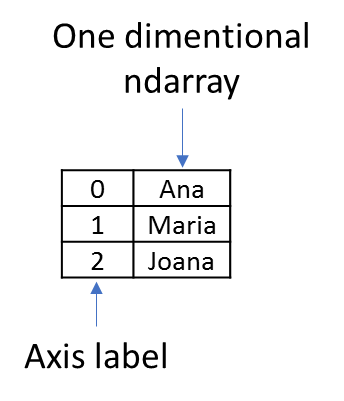
**Pandas module**

A black bear lying on the ground

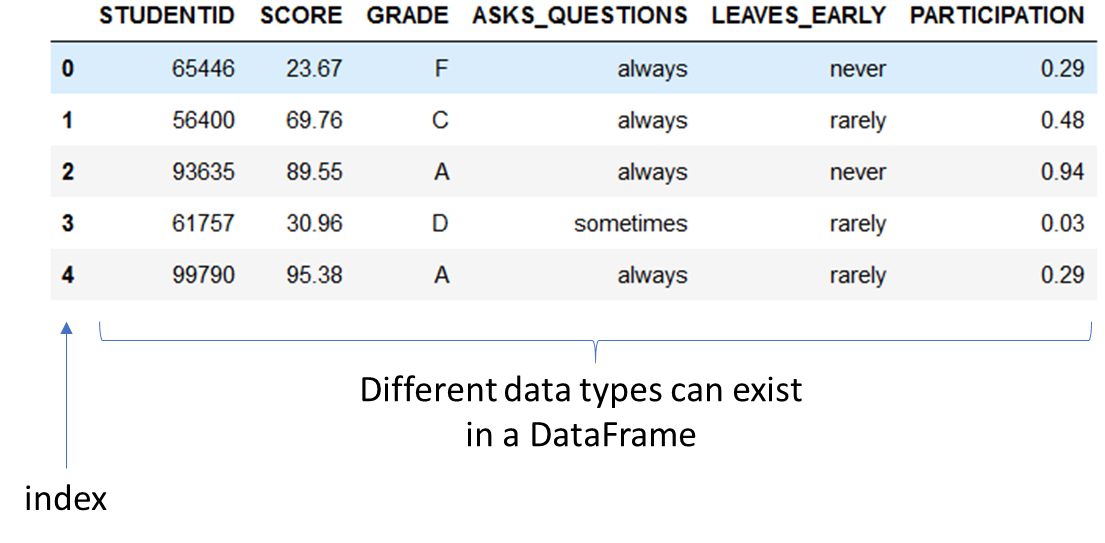
Description automatically generated

There are two things you must know about: Series and DataFrame.

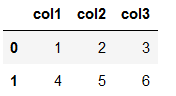
Series is a one-dimensional ndarray with axis labels. When we create a series, this is what be obtain:  
series1 = pd.Series([‘Ana’,’Maria’,’Joana’])



And what is a DataFrame? It’s basically a table with data…

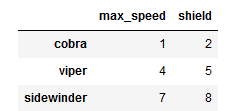


A DataFrame can be created:  
  
df1 = pd.DataFrame([[1,2,3],[4,5,6]], columns=['col1', 'col2', 'col3'])



And the indexes can be numeric, like the example before, or not. For example:

df2 = pd.DataFrame([[1, 2], [4, 5], [7, 8]],index=['cobra', 'viper', 'sidewinder'],columns=['max\_speed', 'shield'])



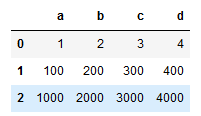
We can also create a DataFrame using a dictionary (explain what a dictionary in python is in case there is someone that does not know):

mydict = [{'a': 1, 'b': 2, 'c': 3, 'd': 4},

{'a': 100, 'b': 200, 'c': 300, 'd': 400},

{'a': 1000, 'b': 2000, 'c': 3000, 'd': 4000 }]

df3 = pd.DataFrame(mydict)



Now we want to select specific rows and columns. There are two ways of doing this (*[iloc](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iloc.html)* and [*loc*](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html) methods). With *iloc*, we select the indexes of the rows/columns we want to select. With *loc* we use the labels.

df.iloc[rows,columns]

Selecting a single column, you can obtain two results: a Serie or a DataFrame.

type(df3.iloc[0]) #Series

type(df3.iloc[[0]]) #DataFrames

Try the different examples and see what you obtain:

df3.iloc[[0, 1]]

df3.iloc[:3]

df3.iloc[0, 1]

df3.iloc[[0, 2], [1, 3]]

df3.iloc[1:3, 0:3]

In python an interval [3:6] translate to an interval from 3 (inclusive) to 6 (exclusive). Meaning that you will get things from positions 3,4 and 5, but not from 6.

Now try the .loc method:

df2.loc['viper']

df2.loc[['viper', 'sidewinder']]

df2.loc['cobra', 'shield']

df2.loc['cobra':'viper', 'max\_speed']  
df2.loc[df['shield'] > 6]

df2.loc[df['shield'] > 6, ['max\_speed']]

Instead of creating a DataFrames you can also be read one from a file. To read from a .txt file, use the command pd.read\_table(). To read from a .csv file, use the command pd.read\_csv().

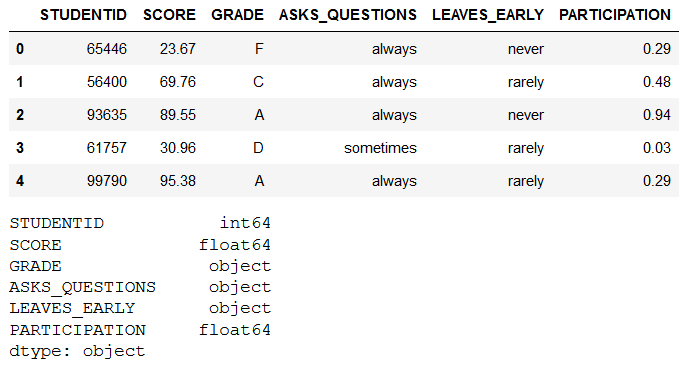
Here we will read from a .csv file. The data was taken from this Kaggle: <https://www.kaggle.com/c/moodychallenge17>

df=pd.read\_csv('predictionchallenge1\\M2017\_train.csv')

Now to have a quick look at what is on this file, and what is the type of data we have:

df.head()

df.dtypes()



We can see, that this is a file that contains students grades information. Let’s explore a bit more…

How many rows and columns does the table have?

df.shape

(839,6)

#(rows,columns)

What are the possible answers to the questions “ASKS\_QUESTIONS” “LEAVES\_EARLY”?

display(df.ASKS\_QUESTIONS.unique())

display(df.LEAVES\_EARLY.unique())



Curious about SCORE and PARTICIPATION distribution? Lets use DataFrame.describe() and DataFrame.plot.kde()

*“In statistics,* [*kernel density estimation*](https://en.wikipedia.org/wiki/Kernel_density_estimation) *(KDE) is a non-parametric way to estimate the probability density function (PDF) of a random variable”*

First “loc” the desired columns, then you can describe the variables:

df.loc[:,['SCORE','PARTICIPATION']].describe()

We see that SCORE ranges from 1.22-98.88 and PARTICIPATION from 0-1

df.loc[:,['SCORE']].plot.kde()

Ever heard of boxplot? Is basically the graphic representation of the .describe() method.

df.boxplot(column='SCORE', by='GRADE')

This plot shows the mean, max, min, etc… for the SCORE for each grade instead of global the mean, max, min, etc…

Can we obtain a pie chart from the LEAVES\_EARLY column?

First we need to have the total of times each answer appears:

df.LEAVES\_EARLY.value\_counts()



Now we can plot the pie plot:

df.LEAVES\_EARLY.value\_counts().plot.pie(figsize=(5, 5))

Now that we know a little bit more the database we can play a little with it.

You can add columns. For example, to add a the column ‘final\_grade’. Imagine that the formula to get the final grade is SCORE\*0.8+PARTICIPATION\*100\*0.2. To add this to the table, just do:

df['final\_grade']=df['SCORE']\*0.8+df['PARTICIPATION']\*100\*0.2

(You can do mathematical operations directly with the the DataFrame)

You can also delete data.

df.drop([‘final\_grade’])

It is possible to filter the data. Lets say that we want to obtain all rows where the SCORE is greater that 95 to highlight the efforts of these students.

best\_students=df[df.SCORE > 95]

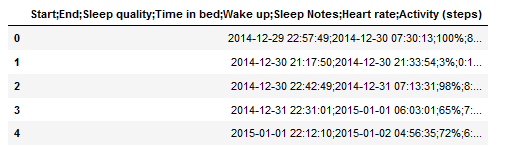
Let’s now analyse another DataBase: <https://www.kaggle.com/danagerous/sleep-data>

It’s a sleep data base…



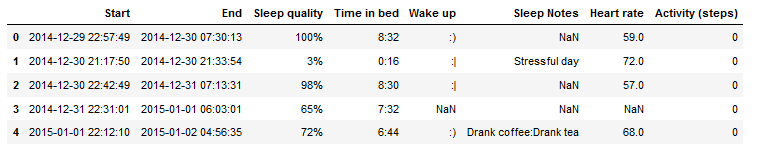
First read the file:

sleep\_data=pd.read\_csv('sleepdata.csv')#,sep=';')   
display(sleep\_data.head())



Oops…something went wrong. You notice the “;” between the numbers? Try adding the argument sep=’;’:

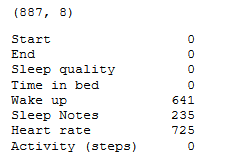
sleep\_data=pd.read\_csv('sleepdata.csv',sep=';')   
display(sleep\_data.head())



Notice the NaN values…sometimes this happens. Due to incorrect introduction of data into de database or because the information was not acquired. Lets check how many NaN values are in each column…

display(sleep\_data.shape)

display(sleep\_data.isnull().sum(axis = 0))



We see that 82% of Heart rate, 26% of sleep notes and 72% of wake up values are Nan. There are two most common ways of dealing with this: deleting all rows that contain NaN, replacing missing values for the global mean. In this case the mean of the heart rate is not informative since we only have 18% of values, and if we drop all Nans, we lose all data. If we where dealing with time series, we could interpolate, but this is not the case.

Even if in this case none of the solutions are good we are going to show you the methods to do it:

#here we replace nan values for heart rate mean  
sleep\_data['HR']=sleep\_data['Heart rate'].replace(np.nan,sleep\_data['Heart rate'].mean())  
display(sleep\_data.head())  
#here we drop all rows that contain nan values  
test2=sleep\_data.dropna().reset\_index()  
display(test2.head())  
#notice we lost almost all rows  
display(test2.shape)

We can also get rid of the columns that we consider non informative. But before that let’s see what is in these NaN infested columns.

display(sleep\_data['Wake up'].unique())  
display(sleep\_data['Sleep Notes'].unique())

The Sleep Notes seem to be things that could affect sleep. The field is a combination of ['Stressful day','Drank coffee','Drank tea','Worked out','Ate late'] separated by “:”. The smileys probably represent how the person felt when they woke up. I would also forget to fill that field to be honest…



Now, we are only interested in the columns without NaN values, so lets drop the others.

sleep\_data=sleep\_data.drop(columns=['Wake up','Sleep Notes','Heart rate'])  
display(sleep\_data.head())

If we use the describe method, it will only look at the numeric type columns.

display(sleep\_data.dtypes)  
display(sleep\_data.describe())

Notice that the Sleep quality was not described…maybe because that “%” symbol? Lets get rid of it.

sleep\_data['Sleep quality']=sleep\_data['Sleep quality'].str.replace('%','').astype(int)

Also, the End and Start are dates. However, they appear are “object”. Let’s change that and see if we can calculate the time lapse.

sleep\_data['End']=pd.to\_datetime(sleep\_data['End'])  
sleep\_data['Start']=pd.to\_datetime(sleep\_data['Start'])  
sleep\_data['elapsed\_time']= sleep\_data['End']-sleep\_data['Start']

Yes…elapsed time is the same as Time in bed. We only wanted to show you that you can subtract date types. Here is another thing you can do with this new column:

sleep\_data['elapsed time (s)']=sleep\_data['elapsed\_time'].dt.total\_seconds()

This method gives the elapsed time in seconds.

Now we can plot, for example, time in bed vs sleep quality to see if there is any correlation. Be sure to order the DataFrame, otherwise you will have a hard time noticing tendencies if there is any.

df\_=sleep\_data.sort\_values(by=['elapsed time (s)'])  
df\_.plot(x='Time in bed',y='Sleep quality')

You can also check relations between the other columns. Remove the commentary from the line you want to test.

#sleep\_data.plot.scatter(x='elapsed time (s)',y='Activity (steps)')  
sleep\_data.plot.scatter(x='elapsed time (s)',y='Sleep quality')  
#sleep\_data.plot.scatter(x='Activity (steps)',y='Sleep quality')

Finally, lets say we want to save the DataFrame description in a file. We can do:

summ=sleep\_data.describe()  
display(summ)  
summ.to\_csv('summary.txt', header=True, index=True, sep=';', mode='a')

Notice that now the elapsed time appears on the summary!

And you can latter access it:

test=pd.read\_table('summary.txt',sep=';')  
display(test)

Thank you for your attention, and remember…Google is your best friend.

