```
#%% md
# Verisk - computer purchase problem

*by Jolanta Śliwa*
#%% md
## Problem Statement
Suppose you're trying to help a company determine which computers to purchase.
### Data - utilization data by employee:
The company has been able to pull utilization data by employee that classifies users into 3 bins, depending on how much they use their computer in their work:
* Low usage - spends a lot of time in meetings, checking email, doing people management
* Average usage - requires some compute power, with balanced mix of heads down/technical work along with a
good amount of meetings/email writing
* High usage - power user, relies heavily on computer performance

#%%
import pandas as pd


utilization = pd.read_csv(
    "https://raw.githubusercontent.com/shubhamkalra27/dsep-2020/main/datasets/util_b_emp.csv"
)
#%%
utilization.info()
#%%
utilization.head()
#%% md
Checking types of utilization
#%%
utilization["utilization_bin"].unique()
#%% md
average usage is stored as medium
#%% md
```

```
## Data - survey
Additionally, they've surveyed employees to collect the relative importance of the following variables describing a
computer's performance:
* Memory
* Processing
* Storage
* Price inverse - this metric was given to you by the company as you can see in the dataset, with the directive that
price inverse being fixed at a 25% weight in the purchase decision
#%%
survey = pd.read_csv(
    "https://raw.githubusercontent.com/shubhamkalra27/dsep-2020/main/datasets/survey_emp.csv"
)
#%%
survey.info()
#%%
survey.head()
#%% md
Checking whether we have 100% in every column
#%%
for i, row in survey.iterrows():
    if sum(row[1:]) != 1:
        print("problem")
        break
print("ok")
#%% md
## Data - computers
Lastly, the company is looking to purchase a maximum of 3 different computer models, and have compiled the following
list scoring their memory, processing, storage, and relative price. Each dimension is scored from 0-10, with 10 being the best.
#%%
computers = pd.read_csv(
    "https://raw.githubusercontent.com/shubhamkalra27
```

```
/dsep-2020/main/datasets/vendor_options.csv"
)
#%%
computers.info()
#%%
computers.head()
#%% md
Checking "real" range of scores:
#%%
print(computers.max())
#%%
print(computers.min())
#%% md
## Task
**Given this information, provide the company with a recommendation on which computers to purchase.**
#%% md
List of parameters:
#%%
parameters = computers.columns[1:]
print(parameters)
#%% md
It will be more convenient for me to store all employees related data in one DataFrame instead of two.

Merging survey and utilization into employees column:
#%%
employees = utilization.merge(survey, left_on="employee_id", right_on="employee_id")
#%%
employees.info()
#%%
employees.head()
#%% md
Normally (if we would like to make some predictions) it would be better to store "object" (nominal data - categorical data) using one-hot-encoding but in this case it is more convenient for me to leave it like that.
```

```
 95 #%% md
 96 Under we will see outputs of different metrics. For
    each there will be solution provided by:
 97 * Simulated Annealing
 98 * Naive algorithm
 99 #%% md
100 Results given by algorithms: id of computer in
    DataFrame - not the one in colum computer_id
101 #%%
102 from problem import (
103     ProblemNothing,
104     ProblemMax,
105     ProblemScale,
106     ProblemMaxHalf,
107     ProblemScaleHalf,
108 )
109 #%% md
110 * ProblemNothing - metric without using utilization
    info
111 * ProblemMax - metric where every group have a
    different max score (3, 7, 10)
112 * ProblemMaxHalf - like above with different values
     (5, 7.5, 10)
113 * ProblemScale - metric where we scale computers
    scores by multiplying them (3/1, 3/2, 3/3)
114 * ProblemScaleHalf - like above with different
    values (4/2, 4/3, 4/4)
115 #%%
116 from simulated_annealing import SimulatedAnnealing,
    SimulatedAnnealingConfig
117 from naive_solution import Naive
118 #%% md
119 ### No utilization value
120 #%%
121 prob_n = ProblemNothing(computers, employees)
122 #%%
123 annealing_n = SimulatedAnnealing(
    SimulatedAnnealingConfig(), prob_n)
124
125 annealing_n.solve()
126 #%%
```

```
naive_n = Naive(prob_n)

naive_n.solve()
#%% md
As we can see simulated annealing sometimes returns solutions that aren't optimal
Let's try one more time
#%%
annealing_n = SimulatedAnnealing(SimulatedAnnealingConfig(), prob_n)

annealing_n.solve()
#%% md
This time we manage to get "the best" solution for this metric

Let's see how it looks like:
#%%
prob_n.calculate_state_cost([10, 1, 8])
#%%
prob_n.calculate_state_cost([8, 2, 1])
#%% md
So actually there are at least 2 optimal solutions
#%% md
### Problem MAX
#### max: (3, 7, 10)
#%%
prob_max = ProblemMax(computers, employees)
#%%
annealing_max = SimulatedAnnealing(SimulatedAnnealingConfig(), prob_max)

result_max = annealing_max.solve()
#%%
top_max = prob_max.get_wanted_computers(result_max)
#%%
print(top_max.keys())
#%%
naive_max = Naive(prob_max)

result_max_n = naive_max.solve()
```

```
#%%
print(prob_max.get_wanted_computers(result_max_n).
