GROUP WORK PROJECT # 2 MScFE 600: FINANCIAL DATA
GROUP NUMBER: 69

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Alper Ülkü	Turkey	alperulku1970@gmail.com	
Reggie Pantig	Philippines	reggie.pantig04@gmail.com	
Tchinda Tanang Romuald	Cameroon	romualdtr@gmail.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).	
Team member 1	Alper Ülkü
Team member 2	Reggie Pantig
Team member 3	Tchinda Tanang Romuald

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed. **Note:** You may be required to provide proof of your outreach to non-contributing members upon request.

STEP1-2_TutorialOnPythonDataframes_AlperUlku

July 18, 2022

1 World Quant University - MsFE Program 2022

1.1 MsFE600 Financial Data Course

1.1.1 Group Work 2 Week 5

1.1.2 A Tutorial on Data Manipulation with Dataframes in Python

By Alper Ülkü

Let us try to understand what a dataframe is and what are practical uses of dataframes in Python.

A Data frame is a two-dimensional data structure, where data is aligned in a tabular fashion in rows and columns.(Ref-1)

Fundamentally dataframes have following properties:

- Potentially columns of a Dataframes can be of different types
- Dataframes are Mutable i.e. (Python mutability refers to being able to change an object. Simply put, a mutable object can be changed, but an immutable object cannot.) (2)
- Dataframes have labeled axes (rows and columns)
- Arithmetic operations can be performed on rows and columns

We need only the pandas module to define and operate a Dataframe so let's add it:

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

We can define columns of dataframe as follows just like we do in a Python dictionary.

[2]:
```
weather_data={'day':
 ↪['01-01-2021','02-01-2021','03-01-2021','04-01-2021','05-01-2021','06-01-2021'],
              'Temperature':[32,35,28,24,32,32],
              'windspeed':[6,7,2,7,4,2],
              'event':['rain','sunny','snow','snow','rain','sunny']}
```

and then we convert this dictionary to a dataframe by the following operation:

[3]:
```
df=pd.DataFrame(weather_data)
```

Now we can print this to see its content clearly

[4]:
```
print(df)
```

```
          day  Temperature  windspeed  event
0  01-01-2021           32          6   rain
1  02-01-2021           35          7  sunny
2  03-01-2021           28          2   snow
3  04-01-2021           24          7   snow
4  05-01-2021           32          4   rain
5  06-01-2021           32          2  sunny
```

First operation we can do to investigate the content of the dataframe is the **shape** command, as it shows the numbers of row and number of column within the dataframe.

[5]:
```
df.shape
```

[5]:
```
(6, 4)
```

The shape method returns a **tuple** and we can the assign the values to the tuple as rows and colums

[6]:
```
rows,columns=df.shape
rows
```

[6]:
```
6
```

[7]:
```
columns
```

[7]:
```
4
```

If we like to print only top 5 items of the Dataframe:

[8]:
```
df.head()
```

[8]:
```
          day  Temperature  windspeed  event
0  01-01-2021           32          6   rain
1  02-01-2021           35          7  sunny
2  03-01-2021           28          2   snow
3  04-01-2021           24          7   snow
4  05-01-2021           32          4   rain
```

or only the top 2 items:

[9]:
```
df.head(2)
```

[9]:
```
          day  Temperature  windspeed  event
0  01-01-2021           32          6   rain
1  02-01-2021           35          7  sunny
```

or only the last 3 items:

[10]:
```
df.tail(3)
```

[10]:
```
          day  Temperature  windspeed  event
3  04-01-2021           24          7   snow
4  05-01-2021           32          4   rain
5  06-01-2021           32          2  sunny
```

We can use slicing method to access rows from the dataframe:

[11]:
```
print(df[2:5])
```

```
          day  Temperature  windspeed event
2  03-01-2021           28          2  snow
3  04-01-2021           24          7  snow
4  05-01-2021           32          4  rain
```

or the first 4 rows can be printed as follows:

[12]:
```
print(df[:4]) #from first to the last row
```

```
          day  Temperature  windspeed  event
0  01-01-2021           32          6   rain
1  02-01-2021           35          7  sunny
2  03-01-2021           28          2   snow
3  04-01-2021           24          7   snow
```

