Exercise 3

Due date: 23/6/2021, submission in pairs

Appointed TA: Jonny

In this exercise you will implement a car agency that operates both purchasing and selling auctions for used cars.

Data

Your data file *Ex3_data.csv* contains ~70,000 used cars offered for sale by their owners. Each row specifies the offer i.d. , brand (Kia, Ford, VolksWagen, BMW or Ferrari), year of production (2008-2017), engine size and the "value" that is the offered price for that car.

id	brand	year	engine_size	value
id_11042	bmw	2016	3500	24350
id_7042	ford	2009	1400	100
id_1420	vw	2014	3600	15500
id_64357	ford	2012	2200	5950
id_79541	kia	2011	1200	4600
id_45687	bmw	2014	1800	13950

Table 1

The file *Uni_dist.csv* contains 536 fictional values sampled from the uniform distribution on [100,5100]. It is given just for your convenience, so you can verify the methods you implement in part B.

Code file

The only code file you need to submit is ex3_<id1>_<id2>.py that now contains all signatures waiting for your implementation. no need to submit main.py.

You are allowed to use pandas, math, ittertools.permutations and no other modules (in particular, numpy).

Good Luck!

Part A – VCG (40%)

The agency opens a new car selling garage. For that purpose, it purchases a variety of used cars via a procurement VCG auction. They want to buy a set of cars S that meets the following demands:

For some $k \in \mathrm{N}$,

- $|S| = 5k$

- There are exactly k cars of each brand (Kia, Ford, VolksWagen, BMW and Ferrari) in S.
- Production years of cars in S vary between 5 different (pre-determined) years, such that there are exactly k cars of each year.

For example, assume that k=2 and the years are 2010-2014. Then the following set of cars meets the criteria:

id_65714	kia	2010
id_6256	kia	2011
id_32702	vw	2014
id_9968	vw	2010
id_48649	ford	2011
id_75682	ford	2012
id_15830	ferrari	2014
id_44692	ferrari	2013
id_21913	bmw	2013
id_85267	bmw	2012

While this one does not:

id_59443	bmw	2010
id_38667	ford	2010
id_79541	kia	2011
id_6884	bmw	2012
id_15830	ferrari	2014
id_40322	vw	2011
id_13930	kia	2014
id_23481	vw	2012
id_38866	bmw	2014
id_44692	ferrari	2013

as it contains too many BMW's and 2014's and not enough Fords and 2013's.

Recall that a VCG auction selects the social welfare maximizing allocation and that payments are defined as every agent's externality on others:

$$P_i = \sum_{j \neq i} (agent\ j's\ utility\ in\ the\ auction) - \sum_{j \neq i} (agent\ j's\ utility\ in\ an\ auction\ without\ i)$$

In a procurement auction, agents offer the items and expect to be paid in return by the house, and not the other way around. Thus, every winning agent (i.e. that her car was bought) **loses** her car's value and gains the payment P_i that the auction assigns her. That is,

$$U_i = \begin{cases} -v_i + P_i & i\ wins \\ 0 & otherwise \end{cases}$$

For example, if the BMW in the first row in Table 1 is bought, its owner loses utility of 24,350 and then gains whatever her payment is. Hence, in our case selecting the social optimum means selecting a bundle with ***minimal*** sum of values that meets the criteria.

To select the socially optimal bundle, use the following algorithm. Let $B = \{ford, bmw, kia, volkswagen, ferrari\}$ and $Y = \{y_1, y_2, y_3, y_4, y_5, \}$. Denote Ω the set of all permutations (1:1 mappings) from B to Y. Now do the following steps k times:

1. For all $\sigma \epsilon \Omega$ (there are 5! =120 of them), Let S_σ be the set of all bundles of five cars that are still available (i.e. has not been selected previously), in which the brand-year assignment follows σ. For example if $\sigma(kia) = 2016, \sigma(ford) = 2012, \sigma(bmw) = 2010, \sigma(ferrari) = 2013 \; and \; \sigma(vw) = 2009$, then every bundle $s \; \epsilon S_\sigma$ consists of a 2016 kia, a 2012 ford, a 2010 bmw, a 2013 ferrari and a 2009 volkswagen.
2. Let $s *_\sigma = \; argmin_{s \epsilon S_\sigma}(\sum_{c \epsilon s} v(c))$ and $V(\sigma) = min_{s \epsilon S_\sigma}(\sum_{c \epsilon s} v(c))$ where $v(c)$ is the value of car $c \epsilon s$.
3. Let $\sigma_* \; = \; argmin_{\sigma \epsilon \Omega} V(\sigma)$
4. Add $s *_{\sigma_*}$ to the optimal bundle.

Meaning, it is a greedy algorithm that repeats k times, and at every step selects the minimal-value set of 5 cars that are still available, one for each year and for each brand.

****Bonus assignment**** (10 points)
Prove that this algorithm is indeed optimal. Submit your proof in a separate pdf file.
This might help - https://en.wikipedia.org/wiki/Matroid#Greedy_algorithm

a. Implement `opt_bnd(data: pd.Dataframe, k:int, years:list)→ Dict` that implements the above algorithm. It outputs the dictionary below:
{"cost": total cost of the winning bundle (int), "bundle": a [list] of all winning cars }.

b. Implement
`proc_VCG(data: pd.Dataframe, k:int, years:list)` → `{ID:P(ID)}`
that runs a procurement VCG auction as on the given bidding data, for the specified k and list of years. The output is a dictionary in which keys are ID's of the winning bids and the values are their corresponding payments.

