Department of Computer Technology and Information Systems

CTIS264 – Computer Algorithms

Spring 2019 - 2020

Lab Guide 8 - Week 11

Instructor : Füsun YÜRÜTEN **Assistant** : Leyla SEZER
OBJECTIVE: Pandas Module

About **Pandas** Module**:**

- Data structures used in pandas include;
  - **Series**
  - **Data Frames**

SERIES

- A series is a one-dimensional array like object.
  - May hold any data type (float, int, string)
  - It is a data structure with two arrays, one contains the indices (labels) and the other contains the data.

```
>>> import pandas as pd
>>> series1 = pd.Series([11,28,72,3,5,8])
>>> print(series1)
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
```

- We can also define our own indexes, instead of using the default pandas indexing. This allows us to give meaning to the data.

```
>>> fruits = ['apple','orange','cherry','pear']
>>> quantities = [20,33,52,10]
>>> s = pd.Series(quantities,fruits)
>>> print(s)
apple     20
orange    33
cherry    52
pear      10
dtype: int64
```

- We can access elements of a series using their indexes using square braces;
  - **`print(s['cherry'])`** returns **52**

- Adding Two Series data; you can use + operator to combine two series with the given indexes. If an index appears in only one of the series, it will add that index to the new series, with a **NaN** value.

```
>>> fruits1 = ['peaches','oranges','cherries','pears']
>>> fruits2 = ['raspberries','oranges','cherries','pears']
>>> s1 = pd.Series([20,33,52,10],index=fruits1)
>>> s2 = pd.Series([17,13,31,32],index=fruits2)
>>> print(s1+s2)
cherries       83.0
oranges        46.0
peaches         NaN
pears          42.0
raspberries     NaN
dtype: float64
```

- To remove null values from a series, use the function; `dropna()`, to fill missing values use `fillna()`.
- You can also make slicing with series.

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
quantities = [20, 33, 52, 10]
S = pd.Series(quantities, index=fruits)
print(S)
print(S['apples':'cherries'])
print(S[0:2])
```

```
apples      20
oranges     33
cherries    52
pears       10
dtype: int64

apples      20
oranges     33
cherries    52
dtype: int64
apples     20
oranges    33
dtype: int64
```

- Like we did with numpy, you can also select elements of a series using relational and logical expressions.
    - Operators in PANDAS: & (and), | (or), ~ (not), != (not equal).

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
quantities = [20, 33, 52, 10]
S = pd.Series(quantities, index=fruits)

print('\nFruits between 30 and 40')
print(S[(S > 30) & (S < 40)])

print('\nFruits not between (20 and 50]')
print(S[(S <= 20) | (S > 50)])

print('\nFruits not equal to 10')
print(S[S!= 10])
```

```
Fruits between 30 and 40
oranges    33
dtype: int64

Fruits not between (20 and 50]
apples      20
cherries    52
pears       10
dtype: int64

Fruits not equal to 10
apples      20
oranges     33
cherries    52
dtype: int64
```

- We can remove elements of a series by using the drop() function;
    - `S.drop(labels=2)` or `S.drop(lables = ['pears','apples'])` assume S is a series.

DATA FRAMES

- Data Frames are tables, like a 'Sheet' in an Excel document.
- Data Frames contain an ordered collection of columns, each with its own data type. Each column must have the same type, but different columns may store different types.
- A Data Frame can be viewed as multiple Series concatenated together.
- Data Frames have a dictionary-like structure, where a dictionary maps a key to a value, a **DataFrame** maps a **column name** to a **Series**.

- Creating Data Frame;

```
years = range(2015,2019)

product1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index = years)
product2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index = years)
product3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index = years)

product_data = pd.DataFrame({'P1':product1,'P2':product2,'P3':product3})
print(product_data)
```

	P1	P2	P3
2015	2409.14	1203.45	3412.12
2016	2941.01	3441.62	3491.16
2017	3496.83	3007.83	3457.19
2018	3119.55	3619.53	1963.10

- Data Frames can also be created using a dictionary of lists. The keys in the dictionary will act as the column labels, and the value associated with each key is the list data.

```
import pandas as pd

cities = {"name": ["London", "Berlin", "Madrid", "Rome",
                   "Paris", "Vienna", "Bucharest", "Hamburg",
                   "Budapest", "Warsaw", "Barcelona",
                   "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1740119, 1602386, 1493900,
                         1350680],
          "country": ["England", "Germany", "Spain", "Italy",
                      "France", "Austria", "Romania",
                      "Germany", "Hungary", "Poland", "Spain",
                      "Germany", "Italy"]}

city_frame = pd.DataFrame(cities)

print(city_frame)
```

	name	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

- DataFrames can be thought of like enhanced two-dimensional array. We can examine the raw data in a DataFrame using the values attribute.

```
import pandas as pd
years = range(2015,2019)

product1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index = years)
product2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index = years)
product3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index = years)

product_data = pd.DataFrame({'P1':product1,'P2':product2,'P3':product3})
print(product_data)

print(product_data.values[0])
print(product_data.values[1,0])
product_data.values[1,0]=999

print(product_data.values[1,0])

print(product_data['P1'])
print(product_data[['P1','P3']])
```

