Pandas

Prof.Dr. Bahadır AKTUĞ

803400815021 Machine Learning with Python

**Compiled from sources given in the references.*

Pandas

- Pandas is a very popular module which provides spread-sheet like data structures for easy processing.
- Pandas uses Numpy internally for keeping data
- Pandas can also be used with SciPy and already is integrated with Matplotlib.
- There are basically two data structures in Pandas:
  - Series
  - DataFrame
- Series are for keeping one-dimensional data and for doing operations over them.
- Dataframes are for two-dimensional data
- There used to be structures Panel and Panel4D (now deprecated)

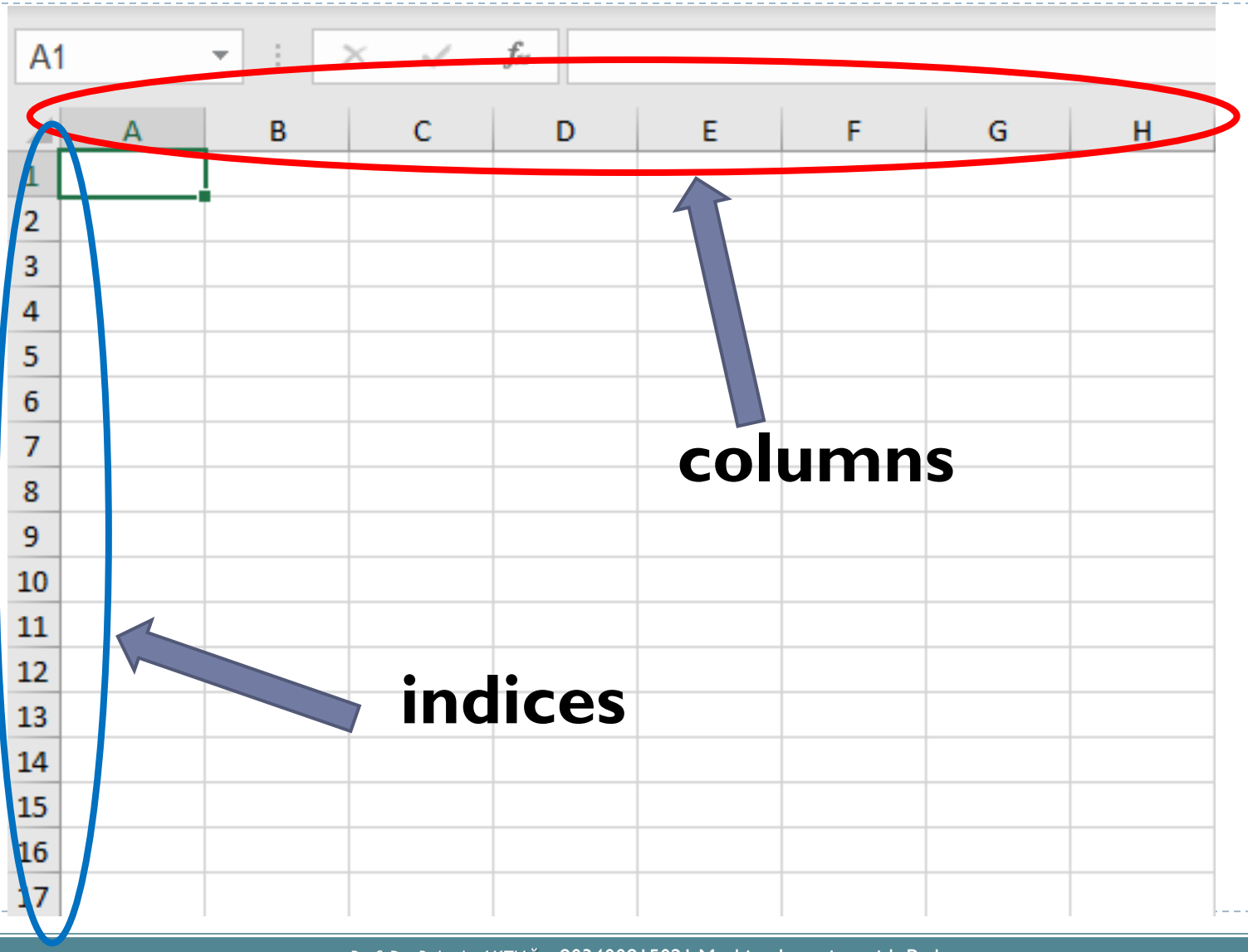
What can Pandas do for you

- Provides tools for reading and writing data from/to a wide variety of formats including reading directly from URLs.
- Helps handling and organizing data in a smart and practical way (indexing, slicing etc.)
- Provides tools to quickly hand missing data
- Helps filtering, restructuring data
- Provides an easy way of row/column based operations just like a spreadsheet (in a programmatic way!)
- Provides tools to do many statistical analysis
- Allows quick visualization of data through integrated plotting system

Index

- Both series and dataframes have indices.
- Indices are more than just number or strings. They have various uses in data processing.
- There is always a default number based index for each series or dataframe
- Indices are used for labelling data. In fact both row and columns have indices but they have different names.
- For rows, «index» is used whereas «column_name» is used for columns.
- While indices are for accessing the data,

Index and Column



Series

- While series are like Numpy array at first sight, they are much more powerful and flexible.
- Series have indices even if they are one-dimensional.
- The index for series can be defined explicitly otherwise an integer index will automatically is assigned implicitly.

```
import pandas as pd
a = pd.Series([5, 54, 12, 22, 15, 87])
print(a)
```

```
0    5
1   54
2   12
3   22
4   15
5   87
dtype: int64
```

