Lab 4 - Recap Pandas 2

```
import pandas as pd
```

head(), tail()

```
df = pd.read_csv("/content/sample_data/california_housing_test.csv")

#df.head() # primele 5 inregistrari
#df.head(3) # primele 3 inregistrari
df.tail() # ultimele inregistrari
df.shape[0]
#df.head(int(df.shape[0] / 2))
print(df.shape)
nr_linii = df.shape[0]
jumatate_din_nr_linii = int(nr_linii / 2)
df.head(jumatate_din_nr_linii)
```

```
(3000, 9)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	
...	...	...	...	...	...	...	...	...	
1495	-121.92	36.63	40.0	1076.0	193.0	406.0	180.0	3.4943	
1496	-120.44	37.31	16.0	3369.0	532.0	1770.0	574.0	5.2662	
1497	-117.18	32.70	44.0	2655.0	514.0	1102.0	489.0	3.6759	
1498	-121.57	39.12	30.0	2601.0	534.0	1702.0	506.0	2.0800	
1499	-122.21	37.79	52.0	762.0	190.0	600.0	195.0	3.0893	

1500 rows × 9 columns

Query

```
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	med
count	3000.000000	3000.00000	3000.000000	3000.000000	3000.000000	3000.000000	3000.00000	3000.000000	
mean	-119.589200	35.63539	28.845333	2599.578667	529.950667	1402.798667	489.91200	3.807272	
std	1.994936	2.12967	12.555396	2155.593332	415.654368	1030.543012	365.42271	1.854512	
min	-124.180000	32.56000	1.000000	6.000000	2.000000	5.000000	2.00000	0.499900	
25%	-121.810000	33.93000	18.000000	1401.000000	291.000000	780.000000	273.00000	2.544000	
50%	-118.485000	34.27000	29.000000	2106.000000	437.000000	1155.000000	409.50000	3.487150	
75%	-118.020000	37.69000	37.000000	3129.000000	636.000000	1742.750000	597.25000	4.656475	
max	-114.490000	41.92000	52.000000	30450.000000	5419.000000	11935.000000	4930.00000	15.000100	

După o singură condiție

```
# doar inregistrari care au o valoare mai > decat media pe coloana de total_rooms
# df[df["total_rooms"] > df["total_rooms"].mean()]
#df["total_rooms"] > df["total_rooms"].mean()
df_filtered = df[df["total_rooms"] > df["total_rooms"].mean()]
#df_filtered.info()
#df_filtered.shape
#df_filtered.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	me
count	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	
mean	-119.592873	35.637136	23.139091	4425.112727	866.115455	2186.998182	790.112727	4.326758	
std	2.037674	2.118103	11.374497	2600.047775	504.337034	1209.063335	433.718715	1.839957	
min	-124.170000	32.570000	2.000000	2601.000000	302.000000	83.000000	45.000000	1.101900	
25%	-121.840000	33.920000	15.000000	2994.000000	581.000000	1478.750000	544.750000	3.072225	
50%	-118.630000	34.360000	22.000000	3652.000000	734.000000	1868.000000	680.500000	4.033700	
75%	-117.900000	37.682500	30.250000	4746.250000	1001.250000	2530.250000	908.250000	5.232300	
max	-114.490000	41.800000	52.000000	30450.000000	5419.000000	11935.000000	4930.000000	15.000100	

După mai multe condiții & = și, | = sau

```
import pandas as pd
df = pd.read_csv("/content/sample_data/california_housing_test.csv")
df_filtered = df[(df["total_rooms"] > df["total_rooms"].mean()) &
 (df["total_bedrooms"] > df["total_bedrooms"].mean())]
#df_filtered.info()
#df_filtered.shape
#print(df_filtered)
```

```
      longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0       -122.05     37.37                27.0       3885.0           661.0
8       -122.84     38.40                15.0       3080.0           617.0
13      -117.03     32.97                16.0       3936.0           694.0
17      -121.20     38.69                26.0       3077.0           607.0
19      -122.59     38.01                35.0       8814.0          1307.0
...         ...       ...                 ...          ...             ...
2981    -120.66     35.49                17.0       4422.0           945.0
2984    -117.59     33.88                13.0       3239.0           849.0
2987    -121.97     37.29                25.0       4096.0           743.0
2991    -117.17     34.28                13.0       4867.0           718.0
2996    -118.14     34.06                27.0       5257.0          1082.0

      population  households  median_income  median_house_value
0         1537.0       606.0         6.6085            344700.0
8         1446.0       599.0         3.6696            194400.0
13        1935.0       659.0         4.5625            231200.0
17        1603.0       595.0         2.7174            137500.0
19        3450.0      1258.0         6.1724            414300.0
...          ...         ...            ...                 ...
2981      2307.0       885.0         2.8285            171300.0
2984      2751.0       813.0         2.6111            107000.0
2987      2027.0       741.0         5.3294            300300.0
2991       780.0       250.0         7.1997            253800.0
2996      3496.0      1036.0         3.3906            237200.0

