CSC 480 –Artificial Intelligence I
2024-25 – Spring Term
Instructor: Kenny Davila Castellanos

Homework #4: Machine Learning and Binary Classification

Objective

This homework has multiple objectives. First, to learn how to use different classification algorithms studied in class: **Decision Tree classifier**, **Random Forest Classifier** and **Logistic Regressor Classifier**. The second objective is to get familiar with Python libraries such as **Numpy** and **Scikit-Learn**. The goal is to learn about the functions that these libraries provide for processing data, model training, and model evaluation. The third objective is to learn how to use different hypermeters that these algorithms have.

Summary

In this work, you will be using the Scikit-learn Python library to train and evaluate the performance of three classification algorithms: Decision Trees, Random Forests and Logistic Regression. To train and evaluate them, you will be provided synthetic data from a toy classification problem. You will then analyze the classification results for each algorithm using different configurations. Finally, you will have to elaborate a written report where you present and analyze your results.

Problem Description

In this homework, you will develop a system to predict the programming language that a software developer will use for a given project. In real life, this is a decision that typically depends on many factors, some related to personal preference and others are based on project requirements. You are given a (synthetic) dataset with (simulated) examples of cases where a programmer decided to use either of these languages to complete a project. Each case is described using a set of attributes which might or might not have a real impact on the choice made by the programmer. You will use this dataset to train models with multiple machine learning algorithms, and you will use these models later to predict the preferences of new programmers given the same set of attributes.

Attributes

The successful application of machine learning algorithms heavily relies on the selection of the right of attributes (or features). Here, you are given the following (synthetic) information about each programmer:

1. **Simple Syntax**. Indicates if the developer prefers programming languages that have a simple syntax. For example, the developer does not like using special symbols such as semicolon or curly brackets.
2. **Indentation**. Indicates if the programmer likes to keep the code properly indented to make it more readable for others.
3. **Performance.** Indicates if the target program has high-performance requirements.

4. **Experience.** A measure of the coding experience (in years) that this programmer has.
5. **Coffee.** Indicates if the programmer drinks large amounts of coffee while coding.
6. **Desktop.** Indicates if the program must run as a desktop application.
7. **Limited Sleep.** Indicates if the programmer loves coding at night, and quite frequently does not sleep enough because of this.
8. **Collaborative**. Indicates if the program will be developed by a team of at least 10 people.
9. **Comments**. Average number of comment lines that this programmer writes for every 100 lines of code.
10. **Video Game.** Indicates if the program to develop is a video game or not.
11. **Functional Programming**. Indicates if the programmer loves using the functional programming paradigm.
12. **Graphical App.** Indicates if the program must be very graphical or visually attractive.
13. **OOP**. Indicates if the programmer loves using the object-oriented programming paradigm.
14. **LTS**. Indicates if this is a program that requires long-term support (at least 5 years).
15. **Comparative Ratio**. Indicates how the salary of this programmer compares to the market. This is ([worker salary for this job] / [market median salary for this kind of job]).
16. **Gamer**. It tells if the programmer is a person who enjoys playing video games.
17. **Multiplatform**. Indicates if the program must run in different platforms (e.g. Windows, Linux, Mac OS, etc.).
18. **Online Documentation**. Indicates if the programmer frequently checks the official documentation of the programming language when coding.

Note that most of these attributes are binary (they can only be “yes” or “no”), while a few others are continuous values. The algorithms considered here work with numerical values only. As such, you must convert categorical values into numbers (“yes” = 1, “no” = 0). Also, the dataset is provided to you as a file in Comma-separated Value (CSV) format. Text files like this one are easy to read and manipulate using the built-in python library. However, feel free to explore other libraries such as CSV or Pandas.

The program (70%)

In this assignment, you will have to consider 4 different processes: cross validation, grid search, general training, and testing. Each of these processes will have its own independent executable file. However, there are many functions that are needed across multiple processes, and it makes sense to separate these functions so that they can be easily called by the processes that need them. In the following, we will first talk about the general (auxiliary) functions, and then will talk about each main process.

Part 0. Auxiliary functions

We can divide the auxiliary functions in three main groups: configuration related functions, dataset related functions, and training related functions. It is important to understand what the inputs and outputs for each of these functions are. These are described below.