Series

- Series can be created by many ways:
  - from lists
  - from dictionaries
  - from Numpy arrays

Series

- Create a Pandas Series from a list:

```
import pandas as pd

# create Pandas Series with default index values
# default index ranges is from 0 to len(list) - 1
x = pd.Series(['Geeks', 'for', 'Geeks'])

# print the Series
print(x)
```

```
0    Geeks
1      for
2    Geeks
dtype: object
```

Series

- Create a Pandas Series from a list with explicit index:

```
# import pandas lib. as pd
import pandas as pd

# create Pandas Series with define indexes
x = pd.Series([10, 20, 30, 40, 50], index =['a', 'b', 'c', 'd', 'e'])

# print the Series
print(x)
```

```
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

Explicit Index

```
import pandas as pd
a = pd.Series([5, 54, 12, 22, 15, 87])
print(a.index)
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
print(a.values)
[ 5 54 12 22 15 87]
```

- Indices can be given separately.

```
import pandas as pd
indisler = ['Ali','Ahmet','Mehmet','Veli','Arda','Doruk']
degerler = [5, 54, 12, 22, 15, 87]
a = pd.Series(degerler,index=indisler)
print(a)
```

```
Ali       5
Ahmet    54
Mehmet   12
Veli     22
Arda     15
Doruk    87
dtype: int64
```

Series

- Create a Pandas Series from a Numpy array:

```
# importing Pandas & numpy
import pandas as pd
import numpy as np

# numpy array
data = np.array(['a', 'b', 'c', 'd', 'e'])

# creating series
s = pd.Series(data)
print(s)
```

Output:

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

Series

- Create a Pandas Series from a dictionary:

```
# import the pandas lib as pd
import pandas as pd

# create a dictionary
dictionary = {'A' : 10, 'B' : 20, 'C' : 30}

# create a series
series = pd.Series(dictionary)

print(series)
```

Output:

```
A    10
B    20
C    30
dtype: int64
```

Series

- When a series is created from a dictionary, the keys of the dictionary will be the indices of the series implicityly.
- One way to deal with it is to create a series with explicit index:

```
# import the pandas lib as pd
import pandas as pd

# create a dictionary
dictionary = {'A' : 50, 'B' : 10, 'C' : 80}

# create a series
series = pd.Series(dictionary, index =['B',
'C', 'A'])

print(series)
```

Output:

```
B    10
C    80
A    50
dtype: int64
```

Series

- The indices for series can be defined explicitly.
- Duplicate Index values are also possible.
- Index object can retrived through «index» command.

```
import pandas as pd
a = pd.Series([5, 54, 12, 22, 15, 87], index=['a','b','c','d','e','f'])
print(a)
```

```
a     5
b    54
c    12
d    22
e    15
f    87
dtype: int64
```

```
>>> a.index
Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
```

Series

- If the series is generated from a Python dictionary then the keys of the dictionary are assigned indices.
- When dictionary keys are used as indices, they are taken as sorted.

```
import pandas as pd
a = pd.Series({'a':5, 'b':54, 'c':12, 'd':22, 'e':15, 'f':87})
print(a)
```

```
a     5
b    54
c    12
d    22
e    15
f    87
dtype: int64
```

Series

- If a Python dictionary is given already with explicit index then explicit index overwrites the keys of the dictionary.

```
import pandas as pd
a = pd.Series({'a':5, 'b':54, 'c':12, 'd':22, 'e':15, 'f':87},
index = ['c', 'd'])
print(a)
```

```
c    12
d    22
dtype: int64
```

Series

- Accessing an element is possible through its index.
- Accessing multiple elements is possible through index slicing.
- Numpy type array indexing is also possible for Series.

```
import pandas as pd
a = pd.Series({'a':5, 'b':54, 'c':12, 'd':22, 'e':15, 'f':87})
```

```
>>> a['c']
12
>>> a[0]
5
>>> a[-1]
87
```

```
>>> a['b':'e']
b    54
c    12
d    22
e    15
dtype: int64
```

Series

- Indices particularly useful to align different Series

```
import pandas as pd
>>> a = pd.Series({'a':5, 'b':54, 'c':12, 'd':22, 'e':15, 'f':87})
>>> b = pd.Series({'b':3, 'c':8, 'f':10})
>>> a+b

a     NaN
b    57.0
c    20.0
d     NaN
e     NaN
f    97.0
dtype: float64
```

DataFrame

- DataFrames consists of multiple series.

```
import pandas as pd
ogrenciler = {
'isim' : ['Ali','Ahmet','Mehmet','Veli','Arda','Doruk'],
'not'  : [5, 54, 12, 22, 15, 87],
'yas'  : [21, 24, 18, 28, 20, 22]}
a = pd.DataFrame(ogrenciler)
print(a)
```

```
0    Ali   5   21
1  Ahmet  54   24
2 Mehmet  12   18
3   Veli  22   28
4   Arda  15   20
5  Doruk  87   22
```

Set Index

- df.set_index() assigns one or multiple columns as index:

```
>>> df
        city  year  population
0   İstanbul  1970     3019032
1   İstanbul  1980     4741890
2   İstanbul  1990     7195773
3   İstanbul  2000    10018735
4     Ankara  1970     2041658
5     Ankara  1980     2854689
6     Ankara  1990     3236378
7     Ankara  2000     4007860
8      İzmir  1970     1427173
9      İzmir  1980     1976763
10     İzmir  1990     2694770
11     İzmir  2000     3370866
12     Konya  1970     1280239
13     Konya  1980     1562139
14     Konya  1990     1752658
15     Konya  2000     2192166
16     Adana  1970     1035377
17     Adana  1980     1485743
18     Adana  1990     1549233
19     Adana  2000     1849478
20     Bursa  1970      847884
21     Bursa  1980     1148492
22     Bursa  1990     1596161
23     Bursa  2000     2125140
```

