Linear Regression

January 31, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
df = pd.read_csv("cars24-car-price-clean.csv")
```

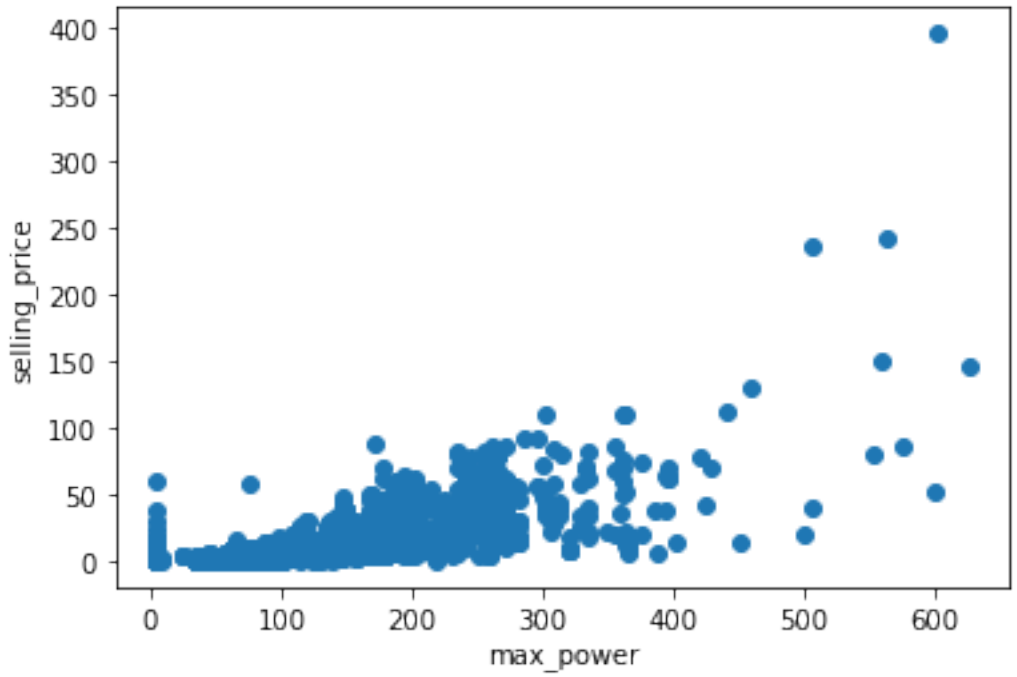
[3]:
```
df.head()
```

[3]:

	selling_price	year	km_driven	mileage	engine	max_power	make	model	transmission_type	seats_coupe	seats_family	seats_large	fuel_cng	fuel_diesel	fuel_electric	fuel_lpg	fuel_petrol	seller_dealer	seller_individual	seller_trustmark dealer
0	1.20	2012.0	120000	19.70	796.0	46.30	Maruti	Alto Std	1	0	1	0	0	0	0	0	1	0	1	0
1	5.50	2016.0	20000	18.90	1197.0	82.00	Hyundai	Grand i10 Asta	1	0	1	0	0	0	0	0	1	0	1	0
2	2.15	2010.0	60000	17.00	1197.0	80.00	Hyundai	i20 Asta	1	0	1	0	0	0	0	0	1	0	1	0
3	2.26	2012.0	37000	20.92	998.0	67.10	Maruti	Alto K10 2010-2014 VXI	1	0	1	0	0	0	0	0	1	0	1	0
4	5.70	2015.0	30000	22.77	1498.0	98.59	Ford	Ecosport 2015-2021 1.5 TDCi Titanium BSIV	1	0	1	0	0	1	0	0	0	1	0	0

[4]:
```
# define X and y
X = df["max_power"].values
```

