Danmarks Tekniske Universitet

Kursusnavn	Introduktion til programmering og databehandling
Kursusnummer	02631, 02632, 02633, 02634, 02692
Hjælpemidler	Alle hjælpemidler er tilladt
Varighed	2 timer
Vægtning	Opgaverne vægtes ens

Indhold

Assignment A: Unique rows in array 2

Assignment B: Approximating π 3

Assignment C: Compound interest calculator 4

Assignment D: Compute assignments for multidimensional cluster analysis 5

Assignment E: Robust location parameter with missing values 6

Aflevering

Du skal aflevere dine løsninger elektronisk:

1. Du kan uploade dine løsninger individuelt på CodeJudge

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

   under *Exam*. Når du afleverer en løsning på CodeJudge bliver det test-eksempel som fremgår i opgavebeskrivelsen kørt på din løsning. Hvis din løsning består denne ene test, vil den fremgå som *Submitted*. Det betyder at din løsning består på dette ene test-eksempel. Du kan uploade til CodeJudge så mange gange som du ønsker under eksamen.

2. Du skal uploade alle dine løsninger på DTU's Onlineeksamen site. Hver assignment skal uploades som en separat .py fil med samme navn som funktionen i opgaven:

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

   Filerne skal afleveres separat (*ikke* som en zip-fil) og skal have præcis disse filnavne.

Efter eksamen vil dine løsninger fra DTU Inside blive automatisk evalueret på CodeJudge på en række forskellige tests for at undersøge om de generelt fungerer korrekt. Bedømmelsen af din aflevering foretages udelukkende på baggrund af hvor mange af de automatiserede test der bestås.

- Du skal sikre dig at din kode følger specifikationerne nøje.
- Hver løsning må ikke indeholde nogen yderligere kode udover den specificerede funktion, dog kan `import`-sætninger inkluderes.
- Husk at du kan tjekke om dine løsninger følger specifikationerne ved at uploade dem til CodeJudge.
- Bemærk at alle vektorer og matricer anvendt som input eller output skal være numpy arrays.

Assignment A Unique rows in array

Problemdefinition

Skriv en funktion med navnet `count_unique_rows` der tager som input et 2-dimensionelt array med tal og returnerer antallet af unikke rækker uanset rækkefølgen af tallene i hver række. Hvis den første element i en række indholder tallet 2 så skal hele rækken ikke tælles med.

Løsningsskabelon

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

Input

`x`	Array med tal.

Output

`count`	Antal af unikke rækker uanset elementernes orden inden for en række og hvor der ses bort fra rækker hvor tallet 2 forekommer i den først element

Eksempel

Forestil dig følgende array

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

Her er antallet 2, da rækkerne 1, 2 og 4 er de samme (4. række har blot en anden rækkefølge i forhold til 1. or 2. række.), række 5 og 6 er også ens og række 3 indeholder tallet 2 i den første element.

A

Assignment B Approximating π

π kan gives ved følgende formel

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

Formlen indeholder en uendelig sum der ikke kan beregnes i endelig tid. I stedet approksimerer vi π ved kun at summere c led, her for $c=3$

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

Med en anden formel kan π beregnes med

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

Problemdefinition

Skriv en funktion med navnet `compute_pi` der tager som input to tal or returnerer en approximation til π hvor antallet af led i approksimationen er bestemt af første input argument og hvor andet input argument bestemmer typen af approksimationen.

Løsningsskabelon

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

Input

`c`	Ikke-negativt tal der angiver antal led.
`t`	Et tal der er enten 2 eller 4, svarende til valg af, henholdsvis formel 2 eller formel 4

Output

`value`	Approksimationen til π baseret på c led og formeltypen t

Eksempel

For $c=3$ og $t=2$ er resultatet, v (`value`),

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

For $c=2$ og $t=4$ er resultatet

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

B

Assignment C Compound interest calculator

Hvis du indsætter et beløb P i banken til en fast årlig rentesats R, vil beløbet på din konto efter N år være steget til $A = P(1+R)^N$. Hvis tre af de fire variable A, P, R, og N er kendte, kan den fjerde ukendte beregnes ud fra formlen:

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

Problemdefinition

Skriv en funktion med navnet `interest` der tager som input de fire variable A, P, R og N. En af de fire input vil indeholde den særlige værdi `math.nan`. Funktionen skal beregne værdien af denne variabel via de tilsvarende ovenstående formel og returnere den beregnede værdi.

