Week 3 - Assignment

Programming for Data Science 2025

Exercises for the topics covered in the third lecture.

The exercise will be marked as passed if you get **at least 10/18** points.

Exercises must be handed in via **ILIAS** (Homework assignments). Deliver your submission as a compressed file (zip) containing one .py or .ipynb file with all exercises. The name of both the .zip and the .py/.ipynb file must be *SurnameName* of the two members of the group. Example: Annina Helmy + Markus Anwander = *HelmyAnnina_AnwanderMarkus.zip* .

It's important to use comments to explain your code and show that you're able to take ownership of the exercises and discuss them.

You are not expected to collaborate outside of the group on exercises and submitting other groups' code as your own will result in 0 points.

For question about the lecture content or exam, contact: *annina.helmy@students.unibe.ch* with the subject: *Programming for Data Science 2025 - Lecture XY*. For questions about the excercise/grading of excercises, contact: *thea.waldleben@students.unibe.ch* or *patricia.gribi@students.unibe.ch* with the subject: *Programming for Data Science 2025 - Excercise XY*. **Deadline: 14:00, March 13, 2025.**

Exercise 1 - Error investigation 2 points

The code below squares and sums the numbers in the array *arr*, and holds the result in the variable *squared_sum*, which should be 1135. However, that is not the case. Correct the code and explain in a comment , clearly and amply, what was wrong.

In [ ]:
```
import numpy as np

arr = np.array([13, 14, 15, 16, 17], dtype=np.int8)
squared_sum = np.sum(arr ** 2)
squared_sum
```

In [ ]:
```
###
# YOUR COMMENT HERE
###
```

Exercise 2 - Vacation selector 3 points

The code below defines five vacation destinations (*locations*) and four attributes for each (*attributes*). Each row describes one destination, and the columns represent scores on the factors scenery, activities, food, and nightlife.

Write a function *vacation_advisor* that asks the user whether they find each of the attributes important or not, and suggests the best vacation spot based on these preferences.

Use techniques from the third lecture to solve the exercise.

Example interaction:

```
Is scenery important to you [y/n]?    > y
Is activities important to you [y/n]? > y
Is food important to you [y/n]?       > n
Is nightlife important to you [y/n]?  > n
Based on your preferences, the best destination is Australia
```

In [ ]:
```
# List of destinations
locations = np.array([ "Hawaii", "Thailand", "Italy", "Australia", "Japan" ]

# List of attributes for each destination. Each column is an attribute. Each
attributes = np.array([
    [8, 8, 7, 6],
    [7, 9, 8, 7],
    [8, 6, 9, 7],
    [9, 8, 8, 6],
    [7, 9, 7, 8]
])

# Declare attribute names and initialize boolean array with preferences
attribute_names = ['scenery', 'activities', 'food', 'nightlife']
```

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```

Exercise 3 - Indexing 3 points

You have two arrays of the same length: temperature *temp*, and humidity, *rh*. Write a program that:

1. Substitutes the values of *temp* for which the corresponding values of *rh* is less than 0.3 with *np.nan*.
2. On this new temperature array, calculate the mean value (do **not** calculate it on the original array).

As an example:

```
temp = [70, 80, 90]
rh = [0.5, 0.2, 0.6]

temp_nan --> [70, np.nan, 90]
temp_avg --> 80
```

In [ ]:
```
# Generate some surrogate data

np.random.seed(29041996)  # Make sure we all have the same data
temp = 20 * np.cos(np.linspace(0, 2 * np.pi, 100)) + 80 + 2 * np.random.ranc
rh = np.abs(0.1 * np.cos(np.linspace(0, 4 * np.pi, 100))
            + 0.3 + 0.05 * np.random.randn(100))
```

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```

Exercise 4 - Base converter   2 points

Write a function *int_to_bin* that takes a positive integer as input and returns the binary equivalent of that integer.

You can **not** use built-in methods such as *bin()* in your solution.

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```

Exercise 5 - Broadcasting   2 points

Reshape *a* so it is possible to multiply *a* and *b*, and explain why you had to reshape *a* to be able to multiply the two arrays.

In [ ]:
```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([2, 3])

###
# YOUR CODE GOES HERE
###
```

In [ ]:
```
###
# YOUR COMMENT HERE
###
```

Exercise 6 - Find nearest neighbor   3 points

Complete the missing functions to implement a k-Nearest Neighbors (k-NN) classifier from scratch. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm Knn is a simple machine learning algorithm. It works by comparing new data points to existing ones and classifying them based on similarity (distance).

When a new data point is given, KNN calculates the distance between this point and all other points in the dataset. The most common method is Euclidean distance. The algorithm selects the K closest points (neighbors) from the dataset. For classification, the algorithm assigns the most common class among the K neighbors to the new data point (Majority Voting).

Your task:

1. Implement the Euclidean distance function.
2. Implement function to compute distance between a test point and all training points.
3. Run and test your implementation.

In [ ]:
```python
import matplotlib.pyplot as plt
# Dataset: [Feature1, Feature2, Class]
data = np.array([
    [2.0, 4.0, 0],
    [3.0, 3.0, 0],
    [6.0, 8.0, 1],
    [7.0, 9.0, 1],
    [4.0, 5.0, 0],
    [5.0, 6.0, 1],
    [3.0, 5.0, 0],
    [4.0, 6.0, 1],
])
```

In [ ]:
```python
# Features and labels
X_train = data[:, :-1] # first two columns
y_train = data[:, -1] # last column

print("Training data:\n", X_train)
print("Labels:", y_train)
plt.figure(figsize=(6,6))
colors = ['blue' if label == 0 else 'green' for label in y_train]
plt.scatter(data[:, 0], data[:, 1], c=colors, edgecolors='black', s=100)
plt.title('Training data')
```

In [ ]:
```python
# Euclidean distance
from collections import Counter

def euclidean_distance(p1, p2):
    """
    Calculate the Euclidean distance between two points.
    :param p1: coordinates of first point
    :param p2: coordinates of second point
    :return: Euclidean distance (float)
    """
    ## YOUR CODE GOES HERE
    return

# Compute distance between all test points and all training points
def compute_distances(X_train, y_train, test_point):
    """
    Compute the Euclidean distance between a test point and all training poi
    :param X_train (numpy array): training data
    :param test_point (numpy array): test point
    :return list of tuples (distance, label) for each training point
    """
    ## YOUR CODE GOES HERE
    return
```

In [ ]:

```
### FOR TESTING YOUR IMPLEMENTAITON - Run this Cell ###
def get_neighbors(X_train, y_train, test_point, k):
    distances = compute_distances(X_train, y_train, test_point)
    print(distances)
    distances.sort()
    neighbors = distances[:k]
    return neighbors

def predict(X_train, y_train,  test_point, k = 3 ):
    neighbors = get_neighbors(X_train, y_train, test_point, k)
    labels = [label for _, label in neighbors]
    most_common = Counter(labels).most_common(1)
    return most_common[0][0]

test_point = np.array([6, 5.5])
k = 3
prediction = predict(X_train, y_train, test_point, k)
print(f"Prediction for {test_point}: {prediction}")


## Plot the dataset
plt.figure(figsize = (6,6))

# Separate class 0 and class 1 points for labeling
class_0 = X_train[y_train == 0]
class_1 = X_train[y_train == 1]

# Scatter plot for training points with different colors
plt.scatter(class_0[:, 0], class_0[:, 1], c='blue', edgecolors='black', s=10
plt.scatter(class_1[:, 0], class_1[:, 1], c='green', edgecolors='black', s=1

# Mark test point
plt.scatter(test_point[0], test_point[1], c='red', edgecolors='black', s=150

# Plot settings
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("KNN Classification Visualization")
plt.legend()
plt.show()
```

Exercise 7 - Moving average (on paper) 3 points

Given the array of values, *a*, we can calculate the moving average by averaging nearby values and repeating the procedure sliding along the array. Here's an example of a 3-point moving average (ignoring the edges), with a for loop:

In [ ]:
```
a = np.round(30 + np.random.randn(20) * 2, 1)
print(a)

# Moving average
a_avg = np.zeros_like(a)
# We're just ignoring the edge effects here
for i in range(1, len(a) - 1):
    sub = a[i - 1:i + 2]
    a_avg[i] = sub.mean()
# For the first and last point, we use the original values.
a_avg[[0, -1]] = a[[0, -1]]
print(a_avg)
```

Write a function *mov_avg* that takes an array in input and returns its 3-point moving average. You **have to use broadcasting** to compute the moving average. As in the example, use the original array values at the borders.

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```