Data-X Spring 2018: Homework 05

Linear regression, logistic regression, matplotlib.

In this homework, you will do some exercises with prediction and plotting.

REMEMBER TO DISLPAY ALL OUTPUTS. If the question asks you to do something, make sure to print your results so we can easily see that you have done it.

Student: Grant Pemberton

ID Number: 3034347047

Part 1 - Regression

Data:

Data Source: Data file is uploaded to bCourses and is named: **Energy.csv**

The dataset was created by Angeliki Xifara (Civil/Structural Engineer) and was processed by Athanasios Tsanas, Oxford Centre for Industrial and Applied Mathematics, University of Oxford, UK).

Data Description:

The dataset contains eight attributes of a building (or features, denoted by X1...X8) and response being the heating load on the building, y1.

- X1 Relative Compactness
- X2 Surface Area
- X3 Wall Area
- X4 Roof Area
- X5 Overall Height
- X6 Orientation
- X7 Glazing Area
- X8 Glazing Area Distribution
- y1 Heating Load

In [144]:
```
import matplotlib.pyplot as plt # always import pyplot module as plt (st
andard)
import numpy as np
import pandas as pd
import seaborn as sns
```

Q1.1

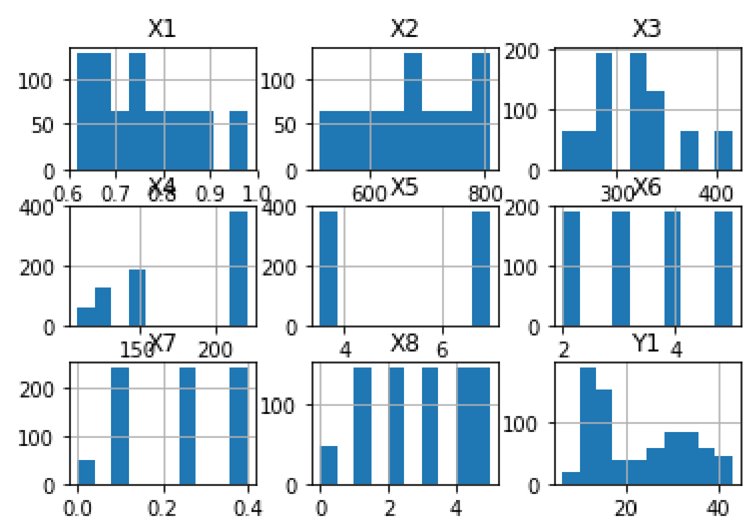
Read the data file in python. Check if there are any NaN values, and print the results.

Describe data features in terms of type, distribution range (max and min), and mean values.

Plot feature distributions. This step should give you clues about data sufficiency.

In [145]:
```
#read the file in
df = pd.read_csv('Energy.csv')


#check for NaNs and print the results
print (df.isnull().values.any())


#describe data features in terms of type, distribution range, and mean v
alues
print(df.describe())


#plot the feature distributions
df.hist()
plt.show()
#plotting the whole pairplot, but limiting it to histogram -- we'll want
 the other graphs later
#also you need to scroll down in the output to see everything
```

```
False
                X1          X2          X3          X4          X5
   X6  \
count  768.000000  768.000000  768.000000  768.000000  768.00000  768.0
00000
mean     0.764167  671.708333  318.500000  176.604167    5.25000    3.5
00000
std      0.105777   88.086116   43.626481   45.165950    1.75114    1.1
18763
min      0.620000  514.500000  245.000000  110.250000    3.50000    2.0
00000
25%      0.682500  606.375000  294.000000  140.875000    3.50000    2.7
50000
50%      0.750000  673.750000  318.500000  183.750000    5.25000    3.5
00000
75%      0.830000  741.125000  343.000000  220.500000    7.00000    4.2
50000
max      0.980000  808.500000  416.500000  220.500000    7.00000    5.0
00000

                X7         X8          Y1
count  768.000000  768.00000  768.000000
mean     0.234375    2.81250   22.307201
std      0.133221    1.55096   10.090196
min      0.000000    0.00000    6.010000
25%      0.100000    1.75000   12.992500
50%      0.250000    3.00000   18.950000
75%      0.400000    4.00000   31.667500
max      0.400000    5.00000   43.100000
```



REGRESSION: LABELS ARE CONTINUOUS VALUES. Here the model is trained to predict a continuous value for each instance. On inputting a feature vector into the model, the trained model is able to predict a continuous value for that instance.

