Handling and visualizing data with Python:

Knowing how to handle files in Python is very important, and we will use them during this course. In this lab you will start with reading the different file formats, then you will create and write your own date into files. Finally, you will use data visualization to perform statistical anomaly detection.

1. Reading content of a file:

You often have to read a file from the disk that contains the dataset for your code, analysis or algorithms. Therefore, it is important to know how to open, read and close the files. You will practice some of the important tools that help with reading and the files.

1.1. **open()**: This is the Python's standard function that can open files for read and write. The open function syntax is as such:

file = open(filePath&Name, mode)

The first parameter is the file path and name to read or to create. The mode determines whether to create a file for writing or open a read only file.
We will often use open with mode='r' for opening a read only file and more='w' for writing a file. We usually use mode='rt' which means opening a textual read only file (file that contains text only) and mode='wt' which means opening a file for writing text into it. You can see the other mode types in the table below.

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open for updating (reading and writing)

Once you opened the file you can read it line by line or you can read it all at once using **file.readline()** and **file.readlines().** Often it is easier to read line by line and process each line of data as they are getting read.

raw_data = file.readline()

Once the content of the file is read, you must close the file using the **file.close().** This make the file accessible outside the Python, otherwise you won't be able to remove, delete or rename the file until Python is closed.

1.2. **Exercise:**

1.2.1. Read the file '**raw_text.txt**' into Python using **open()** and **readline()**.

1.2.2. Print first word in each line

1.3. **pandas.read_csv()**: the **read_csv()** is not a native Python function and it is part of the pandas package. This function can read comma separated value (CSV) files from the disk. The read_csv() automatically detects the location of the commas in the data and removes them. It also, allocates the data into a Pandas' DataFrame structure. Reading a file with read_csv() is very similar to the open(). You only have to provide the file name and path. You can also add some additional parameters to control whether to remove the column labels or not. You can read more about the additional options here

raw_data = pandas.read_csv(filePath&Name)

1.4. **Exercise:**

1.4.1. Read the file "sampleCSV.csv" into Python using **pandas.read_csv()**
1.4.2. Convert the data from DataFrame to NumPy using **pandas.DataFrame.to_numpy**
1.4.3. What is the maximum value of the array?
1.4.4. What is the minimum value of the array?
1.4.5. What is the shape of the data?
1.4.5.1. What is the number of the rows in the data?
1.4.5.2. What is the number of the columns in the data?

1.5. **pandas.read_excel()**: this is very similar to the read_csv() except it read excel files and not CSV.

raw_data = pandas.read_excel(filePath&Name)

1.6. **Exercise:**

1.6.1. Read the file "sampleExcel.xlsx" into Python using **pandas.read_csv()**
1.6.2. Convert the data from DataFrame to NumPy using **pandas.DataFrame.to_numpy**
1.6.3. What is the maximum value of the array?
1.6.4. What is the minimum value of the array?
1.6.5. What is the shape of the data?
1.6.6. What is the number of the rows in the data?
1.6.7. What is the number of the columns in the data?

1.7. **pickle.load()**: pickle is a very powerful Python package that is used to save and load not only data but also Python objects. Pickle writes everything to the disk as a binary file and therefore can save almost anything. This means pickle can be used to write dataset to the disk just like open(), read_csv(), read_excel(). However, datasets that are saved with pickle can only be opened in python with pickle and cannot be viewed by text editor or excel, so we avoid saving dataset with pickle. Pickle can also save Python objects such as a trained model. You use pickle to save the trained models or read trained models from the disk.
pickle.load() loads a file from the disk. For the **pickle.load()** to work we needs **open()** to open the file first. Once the file is opened with **open(filePath, mode='rb')** then **pickle.load()** can read the content of the file. Pickle. Remember that pickle reads and writes everything in binary and therefore when opening a file to read or write we use mode='rb' or mode='wb' for reading and writing respectively. This is an example of how reading a file using **pickle.load()**:

```
file = open(filePath&Name, mode='rb')
data = pickle.load(file)
file.close()
```

1.8. **Exercise:**

1.8.1. Use open() to open the pickle file "samplePickle.pkl" as binary read-only file

1.8.2. Use pickle.load() to read the content of the file

1.8.3. What is the type of the data?

1.8.4. What is the length of the data?

1.8.5. What is the first value in the data?

1.8.6. What is the last value in the data?

1.8.7. What is the maximum value in the data?

1.8.8. What is the minimum value in the data?

Summary table:

Functions	Our uses cases in this course	Advantage/disadvantage
open()	read **raw** text files only with no formatting	reads raw data/have to process raw data
pandas.read_csv()	read csv formatted data files	easy to use/only csv formatted data
pandas.read_excel()	read excel formatted data files	easy to use/only excel formatted data
pickle.load()	load binary data	saves anything/only loads pickle files
Pandas.read_json()	read json formatted data from file	

There is no need to close the file when using pandas functions, as it automatically closes the file for you.

2. Creating and writing to a file:

During the course you need to save datasets, result of computation or a trained model to the disk as a file. open() is one method of saving data on the disk, however it is extremely inefficient and we will not use it, as there are much better ways. The only use of open is to create a binary file for the pickle to write to it.

2.1. **pandas.to_csv() & pandas.to_excel() & pandas.to_json()**: these are perhaps the easiest ways of saving a processed dataset. Once we have a dataset prepared, we need to first convert it to a Pandas' DataFrame and then we can save it to the disk as a CSV file or an excel file. Assume that you have a dataset in a NumPy array format. First convert to pandas' DataFrame and then write to the disk as such:

```
pandasDataset = pandas.DataFrame(numpyDataset,)
pandasDataset.to_csv(filePath&Name, index=False, header=False)
```
or
```
pandasDataset.to_excel(filePath&Name, index=False, header=False)
```

If we do not wish to write the index or headers to the dataset we can set index=False, header=False. Adding the header and index makes the dataset more readable for someone that do not have any info about the dataset, however it is completely optional.

2.2. **Exercise:**

2.2.1. First, generate a random 2D dataset with 50 rows and 30 columns, to create this dataset, use **numpy.random.rand()**.

2.2.2. Convert the dataset from numpy array to pandas' DataFrame so that we can use to_csv() or to_excel() functions from pandas. **Do not include the column labels or index labels**.

2.2.3. Write a file that is named "**my_csv_dataset.csv**" to disk that saves the dataset. Use **to_csv()** for this task.

2.2.4. Write a file that is named "**my_excel_dataset.xlsx**" to disk that saves the dataset. Use to_excel() for this task

2.2.5. Check the disk to make sure the files are saved. The files should be saved in the working directory (the same location as your code), unless you have specified a different address.

2.3. **pickle.dump()**: pickle is used for saving any Python object to the disk, although it could potentially be used to save dataset, however, we avoid doing that in this course. There is no formatting required as pickle saves in a binary format. Using the open(mode='wb') a binary file for writing needs to be created first. Once binary file is created then pickle.dum() can write to it. An example of saving a trained model with and writing to the disk with pickle is demonstrated here:

```
file = open(filePath&Name, mode='wb')
pickle.dum(trainedModel, file)
file.close()
```

2.4. **Exercise:**

As we said, we do not recommend saving data with pickle, and we should use it to save trained models. However, for practice we will save a dataset with pickle just for learning purposes.

2.4.1. First, generate a random 2D dataset with 30 rows and 70 columns, to create this dataset, use **numpy.random.rand()**.

2.4.2. Convert the dataset from numpy array to a Python list, as we specifically want to save the dataset as a Python list in this example.

2.4.3. Create and open a binary file to write to, **name the file "my_pickle_dataset.pkl**"

2.4.4. Use pickle.dump() to save the dataset

2.4.5. Close the file. Don't forget this step. file.close() otherwise your file will not be closed and no data would be saved in it.

2.4.6. Check the disk to make the file is saved, file should be in the working directory.

3. Statistical anomaly detection:

In this section you are provided with a dataset of network traffic and your task is to find the anomalies in the network traffic. You need to complete the following steps.

3.1. Perform statistical anomaly detection on the dataset named **"traffic_anomaly_lab.csv"**

3.1.1. Read the dataset into python

3.1.2. Find Q1, Q2 and Q3

3.1.3. Calculate the interquartile range (IQR)

3.1.4. Calculate the upper and lower range of data

3.1.5. Find the anomaly indexes

3.1.6. Plot the data

3.1.7. Plot the anomalies on the data to visualize the anomalies

3.1.8. How many total anomalous datapoints are in the dataset?

3.2. Perform the similar task using Matplotlib's boxplot

3.2.1. Plot the data using the boxplot which should show you the anomalies in a different format