```
Y = df["selling_price"].values

# normalistion - u = X.mean(); std = X.std(); X = (X-u)/std

# visualise
plt.scatter(X,Y)
plt.xlabel("max_power")
plt.ylabel("selling_price")
plt.show()
```

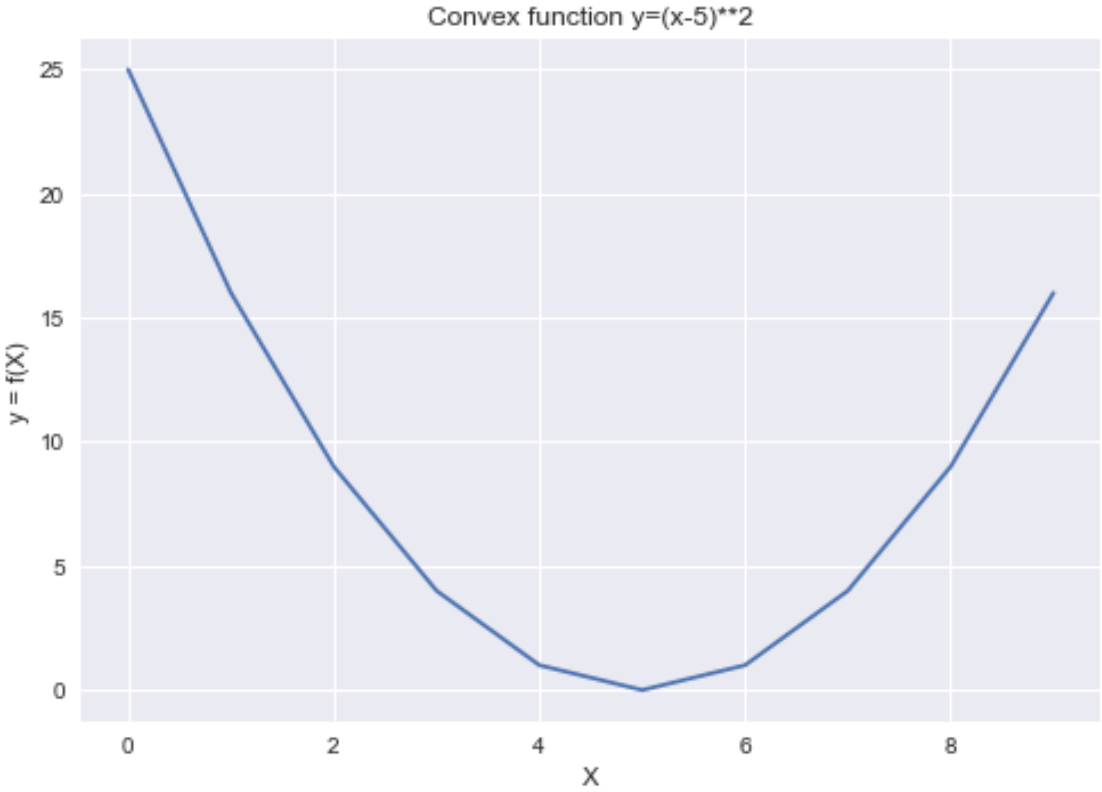


[5]:
```
def univariate_linear_hypothesis(x, theta):
    y_hat = theta[0] + theta[1]*x
    return y_hat
```

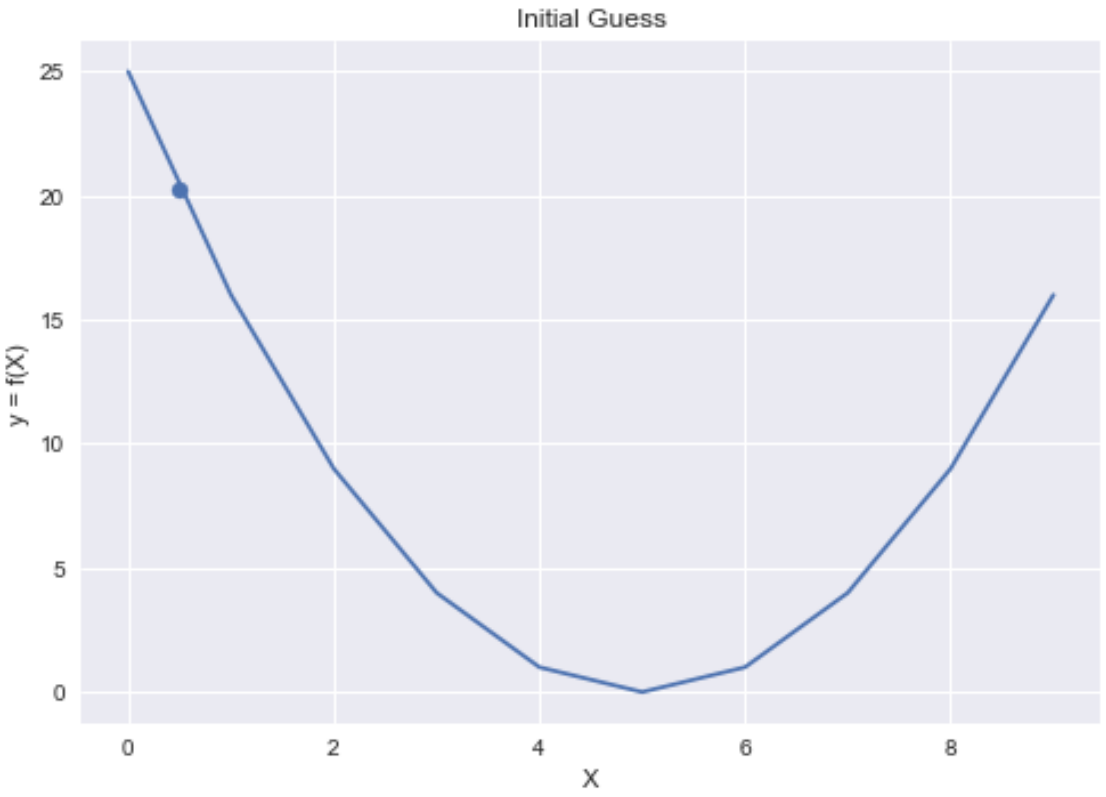
[6]:
```
def cost(X,Y,theta):
    m = X.shape[0] # num of training examples
    total_error = 0.0
    for i in range(m):
        y_ = univariate_linear_hypothesis(X[i],theta)
        total_error += (y_ - Y[i])**2
    return (total_error/m)
```

[7]:
```
X = np.arange(10)
Y = (X-5)**2 ## x = 5 cost function will give minimum(0)
```