Q 1.2: Train a linear regression model on 80 percent of the given dataset, what is the intercept value and coefficient values.

In [155]:
```
X=df.iloc[:,:-1]
#split x columns away from original dataframe

Y=df.iloc[:,8]
#split y column away from original dataframe

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
print ('Number of samples in training data:',len(x_train))
print ('Number of samples in test data:',len(x_test))
#split into test and training data


'''Source: Scikit learn
Code source: Jaques Grobler
License: BSD 3 clause'''
from sklearn import linear_model


# FEATURES
X_reg=x_train

# Y
Y_reg=y_train

# Create linear regression object
LinearRegressionModel= linear_model.LinearRegression()

# Train the model using the training sets
LinearRegressionModel.fit(X_reg, Y_reg)
Z_reg=LinearRegressionModel.predict(X_reg)


Z_test = LinearRegressionModel.predict(x_test)

print('Intercept:' , LinearRegressionModel.intercept_)

# The coefficients
print('Coefficients:', LinearRegressionModel.coef_)

lin_training_accuracy=LinearRegressionModel.score(x_train,y_train)
print ('Training Accuracy:',lin_training_accuracy)
```

```
Number of samples in training data: 614
Number of samples in test data: 154
Intercept: -7.548462311685668
Coefficients: [ 3.44831093e+00 -1.28726264e+10  1.28726264e+10  2.57452
528e+10
  3.36282953e-01  3.06109105e-03  2.20248643e+00 -1.15033502e-02]
Training Accuracy: 0.8192308188406157
```

Q.1.3: Report model performance using 'ROOT MEAN SQUARE' error metric on:

1. **Data that was used for training(Training error)**
2. **On the 20 percent of unseen data (test error)**

In [148]:
```
print("Root Mean squared error (train):",np.sqrt(np.mean((Z_reg - Y_reg)
 ** 2)))




print("Root Mean squared error (test):",np.sqrt(np.mean((Z_test - y_test
) ** 2)))
```

```
Root Mean squared error (train): 2.9293993482317626
Root Mean squared error (test): 2.895351691405278
```

Q1.4:

Lets us see the effect of amount of data on the performance of prediction model. Use varying amounts of Training data (100,200,300,400,500,all) to train regression models and report training error and validation error in each case. Validation data/Test data is the same as above for all these cases.

Plot error rates vs number of training examples. Both the training error and the validation error should be plotted. Comment on the relationship you observe in the plot, between the amount of data used to train the model and the validation accuracy of the model.

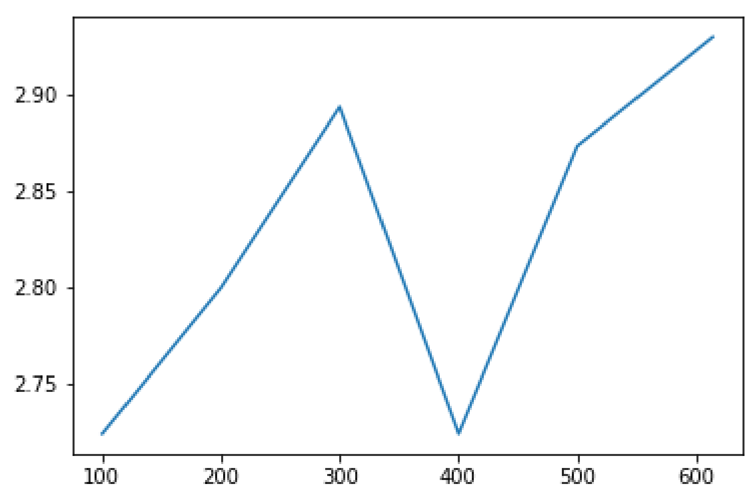
Hint: Use array indexing to choose varying data amounts

