Practice Session N° 1: Introduction to Python and Initiation to Data Science

This is the lab's initial document

During the lab we are going to solve the following exercises and extend this document with the solutions

Task 1. Ensuring Python version

- Check if Python is installed
  - Install Python in case it is not present
    - [https://www.python.org/](https://www.python.org/)
- Ensure the version of Python

**Expected result:**

```
>= 3.9.x
```

Task 2. Ensuring Jupyter notebook

Install Jupyter notebook using python package manager (pip) or conda

- If you prefer conda, refer to the install instructions here:
  - [https://docs.conda.io/en/latest/miniconda.html](https://docs.conda.io/en/latest/miniconda.html)
- Run the Jupyter notebook having the lab's initial IPNB file in the top level directory

**Expected result:**
Task 3. Installing the packages

Try to use the Jupyter inline commands and install the following packages:

- numpy
- pandas
- matplotlib
- ipython

Alternatively, refer to command line and install the packages using pip (or conda if you have chosen it before).

Examples:

Jupyter inline commands are prefixed with "!" in a cell:

```python
In [12]: # calling shell command <b>pip</b> main module using Jupyter inline expression
   !pip --version

pip 22.0.3 from /home/hp/python/pyenv39_its2021/lib/python3.9/site-packages/pip (python 3.9)
```

Expected result:

The mentioned packages can be imported

```python
In [15]: import numpy
       import matplotlib
```
Task 4. Solve some very simple exercises

In the following section the expected result is shown, and your objective is to write the corresponding code into Jupyter cell. Your code has to deliver exactly the expected result.

```python
In [18]: # Print the "Hello world" string
   # TODO: write your code here
   
   HelloWorld

In [19]: # Calculate 2+4
   # TODO:

   6

In [20]: # Assign the variable 'x' to 3 and print x+1
   # TODO:

   4

String in python.

In [22]: # Append 2 strings 'hello' and 'world' and print the result
   # TODO:

   hello world
```
In [23]: # Check the type of the previously declared variable 'x'
    # TODO
Out[23]: int

In [30]: # Generate list of 10 integers incrementally starting with 0
    # assign it to variable 'l' and print
    # TODO
Out[30]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [33]: # Remove '4' from the list 'l' and print
    # TODO
Out[33]: [0, 1, 2, 3, 5, 6, 7, 8, 9]

In [34]: # Remove first and last elements of the list and print
    # TODO
Out[34]: [1, 2, 3, 5, 6, 7, 8]

In [ ]: # Print only odd elements in the list 'l'
    # TODO:

In [ ]: # Print the elements of list 'l' starting from index 2 to index 5

Remember that python counts "indexes" and the order starts at 0 not at 1.

In [ ]: # Append value 11 to list 'l'

More examples

In this section the lab instructor can share more task to solve during the lab session!

In [48]: # Conditional list comprehension
    
    [i+5 for i in range(10) if i % 2 == 0]
Out[48]: [5, 7, 9, 11, 13]
# Show for, while cycles if, else expression

In [30]: # Creating new dictionary
d0 = dict()

In [36]: # Counting letters in a string using for-loop and if-condition
d0 = {}
for x in c:
    # if exists - count
    if x in d0.keys():
        d0[x] += 1
    # if not - initialize
    else:
        d0[x] = 0

In [39]: # Print first 5 letters using while-loop
i = 0
while i <= 5:
    print(c[i])
i += 1

# Import the required packages
import numpy as np
import pandas as pd

In [2]: # Python list are flexible in both types and expansion
[x for x in range(10)] + ['a'] + [0.0]

Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 0.0]

In [3]: # Numpy arrays are non-expansible and type-fixed
A = np.arange(10)
In [4]: # Init array, zero the contents
   np.zeros(10)

Out[4]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

In [5]: # Init array with no init value
   np.empty(10)

Out[5]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

**Vector operations**

In [6]: # True/False mask
   # Rest of a division by 2 and subsequent comparison to 0
   # (each single instruction is performed on entire array at once)
   mask = A % 2 == 0

In [7]: # Access using inverse mask
   A[~mask]

Out[7]: array([1, 3, 5, 7, 9])

In [8]: # Expansion of the array is done through re-init
   B = np.zeros(20)

In [9]: # ... and subsequent copy
   B[:10] = A
   B

Out[9]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 0., 0., 0., 0., 0., 0., 0.,
   0., 0., 0.])

