

```
In [1]: def prepare_country_stats(oecd_bli, gdp_per_capita):
        oecd_bli = oecd_bli[oecd_bli["INEQUALITY"] != "TOT"]
        oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
        gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
        gdp_per_capita.set_index("Country", inplace=True)
        full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                      left_index = True, right_index = True)
        full_country_stats.sort_values(by="GDP per capita", inplace= True)
        remove_indices = [0,1,6,8,33,34,35]
        keep_indices = list(set(range(36)) - set(remove_indices))
        return full_country_stats[["GDP per capita","Life satisfaction"]].iloc[keep_indices]
```

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: import numpy as np
import pandas as pd
import sklearn.linear_model
```

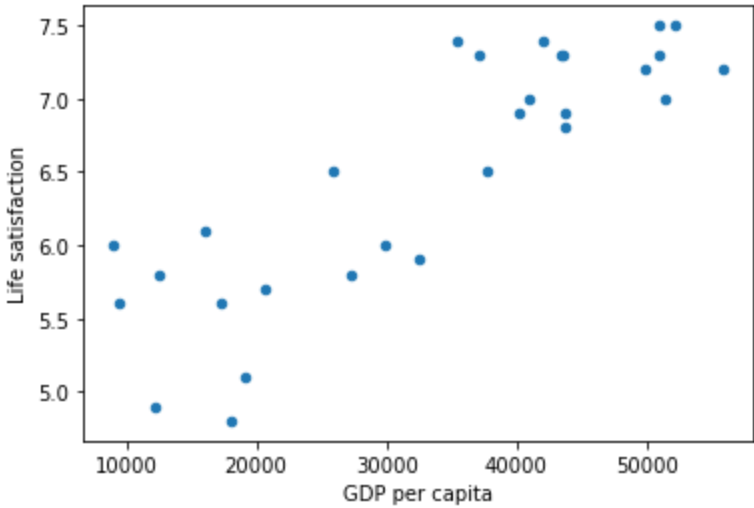
```
In [4]: # Load the data
oecd_bli = pd.read_csv("handson-ml\datasets\lifesat\oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv("handson-ml\datasets\lifesat\gdp_per_capita.csv", thousands=",", delimiter='\t',
                             encoding='latin1', na_values="n/a")
```

```
In [5]: gdp_per_capita.head(5)
```

	Country	Subject Descriptor	Units	Scale	Country/Series-specific Notes	2015	Estimates Start After
0	Afghanistan	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	599.994	2013.0
1	Albania	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	3995.383	2010.0
2	Algeria	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	4318.135	2014.0
3	Angola	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	4100.315	2014.0
4	Antigua and Barbuda	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	14414.302	2011.0

```
In [6]: # Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
x = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]
```

```
In [7]: # Visualize the data
country_stats.plot(kind="scatter", x="GDP per capita", y="Life satisfaction")
plt.show()
```



```
In [8]: # Select a linear model
model = sklearn.linear_model.LinearRegression()
```

```
In [9]: # Train model
model.fit(x,y)
```

```
Out[9]: LinearRegression()
```

```
In [10]: # Make a prediction for Cyprus
x_new = [[22587]] # Cyprus' GDP per capita
print(model.predict(x_new))
```

```
[[5.96242338]]
```

```
In [ ]:
```

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}(\theta_0, \theta_1, \dots, \theta_n)$$

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [2]:

```
def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"] != "TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                  left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    remove_indices = [0,1,6,8,33,34,35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP per capita","Life satisfaction"]].iloc[keep_indices]
```

In [3]:

```
# Load the data
oecd_bli = pd.read_csv("handson-ml\datasets\lifesat\oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv("handson-ml\datasets\lifesat\gdp_per_capita.csv", thousands=",", delimiter='\t',
                             encoding='latin1', na_values="n/a")
```

In [4]:

```
# Brief description of the dataset
gdp_per_capita.head(5)
```

Out[4]:

	Country	Subject Descriptor	Units	Scale	Country/Series-specific Notes	2015	Estimates Start After
0	Afghanistan	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	599.994	2013.0
1	Albania	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	3995.383	2010.0
2	Algeria	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	4318.135	2014.0
3	Angola	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	4100.315	2014.0
4	Antigua and Barbuda	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	14414.302	2011.0

In [5]:

