In [ ]:
```
import pandas as pd
import numpy as np

from matplotlib import pyplot as pplt
import seaborn as sbn
%matplotlib inline

import sklearn

# Setup linear regression for use in this exercise
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Built-in dataset are in one of the libraries i.e. sklearn
from sklearn.datasets import load_boston
```

In [ ]:
```
# Load an built-in dataset from sklearn
bostonDs = load_boston()
```

In [ ]:
```
#data = dataset, target = dependent variable, feature_names = column headers, DESCR = data dictionary
#bostonDs.keys()
```

In [ ]:
```
#bostonDs.data.shape
```

In [ ]:
```
print (bostonDs.DESCR)
```

In [ ]:
```
# load the boston data into a dataframe
bosDf = pd.DataFrame (bostonDs.data)
#bosDf.head ()
```

In [ ]:
```
bosDf.columns = bostonDs.feature_names
bosDf.head ()
```

In [ ]:
```
#assigning dependent variable to column named "Price"
bosDf ['PRICE'] = bostonDs.target
```

In [ ]:
```
bosDf.head ()
```

In [ ]:
```
# Statistical information on numeric columns
bosDf.describe ()
```

In [ ]:
```
# Remove the price column from boston data and assign the new dataset to X
X = bosDf.drop ('PRICE', axis = 1)

# Assign linear regression function to a variable
linReg = LinearRegression ()
linReg
```

In [ ]:
```
# Now fit your data in a linear regression model
linReg.fit (X, bosDf.PRICE)
```

In [ ]:
```
# Now plot price vs no. of rooms
pplt.scatter (bosDf.RM, bosDf.PRICE)
pplt.xlabel ("Average number of rooms per dwelling (RM)")
pplt.ylabel ("Housing Price")
pplt.title ("Relationship between RM and Price")
pplt.show ()
```

In [ ]:
```
# Now plot the actual price vs. predicted price
pplt.scatter (bosDf.PRICE, linReg.predict (X))
pplt.xlabel ("Price")
pplt.ylabel ("Predicted Price")
pplt.title ("Price vs Predicted Price")
pplt.show ()
```

In [ ]:
```
# Now compare the plot of actual price vs no. of rooms against the plot of predicted price vs no. of rooms
pplt.figure (figsize = (9, 2))

pplt.subplot (121)
pplt.scatter (bosDf.RM, bosDf.PRICE)
pplt.grid ()
pplt.xlabel ("Average number of rooms per dwelling (RM)")
pplt.ylabel ("Housing Price")
pplt.title ("Relationship between RM and Price")

pplt.subplot (122)
pplt.scatter (bosDf.RM, linReg.predict (X))
pplt.grid ()
pplt.xlabel ("Average number of rooms per dwelling (RM)")
pplt.ylabel ("Housing Price")
pplt.title ("Relationship between RM and Predicted Price")
pplt.show ()
```

In [ ]:
```
# Accuracy score - coefficient of determination R^2 of the prediction. The best possible score is 1.0 .
linReg.score (X, bosDf.PRICE)
```

In [ ]:
```
#sbn.lmplot (x = 'RM', y = 'PRICE', data = bosDf, fit_reg = True)
```

Splitting data between training and test sets.

Randomize the two sets while running linear regression in your analysis.

In [ ]:
```
# Split the data into training and test sets
#test_size default = 0.25
XTrain, XTest, YTrain, YTest = sklearn.model_selection.train_test_split (X, bosDf.PRICE,
                                              test_size = 0.33,
                                              random_state = 5)
print (XTrain.shape)
print (XTest.shape)
print (YTrain.shape)
print (YTest.shape)
```

In [ ]:
```
# Now perform the linear regression on the training set i.e. fit your model to the data
linReg.fit (XTrain, YTrain)
```

In [ ]:
```
# Now using the same model predict the Price for both sets
predTrain = linReg.predict (XTrain)
predTest = linReg.predict (XTest)
```

In [ ]:
```
# Compute the Mean Squared Error (MSE) - an indication of the accuracy of the model's prediction
print ('Fit a model XTrain, and calculate MSE with YTrain:',
       np.mean ((YTrain - linReg.predict (XTrain)) ** 2))
print ('Fit a model XTrain, and calculate MSE with XTest, YTest:',
       np.mean ((YTest - linReg.predict (XTest)) ** 2))
```

In [ ]:
```
# Accuracy score for the training set
linReg.score (XTrain, YTrain)
```

In [ ]:
```
# Accuracy score for the training set with predicted values
linReg.score (XTrain, predTrain)
```

In [ ]:
```
# Accuracy score for the test set
linReg.score (XTest, YTest)
```

In [ ]:
```
# Accuracy score for the test set with predicted values
linReg.score (XTest, predTest)
```

In [ ]:
```
# Print coefficients, model's accurance, and variance

print ('Coefficients: \n', linReg.coef_)

# The mean squared error
print ("Mean squared error: %.2f"
       % mean_squared_error (YTest, predTest))

# Explained variance score: 1 is perfect prediction
print ('Variance score: %.2f' % r2_score (YTest, predTest))
```

In [ ]:
```
heatmap_df = bosDf.corr ()

pplt.subplots (figsize = (15, 7))
#plt.figure (figsize = (15, 5))
sbn.heatmap (heatmap_df,
             vmin = -1,
             annot = True,
             linewidths = 1)
```

Now let us do a logistic regression exercise.

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
irisDs = load_iris ()
print (irisDs.DESCR)
```

In [ ]:
```
irisDf = pd.DataFrame (irisDs.data)
irisDf.head ()
```

In [ ]:
```
irisDf.columns = irisDs.feature_names
irisDf.head ()
```

In [ ]:
```
irisDf ['class'] = irisDs.target
```

In [ ]:
```
irisDf.head ()
```

In [ ]:
```
irisDf ['class'].value_counts ()
```

In [ ]:
```
# Now plot a couple of variables
pplt.scatter (irisDf ['sepal length (cm)'],
              irisDf ['sepal width (cm)'],
              c = irisDf ['class'])
pplt.show ()
```

In [ ]:
```
X = irisDf.drop (['class'], axis = 1)
Y = irisDs.target

# Assign logistic regression function to a variable
logReg = LogisticRegression ()
```

In [ ]:
```
#80% for training data, 20% for test data
from sklearn.model_selection import train_test_split
XTrain, XTest, YTrain, YTest = train_test_split (X, Y,
                                                  test_size = 0.2,
                                                  random_state = 15)
```

In [ ]:
```
logReg.fit (XTrain, YTrain)
```

In [ ]:
```
#accuracy score of model using training data
logReg.score (XTrain, YTrain)
```

In [ ]:
```
#generate prediction values
YPred = logReg.predict (XTest)
```

In [ ]:
```
#Confusion matrix shows which values model predicted correctly vs incorrectly
sklearn.metrics.confusion_matrix (YTest, YPred)
```

In [ ]:
```
#accuracy score of model on test data
logReg.score (XTest, YTest)
```

In [ ]:
```
#from precision column, model is better at predicting passengers that do not survive
#print (sklearn.metrics.classification_report (YTest, YPred))
```

In [ ]:
```
#Confusion matrix shows which values model predicted correctly vs incorrectly
cm = pd.DataFrame (sklearn.metrics.confusion_matrix (YTest, YPred),
                   columns = ['Predicted Class 0', 'Predicted Class 1', 'Predicted Class 2'],
                   index = ['True Class 0', 'True Class 1', 'True Class 2'])
cm
```

In [ ]:
```
YTest
```

In [ ]:
```
YPred
```

In [ ]:
```

```