EP - Data Manipulation & Advanced Plotting (pandas, seaborn & 3D Plotting)

In this lab, we will study data manipulation and visualization using **pandas**, and explore the high level API of **seaborn** for generating visually appealing plots. We will also take a look at 3D plotting using **mplot3d**.

```
# Some IPython magic
# Put these at the top of every notebook, to get automatic reloading and inline plotting
%reload_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Check out these cheetsheets for fast reference to the common libraries:

Cheat sheets:

- python
- numpy
- matplotlib
- sklearn
- pandas

Other:

- Probabilities & Stats Refresher
- Algebra

Pandas Crash Course

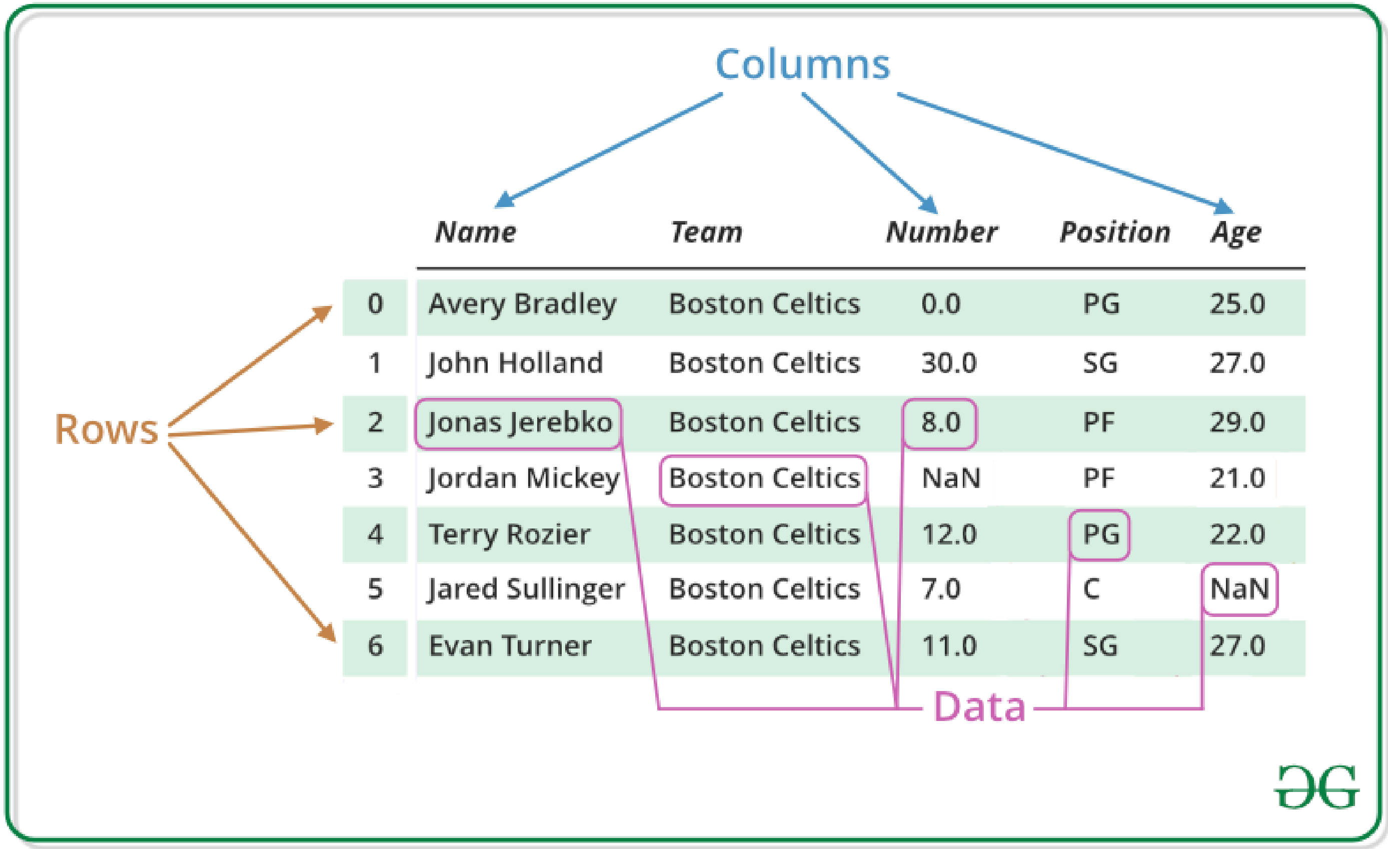
Pandas is a high-level data manipulation tool. It is built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables.

Check this official guide for a started in pandas:

10 minutes to pandas

```
import pandas as pd
```

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.



	Name	*Team*	*Number*	*Position*	*Age*
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Let's load a publicly available *.csv* dataset into a pandas ***DataFrame***. We will use the popular *iris* dataset.

```
file_name = "https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv"

df = pd.read_csv(file_name)
df.head(n = 10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
8	4.4	2.9	1.4	0.2	setosa

A dataframe's **.describe()** method offers descriptive statistics which include those that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.

```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Let's see some ways in which we can access the DataFrames' data. Each column of a pandas `DataFrame` is a pandas `Series`.

```
df['petal_width']
```

```
    0      0.2
    1      0.2
    2      0.2
    3      0.2
    4      0.2
          ...
    145    2.3
    146    1.9
    147    2.0
    148    2.3
```

```
149    1.8
Name: petal_width, Length: 150, dtype: float64
```

We can do any vectorized operation on a `Series`. Moreover, a pandas `Series` allows us to do conditional selection of rows in a `DataFrame`.

```
setosas = df[df['species'] == 'setosa']

setosas.head() # only setosa species selected
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

We can add a new column to a pandas `DataFrame`, simply by specifying its name and its contents.

NB: the data added to the new column must be the same length as the rest of the `DataFrame`.

```
df['sepal_area'] = df['sepal_length'] * df['sepal_width'] # adding new columns
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	sepal_area
0	5.1	3.5	1.4	0.2	setosa	17.85
1	4.9	3.0	1.4	0.2	setosa	14.70
2	4.7	3.2	1.3	0.2	setosa	15.04
3	4.6	3.1	1.5	0.2	setosa	14.26
4	5.0	3.6	1.4	0.2	setosa	18.00

We can work with `Series` as we work with numpy arrays. We perform Min-Max normalization on the `petal_length` column.

