Untitled

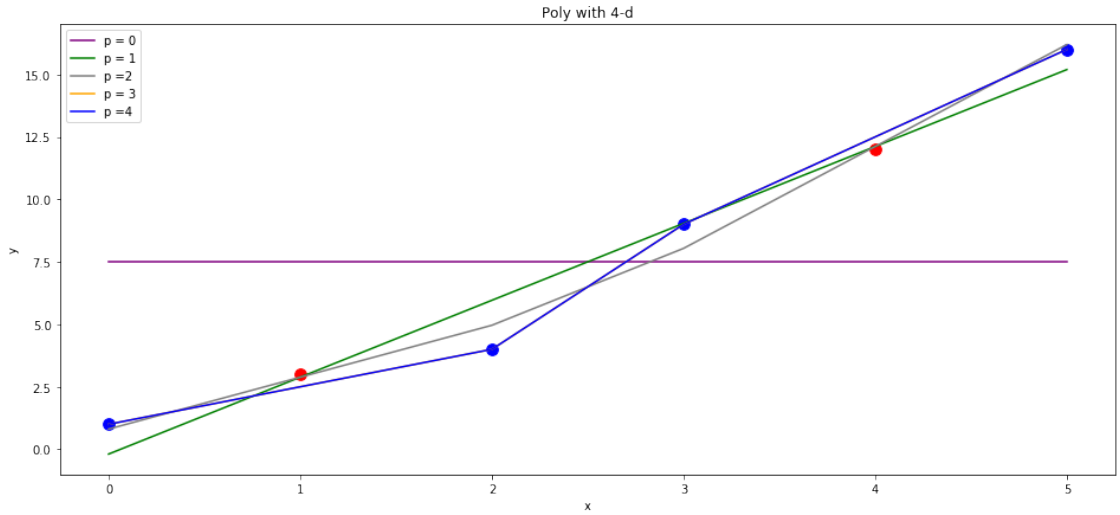
September 23, 2019

[24]:
```
# QUESTION 1
#␣
↪=============================================================================
#␣
↪=============================================================================
#␣
↪=============================================================================
#␣
↪=============================================================================

import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Present the training and testing data points
###################################################
x_train = np.array([ [0],[2],[3],[5] ])
y_train = [1,4,9,16]

x_test = np.array([ [1],[4] ])
y_test = [3,12]

area = np.pi*30

plt.figure(figsize=(16,7))
plt.scatter(x_train, y_train, s=area, c='blue', alpha=1)
plt.scatter(x_test, y_test, s=area, c='red', alpha=1)
plt.title('Poly with 4-d')
plt.xlabel('x')
plt.ylabel('y')
###################################################

# LinearRegression
###################################################
# lin = LinearRegression()
# lin.fit(x_train,y_train)
```

```
# plt.plot(x_train, lin.predict(x_train), color = 'red')
##################################################


# D = 0
##################################################
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 0)
X_poly = poly.fit_transform(x_train)
poly.fit(X_poly, y_train)

lin0 = LinearRegression()
lin0.fit(X_poly,y_train)
plt.plot(x_train, lin0.predict(poly.fit_transform(x_train)), color = 'purple',␣
 ↪label = 'p = 0')
train_result0 = lin0.predict(poly.fit_transform(x_train))
test_result0 = lin0.predict(poly.fit_transform(x_test))

# D = 1
##################################################
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 1)
X_poly = poly.fit_transform(x_train)
poly.fit(X_poly, y_train)

lin1 = LinearRegression()
lin1.fit(X_poly,y_train)
plt.plot(x_train, lin1.predict(poly.fit_transform(x_train)), color = 'green',␣
 ↪label = 'p = 1')
train_result1 = lin1.predict(poly.fit_transform(x_train))
test_result1 = lin1.predict(poly.fit_transform(x_test))

# D = 2
##################################################
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(x_train)
poly.fit(X_poly, y_train)

lin2 = LinearRegression()
lin2.fit(X_poly,y_train)
plt.plot(x_train, lin2.predict(poly.fit_transform(x_train)), color = 'grey',␣
 ↪label = 'p =2')
```

```
train_result2 = lin2.predict(poly.fit_transform(x_train))
test_result2 = lin2.predict(poly.fit_transform(x_test))

# D = 3
##################################################
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(x_train)
poly.fit(X_poly, y_train)

lin3 = LinearRegression()
lin3.fit(X_poly,y_train)
plt.plot(x_train, lin3.predict(poly.fit_transform(x_train)), color = 'orange',␣
 ↪label = 'p = 3')
train_result3 = lin3.predict(poly.fit_transform(x_train))
test_result3 = lin3.predict(poly.fit_transform(x_test))

# D = 4
##################################################
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 4)
X_poly = poly.fit_transform(x_train)
poly.fit(X_poly, y_train)

lin4 = LinearRegression()
lin4.fit(X_poly,y_train)
plt.plot(x_train, lin4.predict(poly.fit_transform(x_train)), color = 'blue',␣
 ↪label = 'p =4')
train_result4 = lin4.predict(poly.fit_transform(x_train))
test_result4 = lin4.predict(poly.fit_transform(x_test))

# D = 8
##################################################
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 8)
X_poly = poly.fit_transform(x_train)
poly.fit(X_poly, y_train)

lin20 = LinearRegression()
lin20.fit(X_poly,y_train)

test_result20 = lin20.predict(poly.fit_transform(x_test))
```

```
plt.legend()
plt.show()
```



