)
         print("m(B1) is: {}.".format(round(B1,8)))
         print("c(B0) is: {}.".format(round(B0,8)))
         r2 = 1 - resid / (y.size * y.var())
         print("R2 is: {}.".format(round(r2[0],8)))
```

```
m(B1) is: -5.63071065.
c(B0) is: 57.47228974.
R2 is: 0.71225482.
```



Q2 Part B]

Given:

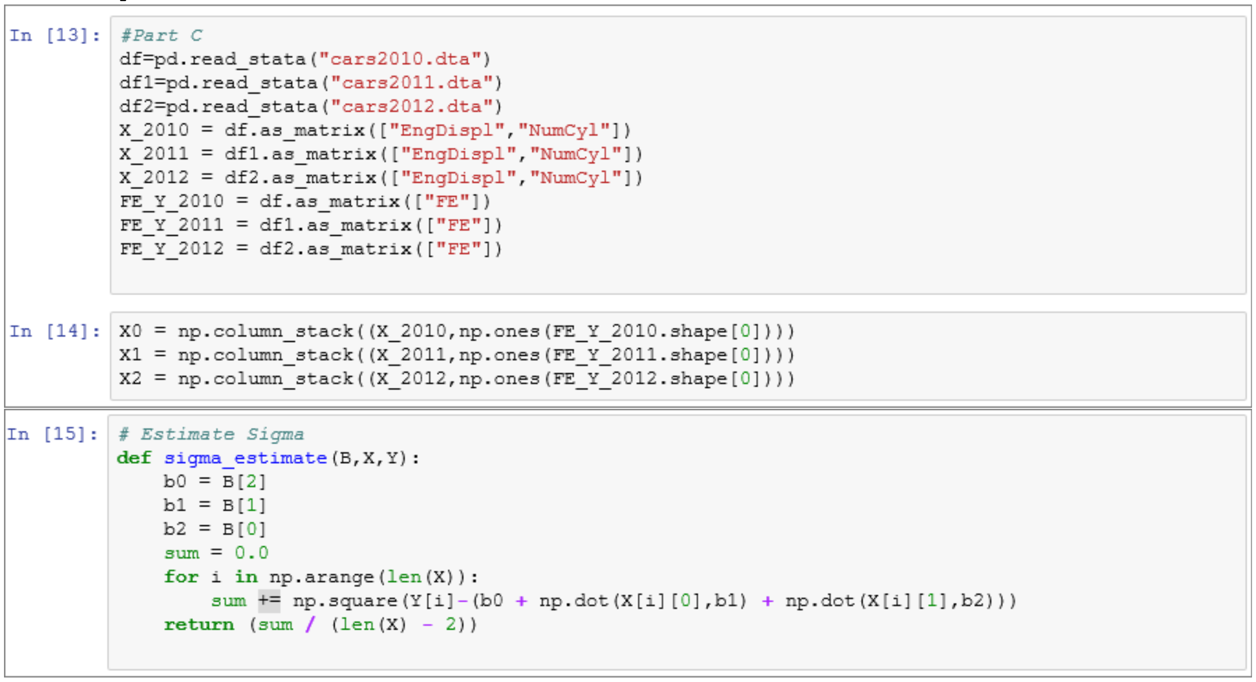
$\mathrm{E}[w_i] = 0$

$\mathrm{E}[w_i w_k] = \mathrm{E}[w_i]\mathrm{E}[w_k] = 0$

$\mathrm{E}\left[w_i^2\right] = \sigma^2$

$$
\begin{aligned}
Cov(w) = {} & \mathrm{E}[ww^T] \\
&= \mathrm{E}\left[\begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \begin{pmatrix} w_1 & \cdots & w_n \end{pmatrix}\right] \\
&= \mathrm{E}\begin{bmatrix} w_1^2 & w_1 w_i & w_i w_n \\ w_j w_1 & \ddots & w_j w_n \\ w_n w_1 & w_n w_i & w_n^2 \end{bmatrix} \\
&= \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix} \\
&= \sigma^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \sigma^2 I
\end{aligned}
$$

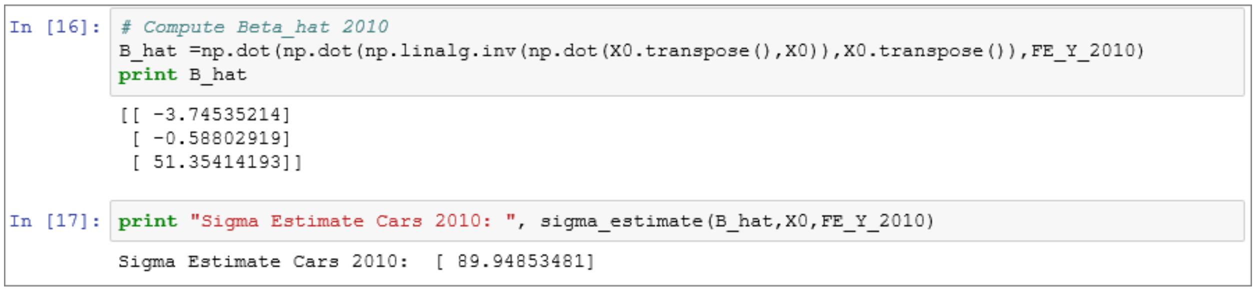
Q2 Part C]



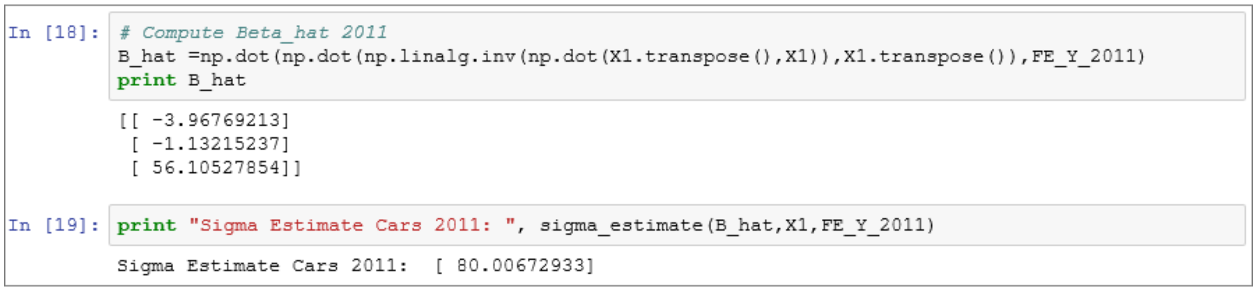
```
In [13]: #Part C
         df=pd.read_stata("cars2010.dta")
         df1=pd.read_stata("cars2011.dta")
         df2=pd.read_stata("cars2012.dta")
         X_2010 = df.as_matrix(["EngDispl","NumCyl"])
         X_2011 = df1.as_matrix(["EngDispl","NumCyl"])
         X_2012 = df2.as_matrix(["EngDispl","NumCyl"])
         FE_Y_2010 = df.as_matrix(["FE"])
         FE_Y_2011 = df1.as_matrix(["FE"])
         FE_Y_2012 = df2.as_matrix(["FE"])
```