```
# Min-Max Normalization
df['petal_length'] = (df['petal_length'] - df['petal_length'].min()) / (df['petal_length'].ma
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	sepal_area
0	5.1	3.5	0.067797	0.2	setosa	17.85
1	4.9	3.0	0.067797	0.2	setosa	14.70
2	4.7	3.2	0.050847	0.2	setosa	15.04
3	4.6	3.1	0.084746	0.2	setosa	14.26

We can also use the `.apply()` method on either a `Series` or a `DataFrame` to modify its contents, or create a new column.

```
def capitalize(col):
  return col.capitalize()

df['species'] = df['species'].apply(capitalize)
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	sepal_area
0	5.1	3.5	0.067797	0.2	Setosa	17.85
1	4.9	3.0	0.067797	0.2	Setosa	14.70
2	4.7	3.2	0.050847	0.2	Setosa	15.04
3	4.6	3.1	0.084746	0.2	Setosa	14.26
4	5.0	3.6	0.067797	0.2	Setosa	18.00

A `DataFrame` also has a `groupby` method, that allows us to work on groupings of rows.

```
df.groupby('species').mean()
```

	sepal_length	sepal_width	petal_length	petal_width	sepal_area
species					
Setosa	5.006	3.418	0.078644	0.244	17.2088
Versicolor	5.936	2.770	0.552542	1.326	16.5262
Virginica	6.588	2.974	0.771525	2.026	19.6846

We can also iterate through each group. A group is another `DataFrame`.

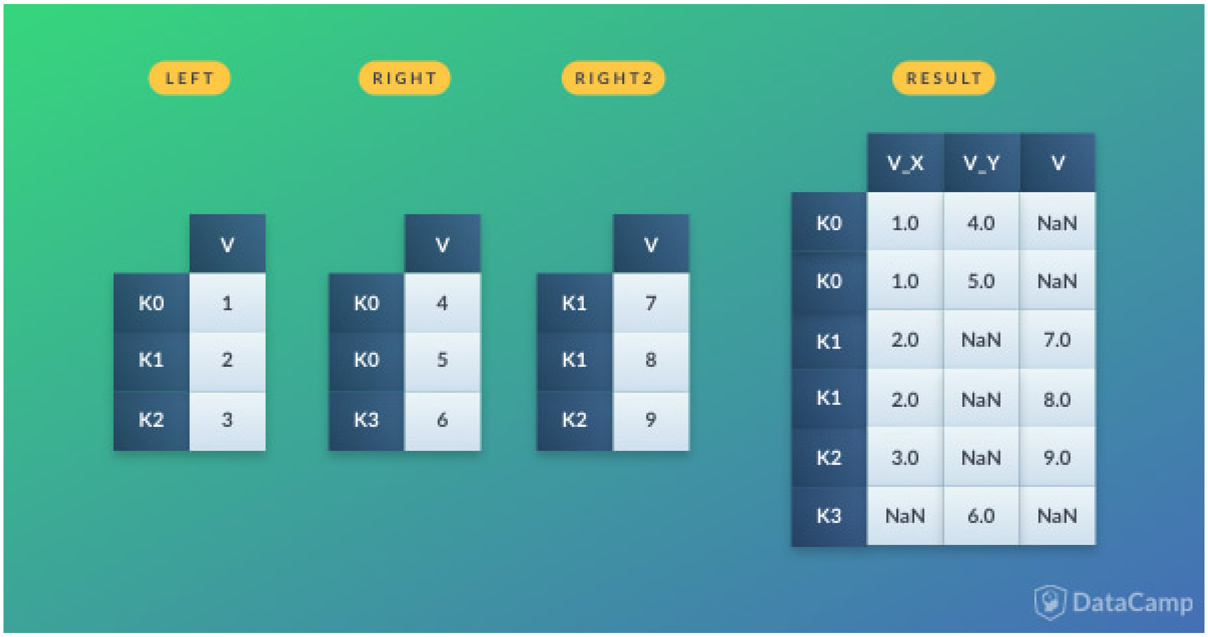
```
for name, group in df.groupby('species'):
  print("Group:", name)
```

```
    print(group.head())
    print("-----")
```

```
Group: Setosa
   sepal_length  sepal_width  petal_length  petal_width species  sepal_area
0           5.1          3.5      0.067797          0.2  Setosa       17.85
1           4.9          3.0      0.067797          0.2  Setosa       14.70
2           4.7          3.2      0.050847          0.2  Setosa       15.04
3           4.6          3.1      0.084746          0.2  Setosa       14.26
4           5.0          3.6      0.067797          0.2  Setosa       18.00
-----
Group: Versicolor
    sepal_length  sepal_width  ...     species  sepal_area
50           7.0          3.2  ...  Versicolor       22.40
51           6.4          3.2  ...  Versicolor       20.48
52           6.9          3.1  ...  Versicolor       21.39
53           5.5          2.3  ...  Versicolor       12.65
54           6.5          2.8  ...  Versicolor       18.20

[5 rows x 6 columns]
-----
Group: Virginica
     sepal_length  sepal_width  ...    species  sepal_area
100           6.3          3.3  ...  Virginica       20.79
101           5.8          2.7  ...  Virginica       15.66
102           7.1          3.0  ...  Virginica       21.30
103           6.3          2.9  ...  Virginica       18.27
104           6.5          3.0  ...  Virginica       19.50

[5 rows x 6 columns]
-----
```

Joins

Pandas allows for joining two or more `DataFrames` together using a common key. We can also do vertical or horizontal concatenation .



LEFT

	V
K0	1
K1	2
K2	3

RIGHT

	V
K0	4
K0	5
K3	6

RIGHT2

	V
K1	7
K1	8
K2	9

RESULT

	V_X	V_Y	V
K0	1.0	4.0	NaN
K0	1.0	5.0	NaN
K1	2.0	NaN	7.0
K1	2.0	NaN	8.0
K2	3.0	NaN	9.0
K3	NaN	6.0	NaN

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']})
df2 = pd.DataFrame({'A': ['A0', 'A1', 'A4', 'A5'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']})
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']})

pd.concat([df1, df2, df3]).reset_index(drop = True)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A0	B4	C4	D4
5	A1	B5	C5	D5
6	A4	B6	C6	D6
7	A5	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
pd.merge(df1, df2, on = 'A', how = 'left')
```