```
>>> df.set_index('city')
          year  population
city
İstanbul  1970     3019032
İstanbul  1980     4741890
İstanbul  1990     7195773
İstanbul  2000    10018735
Ankara    1970     2041658
Ankara    1980     2854689
Ankara    1990     3236378
Ankara    2000     4007860
İzmir     1970     1427173
İzmir     1980     1976763
İzmir     1990     2694770
İzmir     2000     3370866
Konya     1970     1280239
Konya     1980     1562139
Konya     1990     1752658
Konya     2000     2192166
Adana     1970     1035377
Adana     1980     1485743
Adana     1990     1549233
Adana     2000     1849478
Bursa     1970      847884
Bursa     1980     1148492
Bursa     1990     1596161
Bursa     2000     2125140
```

Resetting Index

- df.reset_index() resets index. It basically does two things:
  - Moving the current index to a new column
  - Creating a new integer-based index.

```
>>> df
        city  year  population
0   İstanbul  1970     3019032
1   İstanbul  1980     4741890
2   İstanbul  1990     7195773
3   İstanbul  2000    10018735
4     Ankara  1970     2041658
5     Ankara  1980     2854689
6     Ankara  1990     3236378
7     Ankara  2000     4007860
```

```
>>> df.reset_index()
   index      city  year  population
0      0  İstanbul  1970     3019032
1      1  İstanbul  1980     4741890
2      2  İstanbul  1990     7195773
3      3  İstanbul  2000    10018735
4      4    Ankara  1970     2041658
5      5    Ankara  1980     2854689
6      6    Ankara  1990     3236378
7      7    Ankara  2000     4007860
```

```
>>> df.reset_index().reset_index()
   level_0  index      city  year  population
0        0      0  İstanbul  1970     3019032
1        1      1  İstanbul  1980     4741890
2        2      2  İstanbul  1990     7195773
3        3      3  İstanbul  2000    10018735
4        4      4    Ankara  1970     2041658
5        5      5    Ankara  1980     2854689
6        6      6    Ankara  1990     3236378
7        7      7    Ankara  2000     4007860
```

Reindex

- Both rows and columns can be re-indexed.
- A value of NaN is assigned for new indices without a corresponding row.

```
>>> dfNew
          year  population
city
İstanbul  1970     3019032
Ankara    1970     2041658
İzmir     1970     1427173
Konya     1970     1280239
Adana     1970     1035377
Bursa     1970      847884
```

```
>>> dfNew.reindex(['Ankara','Samsun','Bolu','İzmir','İstanbul'])
             year  population
city
Ankara     1970.0   2041658.0
Samsun        NaN         NaN
Bolu          NaN         NaN
İzmir      1970.0   1427173.0
İstanbul   1970.0   3019032.0
```

```
>>> dfNew.reindex(['population','area'],axis='columns')
          population  area
city
İstanbul     3019032   NaN
Ankara       2041658   NaN
İzmir        1427173   NaN
Konya        1280239   NaN
Adana        1035377   NaN
Bursa         847884   NaN
```

Checking Index

```
>>> df
          year  population
city
İstanbul  1970     3019032
İstanbul  1980     4741890
İstanbul  1990     7195773
İstanbul  2000    10018735
Ankara    1970     2041658
Ankara    1980     2854689
Ankara    1990     3236378
Ankara    2000     4007860
İzmir     1970     1427173
İzmir     1980     1976763
İzmir     1990     2694770
İzmir     2000     3370866
Konya     1970     1280239
Konya     1980     1562139
Konya     1990     1752658
Konya     2000     2192166
Adana     1970     1035377
Adana     1980     1485743
Adana     1990     1549233
Adana     2000     1849478
Bursa     1970      847884
Bursa     1980     1148492
Bursa     1990     1596161
Bursa     2000     2125140
```

```
>>> df.index
Index(['İstanbul', 'İstanbul', 'İstanbul', 'İstanbul', 'Ankara', 'Ankara',
       'Ankara', 'Ankara', 'İzmir', 'İzmir', 'İzmir', 'İzmir', 'Konya',
       'Konya', 'Konya', 'Konya', 'Adana', 'Adana', 'Adana', 'Adana', 'Bursa',
       'Bursa', 'Bursa', 'Bursa'],
      dtype='object', name='city')
```

```
>>> df.index.is_unique
False
```

```
>>> df.index.duplicated()
array([False,  True,  True,  True, False,  True,  True,  True, False,
        True,  True,  True, False,  True,  True,  True, False,  True,
        True,  True, False,  True,  True,  True])
```