In [150]:
```
###100 data entries
X_100 = x_train[0:100]
Y_100 = y_train[0:100]

# Train the model using the training sets
LinearRegressionModel.fit(X_100, Y_100)
Z_100=LinearRegressionModel.predict(X_100)

rmse_100 = np.sqrt(np.mean((Z_100 - Y_100) ** 2))

print (rmse_100)


###200 data entries
X_200 = x_train[0:200]
Y_200 = y_train[0:200]

# Train the model using the training sets
LinearRegressionModel.fit(X_200, Y_200)
Z_200=LinearRegressionModel.predict(X_200)

rmse_200 = np.sqrt(np.mean((Z_200 - Y_200) ** 2))

print (rmse_200)

###300 data entries
X_300 = x_train[0:300]
Y_300 = y_train[0:300]

# Train the model using the training sets
LinearRegressionModel.fit(X_300, Y_300)
Z_300=LinearRegressionModel.predict(X_300)

rmse_300 = np.sqrt(np.mean((Z_300 - Y_300) ** 2))

print (rmse_300)


###400 data entries
X_400 = x_train[0:100]
Y_400 = y_train[0:100]

# Train the model using the training sets
LinearRegressionModel.fit(X_400, Y_400)
Z_400=LinearRegressionModel.predict(X_400)

rmse_400 = np.sqrt(np.mean((Z_400 - Y_400) ** 2))

print (rmse_400)



###500 data entries
X_500 = x_train[0:500]
Y_500 = y_train[0:500]
```

```
# Train the model using the training sets
LinearRegressionModel.fit(X_500, Y_500)
Z_500=LinearRegressionModel.predict(X_500)

rmse_500 = np.sqrt(np.mean((Z_500 - Y_500) ** 2))

print (rmse_500)

#all data below
X_all = x_train
Y_all = y_train

# Train the model using the training sets
LinearRegressionModel.fit(X_all, Y_all)
Z_all=LinearRegressionModel.predict(X_all)

rmse_all = np.sqrt(np.mean((Z_all - Y_all) ** 2))

print (rmse_all)
# simple plot

f, ax = plt.subplots() # returns tuple:
# f is the canvas object, can contain several plots i.e. axes objects
 (p)




ax.plot([100,200,300, 400, 500, 614],[rmse_100, rmse_200,rmse_300,rmse_4
00,rmse_500,rmse_all])
```

```
2.7240868530981803
2.7997194281217883
2.8932421515235847
2.7240868530981803
2.8730293936074265
2.9293993482317626
```

Out[150]: [<matplotlib.lines.Line2D at 0x1a1ee251d0>]



Part 2 - Classification

CLASSIFICATION: LABELS ARE DISCRETE VALUES. Here the model is trained to classify each instance into a set of predefined discrete classes. On inputting a feature vector into the model, the trained model is able to predict a class of that instance. You can also output the probabilities of an instance belnging to a class.

Q 2.1: Bucket values of 'y1' i.e 'Heating Load' from the original dataset into 3 classes:

0: 'Low' (< 14),
1: 'Medium' (14-28),
2: 'High' (>28)

This converts the given dataset into a classification problem, classes being, Heating load is: *low, medium or high*. Use this datset with transformed 'heating load' for creating a logistic regression classifiction model that predicts heating load type of a building. Use test-train split ratio of 0.8 : 0.2.

Report training and test accuracies and confusion matrices.

HINT: Use pandas.cut

In [151]:
```
df['Y1'] = pd.cut(df['Y1'], bins = [0,14,28,50], labels = [0,1,2])

df
```

Out[151]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	1
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	1
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	1
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	1
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	1
5	0.90	563.5	318.5	122.50	7.0	3	0.0	0	1
6	0.90	563.5	318.5	122.50	7.0	4	0.0	0	1
7	0.90	563.5	318.5	122.50	7.0	5	0.0	0	1
8	0.86	588.0	294.0	147.00	7.0	2	0.0	0	1
9	0.86	588.0	294.0	147.00	7.0	3	0.0	0	1
10	0.86	588.0	294.0	147.00	7.0	4	0.0	0	1
11	0.86	588.0	294.0	147.00	7.0	5	0.0	0	1
12	0.82	612.5	318.5	147.00	7.0	2	0.0	0	1
13	0.82	612.5	318.5	147.00	7.0	3	0.0	0	1
14	0.82	612.5	318.5	147.00	7.0	4	0.0	0	1
15	0.82	612.5	318.5	147.00	7.0	5	0.0	0	1
16	0.79	637.0	343.0	147.00	7.0	2	0.0	0	2
17	0.79	637.0	343.0	147.00	7.0	3	0.0	0	2
18	0.79	637.0	343.0	147.00	7.0	4	0.0	0	2
19	0.79	637.0	343.0	147.00	7.0	5	0.0	0	2
20	0.76	661.5	416.5	122.50	7.0	2	0.0	0	1
21	0.76	661.5	416.5	122.50	7.0	3	0.0	0	1
22	0.76	661.5	416.5	122.50	7.0	4	0.0	0	1
23	0.76	661.5	416.5	122.50	7.0	5	0.0	0	1
24	0.74	686.0	245.0	220.50	3.5	2	0.0	0	0
25	0.74	686.0	245.0	220.50	3.5	3	0.0	0	0
26	0.74	686.0	245.0	220.50	3.5	4	0.0	0	0
27	0.74	686.0	245.0	220.50	3.5	5	0.0	0	0
28	0.71	710.5	269.5	220.50	3.5	2	0.0	0	0
29	0.71	710.5	269.5	220.50	3.5	3	0.0	0	0
...	...	...	...	...	...	...	...	...	...
738	0.79	637.0	343.0	147.00	7.0	4	0.4	5	2