	A	B_x	C_x	D_x	B_y	C_y	D_y
0	A0	B0	C0	D0	B4	C4	D4
1	A1	B1	C1	D1	B5	C5	D5
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN

Saving `DataFrames`

Pandas offers a multitude of methods for saving `DataFrames`.

```
df.to_csv('out.csv', index = False) # saves it locally, check out the files in the right tab
```

```
df.to_json()
```

```
'{"sepal_length":{"0":5.1,"1":4.9,"2":4.7,"3":4.6,"4":5.0,"5":5.4,"6":4.6,"7":5.0,"8":
4.4,"9":4.9,"10":5.4,"11":4.8,"12":4.8,"13":4.3,"14":5.8,"15":5.7,"16":5.4,"17":5.1,"1
8":5.7,"19":5.1,"20":5.4,"21":5.1,"22":4.6,"23":5.1,"24":4.8,"25":5.0,"26":5.0,"27":5.
2,"28":5.2,"29":4.7,"30":4.8,"31":5.4,"32":5.2,"33":5.5,"34":4.9,"35":5.0,"36":5.5,"3
7":4.9,"38":4.4,"39":5.1,"40":5.0,"41":4.5,"42":4.4,"43":5.0,"44":5.1,"45":4.8,"46":5.
1,"47":4.6,"48":5.3,"49":5.0,"50":7.0,"51":6.4,"52":6.9,"53":5.5,"54":6.5,"55":5.7,"5
6":6.3,"57":4.9,"58":6.6,"59":5.2,"60":5.0,"61":5.9,"62":6.0,"63":6.1,"64":5.6,"65":6.
```

```
print(df.head().to_markdown())
```

```
|    |   sepal_length |   sepal_width |   petal_length |   petal_width | species    |   s
|---:|---------------:|--------------:|---------------:|--------------:|:-----------|----
|  0 |            5.1 |           3.5 |      0.0677966 |           0.2 | Setosa     |
|  1 |            4.9 |           3   |      0.0677966 |           0.2 | Setosa     |
|  2 |            4.7 |           3.2 |      0.0508475 |           0.2 | Setosa     |
|  3 |            4.6 |           3.1 |      0.0847458 |           0.2 | Setosa     |
|  4 |            5   |           3.6 |      0.0677966 |           0.2 | Setosa     |
```

```
print(df.head().to_latex())
```

```
\begin{tabular}{lrrrrlr}
\toprule
{} &  sepal\_length &  sepal\_width &  petal\_length &  petal\_width & species &  sepal\
\midrule
0 &           5.1 &          3.5 &      0.067797 &          0.2 &  Setosa &       17.85
1 &           4.9 &          3.0 &      0.067797 &          0.2 &  Setosa &       14.70
2 &           4.7 &          3.2 &      0.050847 &          0.2 &  Setosa &       15.04
3 &           4.6 &          3.1 &      0.084746 &          0.2 &  Setosa &       14.26
4 &           5.0 &          3.6 &      0.067797 &          0.2 &  Setosa &       18.00
\bottomrule
\end{tabular}
```

```
print(df.head(n = 3).to_html())
```

```
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>sepal_length</th>
```

```
          <th>sepal_width</th>
          <th>petal_length</th>
          <th>petal_width</th>
          <th>species</th>
          <th>sepal_area</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <th>0</th>
          <td>5.1</td>
          <td>3.5</td>
          <td>0.067797</td>
          <td>0.2</td>
          <td>Setosa</td>
          <td>17.85</td>
        </tr>
        <tr>
          <th>1</th>
          <td>4.9</td>
          <td>3.0</td>
          <td>0.067797</td>
          <td>0.2</td>
          <td>Setosa</td>
          <td>14.70</td>
        </tr>
        <tr>
          <th>2</th>
          <td>4.7</td>
          <td>3.2</td>
          <td>0.050847</td>
          <td>0.2</td>
          <td>Setosa</td>
          <td>15.04</td>
        </tr>
      </tbody>
    </table>
```

```
df.to_sql(name = '<table_name>', con = '<connection>') # insert into a sql database, works wi
# check out https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_sq
```

```
---------------------------------------------------------------------------
ArgumentError                             Traceback (most recent call last)
<ipython-input-19-6d777a02bc3c> in <module>()
----> 1 df.to_sql(name = '<table_name>', con = '<connection>') # insert into a sql
database, works with a valid connection
```

Pandas Plotting

Pandas offers a convenient API for plotting data directly from a DataFrame. Of course, the plotting API is build upon `matplotlib` as a low level backend. We can use that to manipulate plots as in the previous lab. Check out the official documentation for visualization:

[Pandas Plotting Docs]

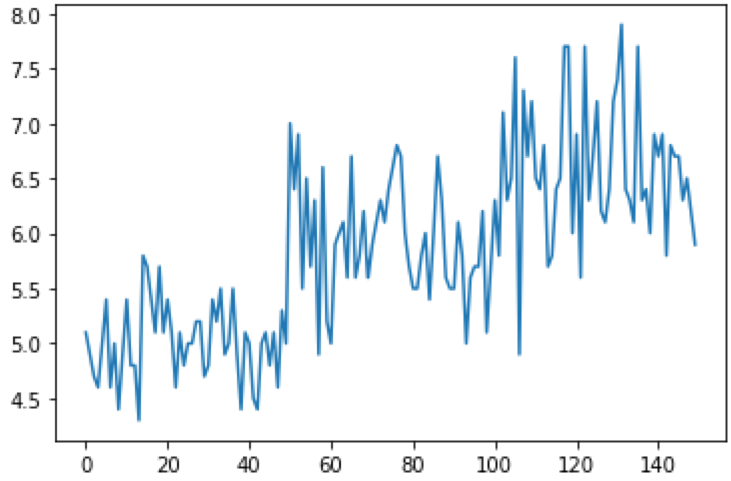
For a quick reference, check the official cookbook.

