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.

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

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 [48]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('Energy.csv')
df

# Check if there are any NaN values
df.isnull().sum()
```

Out[48]:
```
X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
X8    0
Y1    0
dtype: int64
```

In [49]:
```
# data features in terms of type
df.dtypes
```

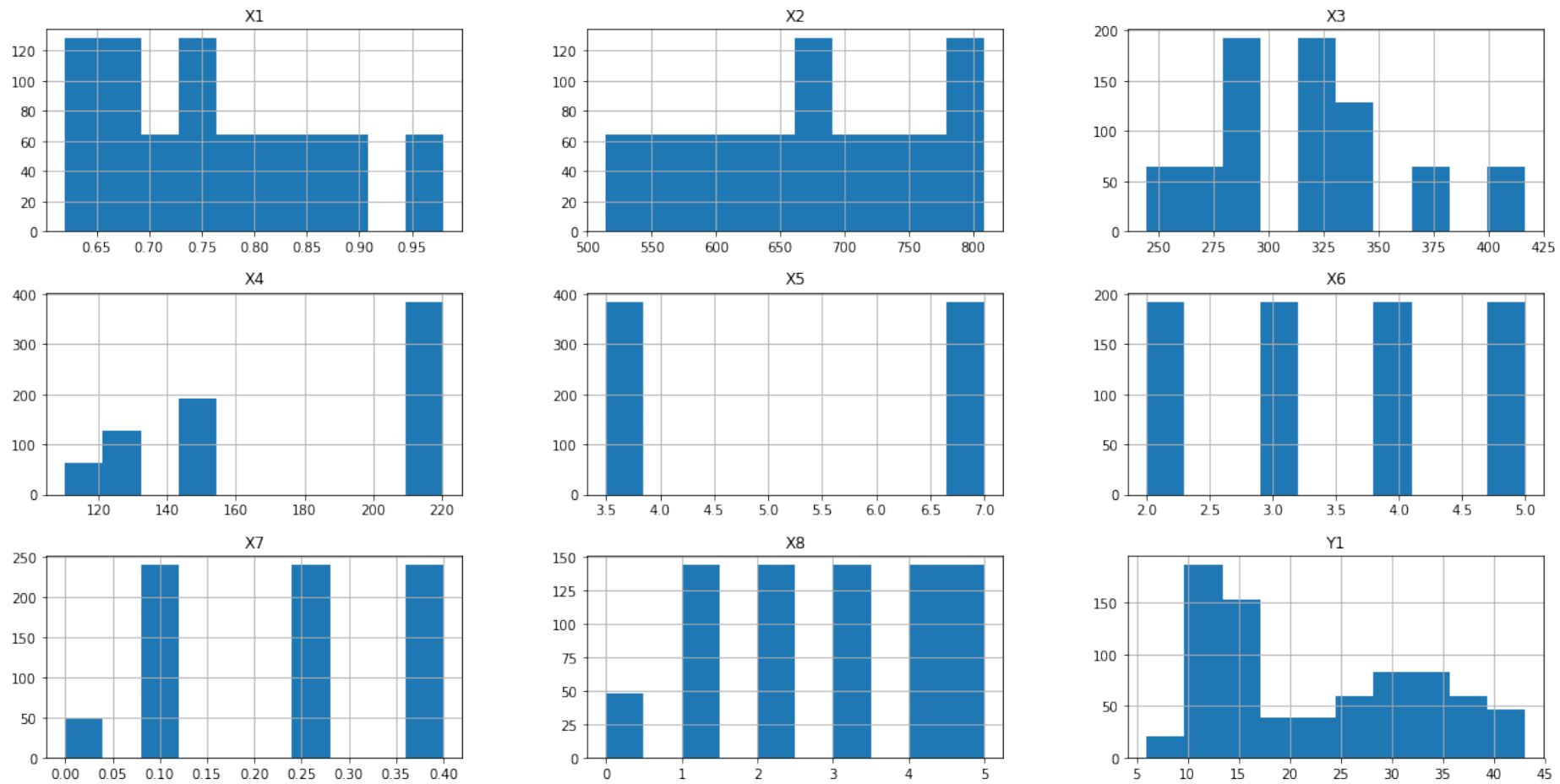
Out[49]:
```
X1    float64
X2    float64
X3    float64
X4    float64
X5    float64
X6      int64
X7    float64
X8      int64
Y1    float64
dtype: object
```

In [50]:
```
df.describe()
```

Out[50]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
count	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000	768.000000	768.00000	768.000000
mean	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000	0.234375	2.81250	22.307201
std	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763	0.133221	1.55096	10.090196
min	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000	0.000000	0.00000	6.010000
25%	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000	0.100000	1.75000	12.992500
50%	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000	0.250000	3.00000	18.950000
75%	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000	0.400000	4.00000	31.667500
max	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000	0.400000	5.00000	43.100000

In [51]:
```
df.hist(figsize=(20,10))
plt.show()
```



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 [52]:
```
# Split data into training and test set using sklearn function