Part B – Expected Revenue

A car *type* is defined by its brand, production year and engine size (we assume that these three parameters indicate the model or at least for some set of "similar" models). In this part we look at every car type separately. You will conduct a procurement VCG auction for some j cars of a specific type and then sell them in a regular VCG auction. The idea of course is to sell the cars for higher average price than the purchase price. We will look at the bidding data from the procurement auction as a histogram of values, then infer the underlying values distribution in order to predict our revenue when we sale these cars in future auctions. (We assume that values distributions for buyers

and sellers are the same). To do that, implement the attributes and methods of the class "Type" below.

**You are welcome to run your code on the Uni_dist.csv file and check that it makes sense. For example, you can verify that the CDF you implemented fairly approximates the uniform CDF.*

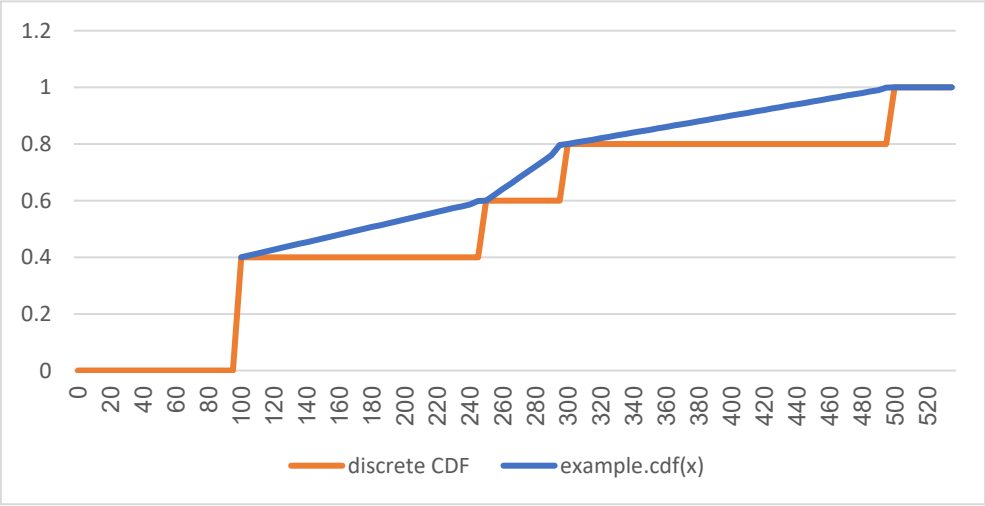
a. The class variables `"cars_num"` and `"buyers_num"` are the number of cars you buy and the numbers of bidders in the future auction where you sale these cars, respectively.

b. The instance variable `data` is a list of all bids for that object. For example, let example.data = [100, 300,250, 100 ,500].

c. **`avg_buy`**`(self)` → `float`
runs a procurement VCG auction on the self.data for *cars_num* cars and returns the average price paid for a winning car.
For example if *cars_num = 1,* example.avg_buy()=100

d. **`cdf`**`(self,x:float)` → `float`
returns F(x) where F is the piecewise-linear CDF defined by *self.data*.

$$\rightarrow \text{example.cdf(x)} = \begin{cases} 0 & x < 100 \\ 0.4 + 0.2\left(\frac{x-100}{250-100}\right) & 100 \le x < 250 \\ 0.6 + 0.2\left(\frac{x-250}{300-250}\right) & 250 \le x < 300 \\ 0.8 + 0.2\left(\frac{x-300}{500-300}\right) & 250 \le x < 300 \\ 1 & 500 \le x \end{cases}$$



e. `os_cdf(self,r:int,n:int,x:float)→float`
returns $F_{X_{(r)}}(x)$ where $F_{X_{(r)}}(\cdot)$ is the r out of n order statistic cdf for distribution F_X. It is given by the following formula-

$$F_{X_{(r)}}(x) = \sum_{j=r}^{n} \binom{n}{j} [F_X(x)]^j [1 - F_X(x)]^{n-j}$$

($F_X(x)$ = self.cdf(x) you implemented above)

f. `exp_rev(self)→ float`
returns the expected revenue in a regular VCG auction for `cars_num` cars and `buyers_num` buyers, based on the histogram `self.data` (and using the cdf functions above). Note that the `self.data` is used here only for constructing the CDF's on which we compute the expected revenue in a future auction. **Do not wrongly take it as the realization of bids in that auction.**

g. `exp_rev_median(self,n∈{2,3})→ float`
returns the expected revenue in a one car 2nd price auction with n buyers, with a reserve price that is the median of *self.data*. We will run it for n = 2,3.

Part C – Competition (10 points)

In this part we want to improve expected profit by adding a reserve price to the auction.
The method `reserve_price(self)→ float`
should return your suggestion for a reserve price in the same auction you implemented in Part B section d. (Hint: you can actually compute an optimal reserve price).