[Pandas Plotting Cookbook]

We will use the high level plotting API to visualize the Iris Dataset.

```
df['sepal_length'].plot()
```

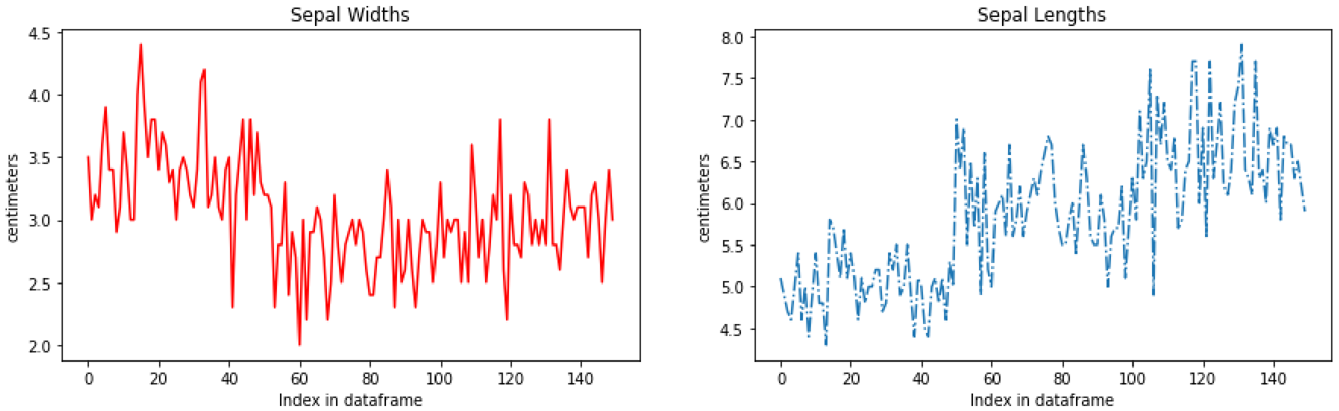
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabb6717f50>
```



```
df[['sepal_width', 'sepal_length']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabb65ff610>
```

```
fig, ax = plt.subplots(1, 2)

df['sepal_width'].plot(ax = ax[0], color = 'r')
df['sepal_length'].plot(ax = ax[1], linestyle = '-.')

ax[0].set_title('Sepal Widths')
ax[1].set_title('Sepal Lengths')

ax[0].set_xlabel('Index in dataframe')
ax[1].set_xlabel('Index in dataframe')

ax[0].set_ylabel('centimeters')
ax[1].set_ylabel('centimeters')


fig.set_size_inches(15, 4)
```

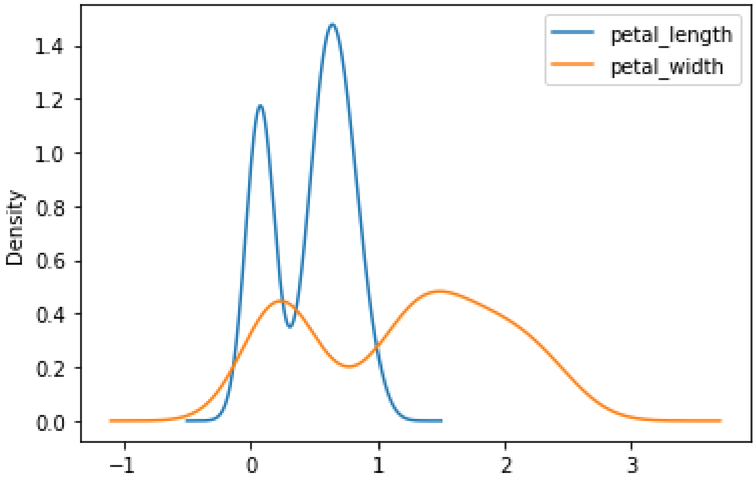


```
df[['petal_width', 'petal_length']].plot.hist(alpha = 0.5, bins = 15)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabb6056d50>
```

```
df[['petal_length', 'petal_width']].plot.kde()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabb6148250>
```



Cool Plots using Seaborn & Pandas

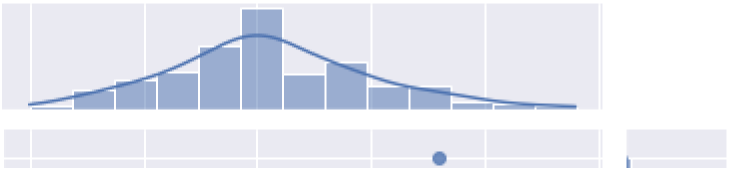
Check out seaborn for more awesome plots.

