**Cs 340 Project Report.**

**Introduction:**

This project is split in two parts. Part A aims to analyze and report on COVID-19 vaccination data. Part B aims to create and train a simple Multi-Layer Perceptron (MLP) that can identify whether a hot-encoded number is odd or even, and if it is larger than 3. Updated vaccination data can be downloaded from the [official EU page](https://www.ecdc.europa.eu/en/publications-data/data-covid-19-vaccination-eu-eea) in csv format. The following modules must be installed and updated to the latest version (as of 6/6/2021) for the program to run:

* pandas
* sklearn
* numpy
* matplotlib

However, the program will install them automatically for most systems (Tested on Windows OS. Latest version of python required. Python must be included in PATH.)

**Design requirements:**

**Part A:** Create a tool that will provide the user with a menu interface leading to the following functions:

* **Option 1:** List last reported week’s number of vaccinated Health Care Workers (HCW) per country. Second dose data are compiled and displayed next to their corresponding country in list format.
* **Option 2:** Presents the user with a list of all available countries and asks them to select one. After the selection is made, data for the second vaccination dose corresponding to the “ALL” age group are compiled and displayed, grouped by the name of the vaccine.
* **Option 3:** Presents the user with a list of all available countries and asks them to select one. After the selection is made, an animation is played charting the second dose of vaccines administered to health care workers in relation to the time passed in weeks, for the selected country.
* **Option 4:** Cumulates the second dose administrations for every country and sorts them based on percentage of population vaccinated. Then, the top 10 and bottom 10 countries are charted along with the percentages of vaccinations. Finally, the sorted vaccination percentages are saved in a txt file for all countries.
* **Option 5:** Exit the program.

**Part B:** Create a tool that will provide the user with a menu interface leading to the following functions:

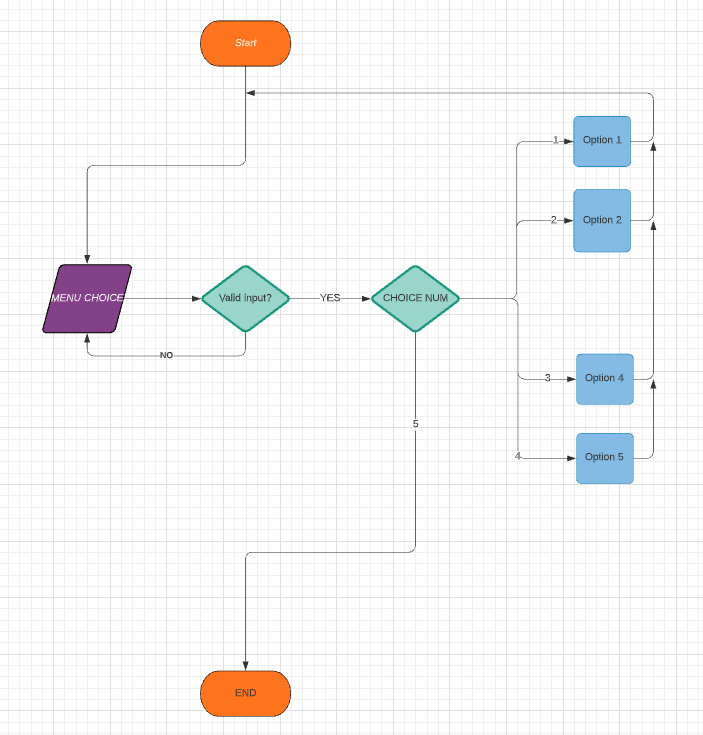
* **Option 1:** Allows the user to enter a custom size for the hidden layer of the MLP. The user can choose to skip this step and the default value of 15 will be used.
* **Option 2:** Prompts the user to enter the name of a data training file and the number of epochs they want the MLP to be trained for, as well as select the activation function (training step) that will be used for training. All these parameters are defaulted, allowing the user to skip entering custom values if they desire. After the parameters are set, the MLP is trained, and the process is recorded in a txt file. The first few lines of the dataset are displayed, along with the confusion matrix and the classification report for the testing performed after the training.
* **Option 3:** The user is asked to provide the name of a testing file, and the MLP is tasked to predict the values for the vectors in that file. The vectors along with the predictions of the MLP are displayed on the screen and written in a text file named training\_output.txt.
* **Option 4:** A simple graph of the error in relation to the epochs of training is created and displayed.
* **Option 5:** Exit the program.

