

experimento 5: 4o run 4 - 11d48772ac

USER

A base de dados anexada no arquivo dados_50_50.csv possui quatro variáveis: 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. Gostaria que me propusesse uma metodologia para solucionar meu problema de alocação de forma a minimizar o PERCENT_PRI ou a DISTANCE_KM ou os dois juntos. Determine e execute o código para solucionar o problema e indique o custo total minimizado.

BOT

To solve the problem of allocating employees to positions while minimizing **PERCENT_PRI**, **DISTANCE_KM**, or both together, we can model this as an **assignment problem**. The assignment problem is a type of optimization problem where we aim to assign resources (employees) to tasks (positions) in a way that minimizes the total cost.

Here's a step-by-step methodology to solve the problem:

Step 1: Understand the Problem

- **Objective**: Minimize the total cost of allocation.
 - Cost can be based on:
 1. **PERCENT_PRI** (priority index).
 2. **DISTANCE_KM** (distance between employee and position).
 3. A **weighted combination** of both (e.g., $w_1 * \text{PERCENT_PRI} + w_2 * \text{DISTANCE_KM}$).
- **Constraints**:
 - Each employee can be assigned to only one position.
 - Each position can be assigned to only one employee.

This is a classic **linear assignment problem** that can be solved using optimization techniques like the **Hungarian algorithm** or linear programming.

Step 2: Formulate the Problem

1. **Decision Variables**:
 - Let $x[i, j]$ be a binary variable:
 - $x[i, j] = 1$ if employee i is assigned to position j .
 - $x[i, j] = 0$ otherwise.
2. **Objective Function**:
 - Minimize the total cost:
 - If minimizing **PERCENT_PRI**: $\text{minimize } \sum(i, j) \text{ PERCENT_PRI}[i, j] * x[i, j]$.
 - If minimizing **DISTANCE_KM**: $\text{minimize } \sum(i, j) \text{ DISTANCE_KM}[i, j] * x[i, j]$.
 - If minimizing both: $\text{minimize } \sum(i, j) (w_1 * \text{PERCENT_PRI}[i, j] + w_2 * \text{DISTANCE_KM}[i, j]) * x[i, j]$.
3. **Constraints**:
 - Each employee is assigned to exactly one position: $\sum(j) x[i, j] = 1$ for all i .
 - Each position is assigned to exactly one employee: $\sum(i) x[i, j] = 1$ for all j .

Step 3: Solve the Problem

We will use Python and the `scipy.optimize.linear_sum_assignment` function, which implements the Hungarian algorithm to solve the assignment problem efficiently.

Step 4: Implement the Solution

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_path = "dados_50_50.csv" # Replace with the correct file path
data = pd.read_csv(file_path, sep=";")

Step 2: Prepare the cost matrices
Convert PERCENT_PRI and DISTANCE_KM to numeric (replace commas with dots for decimals)
data['PERCENT_PRI'] = data['PERCENT_PRI'].str.replace(',', '.').astype(float)
data['DISTANCE_KM'] = data['DISTANCE_KM'].str.replace(',', '.').astype(float)

Pivot the data to create cost matrices
employees = data['IDENTIFICADOR'].unique()
positions = data['CARGO'].unique()

Create cost matrices for PERCENT_PRI and DISTANCE_KM
cost_percent_pri = data.pivot(index='IDENTIFICADOR', columns='CARGO', values='PERCENT_PRI').to_numpy()
cost_distance_km = data.pivot(index='IDENTIFICADOR', columns='CARGO', values='DISTANCE_KM').to_numpy()

Normalize the cost matrices (optional, for combined optimization)
cost_percent_pri /= cost_percent_pri.max()
cost_distance_km /= cost_distance_km.max()

Combine the costs (weighted sum)
w1, w2 = 0.5, 0.5 # Weights for PERCENT_PRI and DISTANCE_KM
combined_cost = w1 * cost_percent_pri + w2 * cost_distance_km

Step 3: Solve the assignment problem
row_ind, col_ind = linear_sum_assignment(combined_cost)

Step 4: Calculate the total minimized cost
total_cost = combined_cost[row_ind, col_ind].sum()

