

Python Introduction - Exercise Set

ISEG Executive Education - PG AAIML

Diogo Silva

Table of contents

Session 1: U0 – U3	2
Exercise 1: Investment analyst – investment over 3 years	2
Exercise 2: Investment analyst – political influence	3
Session 2: U4 – U7	3
Exercise 3: Expense Tracker	3
Exercise 4: Training Data Splitter	3
Exercise 5: Product Inventory	4
Exercise 6: Sales Report Generator	4
Exercise 7: Batch Data Generator	4
Exercise 8: Investment analyst – lists and loops	4
Exercise 9: Investment analyst – one simulation per party	5
Exercise 10: Investment analyst – volatility	5
Exercise 11: Investment analyst – Monte Carlo simulation	6
Exercise 12: Income Tax Calculator	6
Exercise 13: Learning Rate Scheduler (ML)	7
Exercise 14: Password Validator	7
Exercise 15: Confusion Matrix Calculator	7
Exercise 16: Dataset Preparation	7
Exercise 17: Employee Database	8
Exercise 18: Shopping Cart	8
Exercise 19: Text Analyzer	8
Exercise 20: Meeting Scheduler	9
Exercise 21: Model Performance Tracker	9
Exercise 22: Feature Engineering Pipeline	9
Exercise 23: Hyperparameter Grid Generator	10
Exercise 24: Label Distribution Analyzer	10
Exercise 25: Product Sales Analysis	10
Exercise 26: Investment analyst – political influence expected value	11
Exercise 27: Investment analyst – to donate or not to donate	12
Exercise 28: Investment analyst – functions	12
Session 3: U8 – U13	12
Exercise 29: Feature Transform Library	13

Exercise 30: Feature Selection Pipeline	13
Exercise 31: Data Augmentation Generator	14
Exercise 32: Cross-Validation Split Generator	14
Exercise 33: Model Evaluation Module	14
Exercise 34: Investment analyst – files	15
Exercise 35: Investment analyst – modules	15
Exercise 36: Boston Housing Dataset - Data Normalization	15
Exercise 37: Experiment Logger	16
Exercise 38: Dataset File Processor	16
Exercise 39: Training Configuration Manager	17
Exercise 40: Ensemble Results Combiner	17
Exercise 41: Model Checkpoint Manager	20
Background	20
Provided Code	20
Your Tasks	21
Expected Output	22
Exercise 42: Investment analyst – catching errors	22
Exercise 43: Investment analyst – debugging with colleagues	22

Tips:

- Start by understanding what the exercise asks
- Break the problem into smaller steps
- Test your code with the provided examples
- Try variations of the solutions
- Don't hesitate to combine techniques you've learned
- For ML exercises: reflect on why these operations matter in real world projects

Each exercise lists the course units it practises. Exercises named *Investment analyst* build on each other around the same problem, from session 1 to session 3.

Session 1: U0 – U3

In class: Exercise 1 (Exercise 2 if there is time).

Unit	Recommended
U2 Variables, I/O and data types	1, 2
U3 Making decisions	1, 2

Exercise 1: Investment analyst – investment over 3 years

Units: U2, U3

- You are an analyst working for a large investment corporation.
- Your task is to analyze the evolution of investments over time and provide insights to clients.

- For this exercise, you will receive (input) the amount of initial investment value.
- For each year, your forecaster colleague will give you (input) the predicted return.
- You will calculate the investment value for each year, over a period of 3 years.
- You will also determine if the investment has increased or decreased compared to the previous year, for each year.

Exercise 2: Investment analyst – political influence

Units: U2, U3

- An election is coming up, and your clients are concerned about the potential impact of different political parties on their investments.
- You need to factor in the influence of political parties on the investment returns.
- The client will provide you with the initial investment value.
- Your colleague, the forecaster, has provided you with the base annual return rate.
- You have also partnered with a political analyst who has provided you with:
 - the impact factors of different political parties on the annual return rate - this value should be summed to the base annual return rate.
 - a prediction of which party will be in power in each year;
- Your task is to calculate the investment value for each year, considering the influence of political parties, and determine, for each year, if the investment will increased or decreased compared to the previous year.

Session 2: U4 – U7

In class: Exercise 8 after lists and loops, Exercise 17 after dictionaries.