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df[['X1','X2','X3','X4','X5','X6','X7','X8']],  df['
Y1'], 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))
```

```
Number of samples in training data: 614
Number of samples in test data: 154
```

In [53]:
```
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)

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

# intercept
print('intercept:', LinearRegressionModel.intercept_)
```

```
Coefficients: [-6.33926290e+01 -5.86380428e-02  3.46024305e-02 -4.66202367e-02
  4.36194652e+00  1.81224259e-02  1.98760201e+01  2.19167208e-01]
intercept: 79.13116174147392
```

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 [54]:
```
import numpy as np

# root mean square error (Training error)
print("Training error:",np.sqrt(np.mean((Z_reg - Y_reg) ** 2)))
eAll = np.sqrt(np.mean((Z_reg - Y_reg) ** 2))
```

```
Training error: 2.9242420751260143
```

In [55]:
```
# root mean square error (test error)
# FEATURES
X_reg_2 = x_test

# Y
Y_reg_2 = y_test

Z_reg_2=LinearRegressionModel.predict(X_reg_2)

print("test error:",np.sqrt(np.mean((Z_reg_2 - Y_reg_2) ** 2)))
val_e = np.sqrt(np.mean((Z_reg_2 - Y_reg_2) ** 2))
```

```
test error: 2.9054136242997686
```

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 [56]:
```
# Training data 100
X_reg100 = x_train[:100]
Y_reg100 = y_train[:100]

# Train the model using the training sets
LinearRegressionModel_100= linear_model.LinearRegression()
LinearRegressionModel_100.fit(X_reg100, Y_reg100)
Z_reg100=LinearRegressionModel_100.predict(X_reg100)

# Training error
print("Training error:",np.sqrt(np.mean((Z_reg100 - Y_reg100) ** 2)))
e100 = np.sqrt(np.mean((Z_reg100 - Y_reg100) ** 2))

# validation error
Z_reg_101=LinearRegressionModel_100.predict(X_reg_2)
print("Validation error:",np.sqrt(np.mean((Z_reg_101 - Y_reg_2) ** 2)))
v100= np.sqrt(np.mean((Z_reg_101 - Y_reg_2) ** 2))
```

```
Training error: 2.692797626090061
Validation error: 3.029570123008404
```

In [57]:
```
# Training data 200
X_reg200 = x_train[:200]
Y_reg200 = y_train[:200]

# Train the model using the training sets
LinearRegressionModel_200= linear_model.LinearRegression()
LinearRegressionModel_200.fit(X_reg200, Y_reg200)
Z_reg200=LinearRegressionModel_200.predict(X_reg200)

# Training error
print("Training error:",np.sqrt(np.mean((Z_reg200 - Y_reg200) ** 2)))
e200 = np.sqrt(np.mean((Z_reg200 - Y_reg200) ** 2))

# validation error
Z_reg_201=LinearRegressionModel_200.predict(X_reg_2)
print("Validation error:",np.sqrt(np.mean((Z_reg_201 - Y_reg_2) ** 2)))
v200 = np.sqrt(np.mean((Z_reg_201 - Y_reg_2) ** 2))
```

```
Training error: 2.8982041095546793
Validation error: 2.933609137721999
```

In [58]:
```
# Training data 300
X_reg300 = x_train[:300]
Y_reg300 = y_train[:300]

# Train the model using the training sets
LinearRegressionModel_300= linear_model.LinearRegression()
LinearRegressionModel_300.fit(X_reg300, Y_reg300)
Z_reg300=LinearRegressionModel_300.predict(X_reg300)

