Dynamic Modelling Course - TEP4290: Warm-up 9

The point of this exercise is for you to practice what you have learned in the recommended video:

- https://www.youtube.com/watch?v=vmEHCJofsIg (especially between time codes 3:50 and 15:47). The rest of the video is well done as well so do not hesitate to jump to other parts to have a look.
- cheat sheets: https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf Good luck!

Import the package

To use the functionalities offered by the Pandas package, you must first load it. Usually, Pandas is imported under the alias **pd**.

Import the Pandas library under the alias "**pd**".

In [1]:
```
import pandas as pd
```

NumPy will also be used so we also import it.

In [2]:
```
import numpy as np
```

Mastering the basics

The main objects in the Pandas library are called **dataframes**, which are two-dimensional labeled data structures with columns of potentially different types.

Let's start by importing the data from the source file in csv to a dataframe.

In [6]:
```
# Extracting data from the source file
file = 'Warm_up_data.csv'
# This file should be placed in the same directory as this notebook

## Your answer here
data = pd.read_csv(file)
```

Use the **.fillna()** method to replace all the void values in "data" by 0. Remember to add the argument *inplace=True* because we wish to keep the changes.

In [7]:
```
data.fillna(0, inplace=True)
```

Print the first elements by using the method **.head()**.

In [8]:
```
data.head()
```

Out[8]:

	Time	Population	Inflow	Outflow	Stock Change	Stock	cars per 1000 people	Kilometrage per vehicle	Fue efficienc
0	1950	544951.0	1157.864047	0.0	0.0	0.0	0.0	27200.0	9.0
1	1951	555606.0	3085.651022	0.0	0.0	0.0	0.0	27200.0	9.0
2	1952	566261.0	1314.644583	0.0	0.0	0.0	0.0	27200.0	9.0
3	1953	576916.0	4843.545514	0.0	0.0	0.0	0.0	27200.0	9.0
4	1954	587571.0	3575.745174	0.0	0.0	0.0	0.0	27200.0	9.0

Make an extraction of the dataframe for all the years after 1980.

In [11]:
```
data.loc[data['Time'] >= 1980, :]
```

Out[11]:

	Time	Population	Inflow	Outflow	Stock Change	Stock	cars per 1000 people	Kilometrage per vehicle	ef
30	1980	980929.0	61008.02176	0.0	0.0	0.0	0.000000	27200.0	
31	1981	995387.0	62951.92812	0.0	0.0	0.0	0.000000	27200.0	
32	1982	1009845.0	45471.83911	0.0	0.0	0.0	0.000000	27200.0	
33	1983	1024303.0	46989.56580	0.0	0.0	0.0	0.000000	27200.0	
34	1984	1038761.0	97929.70245	0.0	0.0	0.0	0.000000	27200.0	
...	...	...	...	...	...	...	...	...	...
96	2046	1435640.2	0.00000	0.0	0.0	0.0	433.238601	15200.0	
97	2047	1430991.4	0.00000	0.0	0.0	0.0	438.300291	15150.0	
98	2048	1426342.6	0.00000	0.0	0.0	0.0	442.790609	15100.0	
99	2049	1421693.8	0.00000	0.0	0.0	0.0	446.695272	15050.0	
100	2050	1417045.0	0.00000	0.0	0.0	0.0	450.000000	15000.0	

71 rows × 13 columns

Make an extraction of the initial dataframe, this time keeping only uneven years where the number of cars per 1000 people is above 100.