[920 rows x 9 columns]
```

Varinat alternativa de filtrare

si = "and", sau = "or"

```
#df.query("total_rooms > 1000")
#df.query("total_rooms > total_rooms.mean()")
df.query("total_rooms > total_rooms.mean() and total_bedrooms > total_bedrooms.mean()")
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	
8	-122.84	38.40	15.0	3080.0	617.0	1446.0	599.0	3.6696	
13	-117.03	32.97	16.0	3936.0	694.0	1935.0	659.0	4.5625	
17	-121.20	38.69	26.0	3077.0	607.0	1603.0	595.0	2.7174	
19	-122.59	38.01	35.0	8814.0	1307.0	3450.0	1258.0	6.1724	
...	...	...	...	...	...	...	...	...	
2981	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285	
2984	-117.59	33.88	13.0	3239.0	849.0	2751.0	813.0	2.6111	
2987	-121.97	37.29	25.0	4096.0	743.0	2027.0	741.0	5.3294	
2991	-117.17	34.28	13.0	4867.0	718.0	780.0	250.0	7.1997	
2996	-118.14	34.06	27.0	5257.0	1082.0	3496.0	1036.0	3.3906	

920 rows × 9 columns

```
df_filtered = df.query("total_rooms > total_rooms.mean() and total_bedrooms > total_bedrooms.mean()")
display(df_filtered)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	
8	-122.84	38.40	15.0	3080.0	617.0	1446.0	599.0	3.6696	
13	-117.03	32.97	16.0	3936.0	694.0	1935.0	659.0	4.5625	
17	-121.20	38.69	26.0	3077.0	607.0	1603.0	595.0	2.7174	
19	-122.59	38.01	35.0	8814.0	1307.0	3450.0	1258.0	6.1724	
...	...	...	...	...	...	...	...	...	
2981	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285	
2984	-117.59	33.88	13.0	3239.0	849.0	2751.0	813.0	2.6111	
2987	-121.97	37.29	25.0	4096.0	743.0	2027.0	741.0	5.3294	
2991	-117.17	34.28	13.0	4867.0	718.0	780.0	250.0	7.1997	
2996	-118.14	34.06	27.0	5257.0	1082.0	3496.0	1036.0	3.3906	

920 rows × 9 columns

Operations on strings

```
data = {'Country': ['USA', 'China', 'Japan', 'Germany', 'India'],
        'GDP (Billions USD)': [23310, 17700, 4910, 4260, 3170]}
gdp_df = pd.DataFrame(data)
display(gdp_df)
gdp_df["Country"].str.lower()
#gdp_df["CountryLower"] = gdp_df["Country"].str.lower() # adds a new column
gdp_df["Country"] = gdp_df["Country"].str.lower() # updates the values on this column
gdp_df
```

	Country	GDP (Billions USD)
0	USA	23310
1	China	17700
2	Japan	4910
3	Germany	4260
4	India	3170

Performance Aspects

Often multiple options to achieve the same goal with pandas. This section compares such options for adding up two columns element-wise. First, the data set, generated with NumPy:

```
import numpy as np
data = np.random.standard_normal((1000000, 2))
df = pd.DataFrame(data, columns=['A', 'B'])
# The DataFrame object with the random numbers.
df.head()
```

	A	B
0	-0.269533	-0.604914
1	-1.932016	-0.324471
2	0.777510	0.548133
3	-0.544844	-0.961828
4	0.559996	0.191382

```
# Approach 1 - the fastest way
%time res = df['A'] + df['B']
```

```
CPU times: user 2.7 ms, sys: 3.92 ms, total: 6.62 ms
Wall time: 7.32 ms
```

```
# Approach 2
%time res = df.sum(axis=1)
```

```
CPU times: user 257 ms, sys: 28.5 ms, total: 286 ms
Wall time: 298 ms
```

```
# Approach 3 (NumPy)
%time res = np.sum(df, axis=1)
```

```
CPU times: user 150 ms, sys: 20.9 ms, total: 171 ms
Wall time: 172 ms
```