Data Analysis

Pandas has many useful built-in functions to ease data analysis. We will present some here, but take your time to discover them.

Overlay Structure

A very convenient one is the possibility to group the dataframe according to one of the columns. As an example if you have weather data, and you are interested in the average temperature per measuring station. You can group the dataframe by the name of the station (in our example the weather station name is stored in the column "NAME") and get the corresponding average temperature.

```
df.groupby(["NAME"]).mean()
```

```
In [6]: df.groupby(["NAME"]).mean()
```

Out[6]:

NAME	LATITUDE	LONGITUDE	PRCP	TAVG	TMAX	TMIN	TSUN	TSUN_ATTRIBUTES
ADELAIDE AIRPORT ALERT, AS	-34.9552	138.5356	1.34625	<NA>	<NA>	<NA>	<NA>	<NA>
ADELAIDE AIRPORT, AS	-34.9524	138.5204	1.29	61.991667	70.758333	53.241667	<NA>	<NA>
ADELAIDE CLARENCE GARDENS BOW, AS	-34.9726	138.579	1.646	<NA>	<NA>	<NA>	<NA>	<NA>
ADELAIDE GLEN OSMOND, AS	-34.9464	138.6519	2.007273	<NA>	<NA>	<NA>	<NA>	<NA>
ADELAIDE HOPE VALLEY RESERVOI, AS	-34.8564	138.6844	1.925	<NA>	<NA>	<NA>	<NA>	<NA>
...	...	...	...	...	...	...	...	...
WILLIAMSTOWN, AS	-34.6717	138.8885	2.060833	<NA>	<NA>	<NA>	<NA>	<NA>
WOODBRIDGE TWP 1.1 ESE, NJ US	40.5554	-74.274	3.960833	<NA>	<NA>	<NA>	<NA>	<NA>
WOODBRIDGE TWP 1.1 NNE, NJ US	40.575911	-74.284687	4.2725	<NA>	<NA>	<NA>	<NA>	<NA>
WOODSIDE INVERBRACKIE ALERT, AS	-34.9472	138.9261	2.379167	<NA>	<NA>	<NA>	<NA>	<NA>
ZÜRICH FLUNTERN, SZ	47.3831	8.5667	2.820833	52.141667	59.783333	44.491667	<NA>	<NA>

231 rows × 8 columns

If you are interested in a subgroup of a subgroup, Pandas offers the possibility to group the dataframe by a list of columns - this will generate subgroups accordingly

```
df.groupby(["NAME","PRCP"]).mean()
```

```
In [9]: df.groupby(["NAME","PRCP"]).mean()
```

Out[9]:

NAME	PRCP	LATITUDE	LONGITUDE	TAVG	TMAX	TMIN	TSUN	TSUN_ATTRIBUTES
ADELAIDE AIRPORT ALERT, AS	0.33	-34.9552	138.5356	<NA>	<NA>	<NA>	<NA>	<NA>
	0.39	-34.9552	138.5356	<NA>	<NA>	<NA>	<NA>	<NA>
	0.60	-34.9552	138.5356	<NA>	<NA>	<NA>	<NA>	<NA>
	1.17	-34.9552	138.5356	<NA>	<NA>	<NA>	<NA>	<NA>
	1.21	-34.9552	138.5356	<NA>	<NA>	<NA>	<NA>	<NA>
...	...	...	...	...	...	...	...	...
ZÜRICH FLUNTERN, SZ	3.43	47.3831	8.5667	56.6	66.5	46.6	<NA>	<NA>
	3.64	47.3831	8.5667	49.7	55.5	43.9	<NA>	<NA>
	4.69	47.3831	8.5667	43.8	51.0	36.5	<NA>	<NA>
	5.39	47.3831	8.5667	68.1	77.0	59.3	<NA>	<NA>
	5.72	47.3831	8.5667	62.4	70.8	54.0	<NA>	<NA>

2283 rows × 7 columns

The order of the elements in the list corresponds to the substructure Pandas applies This means that df.groupby(["PRCP","NAME"]).mean() is not the same as df.groupby(["NAME","PRCP"]).mean()

Adding Columns and Rows

Sometimes we want to add an additional column to a dataframe. This can be done by just assigning a series as a new column to the dataframe. Use a name that was not used so far and pandas adds it as a new column. Make sure the series has the same size as the rest of the dataframe. Lets say we want to add as an additional information the variation in temperature. We want to store in an additional column the difference between the minimum and the maximum temperature. We can first calculate the difference between the maximum and the minimum temperature measured:

```
df_difft = df["TMAX"]-df["TMIN"]
```

This results in a pandas series with the same number of rows as the original dataframe.

We can now just generate a new dataframe column by assigning this series to the dataframe by choosing the corresponding columnname

```
df["DIFFT"] = df["TMAX"]-df["TMIN"]
```

We could also add as an information if the station of an entry is on the north hemisphere by adding a column to the dataframe, which will have the value "TRUE" for all stations lying in the northern hemisphere and FALSE otherwise

```
df["N_HEM"] =  (df["LATITUDE"] > 0)
```

If you want to add another row to your dataframe you can make use of the "append" functionality. As "append" does not change the dataframe (is immutable) do not forget to reassign the new dataframe. We will enter a copy of the fourth row to avoid all the typing for our weather data, but you can define your new row in the form of a dictionary completely from scratch (as indicated in the code below in a shortened version) For our weather data this could look like that:

```
new_row = df.iloc[3]
#new_row = {'STATION:'US1NJMD0086', 'NAME':'HIGHLAND', 'LATITUDE':40.50542, 'LONGITUDE':-74.42978}
#append row to the dataframe
df = df.append(new_row, ignore_index=True)
```

Changing parts of the dataframe

Sometimes we want to apply changes to a whole column. If we for example want to change the unit of the temperature from Fahrenheit to Celsius, we can do that directly to a whole column. We first distract 32 and then multiply it by 5/9.0 and then you have your average data stored in Celcius.

```
df_new["TAVG"] = df_new["TAVG"]-32
df_new["TAVG"] = df_new["TAVG"]*(5/9.0)
```

Maybe you want to do changes to the whole dataframe, and not only a colum. For that pandas offers a higher order function called apply. A higher order function is a function that takes as an argument another function. With the apply method pandas applies the function to each data entry. If you have rows and columns filled with numbers and you want to have the square root of these numbers instead you can apply the np.sqrt function to each element like this

```
import numpy as np
df.apply(np.sqrt)
```

```
In [79]: df
```

Out[79]:

	A	B
0	4	9
1	4	9
2	16	25

```
In [76]: import numpy as np
```

```
In [80]: df.apply(np.sqrt)
```

Out[80]:

	A	B
0	2.0	3.0
1	2.0	3.0
2	4.0	5.0