```
#Line Regression
#--------------------------
#Import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#Loading the Data
data = pd.read_csv('ex1data.txt', names = ['population', 'profit'])
df = pd.DataFrame(data)
df.head(10)
#Plotting the Data
## Split population and profit into X and y
X_df = pd.DataFrame(data.population)
y_df = pd.DataFrame(data.profit)
## Length, or number of observations, in our data
m = len(y_df)
plt.figure(figsize=(10,8))
plt.plot(X_df, y_df, 'o')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
#generating fit lines
plt.figure(figsize=(10,8))
plt.plot(X_df, y_df, 'k.')
plt.plot([5, 22], [6,6], '-')
plt.plot([5, 22], [0,20], '-')
plt.plot([5, 15], [-5,25], '-')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')

# Gradient descent
#-------------------------------------
x_quad = [n/10 for n in range(0, 100)]
y_quad = [(n-4)**2+5 for n in x_quad]
plt.figure(figsize = (10,7))
plt.plot(x_quad, y_quad, 'k--')
plt.axis([0,10,0,30])
plt.plot([1, 2, 3], [14, 9, 6], 'go')
plt.plot([5, 7, 8],[6, 14, 21], 'bo')
plt.plot(4, 5, 'ro')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Quadratic Equation')
# Fixing no of iterations and Alpha the learning rate
iterations = 1500
alpha = 0.01
## Add a columns of 1s as intercept to X
X_df['intercept'] = 1
## Transform to Numpy arrays for easier matrix math and start theta at 0
X = np.array(X_df)
y = np.array(y_df).flatten()
theta = np.array([0, 0])
def cost_function(X, y, theta):
  """
  cost_function(X, y, theta) computes the cost of using theta as the parameter for linear
```

```
regression to fit the data points in X and y
  """
  ## number of training examples
  m = len(y)
  ## Calculate the cost with the given parameters
  J = np.sum((X.dot(theta)-y)**2)/2/m
  return (J)

cost_function(X, y, theta)
#gradient descent algorithm
def gradient_descent(X, y, theta, alpha, iterations):
  """
  gradient_descent Performs gradient descent to learn theta
  theta = GRADIENTDESENT(X, y, theta, alpha, num_iters) updates theta by
  taking num_iters gradient steps with learning rate alpha
  """
  cost_history = [0] * iterations
  for iteration in range(iterations):
    hypothesis = X.dot(theta)
    loss = hypothesis-y
    gradient = X.T.dot(loss)/m
    theta = theta - alpha*gradient
    cost = cost_function(X, y, theta)
    cost_history[iteration] = cost
  return theta, cost_history
(t, c) = gradient_descent(X,y,theta,alpha, iterations)
## Print theta parameters
print (t)
#Prediction
print(np.array([3.5, 1]).dot(t))
print(np.array([7, 1]).dot(t))
## Plotting the best fit line
best_fit_x = np.linspace(0, 25, 20)
best_fit_y = [t[1] + t[0]*xx for xx in best_fit_x]
plt.figure(figsize=(10,6))
plt.plot(X_df.population, y_df, '.')
plt.plot(best_fit_x, best_fit_y, '-')
plt.axis([0,25,-5,25])
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.title('Profit vs. Population with Linear Regression Line')
```