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
739	0.79	637.0	343.0	147.00	7.0	5	0.4	5	2
740	0.76	661.5	416.5	122.50	7.0	2	0.4	5	2
741	0.76	661.5	416.5	122.50	7.0	3	0.4	5	2
742	0.76	661.5	416.5	122.50	7.0	4	0.4	5	2
743	0.76	661.5	416.5	122.50	7.0	5	0.4	5	2
744	0.74	686.0	245.0	220.50	3.5	2	0.4	5	1
745	0.74	686.0	245.0	220.50	3.5	3	0.4	5	1
746	0.74	686.0	245.0	220.50	3.5	4	0.4	5	1
747	0.74	686.0	245.0	220.50	3.5	5	0.4	5	1
748	0.71	710.5	269.5	220.50	3.5	2	0.4	5	0
749	0.71	710.5	269.5	220.50	3.5	3	0.4	5	0
750	0.71	710.5	269.5	220.50	3.5	4	0.4	5	0
751	0.71	710.5	269.5	220.50	3.5	5	0.4	5	0
752	0.69	735.0	294.0	220.50	3.5	2	0.4	5	1
753	0.69	735.0	294.0	220.50	3.5	3	0.4	5	1
754	0.69	735.0	294.0	220.50	3.5	4	0.4	5	1
755	0.69	735.0	294.0	220.50	3.5	5	0.4	5	1
756	0.66	759.5	318.5	220.50	3.5	2	0.4	5	1
757	0.66	759.5	318.5	220.50	3.5	3	0.4	5	1
758	0.66	759.5	318.5	220.50	3.5	4	0.4	5	1
759	0.66	759.5	318.5	220.50	3.5	5	0.4	5	1
760	0.64	784.0	343.0	220.50	3.5	2	0.4	5	1
761	0.64	784.0	343.0	220.50	3.5	3	0.4	5	1
762	0.64	784.0	343.0	220.50	3.5	4	0.4	5	1
763	0.64	784.0	343.0	220.50	3.5	5	0.4	5	1
764	0.62	808.5	367.5	220.50	3.5	2	0.4	5	1
765	0.62	808.5	367.5	220.50	3.5	3	0.4	5	1
766	0.62	808.5	367.5	220.50	3.5	4	0.4	5	1
767	0.62	808.5	367.5	220.50	3.5	5	0.4	5	1

768 rows × 9 columns

Q2.2: One of the preprocessing steps in Data science is Feature Scaling i.e getting all our data on the same scale by setting same Min-Max of feature values. This makes training less sensitive to the scale of features . Scaling is important in algorithms that use distance based classification, SVM or K means or those that involve gradient descent optimization. If we Scale features in the range [0,1] it is called unity based normalization.

Perform unity based normalization on the above dataset and train the model again, compare model performance in training and validation with your previous model.

refer:http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler (http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler)
more at: https://en.wikipedia.org/wiki/Feature_scaling (https://en.wikipedia.org/wiki/Feature_scaling)

