Home Assignment 1
Machine Learning A

FEDERICO FIORIO

September 10, 2022

1 Make Your Own (10 points)

1. Regarding the profile information, I think it would be interesting to find features which are correlated to the final grades.
These features could be:

- average grades in previous courses
- final grade in calculus
- final grade in statistics

So the sample space would be:

$$\mathcal{X} : \mathbb{R} \times \mathbb{N} \times \mathbb{N}$$

2. The label space would be the set of possible obtainable grades, considering the danish scale it is (maybe I'm wrong I don't know the danish scale yet, I'm sorry):
$\mathcal{Y} : \{12, 10, 7, 4, 2, 0\}$

3. The grades can be used as labels but also as numbers in order to perform regression, in case of regression, however when the algorithm will try to predict the results it will not obtain grades belonging to the $\mathbb{N}$ set, so I will have to map each $\mathbb{R}$ number predicted to the closest grade number after the prediction. For example 3,1 will become a 4.
Since I'm dealing with regression I can use the square-loss function:

$$l(y', y) = (y' - y)^2$$

to obtain grade that make sense, when I want to make a prediction the result has to be rounded to the closest grade
4. I would use the euclidean distance defined as:

$$d(x, x') = \sqrt{\sum_{i=1}^{d}(x_i - x'_i)^2}$$

but for KNN we could save computation just by doing

$$\sum_{i=1}^{d}(x_i - x_i')^2$$

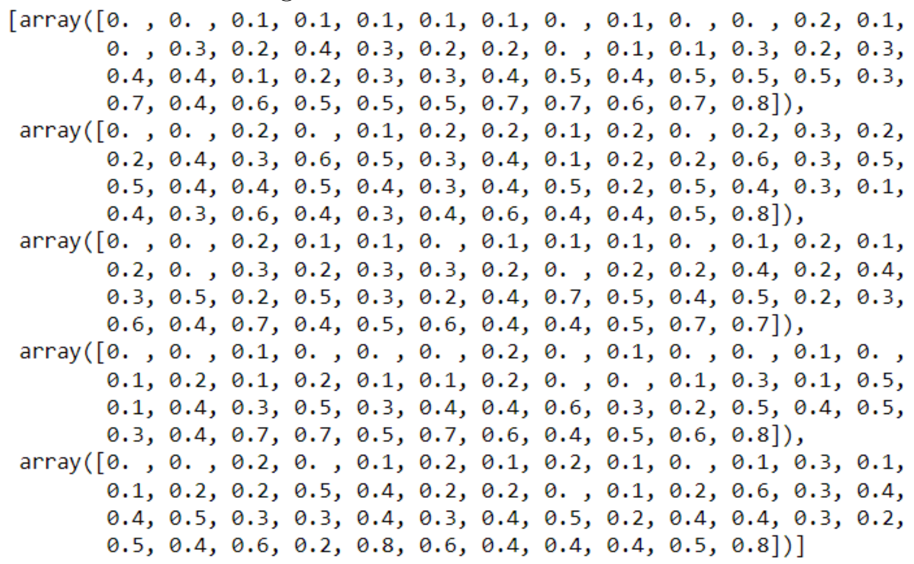
5. I would divide the given dataset into: S_{train}, S_{val} and S_{test}. On S_{train} I would train the various instances of the model, we can use KNN as example.
on S_{val} I would take the best K; so the best possible classifier based on the previously defined error function. (the one with minimized loss function, so here I am evaluating the models based on the error function)
On S_{test} I evaluate the performance of the algorithm on new samples based on the same error function (this time unbiased). This last step will give me an idea of how the algorithm behave when I try to deploy it.

6. yes I would expect some issues, maybe the data on which I trained the algorithm wasn't very representative, so the deployed algorithm would fail to generalize, in this case, I would need more data in order to try to improve the algorithm performances.
It might also happen that the algorithm learns too well the noise and patterns between the data points and at the end will fail to generalize, in this case, I would try to use simpler models in order to obtain better results.

2 Digits Classification with K Nearest Neighbors (45 points)

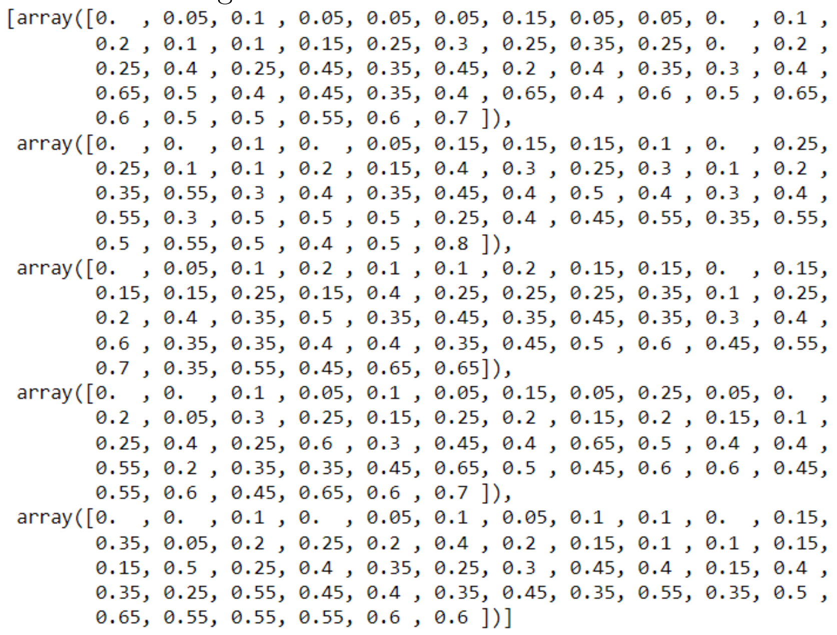
The validation error was computed as error rate, this means that for each validation set of size n for $n \in \{10, 20, 40, 80\}$ I computed the number of samples missclassified for each k and normalized by the size n.
So for example, in the first validation set (1 of 5) for n = 10, if I get for k = 4, 5 miss-classified samples, it means that the error rate for k = 4 is $\frac{5}{10}$
The following are the 4 images for the 4 possibles values of n, a line is represented by an array, an array has len 50 as the number of k requested, each number in the array represent the error rate for that k (K = number of the cell of the array +1) and that specific validation set.
In each image there are the results of 5 (as the size of the set i) validation sets of size n.