In [10]: # ... or through horizontal stacking with empty-array
   np.hstack([A,np.zeros(10)])

Out[10]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 0., 0., 0., 0., 0., 0., 0.,
   0., 0., 0.])
In [11]: # Intecremental 50 integers, reshape to 10x5 and reverse the second axis
np.arange(50).reshape((10,5))[:,::-1]

Out[11]: array([[ 4,  3,  2,  1,  0],
              [ 9,  8,  7,  6,  5],
              [14, 13, 12, 11, 10],
              [19, 18, 17, 16, 15],
              [24, 23, 22, 21, 20],
              [29, 28, 27, 26, 25],
              [34, 33, 32, 31, 30],
              [39, 38, 37, 36, 35],
              [44, 43, 42, 41, 40],
              [49, 48, 47, 46, 45]])

Reading CSV dataset into Pandas dataframe

Here we will load the Accidents dataset into a dataframe and perform some simple actions demonstrating Pandas

- Download the accidents dataset from the first lecture materials:
- Extract the dataset into Jupyter home directory

In [12]: # WARNING Mixed type would trigger more RAM consumption and therefore -> use low_memory=False
df = pd.read_csv('dftRoadSafetyData_Accidents_2018.csv', sep=',', low_memory=False)
In [13]:

Out[13]:

<table>
<thead>
<tr>
<th>Accident_Index</th>
<th>Location_Easting_OSGR</th>
<th>Location_Northing_OSGR</th>
<th>Longitude</th>
<th>Latitude</th>
<th>Police_Force</th>
<th>Accident_Severity</th>
<th>Number_of_V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2018010080971</td>
<td>529150.0</td>
<td>182270.0</td>
<td>-0.139737</td>
<td>51.524587</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1 2018010080973</td>
<td>542020.0</td>
<td>184290.0</td>
<td>0.046471</td>
<td>51.539651</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2 2018010080974</td>
<td>531720.0</td>
<td>182910.0</td>
<td>-0.102474</td>
<td>51.529746</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3 2018010080981</td>
<td>541450.0</td>
<td>183220.0</td>
<td>0.037828</td>
<td>51.530179</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4 2018010080982</td>
<td>543580.0</td>
<td>176500.0</td>
<td>0.065781</td>
<td>51.469258</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>122630</td>
<td>2018984115718</td>
<td>307124.0</td>
<td>594145.0</td>
<td>-3.461918</td>
<td>98</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>122631</td>
<td>2018984115918</td>
<td>313325.0</td>
<td>581065.0</td>
<td>-3.360440</td>
<td>98</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>122632</td>
<td>2018984116018</td>
<td>319337.0</td>
<td>574511.0</td>
<td>-3.264352</td>
<td>98</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>122633</td>
<td>2018984116318</td>
<td>318858.0</td>
<td>566932.0</td>
<td>-3.269695</td>
<td>98</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>122634</td>
<td>2018984116418</td>
<td>316008.0</td>
<td>568771.0</td>
<td>-3.314764</td>
<td>98</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

122635 rows × 32 columns
In [14]: # Check the column names
df.columns

Out[14]: Index(['Accident_Index', 'Location_Easting_OSGR', 'Location_Northing_OSGR', 'Longitude', 'Latitude', 'Police_Force', 'Accident_Severity', 'Number_of_Vehicles', 'Number_of_Casualties', 'Date', 'Day_of_Week', 'Time', 'Local_Authority_(District)', 'Local_Authority_(Highway)', '1st_Road_Class', '1st_Road_Number', 'Road_Type', 'Speed_limit', 'Junction_Detail', 'Junction_Control', '2nd_Road_Class', '2nd_Road_Number', 'Pedestrian_Crossing-Human_Control', 'Pedestrian_Crossing-Physical_Facilities', 'Light_Conditions', 'Weather_Conditions', 'Road_Surface_Conditions', 'Special_Conditions_at_Site', 'Carriageway_Hazards', 'Urban_or_Rural_Area', 'Did_Police_Officer_Attend_Scene_of_Accident', 'LSOA_of_Accident_Location'], dtype='object')

In [15]: # Access the Number of Vehicles (just start typing the column name and hit "Tab" it will autoassist you)
   # Dictionary way:
df['Number_of_Vehicles']

   # Property way:
df.Number_of_Vehicles

   # Both are the same (just the matter of tast, property way is faster -> less typing)

Out[15]:

0    2
1    1
2    2
3    2
4    2
     ...
122630  2
122631  2
122632  3
122633  1
122634  1
Name: Number_of_Vehicles, Length: 122635, dtype: int64
In [16]: # Accessing the raw, this will return the underlying NumPy array integers with no index
df.Number_of_Vehicles.values