```
# Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
```

In [84]:

```
# Build the pices for gradient descent
normalized_country_stats=(country_stats-country_stats.mean())/country_stats.std()
ones = np.ones((country_stats[country_stats.columns[0]].count(),1)) # Ones vector with size equal to data set rows
X = normalized_country_stats["GDP per capita"].to_frame()
# Append an extra column of ones to the fearute vector (X)
X.insert(loc=0, column='X0', value=ones)
Y = normalized_country_stats["Life satisfaction"]
alpha = 0.001 # Learning rate (gradient descent step)
m,n = X.shape
theta = np.ones(n) # Inital column vector of theta
num_of_iterations = 6000
```

In [85]:

```
def cost_function(X, Y, B):
    m = len(Y)
    J = np.sum((X.dot(B) - Y) ** 2)/(2 * m)
    return J
```

In [86]:

```
# Gradient descent algorithm.
# 1) Calculate the hypothesis value for each row(B0x0 + B1x1 + B2X2 +...+BnXn)
# 2) Calculate the loss (difference between hypothesis and y value of data set)
# 3) Gradient calculation
# 4) Add a new record of the cost
def batch_gradient_descent(x, y, theta, alpha, m, iterations_num):

    cost_history = [0] * iterations_num

    for i in range(0, iterations_num):
        # Hypothesis value
        hypothesis = np.dot(x, theta)
        #print("hypotesis: {}".format(hypotesis))

        # Loss
        loss = hypothesis - y
        #print("loss: {}".format(loss))

        # Gradient Calculation
        gradient = np.dot(np.transpose(x), loss) / m
        #print("gradient: {}".format(gradient))

        # Vectorization way to update theta values
        theta = theta - alpha * gradient
        #theta[0] = theta[0] - alpha * gradient[0] # Update theta0
        #theta[1] = theta[1] - alpha * gradient[1] # Update theta1

        # New Cost Value
        cost = cost_function(x, y, theta)
        cost_history[i] = cost

    return theta, cost_history
```

In [87]:

```
thetas_result, cost_history = batch_gradient_descent(X, Y, theta, alpha, m, num_of_iterations)
print(thetas_result)
```

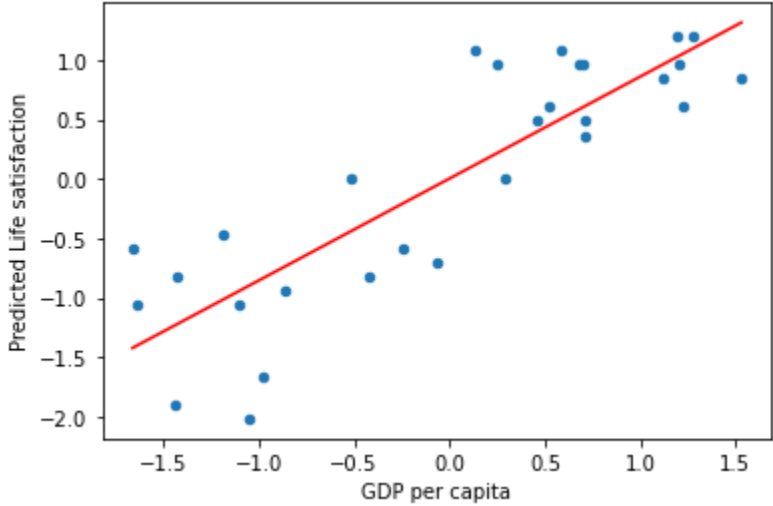
[0.00247132 0.85743032]

In [88]:

```
# For plot purposes let's calculate all the "calculated" y for given x
y_calculated = X.dot(thetas_result)
X_for_plot = X
Y_for_plot = y_calculated
```

In [89]:

```
# Visualize our prediction line in the data set
normalized_country_stats.plot(kind="scatter", x="GDP per capita", y="Life satisfaction")
plt.plot(X_for_plot["GDP per capita"], Y_for_plot, 'r')
plt.ylabel('Predicted Life satisfaction')
plt.xlabel('GDP per capita')
plt.show()
```



In [90]:

```
def predict_value(x_new, theta):
    predicted_value = theta[0] + theta[1]*x_new
    return predicted_value
```

In [91]:

```
# Predict the output (Life satisfaction) for the X input = 22587 GDP and 40000
country_stats_mean_GDP, life_satisfaction_mean = country_stats.mean()
country_stats_std, life_satisfaction_std = country_stats.std()
print("life_satisfaction_mean {}, life_satisfaction_std: {}".format(life_satisfaction_mean, life_satisfaction_std))
print(predict_value(((22587 - country_stats_mean_GDP)/country_stats_std), thetas_result))
print(predict_value(((40000 - country_stats_mean_GDP)/country_stats_std), thetas_result))
```

life_satisfaction_mean 6.493103448275863, life_satisfaction_std: 0.8396134461264043
-0.62990215204203
0.38923481865121135

In [92]:

```
first_predicted_value = predict_value(((22587 - country_stats_mean_GDP)/country_stats_std), thetas_result)
print("first predicted value without normalization: {}".format((first_predicted_value * life_satisfaction_std) +
                                                                life_satisfaction_mean))

second_predicted_value = predict_value(((40000 - country_stats_mean_GDP)/country_stats_std), thetas_result)
print("second predicted value without normalization: {}".format((second_predicted_value * life_satisfaction_std) +
                                                                life_satisfaction_mean))
```

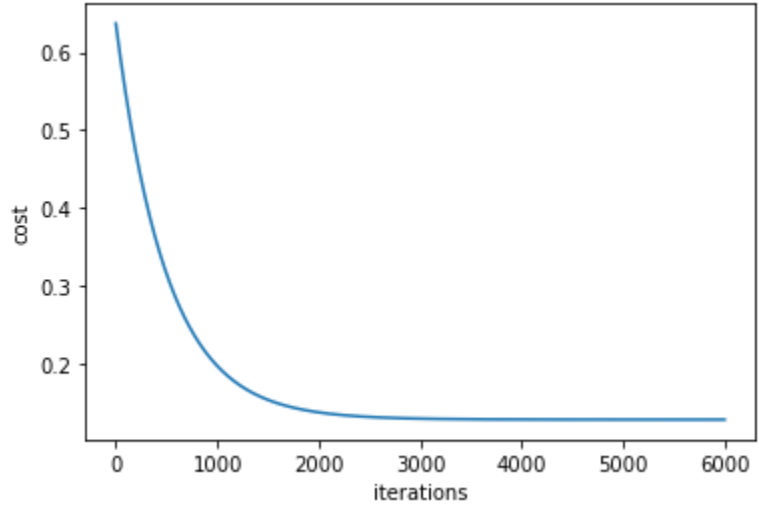
first predicted value without normalization: 5.964229131677416
second predicted value without normalization: 6.819910235715993

In [93]:

```
# Visualice the cost fuction for each iteration in the batch gradient descend algorithm
iterations = list(range(0,num_of_iterations))
plt.plot(iterations, cost_history, label='linear') # Plot some data on the (implicit) axes.
plt.xlabel('iterations')
plt.ylabel('cost')
```

Out[93]:

Text(0, 0.5, 'cost')



In []:

$$\theta = (\mathbf{X}^T \cdot X)^{-1} \cdot (\mathbf{X}^T \cdot Y)$$

```
In [2]: def prepare_country_stats(oecd_bli, gdp_per_capita):
        oecd_bli = oecd_bli[oecd_bli["INEQUALITY"] != "TOT"]
        oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
        gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
        gdp_per_capita.set_index("Country", inplace=True)
        full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                      left_index = True, right_index = True)
        full_country_stats.sort_values(by="GDP per capita", inplace= True)
        remove_indices = [0,1,6,8,33,34,35]
        keep_indices = list(set(range(36)) - set(remove_indices))
        return full_country_stats[["GDP per capita","Life satisfaction"]].iloc[keep_indices]
```

