21-pandas

December 15, 2020

1 Pandas

1.1 Pandas

- The **Pandas** library is built on top of the **NumPy** library
- It provides a special kind of two dimensional data structure called `DataFrame`.

```
In [1]: import pandas as pd    # This is the standard way of importing the Pandas library
        import numpy as np
```

1.1.1 Example of `DataFrame`

```
In [2]: wh = pd.read_csv("kumpula-weather-2017.csv")
        wh.head()   # The head method prints the first 5 rows
```

```
Out[2]:    Year  m  d   Time Time zone  Precipitation amount (mm)  Snow depth (cm)  \
        0  2017  1  1  00:00       UTC                       -1.0             -1.0
        1  2017  1  2  00:00       UTC                        4.4             -1.0
        2  2017  1  3  00:00       UTC                        6.6              7.0
        3  2017  1  4  00:00       UTC                       -1.0             13.0
        4  2017  1  5  00:00       UTC                       -1.0             10.0

           Air temperature (degC)
        0                     0.6
        1                    -3.9
        2                    -6.5
        3                   -12.8
        4                   -17.8
```

```
In [3]: wh["Snow depth (cm)"].head()
```

```
Out[3]: 0    -1.0
        1    -1.0
        2     7.0
        3    13.0
        4    10.0
        Name: Snow depth (cm), dtype: float64
```

```
In [4]: wh["Air temperature (degC)"].mean()     # Mean temperature

Out[4]: 6.527123287671233

In [5]: wh.drop("Time zone", axis=1).head()     # Return a copy with one column removed, the orig

Out[5]:    Year  m  d   Time  Precipitation amount (mm)  Snow depth (cm)  \
        0  2017  1  1  00:00                       -1.0             -1.0
        1  2017  1  2  00:00                        4.4             -1.0
        2  2017  1  3  00:00                        6.6              7.0
        3  2017  1  4  00:00                       -1.0             13.0
        4  2017  1  5  00:00                       -1.0             10.0

           Air temperature (degC)
        0                     0.6
        1                    -3.9
        2                    -6.5
        3                   -12.8
        4                   -17.8

In [6]: wh.head()     # Original DataFrame is unchanged

Out[6]:    Year  m  d   Time Time zone  Precipitation amount (mm)  Snow depth (cm)  \
        0  2017  1  1  00:00       UTC                       -1.0             -1.0
        1  2017  1  2  00:00       UTC                        4.4             -1.0
        2  2017  1  3  00:00       UTC                        6.6              7.0
        3  2017  1  4  00:00       UTC                       -1.0             13.0
        4  2017  1  5  00:00       UTC                       -1.0             10.0

           Air temperature (degC)
        0                     0.6
        1                    -3.9
        2                    -6.5
        3                   -12.8
        4                   -17.8

In [7]: wh["Rainy"] = wh["Precipitation amount (mm)"] > 5
        wh.head()

Out[7]:    Year  m  d   Time Time zone  Precipitation amount (mm)  Snow depth (cm)  \
        0  2017  1  1  00:00       UTC                       -1.0             -1.0
        1  2017  1  2  00:00       UTC                        4.4             -1.0
        2  2017  1  3  00:00       UTC                        6.6              7.0
        3  2017  1  4  00:00       UTC                       -1.0             13.0
        4  2017  1  5  00:00       UTC                       -1.0             10.0

           Air temperature (degC)  Rainy
        0                     0.6  False
        1                    -3.9  False
```

```
2                    -6.5   True
3                   -12.8  False
4                   -17.8  False
```

1.2 Series

- `Series` is one-dimensional version of `DataFrame`

```
In [8]: s=pd.Series([1, 4, 5, 2, 5, 2])
        s

Out[8]: 0    1
        1    4
        2    5
        3    2
        4    5
        5    2
        dtype: int64

In [9]: s1=pd.Series([1, 4, 5, 2, 5, 2], index=list("abcdef"))
        s1

Out[9]: a    1
        b    4
        c    5
        d    2
        e    5
        f    2
        dtype: int64

In [10]: s1.index

Out[10]: Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
```

We can also attach a name to this series:

```
In [11]: s.name = "Grades"
         s

Out[11]: 0    1
         1    4
         2    5
         3    2
         4    5
         5    2
         Name: Grades, dtype: int64
```