```
plt.style.use("seaborn")
plt.plot(X,Y)
plt.ylabel("y = f(X)")
plt.xlabel("X")
plt.title("Convex function y=(x-5)**2") # assume ground truth value is 5
plt.show()
```

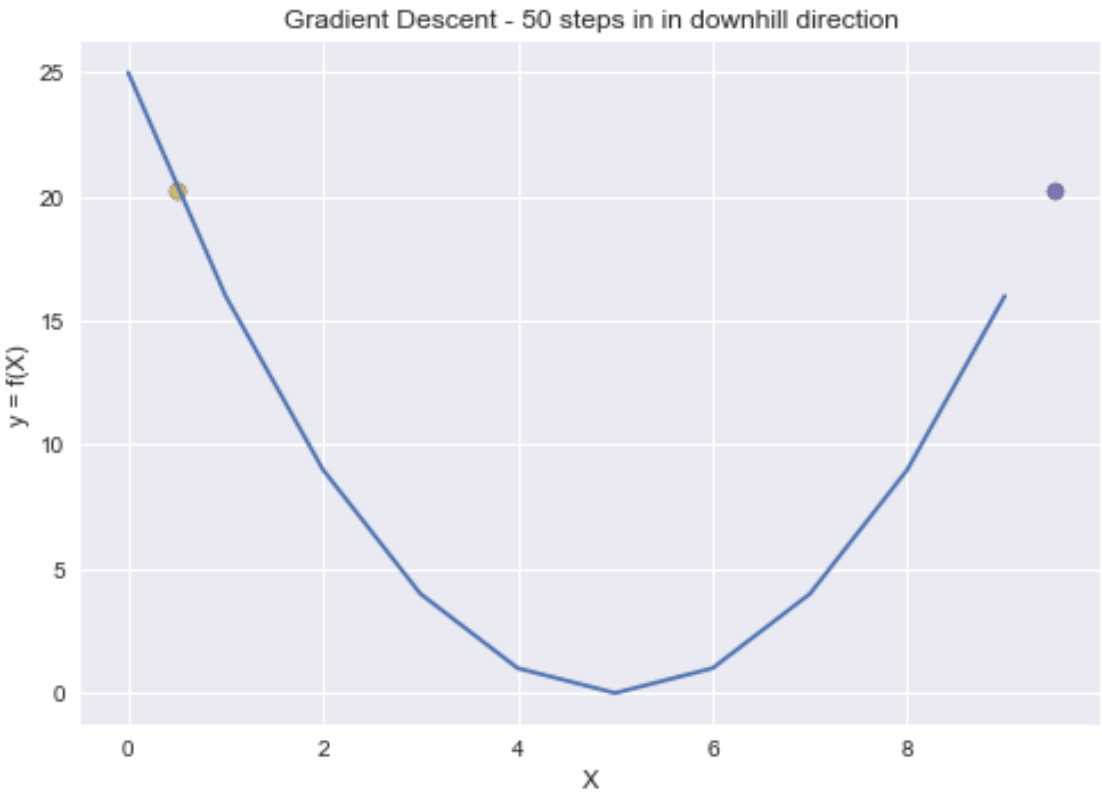


[8]:
```
x = 0.5
y = (x-5)**2 # assume ground truth value is 5
plt.plot(X,Y)
plt.scatter(x,y)
plt.ylabel("y = f(X)")
plt.xlabel("X")
plt.title("Initial Guess")
plt.show()
```



[9]:
```
import time


fig = plt.figure()
ax = fig.add_subplot(111)
plt.ion()

x = 0.5
y = (x-5)**2 # assume ground truth value is 5
plt.plot(X,Y)
plt.scatter(x,y)
plt.ylabel("y = f(X)")
plt.xlabel("X")
plt.title("Gradient Descent - 50 steps in in downhill direction")

