In [ ]:
```python
import numpy as np
from sklearn import datasets, linear_model, metrics
```

Overview:

To predict the progression of diabetes in patients, we will use linear regression with gradient descent in this hands-on assignment. In this tutorial, you will learn how to implement linear regression with gradient descent in Python.

Using scikit-learn, a Python machine learning library, we will first load and train a linear regression model. Our implementation will be tested against the results of scikit-learn. Our next step will be to implement linear regression using gradient descent.

Dataset:

The following code illustrates how to load and split the dataset. Visit the following link (https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html) for more information about the dataset.

In [ ]:
```python
# Load diabetes dataset
diabetes = datasets.load_diabetes()
diabetes_X = diabetes.data  # matrix of dimensions 442x10

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

Data description:

A total of 442 diabetes patients are included in the dataset. There are 10 input variables for each patient - age, sex, body mass index, average blood pressure, and six measurements of blood serum. The blood serum measurements are: Total Cholesterol (TC), Low Density Lipoprotein (LDL), High Density Lipoprotein (HDL), TC/HDL, Low Tension Glaucoma (LTG) and Glucose. The target is a quantitative measure of disease progression after one year.

Sanity check:

In order to determine whether our implementation is correct, we will use the results from scikit-learn. The linear regression model in scikit-learn can be trained with just a call to function fit on the model since scikit-learn is a machine learning library. You can see the documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html).

In [ ]:
```
# with scikit learn:
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
mean_squared_error = metrics.mean_squared_error(diabetes_y_test, di
abetes_y_pred)
print("Mean squared error: %.2f" % mean_squared_error)
print("="*80)
```

We can use the above values of the mean squared error (2004.57) as the target value for the mean squared error for our implementation. We can also compare the coefficients after training our model to check if we get the same results. Note that the numbers might not match exactly, but as long as they match reasonable well (say within 1%), we should be fine.

Implementing Linear Regression with Gradient Descent

Finally, it's time to implement linear regression ourselves. Here's a template for you to get started.

In [ ]:
```
# train
X = diabetes_X_train
y = diabetes_y_train

# train: init
W = ...
b = ...

learning_rate = ...
epochs = ...

# train: gradient descent
for i in range(epochs):
    # calculate predictions
    # TODO

    # calculate error and cost (mean squared error - use can use th
e imported function metrics.mean_squared_error)
    # TODO

    # calculate gradients
    # TODO

    # update parameters
    # TODO

    # diagnostic output
    if i % 5000 == 0:
        print("Epoch %d: %f" % (i, mean_squared_error))
```