In [14]:
```
data.loc[(data['Time'] % 2 != 0) & (data['cars per 1000 people'] > 100), :]
```

Out[14]:

	Time	Population	Inflow	Outflow	Stock Change	Stock	cars per 1000 people	Kilometrage per vehicle	Fue efficienc
67	2017	1410060.8	0.0	0.0	0.0	0.0	106.194133	20642.85714	6.88964
69	2019	1424123.6	0.0	0.0	0.0	0.0	133.096173	19928.57143	6.59892
71	2021	1435552.0	0.0	0.0	0.0	0.0	162.772267	19214.28571	6.30821
73	2023	1444346.0	0.0	0.0	0.0	0.0	191.967249	18500.00000	6.01750
75	2025	1453140.0	0.0	0.0	0.0	0.0	220.247722	17785.71429	5.72678
77	2027	1456871.2	0.0	0.0	0.0	0.0	247.499439	17071.42857	5.43607
79	2029	1460602.4	0.0	0.0	0.0	0.0	273.608152	16357.14286	5.14535
81	2031	1462444.6	0.0	0.0	0.0	0.0	298.459610	15950.00000	5.00000
83	2033	1462397.8	0.0	0.0	0.0	0.0	321.939566	15850.00000	5.00000
85	2035	1462351.0	0.0	0.0	0.0	0.0	343.933770	15750.00000	5.00000
87	2037	1459432.6	0.0	0.0	0.0	0.0	364.327975	15650.00000	5.00000
89	2039	1456514.2	0.0	0.0	0.0	0.0	383.007931	15550.00000	5.00000
91	2041	1452101.8	0.0	0.0	0.0	0.0	399.859389	15450.00000	5.00000
93	2043	1446195.4	0.0	0.0	0.0	0.0	414.768101	15350.00000	5.00000
95	2045	1440289.0	0.0	0.0	0.0	0.0	427.619818	15250.00000	5.00000
97	2047	1430991.4	0.0	0.0	0.0	0.0	438.300291	15150.00000	5.00000
99	2049	1421693.8	0.0	0.0	0.0	0.0	446.695272	15050.00000	5.00000

Taking baby steps in the dynamic MFA course

Now that we have pretreated the data, and looked at a few basic operations, we will start processing the data. For this, we will extract the useful data (time and inflows) into NumPy arrays.

In [32]:
```
time = data.loc[:,'Time']
arr_time = np.array(time)
inflows = data.loc[:,'Inflow']
arr_inflows = np.array(inflows)

print(arr_time)
print(arr_inflows)
```

```
[1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963
 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977
 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033
 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047
 2048 2049 2050]
[1.15786405e+03 3.08565102e+03 1.31464458e+03 4.84354551e+03
 3.57574517e+03 2.61054240e+03 6.04938817e+03 3.19706580e+03
 5.75397706e+03 6.52657867e+03 7.19272301e+02 3.13287589e+03
 3.67418264e+03 3.54452598e+03 6.04292880e+03 9.36723546e+03
 6.21229886e+03 4.26659473e+03 4.41997355e+03 4.86332309e+03
 1.07891040e+04 1.21971060e+04 2.02880069e+04 2.33739903e+04
 2.75679497e+04 3.47901283e+04 3.47669415e+04 2.84238818e+04
 3.81884417e+04 4.40034827e+04 6.10080218e+04 6.29519281e+04
 4.54718391e+04 4.69895658e+04 9.79297024e+04 2.46926421e+05
 1.89560034e+05 1.69567029e+05 2.13787842e+05 1.89120253e+05
 1.90843253e+05 2.69124686e+05 4.54130659e+05 6.50893682e+05
 6.99527911e+05 7.54783148e+05 7.87853661e+05 1.04674934e+06
 8.45216718e+05 9.97514940e+05 1.30459904e+06 1.60256501e+06
 2.32064613e+06 3.04394180e+06 2.90134964e+06 4.35522219e+06
 5.33073384e+06 6.30587946e+06 7.06752064e+06 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00]
```

Write a function that calculates the outflows based on the inflows given above assuming a sharp lifetime.

In [33]:
```
def compute_outflows(inflows, lifetime):
    outflows = []
    '''
    Write your funtion that depends on the inflows and lifetime here and stores the
    To save every iteration in this list, use the meethod outflows.append(**tell py
    '''
    for i in range(len(inflows)):
        if i < lifetime:
            outflows.append(0)
        else:
            outflows.append(inflows[i-lifetime])

    return outflows
```

Using the function you wrote, calculate the outflows for a lifetime of 4 years.

