

experimento 8: quebrou ao pedir normalização - 3f9fbd0eb

USER

o arquivo dados_50_50.csv anexo possui 4 colunas: IDENTIFICADOR, CARGO, PERCENT_PRI E DISTANCE_KM. A primeira variável é o código que identifica o funcionário, a segunda apresenta os códigos dos cargos, a terceira é um índice que varia de 0 a 1, sendo o 0 o melhor e, por fim, a última variável é a distância em KM entre o funcionário e o cargo que ele poderá assumir, ou seja, quanto maior a distância, pior. Resolva o problema de alocação codificando um algoritmo com o método húngaro de forma a minimizar o PERCENT_PRI ou a DISTANCE_KM ou os dois juntos, sempre normalizados. E seguida execute o código

BOT

To solve the problem of allocation using the Hungarian method, we need to minimize the cost of assigning employees (identified by `IDENTIFICADOR`) to positions (`CARGO`). The cost can be based on either `PERCENT_PRI`, `DISTANCE_KM`, or a combination of both, normalized to ensure comparability. Here's how we can approach the problem step by step:

Step 1: Understand the Hungarian Method

The Hungarian method is an optimization algorithm used to solve assignment problems. It minimizes the total cost of assigning `n` workers to `n` tasks, where the cost of assigning worker `i` to task `j` is given in a cost matrix.

Step 2: Normalize the Data

Since we have two variables (`PERCENT_PRI` and `DISTANCE_KM`) to minimize, we need to normalize them to ensure they are on the same scale. Normalization can be done using the formula:

$$\text{Normalized Value} = \frac{\text{Value} - \text{Min}(\text{Value})}{\text{Max}(\text{Value}) - \text{Min}(\text{Value})}$$

This will scale all values to the range [0, 1].

Step 3: Create the Cost Matrix

The cost matrix will be constructed based on the normalized values. Depending on the objective, the cost matrix can be:

1. Based on `PERCENT_PRI` only.
2. Based on `DISTANCE_KM` only.
3. A weighted combination of both (e.g., $\text{Cost} = w_1 \cdot \text{PERCENT_PRI} + w_2 \cdot \text{DISTANCE_KM}$).

Step 4: Implement the Hungarian Algorithm

The Hungarian algorithm will be applied to the cost matrix to find the optimal assignment that minimizes the total cost.

Step 5: Execute the Code

We will implement the above steps in Python, using the `scipy.optimize.linear_sum_assignment` function, which is an efficient implementation of the Hungarian method.

Python Code Implementation

Below is the Python code to solve the problem:

```
```python
import pandas as pd
import numpy as np
from scipy.optimize import linear_sum_assignment

Step 1: Load the data
file_name = "dados_50_50.csv"
data = pd.read_csv(file_name, sep=',')

Step 2: Normalize the columns
data['PERCENT_PRI_NORM'] = (data['PERCENT_PRI'] - data['PERCENT_PRI'].min()) / (data['PERCENT_PRI'].max() - data['PERCENT_PRI'].min())
data['DISTANCE_KM_NORM'] = (data['DISTANCE_KM'] - data['DISTANCE_KM'].min()) / (data['DISTANCE_KM'].max() - data['DISTANCE_KM'].min())

Step 3: Create the cost matrix
Option 1: Minimize PERCENT_PRI only
cost_matrix_percent = data.pivot(index='IDENTIFICADOR', columns='CARGO',
values='PERCENT_PRI_NORM').values

Option 2: Minimize DISTANCE_KM only
cost_matrix_distance = data.pivot(index='IDENTIFICADOR', columns='CARGO',
values='DISTANCE_KM_NORM').values

Option 3: Minimize a combination of both (e.g., equal weights)
weight_percent = 0.5
weight_distance = 0.5
cost_matrix_combined = weight_percent * cost_matrix_percent + weight_distance * cost_matrix_distance

Step 4: Solve the assignment problem using the Hungarian method
Minimize PERCENT_PRI
row_ind_percent, col_ind_percent = linear_sum_assignment(cost_matrix_percent)

Minimize DISTANCE_KM
row_ind_distance, col_ind_distance = linear_sum_assignment(cost_matrix_distance)

Minimize combined cost
row_ind_combined, col_ind_combined = linear_sum_assignment(cost_matrix_combined)

Step 5: Display the results
def display_results(row_ind, col_ind, cost_matrix, method_name):
 total_cost = cost_matrix[row_ind, col_ind].sum()
 assignments = list(zip(row_ind + 1, col_ind + 1)) # Add 1 to match IDENTIFICADOR and CARGO indices
 print(f"\n{method_name} Results:")
 print(f"Total Cost: {total_cost}")
 print("Assignments (IDENTIFICADOR -> CARGO):")
 for worker, task in assignments:
 print(f" {worker} -> {task}")

Display results for each method
display_results(row_ind_percent, col_ind_percent, cost_matrix_percent, "Minimize PERCENT_PRI")
display_results(row_ind_distance, col_ind_distance, cost_matrix_distance, "Minimize DISTANCE_KM")
display_results(row_ind_combined, col_ind_combined, cost_matrix_combined, "Minimize Combined Cost")
```
```

Explanation of the Code

1. **Data Loading**: The CSV file is loaded into a Pandas DataFrame.
2. **Normalization**: Both `PERCENT_PRI` and `DISTANCE_KM` are normalized to the range [0, 1].
3. **Cost Matrix Construction**: Three cost matrices are created:
 - One for `PERCENT_PRI`.
 - One for `DISTANCE_KM`.
 - One for a weighted combination of both.
4. **Hungarian Algorithm**: The `linear_sum_assignment` function is used to find the optimal assignment for each cost matrix.

5. **Results Display**: The total cost and assignments are displayed for each objective.

Execution

Save the provided CSV file as `dados_50_50.csv` in the same directory as the script, then run the code. The output will show the optimal assignments and total costs for each objective.

Let me know if you need further assistance!

Execution Error: TypeError

Traceback (most recent call last)

Cell In[3], line 10

```
7 data = pd.read_csv(file_name, sep=';')
9 # Step 2: Normalize the columns
--> 10 data['PERCENT_PRI_NORM'] = (data['PERCENT_PRI'] - data['PERCENT_PRI'].min()) /
(data['PERCENT_PRI'].max() - data['PERCENT_PRI'].min())
11 data['DISTANCE_KM_NORM'] = (data['DISTANCE_KM'] - data['DISTANCE_KM'].min()) /
(data['DISTANCE_KM'].max() - data['DISTANCE_KM'].min())
13 # Step 3: Create the cost matrix
14 # Option 1: Minimize PERCENT_PRI only
```

...

TypeError: unsupported operand type(s) for -: 'str' and 'str'