**Commentary:**

* All design requirements were met, along with sophisticated user input error trapping.
* Hidden Layer was limited between 2 and 99 neurons. As explained by Kubat(2018), and as shown in **Figure 1**, the number of hidden neurons relates to the error in parabolic fashion. This means that while the initial increase of hidden neurons will benefit our network by decreasing the error of predictions, further increases can have the opposite result due to overfitting of the data. Already, for this problem, the allowed range of neurons is redundant; however, it was chosen for demonstrative purposes should the user wish to examine this phenomenon.  
    
  Kubat further illustrates a method of calculating the optimal number of neurons algorithmically, by increasing their number in small intervals and stopping when the error plateaus and is no longer improving. However, this application was way outside the scope of this program, so a default value of 15 was chosen after a small amount of experimentation.
* A discussion on overfitting is in order. Overfitting occurs from overtraining as well as from over-presentation of data in a neural network and remains one of the most prevalent issues in machine learning (Aggarwal, 2019).   
  To prevent overfitting due to overtraining, sklearn’s MLPclassifier module has an early termination functionality that stops training when the error is not improving more than a predefined tolerance (defaulted at 1e-4 per 10 epochs) (API Reference — scikit-learn 0.24.2 documentation, 2021).   
  For this problem this occurs at around 500 epochs and thus we can tell that this is the optimal model; however, for demonstrative purposes we have overridden the early termination functionality and allowed the user to way overshoot the default epoch number, allowing it to reach 10.000 epochs. Of course, this number is far redundant, but it demonstrates overfitting perfectly.
* To further avoid data redundancy and overfitting, the user is not allowed to use custom made training and testing sets and is instead provided with randomly created data sets. Due to the simplicity of the problem, this solution does not make too much difference, and the model ends up overfitted most of the time anyway. However, as Liu (2019) demonstrates (**Figure 2**), in more complex problems, randomization of the data while maintaining a moderate number of examples is the optimal solution for data creation.
* This problem is not a well-defined AI problem, as described by Flasiński (2016), since it better fits linear programming due to the unambiguous nature of the data and their labeling.
* The user is allowed to choose between three weight optimization solvers, after he is given a brief description. If they choose the sgt solver, adaptive learning is applied, reducing the learning rate for plateaued error values.

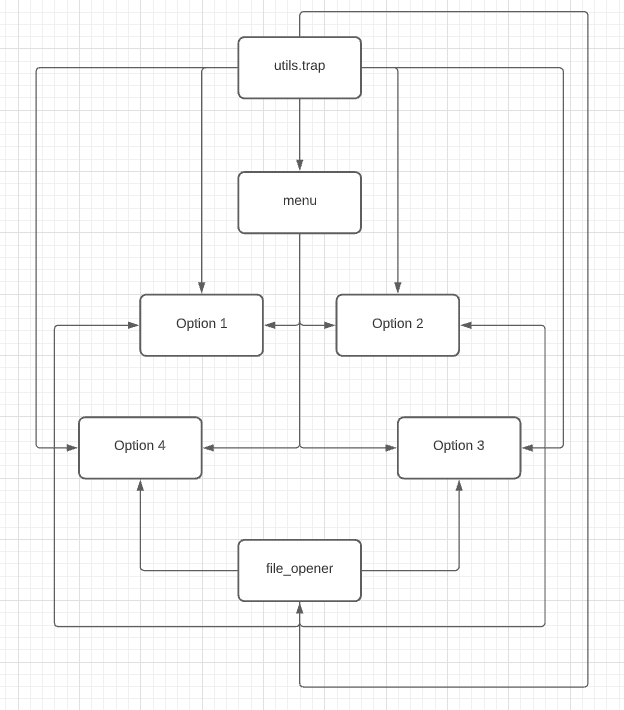
**Flowchart:**

**Note:** Due to the naming of the functions, the top-level flowchart is the same for parts A and B. Going into any more detail would far exceed the nature of a top-level diagram.