```
import seaborn as sns
```

```
sns.set_theme()
```

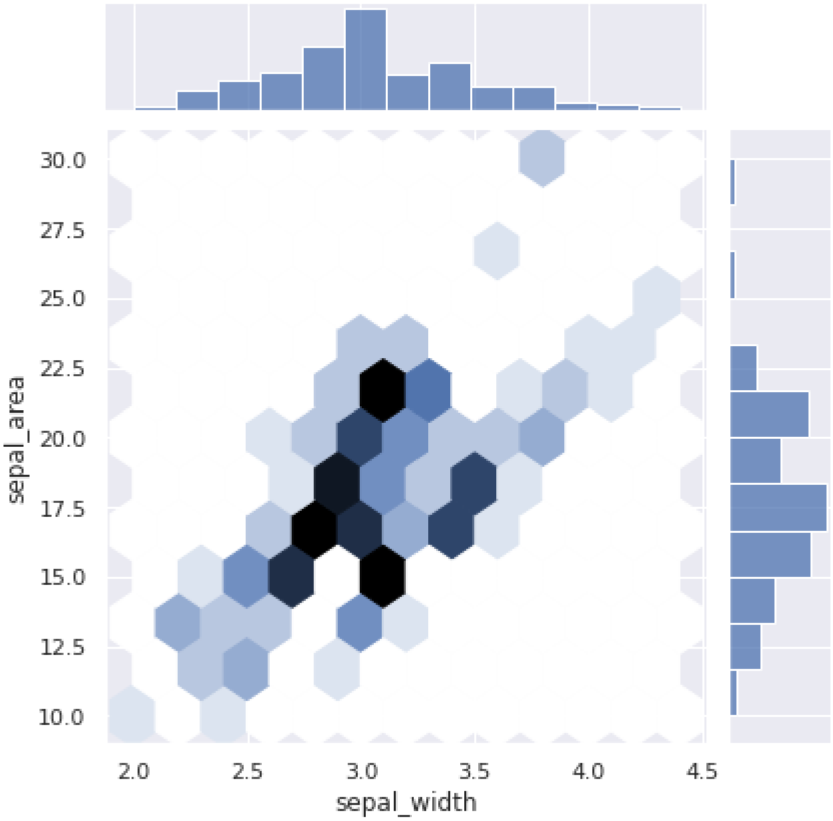
```
sns.jointplot(x = 'sepal_width', y = 'sepal_area', data = df, kind = 'reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7faba3ca7690>
```

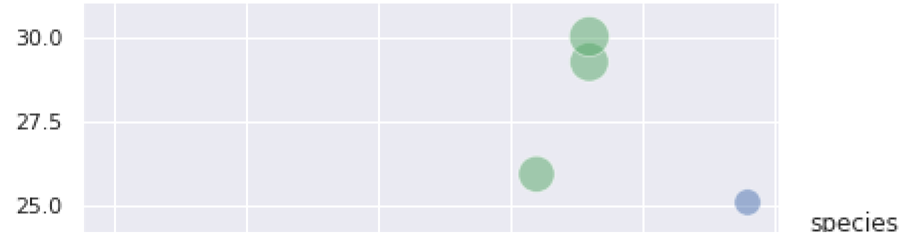


```
sns.jointplot(x = 'sepal_width', y = 'sepal_area', data = df, kind = 'hex')
```

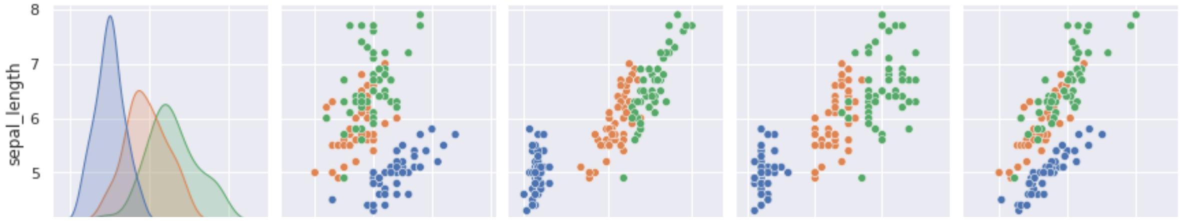
```
<seaborn.axisgrid.JointGrid at 0x7fab9b327990>
```



```
sns.relplot(x="sepal_width", y="sepal_area", hue="species", size="sepal_length", sizes=(40, 4
<seaborn.axisgrid.FacetGrid at 0x7fab9b360850>
```

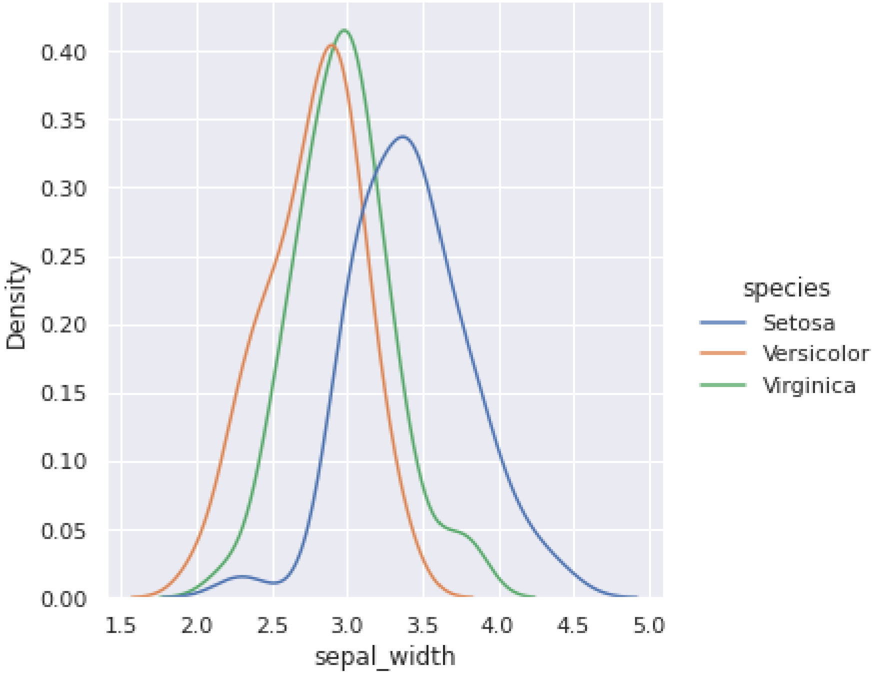