```
>>> df.loc[~df.index.duplicated(), :]
          year  population
city
İstanbul  1970     3019032
Ankara    1970     2041658
İzmir     1970     1427173
Konya     1970     1280239
Adana     1970     1035377
Bursa     1970      847884
```

Hierarchical Indexing (MultiIndex)

- Multi-index is necessary when indexing data requires more than one column to be unique

	city	year	population
0	İstanbul	1970	3019032
1	İstanbul	1980	4741890
2	İstanbul	1990	7195773
3	İstanbul	2000	10018735
4	Ankara	1970	2041658
5	Ankara	1980	2854689
6	Ankara	1990	3236378
7	Ankara	2000	4007860
8	İzmir	1970	1427173
9	İzmir	1980	1976763
10	İzmir	1990	2694770
11	İzmir	2000	3370866
12	Konya	1970	1280239
13	Konya	1980	1562139
14	Konya	1990	1752658
15	Konya	2000	2192166
16	Adana	1970	1035377
17	Adana	1980	1485743
18	Adana	1990	1549233
19	Adana	2000	1849478
20	Bursa	1970	847884
21	Bursa	1980	1148492
22	Bursa	1990	1596161
23	Bursa	2000	2125140

- In the example to the left, neither city nor the year is sufficient to identify any row uniquely.
- One way to deal with this is to use a tuple as index. i.e.
  - (Ankara,1970)
  - (Adana,1980)
- However, this approach is limited. For instance, you can not select the populations in a particular year.

Hierarchical Indexing (MultiIndex)

- A multi-index can be created directly from the existing columns:
  - df = df.set_index(['city','year'])
- Multi-index can also be created manually by using commands:
  - pd.MultiIndex.from_arrays()
  - pd.MultiIndex.from_tuples()
  - pd.MultiIndex.from_product()

```
>>> df.set_index(['city','year'])
                population
city     year
İstanbul 1970     3019032
         1980     4741890
         1990     7195773
         2000    10018735
Ankara   1970     2041658
         1980     2854689
         1990     3236378
         2000     4007860
İzmir    1970     1427173
         1980     1976763
         1990     2694770
         2000     3370866
Konya    1970     1280239
         1980     1562139
         1990     1752658
         2000     2192166
Adana    1970     1035377
         1980     1485743
         1990     1549233
         2000     1849478
Bursa    1970      847884
         1980     1148492
         1990     1596161
         2000     2125140
```

Hierarchical Indexing (MultiIndex)

```
index = pd.MultiIndex.from_arrays(
[['İstanbul','İstanbul','İstanbul','İstanbul',
'Ankara','Ankara','Ankara','Ankara',
'İzmir','İzmir','İzmir','İzmir',
'Konya','Konya','Konya','Konya',
'Adana','Adana','Adana','Adana',
'Bursa','Bursa','Bursa','Bursa'],
[1970,1980,1990,2000,
1970,1980,1990,2000,
1970,1980,1990,2000,
1970,1980,1990,2000,
1970,1980,1990,2000,
1970,1980,1990,2000]])
```

```
>>> df
      city  year  population
0   İstanbul  1970     3019032
1   İstanbul  1980     4741890
2   İstanbul  1990     7195773
3   İstanbul  2000    10018735
4     Ankara  1970     2041658
5     Ankara  1980     2854689
6     Ankara  1990     3236378
7     Ankara  2000     4007860
8      İzmir  1970     1427173
9      İzmir  1980     1976763
10     İzmir  1990     2694770
11     İzmir  2000     3370866
12     Konya  1970     1280239
13     Konya  1980     1562139
14     Konya  1990     1752658
15     Konya  2000     2192166
16     Adana  1970     1035377
17     Adana  1980     1485743
18     Adana  1990     1549233
19     Adana  2000     1849478
20     Bursa  1970      847884
21     Bursa  1980     1148492
22     Bursa  1990     1596161
23     Bursa  2000     2125140
```

```
>>> df.index = index
>>> df
                    city  year  population
İstanbul 1970   İstanbul  1970     3019032
         1980   İstanbul  1980     4741890
         1990   İstanbul  1990     7195773
         2000   İstanbul  2000    10018735
Ankara   1970     Ankara  1970     2041658
         1980     Ankara  1980     2854689
         1990     Ankara  1990     3236378
         2000     Ankara  2000     4007860
İzmir    1970      İzmir  1970     1427173
         1980      İzmir  1980     1976763
         1990      İzmir  1990     2694770
         2000      İzmir  2000     3370866
Konya    1970      Konya  1970     1280239
         1980      Konya  1980     1562139
         1990      Konya  1990     1752658
         2000      Konya  2000     2192166
Adana    1970      Adana  1970     1035377
         1980      Adana  1980     1485743
         1990      Adana  1990     1549233
         2000      Adana  2000     1849478
Bursa    1970      Bursa  1970      847884
         1980      Bursa  1980     1148492
         1990      Bursa  1990     1596161
         2000      Bursa  2000     2125140
```

Hierarchical Indexing (MultiIndex)

- Reseting index of a multi-index dataframe transforms it back to a single index dataframe.

```
>>> df = df.set_index(['city','year'])
>>> df
                population
city     year
İstanbul 1970      3019032
         1980      4741890
         1990      7195773
         2000     10018735
Ankara   1970      2041658
         1980      2854689
         1990      3236378
         2000      4007860
İzmir    1970      1427173
         1980      1976763
         1990      2694770
         2000      3370866
Konya    1970      1280239
         1980      1562139
         1990      1752658
         2000      2192166
Adana    1970      1035377
         1980      1485743
         1990      1549233
         2000      1849478
Bursa    1970       847884
         1980      1148492
         1990      1596161
         2000      2125140
```