# Training error
print("Training error:",np.sqrt(np.mean((Z_reg300 - Y_reg300) ** 2)))
e300 = np.sqrt(np.mean((Z_reg300 - Y_reg300) ** 2))

# validation error
Z_reg_301=LinearRegressionModel_300.predict(X_reg_2)
print("Validation error:",np.sqrt(np.mean((Z_reg_301 - Y_reg_2) ** 2)))
v300 = np.sqrt(np.mean((Z_reg_301 - Y_reg_2) ** 2))
```

```
Training error: 2.990252651825265
Validation error: 2.912236524393437
```

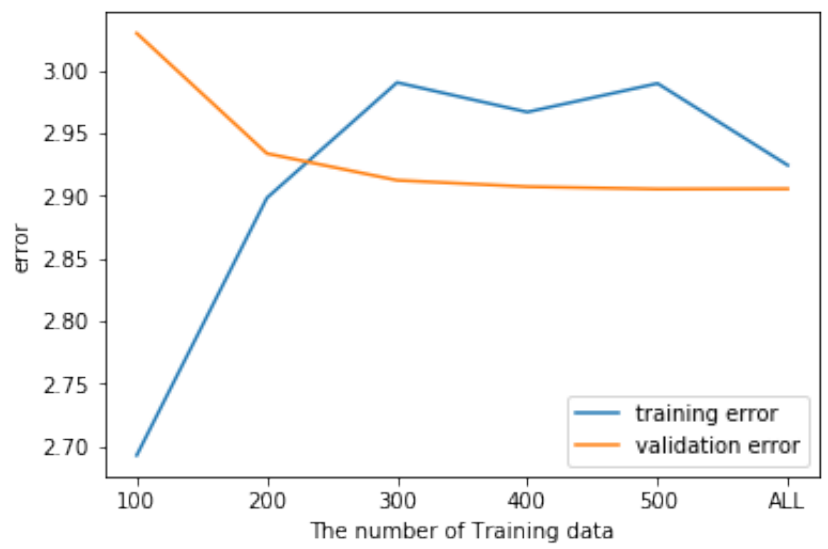
In [59]:
```
# Training data 400
X_reg400 = x_train[:400]
Y_reg400 = y_train[:400]

# Train the model using the training sets
LinearRegressionModel_400= linear_model.LinearRegression()
LinearRegressionModel_400.fit(X_reg400, Y_reg400)
Z_reg400=LinearRegressionModel_400.predict(X_reg400)

# Training error
print("Training error:",np.sqrt(np.mean((Z_reg400 - Y_reg400) ** 2)))
e400 = np.sqrt(np.mean((Z_reg400 - Y_reg400) ** 2))

# validation error
Z_reg_401=LinearRegressionModel_400.predict(X_reg_2)
print("Validation error:",np.sqrt(np.mean((Z_reg_401 - Y_reg_2) ** 2)))
v400 = np.sqrt(np.mean((Z_reg_401 - Y_reg_2) ** 2))
```

```
Training error: 2.96666930498633
Validation error: 2.9071259418855937
```

In [60]:
```
# Training data 500
X_reg500 = x_train[:500]
Y_reg500 = y_train[:500]

# Train the model using the training sets
LinearRegressionModel_500= linear_model.LinearRegression()
LinearRegressionModel_500.fit(X_reg500, Y_reg500)
Z_reg500=LinearRegressionModel_500.predict(X_reg500)

# Training error
print("Training error:",np.sqrt(np.mean((Z_reg500 - Y_reg500) ** 2)))
e500 = np.sqrt(np.mean((Z_reg500 - Y_reg500) ** 2))

# validation error
Z_reg_501=LinearRegressionModel_500.predict(X_reg_2)
print("Validation error:",np.sqrt(np.mean((Z_reg_501 - Y_reg_2) ** 2)))
v500 = np.sqrt(np.mean((Z_reg_501 - Y_reg_2) ** 2))
```

```
Training error: 2.989532662546814
Validation error: 2.9052409119001554
```

In [61]:
```
training_e = [e100,e200,e300,e400,e500,eAll]
validation_e = [v100,v200,v300,v400,v500,val_e]
amount_dat = [100,200,300,400,500,'ALL']
plt.plot(amount_dat,training_e,label='training error')
plt.plot(amount_dat,validation_e,label='validation error')
plt.legend(loc='best')
plt.xlabel("The number of Training data")
plt.ylabel("error")
plt.show()
```



As the number of training data sets is increasing, validation error is decreasing and training error is strictly increasing until 300 training data sets but there is up and down curve on training error from 300 training data sets

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 [62]:
```
Heating_Load = pd.cut(df['Y1'], [0,14,28,100],labels=['Low', 'Medium', 'High'], include_lowest=True)
df['Heating Load'] = Heating_Load