Configuration Related Functions. This work requires you to experiment with different hyperparameters. This time, we will not hardcode these values and we will not provide them to our program one by one using standard input either. Instead, we will follow the standard practices in machine learning, and we will read the hyperparameters from external **configuration files**. The idea main goal is to reduce the risk of errors when repeating experiments. Also, by storing the configuration in a single external

file, we make it easy to modify these without changing the code. It also allows us to create copies of configurations that work well to put them aside while we experiment with other configurations. Figure 1 illustrates this idea.

In our program, we will have a function **"load_hyperparameters"**, which takes the configuration file name and the name of one of the three processes ("cross-validation", "training", "grid-search"). The function loads the entire configuration from the file and returns the subset of the configuration that applies to the given **process** and **active classifier** (also specified in this config). **If you want to run your program using a different classifier, you must change the value for "active_classifier"** in your config file. Change it to name of the desired classifier: **"decision_tree"**, **"random_forest"** or **"logistic_classifier"**.

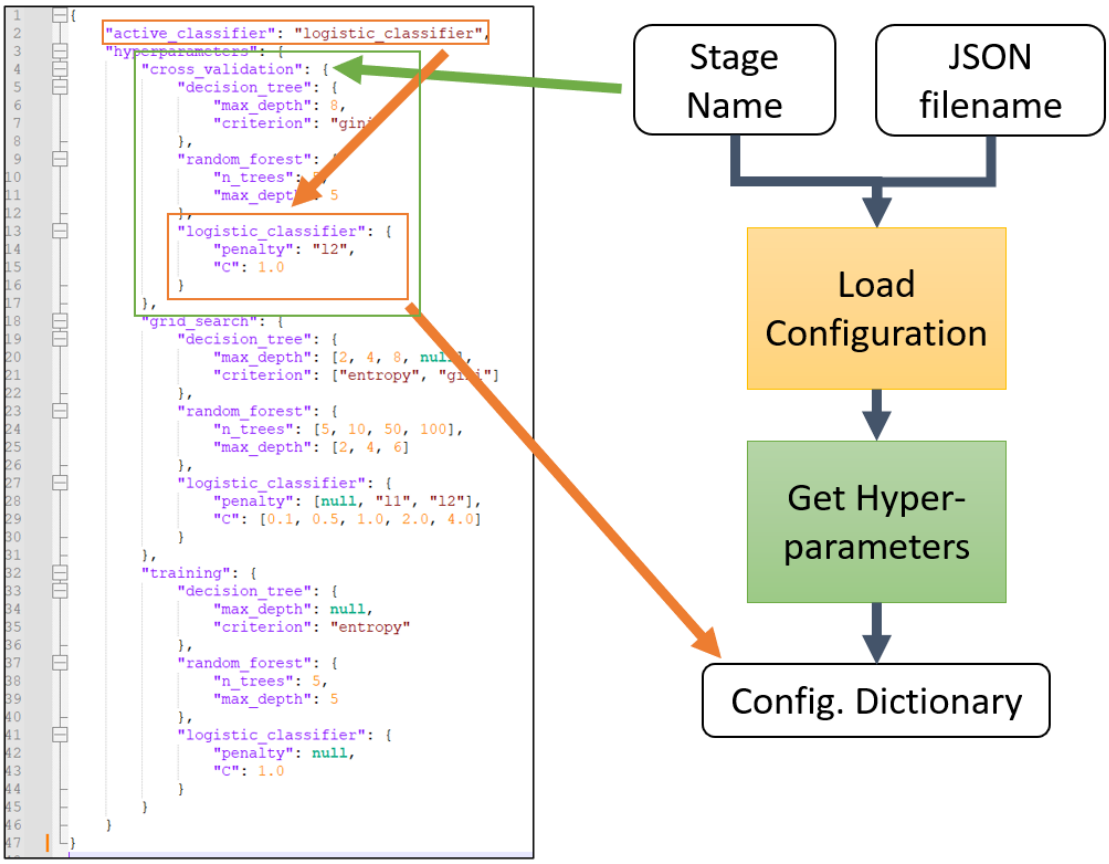

Figure 1. Configuration file with hyperparameters. This shows how the configuration contains the "active_classifier" field to store which classifier should be used. The load_hyperparameters function also receives the stage ("cross_validation", "training" or "grid_search"), which should be used to pick and return the hyperparameters for the active classifier for the given stage. In this example, it will select the hypermeters for a **logistic classifier** for **cross-validation**.