The common attributes of the series are the name, dtype, and size:

```
In [12]: print(f"Name: {s.name}, dtype: {s.dtype}, size: {s.size}")
```

```
Name: Grades, dtype: int64, size: 6
```

```
In [13]: s[1]              # Indexing
```

```
Out[13]: 4
```

```
In [14]: s1["b"]
```

```
Out[14]: 4
```

```
In [15]: s2=s[[0,5]]                    # Fancy indexing
         print(s2)
```

```
0    1
5    2
Name: Grades, dtype: int64
```

```
In [16]: t=s[-2:]                    # Slicing
         t
```

```
Out[16]: 4    5
         5    2
         Name: Grades, dtype: int64
```

```
In [17]: t[4]                        # t[0] would give an error
```

```
Out[17]: 5
```

1.3 Creation of dataframes

The DataFrame is essentially a two dimensional object, and it can be created in three different ways:

- out of a two dimensional NumPy array
- out of given columns
- out of given rows

1.3.1 Creating DataFrames from a NumPy array

```
In [18]: df=pd.DataFrame(np.random.randn(2,3), columns=["First", "Second", "Third"], index=["a",
         df
```

```
Out[18]:      First    Second     Third
         a -1.29709  0.030407 -0.591453
         b -1.18542  1.482094 -1.696548
```

```
In [19]: df.index                              # These are the "row names"
```

```
Out[19]: Index(['a', 'b'], dtype='object')
```

```
In [20]: df.columns                            # These are the "column names"
```

```
Out[20]: Index(['First', 'Second', 'Third'], dtype='object')
```

1.3.2 Creating DataFrames from columns

```
In [21]: s1 = pd.Series([1,2,3])
         s1

Out[21]: 0    1
         1    2
         2    3
         dtype: int64

In [22]: s2 = pd.Series([4,5,6], name="b")
         s2

Out[22]: 0    4
         1    5
         2    6
         Name: b, dtype: int64

In [23]: pd.DataFrame(s1, columns=["a"])

Out[23]:    a
         0  1
         1  2
         2  3
```

Multiple columns - given as the dictionary, whose keys give the column names and values are the actual column content

```
In [24]: pd.DataFrame({"a": s1, "b": s2})

Out[24]:    a  b
         0  1  4
         1  2  5
         2  3  6
```

1.3.3 Creating DataFrames from rows

Each row is given as a dict, list, Series, or NumPy array.

```
In [25]: df=pd.DataFrame([{"Wage" : 1000, "Name" : "Jack", "Age" : 21}, {"Wage" : 1500, "Name" :
         df

Out[25]:    Age  Name  Wage
         0   21  Jack  1000
         1   29  John  1500

In [26]: df = pd.DataFrame([[1000, "Jack", 21], [1500, "John", 29]], columns=["Wage", "Name", "A
         df

Out[26]:    Wage  Name  Age
         0  1000  Jack   21
         1  1500  John   29
```

1.4 Accessing columns and rows of a dataframe

```
In [27]: df[0]
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method,
   3077             try:
-> 3078                 return self._engine.get_loc(key)
   3079             except KeyError:


pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


KeyError: 0


During handling of the above exception, another exception occurred:


KeyError                                  Traceback (most recent call last)

<ipython-input-27-ad11118bc8f3> in <module>()
----> 1 df[0]


~\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   2686             return self._getitem_multilevel(key)
   2687         else:
-> 2688             return self._getitem_column(key)
   2689
   2690     def _getitem_column(self, key):


~\Anaconda3\lib\site-packages\pandas\core\frame.py in _getitem_column(self, key)
   2693         # get column
```

```
   2694         if self.columns.is_unique:
-> 2695             return self._get_item_cache(key)
   2696
   2697         # duplicate columns & possible reduce dimensionality


~\Anaconda3\lib\site-packages\pandas\core\generic.py in _get_item_cache(self, item)
   2487         res = cache.get(item)
   2488         if res is None:
-> 2489             values = self._data.get(item)
   2490             res = self._box_item_values(item, values)
   2491             cache[item] = res


~\Anaconda3\lib\site-packages\pandas\core\internals.py in get(self, item, fastpath)
   4113
   4114             if not isna(item):
-> 4115                 loc = self.items.get_loc(item)
   4116             else:
   4117                 indexer = np.arange(len(self.items))[isna(self.items)]


~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method,
   3078                 return self._engine.get_loc(key)
   3079             except KeyError:
-> 3080                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   3081
   3082         indexer = self.get_indexer([key], method=method, tolerance=tolerance)


pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_


KeyError: 0
```