X_c = df.iloc[:,:-2]
Y_c = df['Heating Load']

df.head()
```

Out[62]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Heating Load
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	Medium
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	Medium
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	Medium
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	Medium
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	Medium

In [63]:
```
# Split data into training and test set using sklearn function

from sklearn.model_selection import train_test_split

x_train_c, x_test_c, y_train_c, y_test_c = train_test_split(X_c, Y_c, test_size=0.2, random_state=100)
print ('Number of samples in training data:',len(x_train_c))
print ('Number of samples in test data:',len(x_test_c))
```

```
Number of samples in training data: 614
Number of samples in test data: 154
```

In [64]:
```
from sklearn import linear_model

# Name our logistic regression object
LogisticRegressionModel_c = linear_model.LogisticRegression()

# we create an instance of logistic Regression Classifier and fit the data.
print ('Training a logistic Regression Model..')
LogisticRegressionModel_c.fit(x_train_c, y_train_c)

training_accuracy=LogisticRegressionModel_c.score(x_train_c,y_train_c)
print ('Training Accuracy:',training_accuracy)
test_accuracy=LogisticRegressionModel_c.score(x_test_c,y_test_c)
print ('Test Accuracy:',test_accuracy)
```

```
Training a logistic Regression Model..
Training Accuracy: 0.8078175895765473
Test Accuracy: 0.7727272727272727
```

In [65]:
```
from sklearn.metrics import confusion_matrix
y_true = y_test_c
y_pred = LogisticRegressionModel_c.predict(x_test_c)
ConfusionMatrix=pd.DataFrame(confusion_matrix(y_true, y_pred),columns=['Predicted Low','Predicted Medium
','Predicted High'],index=['Actual Low','Actual Medium','Actual High'])
print ('Confusion matrix of test data is: \n',ConfusionMatrix)
```

```
Confusion matrix of test data is:
                Predicted Low  Predicted Medium  Predicted High
Actual Low                 55                 0               0
Actual Medium               0                42               1
Actual High                25                 9              22
```

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 [66]:
```
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(x_train_c)
x_train_s = pd.DataFrame(X_train_minmax)
x_train_s.columns=['X1','X2','X3','X4','X5','X6','X7','X8']
x_train_s.head()
```

Out[66]:

	X1	X2	X3	X4	X5	X6	X7	X8
0	0.111111	0.833333	0.428571	1.000000	0.0	0.000000	1.00	0.6
1	0.388889	0.500000	1.000000	0.111111	1.0	0.333333	0.25	0.2
2	0.111111	0.833333	0.428571	1.000000	0.0	0.333333	0.25	0.2
3	0.333333	0.583333	0.000000	1.000000	0.0	1.000000	0.25	0.8
4	0.055556	0.916667	0.571429	1.000000	0.0	0.000000	1.00	0.8

In [67]:
```
X_test_minmax = min_max_scaler.fit_transform(x_test_c)
x_test_s = pd.DataFrame(X_test_minmax)
x_test_s.columns=['X1','X2','X3','X4','X5','X6','X7','X8']
x_test_s.head()
```

Out[67]:

	X1	X2	X3	X4	X5	X6	X7	X8
0	0.250000	0.666667	0.142857	1.000000	0.0	0.333333	0.25	0.6
1	0.555556	0.333333	0.428571	0.333333	1.0	0.333333	0.25	1.0
2	0.555556	0.333333	0.428571	0.333333	1.0	1.000000	0.25	0.8
3	0.472222	0.416667	0.571429	0.333333	1.0	0.333333	1.00	1.0
4	0.000000	1.000000	0.714286	1.000000	0.0	1.000000	0.25	0.6

In [68]:
```
from sklearn import linear_model