Figure 1: validation sets of size 10



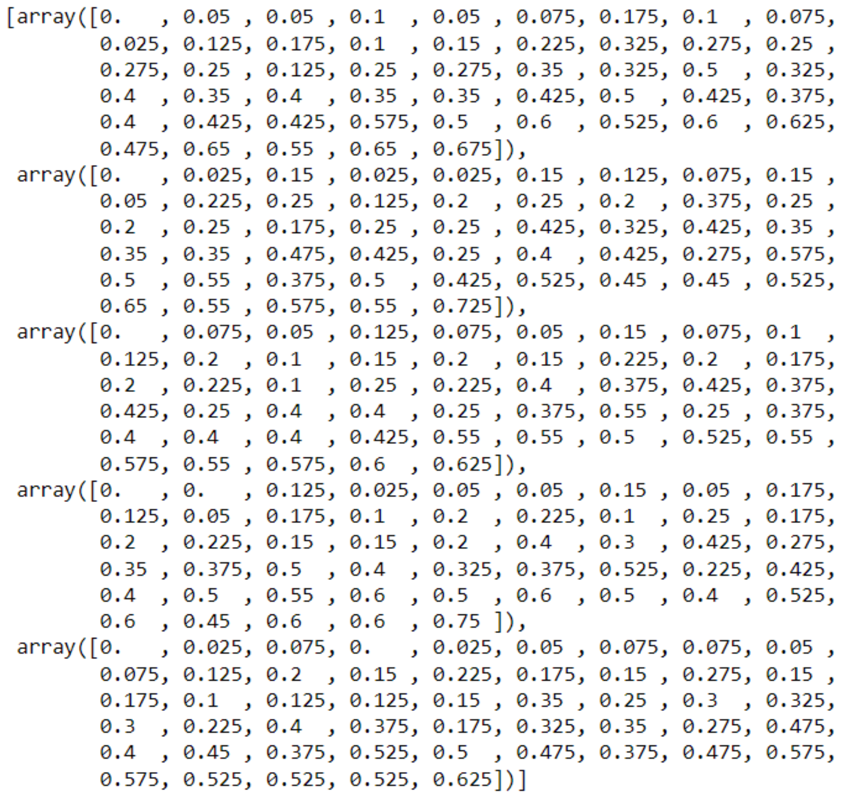
```
[array([0. , 0. , 0.1, 0.1, 0.1, 0.1, 0.1, 0. , 0.1, 0. , 0. , 0.2, 0.1,
        0. , 0.3, 0.2, 0.4, 0.3, 0.2, 0.2, 0. , 0.1, 0.1, 0.3, 0.2, 0.3,
        0.4, 0.4, 0.1, 0.2, 0.3, 0.3, 0.4, 0.5, 0.4, 0.5, 0.5, 0.5, 0.3,
        0.7, 0.4, 0.6, 0.5, 0.5, 0.5, 0.7, 0.7, 0.6, 0.7, 0.8]),
 array([0. , 0. , 0.2, 0. , 0.1, 0.2, 0.2, 0.1, 0.2, 0. , 0.2, 0.3, 0.2,
        0.2, 0.4, 0.3, 0.6, 0.5, 0.3, 0.4, 0.1, 0.2, 0.2, 0.6, 0.3, 0.5,
        0.5, 0.4, 0.4, 0.5, 0.4, 0.3, 0.4, 0.5, 0.2, 0.5, 0.4, 0.3, 0.1,
        0.4, 0.3, 0.6, 0.4, 0.3, 0.4, 0.6, 0.4, 0.4, 0.5, 0.8]),
 array([0. , 0. , 0.2, 0.1, 0.1, 0. , 0.1, 0.1, 0.1, 0. , 0.1, 0.2, 0.1,
        0.2, 0. , 0.3, 0.2, 0.3, 0.3, 0.2, 0. , 0.2, 0.2, 0.4, 0.2, 0.4,
        0.3, 0.5, 0.2, 0.5, 0.3, 0.2, 0.4, 0.7, 0.5, 0.4, 0.5, 0.2, 0.3,
        0.6, 0.4, 0.7, 0.4, 0.5, 0.6, 0.4, 0.4, 0.5, 0.7, 0.7]),
 array([0. , 0. , 0.1, 0. , 0. , 0. , 0.2, 0. , 0.1, 0. , 0. , 0.1, 0. ,
        0.1, 0.2, 0.1, 0.2, 0.1, 0.1, 0.2, 0. , 0. , 0.1, 0.3, 0.1, 0.5,
        0.1, 0.4, 0.3, 0.5, 0.3, 0.4, 0.4, 0.6, 0.3, 0.2, 0.5, 0.4, 0.5,
        0.3, 0.4, 0.7, 0.7, 0.5, 0.7, 0.6, 0.4, 0.5, 0.6, 0.8]),
 array([0. , 0. , 0.2, 0. , 0.1, 0.2, 0.1, 0.2, 0.1, 0. , 0.1, 0.3, 0.1,
        0.1, 0.2, 0.2, 0.5, 0.4, 0.2, 0.2, 0. , 0.1, 0.2, 0.6, 0.3, 0.4,
        0.4, 0.5, 0.3, 0.3, 0.4, 0.3, 0.4, 0.5, 0.2, 0.4, 0.4, 0.3, 0.2,
        0.5, 0.4, 0.6, 0.2, 0.8, 0.6, 0.4, 0.4, 0.4, 0.5, 0.8])]
```

Figure 2: validation sets of size 20



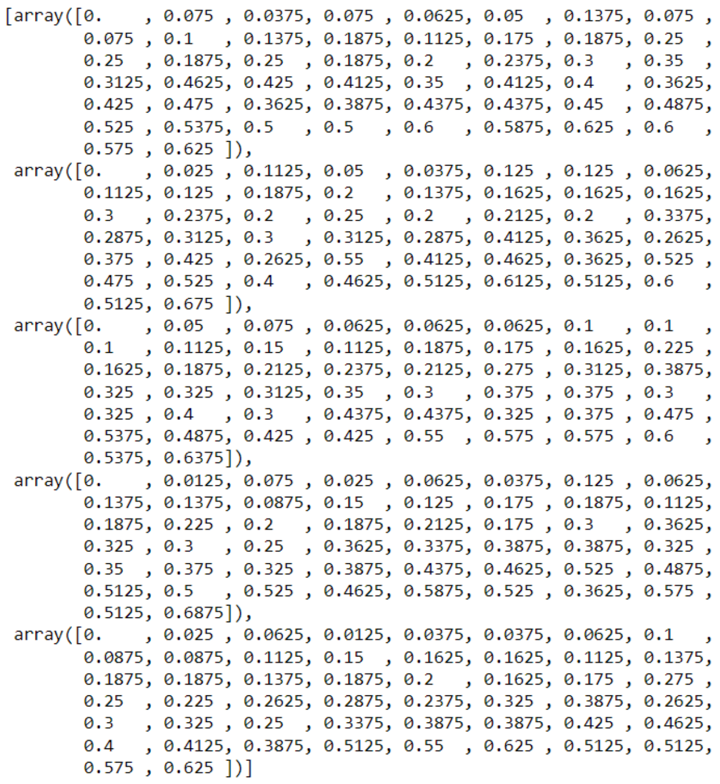
```
[array([0.  , 0.05, 0.1 , 0.05, 0.05, 0.05, 0.15, 0.05, 0.05, 0.  , 0.1 ,
        0.2 , 0.1 , 0.1 , 0.15, 0.25, 0.3 , 0.25, 0.35, 0.25, 0.  , 0.2 ,
        0.25, 0.4 , 0.25, 0.45, 0.35, 0.45, 0.2 , 0.4 , 0.35, 0.3 , 0.4 ,
        0.65, 0.5 , 0.4 , 0.45, 0.35, 0.4 , 0.65, 0.4 , 0.6 , 0.5 , 0.65,
        0.6 , 0.5 , 0.5 , 0.55, 0.6 , 0.7 ]),
 array([0.  , 0.  , 0.1 , 0.  , 0.05, 0.15, 0.15, 0.15, 0.1 , 0.  , 0.25,
        0.25, 0.1 , 0.1 , 0.2 , 0.15, 0.4 , 0.3 , 0.25, 0.3 , 0.1 , 0.2 ,
        0.35, 0.55, 0.3 , 0.4 , 0.35, 0.45, 0.4 , 0.5 , 0.4 , 0.3 , 0.4 ,
        0.55, 0.3 , 0.5 , 0.5 , 0.5 , 0.25, 0.4 , 0.45, 0.55, 0.35, 0.55,
        0.5 , 0.55, 0.5 , 0.4 , 0.5 , 0.8 ]),
 array([0.  , 0.05, 0.1 , 0.2 , 0.1 , 0.1 , 0.2 , 0.15, 0.15, 0.  , 0.15,
        0.15, 0.15, 0.25, 0.15, 0.4 , 0.25, 0.25, 0.25, 0.35, 0.1 , 0.25,
        0.2 , 0.4 , 0.35, 0.5 , 0.35, 0.45, 0.35, 0.45, 0.35, 0.3 , 0.4 ,
        0.6 , 0.35, 0.35, 0.4 , 0.4 , 0.35, 0.45, 0.5 , 0.6 , 0.45, 0.55,
        0.7 , 0.35, 0.55, 0.45, 0.65, 0.65]),
 array([0.  , 0.  , 0.1 , 0.05, 0.1 , 0.05, 0.15, 0.05, 0.25, 0.05, 0.  ,
        0.2 , 0.05, 0.3 , 0.25, 0.15, 0.25, 0.2 , 0.15, 0.2 , 0.15, 0.1 ,
        0.25, 0.4 , 0.25, 0.6 , 0.3 , 0.45, 0.4 , 0.65, 0.5 , 0.4 , 0.4 ,
        0.55, 0.2 , 0.35, 0.35, 0.45, 0.65, 0.5 , 0.45, 0.6 , 0.6 , 0.45,
        0.55, 0.6 , 0.45, 0.65, 0.6 , 0.7 ]),
 array([0.  , 0.  , 0.1 , 0.  , 0.05, 0.1 , 0.05, 0.1 , 0.1 , 0.  , 0.15,
        0.35, 0.05, 0.2 , 0.25, 0.2 , 0.4 , 0.2 , 0.15, 0.1 , 0.1 , 0.15,
        0.15, 0.5 , 0.25, 0.4 , 0.35, 0.25, 0.3 , 0.45, 0.4 , 0.15, 0.4 ,
        0.35, 0.25, 0.55, 0.45, 0.4 , 0.35, 0.45, 0.35, 0.55, 0.35, 0.5 ,
        0.65, 0.55, 0.55, 0.55, 0.6 , 0.6 ])]
```