Unit	Recommended
U4 Lists and tuples	4, 5
U5 Loops	8, 12
U6 Dictionaries and comprehensions	17, 19
U7 Functions	28, 14 (extra challenge)

Exercise 3: Expense Tracker

Units: U4, U5

You're managing your monthly expenses. Create a program that:

1. Stores your expenses in a list: [1250.50, 89.90, 45.00, 320.75, 12.50]
2. Calculates the total spent
3. Finds the average expense
4. Counts how many expenses were above 100€

Exercise 4: Training Data Splitter

Units: U4

You have a dataset of 100 samples that need to be split into training (70%), validation (20%), and test (10%) sets. Create a program that:

1. Creates a list of sample IDs from 0 to 99
2. Splits them into three sets according to the percentages
3. Prints the size of each set and first 5 IDs from each

Extra challenge: randomly sample the samples for each set; hint: use the built-in `random` module.

Exercise 5: Product Inventory

Units: U4, U5, U6

A small shop needs to manage its inventory. Create a program that:

1. Stores products as tuples: (product_name, quantity, price)
2. Creates a list with these products: ("Laptop", 5, 899.99), ("Mouse", 23, 15.50), ("Keyboard", 12, 45.00)
3. Finds products with quantity below x (x is received from user)
4. Calculates total inventory value
5. Creates a dictionary mapping product names to their stock quantity

Extra challenge: receive more products with input, in a loop. Stop when product name is something very specific that you define, e.g. "-1".

Exercise 6: Sales Report Generator

Units: U4, U5

You have daily sales data for a week. Create a program that:

1. Stores sales in a list: [1200, 1450, 980, 1100, 1650, 890, 1320]
2. Identifies which days had sales above average
3. Finds the best and worst sales days

Exercise 7: Batch Data Generator

Units: U4, U5

Simulate creating batches for training. You have 47 samples and need to create batches of size 8.

Create a program that:

1. Creates sample IDs (0 to 46): `range(47)`
2. Splits them into batches of 8 (you must choose the data structure for each batch, and the one that will hold all batches)
3. Shows how many full batches and the size of the last batch

Exercise 8: Investment analyst – lists and loops

Units: U4, U5

- Redo Exercise 1 with lists and loops.

- Now, the user chooses the initial investment value and the number of years to simulate (use `input`).
- Your forecaster colleague will give you the prediction for the returns for those years (comma separated `float` values).
- The user should be able to see the evolution of the investment over the years and if it increased or decreased compared to the previous year.
- Show a formatted table with the results, first column is the year, second column is the value in that year.

Exercise 9: Investment analyst – one simulation per party

Units: U4, U5

- Redo Exercise 2 with lists, as in Exercise 8.
- A country will have elections. There are K parties relevant parties (which you hardcode in, for example, a list).
- The winning party will govern for 8 years.
- The party's policies will influence the annual return rate (ARR) over some type of investments.
- A forecaster has provided you with a base ARR (configure this in the code).
- A political analyst has provided you with how each party might influence the ARR (configure this in code)..
- The ARR for each party is the base ARR plus the influence of the party.
- For each party, do a simulation for the evolution of an investment of M€ over 8 years (M is received from the user).
- Print the final investment for each party, along with the full evolution of the party with the highest investment.
- Tips
 - The value in each year (including the initial value) must be stored.
 - How will you store this data? A list for each party? A list for each year? Something else?
 - You need to iterate over the years for each party, or vice versa. That's a loop inside a loop.
 - You need to keep track of the investment for each party.

Exercise 10: Investment analyst – volatility

Units: U5

- Based on Exercise 9, add the following features.
- Each party may also make the ARR more or less volatile (unstable).
- The volatility is given by a list of 8 floats (one for each party).
- The volatility is the standard deviation of the ROI.
- For each year, each party's final ARR is a random number with a normal distribution with the mean given by the party's ARR of the previous year and the standard deviation given by the volatility.
- Recompute investment for each party.
- Tips
 - You can generate random numbers with a normal distribution with the `numpy` library.

```
import numpy as np
mean = 0
std = 1
np.random.normal(mean, std) # random number with mean 0 and std 1
np.random.normal(mean, std, 10) # list of 10 random numbers with mean 0 and
std 1
```

Exercise 11: Investment analyst – Monte Carlo simulation

Units: U5

- Based on Exercise 10, add the following features.
- Because we're dealing with random events, let's do a Monte Carlo simulation.
- Simulate the evolution of the investment 1000 times for each party.
- Print the average investment and its standard deviation for the parties with the highest final average and the one with lowest standard deviation.
- For the party with the highest average, include also the 25%, 50% and 75% percentiles.
- Tips
 - Compute mean and standard deviation with `numpy`.

```
import numpy as np
l = [1,2,3,4,5]
np.mean(l) # -> mean
np.std(l) # -> standard deviation
```

- Compute percentiles with `numpy`.

```
import numpy as np
l = [1,2,3,4,5]
np.percentile(l, 25) # -> 25% percentile
```

Exercise 12: Income Tax Calculator

Units: U4, U5

Portuguese income tax (IRS) is progressive: income is taxed in brackets.