If we need to call the columns we need to use the **columns** keyword, just like that:

[13]:
```
df.columns
```

[13]:
```
Index(['day', 'Temperature', 'windspeed', 'event'], dtype='object')
```

To print the individual columns; we either write its column name following a '.' after the dataframe object:

[14]:
```
df.day
```

[14]:
```
0    01-01-2021
1    02-01-2021
2    03-01-2021
3    04-01-2021
4    05-01-2021
5    06-01-2021
Name: day, dtype: object
```

Or, we open up a squared bracket and type the column name within single or double quotes (' ' or " "):

[15]:
```
df['event']
```

[15]:
```
0     rain
1    sunny
2     snow
3     snow
4     rain
5    sunny
Name: event, dtype: object
```

Now, we can easily check the type of each column:

[16]:
```
type(df['windspeed'])
```

[16]:
```
pandas.core.series.Series
```

And we can print only the required columns as well like below:

[17]:
```
df[['event','day']]
```

[17]:

	event	day
0	rain	01-01-2021
1	sunny	02-01-2021
2	snow	03-01-2021
3	snow	04-01-2021
4	rain	05-01-2021
5	sunny	06-01-2021

Let us have some statistical operations with dataframes

In order to find the maximum temperature we can use the max() function:

[18]:
```
df["Temperature"].max()
```

[18]:
```
35
```

and average temperature can be found with use of the mean() function:

[19]:
```
df["Temperature"].mean()
```

[19]:
```
30.5
```

and minimum temperature can be found with use of the min() function:

[20]:
```
df["Temperature"].min()
```

[20]:
```
24
```

and standard deviation of temperatures can be found by using std() function:

[21]:
```
df["Temperature"].std()
```

[21]:
```
3.8858718455450894
```

In order to have all possible statistics, together with the percentiles, of a dataframe, describe() function can be used as follows:

[22]:
```
df.describe()
```

[22]:

	Temperature	windspeed
count	6.000000	6.000000
mean	30.500000	4.666667
std	3.885872	2.338090
min	24.000000	2.000000
25%	29.000000	2.500000
50%	32.000000	5.000000
75%	32.000000	6.750000
max	35.000000	7.000000

If we like to filter out data with numerical criteria on a column, a fine example would be:

[23]:
```
df[df.Temperature>=32]
```

[23]:

	day	Temperature	windspeed	event
0	01-01-2021	32	6	rain
1	02-01-2021	35	7	sunny
4	05-01-2021	32	4	rain
5	06-01-2021	32	2	sunny

In order to filter out the row with maximum temperature we can use statements like:

[24]:
```
df[df.Temperature==df["Temperature"].max()]
```

[24]:

	day	Temperature	windspeed	event
1	02-01-2021	35	7	sunny

As dataframes are most basic and compact form of databases, they may also have indexes, i.e. any kind of search can be done according to a certain column which called the **index column.** This index can be viewed and set by commands **index** and **set_index()** respectively.

[25]:
```
df.index
```

[25]:
```
RangeIndex(start=0, stop=6, step=1)
```

[26]:
```
df.set_index("day", inplace=True)
```

After setting the index of a Dataframe, printing the Dataframe yields a little bit of a difference, where **day** column is emphasized as the index column.

[27]:
```
df
```

[27]:
```
            Temperature  windspeed  event
day
01-01-2021           32          6   rain
02-01-2021           35          7  sunny
03-01-2021           28          2   snow
04-01-2021           24          7   snow
05-01-2021           32          4   rain
06-01-2021           32          2  sunny
```

Conversely, if we like to get rid of the index information or reset the index after manipulation of dataframe we can use reset_index() function as:

[28]:
```
df.reset_index(inplace=True)
df
```

[28]:
```
          day  Temperature  windspeed  event
0  01-01-2021           32          6   rain
1  02-01-2021           35          7  sunny
2  03-01-2021           28          2   snow
3  04-01-2021           24          7   snow
4  05-01-2021           32          4   rain
5  06-01-2021           32          2  sunny
```

Here instead of writing **df = df.reset_index()**, *inplace=True* is used in paranthesis.