Figure 3: validation sets of size 40

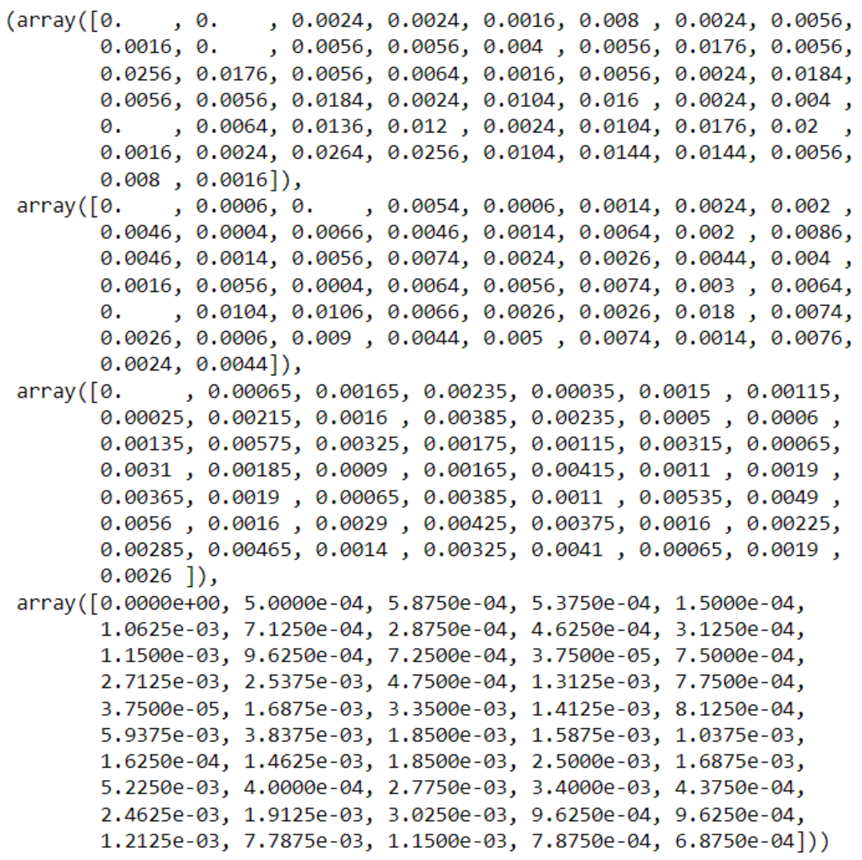


```
[array([0.   , 0.05 , 0.05 , 0.1  , 0.05 , 0.075, 0.175, 0.1  , 0.075,
        0.025, 0.125, 0.175, 0.1  , 0.15 , 0.225, 0.325, 0.275, 0.25 ,
        0.275, 0.25 , 0.125, 0.25 , 0.275, 0.35 , 0.325, 0.5  , 0.325,
        0.4  , 0.35 , 0.4  , 0.35 , 0.35 , 0.425, 0.5  , 0.425, 0.375,
        0.4  , 0.425, 0.425, 0.575, 0.5  , 0.6  , 0.525, 0.6  , 0.625,
        0.475, 0.65 , 0.55 , 0.65 , 0.675]),
 array([0.   , 0.025, 0.15 , 0.025, 0.025, 0.15 , 0.125, 0.075, 0.15 ,
        0.05 , 0.225, 0.25 , 0.125, 0.2  , 0.25 , 0.2  , 0.375, 0.25 ,
        0.2  , 0.25 , 0.175, 0.25 , 0.25 , 0.425, 0.325, 0.425, 0.35 ,
        0.35 , 0.35 , 0.475, 0.425, 0.25 , 0.4  , 0.425, 0.275, 0.575,
        0.5  , 0.55 , 0.375, 0.5  , 0.425, 0.525, 0.45 , 0.45 , 0.525,
        0.65 , 0.55 , 0.575, 0.55 , 0.725]),
 array([0.   , 0.075, 0.05 , 0.125, 0.075, 0.05 , 0.15 , 0.075, 0.1  ,
        0.125, 0.2  , 0.1  , 0.15 , 0.2  , 0.15 , 0.225, 0.2  , 0.175,
        0.2  , 0.225, 0.1  , 0.25 , 0.225, 0.4  , 0.375, 0.425, 0.375,
        0.425, 0.25 , 0.4  , 0.4  , 0.25 , 0.375, 0.55 , 0.25 , 0.375,
        0.4  , 0.4  , 0.4  , 0.425, 0.55 , 0.55 , 0.5  , 0.525, 0.55 ,
        0.575, 0.55 , 0.575, 0.6  , 0.625]),
 array([0.   , 0.   , 0.125, 0.025, 0.05 , 0.05 , 0.15 , 0.05 , 0.175,
        0.125, 0.05 , 0.175, 0.1  , 0.2  , 0.225, 0.1  , 0.25 , 0.175,
        0.2  , 0.225, 0.15 , 0.15 , 0.2  , 0.4  , 0.3  , 0.425, 0.275,
        0.35 , 0.375, 0.5  , 0.4  , 0.325, 0.375, 0.525, 0.225, 0.425,
        0.4  , 0.5  , 0.55 , 0.6  , 0.5  , 0.6  , 0.5  , 0.4  , 0.525,
        0.6  , 0.45 , 0.6  , 0.6  , 0.75 ]),
 array([0.   , 0.025, 0.075, 0.   , 0.025, 0.05 , 0.075, 0.075, 0.05 ,
        0.075, 0.125, 0.2  , 0.15 , 0.225, 0.175, 0.15 , 0.275, 0.15 ,
        0.175, 0.1  , 0.125, 0.125, 0.15 , 0.35 , 0.25 , 0.3  , 0.325,
        0.3  , 0.225, 0.4  , 0.375, 0.175, 0.325, 0.35 , 0.275, 0.475,
        0.4  , 0.45 , 0.375, 0.525, 0.5  , 0.475, 0.375, 0.475, 0.575,
        0.575, 0.525, 0.525, 0.525, 0.625])]
```

Figure 4: validation sets of size 80



```
[array([0.    , 0.075 , 0.0375, 0.075 , 0.0625, 0.05  , 0.1375, 0.075 ,
        0.075 , 0.1   , 0.1375, 0.1875, 0.1125, 0.175 , 0.1875, 0.25  ,
        0.25  , 0.1875, 0.25  , 0.1875, 0.2   , 0.2375, 0.3   , 0.35  ,
        0.3125, 0.4625, 0.425 , 0.4125, 0.35  , 0.4125, 0.4   , 0.3625,
        0.425 , 0.475 , 0.3625, 0.3875, 0.4375, 0.4375, 0.45  , 0.4875,
        0.525 , 0.5375, 0.5   , 0.5   , 0.6   , 0.5875, 0.625 , 0.6   ,
        0.575 , 0.625 ]),
 array([0.    , 0.025 , 0.1125, 0.05  , 0.0375, 0.125 , 0.125 , 0.0625,
        0.1125, 0.125 , 0.1875, 0.2   , 0.1375, 0.1625, 0.1625, 0.1625,
        0.3   , 0.2375, 0.2   , 0.25  , 0.2   , 0.2125, 0.2   , 0.3375,
        0.2875, 0.3125, 0.3   , 0.3125, 0.2875, 0.4125, 0.3625, 0.2625,
        0.375 , 0.425 , 0.2625, 0.55  , 0.4125, 0.4625, 0.3625, 0.525 ,
        0.475 , 0.525 , 0.4   , 0.4625, 0.5125, 0.6125, 0.5125, 0.6   ,
        0.5125, 0.675 ]),
 array([0.    , 0.05  , 0.075 , 0.0625, 0.0625, 0.0625, 0.1   , 0.1   ,
        0.1   , 0.1125, 0.15  , 0.1125, 0.1875, 0.175 , 0.1625, 0.225 ,
        0.1625, 0.1875, 0.2125, 0.2375, 0.2125, 0.275 , 0.3125, 0.3875,
        0.325 , 0.325 , 0.3125, 0.35  , 0.3   , 0.375 , 0.375 , 0.3   ,
        0.325 , 0.4   , 0.3   , 0.4375, 0.4375, 0.325 , 0.375 , 0.475 ,
        0.5375, 0.4875, 0.425 , 0.425 , 0.55  , 0.575 , 0.575 , 0.6   ,
        0.5375, 0.6375]),
 array([0.    , 0.0125, 0.075 , 0.025 , 0.0625, 0.0375, 0.125 , 0.0625,
        0.1375, 0.1375, 0.0875, 0.15  , 0.125 , 0.175 , 0.1875, 0.1125,
        0.1875, 0.225 , 0.2   , 0.1875, 0.2125, 0.175 , 0.3   , 0.3625,
        0.325 , 0.3   , 0.25  , 0.3625, 0.3375, 0.3875, 0.3875, 0.325 ,
        0.35  , 0.375 , 0.325 , 0.3875, 0.4375, 0.4625, 0.525 , 0.4875,
        0.5125, 0.5   , 0.525 , 0.4625, 0.5875, 0.525 , 0.3625, 0.575 ,
        0.5125, 0.6875]),
 array([0.    , 0.025 , 0.0625, 0.0125, 0.0375, 0.0375, 0.0625, 0.1   ,
        0.0875, 0.0875, 0.1125, 0.15  , 0.1625, 0.1625, 0.1125, 0.1375,
        0.1875, 0.1875, 0.1375, 0.1875, 0.2   , 0.1625, 0.175 , 0.275 ,
        0.25  , 0.225 , 0.2625, 0.2875, 0.2375, 0.325 , 0.3875, 0.2625,
        0.3   , 0.325 , 0.25  , 0.3375, 0.3875, 0.3875, 0.425 , 0.4625,
        0.4   , 0.4125, 0.3875, 0.5125, 0.55  , 0.625 , 0.5125, 0.5125,
        0.575 , 0.625 ])]
```



```
(array([0.    , 0.    , 0.0024, 0.0024, 0.0016, 0.008 , 0.0024, 0.0056,
        0.0016, 0.    , 0.0056, 0.0056, 0.004 , 0.0056, 0.0176, 0.0056,
        0.0256, 0.0176, 0.0056, 0.0064, 0.0016, 0.0056, 0.0024, 0.0184,
        0.0056, 0.0056, 0.0184, 0.0024, 0.0104, 0.016 , 0.0024, 0.004 ,
        0.    , 0.0064, 0.0136, 0.012 , 0.0024, 0.0104, 0.0176, 0.02  ,
        0.0016, 0.0024, 0.0264, 0.0256, 0.0104, 0.0144, 0.0144, 0.0056,
        0.008 , 0.0016]),
 array([0.    , 0.0006, 0.    , 0.0054, 0.0006, 0.0014, 0.0024, 0.002 ,
        0.0046, 0.0004, 0.0066, 0.0046, 0.0014, 0.0064, 0.002 , 0.0086,
        0.0046, 0.0014, 0.0056, 0.0074, 0.0024, 0.0026, 0.0044, 0.004 ,
        0.0016, 0.0056, 0.0004, 0.0064, 0.0056, 0.0074, 0.003 , 0.0064,
        0.    , 0.0104, 0.0106, 0.0066, 0.0026, 0.0026, 0.018 , 0.0074,
        0.0026, 0.0006, 0.009 , 0.0044, 0.005 , 0.0074, 0.0014, 0.0076,
        0.0024, 0.0044]),
 array([0.     , 0.00065, 0.00165, 0.00235, 0.00035, 0.0015 , 0.00115,
        0.00025, 0.00215, 0.0016 , 0.00385, 0.00235, 0.0005 , 0.0006 ,
        0.00135, 0.00575, 0.00325, 0.00175, 0.00115, 0.00315, 0.00065,
        0.0031 , 0.00185, 0.0009 , 0.00165, 0.00415, 0.0011 , 0.0019 ,
        0.00365, 0.0019 , 0.00065, 0.00385, 0.0011 , 0.00535, 0.0049 ,
        0.0056 , 0.0016 , 0.0029 , 0.00425, 0.00375, 0.0016 , 0.00225,
        0.00285, 0.00465, 0.0014 , 0.00325, 0.0041 , 0.00065, 0.0019 ,
        0.0026 ]),
 array([0.0000e+00, 5.0000e-04, 5.8750e-04, 5.3750e-04, 1.5000e-04,
        1.0625e-03, 7.1250e-04, 2.8750e-04, 4.6250e-04, 3.1250e-04,
        1.1500e-03, 9.6250e-04, 7.2500e-04, 3.7500e-05, 7.5000e-04,
        2.7125e-03, 2.5375e-03, 4.7500e-04, 1.3125e-03, 7.7500e-04,
        3.7500e-05, 1.6875e-03, 3.3500e-03, 1.4125e-03, 8.1250e-04,
        5.9375e-03, 3.8375e-03, 1.8500e-03, 1.5875e-03, 1.0375e-03,
        1.6250e-04, 1.4625e-03, 1.8500e-03, 2.5000e-03, 1.6875e-03,
        5.2250e-03, 4.0000e-04, 2.7750e-03, 3.4000e-03, 4.3750e-04,
        2.4625e-03, 1.9125e-03, 3.0250e-03, 9.6250e-04, 9.6250e-04,
        1.2125e-03, 7.7875e-03, 1.1500e-03, 7.8750e-04, 6.8750e-04]))
```