```
In [3]: import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
In [4]: # Load the data
oecd_bli = pd.read_csv("handson-ml\datasets\lifesat\oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv("handson-ml\datasets\lifesat\gdp_per_capita.csv", thousands=",", delimiter='\t',
                             encoding='latin1', na_values="n/a")
```

```
In [5]: # Brief description of the dataset
gdp_per_capita.head(5)
```

Out[5]:

	Country	Subject Descriptor	Units	Scale	Country/Series-specific Notes	2015	Estimates Start After
0	Afghanistan	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	599.994	2013.0
1	Albania	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	3995.383	2010.0
2	Algeria	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	4318.135	2014.0
3	Angola	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	4100.315	2014.0
4	Antigua and Barbuda	Gross domestic product per capita, current prices	U.S. dollars	Units	See notes for: Gross domestic product, curren...	14414.302	2011.0

```
In [6]: # Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
```

```
In [7]: # Build the pices for Normal equation
thetas = np.zeros((country_stats.size,1))
ones = np.ones((country_stats[country_stats.columns[0]].count(),1))
X = country_stats["GDP per capita"].to_frame()
Y = country_stats["Life satisfaction"].to_frame()
```

```
In [8]: # Append an extra column of ones to the fearute vector (X)
X.insert(loc=0, column='X0', value=ones)
```

```
In [9]: # Apply the formula
thetas = np.linalg.inv(np.transpose(X).dot(X)).dot((np.transpose(X).dot(Y)))
```

```
In [10]: print(thetas)
X.shape
```

Out[10]:

```
[[4.85305280e+00]
 [4.91154459e-05]]
(29, 2)
```

Create a function to predict new values taking account the hypotesis function $h_{\theta}(x) = \theta_0 + \theta_1 X_1$

```
In [11]: def predit_value(x_new, thetas):
        predicted_value = thetas[0] + thetas[1]*x_new
        return predicted_value
```

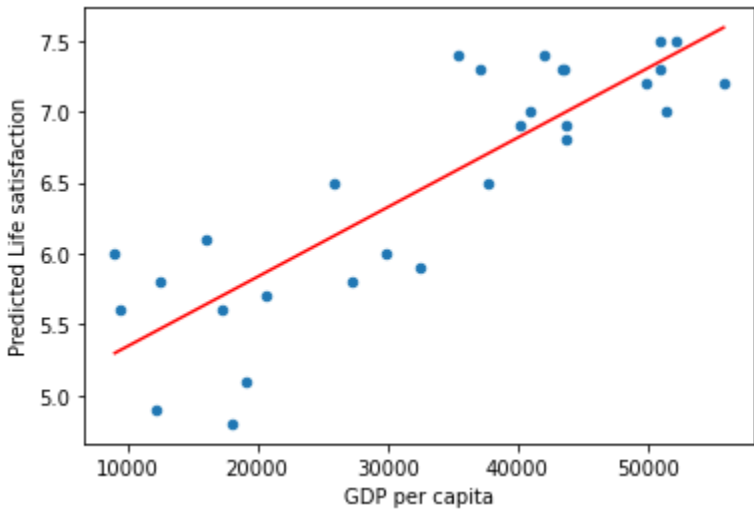
```
In [12]: # Predict the output (Life satisfaction) for the X input = 22587 GDP
print(predit_value(22587, thetas))
print(predit_value(40000, thetas))
```

Out[12]:

```
[5.96242338]
[6.81767064]
```

```
In [13]: # For plot purposes let's calculate all the "calculated" y for given x
y_calculated = X.dot(thetas)
X_for_plot = np.array(X['GDP per capita'].values.reshape(1,29)).ravel()
Y_for_plot = np.array(y_calculated.values.reshape(1,29)).ravel() # ravel() to remove extra bracket in numpy array
```

```
In [14]: # Visualize the initial data set
country_stats.plot(kind="scatter", x="GDP per capita", y="Life satisfaction")
plt.plot(X_for_plot, Y_for_plot, 'r')
plt.ylabel('Predicted Life satisfaction')
plt.xlabel('GDP per capita')
plt.show()
```



```
In [ ]:
```