Output:

```
        P1       P2       P3
2015  2409.14  1203.45  3412.12
2016  2941.01  3441.62  3491.16
2017  3496.83  3007.83  3457.19
2018  3119.55  3619.53  1963.10



[2409.14 1203.45 3412.12]
2941.01
999.0

2015    2409.14
2016     999.00
2017    3496.83
2018    3119.55
Name: P1, dtype: float64
```

```
            P1       P3
2015  2409.14  3412.12
2016   999.00  3491.16
2017  3496.83  3457.19
2018  3119.55  1963.10
```

- Adding Columns to a DataFrame, there are some examples;

Constant scalar value:

```
In [107]: product_data['P4'] = 10

In [108]: product_data
Out[108]:
            P1       P2       P3  P4
2015  2409.14  1203.45  3412.12  10
2016  2941.01  3441.62  3491.16  10
2017  3496.83  3007.83  3457.19  10
2018  3119.55  3619.53  1963.10  10
```

Tuple of column values:

```
In [109]: product_data['P5'] = (2000.0,3000.0,4000.0,5000.0)

In [110]: product_data
Out[110]:
            P1       P2       P3  P4      P5
2015  2409.14  1203.45  3412.12  10  2000.0
2016  2941.01  3441.62  3491.16  10  3000.0
2017  3496.83  3007.83  3457.19  10  4000.0
2018  3119.55  3619.53  1963.10  10  5000.0
```

List of column values:

```
In [111]: product_data['P6'] = [1000.0,6000.0,2000.0,3500.0]

In [112]: product_data
Out[112]:
            P1       P2       P3  P4      P5      P6
2015  2409.14  1203.45  3412.12  10  2000.0  1000.0
2016  2941.01  3441.62  3491.16  10  3000.0  6000.0
2017  3496.83  3007.83  3457.19  10  4000.0  2000.0
2018  3119.55  3619.53  1963.10  10  5000.0  3500.0
```

Using functions:

```
In [115]: product_data['TOTALS'] = np.sum(product_data, axis=1)

In [116]: product_data
Out[116]:
            P1       P2       P3  P4      P5      P6      P7    TOTALS
2015  2409.14  1203.45  3412.12  10  2000.0  1000.0  1500.0  11534.71
2016  2941.01  3441.62  3491.16  10  3000.0  6000.0  9000.0  27883.79
2017  3496.83  3007.83  3457.19  10  4000.0  2000.0  3000.0  18971.85
2018  3119.55  3619.53  1963.10  10  5000.0  3500.0  5250.0  22462.18
```

Using values from existing columns:

```
In [113]: product_data['P7'] = product_data['P6'] * 1.5

In [114]: product_data
Out[114]:
            P1       P2       P3  P4      P5      P6      P7
2015  2409.14  1203.45  3412.12  10  2000.0  1000.0  1500.0
2016  2941.01  3441.62  3491.16  10  3000.0  6000.0  9000.0
2017  3496.83  3007.83  3457.19  10  4000.0  2000.0  3000.0
2018  3119.55  3619.53  1963.10  10  5000.0  3500.0  5250.0
```

- To delete rows, the **drop**() function can be used.

```
In [197]: product_data
Out[197]:
          P1       P2       P3
2015  2409.14  1203.45  3412.12
2016  2941.01  3441.62  3491.16
2017  3496.83  3007.83  3457.19
2018  3119.55  3619.53  1963.10

In [198]: product_data.drop([2017], axis = 0)
Out[198]:
          P1       P2       P3
2015  2409.14  1203.45  3412.12
2016  2941.01  3441.62  3491.16
2018  3119.55  3619.53  1963.10

In [199]: product_data
Out[199]:
          P1       P2       P3
2015  2409.14  1203.45  3412.12
2016  2941.01  3441.62  3491.16
2017  3496.83  3007.83  3457.19
2018  3119.55  3619.53  1963.10
```

```
In [200]: product_data = product_data.drop([2015,2018], axis = 0)

In [201]: product_data
Out[201]:
          P1       P2       P3
2016  2941.01  3441.62  3491.16
2017  3496.83  3007.83  3457.19
```

```
In [7]: df.drop( ['P1','P3'], axis = 1 )
Out[7]:
            P2
2015  1203.45
2016  3441.62
2017  3007.83
2018  3619.53
```