Write a program that receives the gross annual income and shows the total income tax and the final net income.

The brackets are defined in 3 lists:

```
bracket_min = [0, 7479, 11284, 15992, 20700, 26355, 38632, 50483, 78834]
bracket_max = [7479, 11284, 15992, 20700, 26355, 38632, 50483, 78834,
float("inf")]
rate = [14.5, 21, 26.5, 28.5, 35, 37, 43.5, 45, 48]
```

Progressive tax means the same income is taxed in different brackets, e.g.

```
Income: 20000€
Tax in bracket 1: (7479 - 0) * 14.5%
Tax in bracket 2: (11284 - 7479) * 21%
Tax in bracket 3: (15992 - 11284) * 26.5%
Tax in bracket 4: (20000 - 15992) * 28.5%
There is no tax in brackets above 4, because the income is not high enough.
```

Exercise 13: Learning Rate Scheduler (ML)

Units: U5

Create a learning rate scheduler that reduces the learning rate by 50% every 10 epochs. Start with LR=0.1 and simulate 35 epochs. Track and display the learning rate at each epoch. Your only job is to produce a list with the learning rates of all epochs, i.e. the length of that list is the number of epochs.

Exercise 14: Password Validator

Units: U5

Create a simple password validator that checks if a password:

1. Is at least 8 characters long
2. Contains at least one number
3. Contains both uppercase and lowercase letters

Test with: “Secure123”, “weak”, “NOLOWER1”, “noupper1”

Extra challenge (U7): put the checks in a function `is_valid_password(password)` that returns `True` or `False`.

Exercise 15: Confusion Matrix Calculator

Units: U4, U5

You have predictions from a binary classifier. Create a program that:

1. Takes actual labels: [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
2. Takes predictions: [1, 0, 1, 0, 0, 1, 1, 0, 1, 0]
3. Calculates True Positives, True Negatives, False Positives, False Negatives
4. Computes Accuracy, Precision, and Recall
5. Displays the confusion matrix

Exercise 16: Dataset Preparation

Units: U5

You’re preparing a dataset for training. You have raw data with missing values represented as -999: - Features: [1.2, 3.4, -999, 5.6, 2.1, -999, 4.5, 3.2]

Create a program that:

1. Counts how many missing values exist
2. Replaces missing values with the mean of valid values
3. Normalizes all values to 0-1 range (min-max scaling)
4. Prints original vs processed data side-by-side

Exercise 17: Employee Database

Units: U6

You're building a simple employee management system. Create a dictionary with employee IDs as keys and their information as values (this information should also be a `dict`):

- Employee 101: Maria Silva, salary 3500, department "Sales"
- Employee 102: João Santos, salary 3100, department "IT"
- Employee 103: Ana Costa, salary 2800, department "Sales"

Then:

1. Print all employees with salary above 2600
2. Calculate the average salary for the Sales department
3. Add a new employee (104: Pedro Alves, salary 3500, department "IT")

Extra challenge: receive employees from the user with `input` inside a loop, only stop when employee number is a negative number.

Exercise 18: Shopping Cart

Units: U6

Simulate a shopping cart system:

1. Create a dictionary with products and prices: `{"apple": 0.50, "bread": 1.20, "milk": 0.90, "cheese": 2.50}`
2. Create an empty cart (list of tuples: product name and quantity)
3. Add items to cart: 3 apples, 1 bread, 2 milk (or receive from user)
4. Calculate total cost
5. Apply a 10% discount if total is above 5€

Exercise 19: Text Analyzer

Units: U6

Create a program that analyzes a piece of text:

1. Count total words
2. Count how many words start with a vowel
3. Find the longest word
4. Create a dictionary with word lengths as keys and list of words as values

Test with: "Python programming is powerful and versatile"

Added challenge - Bag of words: count the occurrence of each word in a text; test with the text in [this link](#)

Added challenge 2 - n-gram counter: instead of words, count the number of times n letters appear together, e.g. "Istituto" leads to trigrams (3-gram) "ins" "nst" "sti" "tit" "itu" "tut" "uto", but there is no repetition here; make it so n is a parameter, i.e. you can easily change it in a variable, and the rest of the logic remains intact.

Exercise 20: Meeting Scheduler

Units: U4, U6

You're scheduling meetings for a week. Create a program that:

1. Stores meeting slots as tuples: (day, hour, duration_minutes, attendees_count)
2. Create a list with: ("Monday", 10, 60, 5), ("Monday", 14, 30, 3), ("Tuesday", 9, 90, 8), ("Wednesday", 11, 45, 4)
3. Calculate total meeting hours for the week
4. Find the busiest day (most total meeting time)
5. Calculate average number of attendees per meeting
6. Convert the list to a dictionary, with days as keys and list of meetings as values (sorted chronologically).