[26]:
```
# Training Result
################################################

train_result = ['','','','']
train_result = np.row_stack((train_result,train_result0))
train_result = np.row_stack((train_result,train_result1))
train_result = np.row_stack((train_result,train_result2))
train_result = np.row_stack((train_result,train_result3))
train_result = np.row_stack((train_result,train_result4))


train_result = np.delete(train_result, (0), axis=0)

print("------ train_result ------")
print(train_result)


# Test Result
################################################

# x_train_matrix = [0,0,0]
# x_train_matrix = np.row_stack((x_train_matrix,x_1))
# x_train_matrix = np.row_stack((x_train_matrix,x_2))
# x_train_matrix = np.row_stack((x_train_matrix,x_3))
# x_train_matrix = np.row_stack((x_train_matrix,x_4))
# x_train_matrix = np.delete(x_train_matrix, (0), axis=0)
```

```
test_result = ['','']
test_result = np.row_stack((test_result,test_result0))
test_result = np.row_stack((test_result,test_result1))
test_result = np.row_stack((test_result,test_result2))
test_result = np.row_stack((test_result,test_result3))
test_result = np.row_stack((test_result,test_result4))
test_result = np.delete(test_result, (0), axis=0)

print("------ test_result ------")
print(test_result)



# Typical Bias
################################################
x_test = np.array([ [1],[4] ])
y_test = [3,12]


Typical_Bias_0 = pow( (float(test_result[0][0]) - float(x_test[0])),2 ) + pow(␣
 ↪(float(test_result[0][1]) - float(x_test[1])), 2)/2
Typical_Bias_1 = pow( (float(test_result[1][0]) - float(x_test[0])),2 ) + pow(␣
 ↪(float(test_result[1][1]) - float(x_test[1])), 2)/2
Typical_Bias_2 = pow( (float(test_result[2][0]) - float(x_test[0])),2 ) + pow(␣
 ↪(float(test_result[2][1]) - float(x_test[1])), 2)/2
Typical_Bias_3 = pow( (float(test_result[3][0]) - float(x_test[0])),2 ) + pow(␣
 ↪(float(test_result[3][1]) - float(x_test[1])), 2)/2
Typical_Bias_4 = pow( (float(test_result[4][0]) - float(x_test[0])),2 ) + pow(␣
 ↪(float(test_result[4][1]) - float(x_test[1])), 2)/2


Typical_Bias = []
Typical_Bias.append(Typical_Bias_0)
Typical_Bias.append(Typical_Bias_1)
Typical_Bias.append(Typical_Bias_2)
Typical_Bias.append(Typical_Bias_3)
Typical_Bias.append(Typical_Bias_4)

print(Typical_Bias)

# Variance
################################################

Variance = []
k = 0

print(len(Typical_Bias))
```

```
for i in range(len(Typical_Bias)):
    Variance.append( Typical_Bias[k]/2 )
    k += 1

print(Variance)

# Total Error
################################################
Total_Error = []

for i in range(len(test_result)):
    Total_Error.append(float(test_result[i][0]) - float(x_test[0])  + 
 ↪float(test_result[i][1]) - float(x_test[1]) )

print(Total_Error)

# Training Error
################################################
x_train = np.array([ [0],[2],[3],[5] ])
y_train = [1,4,9,16]

training_error = []
for i in train_result:# Get every training result

    k = 0
    training_error_sum = 0
    for j in i:
        training_error_sum = training_error_sum + pow( y_train[int(k)] -
 ↪float(i[k]), 2 )
        k += 1

    training_error.append(training_error_sum/4)

print("------- Training Error ------")
print(training_error)



# Test Error
################################################
testing_error = []

for i in test_result:# Get every training result

    k = 0
    testing_error_sum = 0
```

```
    for j in i:
        testing_error_sum = testing_error_sum + pow( y_test[int(k)] -␣
↪float(i[k]), 2 )
        k += 1

    testing_error.append(testing_error_sum/4)

print("------- Testing Error ------")