```
In [14]: X0 = np.column_stack((X_2010,np.ones(FE_Y_2010.shape[0])))
         X1 = np.column_stack((X_2011,np.ones(FE_Y_2011.shape[0])))
         X2 = np.column_stack((X_2012,np.ones(FE_Y_2012.shape[0])))
```

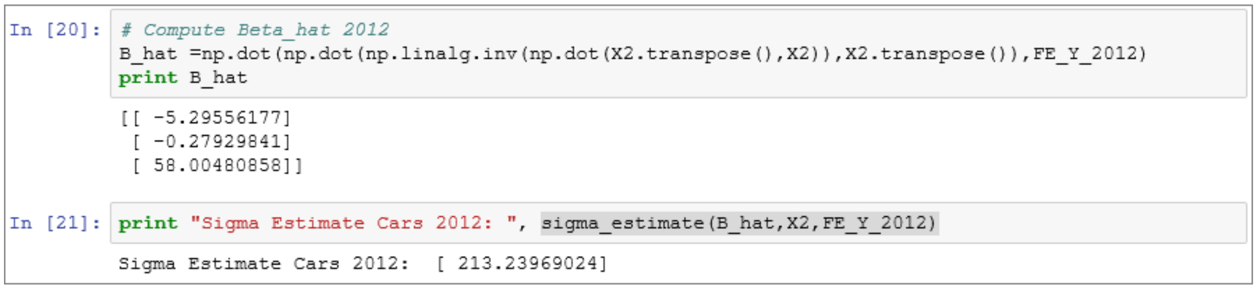
```
In [15]: # Estimate Sigma
         def sigma_estimate(B,X,Y):
             b0 = B[2]
             b1 = B[1]
             b2 = B[0]
             sum = 0.0
             for i in np.arange(len(X)):
                 sum += np.square(Y[i]-(b0 + np.dot(X[i][0],b1) + np.dot(X[i][1],b2)))
             return (sum / (len(X) - 2))
```



```
In [16]: # Compute Beta_hat 2010
         B_hat =np.dot(np.dot(np.linalg.inv(np.dot(X0.transpose(),X0)),X0.transpose()),FE_Y_2010)
         print B_hat