Exercise 21: Model Performance Tracker

Units: U6

You're training multiple models and need to track their performance. Create a system that:

1. Stores model results as dictionaries with: model_name, accuracy, training_time (minutes), parameter_count
2. Creates a list with results for: CNN (0.87, 356, 500000), SwinT (0.91, 1200, 1200000), FCN (0.82, 3, 50000), CNN2 (0.85, 65, 200000)
3. Finds the best model by accuracy
4. Calculates the average training time in hours
5. Identifies models with accuracy > 0.85 and training time < 2 hours

Note you might want to revisit this exercise with Exercise 37.

Exercise 22: Feature Engineering Pipeline

Units: U5, U6

You have raw sensor data and need to create features. Given temperature readings: [20.5, 21.3, 22.1, 21.8, 23.2, 24.1, 23.5, 22.9]

Create a program that generates these features:

1. Moving average (window size 3)
2. Temperature difference from previous reading
3. Boolean feature: is_increasing (compared to previous)
4. Store all features in a dictionary with descriptive keys

Exercise 23: Hyperparameter Grid Generator

Units: U5, U6

You need to create all combinations of hyperparameters for grid search. Generate combinations for:

- learning rate: [0.001, 0.01, 0.1]
- batch size: [8, 16, 32, 64]
- epochs: [10, 20]

Create a program that:

1. Generates all possible combinations
2. Counts total combinations
3. Estimates total training time (assume each takes 5 minutes)
4. Stores combinations in a list of dictionaries

Extra challenge - parameters with dependencies Consider that this model has some architectural modules which can be easily swapped: a feature extractor and an aggregation head. The feature extractors include `resnet18`, `resnet50`, `resnet100`, `resnet152` and `ViT`. The heads include `GeM`, `NetVLAD` and `MLP`. For `resnet100` and `resnet152`, batch size may not be higher than 8. For `ViT`, `NetVLAD` will not work. Redo the exercise with these 2 added parameters, and with the dependencies in mind.

Note: this kind of logic is not supported in most built-in libraries, e.g. Scikit-Learn.

Exercise 24: Label Distribution Analyzer

Units: U6

You have a dataset with class labels. Analyze if there's class imbalance:

- Labels: [0, 1, 0, 0, 1, 2, 0, 1, 0, 2, 0, 0, 1, 0, 2, 0, 0, 0, 1, 0]

Create a program that:

1. Counts samples per class
2. Calculates the percentage distribution
3. Identifies if dataset is imbalanced (majority class > 60%)
4. Suggests whether resampling might be needed
5. Creates a dictionary mapping class labels to their counts

Extra challenge: sample classes so that number of samples per class is roughly the same. The output sample should be the indices of the original samples.

Exercise 25: Product Sales Analysis

Units: U4, U6

The code snippet at the bottom, contains 2 lists: 1. the IDs of the products that were transacted in a given day 2. a list of the price of the products

Copy these lists and write code to answer the following questions.

1. Which product was most transacted?
2. What is the ID of the cheapest product?
3. What is the ID of the most expensive product?
4. What is the average product price?
5. What is the total revenue for the day?
6. How many times was product with ID 1 transacted? And ID 2?
7. Create a dictionary where the keys are IDs of the products transacted and the values are the counts of the times that product was transacted in the day.
8. Change the values of the dictionary so that now we have a tuple, where the first element is the number of transactions (as before) and the second is the total revenue of that product, e.g. (1, 1013.73) -> not a the real value.
9. The products with an ID less than 10 are for pet products. How many transactions for pet products?
10. How much revenue for pet products?

```
products_ids_day1 = [33, 28, 37, 25, 18, 4, 0, 14, 42, 6, 24, 20, 44, 14, 49,
10, 4, 32, 11, 49, 48, 25, 11, 10, 43, 28, 2, 2, 26, 9, 9, 39, 20, 43, 37, 15,
26, 18, 37, 38, 40, 32, 2, 19, 48, 38, 33, 17, 47, 24, 19, 12, 43, 50, 5, 41,
30, 44, 44, 44, 13, 5, 50, 23, 36, 37, 49, 28, 44, 26, 45, 39, 30, 34, 17, 44,
47, 13, 23, 2, 10, 39, 34, 44, 28, 33, 6, 41, 31, 10, 48, 0, 31, 13, 15, 23, 30,
50, 48, 27, 29, 25, 41, 44, 3, 49, 14, 13, 4, 21, 47, 15, 28, 44, 29, 20, 14,
17, 50, 45, 17, 38, 45, 20, 34, 11, 49, 17, 1, 22, 31, 41, 31, 0, 12, 48, 49, 7,
48, 40, 38, 6, 36, 37, 27, 10, 26, 41, 21, 26, 3, 12, 31, 36, 48, 15, 41, 49,
38, 3, 32, 47, 39, 28, 35, 5, 17, 10, 14, 5, 27, 5, 18, 35, 2, 8, 40, 41, 30,
26, 2, 34, 23, 21, 30, 45, 9, 41, 27, 35, 47, 24, 37, 17, 16, 35, 14, 4, 7, 28]

# value at index 0 has the price of product with ID 1, index 1 -> ID 2, ...
ids_price = [37.73, 615.92, 789.51, 362.33, 159.15, 482.15, 332.99, 264.94,
110.41, 917.21, 3.99, 358.15, 6.37, 15.0, 495.94, 26.12, 202.62, 438.0, 672.33,
369.49, 79.68, 703.73, 93.0, 89.97, 3.34, 101.3, 151.12, 807.89, 2.45, 117.8,
426.9, 33.87, 260.01, 341.21, 74.43, 709.64, 218.8, 851.58, 100.12, 217.5, 35.91,
534.81, 114.68, 189.61, 441.05, 65.25, 172.97, 494.9, 87.64, 524.5]
```