Figure 5: Variance over the 5 validation sets for each n as a function of K

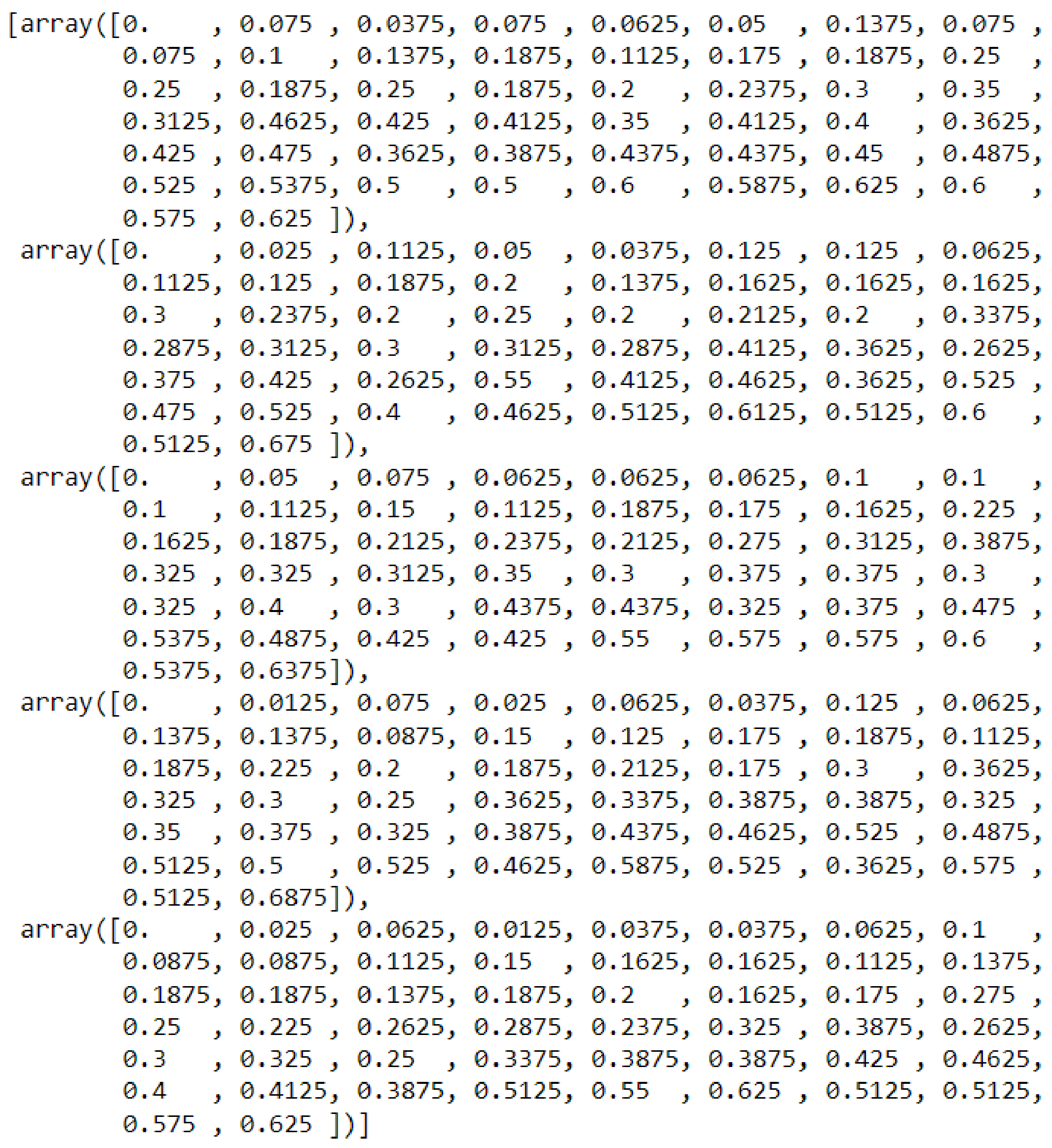
- What can you say about fluctuations of the validation error as a function of n?

I observed that as n increases the fluctuations of the validation error decrease.

- What can you say about the prediction accuracy of K-NN as a function of K?

Once again as K increases the accuracy in prediction should decrease, but the highest accuracy doesn't indicate the best K for K-NN, in fact in the extreme case of $K = 0$, where we take into consideration for prediction only the sample that we want to predict, we obtain an accuracy of 100% but, it is actually the worst possible model for deployment, because it won't evaluate new samples.