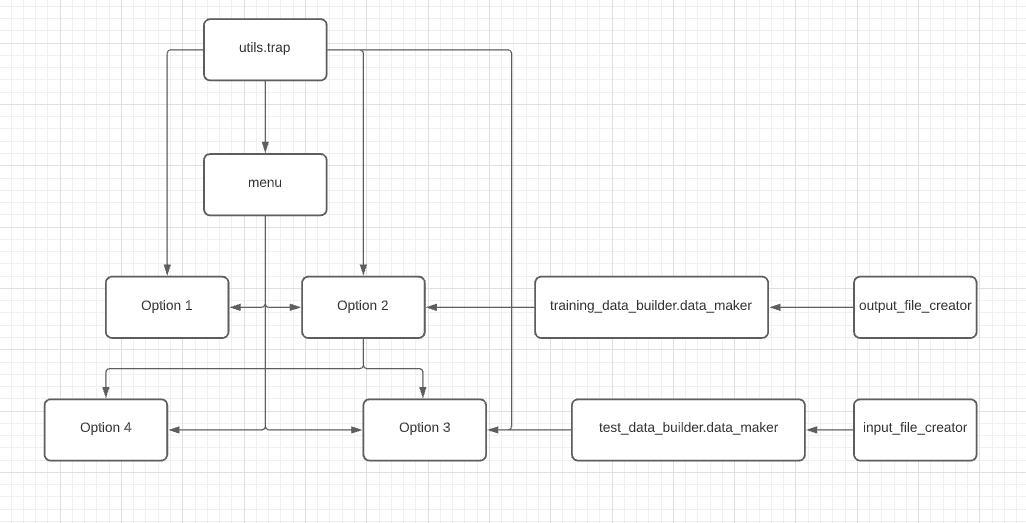


**Entity relationship diagrams:**

**Part A:**

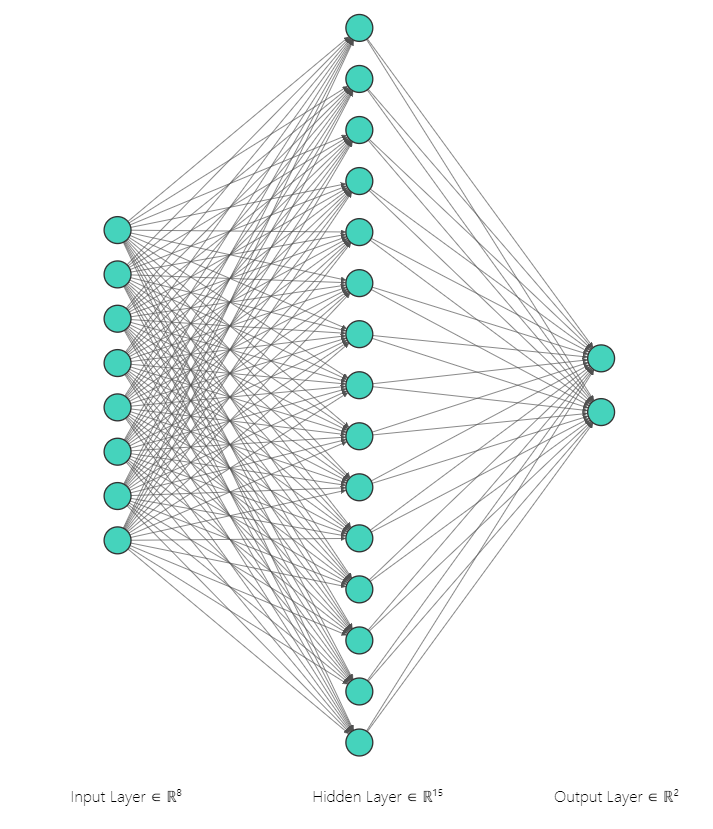


**Part B:**



**ANN Diagram:**

**Input Hidden Output**

****

(NN SVG, 2021)