Out[16]: array([2, 1, 2, ..., 3, 1, 1])

In [17]: # In contrast, here we return a column of a dataframe so the index is preserved
df.Number_of_Vehicles

Out[17]:
0   2
1   1
2   2
3   2
4   2
..  ...
122630  2
122631  2
122632  3
122633  1
122634  1

Name: Number_of_Vehicles, Length: 122635, dtype: int64
In [18]: # Define new dataframe, rename columns Longitude->lon, Latitude->lat
df_cord = df.rename(columns = {'Longitude': 'lon', 'Latitude': 'lat'})[['lon', 'lat']]
df_cord

Out[18]:

<table>
<thead>
<tr>
<th></th>
<th>lon</th>
<th>lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.139737</td>
<td>51.524587</td>
</tr>
<tr>
<td>1</td>
<td>0.046471</td>
<td>51.539651</td>
</tr>
<tr>
<td>2</td>
<td>-0.102474</td>
<td>51.529746</td>
</tr>
<tr>
<td>3</td>
<td>0.037828</td>
<td>51.530179</td>
</tr>
<tr>
<td>4</td>
<td>0.065781</td>
<td>51.469258</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>122630</td>
<td>-3.461918</td>
<td>55.232746</td>
</tr>
<tr>
<td>122631</td>
<td>-3.360440</td>
<td>55.116374</td>
</tr>
<tr>
<td>122632</td>
<td>-3.264352</td>
<td>55.058510</td>
</tr>
<tr>
<td>122633</td>
<td>-3.269695</td>
<td>54.990344</td>
</tr>
<tr>
<td>122634</td>
<td>-3.314764</td>
<td>55.006392</td>
</tr>
</tbody>
</table>

122635 rows × 2 columns

Creating functions

In [19]: # Now let's create the function that would calculate an euclidean distance between 2 coordinates
   # (remember the formula ??)

   \[ \sqrt{\sum (x_1 - x_2)^2, (y_1 - y_2)^2} \]

In [20]: # Question to students
def euclidean_distance(x1,y1,x2,y2):
    # TODO: implement
    return 0
In [21]: # Get the coordinate points out of the dataset
x1, y1 = df_cord[['lon', 'lat']].values[0]
x2, y2 = df_cord[['lon', 'lat']].values[1]
x1, y1, x2, y2

Out[21]: (-0.139737, 51.524587, 0.046471, 51.539651)

In [22]: # Apply formula
euclidean_distance(x1, y1, x2, y2)

Out[22]: 0

In [23]: # What are the units returned? are these meters? kilometers?
   # (question to students)

In [24]: # Can we modify our euclidean distance to support vector operations?
   # (question to students)

In [25]: # Functions with named-arguments, and default values
   # In Python we can pass argument to a function "as a list"
   # ... and then expand the list to arguments using '*' operator
args = [1., 1., -1., -1.]
      euclidean_distance(*args)

Out[25]: 0

In [26]: # We can also collect arguments into a dictionary an pass to a function using '***' operator
kwargs = {'x1': 1.,
          'y1': 1.,
          'x2': -1.,
          'y2': -1.}
      euclidean_distance(**kwargs)

Out[26]: 0
# Recursion-vs-loop

# Infamous Fibonacci example using loop

```python
fib_length = 9
n1, n2 = 0, 1
i = 0
while i < fib_length:
    print(n1)
    n = n1 + n2
    n1 = n2
    n2 = n
    i += 1
```

```
0
1
1
2
3
5
8
13
21
```

```python
def fib_n(n):
    if n <= 1:
        return n
    else:
        return (fib_n(n-1) + fib_n(n-2))

[fib_n(number) for number in range(9)]
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21]
```

I/O routines

```python
# Open file, write one line of text
with open('text.txt', 'w') as _fd:
    _fd.writelines(['Hello!'])
    _fd.flush()
```
```python
In [3]: # Open as a text, read all the content
with open('text.txt', 'r') as _fd:
    print(_fd.read())

Hello!

Vizualization

In [20]: import matplotlib.pyplot as plt
xs = [3, 6, 3]
y = [10, 12, 11]
plt.plot(xs, y, 'r-', linewidth=3)  # thick red line
plt.scatter(xs, y, color='blue', marker='x', s=50.)  # blue x's
plt.text(xs[1], y[1], 'some text')
plt.title('my graph')
plt.ylabel('vehicle count')
plt.gca().invert_yaxis()  # flip the y axis
plt.grid()
```

![my graph](image)

In [ ]: