Technical University of Denmark

Course name	Introduction to programming and data processing
Course number	02631, 02632, 02633, 02634, 02692
Aids allowed	All Aids
Exam duration	2 hours
Weighting	All exercises have equal weight

Contents

Submission details

You must hand in your solution electronically:

1. You can test your solutions individually on CodeJudge

   `https://dtu.codejudge.net/prog-jan19/assignments`

   under *Exam*. When you hand in a solution on CodeJudge, the test example given in the assignment description will be run on your solution. If your solution passes this single test, it will appear as *Submitted*. This means that your solution passes on this single test example. You can upload to CodeJudge as many times as you like during the exam.

2. You must upload all your solutions at DTU's Onlineeksamen site. Each assignment must be uploaded as one separate .py file, given the same name as the function in the assignment:

   (a) `count_unique_rows.py`
   (b) `compute_pi.py`
   (c) `interest.py`
   (d) `computeAssignments.py`
   (e) `robustLocation.py`

   The files must be handed in separately (*not* as a zip-file) and must have these exact filenames.

After the exam, your solutions submitted to DTU Inside will be automatically evaluated on CodeJudge on a range of different tests, to check that they work correctly in general. The assessment of your solution is based only on how many of the automated tests it passes.

- Make sure that your code follows the specifications exactly.
- Each solution shall not contain any additional code beyond the specified function, though `import` statements can be included.
- Remember, you can check if your solutions follow the specifications by uploading them to CodeJudge.
- Note that all vectors and matrices used as input or output must be numpy arrays.

Assignment A Unique rows in array

Problem definition

Create a function named `count_unique_rows` that takes as input a 2-dimentional array of numbers and returns the number of unique rows regardless of the order of the numbers in each row. If the first entry of a row contains the number 2, then the complete row should not be counted.

Solution template

```python
def count_unique_rows(x):
  #insert your code
  return count
```

Input	
`x`	Array with numbers
Output	
`count`	Count for the number of unique rows regardless of order of the entries within the row and ignoring rows where the first entry is 2

Example

Consider the following array

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 2 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 2 \end{bmatrix}. \tag{1}$$

Here the count is 2 as rows 1, 2 and 4 are the same (The 4th row has just an other order compared to rows 1 and 2), rows 5 and 6 are also the same, and row 3 contains the number 2 in the first entry.

A

Assignment B Approximating π

π can be given by the following equation

$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots = \frac{\pi^2}{6} \tag{2}$$

The equation contains an infinite sum which cannot be computed in finite time. Instead we will approximate π by only summing c terms, here for $c = 3$

$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} = \frac{v^2}{6}, \qquad v \approx \pi \tag{3}$$

A different formula to compute π is

$$\frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \cdots = \frac{\pi^4}{90} \tag{4}$$

Problem definition

Create a function named `compute_pi` that takes as input two numbers and returns an approximation of π, where the number of terms in the approximation is determined by the first input argument and where the second input argument determines the type of approximation.

Solution template

```python
def compute_pi(c, t):
  #insert your code
  return value
```

Input

`c`	Non-zero number which indicates the number of terms
`t`	A number that is either a 2 or 4, where 2 corresponding to selecting equation 2 or equation 4, respectively

Output

`value`	Approximation to π based on c terms and the equation type t

Example

For $c = 3$ and $t = 2$ the result, v (`value`), is

$$v = \sqrt{6\left(1 + \frac{1}{4} + \frac{1}{9}\right)} \approx 2.8577. \tag{5}$$

For $c = 2$ and $t = 4$ the result is

$$v = \sqrt[4]{90\left(1 + \frac{1}{2^4}\right)} \approx 3.1271. \tag{6}$$

B

Assignment C Compound interest calculator

If you deposit an amount P in the bank at a fixed annual interest rate R, after N years the amount in your account will have increased to $A = P(1+R)^N$. If three of the four variables A, P, R, and N are known, the fourth unknown can be computed from the formula as:

$$\begin{aligned} A &= P(1+R)^N \\ P &= \frac{A}{(1+R)^N} \\ R &= \sqrt[N]{\frac{A}{P}} - 1 \\ N &= \frac{\log(A) - \log(P)}{\log(1+R)} \end{aligned}$$

Problem definition

Create a function named `interest` that takes as input the four variables A, P, R, and N. One of the four input variables will contain the special value `math.nan`. The function must compute the value of this variable according to the appropriate formula above, and return the computed value.

Solution template

```python
def interest(A, P, R, N):
  #insert your code
  return val
```

Input	
`A`, `P`, `R`, `N`	Amount, principal, interest rate and number of years (decimal numbers, one of these takes the value `math.nan`).
Output	
`val`	Computed value for the "missing" input (decimal number).