```
grid = sns.pairplot(df, hue = 'species')
```



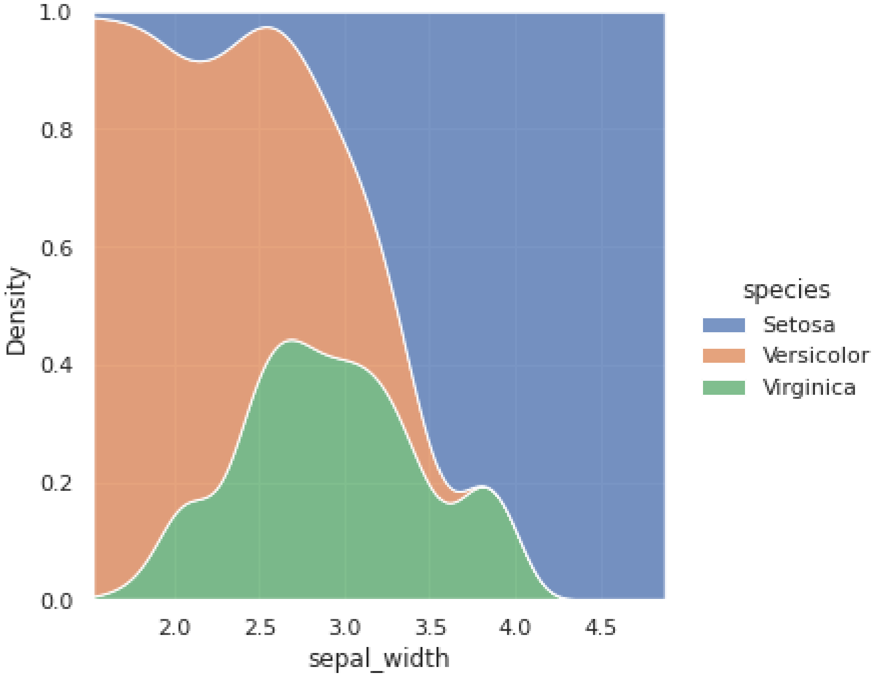
```
sns.displot(data = df, x = 'sepal_width', kind = 'kde', hue = 'species')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fab9ac16a90>
```



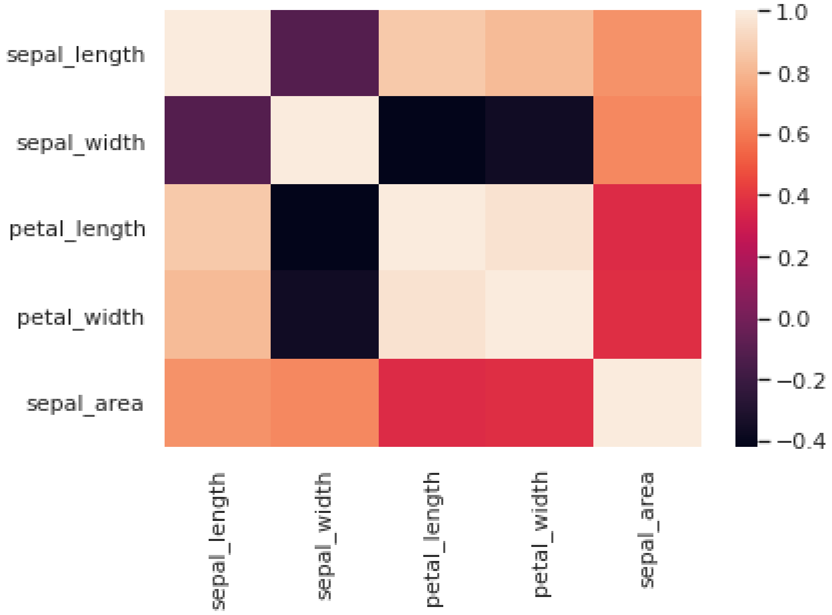
```
sns.displot(data = df, x = 'sepal_width', kind = 'kde', hue = 'species', multiple = 'fill')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fab9b0c9990>
```



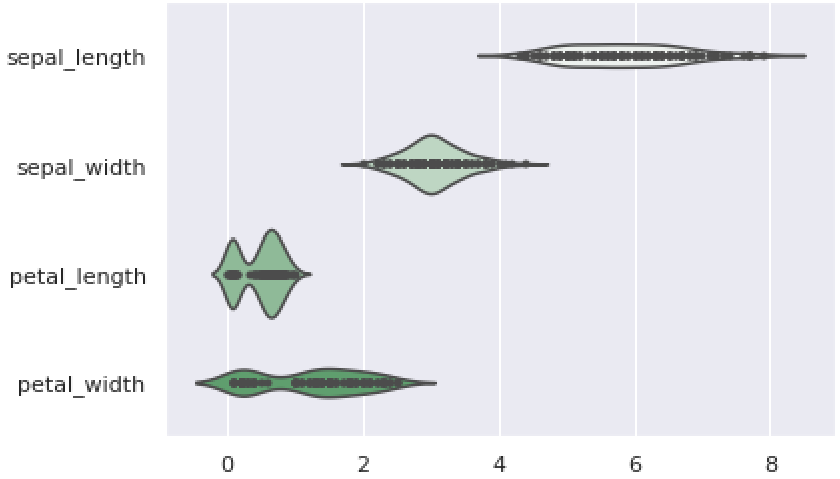
```
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab9a57b450>
```



```
sns.violinplot(data=df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], palet
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab9a462a90>
```



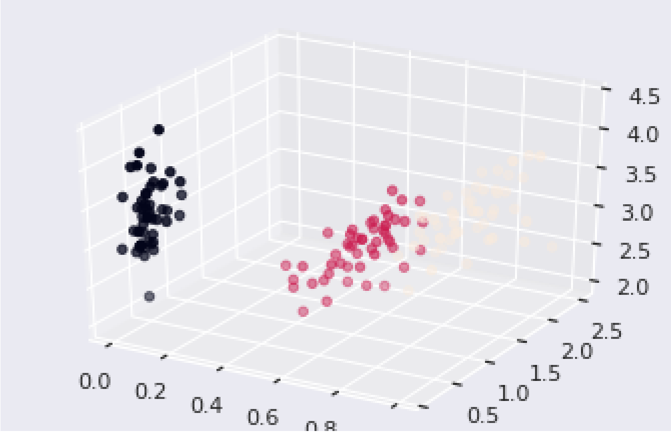
3D Plotting

```
from mpl_toolkits.mplot3d import Axes3D
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
ax.scatter(df['petal_length'], df['petal_width'], df['sepal_width'], zdir='z', s=20, c=df['sp
```

```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fab9a3a3190>
```



Exercises

After you finish your lab exercises, you should export this notebook as **pdf** and upload it to Moodle. (i.e. **File -> Print**, Destintation: Save as PDF).

In this lab, we will look at COVID data for Romania. The data was retrieved from https://graphs.ro/.