If like to locate the data, some kind of indexing should be used. When we need to find data according to **date** as index, (that we specified in **day** column), we use **loc** function with the index, embraced in square brackets and quote marks, provided that we have set the index, like this:

[29]:
```
df.set_index("day", inplace=True)
df.loc['01-01-2021']
```

[29]:
```
Temperature      32
windspeed         6
event          rain
Name: 01-01-2021, dtype: object
```

If we want to have a summary of object within our dataframe we need to use info() function as follows:

[30]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6 entries, 01-01-2021 to 06-01-2021
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Temperature  6 non-null      int64
 1   windspeed    6 non-null      int64
```

```
 2   event        6 non-null      object
dtypes: int64(2), object(1)
memory usage: 364.0+ bytes
```

Dataframes do have full flexibility, when we like to change the name of column, column name **event** is replaced by name **weather**, as follows:

[31]:
```
txt_df=df.rename(columns={"event":"weather"})
print(txt_df)
```

```
            Temperature  windspeed weather
day
01-01-2021           32          6    rain
02-01-2021           35          7   sunny
03-01-2021           28          2    snow
04-01-2021           24          7    snow
05-01-2021           32          4    rain
06-01-2021           32          2   sunny
```

Insertion of a new column is as easy as below, contents of which may be calculated from other column contents:

[32]:
```
df["Temp x Wind"]=df["Temperature"]*df["windspeed"]
print(df)
```

```
            Temperature  windspeed  event  Temp x Wind
day
01-01-2021           32          6   rain          192
02-01-2021           35          7  sunny          245
03-01-2021           28          2   snow           56
04-01-2021           24          7   snow          168
05-01-2021           32          4   rain          128
06-01-2021           32          2  sunny           64
```

When counts of the data are required, value_counts() function can be used as follows:

[33]:
```
df["event"].value_counts()
```

[33]:
```
rain     2
sunny    2
snow     2
Name: event, dtype: int64
```

A simple database query can be performed either by using query() function;

[34]:
```
df.query('event=="rain" and Temperature==32')
```

[34]:
```
            Temperature  windspeed event  Temp x Wind
day
01-01-2021           32          6  rain          192
```

```
05-01-2021           32           4   rain           128
```

or by using following command, yielding same output:

[35]:
```
df[(df['event']=="rain")|(df['Temperature']=='32')]
```

[35]:
```
            Temperature  windspeed event  Temp x Wind
day
01-01-2021           32           6  rain           192
05-01-2021           32           4  rain           128
```

Another database lookup can be done by isin() function, checks whether the data can be located within the dataframe, as follows:

[36]:
```
x=["rain","snow"]
df[df.event.isin(x)]
```

[36]:
```
            Temperature  windspeed event  Temp x Wind
day
01-01-2021           32           6  rain           192
03-01-2021           28           2  snow            56
04-01-2021           24           7  snow           168
05-01-2021           32           4  rain           128
```

Finally we sooner or later need to save our data to disk, we can do this with to_csv() function or read_csv() function:

[37]:
```
df.to_csv("weather.csv")
```

[38]:
```
df2 = pd.read_csv("weather.csv")
```

[39]:
```
df2
```

[39]:
```
          day  Temperature  windspeed  event  Temp x Wind
0  01-01-2021           32          6   rain           192
1  02-01-2021           35          7  sunny           245
2  03-01-2021           28          2   snow            56
3  04-01-2021           24          7   snow           168
4  05-01-2021           32          4   rain           128
5  06-01-2021           32          2  sunny            64
```

That finalized our tutorial on how to use Dataframes in Python.

References: 1. TutorialsPointWebPage 2. CodingGemWebPage

[ ]:
```

```

STEP1-2_TutorialOnPythonDictionary_ReggiePantig

July 18, 2022

World Quant University - MsFE Program 2022 \ MsFE600 Financial Data Course \ **Introduction to Python Dictionary** \ by **Reggie C. Pantig**

A dictionary is a structure that maps arbitrary keys to a set of arbitrary values. A dictionary in Python has the following properties:
(1) mutable or subjected to change;
(2) dynamic for they can grow and shrink as prefered;
(3) It can be nested. This means that a dictionary can contain a dictionary (or lists).

Let's study through demonstrations the basics of dictionaries

An empty dictionary

[1]:
```
my_dict = {}
print(my_dict)
```