2.1 Task 2



```
[array([0.    , 0.075 , 0.0375, 0.075 , 0.0625, 0.05  , 0.1375, 0.075 ,
        0.075 , 0.1   , 0.1375, 0.1875, 0.1125, 0.175 , 0.1875, 0.25  ,
        0.25  , 0.1875, 0.25  , 0.1875, 0.2   , 0.2375, 0.3   , 0.35  ,
        0.3125, 0.4625, 0.425 , 0.4125, 0.35  , 0.4125, 0.4   , 0.3625,
        0.425 , 0.475 , 0.3625, 0.3875, 0.4375, 0.4375, 0.45  , 0.4875,
        0.525 , 0.5375, 0.5   , 0.5   , 0.6   , 0.5875, 0.625 , 0.6   ,
        0.575 , 0.625 ]),
 array([0.    , 0.025 , 0.1125, 0.05  , 0.0375, 0.125 , 0.125 , 0.0625,
        0.1125, 0.125 , 0.1875, 0.2   , 0.1375, 0.1625, 0.1625, 0.1625,
        0.3   , 0.2375, 0.2   , 0.25  , 0.2   , 0.2125, 0.2   , 0.3375,
        0.2875, 0.3125, 0.3   , 0.3125, 0.2875, 0.4125, 0.3625, 0.2625,
        0.375 , 0.425 , 0.2625, 0.55  , 0.4125, 0.4625, 0.3625, 0.525 ,
        0.475 , 0.525 , 0.4   , 0.4625, 0.5125, 0.6125, 0.5125, 0.6   ,
        0.5125, 0.675 ]),
 array([0.    , 0.05  , 0.075 , 0.0625, 0.0625, 0.0625, 0.1   , 0.1   ,
        0.1   , 0.1125, 0.15  , 0.1125, 0.1875, 0.175 , 0.1625, 0.225 ,
        0.1625, 0.1875, 0.2125, 0.2375, 0.2125, 0.275 , 0.3125, 0.3875,
        0.325 , 0.325 , 0.3125, 0.35  , 0.3   , 0.375 , 0.375 , 0.3   ,
        0.325 , 0.4   , 0.3   , 0.4375, 0.4375, 0.325 , 0.375 , 0.475 ,
        0.5375, 0.4875, 0.425 , 0.425 , 0.55  , 0.575 , 0.575 , 0.6   ,
        0.5375, 0.6375]),
 array([0.    , 0.0125, 0.075 , 0.025 , 0.0625, 0.0375, 0.125 , 0.0625,
        0.1375, 0.1375, 0.0875, 0.15  , 0.125 , 0.175 , 0.1875, 0.1125,
        0.1875, 0.225 , 0.2   , 0.1875, 0.2125, 0.175 , 0.3   , 0.3625,
        0.325 , 0.3   , 0.25  , 0.3625, 0.3375, 0.3875, 0.3875, 0.325 ,
        0.35  , 0.375 , 0.325 , 0.3875, 0.4375, 0.4625, 0.525 , 0.4875,
        0.5125, 0.5   , 0.525 , 0.4625, 0.5875, 0.525 , 0.3625, 0.575 ,
        0.5125, 0.6875]),
 array([0.    , 0.025 , 0.0625, 0.0125, 0.0375, 0.0375, 0.0625, 0.1   ,
        0.0875, 0.0875, 0.1125, 0.15  , 0.1625, 0.1625, 0.1125, 0.1375,
        0.1875, 0.1875, 0.1375, 0.1875, 0.2   , 0.1625, 0.175 , 0.275 ,
        0.25  , 0.225 , 0.2625, 0.2875, 0.2375, 0.325 , 0.3875, 0.2625,
        0.3   , 0.325 , 0.25  , 0.3375, 0.3875, 0.3875, 0.425 , 0.4625,
        0.4   , 0.4125, 0.3875, 0.5125, 0.55  , 0.625 , 0.5125, 0.5125,
        0.575 , 0.625 ])]
```

Figure 6: error rate for each validation set as a function of K, the error is computed as before, for each K: num-wrong-predictions / size-validation-set