In [160]:

```
from sklearn import preprocessing


X=df.iloc[:,:-1]
#split x columns away from original dataframe

Y=df.iloc[:,8]
#split y column away from original dataframe


min_max_scaler = preprocessing.MinMaxScaler()

x_train_minmax = min_max_scaler.fit_transform(X)

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
 random_state=100)
print ('Number of samples in training data:',len(x_train))
print ('Number of samples in test data:',len(x_test))

LogisticRegressionModel = linear_model.LogisticRegression()

# we create an instance of logistic Regression Classifier and fit the da
ta.
print ('Training a logistic Regression Model..')
LogisticRegressionModel.fit(x_train, y_train)

training_accuracy=LogisticRegressionModel.score(x_train,y_train)
print ('Training Accuracy:', training_accuracy)

print ('Linear regression is', (lin_training_accuracy - training_accurac
y)*100, "% better than Logistic Regression for this problem")
```

```
Number of samples in training data: 614
Number of samples in test data: 154
Training a logistic Regression Model..
Training Accuracy: 0.8078175895765473
Linear regression is 1.1413229264068403 % better than Logistic Regressi
on for this problem
```

Part 3 - Matplotlib

Q 3.1a. Create a dataframe called `icecream` that has column `Flavor` with entries `Strawberry`, `Vanilla`, and `Chocolate` and another column with `Price` with entries `3.50`, `3.00`, and `4.25`.

In [68]:
```
Flavor = ['Strawberry', 'Vanilla', 'Chocolate']

Price = [3.50, 3.00, 4.25]

#icecream = pd.DataFrame(Price)

icecream = pd.DataFrame({'Flavor':Flavor, 'Price':Price})

icecream
```

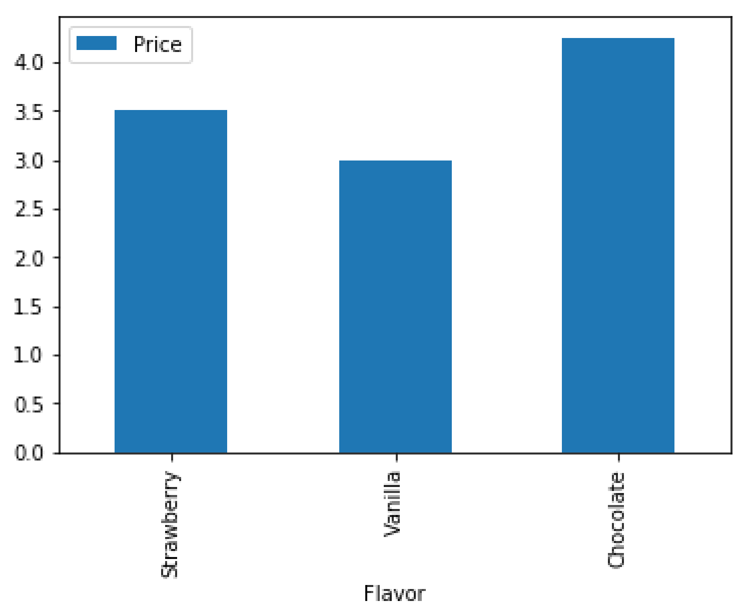
Out[68]:

	Flavor	Price
0	Strawberry	3.50
1	Vanilla	3.00
2	Chocolate	4.25

Q 3.1b Create a bar chart representing the three flavors and their associated prices.

In [69]:
```
icecream.plot.bar(x = 'Flavor')
```

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b22c358>



Q 3.2 Create 9 random plots (Hint: There is a numpy function for generating random data). The top three should be scatter plots (one with green dots, one with purple crosses, and one with blue triangles. The middle three graphs should be a line graph, a horizontal bar chart, and a histogram. The bottom three graphs should be trignometric functions (one sin, one cosine, one tangent).

In [80]:
```
N = 50
x = np.arange(1,N+1)
y = np.random.randn(N)

vals = np.random.randint(0,11,N)


xtrig = np.linspace(0,2*np.pi,10000)
y7 = np.sin(xtrig)
y8 = np.cos(xtrig)
y9 = (y7/y8)


f, ax = plt.subplots(nrows=3,ncols=3, )




ax[0,0].scatter(x, y, color = "green")
#green dots

ax[0,1].scatter(x, y, color = 'purple', marker = "+")
#purple crosses

ax[0,2].scatter(x,y, color = "blue", marker = "^")
#blue triangles

ax[1,0].plot(x, y)
#

ax[1,1].barh(x,np.abs(y))
#horizontal bar chart


ax[1,2].hist(vals)


ax[2,0].plot(xtrig, y7)


ax[2,1].plot(xtrig, y8)

ax[2,2].plot(xtrig, y9)
```

```
Out[80]: [<matplotlib.lines.Line2D at 0x1a1b208780>]
```