         [[ -3.74535214]
          [ -0.58802919]
          [ 51.35414193]]

In [17]: print "Sigma Estimate Cars 2010: ", sigma_estimate(B_hat,X0,FE_Y_2010)

         Sigma Estimate Cars 2010:  [ 89.94853481]
```

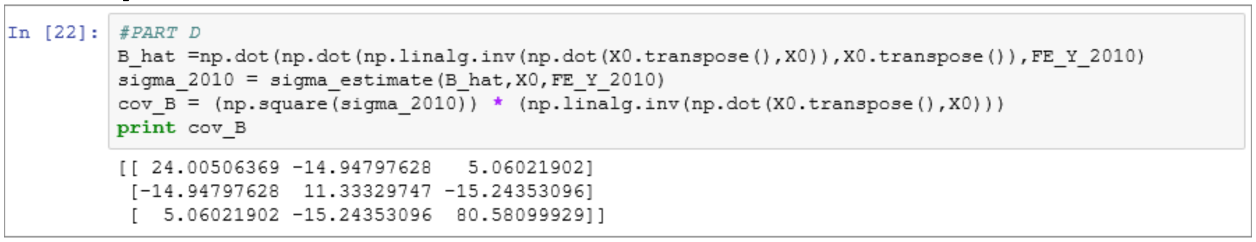


```
In [18]: # Compute Beta_hat 2011
         B_hat =np.dot(np.dot(np.linalg.inv(np.dot(X1.transpose(),X1)),X1.transpose()),FE_Y_2011)
         print B_hat

         [[ -3.96769213]
          [ -1.13215237]
          [ 56.10527854]]

In [19]: print "Sigma Estimate Cars 2011: ", sigma_estimate(B_hat,X1,FE_Y_2011)

         Sigma Estimate Cars 2011:  [ 80.00672933]
```



```
In [20]: # Compute Beta_hat 2012
         B_hat =np.dot(np.dot(np.linalg.inv(np.dot(X2.transpose(),X2)),X2.transpose()),FE_Y_2012)
         print B_hat

         [[ -5.29556177]
          [ -0.27929841]
          [ 58.00480858]]

In [21]: print "Sigma Estimate Cars 2012: ", sigma_estimate(B_hat,X2,FE_Y_2012)

         Sigma Estimate Cars 2012:  [ 213.23969024]
```

Q2 Part D]



```
In [22]: #PART D
         B_hat =np.dot(np.dot(np.linalg.inv(np.dot(X0.transpose(),X0)),X0.transpose()),FE_Y_2010)
         sigma_2010 = sigma_estimate(B_hat,X0,FE_Y_2010)
         cov_B = (np.square(sigma_2010)) * (np.linalg.inv(np.dot(X0.transpose(),X0)))
         print cov_B

         [[ 24.00506369 -14.94797628   5.06021902]
          [-14.94797628  11.33329747 -15.24353096]
          [  5.06021902 -15.24353096  80.58099929]]
```

Q2 Part E]

I think there is some relationship between the standard deviation for each hear and the computed beta hat. The 2012 dataset had the highest standard deviation of the three datasets.

Q3 Part A]

```
In [4]: # PART A - 2010 data
```

```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import datasets, linear_model
        import pandas as pd
        from pandas import DataFrame, Series
        from __future__ import division
```

```
In [6]: df=pd.read_stata("cars2010.dta")
```

```
In [7]: # get XMean
        X0 = df.as_matrix(["EngDispl","NumCyl"])
        FE_Y_2010 = df.as_matrix(["FE"])
        Xmean=[]
        Xmean.append(np.mean(X0[:,0]))
        Xmean.append(np.mean(X0[:,1]))
        print Xmean

        [3.5074074074074071, 5.9710930442637764]
```

```
In [8]: sigma=[]
        sigma.append(np.std(X0[:,0]))
        sigma.append(np.std(X0[:,1]))
        print sigma
        Ymean = np.mean(FE_Y_2010)
        print Ymean

        [1.3053151226197799, 1.8997158959884082]
        34.7064890696
```

```
In [9]: std_X0=X0
        for i in np.arange(len(X0)):
            for j in np.arange(len(X0[0])):
                std_X0[i][j]=(X0[i][j] - Xmean[j])/sigma[j]
        print std_X0

        [[ 0.91364344  1.06800546]
         [ 0.91364344  1.06800546]
         [ 0.53059417  1.06800546]
         ...,
         [-0.23550436  0.01521646]
         [-0.23550436  0.01521646]
         [ 0.68381388  1.06800546]]
```

```
In [10]: Y_Center = FE_Y_2010
         for i in np.arange(len(FE_Y_2010)):
             Y_Center[i]=FE_Y_2010[i]-Ymean
         print Y_Center