```
{}
```

Let's add some *keys* and *values* in the dictionary and this can be specified through the curly brackets:

[2]:
```
my_dict = {'A':1,'B':2,'C':3,'D':4}
print(my_dict)
```

```
{'A': 1, 'B': 2, 'C': 3, 'D': 4}
```

In the above output, A is the key, and 1 is the element. Let's now access some elements

[3]:
```
print(my_dict['A'])
```

```
1
```

as expected. Another method to do this is by

[4]:
```
print(my_dict.get('B'))
```

```
2
```

We can create another dictionary in a different way, using a .zip() function. If we have keys and values as a list,

[5]:
```
keys = ['A','B','C','D']
values = [1,2,3,4]
dictionary = dict(zip(keys, values))
print(dictionary)
```

```
{'A': 1, 'B': 2, 'C': 3, 'D': 4}
```

To call several keys, values, and key-value pairs respectively,

[6]:
```
print(my_dict.keys())
print(my_dict.values())
print(my_dict.items())
```

```
dict_keys(['A', 'B', 'C', 'D'])
dict_values([1, 2, 3, 4])
dict_items([('A', 1), ('B', 2), ('C', 3), ('D', 4)])
```

Suppose we want to add a new definition to our dictionary. We can do this by

[7]:
```
my_dict['E'] = 2
print(my_dict)
```

```
{'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 2}
```

and if we have a wrong definition, we can alter it through the code

[8]:
```
my_dict['D'] = 5
print(my_dict)
```

```
{'A': 1, 'B': 2, 'C': 3, 'D': 5, 'E': 2}
```

If we can add a key-value pair, we can also erase it through the .pop() function. Suppose I want to erase 'D':5 pair,

[9]:
```
a = my_dict.pop('D')
print('Value:', a)
print('Dictionary:', my_dict)
```

```
Value: 5
Dictionary: {'A': 1, 'B': 2, 'C': 3, 'E': 2}
```

[10]:
```
print(my_dict)
```

```
{'A': 1, 'B': 2, 'C': 3, 'E': 2}
```

If we want to clear the whole dictionary, we just simply do

[11]:
```
my_dict.clear()
print(my_dict)
```

```
{}
```

So far, what we have considered above is just a one-to-one definition of the keys. It is possible to assign a key to several values.

[12]:
```
new_dict = {'A':[1,2,3],'B':[4,5,6]}
print(new_dict)
```

```
{'A': [1, 2, 3], 'B': [4, 5, 6]}
```

Dictionary is useful since it can be used in conjunction with pandas (see Alper's tutorial for more details). Such a dataframe can be used with seaborn or matplotlib to produce plots. For example,

[13]:
```
import pandas as pd
df=pd.DataFrame(new_dict)
print(df)
```