```
import requests
import json

covid_data = json.loads(requests.get('https://www.graphs.ro/json.php').content)

covid_df = pd.DataFrame(covid_data['covid_romania'])
covid_df.head()

covid_county_data_dfs = []

for i, row in covid_df.iterrows():
  try: # some days have no county information
    county_df = pd.DataFrame(row['county_data'])
  except:
    continue

  county_df['reporting_date'] = row['reporting_date']
  covid_county_data_dfs.append(county_df)


county_df = pd.concat(covid_county_data_dfs)
covid_df = covid_df.drop(['sourceUrl', 'county_data'], axis = 1)


covid_df.head()
```

	reporting_date	total_cases	new_cases_today	total_tests	new_tests_today	total_de
0	2021-11-12	1735277	4844	10509317.0	15540.0	525
1	2021-11-11	1730433	5416	10493777.0	17114.0	522
2	2021-11-10	1725017	6291	10476663.0	18259.0	518
3	2021-11-09	1718726	7589	10458404.0	19449.0	514
4	2021-11-08	1711137	4255	10438955.0	7702.0	509

```
county_df.head()
```

	county_id	county_name	county_population	total_cases	reporting_date
0	AB	Alba	323778	32666	2021-11-12
1	AR	Arad	415732	39287	2021-11-12
2	AG	Arges	574920	44375	2021-11-12
3	BC	Bacau	580912	39953	2021-11-12
4	BH	Bihor	559992	47701	2021-11-12

1. Basic Visualizations

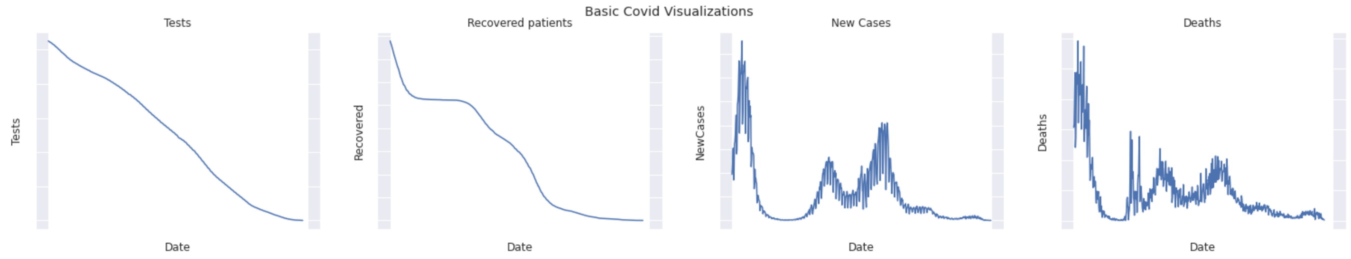
Make 4 subplots. Using pandas as seaborn, plot the number of new cases in a day, the number of recovered patients in day, number of tests in a day, and the number of deaths in day. We are trying to explore the evolution of COVID from the start of the pandemic until today.

NB: Make sure to add proper labels, title, axes and legend where necessary.

```
# TODO your code here
import matplotlib.dates as mdates
import  datetime

fig, axes = plt.subplots(1, 4, sharex=True, figsize=(26,4))
fig.suptitle('Basic Covid Visualizations')

axes[0].set(yticklabels=[])
axes[0].set(xticklabels=[])
df_0 = pd.DataFrame({"Date": covid_df['reporting_date'],
                   "Tests": covid_df['total_tests']})
sns.lineplot(ax=axes[0], x='Date', y='Tests',data = df_0)
axes[0].set_title('Tests')

axes[1].set(yticklabels=[])
axes[1].set(xticklabels=[])
```

```
df_1 = pd.DataFrame({"Date": covid_df['reporting_date'],
                   "Recovered": covid_df['total_recovered']})
sns.lineplot(ax=axes[1], x='Date', y='Recovered',data = df_1)
axes[1].set_title('Recovered patients')

axes[2].set(yticklabels=[])
axes[2].set(xticklabels=[])
df_2 = pd.DataFrame({"Date": covid_df['reporting_date'],
                   "NewCases": covid_df['new_cases_today']})
sns.lineplot(ax=axes[2], x='Date', y='NewCases',data = df_2)
axes[2].set_title('New Cases')


axes[3].set(yticklabels=[])
axes[3].set(xticklabels=[])
df_3 = pd.DataFrame({"Date": covid_df['reporting_date'],
                   "Deaths": covid_df['new_deaths_today']})
sns.lineplot(ax=axes[3], x='Date', y='Deaths',data = df_3)
axes[3].set_title('Deaths')
```

```
Text(0.5, 1.0, 'Deaths')
```



2. Positive testing percentage

Using `pandas`, create a new column that computes the percentage of positive tests in a given day. This new column should be the number of infected people in a day over the number of tests per day.

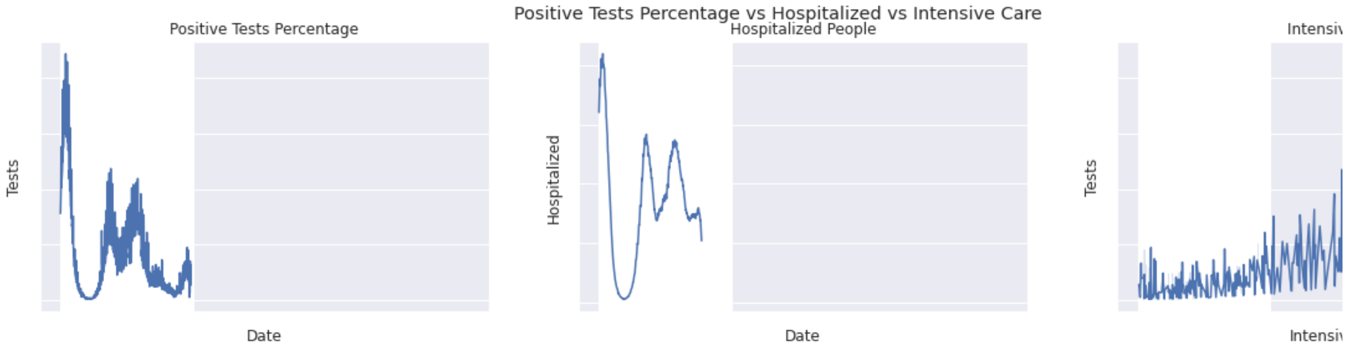
Plot the evolution of positive tests across time. Compare this to the number of hospitalized patients.

In a different plot, visualize the correlation between positive tests and the number of intensive care patients.

NB: Make sure to add proper labels, title, axes and legend where necessary.