Exercise 26: Investment analyst – political influence expected value

Units: U6

- The client has given up on trying to predict which party will be in power each year.
- Instead, they just want to know the expected value of their investments based on the probability of each party being in power.
- They are assuming the probabilities will not change over the next N years.
- The client will provide you with
 - the initial investment value;

- number of years to simulate.
- Your colleague, the forecaster, has provided you a base annual return rate.
- Your colleague, the political analyst, has provided you with
 - the impact factors of different political parties on the annual return rate;
 - the probabilities of each party winning the upcoming election.
- Your task is to calculate the expected investment value for each year during N years, considering the probabilities of each party being in power, and determine if the investment has increased or decreased compared to the previous year, for each year.

Exercise 27: Investment analyst – to donate or not to donate

Units: U6

- Your client is considering making a large donation to a political party.
- They want to understand the potential impact of this donation on their investments.
- Your colleague, the forecaster, has provided you with the base annual return rate.
- Your colleague, the political analyst, has provided you with
 - impact factor of each party on the annual return rate;
 - the initial probability of each party winning the upcoming election;
 - the additional probability of each party winning the election if the client makes a donation.
- Your task is to determine if the client should make the donation based on the expected investment value over the next N years. And, if yes, to which party.
 - For each donation scenario (no donation, donation to party A, B, ...), calculate the expected investment value over the next N years.
 - Compare the final value of each scenario.
 - Provide a recommendation to the client based on the comparison.

Exercise 28: Investment analyst – functions

Units: U7

- Redo exercises from session 2, but make them more modular and flexible by using functions.
- For example, redo the Monte Carlo simulation (Exercise 11).
- Make a function that receives the initial investment, base ARR and standard deviation, and makes the simulation for N years.
- Then make another function that calls the first function 1000 times and computes the average and standard deviation of the final investment.
- Make sure to test your functions with different inputs.
- Combine those functions to implement all features from that exercise and implement other functions you think will make the code cleaner and more reusable.

Session 3: U8 – U13

In class: Exercise 41 and Exercise 42. Exercise 36 is walked through as a demo, including exception handling.

Unit	Recommended
U8 map and filter	29, 30
U9 Modules and packages	33, 35
U10 Files	37, 41
U11 Exceptions	42, 39
U12 Classes	41 (uses a class), 39 (extra challenge)
U13 Debugging	43

Tips for this session:

- Always use context managers (`with` statement) when working with files
- Use `pathlib.Path` instead of string manipulation for file paths
- Keep functions focused on a single task (single responsibility principle)
- Use meaningful function and parameter names
- Test edge cases (empty lists, missing files, invalid inputs)
- Remember that `map()` returns an iterator - convert to list if needed
- When creating modules, think about code organization and reusability

Exercise 29: Feature Transform Library

Units: U7, U8, U9

Create a module `transformers.py` with transformation functions and use `map` to apply them:

1. Function `log_transform(x)` - applies logarithmic transformation (handle $x \leq 0$)
2. Function `square_transform(x)` - squares the value
3. Function `clip_outliers(x, lower, upper)` - clips values outside the range
4. Function `apply_transform(values, transform_func)` - uses `map()` to apply any transformation to a list. Note that what is being passed in the second argument is the actual function.