Step 5: Display the results
assignments = pd.DataFrame({
 "Employee": employees[row_ind],
 "Position": positions[col_ind],
 "Cost": combined_cost[row_ind, col_ind]
})

print("Optimal Assignments:")
print(assignments)
print(f"Total Minimized Cost: {total_cost}")
```

```

Step 5: Explanation of the Code

1. **Data Preparation**:

- The data is loaded and cleaned (e.g., converting `PERCENT_PRI` and `DISTANCE_KM` to numeric).
- Cost matrices are created for `PERCENT_PRI` and `DISTANCE_KM`.

2. **Cost Combination**:

- If both criteria are to be minimized, a weighted combination of the two matrices is created.

3. **Optimization**:

- The `linear_sum_assignment` function is used to find the optimal assignment that minimizes the total cost.

4. **Results**:

- The optimal assignments and the total minimized cost are displayed.

Step 6: Output

The output will include:

1. A table showing the optimal assignment of employees to positions and the associated cost for each assignment.
2. The total minimized cost.

Step 7: Next Steps

- Adjust the weights `w1` and `w2` to prioritize either **PERCENT_PRI** or **DISTANCE_KM**.
- Analyze the results to ensure the solution aligns with your business goals.

Let me know if you need further assistance!

Execution Result:

Optimal Assignments:

| | Employee | Position | Cost |
|----|----------|----------|----------|
| 0 | 1 | 14 | 0.609065 |
| 1 | 2 | 2 | 0.658924 |
| 2 | 3 | 3 | 0.769260 |
| 3 | 4 | 5 | 0.439027 |
| 4 | 5 | 55 | 0.420782 |
| 5 | 6 | 17 | 0.931846 |
| 6 | 7 | 56 | 0.512248 |
| 7 | 8 | 6 | 0.695864 |
| 8 | 9 | 57 | 0.511853 |
| 9 | 10 | 7 | 0.962081 |
| 10 | 11 | 70 | 0.392220 |
| 11 | 12 | 9 | 0.043284 |
| 12 | 13 | 10 | 0.341080 |
| 13 | 14 | 11 | 0.907152 |
| 14 | 15 | 58 | 0.531797 |
| 15 | 16 | 59 | 0.724956 |
| 16 | 17 | 60 | 0.668815 |
| 17 | 18 | 12 | 0.832347 |
| 18 | 19 | 71 | 0.679455 |
| 19 | 20 | 18 | 0.622527 |
| 20 | 21 | 61 | 0.726129 |
| 21 | 22 | 15 | 0.715622 |
| 22 | 23 | 16 | 0.948931 |
| 23 | 24 | 20 | 0.332597 |
| 24 | 25 | 22 | 0.529328 |
| 25 | 26 | 62 | 0.042555 |
| 26 | 27 | 19 | 0.545412 |
| 27 | 28 | 25 | 0.764711 |
| 28 | 29 | 21 | 0.673072 |
| 29 | 30 | 29 | 0.532049 |
| 30 | 31 | 23 | 0.507430 |
| 31 | 32 | 63 | 0.658180 |
| 32 | 33 | 64 | 0.611315 |
| 33 | 34 | 24 | 0.522967 |
| 34 | 35 | 65 | 0.668815 |
| 35 | 36 | 73 | 0.593167 |
| 36 | 37 | 66 | 0.689401 |
| 37 | 38 | 26 | 0.313706 |
| 38 | 39 | 27 | 0.647186 |
| 39 | 40 | 67 | 0.647868 |
| 40 | 41 | 28 | 0.521633 |
| 41 | 42 | 69 | 0.348171 |
| 42 | 43 | 68 | 0.479090 |
| 43 | 44 | 8 | 0.502807 |
| 44 | 45 | 13 | 0.589484 |
| 45 | 46 | 30 | 0.789665 |
| 46 | 47 | 1 | 0.508048 |
| 47 | 48 | 72 | 0.658180 |
| 48 | 49 | 4 | 0.423548 |
| 49 | 50 | 74 | 0.562435 |

Total Minimized Cost: 29.308085556184693