Dataset Related Functions. In all stages, the program will need to load one of the provided dataset files. As mentioned before, these files are in CSV format. The first function is **"load_raw_dataset"**, which takes as input the file name and produces as output a Tuple (X, Y), where X and Y are numpy arrays with the raw data. Here, X is a 2D numpy array where every row represents an example in the dataset, and every column represents the values of a given attribute. For example, X[10, 2] would refer to the third

attribute (0-based indexing) of the 11-th example in the dataset. Meanwhile, Y is a 1D numpy array containing all labels loaded from the file. The dtype of X should be a **floating-point** type, while Y can have a string-based type. This means that all labels can remain as text before putting them in the numpy array, but the attributes need to be converted to numbers using the **label encoding** techniques studied in class.

After loading the dataset, the next step is to **normalize** it. This is not performed at the same time of loading because the normalization process depends on parameters that are learned from a training dataset. That is, **all training splits will be normalized using their own parameters, but the validation and testing splits need to be normalized with the same parameters of the training dataset**. This is particularly tricky when running cross-validation because here training and testing splits alternate, meaning that the normalization parameters need to be learned independently for each combination of training and testing splits. The **"apply_normalization"** function will be used for normalization. This function takes as input a raw dataset (numpy array) and either a trained normalized (**StandardScaler** from Scikit-learn) or None. If the function receives None (e.g., when normalizing training data), then it should create a new StandardScaler, and fit it using the dataset and then transform the dataset with it (check the fit_transform function). If the function receives a StandardScaler, it should use it to normalize the given dataset. In both cases, the function should return the StandardScaler and the normalized dataset.

Finally, the **"split_dataset"** function takes as input a raw dataset (both x and y), and a value N and creates and returns a List with N tuples of the form (split_X, split_Y). The given datasets are initially sorted by class. To ensure that after splitting each portion keeps the same ratios for each class, it is important to **shuffle the rows** before creating these partitions. At the same time, it is important to make sure that **the same shuffling is applied to both X and Y**, or the resulting splits will have mislabeled examples. This can be achieved by shuffling an independent array of indices, and then using the same shuffled array to access the rows of X and Y. Afterwards, the algorithm should compute equally sized splits. Note that it is possible that the number of examples is not divisible by the given number of splits. In that case, the sizes should be distributed so that the difference between the largest and the smallest split is just 1. For example, to create 5 splits with 42 examples, the sizes should be 9, 9, 8, 8, and 8.

Training Related Functions. During the cross-validation and training stages, the program needs to train a model of the selected type. Again, the configuration determines what type of classifier should be trained by the program. Only three types are considered in this assignment: **Decision Tree Classifier**, **Random Forest Classifier**, and **Logistic Regressor**. Note that in Scikit-learn, different types of classifiers are "compatible" at some general level since they support common functions such as "fit" and "predict". The **"train_classifier"** function takes as input the name of the active classifier, the corresponding hyperparameters and a dataset (both X and Y), and it must create a classifier of the specified type with the given hyperparameters, and it must train this classifier using the provided dataset. The function finally returns the trained classifier.

Part 1. Cross-validation

The cross-validation process can be really helpful when evaluating the performance of different machine learning models on limited amounts of data. The first part of this assignment will be to build a program (Python script: **part_01_cross_validation.py**) which runs cross-validation for a given dataset. The script takes the following command line arguments: **in_config**, which refers to the name of the configuration file in JSON format; **in_raw_data**, which refers to the dataset filename; and **n_folds**, which

refers to the number of partitions to use in the cross-validation process. Note that the cross-validation process itself is part of the next stage, grid search. The cross-validation process is illustrated in Figure 2.

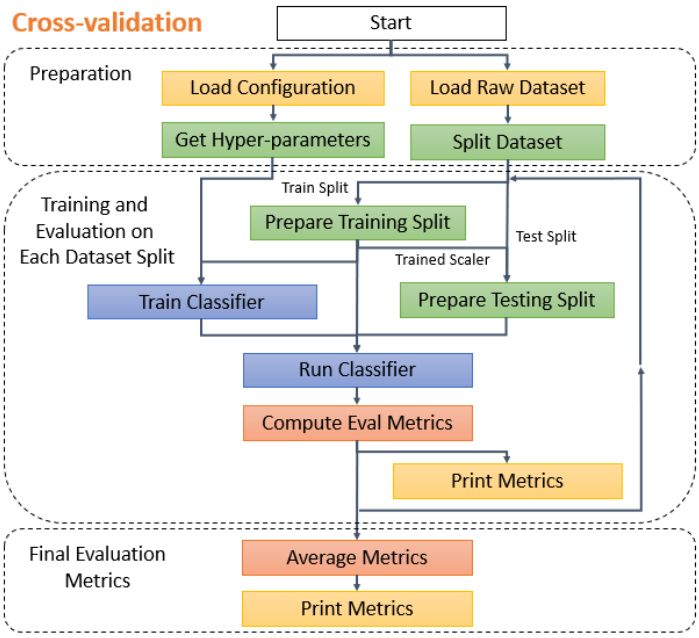

Figure 2. Cross-validation Process. It can be coarsely divided into three stages: preparation, training and evaluation per data split, and final evaluation based on averaged metrics.