- `loc`: use explicit indices
- `iloc`: use the implicit integer indices

```
In [28]: df.loc[1, "Wage"]
```

```
Out[28]: 1500

In [29]: df.iloc[-1,-1]            # Right lower corner of the DataFrame

Out[29]: 29

In [30]: df.loc[1, ["Name", "Wage"]]

Out[30]: Name    John
         Wage    1500
         Name: 1, dtype: object
```

1.5 Summary statistics

```
In [31]: wh.head()

Out[31]:    Year  m  d   Time Time zone  Precipitation amount (mm)  Snow depth (cm)  \
         0  2017  1  1  00:00       UTC                       -1.0             -1.0
         1  2017  1  2  00:00       UTC                        4.4             -1.0
         2  2017  1  3  00:00       UTC                        6.6              7.0
         3  2017  1  4  00:00       UTC                       -1.0             13.0
         4  2017  1  5  00:00       UTC                       -1.0             10.0

            Air temperature (degC)  Rainy
         0                     0.6  False
         1                    -3.9  False
         2                    -6.5   True
         3                   -12.8  False
         4                   -17.8  False

In [32]: wh2 = wh.drop(["Year", "m", "d", "Time", "Time zone"], axis=1)  # taking averages over
         wh2.mean()

Out[32]: Precipitation amount (mm)    1.966301
         Snow depth (cm)              0.966480
         Air temperature (degC)       6.527123
         Rainy                        0.158904
         dtype: float64
```

- The `describe` method of the `DataFrame` object gives different summary statistics for each (numeric) column.
- This method gives a good overview of the data, and is typically used in the exploratory data analysis phase.

```
In [33]: wh.describe()

Out[33]:          Year           m           d  Precipitation amount (mm)  \
         count   365.0  365.000000  365.000000                 365.000000
         mean   2017.0    6.526027   15.720548                   1.966301
         std       0.0    3.452584    8.808321                   4.858423
```

```
min    2017.0    1.000000    1.000000                  -1.000000
25%    2017.0    4.000000    8.000000                  -1.000000
50%    2017.0    7.000000   16.000000                   0.200000
75%    2017.0   10.000000   23.000000                   2.700000
max    2017.0   12.000000   31.000000                  35.000000

       Snow depth (cm)  Air temperature (degC)
count       358.000000              365.000000
mean          0.966480                6.527123
std           3.717472                7.183934
min          -1.000000              -17.800000
25%          -1.000000                1.200000
50%          -1.000000                4.800000
75%           0.000000               12.900000
max          15.000000               19.600000
```

1.6 Missing data

- The minimum value in both precipitation and snow depth fields is -1. The special value -1 means that on that day there was absolutely no snow or rain, whereas the value 0 might indicate that the value was close to zero.
- The snow depth column has count 358, whereas the other columns have count 365, one measurement/value for each day of the year.

```
In [34]: wh["Snow depth (cm)"].unique()

Out[34]: array([-1.,  7., 13., 10., 12.,  9.,  8.,  5.,  6.,  4.,  3., 15., 14.,
                2., nan,  0.])
```

The `nan` value tells us that the measurement from that day is not available

For non-numeric types the special value `None` is used to denote a missing value, and the dtype is promoted to `object`.

```
In [35]: pd.Series(["jack", "joe", None])

Out[35]: 0    jack
         1     joe
         2    None
         dtype: object
```

The missing values can be located with the `isnull` method:

```
In [36]: wh[wh.isnull().any(axis=1)]

Out[36]:      Year   m   d   Time Time zone  Precipitation amount (mm)  \
         74   2017   3  16  00:00       UTC                        1.8
         163  2017   6  13  00:00       UTC                        0.6
         308  2017  11   5  00:00       UTC                        0.2
         309  2017  11   6  00:00       UTC                        2.0
```

```
313  2017  11  10  00:00       UTC                        3.6
321  2017  11  18  00:00       UTC                       11.3
328  2017  11  25  00:00       UTC                        8.5

     Snow depth (cm)  Air temperature (degC)  Rainy
74               NaN                     3.4  False
163              NaN                    12.6  False
308              NaN                     8.4  False
309              NaN                     7.5  False
313              NaN                     7.2  False
321              NaN                     5.9   True
328              NaN                     4.2   True
```

The `notnull` method works conversively to the `isnull` method.