- Pandas provides us functionality to import data from csv (comma-separated values) and xlsx (Excel) files directly into DataFrames.

```
#import data from files
area_df = pd.read_excel('world_area.xlsx')
pop_df = pd.read_excel('world_pop.xlsx')

#display dataframes
print(area_df)
print('Type of area_df:',type(area_df))
print(pop_df)
print('Type of pop_df:',type(pop_df))
```

```
        Country   Area(sqm)
0         China         NaN
1         India   2973190.0
2          U.S.   9147420.0
3     Indonesia   1811570.0
4        Brazil   8358140.0
5      Pakistan    770880.0
6       Nigeria    910770.0
7    Bangladesh    130170.0
8        Russia  16376870.0
9        Mexico   1943950.0
10        Japan    364555.0
11     Ethiopia   1000000.0
12  Philippines    298170.0
13        Egypt    995450.0
14     Viet-Nam         NaN
15      Germany    348560.0
16         Iran   1628550.0
17       Turkey    769630.0
18     Thailand    510890.0
19         U.K.    241930.0
20       France    547557.0
21        Italy    294140.0
Type of area_df: <class 'pandas.core.frame.DataFrame'>
```

```
        Country    Population
0         China  1.415046e+09
1         India  1.354052e+09
2          U.S.  3.267667e+08
3     Indonesia  2.667950e+08
4        Brazil           NaN
5      Pakistan  2.008138e+08
6       Nigeria  1.958752e+08
7    Bangladesh  1.663681e+08
8        Russia  1.439647e+08
9        Mexico  1.307591e+08
10        Japan  1.271853e+08
11     Ethiopia  1.075349e+08
12  Philippines           NaN
13        Egypt  9.937574e+07
14     Viet-Nam  9.649115e+07
15     DR-Congo  8.400499e+07
16      Germany  8.229346e+07
17         Iran  8.201174e+07
18       Turkey  8.191687e+07
19     Thailand  6.918317e+07
20         U.K.  6.657350e+07
21       France  6.523327e+07
22        Italy  5.929097e+07
23  Afghanistan  1.351853e+08
Type of pop_df: <class 'pandas.core.frame.DataFrame'>
```

- Pandas gives functionality to merge DataFrames based on the indexes or columns. This is often called joining.
- There are different ways to join data sets:
  - Inner join – merges the data sets and excludes any column/index values that are not common to both.

- Left join – merges the data sets and includes all leftmost column/index values but excludes any columns in the rightmost DataFrame not common to both.
- Right join - merges the data sets and includes all rightmost column/index values but excludes any columns in the leftmost DataFrame not common to both

```
merged_inner = pd.merge(left=area_df,right=pop_df, left_on='Country', right_on='Country')
print(merged_inner)
```

	Country	Area(sqm)	Population
0	China	NaN	1.415046e+09
1	India	2973190.0	1.354052e+09
2	U.S.	9147420.0	3.267667e+08
3	Indonesia	1811570.0	2.667950e+08
4	Brazil	8358140.0	NaN
5	Pakistan	770880.0	2.008138e+08
6	Nigeria	910770.0	1.958752e+08
7	Bangladesh	130170.0	1.663681e+08
8	Russia	16376870.0	1.439647e+08
9	Mexico	1943950.0	1.307591e+08
10	Japan	364555.0	1.271853e+08
11	Ethiopia	1000000.0	1.075349e+08
12	Philippines	298170.0	NaN
13	Egypt	995450.0	9.937574e+07
14	Viet-Nam	NaN	9.649115e+07
15	Germany	348560.0	8.229346e+07
16	Iran	1628550.0	8.201174e+07
17	Turkey	769630.0	8.191687e+07
18	Thailand	510890.0	6.918317e+07
19	U.K.	241930.0	6.657350e+07
20	France	547557.0	6.523327e+07
21	Italy	294140.0	5.929097e+07

- As we saw with Series, we can choose to either drop rows (`dropna`()) that contain `NaN` data or fill the fields with another value(`fillna`())

```
cleaned_df = merged_inner.fillna(0)
print(cleaned_df)
```