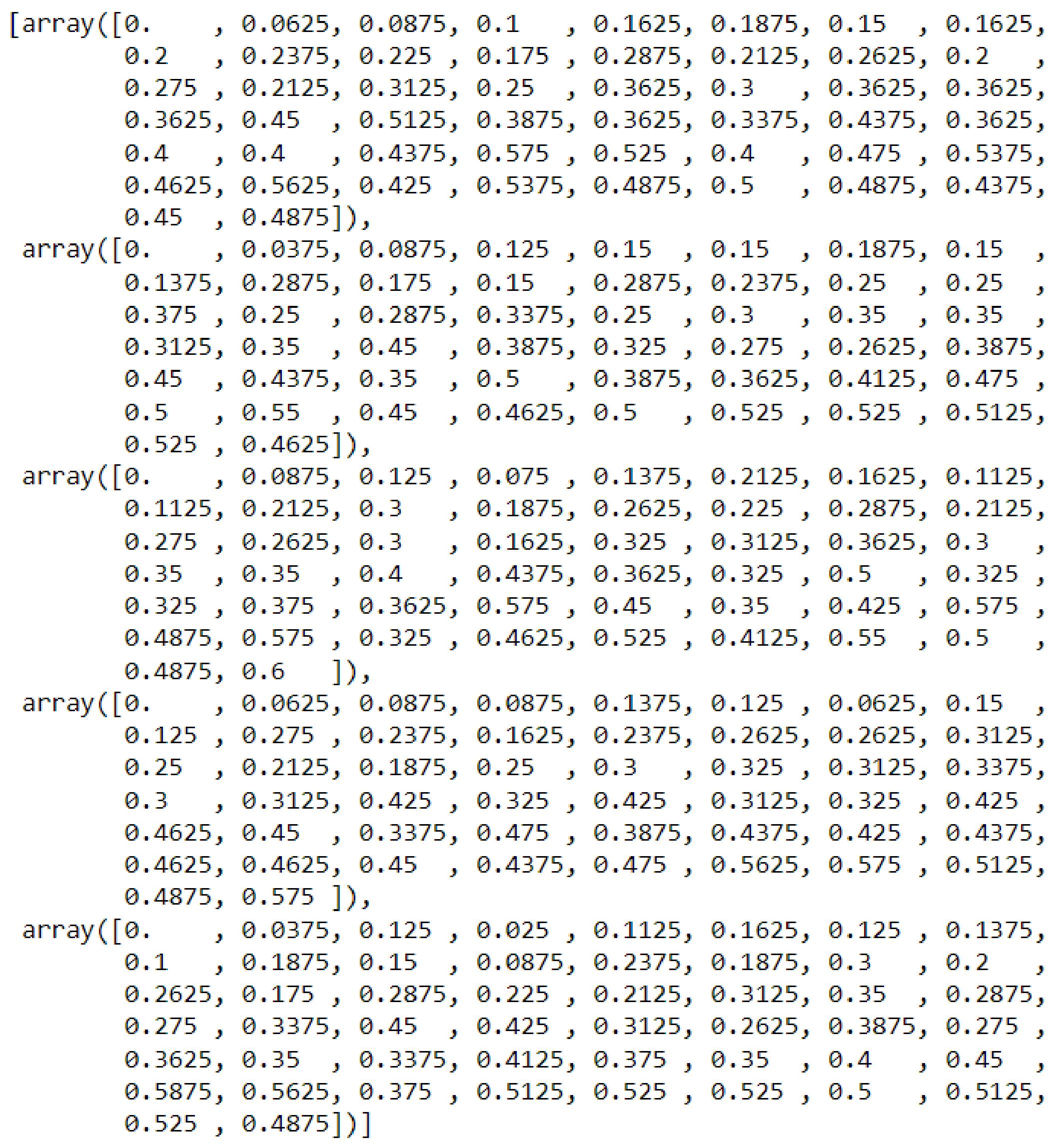
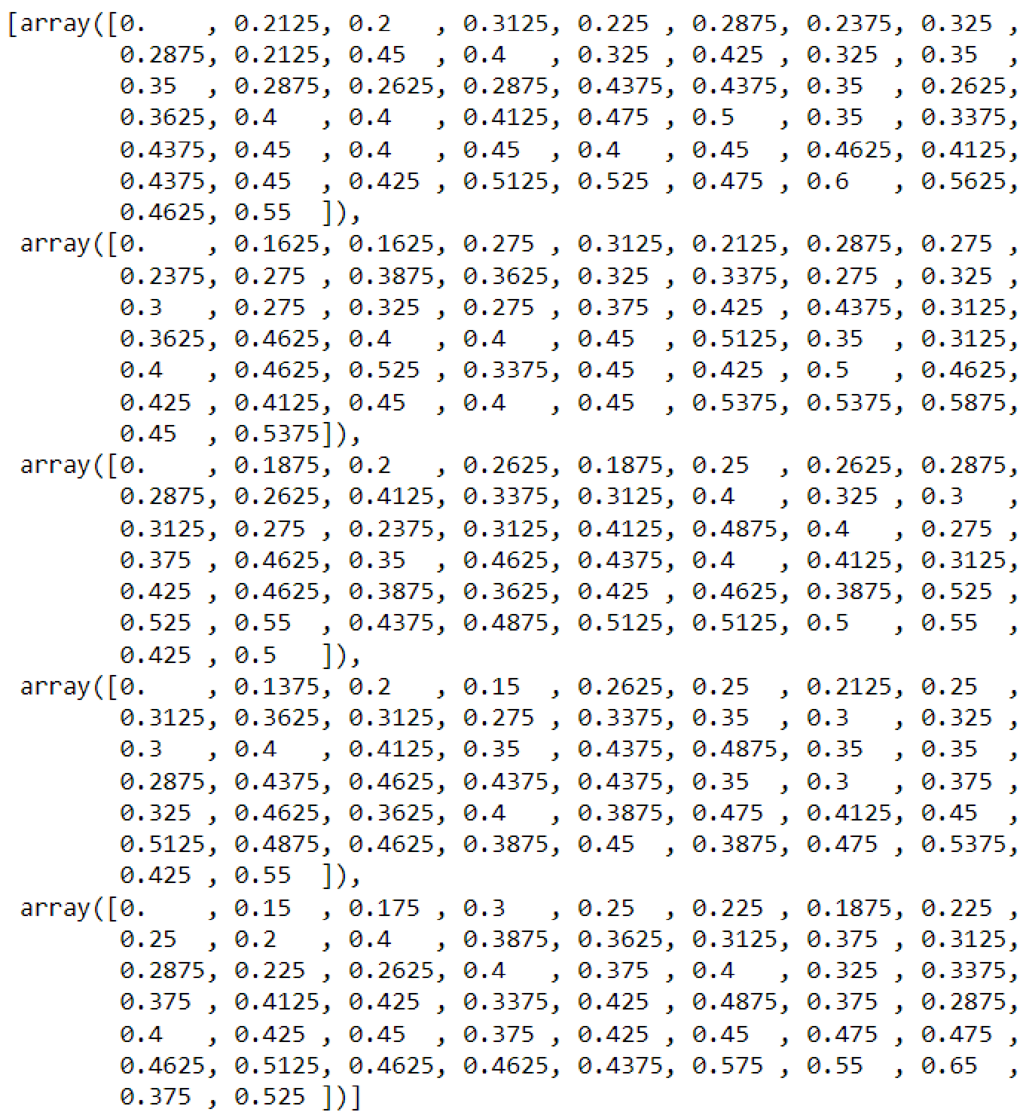
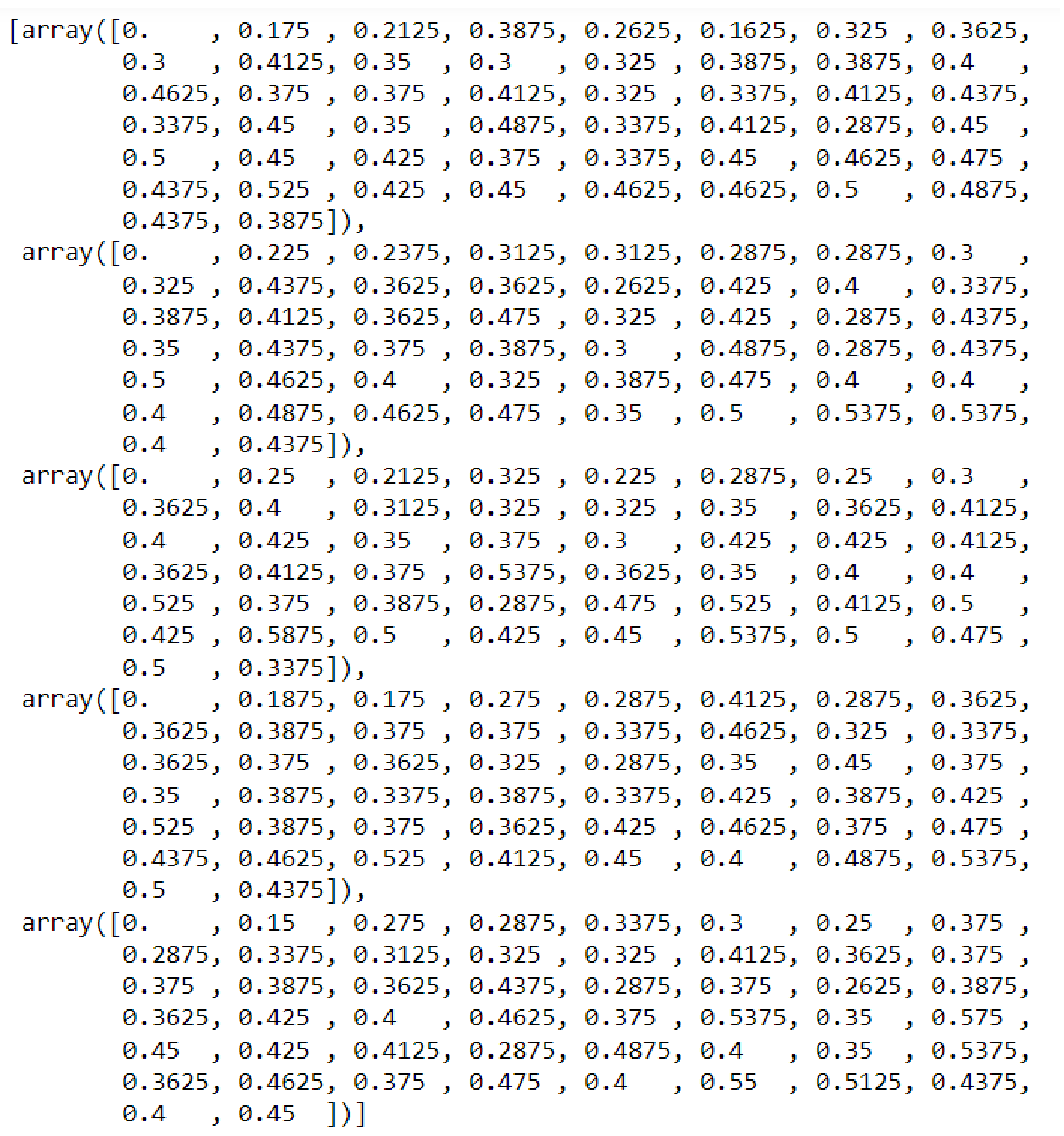

```
[array([0.    , 0.0625, 0.0875, 0.1   , 0.1625, 0.1875, 0.15  , 0.1625,
        0.2   , 0.2375, 0.225 , 0.175 , 0.2875, 0.2125, 0.2625, 0.2   ,
        0.275 , 0.2125, 0.3125, 0.25  , 0.3625, 0.3   , 0.3625, 0.3625,
        0.3625, 0.45  , 0.5125, 0.3875, 0.3625, 0.3375, 0.4375, 0.3625,
        0.4   , 0.4   , 0.4375, 0.575 , 0.525 , 0.4   , 0.475 , 0.5375,
        0.4625, 0.5625, 0.425 , 0.5375, 0.4875, 0.5   , 0.4875, 0.4375,
        0.45  , 0.4875]),
 array([0.    , 0.0375, 0.0875, 0.125 , 0.15  , 0.15  , 0.1875, 0.15  ,
        0.1375, 0.2875, 0.175 , 0.15  , 0.2875, 0.2375, 0.25  , 0.25  ,
        0.375 , 0.25  , 0.2875, 0.3375, 0.25  , 0.3   , 0.35  , 0.35  ,
        0.3125, 0.35  , 0.45  , 0.3875, 0.325 , 0.275 , 0.2625, 0.3875,
        0.45  , 0.4375, 0.35  , 0.5   , 0.3875, 0.3625, 0.4125, 0.475 ,
        0.5   , 0.55  , 0.45  , 0.4625, 0.5   , 0.525 , 0.525 , 0.5125,
        0.525 , 0.4625]),
 array([0.    , 0.0875, 0.125 , 0.075 , 0.1375, 0.2125, 0.1625, 0.1125,
        0.1125, 0.2125, 0.3   , 0.1875, 0.2625, 0.225 , 0.2875, 0.2125,
        0.275 , 0.2625, 0.3   , 0.1625, 0.325 , 0.3125, 0.3625, 0.3   ,
        0.35  , 0.35  , 0.4   , 0.4375, 0.3625, 0.325 , 0.5   , 0.325 ,
        0.325 , 0.375 , 0.3625, 0.575 , 0.45  , 0.35  , 0.425 , 0.575 ,
        0.4875, 0.575 , 0.325 , 0.4625, 0.525 , 0.4125, 0.55  , 0.5   ,
        0.4875, 0.6   ]),
 array([0.    , 0.0625, 0.0875, 0.0875, 0.1375, 0.125 , 0.0625, 0.15  ,
        0.125 , 0.275 , 0.2375, 0.1625, 0.2375, 0.2625, 0.2625, 0.3125,
        0.25  , 0.2125, 0.1875, 0.25  , 0.3   , 0.325 , 0.3125, 0.3375,
        0.3   , 0.3125, 0.425 , 0.325 , 0.425 , 0.3125, 0.325 , 0.425 ,
        0.4625, 0.45  , 0.3375, 0.475 , 0.3875, 0.4375, 0.425 , 0.4375,
        0.4625, 0.4625, 0.45  , 0.4375, 0.475 , 0.5625, 0.575 , 0.5125,
        0.4875, 0.575 ]),
 array([0.    , 0.0375, 0.125 , 0.025 , 0.1125, 0.1625, 0.125 , 0.1375,
        0.1   , 0.1875, 0.15  , 0.0875, 0.2375, 0.1875, 0.3   , 0.2   ,
        0.2625, 0.175 , 0.2875, 0.225 , 0.2125, 0.3125, 0.35  , 0.2875,
        0.275 , 0.3375, 0.45  , 0.425 , 0.3125, 0.2625, 0.3875, 0.275 ,
        0.3625, 0.35  , 0.3375, 0.4125, 0.375 , 0.35  , 0.4   , 0.45  ,
        0.5875, 0.5625, 0.375 , 0.5125, 0.525 , 0.525 , 0.5   , 0.5125,
        0.525 , 0.4875])]
```