```
   A  B
0  1  4
1  2  5
2  3  6
```

Manually constructing a python dictionary is useful for low numbers of columns and rows for data analysis. However, we would prefer reading data through a .csv file when very large data is involved or importing data from the web.

References

[1] Programiz
[2] Python Dictionaries
[3] RealPython
[4] VanderPlas, Jake. Python data science handbook: Essential tools for working with data. " O'Reilly Media, Inc.", 2016.
[5] Johansson, Robert. Numerical Python: A Practical Techniques Approach for Industry. Apress, 2015.

MScFE 600 FINANCIAL DATA
Group Work Project # 2
DATA STRUCTURES

Lists:

1- Definition

In Python, a list is a data structure that is used to store data of different data types in a sequential manner. Any element of a list can be accessed by using his Address or Index. Unlike other programming languages, python Indexes start by zero (0). Notice that Python also use negative Index to move from the end of the list.

2- Creation

A list is created by the code: my_list = ['element1','element2',...].

Examples:

- *my_list1 = [ ]* create an empty list named my_list1
- *my_list2 = ['name','age','street',45,12]* create a list with five elements, *'name', 'age', 'street', 45, and 12*

3- Subset and modification

- Element3 = my_list [2], Extracts the third element from the list named my_list: my_list [0] for the first element of the list; my_list [-1] for the last element.
- my_list.insert(1, 'insert_hous') add element '1' (second position because the first one is '0').
- del my_list2[3] delete element at index 3
- my_list2.remove('element') #remove the element 'element' from the list.

The len() function returns to us the length of the list.

The index() function finds the index value of value passed where it has been encountered the first time.

The count() function finds the count of the value passed to it.

The sorted() and sort() functions do the same thing, that is to sort the values of the list.

The sorted() has a return-type whereas the sort() modifies the original list.

ADVANTAGES AND DESADVANTAGES OF LISTS

ADVANTAGES OF LISTS:

1- **Mutable,**
2- **Dynamic,**
3- **Easy to use,**
4- **Easy to create.**

DISADVANTAGES OF LISTS:

1- **Use much memory than tuple,**
2- **Can be modify by mistake.**

study aid for best practices with Lists

Definition

A list is a data structure that is used to store data of different data types in a sequential manner. Any element of a list can be accessed by using his Address or Index. Unlike other programmiing languages, python Indexes start by zero (0). Notice that Python also use negative Index to move from the end of the list.

Creation

A list is created by the code: my_list = ['element1','element2',...]

Exemple

Entrée [93]:

```
my_list1 = [] #create empty list named my_list1
my_list2 = ['name','age','street',45,12] #create a list with five elements
```

```
## Others operations on lists
```

```
# One can extract any element from the list
```

Entrée [78]:

```
my_list2[2]
```

Out[78]:

'street'

Entrée [79]:

```
my_list2[0]
```

Out[79]:

'name'

Entrée [80]:

```
my_list2[-1] # for the last element
```

Out[80]:

12

Entrée [81]:

```
my_list2[-2]
```

Out[81]:

45

```
Unlique tupples, it is possible to add or remove an element from a list at a precise position
```

Entrée [82]:

```
my_list2.insert(1, 'insert_hous') #add element 1 (second position because the first one
print(my_list2)
```

['name', 'insert_hous', 'age', 'street', 45, 12]

Entrée [83]:

```
del my_list2[3] # delete element at index 3
print(my_list2)
```

['name', 'insert_hous', 'age', 45, 12]

```
There are many other funtion
```

Entrée [84]:

```
my_list2.remove('age') #remove the element 'street'
print(my_list2)
```

['name', 'insert_hous', 45, 12]

Entrée [85]:

```
b = my_list2.pop(1) #pop element 1
print('Popped Element: ', b, ' List remaining: ', my_list2)
```

Popped Element: insert_hous List remaining: ['name', 45, 12]

Entrée [ ]:

```

```

```
- The len() function returns to us the length of the list.
- The index() function finds the index value of value passed where it has been encountered the first time.
- The count() function finds the count of the value passed to it.
- The sorted() and sort() functions do the same thing, that is to sort the values of the list.
- he sorted() has a return-type whereas the sort() modifies the original list.
```

Entrée [86]:

```
my_list3 = [2, 14, 'm', 14, 102, 'ln10',14]
```

Entrée [87]:

```
print(len(my_list3)) #find length of list
```

```
7
```

Entrée [88]:

```
print(my_list3.index(14)) #find index of element that occurs first
```

```
1
```

Entrée [89]:

```
print(my_list3.count(14)) #find count of the element
```

```
3
```

Entrée [90]:

```
my_list4 = [2, 14, 14, 102, 0 ,14]
print(sorted(my_list4)) #print sorted list but not change original
```

```
[0, 2, 14, 14, 14, 102]
```

Entrée [91]:

```
my_list4.sort(reverse=True) #sort original list
print(my_list4)
```

```
[102, 14, 14, 14, 2, 0]
```

Entrée [92]:

```
for k in my_list3: #access elements one by one
    print(k)

print(my_list3[0:2]) #access elements from 0 to 1 and exclude 2
print(my_list3[::-1]) #access elements in reverse
```

```
2
14
m
14
102
ln10
14
[2, 14]
[14, 'ln10', 102, 14, 'm', 14, 2]
```