         [[-6.68668907]
          [-9.09708907]
          [-7.90648907]
          ...,
          [-4.21388907]
          [-4.96338907]
          [-8.50648907]]
```

Q3 Part B]

```
In [9]:  #Part B
         lamda = 0.01
         model_rr = linear_model.Ridge(alpha =lamda)
         model_rr.fit(std_X0, Y_Center)
         coeff=model_rr.coef_
         print 'When Lambda=', lamda
         print 'B1 =', coeff[0][0]
         print 'B2 =', coeff[0][1]
         print 'B0 ', model_rr.intercept_
         print 'r2 = ', model_rr.score(std_X0, Y_Center)

         When Lambda= 0.01
         B1 = -4.88866883941
         B2 = -1.11725589258
         B0  [ -8.38049530e-16]
         r2 =  0.623958939675
```

```
In [10]: lamda = 5
         model_rr = linear_model.Ridge(alpha =lamda)
         model_rr.fit(std_X0, Y_Center)
         coeff=model_rr.coef_
         print 'When Lambda=', lamda
         print 'B1 =', coeff[0][0]
         print 'B2 =', coeff[0][1]
         print 'B0 ', model_rr.intercept_
         print 'r2 = ', model_rr.score(std_X0, Y_Center)

         When Lambda= 5
         B1 = -4.79507488508
         B2 = -1.19668137988
         B0  [ -8.77082684e-16]
         r2 =  0.62393043522
```

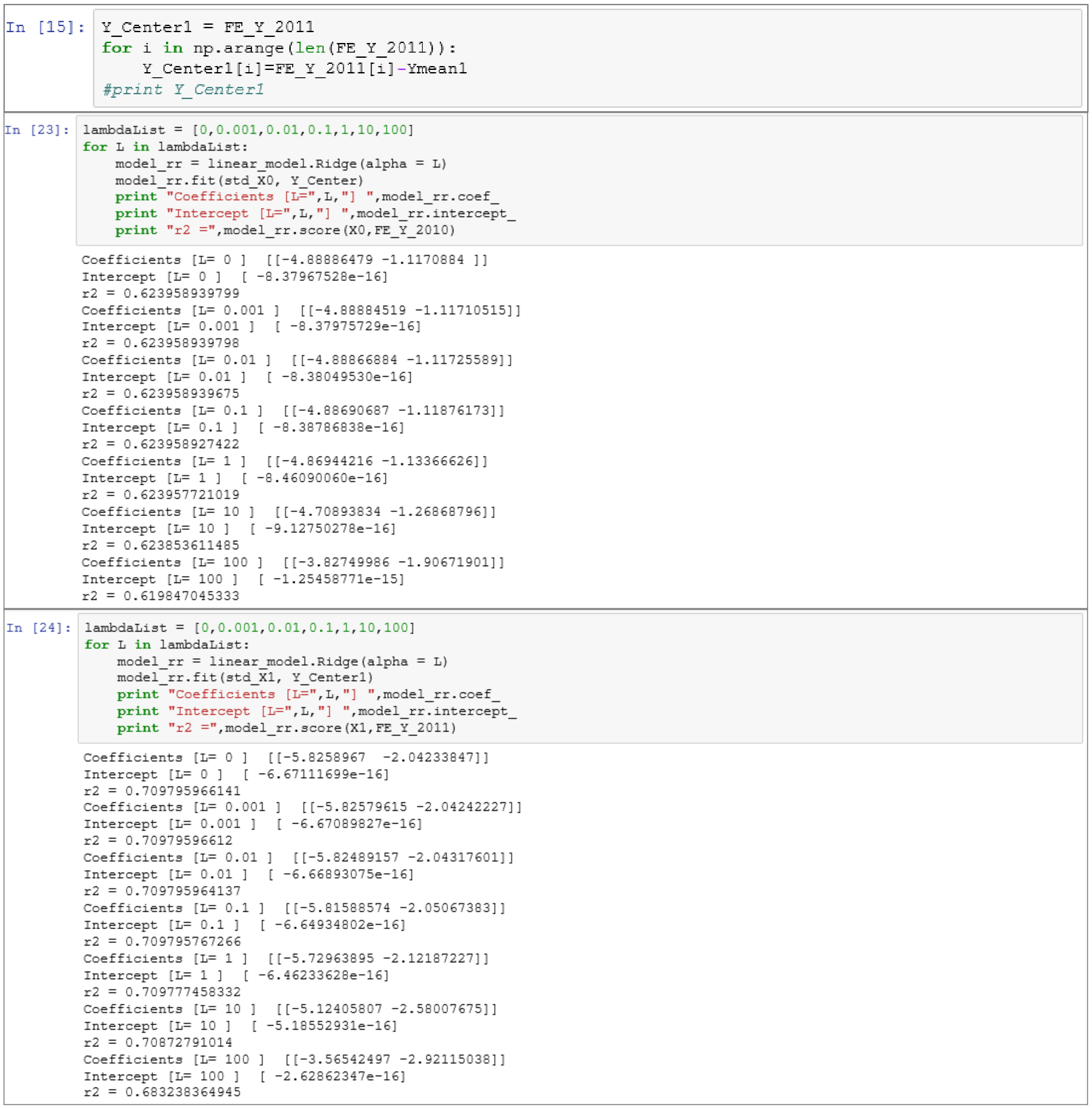
```
In [11]: lamda = 10000
         model_rr = linear_model.Ridge(alpha =lamda)
         model_rr.fit(std_X0, Y_Center)
         coeff=model_rr.coef_
         print 'When Lambda=', lamda
         print 'B1 =', coeff[0][0]
         print 'B2 =', coeff[0][1]
         print 'B0 ', model_rr.intercept_
         print 'r2 = ', model_rr.score(std_X0, Y_Center)