	Country	Area(sqm)	Population
0	China	0.0	1.415046e+09
1	India	2973190.0	1.354052e+09
2	U.S.	9147420.0	3.267667e+08
3	Indonesia	1811570.0	2.667950e+08
4	Brazil	8358140.0	0.000000e+00
5	Pakistan	770880.0	2.008138e+08
6	Nigeria	910770.0	1.958752e+08
7	Bangladesh	130170.0	1.663681e+08
8	Russia	16376870.0	1.439647e+08
9	Mexico	1943950.0	1.307591e+08
10	Japan	364555.0	1.271853e+08
11	Ethiopia	1000000.0	1.075349e+08
12	Philippines	298170.0	0.000000e+00
13	Egypt	995450.0	9.937574e+07
14	Viet-Nam	0.0	9.649115e+07
15	Germany	348560.0	8.229346e+07
16	Iran	1628550.0	8.201174e+07
17	Turkey	769630.0	8.191687e+07
18	Thailand	510890.0	6.918317e+07
19	U.K.	241930.0	6.657350e+07
20	France	547557.0	6.523327e+07
21	Italy	294140.0	5.929097e+07

- Add a new column ‘Density’.

```
cleaned_df['Density'] = cleaned_df['Population'] / cleaned_df['Area(sqm)']
print(cleaned_df)
```

	Country	Area(sqm)	Population	Density
0	China	0.0	1.415046e+09	inf
1	India	2973190.0	1.354052e+09	455.420560
2	U.S.	9147420.0	3.267667e+08	35.722285
3	Indonesia	1811570.0	2.667950e+08	147.272797
4	Brazil	8358140.0	0.000000e+00	0.000000
5	Pakistan	770880.0	2.008138e+08	260.499453
6	Nigeria	910770.0	1.958752e+08	215.065535
7	Bangladesh	130170.0	1.663681e+08	1278.083652
8	Russia	16376870.0	1.439647e+08	8.790734
9	Mexico	1943950.0	1.307591e+08	67.264628
10	Japan	364555.0	1.271853e+08	348.878309
11	Ethiopia	1000000.0	1.075349e+08	107.534882
12	Philippines	298170.0	0.000000e+00	0.000000
13	Egypt	995450.0	9.937574e+07	99.829967
14	Viet-Nam	0.0	9.649115e+07	inf
15	Germany	348560.0	8.229346e+07	236.095527
16	Iran	1628550.0	8.201174e+07	50.358746
17	Turkey	769630.0	8.191687e+07	106.436692
18	Thailand	510890.0	6.918317e+07	135.416965
19	U.K.	241930.0	6.657350e+07	275.176721
20	France	547557.0	6.523327e+07	119.135124
21	Italy	294140.0	5.929097e+07	201.573975

- For some records, the Density was inf, which is an infinite value. This is because for some records the area was zero. We want to drop these from the DataFrame. In order to use dropna() we need to replace the infinite values (np.inf) with np.NaN.

```
cleaned_df = cleaned_df.replace(np.inf,np.NaN)
cleaned_df = cleaned_df.dropna()
print(cleaned_df)
```

	Country	Area(sqm)	Population	Density
1	India	2973190.0	1.354052e+09	455.420560
2	U.S.	9147420.0	3.267667e+08	35.722285
3	Indonesia	1811570.0	2.667950e+08	147.272797
4	Brazil	8358140.0	0.000000e+00	0.000000
5	Pakistan	770880.0	2.008138e+08	260.499453
6	Nigeria	910770.0	1.958752e+08	215.065535
7	Bangladesh	130170.0	1.663681e+08	1278.083652
8	Russia	16376870.0	1.439647e+08	8.790734
9	Mexico	1943950.0	1.307591e+08	67.264628
10	Japan	364555.0	1.271853e+08	348.878309
11	Ethiopia	1000000.0	1.075349e+08	107.534882
12	Philippines	298170.0	0.000000e+00	0.000000
13	Egypt	995450.0	9.937574e+07	99.829967
15	Germany	348560.0	8.229346e+07	236.095527
16	Iran	1628550.0	8.201174e+07	50.358746
17	Turkey	769630.0	8.191687e+07	106.436692
18	Thailand	510890.0	6.918317e+07	135.416965
19	U.K.	241930.0	6.657350e+07	275.176721
20	France	547557.0	6.523327e+07	119.135124
21	Italy	294140.0	5.929097e+07	201.573975

- We can sort the DataFrame values with respect to the Country Names.

```
sorted_df = cleaned_df.sort_values('Country')
print(sorted_df)
```