```
>>> df.reset_index()
       city  year  population
0   İstanbul  1970     3019032
1   İstanbul  1980     4741890
2   İstanbul  1990     7195773
3   İstanbul  2000    10018735
4     Ankara  1970     2041658
5     Ankara  1980     2854689
6     Ankara  1990     3236378
7     Ankara  2000     4007860
8      İzmir  1970     1427173
9      İzmir  1980     1976763
10     İzmir  1990     2694770
11     İzmir  2000     3370866
12     Konya  1970     1280239
13     Konya  1980     1562139
14     Konya  1990     1752658
15     Konya  2000     2192166
16     Adana  1970     1035377
17     Adana  1980     1485743
18     Adana  1990     1549233
19     Adana  2000     1849478
20     Bursa  1970      847884
21     Bursa  1980     1148492
22     Bursa  1990     1596161
23     Bursa  2000     2125140
```

Stack

	city	year	population
0	İstanbul	1970	3019032
1	İstanbul	1980	4741890
2	İstanbul	1990	7195773
3	İstanbul	2000	10018735
4	Ankara	1970	2041658
5	Ankara	1980	2854689
6	Ankara	1990	3236378
7	Ankara	2000	4007860
8	İzmir	1970	1427173
9	İzmir	1980	1976763
10	İzmir	1990	2694770
11	İzmir	2000	3370866
12	Konya	1970	1280239
13	Konya	1980	1562139
14	Konya	1990	1752658
15	Konya	2000	2192166
16	Adana	1970	1035377
17	Adana	1980	1485743
18	Adana	1990	1549233
19	Adana	2000	1849478
20	Bursa	1970	847884
21	Bursa	1980	1148492
22	Bursa	1990	1596161
23	Bursa	2000	2125140

- «stack» function pivots a level of index labels, returning a DataFrame having a new level of column labels whose inner-most level consists of the pivoted index labels.

```
>>> df.stack()
0   city          İstanbul
    year              1970
    population     3019032
1   city          İstanbul
    year              1980
                       ...
22  year              1990
    population     1596161
23  city             Bursa
    year              2000
    population     2125140
Length: 72, dtype: object
```

Unstack

```
>>> df = df.set_index(['city','year'])
>>> df
              population
city     year
İstanbul 1970     3019032
         1980     4741890
         1990     7195773
         2000    10018735
Ankara   1970     2041658
         1980     2854689
         1990     3236378
         2000     4007860
İzmir    1970     1427173
         1980     1976763
         1990     2694770
         2000     3370866
Konya    1970     1280239
         1980     1562139
         1990     1752658
         2000     2192166
Adana    1970     1035377
         1980     1485743
         1990     1549233
         2000     1849478
Bursa    1970      847884
         1980     1148492
         1990     1596161
         2000     2125140
```

- «unstack» does the exact opposite of «stack» function.

```
>>> df.unstack()
         population
year           1970     1980     1990      2000
city
Adana       1035377  1485743  1549233   1849478
Ankara      2041658  2854689  3236378   4007860
Bursa        847884  1148492  1596161   2125140
Konya       1280239  1562139  1752658   2192166
İstanbul    3019032  4741890  7195773  10018735
İzmir       1427173  1976763  2694770   3370866
```

Dataframes

- Dataframes can be created by many ways:
  - from a Series object
  - from lists
  - from Numpy arrays
  - from dictionaries
  - from files
    - *read_table( )*
    - *read_csv( )*
    - *read_html( )*
    - *read_json( )*
    - *read_pickle( )*
    - *read_excel()*

DataFrame

- Column-wise arithmetic operations can be used for DataFrame columns
- Aggregate functions can also be used over DataFrame columns.

```
import pandas as pd
ogrenciler = {
'isim' :
['Ali','Ahmet','Mehmet','Veli','Arda','Doruk'],
'not'  : [5, 54, 12, 22, 15, 87],
'yas'  : [21, 24, 18, 28, 20, 22]}
a = pd.DataFrame(ogrenciler)
print(a['not']*1.5)
print(a['yas'].sum())
```

```
0     7.5
1    81.0
2    18.0
3    33.0
4    22.5
5   130.5
133
```

DataFrame

- If a column is created with no data, a NaN value will automatically be assigned, which in turn could be used to filter.

```
import pandas as pd
ogrenciler = {
'isim' : ['Ali','Ahmet','Mehmet',
'Veli','Arda','Doruk'],
'not'  : [5, 54, 12, 22, 15, 87],
'yas'  : [21, 24, 18, 28, 20, 22]}
a = pd.DataFrame(ogrenciler,
columns=['isim','not','yas','giris'])
print(a)
```

```
   isim  not  yas giris
0   Ali    5   21   NaN
1 Ahmet   54   24   NaN
2 Mehmet  12   18   NaN
3  Veli   22   28   NaN
4  Arda   15   20   NaN
5 Doruk   87   22   NaN
```

DataFrame

- DataFrames can be transposed like a matrix.

```
import pandas as pd
ogrenciler = {
'isim' : ['Ali','Ahmet','Mehmet','Veli','Arda','Doruk'],
'not'  : [5, 54, 12, 22, 15, 87],
'yas'  : [21, 24, 18, 28, 20, 22]}
a = pd.DataFrame(ogrenciler)
print(a)
print(a.T)
```

	isim	not	yas
0	Ali	5	21
1	Ahmet	54	24
2	Mehmet	12	18
3	Veli	22	28
4	Arda	15	20
5	Doruk	87	22

	0	1	2	3	4	5
isim	Ali	Ahmet	Mehmet	Veli	Arda	Doruk
not	5	54	12	22	15	87
yas	21	24	18	28	20	22

Example:

- Find the weighted average grade of each student (25% mid-term, 25% homework and 50% final)

isim	arasinav	odev	final
Ali	5	68	55
Ahmet	54	44	64
Mehmet	12	87	77
Veli	22	74	88
Arda	15	68	100
Doruk	87	70	90

Solution:

- Find the weighted average grade of each student (25% mid-term, 25% homework and 50% final)