Figure 7: LIGHT CORRUPTION, error rate for each validation set as a function of K, the error is computed as before, for each K: num-wrong-predictions / size-validation-set



```
[array([0.    , 0.2125, 0.2   , 0.3125, 0.225 , 0.2875, 0.2375, 0.325 ,
        0.2875, 0.2125, 0.45  , 0.4   , 0.325 , 0.425 , 0.325 , 0.35  ,
        0.35  , 0.2875, 0.2625, 0.2875, 0.4375, 0.4375, 0.35  , 0.2625,
        0.3625, 0.4   , 0.4   , 0.4125, 0.475 , 0.5   , 0.35  , 0.3375,
        0.4375, 0.45  , 0.4   , 0.45  , 0.4   , 0.45  , 0.4625, 0.4125,
        0.4375, 0.45  , 0.425 , 0.5125, 0.525 , 0.475 , 0.6   , 0.5625,
        0.4625, 0.55  ]),
 array([0.    , 0.1625, 0.1625, 0.275 , 0.3125, 0.2125, 0.2875, 0.275 ,
        0.2375, 0.275 , 0.3875, 0.3625, 0.325 , 0.3375, 0.275 , 0.325 ,
        0.3   , 0.275 , 0.325 , 0.275 , 0.375 , 0.425 , 0.4375, 0.3125,
        0.3625, 0.4625, 0.4   , 0.4   , 0.45  , 0.5125, 0.35  , 0.3125,
        0.4   , 0.4625, 0.525 , 0.3375, 0.45  , 0.425 , 0.5   , 0.4625,
        0.425 , 0.4125, 0.45  , 0.4   , 0.45  , 0.5375, 0.5375, 0.5875,
        0.45  , 0.5375]),
 array([0.    , 0.1875, 0.2   , 0.2625, 0.1875, 0.25  , 0.2625, 0.2875,
        0.2875, 0.2625, 0.4125, 0.3375, 0.3125, 0.4   , 0.325 , 0.3   ,
        0.3125, 0.275 , 0.2375, 0.3125, 0.4125, 0.4875, 0.4   , 0.275 ,
        0.375 , 0.4625, 0.35  , 0.4625, 0.4375, 0.4   , 0.4125, 0.3125,
        0.425 , 0.4625, 0.3875, 0.3625, 0.425 , 0.4625, 0.3875, 0.525 ,
        0.525 , 0.55  , 0.4375, 0.4875, 0.5125, 0.5125, 0.5   , 0.55  ,
        0.425 , 0.5   ]),
 array([0.    , 0.1375, 0.2   , 0.15  , 0.2625, 0.25  , 0.2125, 0.25  ,
        0.3125, 0.3625, 0.3125, 0.275 , 0.3375, 0.35  , 0.3   , 0.325 ,
        0.3   , 0.4   , 0.4125, 0.35  , 0.4375, 0.4875, 0.35  , 0.35  ,
        0.2875, 0.4375, 0.4625, 0.4375, 0.4375, 0.35  , 0.3   , 0.375 ,
        0.325 , 0.4625, 0.3625, 0.4   , 0.3875, 0.475 , 0.4125, 0.45  ,
        0.5125, 0.4875, 0.4625, 0.3875, 0.45  , 0.3875, 0.475 , 0.5375,
        0.425 , 0.55  ]),
 array([0.    , 0.15  , 0.175 , 0.3   , 0.25  , 0.225 , 0.1875, 0.225 ,
        0.25  , 0.2   , 0.4   , 0.3875, 0.3625, 0.3125, 0.375 , 0.3125,
        0.2875, 0.225 , 0.2625, 0.4   , 0.375 , 0.4   , 0.325 , 0.3375,
        0.375 , 0.4125, 0.425 , 0.3375, 0.425 , 0.4875, 0.375 , 0.2875,
        0.4   , 0.425 , 0.45  , 0.375 , 0.425 , 0.45  , 0.475 , 0.475 ,
        0.4625, 0.5125, 0.4625, 0.4625, 0.4375, 0.575 , 0.55  , 0.65  ,
        0.375 , 0.525 ])]
```

Figure 8: MODERATE CORRUPTION, error rate for each validation set as a function of K, the error is computed as before, for each K: num-wrong-predictions / size-validation-set



```
[array([0.    , 0.175 , 0.2125, 0.3875, 0.2625, 0.1625, 0.325 , 0.3625,
        0.3   , 0.4125, 0.35  , 0.3   , 0.325 , 0.3875, 0.3875, 0.4   ,
        0.4625, 0.375 , 0.375 , 0.4125, 0.325 , 0.3375, 0.4125, 0.4375,
        0.3375, 0.45  , 0.35  , 0.4875, 0.3375, 0.4125, 0.2875, 0.45  ,
        0.5   , 0.45  , 0.425 , 0.375 , 0.3375, 0.45  , 0.4625, 0.475 ,
        0.4375, 0.525 , 0.425 , 0.45  , 0.4625, 0.4625, 0.5   , 0.4875,
        0.4375, 0.3875]),
 array([0.    , 0.225 , 0.2375, 0.3125, 0.3125, 0.2875, 0.2875, 0.3   ,
        0.325 , 0.4375, 0.3625, 0.3625, 0.2625, 0.425 , 0.4   , 0.3375,
        0.3875, 0.4125, 0.3625, 0.475 , 0.325 , 0.425 , 0.2875, 0.4375,
        0.35  , 0.4375, 0.375 , 0.3875, 0.3   , 0.4875, 0.2875, 0.4375,
        0.5   , 0.4625, 0.4   , 0.325 , 0.3875, 0.475 , 0.4   , 0.4   ,
        0.4   , 0.4875, 0.4625, 0.475 , 0.35  , 0.5   , 0.5375, 0.5375,
        0.4   , 0.4375]),
 array([0.    , 0.25  , 0.2125, 0.325 , 0.225 , 0.2875, 0.25  , 0.3   ,
        0.3625, 0.4   , 0.3125, 0.325 , 0.325 , 0.35  , 0.3625, 0.4125,
        0.4   , 0.425 , 0.35  , 0.375 , 0.3   , 0.425 , 0.425 , 0.4125,
        0.3625, 0.4125, 0.375 , 0.5375, 0.3625, 0.35  , 0.4   , 0.4   ,
        0.525 , 0.375 , 0.3875, 0.2875, 0.475 , 0.525 , 0.4125, 0.5   ,
        0.425 , 0.5875, 0.5   , 0.425 , 0.45  , 0.5375, 0.5   , 0.475 ,
        0.5   , 0.3375]),
 array([0.    , 0.1875, 0.175 , 0.275 , 0.2875, 0.4125, 0.2875, 0.3625,
        0.3625, 0.3875, 0.375 , 0.375 , 0.3375, 0.4625, 0.325 , 0.3375,
        0.3625, 0.375 , 0.3625, 0.325 , 0.2875, 0.35  , 0.45  , 0.375 ,
        0.35  , 0.3875, 0.3375, 0.3875, 0.3375, 0.425 , 0.3875, 0.425 ,
        0.525 , 0.3875, 0.375 , 0.3625, 0.425 , 0.4625, 0.375 , 0.475 ,
        0.4375, 0.4625, 0.525 , 0.4125, 0.45  , 0.4   , 0.4875, 0.5375,
        0.5   , 0.4375]),
 array([0.    , 0.15  , 0.275 , 0.2875, 0.3375, 0.3   , 0.25  , 0.375 ,
        0.2875, 0.3375, 0.3125, 0.325 , 0.325 , 0.4125, 0.3625, 0.375 ,
        0.375 , 0.3875, 0.3625, 0.4375, 0.2875, 0.375 , 0.2625, 0.3875,
        0.3625, 0.425 , 0.4   , 0.4625, 0.375 , 0.5375, 0.35  , 0.575 ,
        0.45  , 0.425 , 0.4125, 0.2875, 0.4875, 0.4   , 0.35  , 0.5375,
        0.3625, 0.4625, 0.375 , 0.475 , 0.4   , 0.55  , 0.5125, 0.4375,
        0.4   , 0.45  ])]
```

Figure 9: HEAVY CORRUPTION, error rate for each validation set as a function of K, the error is computed as before, for each K: num-wrong-predictions / size-validation-set

• Discuss how corruption magnitude influences the prediction accuracy of K-NN and the optimal value of K.

The higher the corruption, the lower tends to be the accuracy, with high corruption the algorithm tends to have worse overall performances compared to the inexistent corruption data.
With higher corruption the algorithm tend to have a balanced accuracy as K increases, I mean that after some K the algorithm doesn't really become worse as K increases, the accuracy values remain stable.
The optimal value of K tends still to be with a low value of K, however with low presence of corruption we can still find low amount of errors as K increase, for example, k = 9 in the first validation set of the first image of this second task.