```
# TODO your code here
covid_df['positive_tests_percentage'] = (covid_df['new_cases_today'] * 100 ) / covid_df['new_
covid_df[['new_cases_today', 'new_tests_today', 'positive_tests_percentage']].head()

fig, axes = plt.subplots(1, 3, sharex=True, figsize=(22,4))
fig.suptitle('Positive Tests Percentage vs Hospitalized vs Intensive Care')

axes[0].set(yticklabels=[])
axes[0].set(xticklabels=[])
df_0 = pd.DataFrame({"Date": covid_df['reporting_date'],
                   "Tests": covid_df['positive_tests_percentage']})
sns.lineplot(ax=axes[0], x='Date', y='Tests',data = df_0)
axes[0].set_title('Positive Tests Percentage')

axes[1].set(yticklabels=[])
axes[1].set(xticklabels=[])
df_1 = pd.DataFrame({"Date": covid_df['reporting_date'],
                   "Hospitalized": covid_df['infected_hospitalized']})
sns.lineplot(ax=axes[1], x='Date', y='Hospitalized',data = df_1)
axes[1].set_title('Hospitalized People')

axes[2].set(yticklabels=[])
axes[2].set(xticklabels=[])
df_2 = pd.DataFrame({"Tests": covid_df['positive_tests_percentage'],
                   "IntensiveCare": covid_df['intensive_care_right_now']})
sns.lineplot(ax=axes[2], x='IntensiveCare', y='Tests',data = df_2)
axes[2].set_title('Intensive Care')
```

```
Text(0.5, 1.0, 'Intensive Care')
```



3. County Information

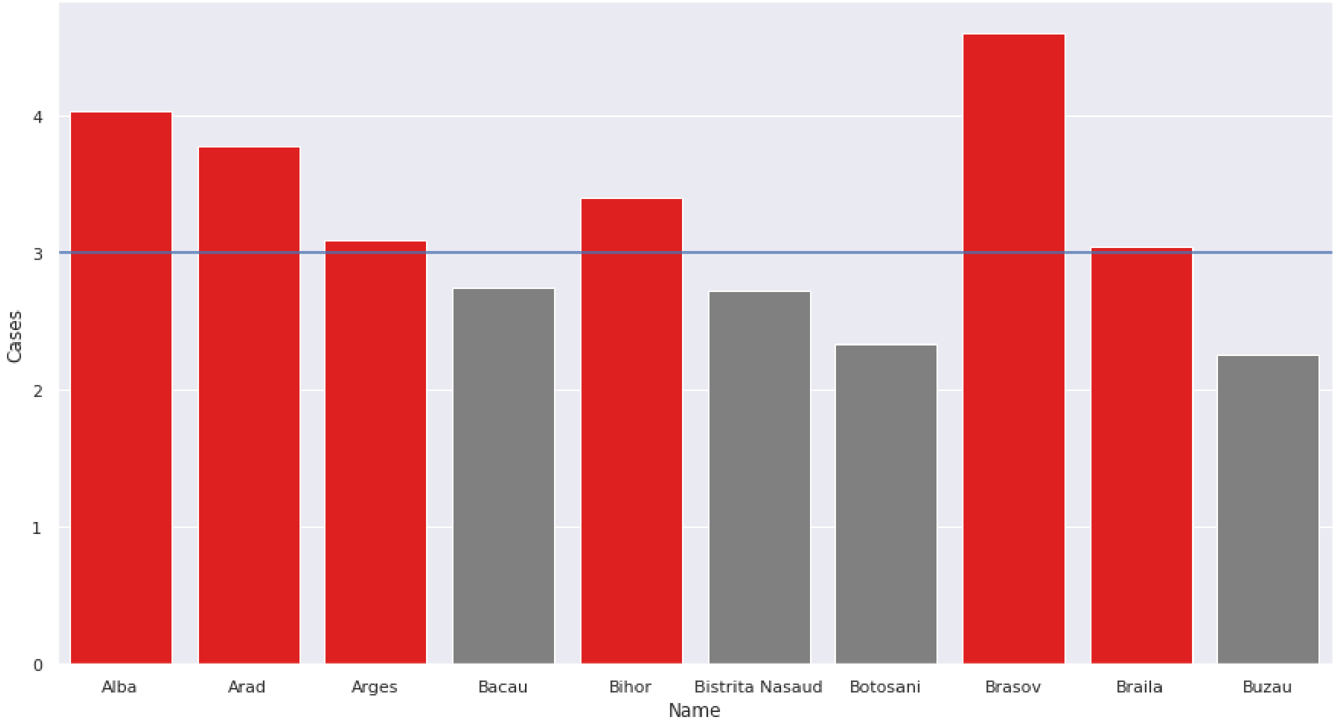
Select at least 10 counties of your choosing, and plot the evolution of cases across time.

Plot the number of positive cases on 1000 persons for each of your selected counties. Plot a horizontal line at the 3 / 1000 mark. When the number of positive cases per 1000 persons exceeds 3 / 1000, color your points in a different color from that point onwards.

NB: Make sure to add proper labels, title, axes and legend where necessary.

```
# Compute the cases / 1000 people then create a new column
county_df['cases_percentage'] = county_df['total_cases'] / (county_df['county_population'] /
county_df[['county_name', 'total_cases', 'county_population', 'cases_percentage']].head(10)

limit = 3.0

colors = ['grey' if (x < limit) else 'red' for x in county_df['cases_percentage']]
sns.set(rc = {'figure.figsize':(15,8)})

df = pd.DataFrame({"Cases": county_df['cases_percentage'].head(10),
                   "Name": county_df['county_name'].head(10)})

# Counties that go over the limit have red bars
sns.barplot(x='Name', y='Cases', data=df, palette=colors).axhline(limit)
```

```
<matplotlib.lines.Line2D at 0x7fab77555f10>
```



BONUS

Further expore the dataset, and come up with interesting visualizations of the COVID evolution in Romania.

```
# TODO your code here (maybe)
```

✓ 0s completed at 11:35 AM