    keys())
#%% md
#### max: (5, 7.5, 10)
#%%
prob_max_half = ProblemMaxHalf(computers, employees)
#%%
annealing_max_half = SimulatedAnnealing(
    SimulatedAnnealingConfig(), prob_max_half)

result_max_half = annealing_max_half.solve()
#%%
print(prob_max_half.get_wanted_computers(
    result_max_half).keys())
#%%
naive_max_half = Naive(prob_max_half)

result_max_half_n = naive_max_half.solve()
#%%
print(prob_max_half.get_wanted_computers(
    result_max_half_n).keys())
#%% md
### Problem Scale
#### scale: (3/1, 3/2, 3/3)
#%%
prob_scale = ProblemScale(computers, employees)
#%%
annealing_scale = SimulatedAnnealing(
    SimulatedAnnealingConfig(), prob_scale)

result_scale = annealing_scale.solve()
#%%
print(prob_scale.get_wanted_computers(result_scale).
    keys())
#%%
naive_scale = Naive(prob_scale)

result_scale_n = naive_scale.solve()
#%%
print(prob_scale.get_wanted_computers(result_scale_n
```

```
198 ).keys())
199 #%% md
200 #### scale: (4/2, 4/3, 4/4)
201 #%%
202 prob_scale_half = ProblemScaleHalf(computers,
    employees)
203 #%%
204 annealing_scale_half = SimulatedAnnealing(
    SimulatedAnnealingConfig(), prob_scale_half)
205
206 result_scale_half = annealing_scale_half.solve()
207 #%%
208 print(prob_scale_half.get_wanted_computers(
    result_scale_half).keys())
209 #%%
210 naive_scale_half = Naive(prob_scale_half)
211
212 result_scale_half_n = naive_scale_half.solve()
213 #%%
214 print(prob_scale_half.get_wanted_computers(
    result_scale_half_n).keys())
215 #%% md
216 There is still a question which 3 computers are the
    best for that company?
217
218 We can see according to all presented metrics
    computer in row nr 1 is always in top three
219
220 Others can vary depending on metric we use. We can
    choose one of the above mathod but there are only 3
    candidates for 2 positions.
221 nr five wasn't choosen only by problem max
222 #%%
223 print(result_max_half)
224 #%%
225 print(prob_max_half.calculate_state_cost(
    result_max_half))
226 #%%
227 print(prob_max_half.calculate_state_cost([1, 3, 5]))
228 #%%
229 print(prob_max_half.calculate_state_cost([1, 5, 4]))
```

```
#%% md
For that metric there is not a big difference between computer 5 and 4
nr 3 seems to better option to leave
#%% md
On the other hand we see that nr 4 also was't choosen only by on metric: scale half
So it seems that we have final three
#%%
print(result_scale_half)
#%%
print(prob_scale_half.calculate_state_cost(result_scale_half))
#%%
print(prob_scale_half.calculate_state_cost([1, 5, 4]))
#%%
print(prob_scale_half.calculate_state_cost([1, 4, 3]))
#%% md
Computer nr 4 seems to be less valuable and changing it with 3 or 5 give almost the same results
#%% md
Just in case: Let's check nr 3 - wasn't choseen by 2 metrics
#%%
print(result_max)
#%%
print(prob_max.calculate_state_cost(result_max))
#%%
print(prob_max.calculate_state_cost([3, 5, 1]))
#%%
print(prob_max.calculate_state_cost([3, 4, 1]))
#%%
print(result_scale)
#%%
print(prob_scale.calculate_state_cost(result_scale))
#%%
print(prob_scale.calculate_state_cost([3, 5, 1]))
#%%
print(prob_scale.calculate_state_cost([3, 4, 1]))
```

```
#%% md
As we can see using above metrics and adding computer 3 instead of any other (4 or 5) results in a bigger i other cases derease of cost value
#%% md
Under we I displayed top three computers:
#%%
computers.iloc[[1, 4, 5]]
#%% md
seems like we ended up with rather ballanced final state
#%%
computers.iloc[[1, 4, 5]]["computer_id"]
#%% md
And here we have computer 3 as an addiction
#%%
computers.iloc[[3]]
#%% md
it is simmilar to computer nr 3 but with a slightly better price but worse memory
#%%

```