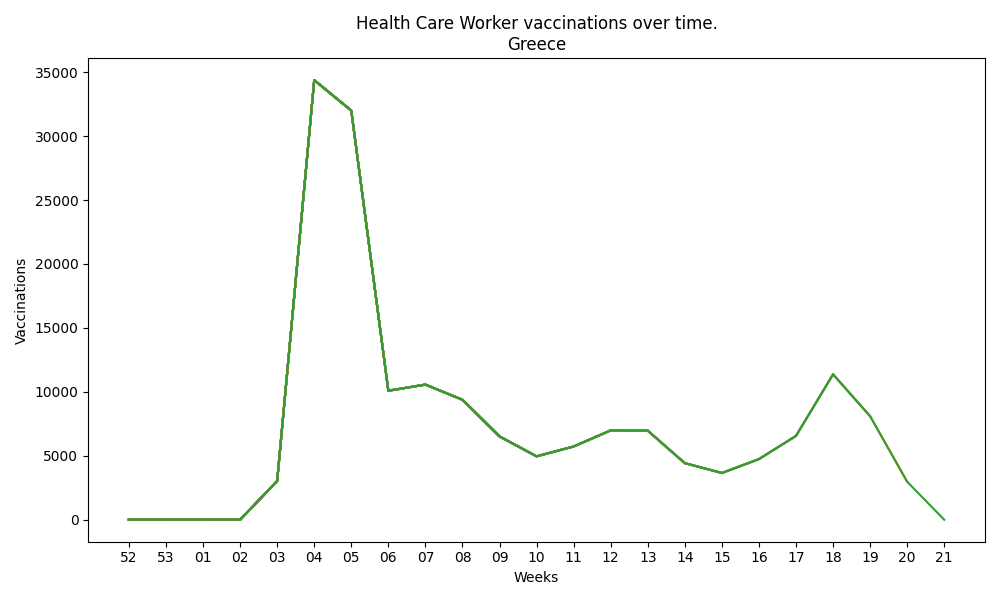
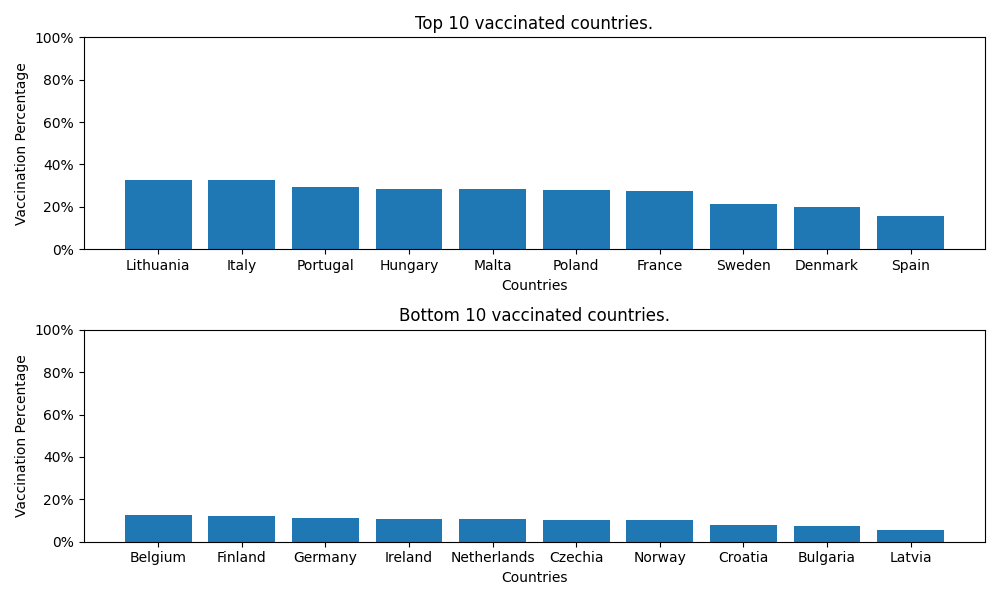
**Sample I/O:**

For part B, the supporting files that are submitted along with the project are made using only the default values (pressing enter without inputting anything in every prompt and not using option 1). Since the data rely on random generation, they will be different every time.