lr = 0.1
errors = []
# 10 steps in the downhill direction
for i in range(10):
    grad = 2*(x-5)
    x = x - grad
```

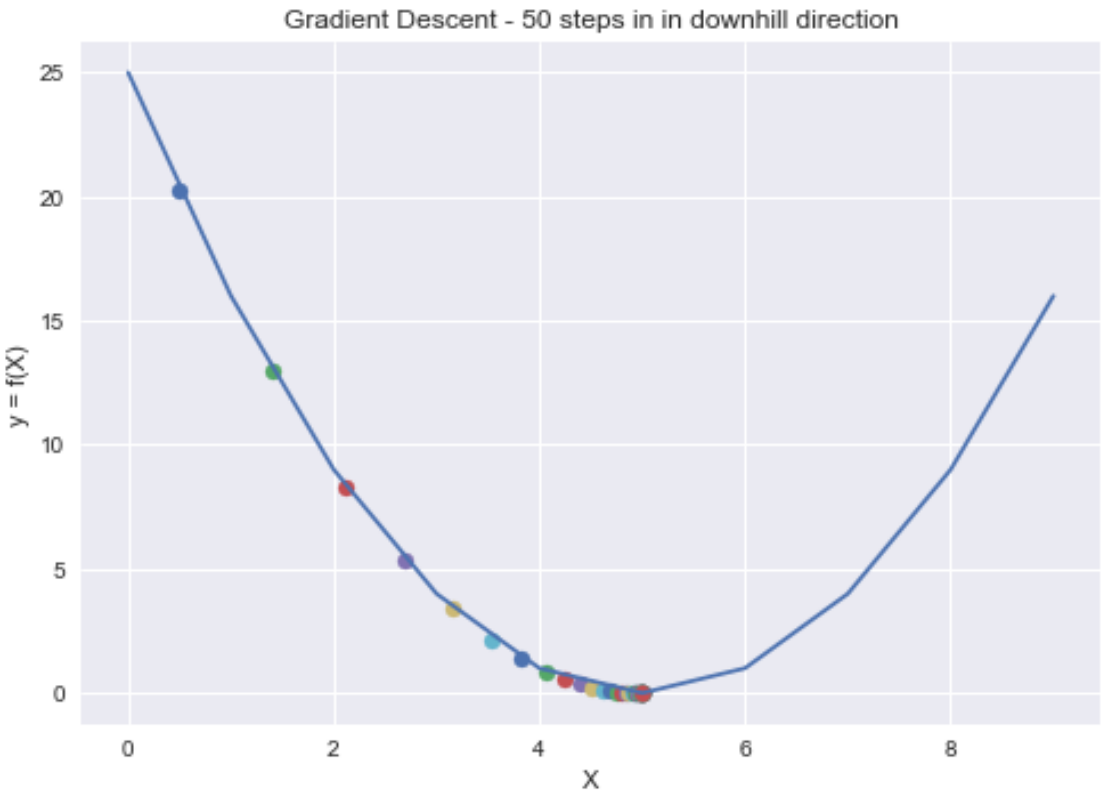
```
    y = (x-5)**2
    error = y - 0
    errors.append(error)
    plt.scatter(x,y)
    fig.canvas.draw()
    time.sleep(0.5)
plt.show()
```



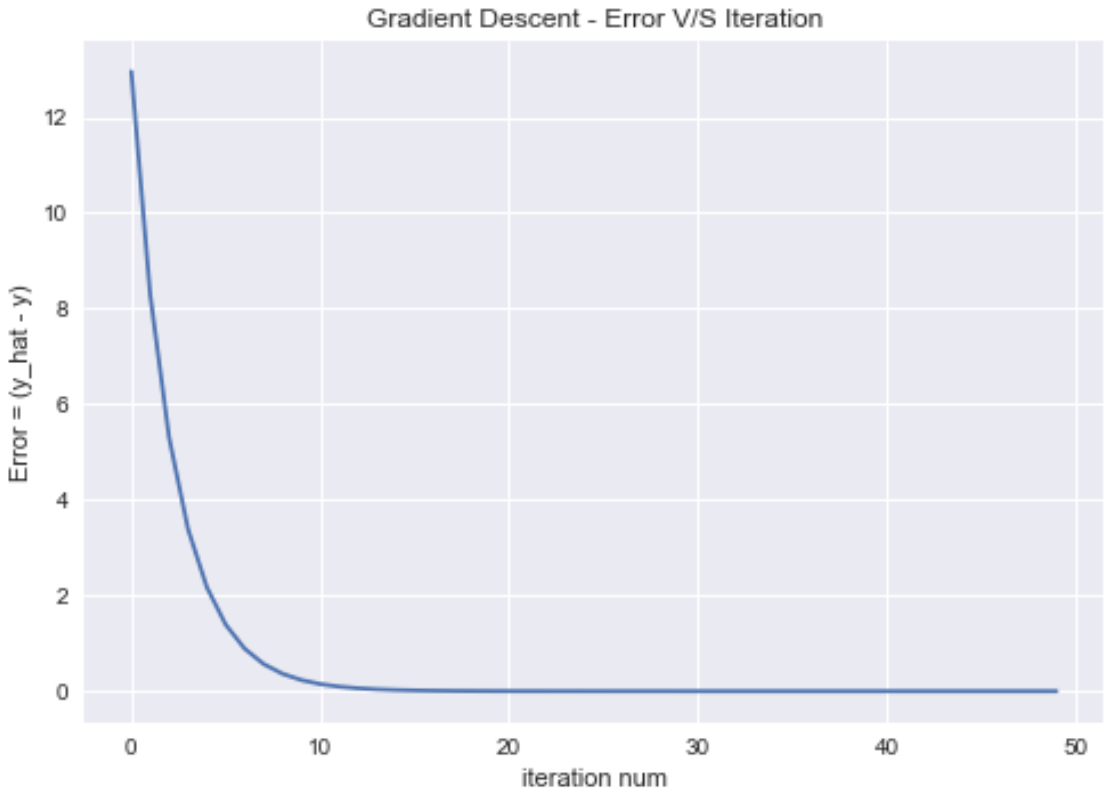
[10]:
```
import time


fig = plt.figure()
ax = fig.add_subplot(111)
plt.ion()
fig.show()
fig.canvas.draw()

x = 0.5
y = (x-5)**2 # assume ground truth value is 5
plt.plot(X,Y)
plt.scatter(x,y)
```