```
import pandas as pd
ogrenciler = {
'isim' : ['Ali','Ahmet','Mehmet','Veli','Arda','Doruk'],
'arasinav'  : [5, 54, 12, 22, 15, 87],
'odev'  : [68, 44, 87, 74, 68, 70],
'final'  : [55, 64, 77, 88, 100, 90]}
a = pd.DataFrame(ogrenciler,columns=['isim','arasinav','odev','final','ortalama'])
a['ortalama'] = a['arasinav']*0.25+a['odev']*0.25+a['final']*0.50
print(a)
```

	isim	arasinav	odev	final	ortalama
0	Ali	5	68	55	45.75
1	Ahmet	54	44	64	56.50
2	Mehmet	12	87	77	63.25
3	Veli	22	74	88	68.00
4	Arda	15	68	100	70.75
5	Doruk	87	70	90	84.25

Analyzing Data (IMDB Top 1000)

- For exercise, top 1000 movie in the last 10 years will be analyzed.
- Data is from IMDB (International Movie Database) and in public domain (https://www.kaggle.com/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows)
- Many other versions of the movie data can be found on https://www.imdb.com/interfaces/
- Data is in .csv (comma separated values) format and can directly be imported into a dataframe through the pandas function read_csv()

Analyzing Data (IMDB Top 1000)

- Let's read the data;

```
import pandas as pd
# Read data from .csv file
data = pd.read_csv('IMDB-Movie-Data.csv')
```

- By default, a default index column is assigned:

```
>>> data
      Rank                    Title                     Genre  ...   Votes  Revenue (Millions)  Metascore
0        1  Guardians of the Galaxy   Action,Adventure,Sci-Fi  ...  757074              333.13       76.0
1        2               Prometheus  Adventure,Mystery,Sci-Fi  ...  485820              126.46       65.0
2        3                    Split           Horror,Thriller  ...  157606              138.12       62.0
3        4                     Sing   Animation,Comedy,Family  ...   60545              270.32       59.0
4        5             Suicide Squad  Action,Adventure,Fantasy  ...  393727              325.02       40.0
..     ...                      ...                       ...  ...     ...                 ...        ...
995    996     Secret in Their Eyes      Crime,Drama,Mystery  ...   27585                 NaN       45.0
996    997          Hostel: Part II                   Horror  ...   73152               17.54       46.0
997    998   Step Up 2: The Streets      Drama,Music,Romance  ...   70699               58.01       50.0
998    999             Search Party         Adventure,Comedy  ...    4881                 NaN       22.0
999   1000              Nine Lives    Comedy,Family,Fantasy  ...   12435               19.64       11.0
```

Display Settings

- The display is truncated since it is not always possible to show all the rows and columns.
- This can be set using pandas display options

```
>>> data
                        Rank                    Genre  ... Revenue (Millions) Metascore
Title                                                  ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  ...             333.13       76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  ...             126.46       65.0
Split                      3           Horror,Thriller  ...             138.12       62.0
Sing                       4   Animation,Comedy,Family  ...             270.32       59.0
Suicide Squad              5  Action,Adventure,Fantasy  ...             325.02       40.0
...                      ...                       ...  ...                ...        ...
Secret in Their Eyes     996       Crime,Drama,Mystery  ...                NaN       45.0
Hostel: Part II          997                    Horror  ...              17.54       46.0
Step Up 2: The Streets   998       Drama,Music,Romance  ...              58.01       50.0
Search Party             999          Adventure,Comedy  ...                NaN       22.0
Nine Lives              1000     Comedy,Family,Fantasy  ...              19.64       11.0
```

Display Settings

- The most common display options are:
  - pd.options.display.max_rows
  - pd.options.display.min_rows
  - pd.options.display.max_columns
  - pd.options.display.min_columns
  - pd.options.display.max_colwidth
  - pd.options.display.precision
  - pd.options.display.float_format
  - pd.options.display.plotting.backend
  - pd.display.max_info_columns
  - pd.display.max_info_rows
  - pd.describe_option() (a general helper for available options)
- The current values can be retrieved by *pd.get_option()*
- The options can be set by *pd.set_option()*

Display Settings

Some rules for display of rows/columns:

- To display all the rows, set max_rows greater than the number of rows in the DataFrame.
- To display more than 10 rows when the dataframe is truncated, set min_rows greater than 10.
- With more than 200 rows of data, if max_rows is 200 and min_rows is 20, 10 from the head and 10 from the tail will be displayed.
- With more than 200 rows of data, if max_rows is 200 and min_rows is None, 100 from the head and 100 from the tail will be displayed.

Display Settings