         When Lambda= 10000
         B1 = -0.542643100054
         B2 = -0.503905302656
         B0  [ -1.64280226e-15]
         r2 =  0.194972254986
```

Q3 Part C]

```
In [12]: # Part C
         df1=pd.read_stata("cars2011.dta")
         X1 = df1.as_matrix(["EngDispl","NumCyl"])
         FE_Y_2011 = df1.as_matrix(["FE"])
         X1mean=[]
         X1mean.append(np.mean(X1[:,0]))
         X1mean.append(np.mean(X1[:,1]))
```

```
In [13]: sigma1=[]
         sigma1.append(np.std(X1[:,0]))
         sigma1.append(np.std(X1[:,1]))
         Ymean1 = np.mean(FE_Y_2011)
```

```
In [14]: std_X1=X1
         for i in np.arange(len(X1)):
             for j in np.arange(len(X1[0])):
                 std_X1[i][j]=(X1[i][j] - X1mean[j])/sigma1[j]
```



```
In [15]:  Y_Center1 = FE_Y_2011
          for i in np.arange(len(FE_Y_2011)):
              Y_Center1[i]=FE_Y_2011[i]-Ymean1
          #print Y_Center1
```

```
In [23]:  lambdaList = [0,0.001,0.01,0.1,1,10,100]
          for L in lambdaList:
              model_rr = linear_model.Ridge(alpha = L)
              model_rr.fit(std_X0, Y_Center)
              print "Coefficients [L=",L,"] ",model_rr.coef_
              print "Intercept [L=",L,"] ",model_rr.intercept_
              print "r2 =",model_rr.score(X0,FE_Y_2010)
```

```
Coefficients [L= 0 ]  [[-4.88886479 -1.1170884 ]]
Intercept [L= 0 ]  [ -8.37967528e-16]
r2 = 0.623958939799
Coefficients [L= 0.001 ]  [[-4.88884519 -1.11710515]]
Intercept [L= 0.001 ]  [ -8.37975729e-16]
r2 = 0.623958939798
Coefficients [L= 0.01 ]  [[-4.88866884 -1.11725589]]
Intercept [L= 0.01 ]  [ -8.38049530e-16]
r2 = 0.623958939675
Coefficients [L= 0.1 ]  [[-4.88690687 -1.11876173]]
Intercept [L= 0.1 ]  [ -8.38786838e-16]
r2 = 0.623958927422
Coefficients [L= 1 ]  [[-4.86944216 -1.13366626]]
Intercept [L= 1 ]  [ -8.46090060e-16]
r2 = 0.623957721019
Coefficients [L= 10 ]  [[-4.70893834 -1.26868796]]
Intercept [L= 10 ]  [ -9.12750278e-16]
r2 = 0.623853611485
Coefficients [L= 100 ]  [[-3.82749986 -1.90671901]]
Intercept [L= 100 ]  [ -1.25458771e-15]
r2 = 0.619847045333
```

```
In [24]:  lambdaList = [0,0.001,0.01,0.1,1,10,100]
          for L in lambdaList:
              model_rr = linear_model.Ridge(alpha = L)
              model_rr.fit(std_X1, Y_Center1)
              print "Coefficients [L=",L,"] ",model_rr.coef_
              print "Intercept [L=",L,"] ",model_rr.intercept_
              print "r2 =",model_rr.score(X1,FE_Y_2011)
```