print(testing_error)


# Plot
################################################
x_point = [0,1,2,3,4]

plt.figure(figsize=(14,7))

plt.plot(x_point, Typical_Bias , color = 'yellow', label = 'Typical_Bias')
plt.plot(x_point, Variance , color = 'blue', label = 'Variance')
plt.plot(x_point, Total_Error , color = 'pink', label = 'Total_Error')
plt.plot(x_point, training_error , color = 'black', label = 'Training_error')
plt.plot(x_point, testing_error , color = 'grey', label = 'Testing_error')

plt.title('Error')
plt.xlabel('x')
plt.ylabel('y')

plt.legend()
plt.show()

######################################################
# (3) Explain why each of the five curves has the shape displayed in part (2).
# --> Here we can see that when d=2 the error come to the minumum. Why? Because␣
↪1) when d is 0 or 1, it is not enough to fit the data probably.2) While when␣
↪d > 2, the curve is overfited - which means the curve is fluctuate too muc -␣
↪which away from the ground truth more and more when d become large




import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns  # data visualization library
from scipy.optimize import minimize
from sklearn.datasets import load_boston
from sklearn.model_selection import KFold
```

```
boston_dataset = load_boston()
df = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

print (df.head(5))

#the data can be downloaded from "https://github.com/jcrouser/islr-python/blob/
 ↪master/data/Smarket.csv"
df = pd.read_csv('Smarket.csv', usecols=range(0,10), index_col=0,␣
 ↪parse_dates=True)

X = df[['Lag1','Lag2']].values
Y = df['Today'].values


print(X[:,1])

print(len(X))
print(len(Y))

##########################################
def forward(X_1,X_2,params):
    # 3d Line Z = aX_1 + bX_2 +c
    return X_1.dot(params[:1]) + X_2.dot(params[1:2]) + params[2]

def cost_function(params,X_1,X_2,y,p):
    error_vector = y - forward(X_1,X_2,params)
    return np.linalg.norm(error_vector, ord=p)
##########################################

kf = KFold(n_splits=5)
kf.get_n_splits(X)

#print(kf)
#KFold(n_splits=5, random_state=None, shuffle=False)

#print("------------------------------")
#print(type(kf.split(X)))
#


L2_Norm_P1 = 0
L2_Norm_P2 = 0
for train_index, test_index in kf.split(X):
  #  print("TRAIN:", train_index, "TEST:", test_index)

    ### For X part
    X_train, X_test = X[train_index], X[test_index]
```

```
    ### For Y part
    Y_train, Y_test = Y[train_index], Y[test_index]

    ############################ P = 1
    output1 = minimize(  cost_function, [0.5,0.5,1], args=(  np.c_[X_train[:
↪,0]], np.c_[X_train[:,1]],Y_train,1    )    )
    y_hat1 = forward(np.c_[X_train[:,0]],np.c_[X_train[:,1]], output1.x) #␣
↪print(y_hat1)

    ### Validation
    L2_Norm_Sum = 0

    for i in range(len(Y_test)):
         L2 = np.power((y_hat1[i] - Y_test[i]),2)
         L2_Norm_Sum = L2_Norm_Sum + L2

    # Add all together
    L2_Norm_P1 = L2_Norm_P1 + np.sqrt(L2_Norm_Sum)


    ############################ P = 2
    ### Training
    output2 = minimize(  cost_function, [0.5,0.5,1], args=(  np.c_[X_train[:
↪,0]], np.c_[X_train[:,1]],Y_train,2    )    )
    y_hat2 = forward(np.c_[X_train[:,0]],np.c_[X_train[:,1]], output2.x) #␣
↪print(y_hat2)

    ### Validation
    L2_Norm_Sum = 0

    for i in range(len(Y_test)):
         L2 = np.power((y_hat2[i] - Y_test[i]),2)
         L2_Norm_Sum = L2_Norm_Sum + L2

    # Add all together
    L2_Norm_P2 = L2_Norm_P2 + np.sqrt(L2_Norm_Sum)


L2_Norm_P1_Mean = L2_Norm_P1/5
L2_Norm_P2_Mean = L2_Norm_P2/5


print(L2_Norm_P1_Mean)
print(L2_Norm_P2_Mean)

############################################################
```