Part 2. Grid Search

The second stage is the grid search process that evaluates combinations of hyperparameters to identify the configuration that leads to the best performance. This process is combined with cross-validation to obtain more robust evaluations. This idea is illustrated in **Figure 3**. The command line arguments for this python script (**part_02_grid_search.py**) are the same ones used for cross-validation.

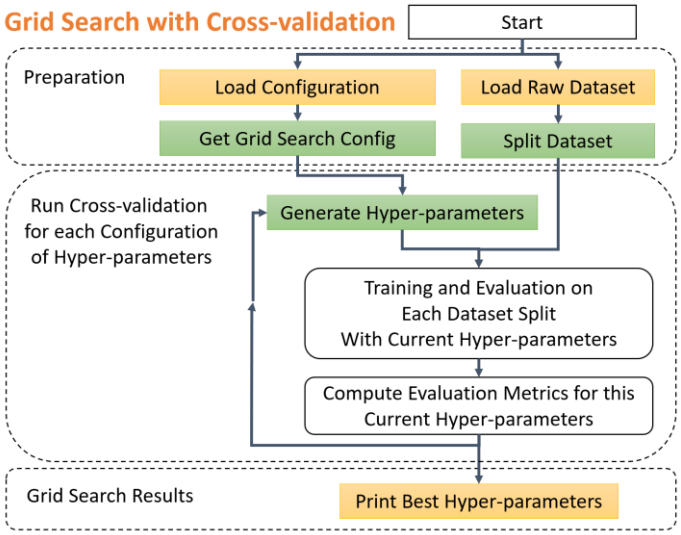

Figure 3. Grid Search with Cross-validation. Combinations of hyperparameters are exhaustively generated and evaluated using Cross-validation.

Decision Tree Classifier. You are required to use different **training datasets** ("training_data_" + [small/medium/large/very_large] + ".csv") to train different decision tree classifiers using combinations of the available **criteria** ("Gini" or "Entropy") and **maximum depths** (2, 4, 8, and None). You are expected to systematically evaluate at least 32 configurations (4 **datasets** x 2 **criteria** x 4 **max depths**). Other parameters such as "splitter", "max_features", etc., should keep their default values. Using cross-validation, you must report the following metrics for each decision tree: Accuracy (training and validation), Average Recall (validation), Average Precision (validation), average F-1 (Validation), and total time (training and validation). In the final report, you must use a single table to report all these results. Represent the values for the metrics (and times) using **4 significant figures**. The table should look like the example in Table 1.

Train Dataset	Criterion	Depth	Train Acc.	Val. Acc.	Val. Avg Rec.	Val. Avg Prec.	Val. Avg F1	Time Train	Time Val.
Small	Gini	2	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	T s	T s
…	….	…	…	…	…	…	…	…	…
Very Large	Entropy	None	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	T s	T s

Table 1. Example of Results Table for Decision Trees.

Random Forest Classifier. You are required to use the different **training datasets** available to train different Random Forest models using different combinations of hyperparameters: **n of estimators** (5, 10, 50, 100) and **max depth** (2, 4, 6). You are expected to systematically evaluate at least 48 configurations (4 **datasets** x 4 **N estimators** x 3 **max depths**). You must report the same metrics used for decision trees. You also need to build a result table following the same requirements as the one for Decision Trees. The table should look like the example in Table 2.

Train Dataset	N trees	Depth	Train Acc.	Val. Acc.	Val. Avg Rec.	Val. Avg Prec.	Val. Avg F1	Time Train	Time Val
Small	5	2	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	T s	T s
…	….	…	…	…	…	…	…	…	…
Very Large	100	6	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	T s	T s

Table 2. Example of Results Table for Random Forests.

Logistic Regression. You are required to use the different **training datasets** available to train different Logistic Regression models using different combinations of hyperparameters: **penalty** (None, "l1", "l2"), and **C** (0.1, 0.5, 1.0, 2.0, 4.0). Note that you need to use the **'saga'** solver to iterate between these different penalties. You are expected to systematically evaluate at least 60 configurations (4 **datasets** x 3 **Penalties** x 5 **C values**). You must report the same metrics used for decision trees.