```
Coefficients [L= 0 ]  [[-5.8258967  -2.04233847]]
Intercept [L= 0 ]  [ -6.67111699e-16]
r2 = 0.709795966141
Coefficients [L= 0.001 ]  [[-5.82579615 -2.04242227]]
Intercept [L= 0.001 ]  [ -6.67089827e-16]
r2 = 0.70979596612
Coefficients [L= 0.01 ]  [[-5.82489157 -2.04317601]]
Intercept [L= 0.01 ]  [ -6.66893075e-16]
r2 = 0.709795964137
Coefficients [L= 0.1 ]  [[-5.81588574 -2.05067383]]
Intercept [L= 0.1 ]  [ -6.64934802e-16]
r2 = 0.709795767266
Coefficients [L= 1 ]  [[-5.72963895 -2.12187227]]
Intercept [L= 1 ]  [ -6.46233628e-16]
r2 = 0.709777458332
Coefficients [L= 10 ]  [[-5.12405807 -2.58007675]]
Intercept [L= 10 ]  [ -5.18552931e-16]
r2 = 0.70872791014
Coefficients [L= 100 ]  [[-3.56542497 -2.92115038]]
Intercept [L= 100 ]  [ -2.62862347e-16]
r2 = 0.683238364945
```

Q3 Part D]

Q3 Part E]

We need three separated datasets for the following reasons:

- Training – Develop optimal parameters for the model we intend to use
- Cross Validation – Test the quality of the model against data not sampled to test predicted result against known results
- Testing – Exercise model against test samples to predict results using the model we developed

Q4 Part A]

```
In [4]: # PART A
        X_train = pd.read_csv('lasso_X_train.csv', sep=',', index_col=0)
        Y_train = pd.read_csv('lasso_Y_train.csv', sep=',', index_col=0, header=None)

        X_test = pd.read_csv('lasso_X_test.csv', sep=',', index_col=0)
        Y_test = pd.read_csv('lasso_Y_test.csv', sep=',', index_col=0, header=None)

        X_cv = pd.read_csv('lasso_X_cv.csv', sep=',', index_col=0)
        Y_cv = pd.read_csv('lasso_Y_cv.csv', sep=',', index_col=0, header=None)
```

Q4 Part B]

```
In [3]: # PART B
        model_l = linear_model.Lasso(alpha=0.001)

        # Fit
        model_l.fit(X_train.values, Y_train.values)

        Train_Error = np.mean((Y_train.values - model_l.predict(X_train.values)))
        Test_Error = np.mean(np.square(Y_test.values - model_l.predict(X_test.values)))

        print "Training Error: ", Train_Error
        print "Testing Error: ", Test_Error

        Training Error:  4.0017766878e-17
        Testing Error:  10.639848409
```

Q4 Part C]

```
In [8]: # PART C
        n_lambdas = 1000
        lambdas = np.append(0.0, np.logspace(-9,-1, n_lambdas))
        model_l = linear_model.Lasso()
        errors_train = []
        errors_test = []
        errors_cv = []
```

```
In [9]: for a in lambdas:
            model_l.set_params(alpha=a)
            model_l.fit(X_train.values, Y_train.values)

            # Compute Errors for each
            errors_train.append(np.mean(np.square(Y_train.values - model_l.predict(X_train.values))))
            errors_test.append(np.mean(np.square(Y_test.values - model_l.predict(X_test.values))))
            errors_cv.append(np.mean(np.square(Y_cv.values - model_l.predict(X_cv.values))))
```

```
In [12]: print np.argmin(errors_test)
         best_lambda = lambdas[np.argmin(errors_test)]
         print "Best Lambda: ", best_lambda

         625
         Best Lambda:  9.93109181375e-05
```

Q4 Part D]

```
In [13]: #PART D
         model_l = linear_model.Lasso(alpha=best_lambda)
         model_l.fit(X_train.values, Y_train.values)

         # Compute the training/testing errors
         Train_Error = np.mean((Y_train.values - model_l.predict(X_train.values)))
         Test_Error = np.mean(np.square(Y_test.values - model_l.predict(X_test.values)))

         print "Training Error: ", Train_Error
         print "Testing Error: ", Test_Error