```
plt.ylabel("y = f(X)")
plt.xlabel("X")
plt.title("Gradient Descent - 50 steps in in downhill direction")

lr = 0.1
errors = []
# 50 steps in the downhill direction
for i in range(50):
    grad = 2*(x-5)
    x = x - lr*grad
    y = (x-5)**2
    error = y - 0
    errors.append(error)
    plt.scatter(x,y)
    fig.canvas.draw()
    time.sleep(0.5)
plt.show()
```

```
<ipython-input-10-1fbf92883d4d>:7: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```



[11]:
```
plt.plot(errors)
plt.xlabel("iteration num")
plt.ylabel("Error = (y_hat - y)")
plt.title("Gradient Descent - Error V/S Iteration")
plt.show()
```



[16]:
```
# define X and y
X = df["max_power"].values
Y = df["selling_price"].values

u = X.mean()
std = X.std()
X = (X-u)/std
```

[13]:
```
def hypothesis(x,theta):
    y_hat = theta[0] + theta[1]*x
    return y_hat
```

[12]:
```
def gradient(X,Y,theta):
    m = X.shape[0]
    grad = np.zeros((2,))
```

```
    for i in range(m):
        x = X[i]
        y_hat = hypothesis(x,theta)
        y = Y[i]
        grad[0] += (y_hat - y)
        grad[1] += (y_hat - y)*x
    return grad/m
```

[14]:
```
def error(X,Y,theta):
    m = X.shape[0]
    total_error = 0.0
    for i in range(m):
        y_hat = hypothesis(X[i],theta)
        total_error += (y_hat - Y[i])**2
    return (total_error/m)
```

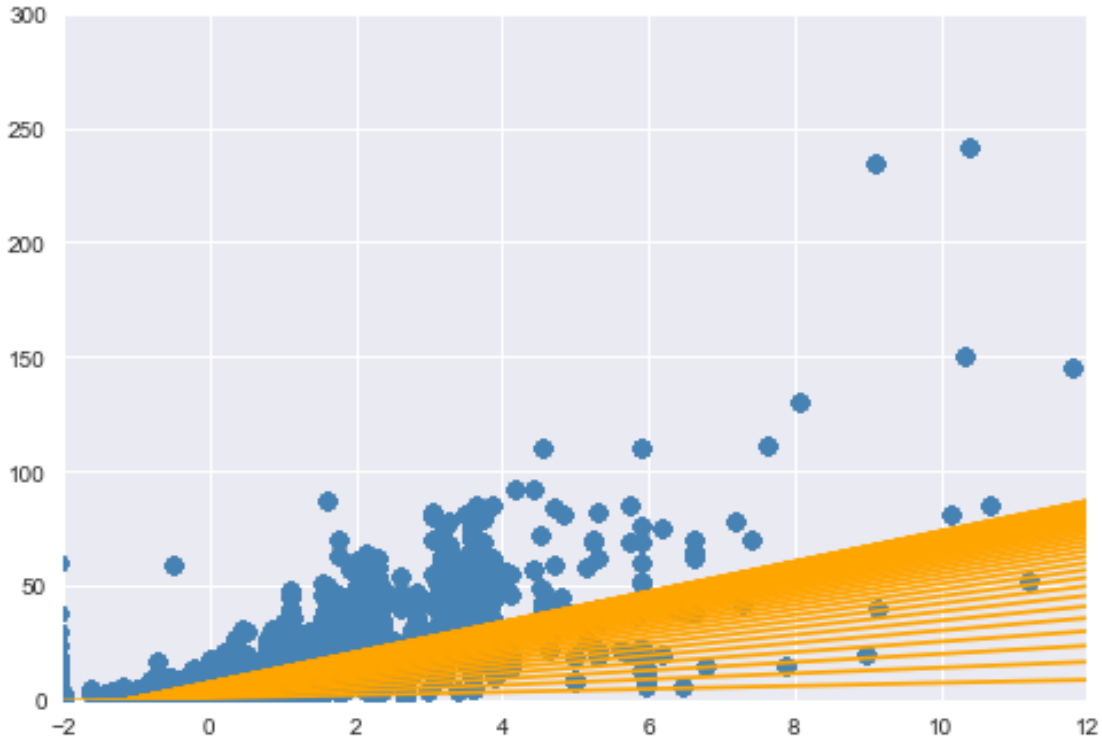
[15]:
```
def gradient_descent(X,Y, max_steps=100,learning_rate =0.1):
    theta = np.zeros((2,))
    error_list = []
    theta_list = []

    for i in range(max_steps):
        # Compute grad
        grad = gradient(X,Y,theta)
        e = error(X,Y,theta)

        #Update theta
        theta[0] = theta[0] - learning_rate*grad[0]
        theta[1] = theta[1] - learning_rate*grad[1]
        # Storing the theta values during updates
        theta_list.append((theta[0],theta[1]))
        error_list.append(e)

    return theta,error_list,theta_list
```