You also need to build a result table following the same requirements as the one for Decision Trees. The table should look like the example in Table 3.

Train Dataset	Penalty	C	Train Acc.	Val. Acc.	Val. Avg Rec.	Val. Avg Prec.	Val. Avg F1	Time Train	Time Val
Small	None	0.1	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	T s	T s
…	….	…	…	…	…	…	…	…	…
Very Large	L2	4.0	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	0.XXXX%	T s	T s

Table 3. Example of Results Table for Logistic Regression.

Part 3. Training

This is a simpler script (**part_03_training.py**) that takes the entire training set and trains a classifier using all data available. Both the trained classifier and the trained normalizer (StandardScaler) must be stored in temporary files for later usage on the next stage (testing). The script takes the following command line arguments: config filename, training dataset file name, file name for saving the trained Standard Scaler, file name for saving the trained classifier. This process is illustrated in Figure 4.

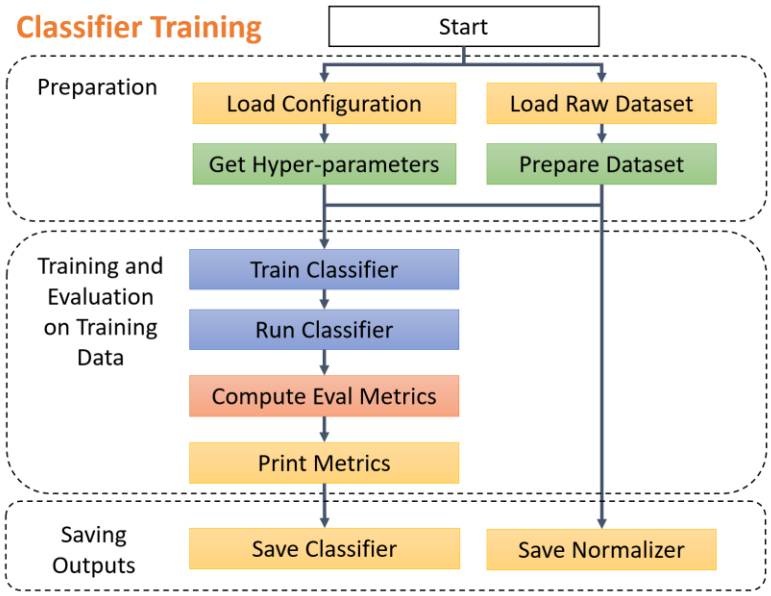

Figure 4. Training a Classifier model. After training, the performance on training set is evaluated. In the end, both the trained classifier and the StandardScaler must be stored in temporary files.

Again, the specific classifier to train is given by the **active classifier** parameter in the configuration. In addition, you are expected to use the results from running the grid search script to identify the **best configuration for each classifier**. You have to **manually set** these best hyper-parameters found in the corresponding entries in the configuration.

Part 4. Testing

This script (part_04_testing.py) that takes a dataset (training or testing), normalizes it using a previously trained standard scaler, and then predicts the class for each example using the trained classifier model. The script must then compute and print the evaluation metrics. The script takes the following command line arguments: dataset file name, trained standard scaler file name, trained classifier filename. This process is illustrated in Figure 5.

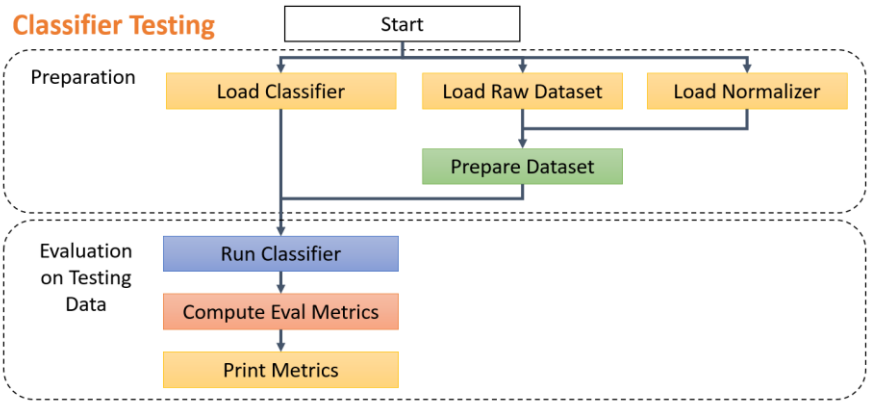

Figure 5. Testing a trained classifier model. This requires loading both the trained classifier and trained normalizer The test dataset must be processed using the trained StandardScaler.