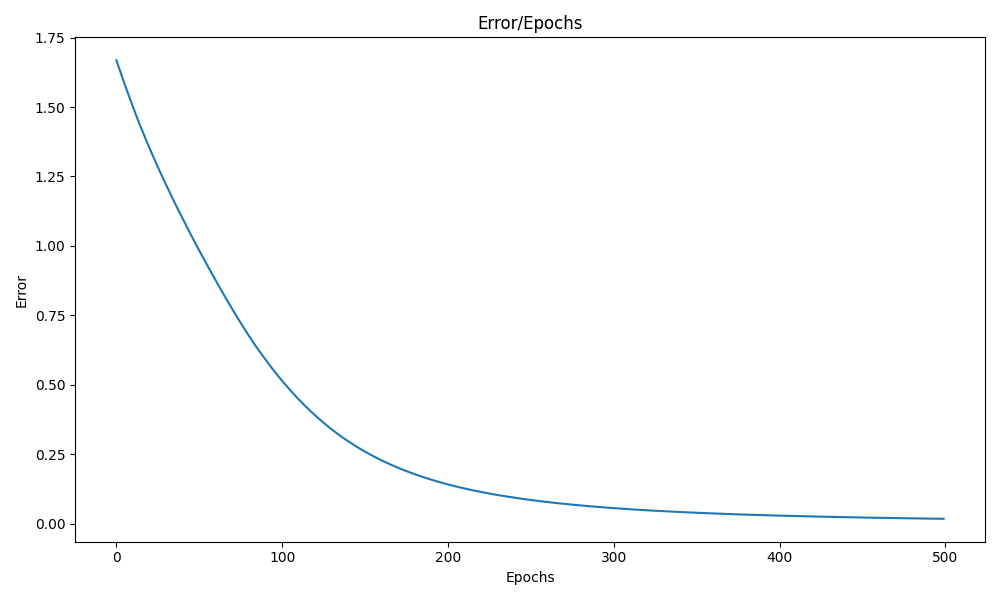
For part A the data set used for testing is included, along with the output files.

**Graph samples:**

**Part A:**



**Part B:**

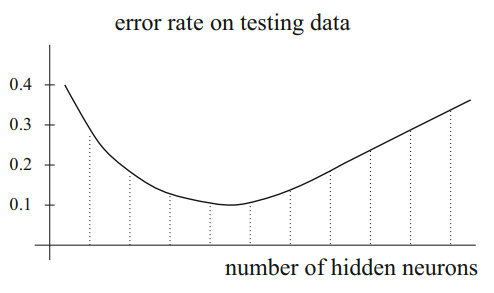


**Conclusion:**

To complete this project, several skills were required. Most programming practices like documentation combing, deductive reasoning, creativity, critical thinking and problem-solving were tested and honed during the process. Furthermore, new skills had to be developed like understanding and usage of graphing libraries, deep understanding and usage of machine learning principals, libraries, and best practices, along with extraordinary amounts of troubleshooting and trial and error. Time management, and patience were the most stressed aspects of this project, and especially the report, along with the need to accumulate very large amounts of information in a very short time span (part B was done in 2 days start to finish including all the research). Finally, the most exciting part of the project was experimenting with all the different parameters, both during the data presentation stages and the MLP training. The New Abnormal by The Strokes was repeated over 100 times during this project, highly recommended.

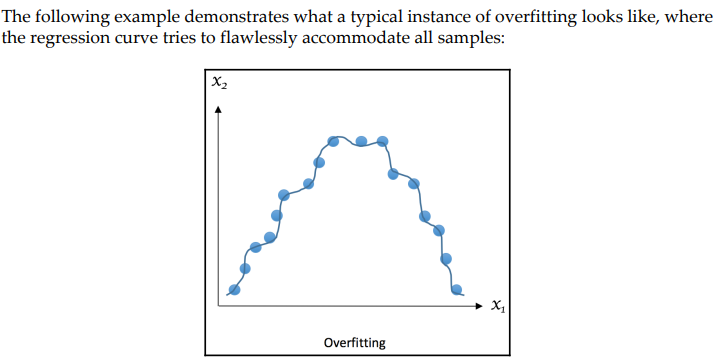
**Appendix:**

**Figure 1:**



Kubat(2018)

**Figure 2:**



Liu, Y. (2019)

**References**

* Aggarwal, C. (2019) *Neural Networks And Deep Learning*, New York, Springer.
* API Reference — scikit-learn 0.24.2 documentation (2021) *Scikit-Learn.Org*, [Online]. Available at https://scikit-learn.org/stable/modules/classes.html#module-sklearn.neural\_network (Accessed 6 June 2021).
* Flasiński, M. (2016) *Introduction To Artificial Intelligence*,.
* KUBAT, M. (2018) *INTRODUCTION TO MACHINE LEARNING*, [Place of publication not identified], SPRINGER INTERNATIONAL PU.
* Liu, Y. (2019) *Python Machine Learning By Example*, 2nd ed. Birmingham, UK, Packt Publishing.
* NN SVG (2021) *Alexlenail.Me*, [Online]. Available at http://alexlenail.me/NN-SVG/index.html (Accessed 6 June 2021).
* Russell, S. and Norvig, P. (2020) *Artificial Intelligence: A Modern Approach*, 4th ed. London, Pearson.