The `dropna` method of a DataFrame drops columns or rows that contain missing values from the DataFrame, depending on the `axis` parameter.

```
In [37]: wh.dropna().shape    # Default axis is 0

Out[37]: (358, 9)

In [38]: wh.dropna(axis=1).shape # Drops the columns containing missing values

Out[38]: (365, 8)
```

1.7 Converting columns from one type to another

- For converting single columns (a Series) one can use the `pd.to_numeric` function or the map method.
- For converting several columns in one go one can use the `astype` method.

```
In [39]: pd.Series(["1","2"]).map(int)                              # str -> int

Out[39]: 0    1
         1    2
         dtype: int64

In [40]: pd.Series([1,2]).map(str)                                  # int -> str

Out[40]: 0    1
         1    2
         dtype: object

In [41]: pd.to_numeric(pd.Series([1,1.0]), downcast="integer")    # object -> int

Out[41]: 0    1
         1    1
         dtype: int8

In [42]: pd.to_numeric(pd.Series([1,"a"]), errors="coerce")       # conversion error produces Nan
```

```
Out[42]: 0    1.0
         1    NaN
         dtype: float64

In [43]: pd.Series([1,2]).astype(str)                          # works for a single series

Out[43]: 0    1
         1    2
         dtype: object

In [44]: df = pd.DataFrame({"a": [1,2,3], "b" : [4,5,6], "c" : [7,8,9]})
         print(df.dtypes)
         print(df)

a    int64
b    int64
c    int64
dtype: object
   a  b  c
0  1  4  7
1  2  5  8
2  3  6  9


In [45]: df.astype(float)                        # Convert all columns

Out[45]:      a    b    c
         0  1.0  4.0  7.0
         1  2.0  5.0  8.0
         2  3.0  6.0  9.0

In [46]: df2 = df.astype({"b" : float, "c" : str})     # different types for columns
         print(df2.dtypes)
         print(df2)

a      int64
b    float64
c     object
dtype: object
   a    b  c
0  1  4.0  7
1  2  5.0  8
2  3  6.0  9
```

1.8 String processing

```
In [47]: names = pd.Series(["donald", "theresa", "angela", "vladimir"])
         names.str.capitalize()
```

```
Out[47]: 0      Donald
         1     Theresa
         2      Angela
         3    Vladimir
         dtype: object

In [48]: # names.str. # Press the tab key

In [49]: full_names = pd.Series(["Donald Trump", "Theresa May", "Angela Merkel", "Vladimir Putin
         full_names.str.split()  # one column

Out[49]: 0      [Donald, Trump]
         1       [Theresa, May]
         2     [Angela, Merkel]
         3    [Vladimir, Putin]
         dtype: object

In [50]: full_names.str.split(expand=True) # two columns

Out[50]:           0       1
         0    Donald   Trump
         1   Theresa     May
         2    Angela  Merkel
         3  Vladimir   Putin
```

1.9 Catenating datasets

```
In [51]: def makedf(cols, ind):
             data = {c : [str(c) + str(i) for i in ind] for c in cols}
             return pd.DataFrame(data, ind)

In [52]: a=makedf("AB", [0,1])
         a

Out[52]:     A   B
         0  A0  B0
         1  A1  B1

In [53]: b=makedf("AB", [2,3])
         b

Out[53]:     A   B
         2  A2  B2
         3  A3  B3

In [54]: c=makedf("CD", [0,1])
         c

Out[54]:     C   D
         0  C0  D0
         1  C1  D1
```

```
In [55]: d=makedf("BC", [2,3])
         d

Out[55]:     B   C
         2  B2  C2
         3  B3  C3

In [56]: pd.concat([a,b])    # The default axis is 0

Out[56]:     A   B
         0  A0  B0
         1  A1  B1
         2  A2  B2
         3  A3  B3

In [57]: pd.concat([a,c], axis=1)

Out[57]:     A   B   C   D
         0  A0  B0  C0  D0
         1  A1  B1  C1  D1

In [58]: r=pd.concat([a,a])
         r                   # This is not usually what we want!

Out[58]:     A   B
         0  A0  B0
         1  A1  B1
         0  A0  B0
         1  A1  B1

In [59]: ## 1. automatic renumbering of rows:
         pd.concat([a,a], ignore_index=True)

Out[59]:     A   B
         0  A0  B0
         1  A1  B1
         2  A0  B0
         3  A1  B1

In [60]: ## 2. hierarchical indexing
         r2=pd.concat([a,a], keys=['first', 'second'])
         r2

Out[60]:              A   B
         first  0   A0  B0
                1   A1  B1
         second 0   A0  B0
                1   A1  B1

In [61]: r2["A"]["first"][0]
```

```
Out[61]: 'A0'
```

```
In [62]: pd.concat([a,d])
```

```
C:\Users\wench\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Sorting becau
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

```
Out[62]:
```

	A	B	C
0	A0	B0	NaN
1	A1	B1	NaN
2	NaN	B2	C2
3	NaN	B3	C3

```
In [63]: pd.concat([a,d], sort=False)
```

```
Out[63]:
```

	A	B	C
0	A0	B0	NaN
1	A1	B1	NaN
2	NaN	B2	C2
3	NaN	B3	C3

```
In [64]: pd.concat([a,d], join="inner")
```

```
Out[64]:
```

	B
0	B0
1	B1
2	B2
3	B3

1.10 Merging dataframes

```
In [65]: df = pd.DataFrame([[1000, "Jack", 21], [1500, "John", 29]], columns=["Wage", "Name", "A
         df
```

```
Out[65]:
```

	Wage	Name	Age
0	1000	Jack	21
1	1500	John	29

```
In [66]: df2 = pd.DataFrame({"Name" : ["John", "Jack"], "Occupation": ["Plumber", "Carpenter"]})
         df2
```

```
Out[66]:
```

	Name	Occupation
0	John	Plumber
1	Jack	Carpenter

```
In [67]: pd.merge(df, df2)
```

```
Out[67]:    Wage  Name  Age Occupation
         0  1000  Jack   21  Carpenter
         1  1500  John   29    Plumber
```

```
In [68]: df3 = pd.concat([df2, pd.DataFrame({ "Name" : ["James"], "Occupation":["Painter"]})], i
         df3
```

```
Out[68]:     Name Occupation
         0   John    Plumber
         1   Jack  Carpenter
         2  James    Painter
```

```
In [69]: pd.merge(df, df3)                # By default an inner join is computed
```

```
Out[69]:    Wage  Name  Age Occupation
         0  1000  Jack   21  Carpenter
         1  1500  John   29    Plumber
```

```
In [70]: pd.merge(df, df3, how="outer")   # Outer join
```

```
Out[70]:      Wage   Name   Age Occupation
         0  1000.0   Jack  21.0  Carpenter
         1  1500.0   John  29.0    Plumber
         2     NaN  James   NaN    Painter
```

1.11 Aggregates and groupings

```
In [71]: wh.head()
```

```
Out[71]:    Year  m  d   Time Time zone  Precipitation amount (mm)  Snow depth (cm)  \
         0  2017  1  1  00:00       UTC                       -1.0             -1.0
         1  2017  1  2  00:00       UTC                        4.4             -1.0
         2  2017  1  3  00:00       UTC                        6.6              7.0
         3  2017  1  4  00:00       UTC                       -1.0             13.0
         4  2017  1  5  00:00       UTC                       -1.0             10.0

            Air temperature (degC)  Rainy
         0                     0.6  False
         1                    -3.9  False
         2                    -6.5   True
         3                   -12.8  False
         4                   -17.8  False
```

```
In [72]: wh3 = wh.rename(columns={"m": "Month", "d": "Day", "Precipitation amount (mm)" : "Preci
                                  "Snow depth (cm)" : "Snow", "Air temperature (degC)" : "Temper
         wh3.head()
```

Out[72]:

	Year	Month	Day	Time	Time zone	Precipitation	Snow	Temperature	Rainy
0	2017	1	1	00:00	UTC	-1.0	-1.0	0.6	False
1	2017	1	2	00:00	UTC	4.4	-1.0	-3.9	False
2	2017	1	3	00:00	UTC	6.6	7.0	-6.5	True
3	2017	1	4	00:00	UTC	-1.0	13.0	-12.8	False
4	2017	1	5	00:00	UTC	-1.0	10.0	-17.8	False

```
In [73]: groups = wh3.groupby("Month")
         len(groups)
```

```
Out[73]: 12
```

```
In [74]: for key, group in groups:
             print(key, len(group))
```

```
1 31
2 28
3 31
4 30
5 31
6 30
7 31
8 31
9 30
10 31
11 30
12 31
```

```
In [75]: groups.get_group(2).head()                # Group with index two is February
```

Out[75]:

	Year	Month	Day	Time	Time zone	Precipitation	Snow	Temperature	Rainy
31	2017	2	1	00:00	UTC	1.5	4.0	-0.6	False
32	2017	2	2	00:00	UTC	0.2	5.0	-0.8	False
33	2017	2	3	00:00	UTC	-1.0	6.0	-0.2	False
34	2017	2	4	00:00	UTC	2.7	6.0	0.4	False
35	2017	2	5	00:00	UTC	-1.0	7.0	-2.5	False

```
In [76]: groups["Temperature"].mean()
```

```
Out[76]: Month
         1     -2.316129
         2     -2.389286
         3      0.983871
         4      2.676667
         5      9.783871
         6     13.726667
         7     16.035484
         8     16.183871
```

```
9     11.826667
10     5.454839
11     3.950000
12     1.741935
Name: Temperature, dtype: float64
```

In [77]:
```
groups["Precipitation"].sum()
```

Out[77]:
```
Month
1      26.9
2      21.0
3      29.7
4      26.9
5      -5.9
6      59.3
7      14.2
8      70.1
9      51.2
10    173.5
11    117.2
12    133.6
Name: Precipitation, dtype: float64
```

In [78]:
```
wh4 = wh3.copy()
wh4.loc[wh4.Precipitation == -1, "Precipitation"] = 0
wh4.loc[wh4.Snow == -1, "Snow"] = 0
wh4.head()
```

Out[78]:

	Year	Month	Day	Time	Time zone	Precipitation	Snow	Temperature	Rainy
0	2017	1	1	00:00	UTC	0.0	0.0	0.6	False
1	2017	1	2	00:00	UTC	4.4	0.0	-3.9	False
2	2017	1	3	00:00	UTC	6.6	7.0	-6.5	True
3	2017	1	4	00:00	UTC	0.0	13.0	-12.8	False
4	2017	1	5	00:00	UTC	0.0	10.0	-17.8	False

In [79]:
```
wh4.groupby("Month")["Precipitation"].sum()
```

Out[79]:
```
Month
1      38.9
2      35.0
3      41.7
4      39.9
5      16.1
6      76.3
7      31.2
8      86.1
9      65.2
10    184.5
11    120.2
12    140.6
Name: Precipitation, dtype: float64
```

1.11.1 apply

The **apply** method is very generic and only requires that for each group's DataFrame the given function returns a DataFrame, Series, or a scalar.

```
In [80]: wh4.groupby("Month").apply(lambda df : df.sort_values("Temperature"))
```

Out[80]:

Month		Year	Month	Day	Time	Time zone	Precipitation	Snow
1	4	2017	1	5	00:00	UTC	0.0	10.0
	5	2017	1	6	00:00	UTC	0.3	10.0
	3	2017	1	4	00:00	UTC	0.0	13.0
	2	2017	1	3	00:00	UTC	6.6	7.0
	15	2017	1	16	00:00	UTC	0.0	8.0
	1	2017	1	2	00:00	UTC	4.4	0.0
	24	2017	1	25	00:00	UTC	0.6	6.0
	6	2017	1	7	00:00	UTC	5.3	10.0
	16	2017	1	17	00:00	UTC	0.2	8.0
	11	2017	1	12	00:00	UTC	8.0	7.0
	14	2017	1	15	00:00	UTC	0.0	8.0
	23	2017	1	24	00:00	UTC	0.0	6.0
	20	2017	1	21	00:00	UTC	0.4	5.0
	10	2017	1	11	00:00	UTC	0.0	7.0
	19	2017	1	20	00:00	UTC	0.3	5.0
	7	2017	1	8	00:00	UTC	0.0	12.0
	22	2017	1	23	00:00	UTC	0.1	6.0
	30	2017	1	31	00:00	UTC	0.0	4.0
	8	2017	1	9	00:00	UTC	1.1	12.0
	28	2017	1	29	00:00	UTC	2.6	3.0
	0	2017	1	1	00:00	UTC	0.0	0.0
	13	2017	1	14	00:00	UTC	0.1	8.0
	27	2017	1	28	00:00	UTC	1.8	4.0
	29	2017	1	30	00:00	UTC	5.6	5.0
	21	2017	1	22	00:00	UTC	0.2	5.0
	12	2017	1	13	00:00	UTC	0.1	13.0
	17	2017	1	18	00:00	UTC	0.9	8.0
	18	2017	1	19	00:00	UTC	0.0	5.0
	26	2017	1	27	00:00	UTC	0.0	4.0
	9	2017	1	10	00:00	UTC	0.3	9.0
...	...	...	...	...	...	...	...	...
12	340	2017	12	7	00:00	UTC	16.3	0.0
	357	2017	12	24	00:00	UTC	0.0	0.0
	355	2017	12	22	00:00	UTC	0.0	0.0
	338	2017	12	5	00:00	UTC	0.7	0.0
	350	2017	12	17	00:00	UTC	0.0	5.0
	358	2017	12	25	00:00	UTC	5.9	0.0
	334	2017	12	1	00:00	UTC	3.4	0.0
	352	2017	12	19	00:00	UTC	0.2	3.0
	356	2017	12	23	00:00	UTC	7.6	0.0

337	2017	12	4	00:00	UTC	0.0	0.0
335	2017	12	2	00:00	UTC	5.3	5.0
344	2017	12	11	00:00	UTC	1.3	0.0
364	2017	12	31	00:00	UTC	3.2	0.0
346	2017	12	13	00:00	UTC	4.2	5.0
345	2017	12	12	00:00	UTC	35.0	0.0
347	2017	12	14	00:00	UTC	5.2	4.0
348	2017	12	15	00:00	UTC	10.0	10.0
359	2017	12	26	00:00	UTC	7.8	0.0
351	2017	12	18	00:00	UTC	3.5	5.0
343	2017	12	10	00:00	UTC	0.0	0.0
349	2017	12	16	00:00	UTC	1.3	6.0
363	2017	12	30	00:00	UTC	4.1	0.0
354	2017	12	21	00:00	UTC	0.0	0.0
353	2017	12	20	00:00	UTC	3.6	3.0
361	2017	12	28	00:00	UTC	3.7	0.0
360	2017	12	27	00:00	UTC	1.1	0.0
362	2017	12	29	00:00	UTC	7.8	0.0
342	2017	12	9	00:00	UTC	0.2	0.0
336	2017	12	3	00:00	UTC	7.2	0.0
341	2017	12	8	00:00	UTC	2.0	0.0

		Temperature	Rainy
Month			
1	4	-17.8	False
	5	-17.8	False
	3	-12.8	False
	2	-6.5	True
	15	-4.2	False
	1	-3.9	False
	24	-3.8	False
	6	-3.8	True
	16	-3.5	False
	11	-2.8	True
	14	-2.8	False
	23	-2.2	False
	20	-1.8	False
	10	-1.6	False
	19	-0.6	False
	7	-0.5	False
	22	0.1	False
	30	0.2	False
	8	0.5	False
	28	0.6	False
	0	0.6	False
	13	0.8	False
	27	0.8	False
	29	1.0	True

```
      21           1.0  False
      12           1.1  False
      17           1.1  False
      18           1.6  False
      26           1.6  False
      9            1.7  False
...                ...    ...
12    340         -0.8   True
      357         -0.3  False
      355         -0.1  False
      338          0.0  False
      350          0.1  False
      358          0.3   True
      334          0.9  False
      352          1.0  False
      356          1.2   True
      337          1.3  False
      335          1.4   True
      344          1.4  False
      364          1.6  False
      346          1.6  False
      345          1.6   True
      347          1.6   True
      348          1.7   True
      359          1.9   True
      351          2.0  False
      343          2.0  False
      349          2.4  False
      363          2.5  False
      354          2.5  False
      353          2.6  False
      361          2.8  False
      360          3.8  False
      362          3.8   True
      342          4.2  False
      336          5.0   True
      341          5.2  False

[365 rows x 9 columns]
```