1. Write a Python function called `LocalAvg` that has the following function definition

   ```
   def LocalAvg(Y, X, tgrid, bw):
   ```

   - `Y` - should be a 1-dimensional ndarray with length n.
   - `X` - should be a 1-dimensional ndarray with length n.
   - `tgrid` - should be an 1-dimensional ndarray with length q.
   - `bw` - should be a `float` numeric variable.

   This function should return a 1-dimensional ndarray with length q.

   The k^{th} element of the returned array should be the mean of the elements of `Y` only using the indeces where `X` is greater than `tgrid[k] - bw` and less than `tgrid[k] + bw`.

   If `X` has no elements that are greater than `tgrid[k] - bw` and less than `tgrid[k] + bw`, then the k^{th} element of the returned array should equal zero.

   As an example, if `Y = np.array([3, 7, 12, 1, 8, 17])`, `X = np.array([0, 1, 2, 3, 4, 5])`, `tgrid = np.array([1.5, 3.5, 10.5])`, and `bw = 1.0`, then the function call `LocalAvg(Y, X, tgrid, bw)` should return the NumPy array `[9.5 4.5 0.]`.

   Check that your function works properly by running the following Python code:

   ```
   Y = np.array([3, 7, 12, 1, 8, 17, 23, 26])
   X = np.arange(8)
   tgrid = np.array([1.5, 3.5, 6.0, 11.0])

   print( LocalAvg(Y=Y, X=X, tgrid=tgrid, bw=0.1) )
   print( LocalAvg(Y=Y, X=X, tgrid=tgrid, bw=0.6) )
   print( LocalAvg(Y=Y, X=X, tgrid=tgrid, bw=1.0) )
   print( LocalAvg(Y=Y, X=X, tgrid=tgrid, bw=12.0) )
   print( LocalAvg(Y=Y, X=X, tgrid=np.array([2.0]), bw=1.0) )
   ```

2. For this problem you will use the `diabetes` dataset from the `sklearn.datasets` library. You can load this dataset using the following Python code:

   ```
   import numpy as np
   import pandas as pd
   from sklearn.datasets import load_diabetes

   diabet = load_diabetes()
   ```

```
num_diabet = diabet.data  # This is a 2-d numpy array with dimension 442 x 10
                          # Each column from num_diabet represents a different
                          # variable from the diabetes data
outcome = diabet.target  # This will be a 1-d numpy array with length 442
var_names = diabet.feature_names  # List of column names for num_diabet
```

(a) What is the mean and median of the numbers in the `outcome` array? How many of the elements in `outcome` are in between 100 and 200 (that is, greater than 100 and less than 200)?

(b) Compute the 25^{th}, 50^{th}, and 75^{th} percentiles of the age column in `num_diabet`. Store these three values in a 1-d ndarray called `perc_age`.

(c) Create a 1-d ndarray with length 442 that is called `age_category`. The i^{th} element of `age_category` should be filled in using the following rule:

* `age_category[i] = 0` if the i^{th} component of the `age` column is less than the 25^{th} percentile of `age`.
* `age_category[i] = 1` if the i^{th} component of the `age` column is greater than or equal to the 25^{th} percentile of `age` and less than the 50^{th} percentile of `age`.
* `age_category[i] = 2` if the i^{th} component of the `age` column is greater than or equal to the 50^{th} percentile of `age` and less than the 75^{th} percentile of `age`.
* `age_category[i] = 3` if the i^{th} component of the `age` column is greater than or equal to the 75^{th} percentile of `age`.

(d) Compute the median of the `outcome` array for the subset of observations where `age_category == k`. Do this for $k = 0$, $k = 1$, $k = 2$, and $k = 3$.

(e) Create a `dict` with 10 *key-value* pairs, the 10 keys are the column names of `num_diabet`, and the corresponding values are the maximum value from the numbers of that column. For example, the value associated with the key `'age'` should be the maximum value of the numbers from the `age` column.