         Training Error:  -4.36557456851e-17
         Testing Error:  7.26776723072
```

Q4]

PROBABILITY SECTION

4. For X, Y, and Z random variables, the *Covariance Matrix* of the random vector (X, Y, Z) is defined as:

$$\mathbb{E}\left[\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}\begin{pmatrix} X & Y & Z \end{pmatrix}\right] = \mathbb{E}\begin{bmatrix} X^2 & XY & XZ \\ XY & Y^2 & YZ \\ XZ & YZ & Z^2 \end{bmatrix}.$$

Suppose X, Y and Z represent the result of rolling three independent dice. What is the covariance matrix of (X, Y, Z)?

PMF of rolling a dice

$$f_X(y) = \begin{cases} \frac{1}{6} \ if \ y \in \{1,2,3,4,5,6\} \\ 0 \qquad otherwise \end{cases}$$

$$\mu_X = E(X) = \sum_{y=1}^{6} y \cdot f_Z(y) = (1)\left(\frac{1}{6}\right) + (2)\left(\frac{1}{6}\right) + (3)\left(\frac{1}{6}\right) + (4)\left(\frac{1}{6}\right) + (5)\left(\frac{1}{6}\right) + (6)\left(\frac{1}{6}\right)$$

$$\mu_X = \frac{1+2+3+4+5+6}{6} = \frac{21}{6} = \frac{7}{2} = 3.5$$

$$\sigma_X{}^2 = var(X) = \textstyle\sum_{y=1}^{6}(y - \mu_X)^2 \cdot f_Z(y)$$

$$\sigma_X{}^2 = \left(1-\frac{7}{2}\right)^2\left(\frac{1}{6}\right) + \left(2-\frac{7}{2}\right)^2\left(\frac{1}{6}\right) + \left(3-\frac{7}{2}\right)^2\left(\frac{1}{6}\right) + \left(4-\frac{7}{2}\right)^2\left(\frac{1}{6}\right) + \left(5-\frac{7}{2}\right)^2\left(\frac{1}{6}\right) + \left(6-\frac{7}{2}\right)^2\left(\frac{1}{6}\right)$$

$$\sigma_X{}^2 = \left(-\frac{5}{2}\right)^2\left(\frac{1}{6}\right) + \left(-\frac{3}{2}\right)^2\left(\frac{1}{6}\right) + \left(-\frac{1}{2}\right)^2\left(\frac{1}{6}\right) + \left(\frac{1}{2}\right)^2\left(\frac{1}{6}\right) + \left(\frac{3}{2}\right)^2\left(\frac{1}{6}\right) + \left(\frac{5}{2}\right)^2\left(\frac{1}{6}\right)$$

$$\sigma_X{}^2 = \left(\frac{25}{24}\right) + \left(\frac{9}{24}\right) + \left(\frac{1}{24}\right) + \left(\frac{1}{24}\right) + \left(\frac{9}{24}\right) + \left(\frac{25}{24}\right) = \left(\frac{70}{24}\right) = \frac{35}{12} = 2\frac{11}{12}$$

Rules:

$E(XY) = E(X)E(Y)$

$E(X \pm a) = E(X) \pm a$

Covariance Matrix is Σ

$$\Sigma = \begin{pmatrix} E(X-3.5)^2 & E(X-3.5)(Y-3.5) & E(X-3.5)(Z-3.5) \\ E(X-3.5)(Y-3.5) & E(Y-3.5)^2 & E(Y-3.5)(Z-3.5) \\ E(X-3.5)(Z-3.5) & E(Y-3.5)(Z-3.5) & E(Z-3.5)^2 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 2\frac{11}{12} & E(X-3.5)E(Y-3.5) & E(X-3.5)E(Z-3.5) \\ E(X-3.5)E(Y-3.5) & 2\frac{11}{12} & E(Y-3.5)E(Z-3.5) \\ E(X-3.5)E(Z-3.5) & E(Y-3.5)E(Z-3.5) & 2\frac{11}{12} \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 2\frac{11}{12} & 0 & 0 \\ 0 & 2\frac{11}{12} & 0 \\ 0 & 0 & 2\frac{11}{12} \end{pmatrix}$$

Q7]

7. Consider flipping a fair coin. Let Z denote the random variable that is the number of Heads that come up in a row. Thus, if the first flip comes up tails, $Z = 0$. If the flip sequence is

$$HHTHHHHT....$$

then $Z = 2$, and so on.

- Write the probability mass function of Z.
- Compute the mean and variance of Z.

For a Fair Coin.
P(H) = $p = \frac{1}{2}$
P(T) = $(1-p) = \frac{1}{2}$

Random Variable

Compound Event	Elementary Event	Probability