Example

Consider the following input:

`A`: 1000, `P`: 200, `R`:nan `N`:5

Since R is given as nan, we must compute the value of R:

$$R = \sqrt[N]{\frac{A}{P}} - 1 = \sqrt[5]{\frac{1000}{200}} - 1 \approx \underline{0.3797297}$$

C

Assignment D Compute assignments for multidimensional cluster analysis

In cluster analysis, we would like to assign each of N multidimensional data points to one of M multidimensional cluster centers (centroids). The data points are represented in a data matrix, $\mathbf{X}$, and the centroids are represented in a centroid matrix, $\mathbf{C}$. The task is to assign each row in the data matrix to the closest centroid (row).

The distances are computed as the squared Euclidean distance. For instance, the squared distance, $d_{m,n}$, between the m'th centroid and the n'th data point is computed as

$$d_{m,n} = \sum_p (x_{n,p} - c_{m,p})^2 \tag{7}$$

The n'th data point, $\mathbf{x}_n$, is assigned to the centroid, $\mathcal{C}_{\tilde{m}}$, if the distance $d_{\tilde{m},n}$ is less than or equal to other distances from that data point to any other centroid.

$$\mathcal{C}_{\tilde{m}} = \{x_n : d_{\tilde{m},n} <= d_{m,n}, \forall m \leq M\} \tag{8}$$

Problem definition

Create a function name `computeAssignments` that takes two matrices as input and returns center indices in a vector, where the indices start from 1. One matrix, $\mathbf{C}(M \times P)$, represents centers where each row is a vector representing a centroid. The other matrix, $\mathbf{X}(N \times P)$, is a data matrix where each row represents a multidimensional data point. The number of columns of the centroid and the data matrix, P, is the same and it may range from one and upwards. The number of rows in either matrices may range from one and upwards.

Solution template

```python
def computeAssignments(C, X):
    #insert your code
    return assignments
```

Input	
`C`	Centroid matrix
`X`	Data matrix
Output	
`assignments`	Vector with elements from 1 to M

Example

$$\mathbf{C} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 0 & 0 \\ -4 & -1 \\ 3 & 3 \end{bmatrix} \tag{9}$$

$$\begin{aligned}
d_{1,1} &= d([0,1],[0,0]) = (0-0)^2 + (0-1)^2 = 1 && (10)\\
d_{2,1} &= d([-2,-3],[0,0]) = (0-(-2))^2 + (0-(-3))^2 = 4+9 = 13 && (11)\\
d_{1,2} &= d([0,1],[-4,-1]) = (-4-0)^2 + (-1-1)^2 = 16+4 = 20 && (12)\\
d_{2,2} &= d([-2,-3],[-4,-1]) = (-4-(-2))^2 + (-1-(-3))^2 = 4+4 = 8 && (13)\\
d_{1,3} &= d([0,1],[3,3]) = (3-0)^2 + (3-1)^2 = 9+4 = 13 && (14)\\
d_{2,3} &= d([-2,-3],[3,3]) = (3-(-2))^2 + (3-(-3))^2 = 25+36 = 61 && (15)
\end{aligned}$$

Among the two distances, $d_{1,1}$ and $d_{2,1}$, for the first data point, it is the first centroid that is closet: $1 < 13$. For the second data point, the distance to the second centroid, $d_{2,2}$, is the closest $(8 < 20)$. For the third data point, the first centroid is closest $(13 < 61)$. The `assignments` vector, $\mathbf{a}$, is therefore:

$$\mathbf{a} = [1, 2, 1] \tag{16}$$

Assignment E Robust location parameter with missing values

Given a set of numbers, we would like to estimate a typical location parameter. Instead of using the mean or the conventional median, we will here use the median of the numbers concatenated with the mean of pairs, and where any "missing values" are ignored.

Problem definition

Create a function named `robustLocation` that takes a vector as input and returns the location parameter as the median of all pair-wise means (drawing without replacement and ignoring ordering) concatenated with the original data, excluding any missing values indicated with `-999`. If there is no non-missing values in the vector, then the function should return 0.

Solution template

```
def robustLocation(x):
  #insert your code
  return location
```

Input	
`x`	List of one or more elements, i.e., a vector. The list may include zero or more "missing values" indicated with -999. The rest is ordinary numbers.
Output	
`location`	Robust location parameter

Example

Consider the following vector, $\mathbf{x}$, with 5 element where the third element is indicated as "missing"

$$\mathbf{x} = [1, 1.5, -999, 19, 2] \tag{17}$$

All the pairs are (excluding the "missing value" indicated with `-999`)

$$\{1, 1.5\}, \{1, 19\}, \{1, 2\}, \{1.5, 19\}, \{1.5, 2\}, \{19, 2\} \tag{18}$$

Their pair-wise means are

$$1.25, 10, 1.5, 10.25, 1.75, 10.5 \tag{19}$$

Now we take these pair-wise means and concatenate them to the original data set, excluding the missing value.

$$\tilde{\mathbf{x}} = [1, 1.5, 19, 2, 1.25, 10, 1.5, 10.25, 1.75, 10.5] \tag{20}$$

The result, l (`location`), is the median of this vector. In the vector, 1.75 and 2 are the middle values and the median is computed as the mean of these two values, so the result becomes

$$l = (1.75 + 2)/2 = 1.875 \tag{21}$$

```
location = 1.875
```

E