Løsningsskabelon

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

Input

`A`, `P`, `R`, `N`	Beløb, principal, rente og antal år (decimaltal, et af disse antager værdien nan).

Output

`val`	Beregnet værdi for det "manglende" input (decimaltal).

Eksempel

Forestil dig følgende input:

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

Da R antager værdien nan skal vi beregne værdien af 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

I klynge-analyse (clusteranalyse) vil vi gerne tilskrive hver af N flerdimensionelle datapunkter til et af M flerdimensioneller klyngecentre. Datapunkterne er repræsenteret i en datamatrix, $\mathbf{X}$, og centrene er repræsenteret i en centermatrix, $\mathbf{C}$. Opgaven er at tilskrive hver række i datamatricen til det nærmeste center (række).

Afstandene er beregnet som den kvadrede Euklidiske afstand. For eksempel er den kvadrede afstand, $d_{m,n}$, mellem det m'te center og det n'te datapunkt beregnet som

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

Det n'te datapunkt, $\mathbf{x}_n$, er tilskrevet centret $\mathcal{C}_{\tilde{m}}$, hvis afstanden $d_{\tilde{m},n}$ er mindre end eller lig med andre afstande fra det datapunkt til andre centre.

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

Problemdefinition

Skriv en funktion med navnet `computeAssignments` der tager to matricer som input og returnerer centerindeks i en vektor, hvor indeks begynder fra 1. En matrix, $\mathbf{C}(M \times P)$, repræsenterer centre hvor hver række er en vektor der repræsentere et center. Den anden matrix, $\mathbf{X}(N \times P)$, er en datamatrix hvor hver række repræsenterer et flerdimensionelt datapunkt. Antallet af kolonner af centermatricen og datamatricen, P, er ens og kan være fra 1 og opad. Antallet af rækker i hver matrix can være fra 1 og opad.

Løsningsskabelon

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

Input	
`C`	Center matrix
`X`	Datamatrix

Output	
`assignments`	Vector med elementer fra 1 til M

Eksempel

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

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

For de to afstande, $d_{1,1}$ og $d_{2,1}$, der tilhører det første datapunkt, er det første center det nærmeste: $1 < 13$. For det andet datapunkt er afstanden til det andet center, $d_{2,2}$, det nærmeste ($8 < 20$). For det tredje datapunkt er det første center nærmest ($13 < 61$). `assignments` vektoren, $\mathbf{a}$, er derfor:

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

Assignment E Robust location parameter with missing values

Givet et antal tal vil vi gerne estimere en typisk lokationsparameter. I stedet for at benytte middelværdien eller den konventionelle median vil vi her benytte median af tallene sammenkædet (*concatenated*) med middelværdien af par, og hvor "manglende værdier" er ignoreret.

Problemdefinition

Skriv en funktion med navnet `robustLocation` der tager en vektor som input og returnerer lokationsparameteren som medianen af alle parvise middelværdier (taget uden tilbagelægning og hvor rækkefølge er ignoreret) sammenkædet med de originale data, og hvor manglende værdier er angivet med `-999`. Hvis der ikke er nogen ikke-manglende værdier i vektoren, så skal funktionen returnere 0.

Løsningsskabelon

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

Input

`x`	Liste med et eller flere elements, dvs. en vektor. Listen kan indeholde nul eller flere "manglende værdier" indikeret med -999. Resten er almindelige tal.

Output

`location`	Robust lokationsparameter

Eksempel

Betragt den følgende vektor, $\mathbf{x}$, med 5 elementer hvor det tredje element er angivet som "manglende"

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

Alle parrene er (hvor de "manglende værdier" indikeret med `-999` er ekskluderet)

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

De parvise middelværdier er

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

Nu tager vi de parvise middelværdier og sammensætter dem med det originale datasæt, samtidig med at de manglende værdier ekskluderes.

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

Resultatet, l (`location`), er medianen af denne vektor. For vektoren er 1.75 og 2 de midterste værdier og medianen er beregnet som middelværdien af disse to værdier, så resultatet bliver

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

```
location = 1.875
```

E