(Z=0)	(T..) =	$\frac{1}{2}$
(Z=1)	(HT..)	$\left(\frac{1}{2}\right)^1\left(\frac{1}{2}\right) = \frac{1}{4}$
(Z=2)	(HHT..)	$\left(\frac{1}{2}\right)^2\left(\frac{1}{2}\right) = \frac{1}{8}$
(Z=3)	(HHHT..)	$\left(\frac{1}{2}\right)^3\left(\frac{1}{2}\right) = \frac{1}{16}$
(Z=4)	(HHHHT..)	$\left(\frac{1}{2}\right)^4\left(\frac{1}{2}\right) = \frac{1}{32}$

PMF:

y	$f_Z(y)$	
0	$\frac{1}{2}$	$p^0(1-p)$
1	$\left(\frac{1}{2}\right)^1\left(\frac{1}{2}\right) = \frac{1}{4}$	$p^1(1-p)$
2	$\left(\frac{1}{2}\right)^2\left(\frac{1}{2}\right) = \frac{1}{8}$	$p^2(1-p)$
3	$\left(\frac{1}{2}\right)^3\left(\frac{1}{2}\right) = \frac{1}{16}$	$p^3(1-p)$
4	$\left(\frac{1}{2}\right)^4\left(\frac{1}{2}\right) = \frac{1}{32}$	$p^4(1-p)$

So **PMF** for Z is $f_Z(y) = p^y(1-p)$ or for fair coin where $p = (1-p)$

$$f_Z(y) = p^{y+1} = \left(\tfrac{1}{2}\right)^{y+1} \qquad \text{where } y = 0,1,2,\dots,n$$

Compute Mean & Variance:

$$\mu = E(Z) = \sum_{y=0}^{n} y \cdot f_Z(y) = \sum_{y=0}^{n} y \cdot \left(\frac{1}{2}\right)^{y+1} = \sum_{y=1}^{n} y \cdot \left(\frac{1}{2}\right)^{y+1} = \frac{1}{4}\sum_{y=1}^{n} y \cdot \left(\frac{1}{2}\right)^{y-1}$$

From http://www.trans4mind.com/personal_development/mathematics/series/airthmeticGeometricSeries.htm

$$\boxed{1 + 2\,r + 3\,r^2 \dots n\,r^{(n-1)} = \frac{\left(1 - (n+1)\,r^n + n\,r^{(n+1)}\right)}{(1-r)^2}}$$

$r = \dfrac{1}{2}$

$$\mu = E(Z) = \frac{1}{4}\sum_{y=1}^{n} y \cdot \left(\frac{1}{2}\right)^{y-1} = \frac{1}{4}\frac{\left(1 - (n+1)\left(\frac{1}{2}\right)^n + n\left(\frac{1}{2}\right)^{n+1}\right)}{\left(\frac{1}{2}\right)^2}$$

$$\mu = \frac{1}{4}\frac{\left(1 - (n+1)\left(\frac{1}{2}\right)^n + n\left(\frac{1}{2}\right)^{n+1}\right)}{\frac{1}{4}} = \left(1 + n\left(\frac{1}{2}\right)^{n+1} - (n+1)\left(\frac{1}{2}\right)^n\right) =$$

$$\mu = \left(1 + n\left(\frac{1}{2}\right)^{n+1} - (n+1)\left(\frac{1}{2}\right)^n\right) = \left(1 + \left(\frac{1}{2}\right)n\left(\frac{1}{2}\right)^n - (n+1)\left(\frac{1}{2}\right)^n\right)$$

$\lim_{n\to\infty}\left(\frac{1}{2}\right)^n = 0$, then $\mu = 1$

Variance:
Use:

Properties of Variance

- $\operatorname{var}(X) = E\{(X - E[X])^2\} = E[X^2] - (E[X])^2$

Take advantage of:
$E[Z^2] = E[Z(Z-1)] + E[Z]$

Compute:
$E[Z(Z-1)] = \sum_{y=1}^{n} y(y-1) \cdot \left(\frac{1}{2}\right)^{y+1}$

$\sigma_Z{}^2 = E[(Z-\mu)^2] = E[Z^2] - (E[Z])^{\,2} = E[Z^2] - \mu^{\,2}$

$$\sum_{y=0}^{n}(y-\mu)^2 f_Z(y) = \sum_{y=0}^{n}(y-1)^2\left(\frac{1}{2}\right)^{y+1} = \sum_{y=0}^{n}(y-1)^2\left(\frac{1}{2}\right)^{y+1}$$

$$\sigma_Z{}^2 = (0-1)^2\left(\frac{1}{2}\right)^{0+1} + \sum_{y=1}^{n}(y-1)^2\left(\frac{1}{2}\right)^{y+1} = \frac{1}{2} + \sum_{y=1}^{n}(y-1)^2\left(\frac{1}{2}\right)^{y+1}$$

Use from http://math2.org/math/expansion/power.htm:

$\sum_{k=1}^{n} k^2 = \frac{1}{6}n(n+1)(2n+1) \quad \rightarrow \quad \sum_{j=1}^{n}(j-1)^2 = (1-1)^2 + \sum_{j=2}^{n}(j-1)^2 = \sum_{k=1}^{n} k^2$

Use from http://math2.org/math/expansion/geom.htm:

$$\sum_{k=1}^{n} k^2$$

Revisit with:
http://arnoldkling.com/apstats/geometric.html