In the high corruption data, the KNN struggle to find a good K for generalization and the errors for predictions tend to increase drastically as K increase and then start to stabilize as K increases even more

3 Linear Regression (45 points)

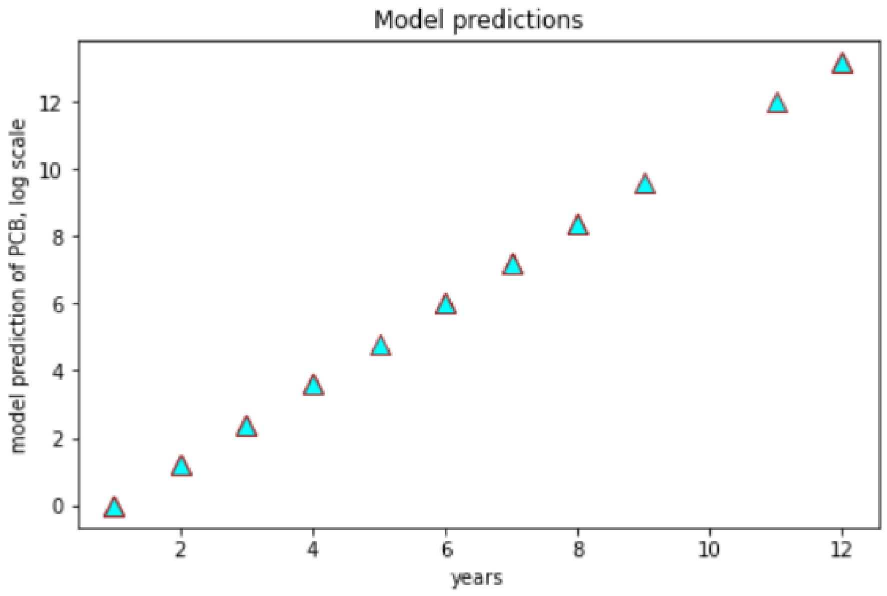
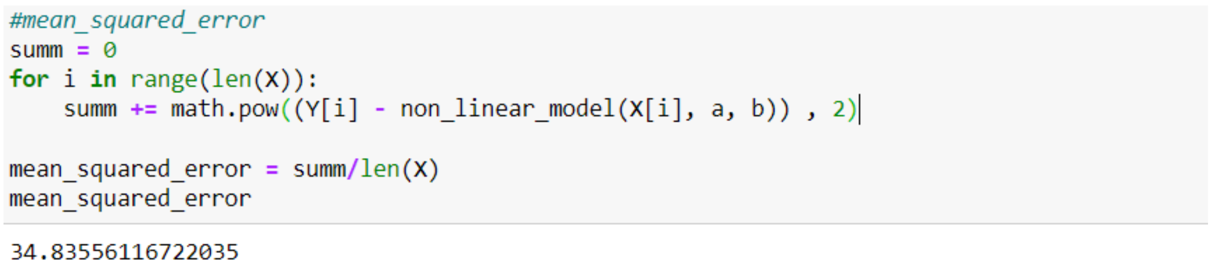


Figure 10: age vs model output, age vs log of PCB

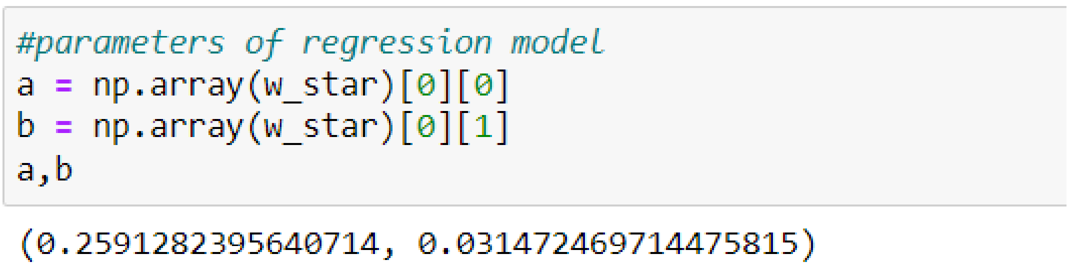


```
#mean_squared_error
summ = 0
for i in range(len(X)):
    summ += math.pow((Y[i] - non_linear_model(X[i], a, b)) , 2)

mean_squared_error = summ/len(X)
mean_squared_error
```

```
34.83556116722035
```

Figure 11: MSE of non linear model



```
#parameters of regression model
a = np.array(w_star)[0][0]
b = np.array(w_star)[0][1]
a,b
```

```
(0.2591282395640714, 0.031472469714475815)
```

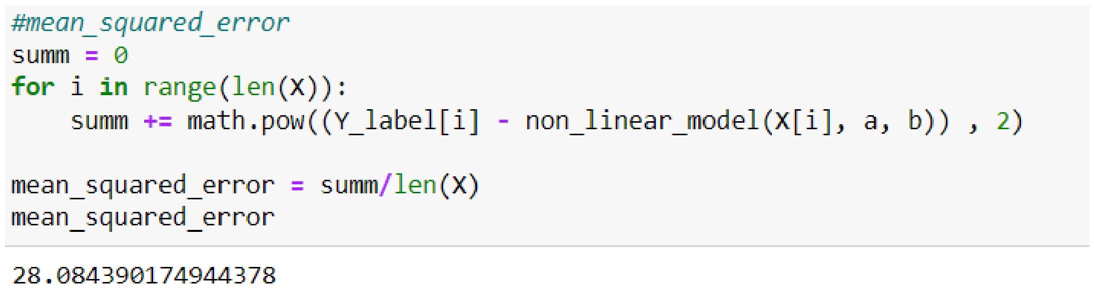
Figure 12: Parameters of regression model

4. Discuss this quantity. What does it mean if R2 is 1 and especially if R2 is 0? Can R2 be negative?

if R2 = 1 it means that the model can perfectly explain the data (the numerator goes to 0), in other words, how much a variable's behavior (independent variable) can explain the behavior of another variable (dependent variable).

if R2 = 0 it means that most probably the regression model needs to be adjusted because through this model the independent variable cannot explain the dependent variable.

R2 it's a statistical measure, and even if the R2 values are high, we have no guarantees that the model once deployed will be perfect, same for low values of R2, we have no guarantees that the model is bad.
R2 have values ranging from 0 to 1, thus it cannot be negative.



```
#mean_squared_error
summ = 0
for i in range(len(X)):
    summ += math.pow((Y_label[i] - non_linear_model(X[i], a, b)) , 2)

mean_squared_error = summ/len(X)
mean_squared_error
```

```
28.084390174944378
```

Figure 13: mean-squared error of model with transformed inputs

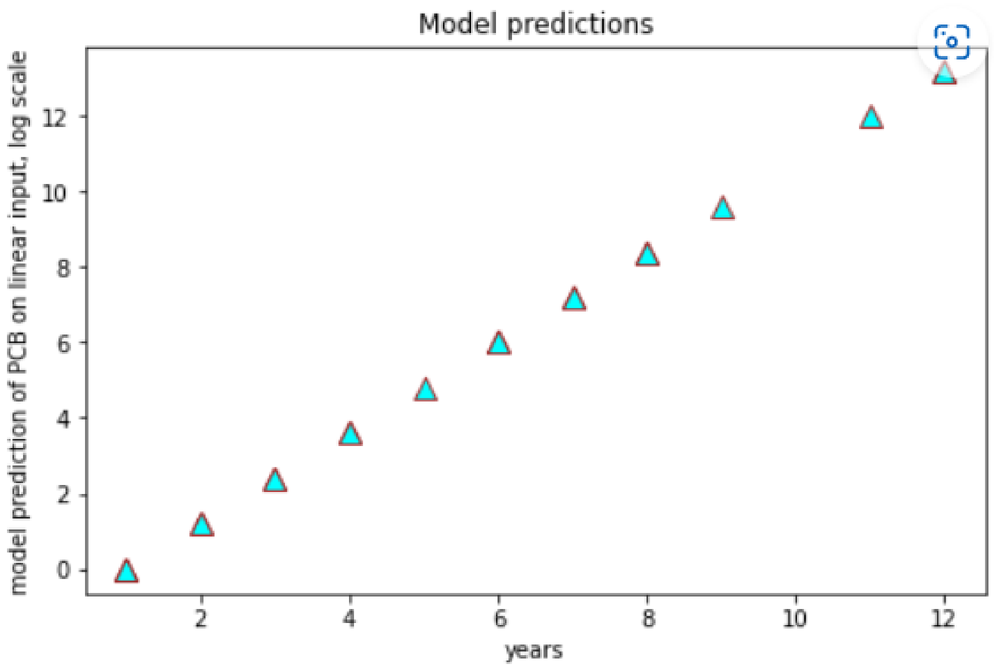


Figure 14: linear scale for years, target log-scale

R^2 of second model: 0.4816250669292409

The result shows an higher R^2 coefficient, this means that the second model should capture better the relation between the independent variable x and the dependent variable y