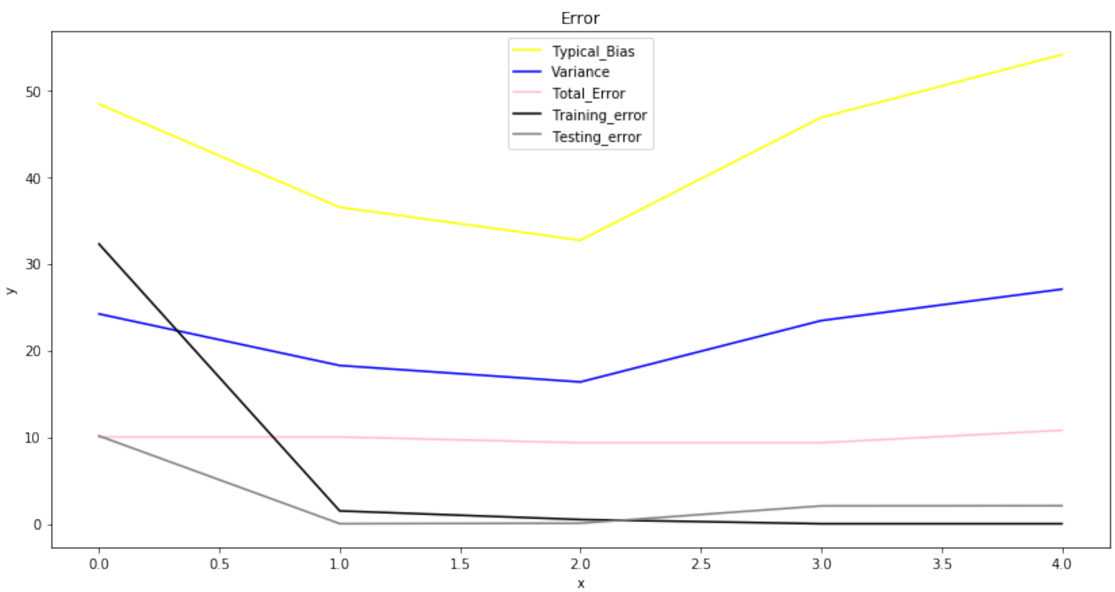
```
# Please compare the mean values across all the 5 folds together. Justify your results.
# --> I believe the reason is that, after cross validation, the error can be minimized. Thus even we trained with different P value, the result is similar




# QUESTION 2
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
# ===========================================================================================
```

```
# ============================================================================================
# ============================================================================================




# QUESTION 3
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
# ============================================================================================
```

```
#
 ↪==========================================================================================
#
 ↪==========================================================================================
#
 ↪==========================================================================================



# ============================================================================
# (2) Logistic Regression Regularization Comparison with Bootstrapping:

#Using 80% of the data as a training set and 20% as a testing set in each
 ↪bootstrap repeated 1000 times each, please implement and compare the average
 ↪and the standard deviation of coefficients obtained from Ridge regression
 ↪and LASSO regularized logistic regression.

#By averaging the coefficients it is meant that all the different coefficient
 ↪values for a specific variable is averaged. Coefficient average values then
 ↪are to be compared by plotting them against each other. Are all input
 ↪features necessary? Please describe how categorical features are handled.

# (3) Please plot the ROC curve for both models for a single bootstrap data.
 ↪What are the area under the curve measurements?
# (4) What is the optimal decision threshold to maximize the f1 score?
# (5) Please provide a mean and standard deviation for the AUROCs for each
 ↪model . (6) Please provide a mean and standard deviation for the f1 score
 ↪for each model.
# ============================================================================
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score
```

```
------ train_result ------
[['7.5' '7.5' '7.5' '7.5']
 ['-0.19230769230769518' '5.961538461538461' '9.038461538461538'
  '15.192307692307695']
 ['0.8076923076923119' '4.961538461538469' '8.038461538461542'
  '16.19230769230768']
 ['1.0' '3.999999999999992' '9.000000000000007' '16.000000000000004']
 ['1.0000000000000226' '4.000000000000016' '9.000000000000007'
  '15.99999999999995']]
------ test_result ------
[['7.5' '7.5']
 ['2.884615384615383' '12.115384615384617']
 ['2.551282051282058' '11.78205128205128']
```

```
 ['0.6666666666666536' '13.66666666666668']
 ['1.3849813115868361' '14.384981311586802']]
[48.375, 36.48150887573965, 32.686637080867854, 46.83333333333348,
54.07212903127469]
5
[24.1875, 18.240754437869825, 16.343318540433927, 23.41666666666674,
27.036064515637346]
[10.0, 10.0, 9.333333333333337, 9.333333333333334, 10.769962623173639]
------- Training Error ------
[32.25, 1.4807692307692308, 0.48076923076923095, 3.1751651435145725e-29,
8.232256584047021e-28]
------- Testing Error ------
[10.125, 0.006656804733727994, 0.06221236028928212, 2.055555555555582,
2.07410530513552]
```