In [36]:
```python
lifetime = 4
outflows = compute_outflows(inflows, lifetime)
print (outflows)
```

```
[0, 0, 0, 0, np.float64(1157.864047), np.float64(3085.651022), np.float64(1314.64458
3), np.float64(4843.545514), np.float64(3575.745174), np.float64(2610.542397), np.fl
oat64(6049.388165), np.float64(3197.065801), np.float64(5753.977063), np.float64(652
6.578667), np.float64(719.2723009), np.float64(3132.875888), np.float64(3674.18264
4), np.float64(3544.525976), np.float64(6042.928801), np.float64(9367.235464), np.fl
oat64(6212.29886), np.float64(4266.59473), np.float64(4419.973554), np.float64(4863.
323094), np.float64(10789.10402), np.float64(12197.10599), np.float64(20288.0069), n
p.float64(23373.99032), np.float64(27567.94971), np.float64(34790.1283), np.float64
(34766.9415), np.float64(28423.88177), np.float64(38188.44172), np.float64(44003.482
69), np.float64(61008.02176), np.float64(62951.92812), np.float64(45471.83911), np.f
loat64(46989.5658), np.float64(97929.70245), np.float64(246926.4207), np.float64(189
560.0339), np.float64(169567.0292), np.float64(213787.8421), np.float64(189120.252
9), np.float64(190843.2527), np.float64(269124.6856), np.float64(454130.659), np.flo
at64(650893.6825), np.float64(699527.9112), np.float64(754783.1479), np.float64(7878
53.6607), np.float64(1046749.342), np.float64(845216.7179), np.float64(997514.9398),
np.float64(1304599.044), np.float64(1602565.015), np.float64(2320646.128), np.float6
4(3043941.799), np.float64(2901349.637), np.float64(4355222.189), np.float64(533073
3.844), np.float64(6305879.464), np.float64(7067520.637), np.float64(0.0), np.float6
4(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float
64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.floa
t64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.flo
at64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.fl
oat64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.f
loat64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.
float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), np.float64(0.0), n
p.float64(0.0), np.float64(0.0)]
```

Convert outflows list to NumPy array.

In [37]:
```python
arr_outflows = np.array(outflows)
arr_outflows
```

```
Out[37]: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               1.15786405e+03, 3.08565102e+03, 1.31464458e+03, 4.84354551e+03,
               3.57574517e+03, 2.61054240e+03, 6.04938817e+03, 3.19706580e+03,
               5.75397706e+03, 6.52657867e+03, 7.19272301e+02, 3.13287589e+03,
               3.67418264e+03, 3.54452598e+03, 6.04292880e+03, 9.36723546e+03,
               6.21229886e+03, 4.26659473e+03, 4.41997355e+03, 4.86332309e+03,
               1.07891040e+04, 1.21971060e+04, 2.02880069e+04, 2.33739903e+04,
               2.75679497e+04, 3.47901283e+04, 3.47669415e+04, 2.84238818e+04,
               3.81884417e+04, 4.40034827e+04, 6.10080218e+04, 6.29519281e+04,
               4.54718391e+04, 4.69895658e+04, 9.79297024e+04, 2.46926421e+05,
               1.89560034e+05, 1.69567029e+05, 2.13787842e+05, 1.89120253e+05,
               1.90843253e+05, 2.69124686e+05, 4.54130659e+05, 6.50893682e+05,
               6.99527911e+05, 7.54783148e+05, 7.87853661e+05, 1.04674934e+06,
               8.45216718e+05, 9.97514940e+05, 1.30459904e+06, 1.60256501e+06,
               2.32064613e+06, 3.04394180e+06, 2.90134964e+06, 4.35522219e+06,
               5.33073384e+06, 6.30587946e+06, 7.06752064e+06, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
               0.00000000e+00])
```

```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```

These results will be plotted in the Warm-up dedicated to visualization.

```
In [ ]:
```