[17]:
```
theta, error_list, theta_list = gradient_descent(X,Y,max_steps=50)
print(theta)
```

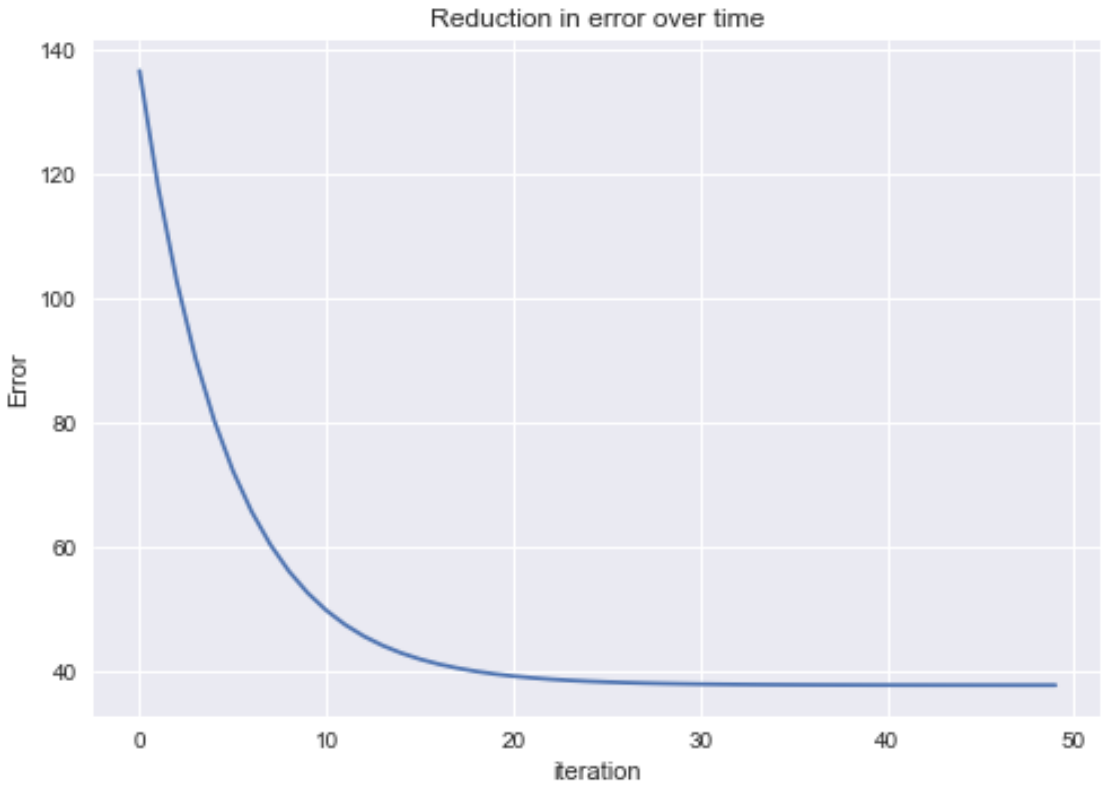
```
[7.35034462 6.62351384]
```

[18]:
```
import time
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
plt.ion()

for intercept, slope in theta_list:
```

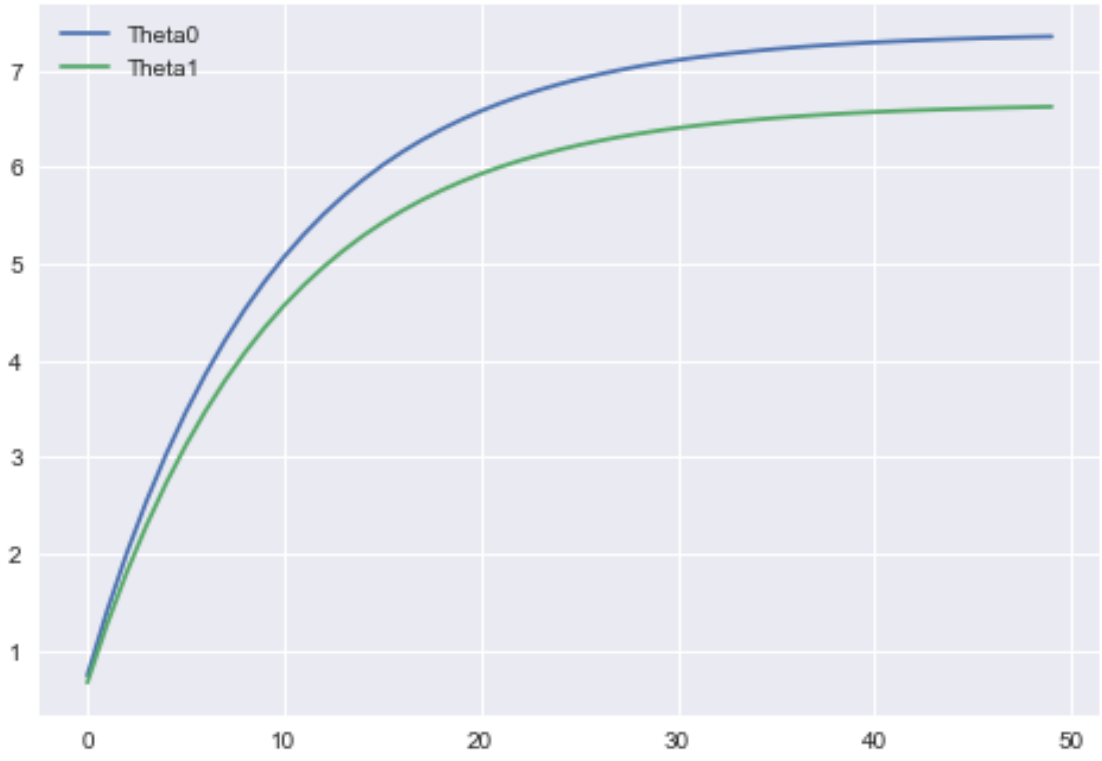
```
    # Plot a line from slope and intercept
    #ax.clear()
    ax.set_xlim([-2, 12])
    ax.set_ylim([0, 300])
    x_vals = np.array(ax.get_xlim())
    y_vals = intercept + slope * x_vals
    ax.scatter(X, Y, color="steelblue")
    ax.plot(x_vals, y_vals , color='orange')
    fig.canvas.draw()
    time.sleep(0.5)
plt.show()
```



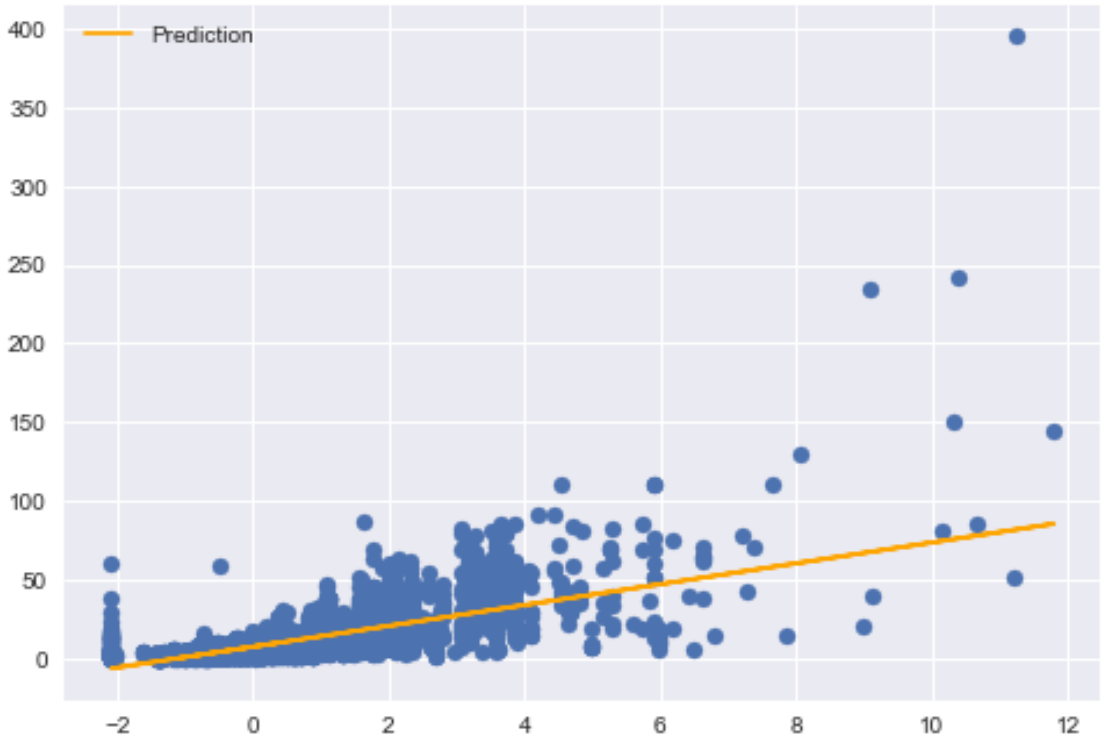
[19]:
```
fig = plt.figure()
plt.plot(error_list)
plt.title("Reduction in error over time")
plt.xlabel("iteration")
plt.ylabel("Error")
plt.show()
```