Test with: `[1, 5, 10, 50, 100, 500, 1000]`

Exercise 30: Feature Selection Pipeline

Units: U8, U10

Create functions to select the most informative features:

1. Function `calculate_variance(values)` - calculates variance of a feature
2. Function `select_high_variance_features(data_dict, threshold)` - `data_dict` has feature names as keys and lists of values as values. Use `filter()` to keep only features with variance above threshold
3. Function `save_selected_features(selected_features, filepath)` - saves the list of selected feature names to a file
4. Function `load_feature_config(filepath)` - loads the feature names from the file

5. Function `apply_feature_selection(data_dict, feature_names)` - returns a new dict with only the selected features

Test with `BostonHousing.csv` dataset ([link](#)).

Reflection: What threshold should you pick? Should some normalization be applied first?

Exercise 31: Data Augmentation Generator

Units: U8, U10

Create functions that generate augmented versions of data:

1. Function `add_noise(value, noise_level=0.1)` - adds Gaussian noise to a value (simulate by adding/subtracting `noise_level * value`).
2. Function `scale_value(value, scale_factor)` - scales a value by a factor
3. Function `augment_sample(sample, augmentation_funcs)` - takes a list of values and a list of augmentation functions, applies all functions
4. Use `map()` to create augmented versions of an entire dataset
5. Function `save_augmented_data(original, augmented, filepath)` - saves both original and augmented data to a file with clear labels

Test with: `[1.0, 2.5, 3.7, 4.2, 5.8]` and create 2 augmented versions.

Added challenge: augment the `BostonHousing.csv` dataset ([link](#)) and use column-wise variance for the random noise augmentation;

Exercise 32: Cross-Validation Split Generator

Units: U8, U9, U10

Create a module for generating cross-validation splits using k-fold:

1. Function `create_folds(data_indices, k=5)` - splits indices into k approximately equal folds and returns a list of folds
2. Function `get_train_test_split(folds, test_fold_index)` - given the folds and which fold to use as test, returns train and test indices
3. Function `save_cv_splits(folds, directory)` - saves each fold to a separate file (`fold_0.txt`, `fold_1.txt`, etc.) using `pathlib`
4. Function `load_cv_splits(directory)` - loads all fold files from the directory and returns them as a list
5. Write a function that uses `map()` to apply a validation function to all k folds

Test with 50 sample indices and k=5 folds.

Reflection: imagine you have an imbalanced dataset (class wise) - would you change anything when creating the folds?

Exercise 33: Model Evaluation Module

Units: U7, U9

Create a Python module called `ml_metrics.py` with functions to calculate classification metrics:

1. Function `accuracy(y_true, y_pred)` - returns accuracy score
2. Function `precision(y_true, y_pred)` - returns precision for class 1
3. Function `recall(y_true, y_pred)` - returns recall for class 1
4. Function `f1_score(y_true, y_pred)` - returns F1 score (harmonic mean of precision and recall)
5. Function `evaluate_model(y_true, y_pred)` - returns a dictionary with all metrics

In a separate notebook or script, import your module and test it with: - `y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]` - `y_pred = [1, 0, 1, 0, 0, 1, 1, 0, 1, 0]`

Exercise 34: Investment analyst – files

Units: U10

- Redo the previous exercise to load the parties and their details from a file.

Exercise 35: Investment analyst – modules

Units: U9, U10

- Redo the previous exercise and make a module with the functions for the individual simulations and the Monte Carlo simulation in a module.
- On another module, add the functions to load the data and returned it in a format that the simulation functions can use.
- Modify the original program to implement the full Monte Carlo simulation using the functions from the module.

Exercise 36: Boston Housing Dataset - Data Normalization

Units: U7, U10

In this exercise, you will work with a real dataset - the well known Boston House Prices dataset. You can download it from [here](#) or find it easily online (e.g. Kaggle). The exercise will require you to program the data loading functionality, as well as some basic normalization routines. Don't use any external package, like Pandas, as that has all these features built-in.

- **Data loading**
 - Create a function that will load the dataset, in Comma Separated Values (CSV) format, and returns a `dict` with pairs of {column, list of values}.
 - The name of each column is specified in the first line of the CSV file.
 - Convert each value to float.
- **Scaling**
 - Write a function `mean(values)` that computes the mean of a list of values.
 - Write a function `std(values)` that computes the standard deviation of a list of numbers.
 - Output the mean and std of every column.
 - Write a function `min_max_scale(values)` that normalizes a list of numbers to the 0-1 range
 - Write a function `standardize(values)` that standardizes data (subtract mean, divide by standard deviation)