	Country	Area(sqm)	Population	Density
7	Bangladesh	130170.0	1.663681e+08	1278.083652
4	Brazil	8358140.0	0.000000e+00	0.000000
13	Egypt	995450.0	9.937574e+07	99.829967
11	Ethiopia	1000000.0	1.075349e+08	107.534882
20	France	547557.0	6.523327e+07	119.135124
15	Germany	348560.0	8.229346e+07	236.095527
1	India	2973190.0	1.354052e+09	455.420560
3	Indonesia	1811570.0	2.667950e+08	147.272797
16	Iran	1628550.0	8.201174e+07	50.358746
21	Italy	294140.0	5.929097e+07	201.573975
10	Japan	364555.0	1.271853e+08	348.878309
9	Mexico	1943950.0	1.307591e+08	67.264628
6	Nigeria	910770.0	1.958752e+08	215.065535
5	Pakistan	770880.0	2.008138e+08	260.499453
12	Philippines	298170.0	0.000000e+00	0.000000
8	Russia	16376870.0	1.439647e+08	8.790734
18	Thailand	510890.0	6.918317e+07	135.416965
17	Turkey	769630.0	8.191687e+07	106.436692
19	U.K.	241930.0	6.657350e+07	275.176721
2	U.S.	9147420.0	3.267667e+08	35.722285

Q1. Write a Python program that;

- Downloads the World Bank Data sets, containing Initial government funding per secondary student as a percentage of GDP per capita for a set of countries for 2014-2015 and 2013.
    - education_gdp_2014_15.xlsx
    - education_gdp_2013.xlsx

Output the resulting DataFrame /result after each step:

- Read the files education_GDP_2014_15.xls and education_GDP_2013.xlsx into two DataFrames.
- Join(merge) the data sets, to include 2013-2016 data, only for countries that appear in both sets.
- Replace all .. values with np.NaN.
- Drop all records with NaN values.
- Add a new column which stores the mean of 2013-2015 for each country.

Output:

2014_15 Data:

	Country Name	2014	2015
0	Turkey	11.2044	11.5303
1	Sweden	24.499	23.5969
2	South Africa	..	19.0818
3	Saudi Arabia	..	..
4	Poland	22.3213	22.1078
5	Pakistan	13.2514	15.2217
6	Italy	22.2214	22.8974
7	Finland	26.0708	25.8467
8	Canada	..	..
9	Brazil	20.715	21.6835

2013 Data:

	Country Name	2013
0	Turkey	11.7506
1	Sweden	24.8198
2	South Africa	..
3	Saudi Arabia	..
4	Poland	22.4415
5	Pakistan	10.3291
6	Italy	22.9081
7	Finland	..
8	Canada	..
9	Croatia	..
10	Cuba	…

Merged Data Set:

	Country Name	2013	2014	2015
0	Turkey	11.7506	11.2044	11.5303
1	Sweden	24.8198	24.499	23.5969
2	South Africa	..	..	19.0818
3	Saudi Arabia	..	..	..
4	Poland	22.4415	22.3213	22.1078
5	Pakistan	10.3291	13.2514	15.2217
6	Italy	22.9081	22.2214	22.8974
7	Finland	..	26.0708	25.8467
8	Canada	..	..	..

Data Set after Replace:

	Country Name	2013	2014	2015
0	Turkey	11.75063	11.20437	11.53028
1	Sweden	24.81979	24.49895	23.59689
2	South Africa	NaN	NaN	19.08178
3	Saudi Arabia	NaN	NaN	NaN
4	Poland	22.44155	22.32134	22.10783
5	Pakistan	10.32910	13.25144	15.22174
6	Italy	22.90814	22.22138	22.89744
7	Finland	NaN	26.07076	25.84670
8	Canada	NaN	NaN	NaN

Data Set after Dropping NaN:

	Country Name	2013	2014	2015
0	Turkey	11.75063	11.20437	11.53028
1	Sweden	24.81979	24.49895	23.59689
4	Poland	22.44155	22.32134	22.10783
5	Pakistan	10.32910	13.25144	15.22174
6	Italy	22.90814	22.22138	22.89744

Data Set with Mean:

	Country Name	2013	2014	2015	Mean 2013-2015
0	Turkey	11.75063	11.20437	11.53028	11.495093
1	Sweden	24.81979	24.49895	23.59689	24.305210
4	Poland	22.44155	22.32134	22.10783	22.290240
5	Pakistan	10.32910	13.25144	15.22174	12.934093
6	Italy	22.90814	22.22138	22.89744	22.675653

Sorted by Mean:

	Country Name	2013	2014	2015	Mean 2013-2015
0	Turkey	11.75063	11.20437	11.53028	11.495093
5	Pakistan	10.32910	13.25144	15.22174	12.934093
4	Poland	22.44155	22.32134	22.10783	22.290240
6	Italy	22.90814	22.22138	22.89744	22.675653
1	Sweden	24.81979	24.49895	23.59689	24.305210