# Name our logistic regression object
LogisticRegressionModel_s = linear_model.LogisticRegression()

# we create an instance of logistic Regression Classifier and fit the data.
print ('Training a logistic Regression Model..')
LogisticRegressionModel_s.fit(x_train_s, y_train_c)

training_accuracy_s=LogisticRegressionModel_s.score(x_train_s,y_train_c)
print ('Training Accuracy:',training_accuracy_s)

test_accuracy_s=LogisticRegressionModel_s.score(x_test_s,y_test_c)
print ('Test Accuracy:',test_accuracy_s)
```

```
Training a logistic Regression Model..
Training Accuracy: 0.8224755700325733
Test Accuracy: 0.8116883116883117
```

In [69]:
```
from sklearn.metrics import confusion_matrix

y_true_s = y_test_c
y_pred_s = LogisticRegressionModel_s.predict(x_test_s)
ConfusionMatrix_s=pd.DataFrame(confusion_matrix(y_true_s, y_pred_s),columns=['Predicted Low','Predicted
Medium','Predicted High'],index=['Actual Low','Actual Medium','Actual High'])
print ('Confusion matrix of test data is: \n',ConfusionMatrix_s)
```

```
Confusion matrix of test data is:
                Predicted Low  Predicted Medium  Predicted High
Actual Low                 55                 0               0
Actual Medium               0                40               3
Actual High                24                 2              30
```

previous model(Q2.1) Training Accuracy: 0.8078175895765473 Test Accuracy: 0.7727272727272727

current model(Q2.2) Training Accuracy: 0.8224755700325733 Test Accuracy: 0.8116883116883117

current model has higher accuracy of both training and test

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`.

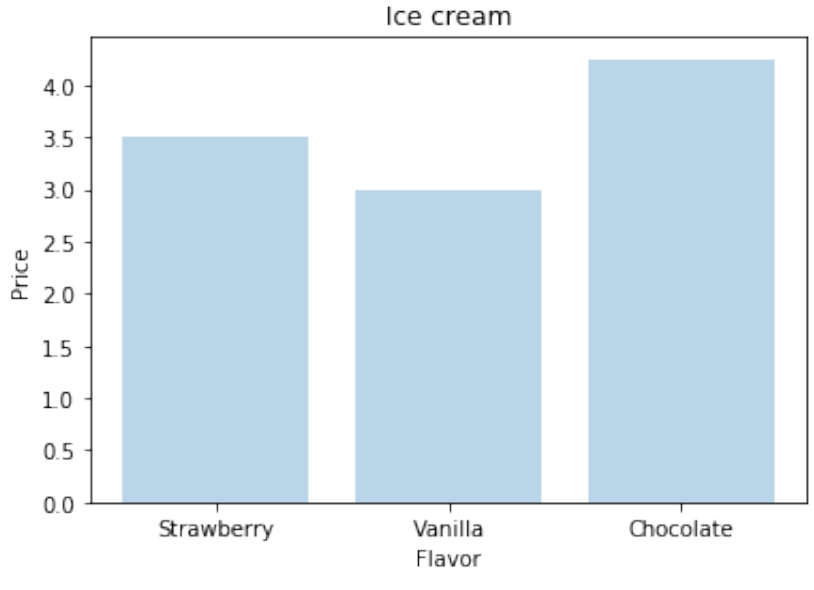
In [70]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

Flavor = np.array(['Strawberry', 'Vanilla', 'Chocolate'])
icecream = pd.DataFrame(Flavor,columns=['Flavor'])
icecream['Price'] = pd.DataFrame([3.50, 3.00, 4.25])

icecream
```

Out[70]:

	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 [71]:
```
plt.bar(icecream['Flavor'], icecream['Price'], align='center', alpha=0.3)
plt.xlabel('Flavor')
plt.ylabel('Price')
plt.title('Ice cream')

plt.show()
```



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 [72]:
```
N = 50
x = np.arange(1,N+1)
y = np.random.randn(N)

f, ax = plt.subplots(nrows=3,ncols=3)
ax[0,0].scatter(x, y,color='green')
ax[0,1].scatter(x, y,color='purple',marker='x')
ax[0,2].scatter(x, y,color ='blue',marker='^')

ax[1,0].plot(x, y)
ax[1,1].barh(x,np.abs(y))
ax[1,2].hist(y)

ax[2,0].plot(np.sin(x))
ax[2,1].plot(np.cos(x))
ax[2,2].plot(np.tan(x))

plt.show()
```