[20]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
theta_list = np.array(theta_list)
plt.plot(theta_list[:,0],label="Theta0")
plt.plot(theta_list[:,1],label="Theta1")
plt.legend()
plt.show()
```



[21]:
```
fig = plt.figure()
Y_hat = hypothesis(X,theta)
plt.scatter(X,Y)
plt.plot(X,Y_hat,color='orange',label="Prediction")
plt.legend()
plt.show()
```



[ ]:
```
n_reps = 1000
bs_thetas = []
for i in range(n_reps):
    inds = np.arange(len(X)) #setup an array of indices
    bs_inds = np.random.choice(inds, size=len(inds)) # bootstrap the indices
    X_bs, Y_bs = X[bs_inds], Y[bs_inds] # generate the bootstrapped sample
    bs_theta, _, _ = gradient_descent(X_bs, Y_bs, max_steps=50) # perform␣
 ↪gradient descent
    bs_thetas.append(bs_theta)
```

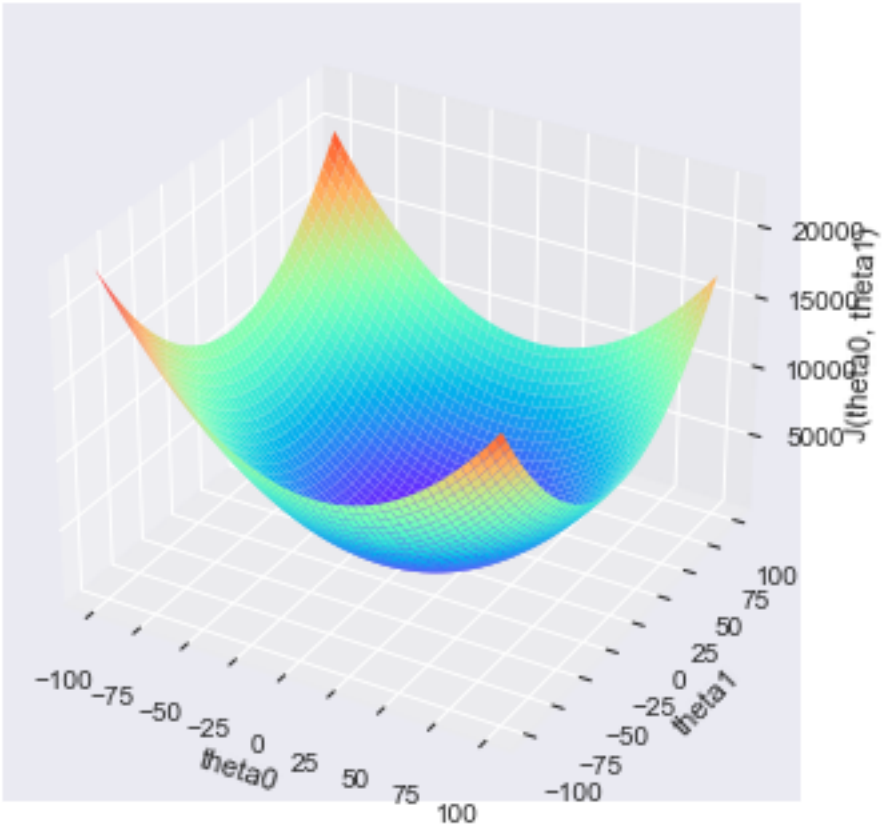
[ ]:
```
plt.figure()
Y_hat = hypothesis(X,theta)
plt.scatter(X,Y)
for i in range(n_reps):
    plt.plot(X, hypothesis(X, bs_thetas[i]), alpha=0.1, color='yellow')
plt.plot(X,Y_hat,color='red')
plt.show()
```

[22]:
```
def r2_score(Y,Y_hat):
    num = np.sum((Y-Y_hat)**2)
    denom = np.sum((Y- Y.mean())**2)
    score = (1 - num/denom)
    return score
```

[23]:
```
r2_score(Y,Y_hat)
```

[23]:
```
0.5402225660763946
```

[24]:
```
# Loss Actually
T0 = np.arange(-100,100,1)
T1 = np.arange(-100,100,1)

T0,T1 = np.meshgrid(T0,T1)
J = np.zeros(T0.shape)
for i in range(J.shape[0]):
    for j in range(J.shape[1]):
        Y_hat = T1[i,j]*X + T0[i,j]
        J[i,j] = np.sum((Y-Y_hat)**2)/Y.shape[0]

fig = plt.figure()
axes = fig.gca(projection='3d')
axes.plot_surface(T0,T1,J, cmap="rainbow")
axes.set_xlabel("theta0")
axes.set_ylabel("theta1")
axes.set_zlabel("J(theta0, theta1)")
plt.show()
```



[25]:
```
from sklearn.linear_model import LinearRegression
```

[30]:
```
print(X.shape)
print(Y.shape)
```

```
(19820, 1)
(19820, 1)
```

[31]:
```
model = LinearRegression()
```

[28]:
```
X = X.reshape(X.size, 1)
Y = Y.reshape(Y.size, 1)
```

[32]:
```
model.fit(X,Y) # traiing
```

[32]:
```
LinearRegression()
```

[33]:
```
model.intercept_
```

[33]:
```
array([7.38842289])
```

[34]:
```
model.coef_
```

[34]:
```
array([[6.65782679]])
```

[35]:
```
model.score(X,Y)
```

[35]:
```
0.5402545880839582
```

[37]:
```
output= model.predict(X)
```

[38]:
```
fig = plt.figure()
plt.scatter(X,Y,label='data')
plt.scatter(X,output,color='orange',label='prediction')
plt.legend()
plt.show()
```


data
prediction
400
350
300
250
200
150
100
50
0
−2
0
2
4
6
8
10
12

[ ]: