COURSE HANDOUT · 4TH YEAR SOFTWARE ENGINEERING

Data Mining

Practical Labs (TP)

Chapter 1

Data Collection and Exploration

PRESENTED BY	Dr. Sihem BENFRIHA
DEPARTMENT	Computer Sciences — University of Tlemcen
CONTACT	benfriha.sihem@univ-tlemcen.dz · benfriha.sihem@gmail.com

VERSION	4.0
DATE	22/04/2026

Table of Contents

Chapter 1 : Data Collection and Exploration

I

1. Objectives

- **Identify** and recall basic Python data types (integers, floats, strings) and fundamental syntax.
- **Explain** how basic data types work and interpret simple Python expressions and string operations.
- **Use** Python to write simple programs that manipulate numbers and strings.
- **Break down** simple problems into logical steps and identify errors in basic Python code.
- **Integrate** multiple basic programming elements to produce a structured and functional Python script.
- **Compare** different approaches to solving simple tasks and assess the correctness of program outputs.

2. Prerequisites

- Basic computer literacy (using a computer, keyboard, file management)
- Fundamental mathematical skills (basic arithmetic operations: addition, subtraction, multiplication, division)
- Basic logical thinking (understanding sequences of steps, simple problem-solving)

3. Prerequisite test

Exercice

Compute the following expressions manually, then verify using Python:

$10+5\times2$

$(8+2)\div5$

Exercice

Write the steps (in plain language, not code) to:

- Calculate the average of three numbers
- Display a message: “Student passed” if the average is ≥ 10, otherwise “Student failed”

Exercice

Consider the following simple instructions:

- Take a number
- Multiply it by 2
- Add 3
- Display the result

1. Apply these steps to the number 5
2. What is the final result?

4. TP1 session 1

The first notebook

```
print('This isnt a string of text')
```

```
print("This isn't a string of text")
```

```
print(3 + 5)
```

You can assign a numerical value or string of text to an object in Python, and then interact with it.

```
greeting = 'Hello and welcome to the course '
```

```
print(greeting)
```

```
a = 90
```

```
b = 12
```

```
print(a + b)
```

Variable Types in Python

Python has 4 native variable types -- integers, floats, strings, and booleans. You can determine the variable type using the built-in type() function.

Print different object types

```
print(type(3))
```

```
print(type(3.5))
```

```
print(type('a'))
```

print(type(True))

ou'll notice this doesn't print everything

type(3)

type(3.5)

Remarque : Mathematical Operations

You can perform mathematical operations on each of the different data types here.

Print mathematical operations

print(3 + 5)

print(3.4 + 4.5)

print('a' + 'b' + 'c' + 'd')

print(True + True)

You can add, subtract, multiply and divide numerical values in Python. You can also denote an exponent like this one, 3^2 , using two stars 3**2.

Rappel

 # subtraction

print(3.3 - 1.1)

 # multiplication

print(3 * 2)

 # division

print(4 / 2)

 # exponent

print(3**10)

print(True*True*False)

What happens if you multiply a and b in the following cell? Subtract them?

a = 'Hi'

b = 9

print(a*b)

What do the following errors mean? How do you begin to investigate the cause?

print(This is a course)

print(a**b)

print()

Data types

– Numbers

– Strings

– Lists

– Dictionaries

– Booleans

– Tuples

– Sets

• Comparison Operators

• if,elif, else Statements

• for Loops

• while Loops

5. Numbers and more in Python

Types of numbers

Python has various "types" of numbers (numeric literals). We'll mainly focus on integers and floating point numbers.

Integers are just whole numbers, positive or negative. For example: 2 and -2 are examples of integers.

Floating point numbers in Python are notable because they have a decimal point in them, or use an exponential (e) to define the number. For example 2.0 and -2.1 are examples of floating point numbers.

⚠ *Attention*

What just happened? Last time I checked, 3 divided by 2 is equal 1.5 not 1

The reason we get this result is because we are using Python 2. In Python 2, the / symbol performs what is known as "classic" division, this means that the decimal points are truncated (cut off). In Python 3 however, a single / performs "true" division. So you would get 1.5 if you had inputed 3/2 in Python 3.

So what do we do if we are using Python 2 to avoid this?

Specify one of the numbers to be a float:

```
# Specifying one of the numbers as a float
3.0/2
float(3)/2
# Order of Operation
(2 + 3) * (5 + 5)
```

Variable Assignments

We use a single equals sign to assign labels to variables.

The names you use when creating these labels need to follow a few rules:

1. Names can not start with a number.

2. There can be no spaces in the name, use _ instead.

3. Can't use any of these symbols :'",<>/?|\() @#$%^&*~-+

3. It's considered best practice (PEP8) that the names are lowercase.

For example:

```
# Can not start with number or special characters
name_of_var = 2
x = 2
y = 3
z = x + y
# Use object names to keep better track of what's going on in your code
my_income = 100
tax_rate = 0.1
my_taxes = my_income*tax_rate
# Show my taxes
```

6. Strings

Définition

Strings are used in Python to record text information, such as name. Strings in Python are actually a sequence, which basically means Python keeps track of every element in the string as a sequence. For example, Python understands the string "hello' to be a sequence of letters in a specific order. This means we will be able to use indexing to grab particular letters (like the first letter, or the last letter).

Creating a String

To create a string in Python you need to use either single quotes or double quotes. For example:

```
# Single word
```

'hello'

Entire phrase

'This is also a string'

We can also use double quote

"String built with double quotes"

I'm using single quotes, but will create an error'

The reason for the error above is because the single quote in I'm stopped the string. You can use combinations of double and single quotes to get the complete statement.

Printing a String

Using Jupyter notebook with just a string in a cell will automatically output strings, but the correct way to display strings in your output is by using a print function.

x = 'Hello DM Class, This is my first Python Program'

x

print(x)

num = 12

name ='Mo'

`print`('My number is: {one}, and my name is: {two}'.format(one=num,two=name))

Use curley Brakcet

`print`('My number is: {}, and my name is: {}'.format(num,name))

We can simply declare a string

'Hello World'

note that we can't output multiple strings this way

'Hello World 1'

'Hello World 2'

`print` 'Hello World 1'

`print` 'Hello World 2'

`print` 'Use \n to print a new line'

`print` '\n'

`print` 'See what I mean?'

[cf. TP1_DataMining]

7. TP1 Session 2

Numpy

Creating a numpy array

```
import numpy as np
ax = np.array([1,2,3,4,5])
print(x,id(x),len(x))
```

Specifying the type

Useful when reading a text stream directly into a numerical array

```
x=['1','2','3']
xi = np.array(x,'int')
xf = np.array(x,'float')
xs = np.array(x,'str')
print(xi,xf,xs,sep='\n')
```

Basic operations

```
x = np.array([13,24,21.2,17.6,21.7],'float')
print(x.sum(),x.mean(),x.std(),sep='\n')
```

Indexing

```
ax[1,3] #indexing
```

Slicing

```
ax[1:3,2:4]
#ax[:,2:]
```

Matrix multiplication

```
ax = np.arange(10)
ay = np.array([ax,ax])
#Scalar multiplication
ay*2
```

Selecting elements from an np array

```
x=[[0,1,2,3,4,5],[10,11,12,13,14,15],[20,21,22,23,24,25]]
ax=np.array(x,float)
np.where(ax%2==0,1,0)
#linalg, a linear algebra module
#functions dealing with polynomials, differentials, etc
```

```
import scipy
scipy.nanmean(x)
```

Pandas

installing pandas libraries

```
!pip install pandas-datareader
!pip install --upgrade html5lib==1.0b8
#There is a bug in the latest version of html5lib so install an earlier version
#Restart kernel after installing html5lib
```

Imports

```
import pandas as pd #pandas library
from pandas_datareader import data #data readers (google, html, etc.)
#The following line ensures that graphs are rendered in the notebook
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt #Plotting library
import datetime as dt #datetime for timeseries support
```

The structure of a dataframe

```
pd.DataFrame([[1,2,3],[1,2,3]],columns=['A','B','C'])
A B C
0 1 2 3
1 1 2 3
```

Accessing columns and rows

```
df = pd.DataFrame([['r1','00','01','02'],['r2','10','11','12'],['r3','20','21','22']],columns=['row_label','A','B','C'])
print(id(df))
df.set_index('row_label',inplace=True)
print(id(df))
df
```

Getting column data

```
df['B']
```

Getting row data

```
df.loc['r1']
```

Getting a row by row number

```
df.iloc[0]
```

Getting multiple columns

df[['B','A']] #Note that the column identifiers are in a list

Getting a specific cell

df.loc['r2','B']

df.loc['r2']['A']

Slicing

df.loc['r1':'r2']

df.loc['r1':'r2','B':'C']

8. TP2 Session 1

1. Data Collection and Exploration:

1. Import necessary libraries (pandas, numpy).

[Pandas](http://pandas.pydata.org) is a Python library that provides extensive means for data analysis. Data scientists often work with data stored in table formats like `.csv`, `.tsv`, or `.xlsx`. Pandas makes it very convenient to load, process, and analyze such tabular data using SQL-like queries.

Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

sns.set() # Will import Seaborn functionalities

we don't like warnings

you can comment the following 2 lines if you'd like to

import warnings

warnings.filterwarnings('ignore')

[cf.]

Méthode

2. Load the dataset into a DataFrame.

Disply all Columns

pd.options.display.max_columns = 40

autinsurance = pd.read_csv('insurance_claims.csv')

autinsurance

3. Display the first rows, number of rows/columns, and general information.

```
#autinsurance.info()
Column names may be accessed (and changed) using the `.columns` attribute as below
print("Old Column Names:\n", autinsurance.columns)
# print(autinsurance.columns)
! pip install pyjanitor
import janitor
autinsurance = autinsurance.clean_names()
print("New Janitor Column Names:\n", autinsurance.columns)
```

4. Identify data types and missing values.

```
#autinsurance.isna().sum() ## Count Filled cells
## Sum empty cells
```

5. dimensionality, features names, and feature types.

```
#dir(autinsurance)
#autinsurance.shape
# Number of rows
#len(autinsurance)
```

9. Exercice : Self Evaluation (15 min) Multiple choice Quiz

Exercice

Which of the following is a string in Python?

- ☐ 10
- ☐ 3.14
- ☐ "hello"
- ☐ True

Exercice

What is the correct way to display a message in Python?

- ☐ echo("Hello")
- ☐ print("Hello")
- ☐ display("Hello")
- ☐ show("Hello")

Exercice

What is the output of the following code?

```
print(10 + 5 * 2)
```

- ☐ 30
- ☐ 20
- ☐ 25
- ☐ 15

10. Exploratory Data Analysis (EDA) & Visualization

The `describe` method shows basic statistical characteristics of each numerical feature (`int64` and `float64` types): number of non-missing values, mean, standard deviation, range, median, 0.25 and 0.75 quartiles.

1. Use .describe() to get mean, median, mode, min, max, std.

autinsurance.describe()

2. Distribution of target variable

autinsurance.describe(include=['object'])

autinsurance['fraud_reported'].value_counts()

autinsurance['fraud_reported'].value_counts(normalize=True)

(autinsurance['fraud_reported'].value_counts().plot(

kind='bar',

figsize=(4, 2),

title='Distribution of Target Variable',

)

);

plt.show()

3. Plot histograms for numeric variables.

features = ['age', 'months_as_customer', 'capital_gains', 'capital_loss']

autinsurance[features].hist(figsize=(6, 4));

#Second

autinsurance[features].plot(kind='density', subplots=True, layout=(2, 2),

sharex=False, figsize=(6, 4));

4. Correlation matrix using Seaborn (sns.heatmap).

Let's look at the correlations among the numerical variables in our dataset.First, we will use the method [`corr()`] on a `DataFrame` that calculates the correlation between each pair of features. Then, we pass the resulting *correlation matrix* to [`heatmap()`] from `seaborn`, which renders a color-coded matrix for the provided values:

autinsurance.head(5)

```
corr_matrix = autinsurance.corr(method = 'kendall')
corr_matrix
```

Highly correlated items = not good

Low correlated items = good

#How to calculate correlation between all columns and remove highly correlated ones using pandas?

```
# import numpy as np
# # Create correlation matrix
# corr_matrix = autinsurance.corr().abs()
# # Select upper triangle of correlation matrix
# upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# # Find features with correlation greater than 0.95
# to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]
# # Drop features
# autinsurance.drop(to_drop, axis=1, inplace=True)
# autinsurance.shape
```

5. Identify possible relationships between features and the target.

```
# # Correlation with Target Variable
# autinsurance.corr()['age'].plot()
# #autinsurance.corr()[:].plot()
# #autinsurance.corr().plot()
```