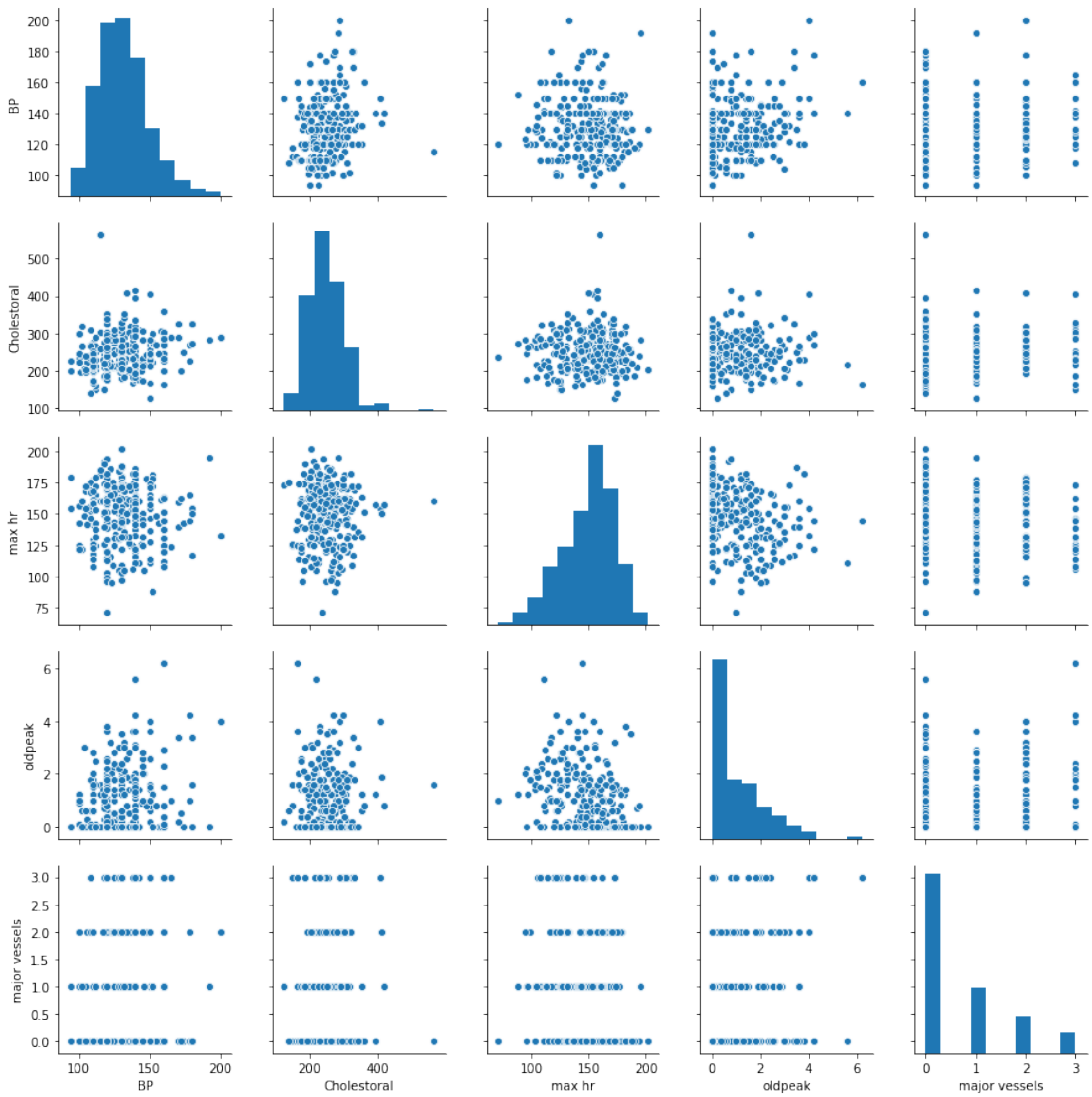

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

```
[-0.192  0.381  0.959 ...  0.043 -0.955  0.13 ]
1250
1250
16.972221525679736
16.926771309143838
```

[27]:
```
import numpy as np
import pandas as pd
import seaborn as sns

df = pd.read_csv('hw1_input.csv', usecols=range(0,14), index_col=0,␣
 ↪parse_dates=True)
```

[18]:
```
sns.pairplot(df)
# this is to give relationship between different variables. For example, here␣
 ↪we have "BP" as the x-axis and Cholestoral as the y-axis, to plot an image
```

[18]:
```
<seaborn.axisgrid.PairGrid at 0x1a25d656d8>
```




```
[9]: df.describe()
```

[9]:

	BP	Cholestoral	max hr	oldpeak
count	270.000000	270.000000	270.000000	270.00000
mean	131.344444	249.659259	149.677778	1.05000
std	17.861608	51.686237	23.165717	1.14521
min	94.000000	126.000000	71.000000	0.00000
25%	120.000000	213.000000	133.000000	0.00000
50%	130.000000	245.000000	153.500000	0.80000
75%	140.000000	280.000000	166.000000	1.60000
max	200.000000	564.000000	202.000000	6.20000

```
[13]: df.head(1)
```

[13]:

```
        Sex Chest Pain   BP  Cholestoral fasting blood sugar > 120  \
Age
70   Female   Abnormal  130         322                       No

    resting ECG  max hr angina  oldpeak slope  major vessels  defect  \
Age
70        hyper     109     No      2.4  Flat              3  Normal

    heart disease
Age
70            Yes
```

[20]:

```python
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score
```

[ ]:

```python

```