The Report (30%)

You have to prepare a written report with the results for the experiments carried out here. The report should clearly state the name of the author. It must have the following sections:

1. **Introduction**
    a. Briefly explain the whole assignment in your own words.
2. **Code architecture (0.5 – 1.0 pages max)**
    a. Briefly describe how your custom code works.
    b. Usage of diagrams is highly encouraged.
    c. Code snippets might be included but you still have to explain what they do with words.
3. **Computer Specs.**
    a. Provide all the relevant specs for the computer that you used to run your experiments. In particular, mention the CPU used, the operating system, and the memory ram available.
    b. Keep in mind that a single computer should be used to collect all data.
4. **Experimental Results for Part 2 (Grid Search).**
    a. For each classifier (Decision trees, Random Forests and Logistic Regressor):
        i. Include the raw Result tables. **Must be a proper table**, do not copy/paste raw outputs.
        ii. Provide an **in-depth analysis** of these Results.
            1. Consider how different hyperparameters influences different evaluation metrics.
            2. Analyze what configurations had the highest and the lowest levels of overfitting.

5. **Experimental Results for Parts 3 and 4 (Training and Testing).**
    a. For each classifier (Decision trees, Random Forests and Logistic Regressor):
        i. Include a table with the evaluation metrics for the training dataset.
        ii. Include a table with the evaluation metrics for the final test dataset.
        iii. Provide an in-depth analysis of these Results.
            1. Are these results consistent with the performance on grid-search? Explain in detail.
            2. Are these results showing over-fitting? Provide an explanation based on your data.
6. **General Analysis**.
    a. Determine which classifier achieved the best performance here, and try to explain why.
    b. Are these numbers good? Explain in detail.
    c. Can these numbers be improved? How?
7. **Difficulties (Optional).**
    a. Provide a brief summary of difficulties that you faced while implementing this project.
8. **New application (0.25 – 0.5 pages).**
    a. Propose an entirely different application where you would use any of these classification algorithms.
    b. Describe the application in detail.
    c. What attributes can be used?
    d. Why do you think that the selected classifier would work better than the other two options?
9. **Feedback**
    a. What did you think about this assignment? (THIS IS NOT A conclusion)
10. **Conclusions.**
    a. NOT part of the feedback. Any final remarks and reflections on what you learned from doing this assignment.

Delivery Instructions

You are being provided with a Zip file including a basic PyCharm project. You are required to implement your solution using the code provided. When done, **you should simply zip back the entire project and submit as a ZIP file**.

1. **DO NOT rename any internal project files!**
2. **Using other compression formats (.rar) will lead to a penalty.**
3. **If you are using virtual environments in python, DO NOT include it on your zip file! It will make the project huge!**

File names. The zip file that you submit should use "[Last Name(s)], [Given Name(s)].zip" as it appears in D2L. For example, "Kenny Davila Castellanos" (Davila Castellanos is two last names), would have to submit the homework with the name "Davila Castellanos, Kenny.zip". Another student named "Kenny Mauricio Davila" (Mauricio is a middle name), would have to submit the homework as "Davila, Kenny Mauricio.zip".

Policies

1. This homework is meant to be worked **individually**.
2. All general policies about Plagiarism and Cheating apply to this mini-project. If you plagiarize or receive code from other people, you will be caught and you will receive a score of 0, and the academic integrity violation will be filed.
3. There is a data collection component in this assignment. Creating fake numbers is highly unethical and is considered Cheating.
4. You must not use Chat-GPT or any other code generators for completing this assignment. Doing so will be considered a form of cheating and are also eligible to receive a score of 0.
5. Do not post your solutions online and do not share them with anyone. It is your responsibility to safe guard your private data.
6. Code that does not compile due to syntax and/or semantic errors will automatically receive a score of 0. It is hard to assign partial credit when I cannot even run your code.
7. Must use PyCharm if you want to receive technical support from my end.
8. You must follow the delivery instructions.
9. Late submissions without justification might receive penalties proportional to the number of hours late. See the syllabus for concrete details on late penalties.
10. The code and documentation highlight using specific approaches for certain things. **You are expected to follow these approaches. Any different approach that you use without consulting me first is likely to receive a score of 0**.
11. Do ask for help if anything is unclear, but do it in a timely manner (e.g., by e-mail or during the Office Hours).