- Write a function `apply_scaling(data, method)` that takes a list and a method name (“minmax” or “standard”) and applies the appropriate scaling
- Use `apply_scaling` with both methods to an arbitrary column and compare results.
- **Data card** (capitalized column name, followed by description)
 - CRIM per capita crime rate by town
 - ZN proportion of residential land zoned for lots over 25,000 sq.ft.
 - INDUS proportion of non-retail business acres per town
 - CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
 - NOX nitric oxides concentration (parts per 10 million)
 - RM average number of rooms per dwelling
 - AGE proportion of owner-occupied units built prior to 1940
 - DIS weighted distances to five Boston employment centres
 - RAD index of accessibility to radial highways
 - TAX full-value property-tax rate per \$10,000
 - PTRATIO pupil-teacher ratio by town
 - B 1000($B_k - 0.63$)² where B_k is the proportion of blacks by town
 - LSTAT % lower status of the population
 - MEDV Median value of owner-occupied homes in \$1000’s

Exercise 37: Experiment Logger

Units: U7, U10

Create a system that logs ML experiments to a text file:

1. Write a function `log_experiment(model_name, hyperparameters, metrics, filename="experiments.txt")` that appends experiment details to a file
2. The log entry should include: timestamp (you can use a simple counter), model name, hyperparameters (as dict), and metrics (as dict)
3. Write a function `read_experiments(filename="experiments.txt")` that reads and prints all logged experiments
4. Test by logging 3 different experiments with different models and hyperparameters
5. Use context managers for file operations

Exercise 38: Dataset File Processor

Units: U10

You have multiple CSV files with training data from different sources. Create functions to process them:

1. Write a function `load_dataset(filepath)` that reads a CSV file and returns the data as a list of dictionaries (each row is a dict).
2. Write a function `filter_by_threshold(data, column, threshold)` that filters rows where a column value exceeds a threshold
3. Write a function `save_processed_data(data, output_path)` that saves the processed data to a new CSV file

4. Use `pathlib.Path` for file operations
5. Create a function `process_pipeline([input_file1, input_file2], output_file, column, threshold)` that combines all steps, but also accepts a list of input CSV files instead of just one. The data from those files must be combined before being fed to `filter_by_threshold` and `save_processed_data`.

Split the `BostonHousing.csv` ([link](#)) dataset into 3 different files manually, e.g. first 100 rows for file 1, rows 100-200 for file 2, and the remaining rows for file 3.

Exercise 39: Training Configuration Manager

Units: U9, U10, U11

Create a system to manage training configurations:

1. Function `create_config(learning_rate, batch_size, epochs, optimizer)` - returns a configuration dictionary
2. Function `save_config(config, filepath)` - saves the configuration to a text file in a readable format (key: value pairs)
3. Function `load_config(filepath)` - loads configuration from file and returns it as a dictionary
4. Function `validate_config(config)` - checks that all required keys exist and values are within reasonable ranges
5. Create a package structure with separate modules for config management and validation

Test by creating, saving, loading, and validating different configurations.

Tip: use the `json` module.

`validate_config` should raise a `ValueError` with an informative message when the configuration is invalid (U11).

Extra challenge (U12): implement the configuration as a `TrainingConfig` class with `save`, `load` and `validate` methods.

Exercise 40: Ensemble Results Combiner

Units: U10

Create functions that combine predictions from multiple models (stored in separate files) using different strategies: majority voting for classification, averaging for regression.

Added challenge: for classification, use the estimated confidence alongside the actual prediction for refining the final ensemble result.

Data:

```
# Mock Classification Predictions (3 models predicting classes 0, 1, 2)
modell_classification = """sample_id,prediction,confidence
0,1,0.85
1,0,0.92
```

```

2,2,0.78
3,1,0.88
4,0,0.95
5,2,0.71
6,1,0.83
7,0,0.89
8,2,0.76
9,1,0.91"""

model2_classification = """sample_id,prediction,confidence
0,1,0.82
1,0,0.88
2,1,0.65
3,1,0.90
4,0,0.93
5,2,0.68
6,1,0.86
7,1,0.72
8,2,0.81
9,1,0.87"""

model3_classification = """sample_id,prediction,confidence
0,1,0.79
1,0,0.91
2,2,0.73
3,0,0.67
4,0,0.96
5,1,0.62
6,1,0.84
7,0,0.85
8,2,0.79
9,1,0.89"""

# Model 1 Regression Predictions
modell_regression = """sample_id,prediction
0,42.5
1,38.2
2,55.7
3,61.3
4,29.8
5,48.9
6,52.1

```

```

7,45.6
8,33.4
9,58.2"""

# Model 2 Regression Predictions
model2_regression = """sample_id,prediction
0,43.1
1,37.8
2,54.9
3,62.5
4,30.2
5,49.5
6,51.8
7,46.2
8,34.1
9,57.8"""

# Model 3 Regression Predictions
model3_regression = """sample_id,prediction
0,42.8
1,38.5
2,56.2
3,60.9
4,29.5
5,48.2
6,52.5
7,45.1
8,33.8
9,58.9"""

ground_truth_classification = """sample_id,true_label
0,1
1,0
2,2
3,1
4,0
5,2
6,1
7,0
8,2
9,1"""

```

```
ground_truth_regression = """sample_id,true_value
0,42.7
1,38.0
2,55.5
3,61.5
4,30.0
5,49.0
6,52.0
7,45.5
8,33.5
9,58.5"""
```

Exercise 41: Model Checkpoint Manager

Units: U10, U12

#In machine learning, it's crucial to save model states during training so you can:

- Resume training if it gets interrupted
- Compare performance across different epochs
- Select the best model based on validation metrics
- Detect and analyze overfitting

In this exercise, you'll implement a checkpoint management system that saves model states during training and helps identify overfitting patterns.

Background

You are provided with a `MockModel` class in `mockmodel.py` that simulates realistic neural network training. The model exhibits a common problem in deep learning: **overfitting**. Overfitting generally happens when the training accuracy continues to improve, but validation accuracy starts to decrease, indicating the model is memorizing the training data rather than learning generalizable patterns.

Your task is to implement a checkpoint system to track performance metrics (loss and accuracy), and store checkpoints of model parameters, during the training procedure.

Provided Code

- The `mockmodel` module with the `MockModel` class is provided. This class has the following relevant methods:
 - `get_model_state()` - Returns the current model weights/parameters
 - `train_eval_model_iteration()` - Simulates one epoch and returns `(train_acc, train_loss, val_acc, val_loss)`
- **Basic training loop** that runs for 80 epochs
- Download the module with the mock model from this link: [mockmodel.py](#)

- Download the starter code with the basic training loop from this link: [startercode.py](#)

Your Tasks

1. Implement Checkpoint Saving Function

Write a function that saves model checkpoints to disk:

- **Input:** Current epoch number, model state (from `model.get_model_state()`), and metrics dictionary
- **Format:** Should you save in text or binary format? Consider what you're storing (numpy arrays)
- **Filename:** Design a unique naming scheme for each checkpoint (hint: include the epoch number)
- **Directory:** Save all checkpoints to a directory called `"model_checkpoints"`
- **Path handling:** Use `pathlib.Path` to create the directory if it doesn't exist
 - Look up the `makedirs()` method and understand what `parents` and `exist_ok` arguments do
 - Reference: <https://docs.python.org/3/library/pathlib.html>

2. Save Checkpoints During Training

Modify the training loop to save checkpoints **every 5 epochs**.

3. Post-Training Analysis

After training completes, implement the following analyses:

3.1 Display All Checkpoints

Print the training and validation accuracies of all saved checkpoints, ordered by epoch number.

3.2 Load Best Checkpoint

Load the checkpoint with the best validation accuracy and print:

- Which epoch it was from
- Its validation and training accuracies

3.3 Detect Overfitting Point

Write code to automatically detect around which epoch the validation accuracy started decreasing (overfitting began).

Hints for detecting overfitting:

- Compare consecutive validation accuracies
- Look for when validation accuracy peaks and then declines
- Consider using a window of epochs to avoid noise

4. Early stopping (Optional challenge)

Implement a **patience mechanism** inside the training loop:

- Track if validation accuracy hasn't improved for 5 consecutive epochs
- Stop training early if this condition is met
- Print a message indicating early stopping was triggered and at which epoch

5. Spaced checkpoints (Optional challenge)

Instead of saving a checkpoint at every epoch, it's common to save at every N epochs (e.g. N=10) and also keep a checkpoint of the model parameters with the best validation accuracy so far. Change the training loop to reflect this idea.

Expected Output

Your implementation should:

1. Create a `model_checkpoints/` directory
2. Save checkpoint files
3. Display all checkpoint metrics after training
4. Identify the best checkpoint
5. Detect when overfitting started
6. (Optional) Stop early if patience limit is reached
7. (Optional) Implement spaced checkpoints

Exercise 42: Investment analyst – catching errors

Units: U11

- Following the Monte Carlo simulation exercises, add error handling to the functions.
- Make sure to handle errors such as incorrect file paths, incorrect file formats, and invalid user inputs, e.g. negative investment values, negative standard deviations, non numeric values, etc.
- Use try-except blocks to catch errors and provide informative messages to the user.

Exercise 43: Investment analyst – debugging with colleagues

Units: U13

- Use your implementation from the previous exercises and introduce bugs in the code.
- Share the code with a colleague and ask them to debug it (don't tell them what is the bug).
- Analogously, find and fix bugs in your colleague's code.