```
>>> data
                        Rank                 Genre                                        Description         Director  ... Rating   Votes  Revenue (Millions)  Metascore
Title                                                                                                              ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  A group of intergalactic criminals are forced ...     James Gunn  ...    8.1  757074              333.13       76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  Following clues to the origin of mankind, a te...    Ridley Scott  ...    7.0  485820              126.46       65.0
Split                      3          Horror,Thriller  Three girls are kidnapped by a man with a diag...  M. Night Shyamalan  ...    7.3  157606              138.12       62.0
Sing                       4   Animation,Comedy,Family  In a city of humanoid animals, a hustling thea...  Christophe Lourdelet  ...    7.2   60545              270.32       59.0
Suicide Squad              5  Action,Adventure,Fantasy  A secret government agency recruits some of th...     David Ayer  ...    6.2  393727              325.02       40.0
...                      ...                      ...                                                ...           ...  ...    ...     ...                 ...        ...
Secret in Their Eyes     996      Crime,Drama,Mystery  A tight-knit team of rising investigators, alo...     Billy Ray  ...    6.2   27585                NaN       45.0
Hostel: Part II          997                   Horror  Three American college students studying abroa...      Eli Roth  ...    5.5   73152               17.54       46.0
Step Up 2: The Streets   998       Drama,Music,Romance  Romantic sparks occur between two dance studen...      Jon M. Chu  ...    6.2   70699               58.01       50.0
Search Party             999          Adventure,Comedy  A pair of friends embark on a mission to reuni...  Scot Armstrong  ...    5.6    4881                NaN       22.0
Nine Lives              1000     Comedy,Family,Fantasy  A stuffy businessman finds himself trapped ins...  Barry Sonnenfeld  ...    5.3   12435               19.64       11.0
```

pd.set_option('display.min_rows',10)

```
>>> pd.set_option('display.min_rows',20)
>>> data
                        Rank                 Genre                                        Description         Director  ... Rating   Votes  Revenue (Millions)  Metascore
Title                                                                                                              ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  A group of intergalactic criminals are forced ...     James Gunn  ...    8.1  757074              333.13       76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  Following clues to the origin of mankind, a te...    Ridley Scott  ...    7.0  485820              126.46       65.0
Split                      3          Horror,Thriller  Three girls are kidnapped by a man with a diag...  M. Night Shyamalan  ...    7.3  157606              138.12       62.0
Sing                       4   Animation,Comedy,Family  In a city of humanoid animals, a hustling thea...  Christophe Lourdelet  ...    7.2   60545              270.32       59.0
Suicide Squad              5  Action,Adventure,Fantasy  A secret government agency recruits some of th...     David Ayer  ...    6.2  393727              325.02       40.0
The Great Wall             6  Action,Adventure,Fantasy  European mercenaries searching for black powde...     Yimou Zhang  ...    6.1   56036               45.13       42.0
La La Land                 7        Comedy,Drama,Music  A jazz pianist falls for an aspiring actress i...  Damien Chazelle  ...    8.3  258682              151.06       93.0
Mindhorn                   8                    Comedy  A has-been actor best known for playing the ti...     Sean Foley  ...    6.4    2490                NaN       71.0
The Lost City of Z         9  Action,Adventure,Biography  A true-life drama, centering on British explor...     James Gray  ...    7.1    7188                8.01       78.0
Passengers                10   Adventure,Drama,Romance  A spacecraft traveling to a distant colony pla...    Morten Tyldum  ...    7.0  192177              100.01       41.0
...                      ...                      ...                                                ...           ...  ...    ...     ...                 ...        ...
Underworld: Rise of the Lycans  991  Action,Adventure,Fantasy  An origins story centered on the centuries-old...  Patrick Tatopoulos  ...    6.6  129708               45.80       44.0
Taare Zameen Par          992        Drama,Family,Music  An eight-year-old boy is thought to be a lazy ...      Aamir Khan  ...    8.5  102697                1.20       42.0
Take Me Home Tonight      993       Comedy,Drama,Romance  Four years after graduation, an awkward high s...     Michael Dowse  ...    6.3   45419                6.92        NaN
Resident Evil: Afterlife  994  Action,Adventure,Horror  While still out to destroy the evil Umbrella C...  Paul W.S. Anderson  ...    5.9  140900               60.13       37.0
Project X                 995                    Comedy  3 high school seniors throw a birthday party t...  Nima Nourizadeh  ...    6.7  164088               54.72       48.0
Secret in Their Eyes      996      Crime,Drama,Mystery  A tight-knit team of rising investigators, alo...     Billy Ray  ...    6.2   27585                NaN       45.0
Hostel: Part II           997                   Horror  Three American college students studying abroa...      Eli Roth  ...    5.5   73152               17.54       46.0
Step Up 2: The Streets    998       Drama,Music,Romance  Romantic sparks occur between two dance studen...      Jon M. Chu  ...    6.2   70699               58.01       50.0
Search Party              999          Adventure,Comedy  A pair of friends embark on a mission to reuni...  Scot Armstrong  ...    5.6    4881                NaN       22.0
Nine Lives               1000     Comedy,Family,Fantasy  A stuffy businessman finds himself trapped ins...  Barry Sonnenfeld  ...    5.3   12435               19.64       11.0
```

Analyzing Data (IMDB Top 1000)

- Alternatively, we can assign one of the columns as index column:

```
data = pd.read_csv('IMDB-Movie-Data.csv', index_col="Title")
```

```
>>> data
                        Rank                      Genre  ... Revenue (Millions) Metascore
Title                                                    ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  ...              333.13      76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  ...              126.46      65.0
Split                      3           Horror,Thriller  ...              138.12      62.0
Sing                       4   Animation,Comedy,Family  ...              270.32      59.0
Suicide Squad              5  Action,Adventure,Fantasy  ...              325.02      40.0
...                      ...                       ...  ...                 ...       ...
Secret in Their Eyes     996       Crime,Drama,Mystery  ...                 NaN      45.0
Hostel: Part II          997                    Horror  ...               17.54      46.0
Step Up 2: The Streets   998       Drama,Music,Romance  ...               58.01      50.0
Search Party             999          Adventure,Comedy  ...                 NaN      22.0
Nine Lives              1000     Comedy,Family,Fantasy  ...               19.64      11.0
```

Analyzing Data (IMDB Top 1000)

- All the columns (fields) will be shown along with the index column:

```
>>> data.Rank
Title
Guardians of the Galaxy       1
Prometheus                    2
Split                         3
Sing                          4
Suicide Squad                 5
                            ...
Secret in Their Eyes        996
Hostel: Part II             997
Step Up 2: The Streets      998
Search Party                999
Nine Lives                 1000
Name: Rank, Length: 1000, dtype: int64
```

```
>>> data.Genre
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi
Prometheus                Adventure,Mystery,Sci-Fi
Split                              Horror,Thriller
Sing                       Animation,Comedy,Family
Suicide Squad             Action,Adventure,Fantasy
                                               ...
Secret in Their Eyes            Crime,Drama,Mystery
Hostel: Part II                                Horror
Step Up 2: The Streets          Drama,Music,Romance
Search Party                       Adventure,Comedy
Nine Lives                   Comedy,Family,Fantasy
Name: Genre, Length: 1000, dtype: object
```

Info function

- The info function can be used on a data frame to provide information about the data frame usually relating to performance more than statistics.
- This is useful if you want to check your memory allocations or the data types of each series inside of a data frame.
- Using .info() method many useful information can be retrieved:

```
>>> data.info()
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Rank                1000 non-null   int64
 1   Genre               1000 non-null   object
 2   Description         1000 non-null   object
 3   Director            1000 non-null   object
 4   Actors              1000 non-null   object
 5   Year                1000 non-null   int64
 6   Runtime (Minutes)   1000 non-null   int64
 7   Rating              1000 non-null   float64
 8   Votes               1000 non-null   int64
 9   Revenue (Millions)  872 non-null    float64
 10  Metascore           936 non-null    float64
dtypes: float64(3), int64(4), object(4)
memory usage: 126.0+ KB
```

```
>>> data.dtypes
Rank                    int64
Genre                  object
Description            object
Director               object
Actors                 object
Year                    int64
Runtime (Minutes)       int64
Rating                float64
Votes                   int64
Revenue (Millions)    float64
Metascore             float64
dtype: object
```

Describe function

- One of the most underrated features in Pandas is a simple function called describe().
- Using the describe function on a data frame yields a very statistical result that will tell you all that you need to know about each column's values independently.
- Using this function is a great way to get all of the following statistics for each column incredibly fast:
  - mean
  - count
  - standard deviation
  - 1st quartile
  - 2nd quartile (median)
  - 3rd quartile
  - minimum value
  - maximum value

Describe function

- Using .describe() method a summary descriptive statistics information can be retrieved:

```
>>> data.describe()
              Rank         Year  Runtime (Minutes)       Rating         Votes  Revenue (Millions)   Metascore
count  1000.000000  1000.000000        1000.000000  1000.000000  1.000000e+03          872.000000  936.000000
mean    500.500000  2012.783000         113.172000     6.723200  1.698083e+05           82.956376   58.985043
std     288.819436     3.205962          18.810908     0.945429  1.887626e+05          103.253540   17.194757
min       1.000000  2006.000000          66.000000     1.900000  6.100000e+01            0.000000   11.000000
25%     250.750000  2010.000000         100.000000     6.200000  3.630900e+04           13.270000   47.000000
50%     500.500000  2014.000000         111.000000     6.800000  1.107990e+05           47.985000   59.500000
75%     750.250000  2016.000000         123.000000     7.400000  2.399098e+05          113.715000   72.000000
max    1000.000000  2016.000000         191.000000     9.000000  1.791916e+06          936.630000  100.000000
```

- Another useful function is memory_usage to show the amount of memory in bytes allocated for each column

```
>>> df.memory_usage()
Index                128
Rank                8000
Genre               8000
Description         8000
Director            8000
Actors              8000
Year                8000
Runtime (Minutes)   8000
Rating              8000
Votes               8000
Revenue (Millions)  8000
Metascore           8000
dtype: int64
```

DataFrame Properties

- Dataset consists of 1000 records and 11 columns

```
>>> data.shape
(1000, 11)
```

- Alternatively, the number of columns can be found by:

```
>>> len(data.columns)
11
```

- Total number of elements:

```
>>> data.size
11000
```

- Columns can be shown as:

```
>>> data.columns
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore'],
      dtype='object')
```

Head & Tail Functions

- We can use head(n) or tail(n) functions to show the first and the last n records, respectively:

```
>>> data.head(5)
                        Rank                      Genre  ... Revenue (Millions) Metascore
Title                                                    ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  ...              333.13      76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  ...              126.46      65.0
Split                      3           Horror,Thriller  ...              138.12      62.0
Sing                       4   Animation,Comedy,Family  ...              270.32      59.0
Suicide Squad              5  Action,Adventure,Fantasy  ...              325.02      40.0

[5 rows x 11 columns]
```

```
>>> data.tail(5)
                        Rank                     Genre  ... Revenue (Millions) Metascore
Title                                                   ...
Secret in Their Eyes     996   Crime,Drama,Mystery  ...                NaN      45.0
Hostel: Part II          997                Horror  ...              17.54      46.0
Step Up 2: The Streets   998   Drama,Music,Romance  ...              58.01      50.0
Search Party             999      Adventure,Comedy  ...                NaN      22.0
Nine Lives              1000  Comedy,Family,Fantasy  ...              19.64      11.0

[5 rows x 11 columns]
```

Resampling

- For large datasets, a smaller subset can be selected.
- Sample(*n*) function does the sampling randomly. The parameter *n* can be given to specify the number of records.

```
>>> data.shape
(1000, 11)
>>> data.sample(250).shape
(250, 11)
>>> data.shape
(1000, 11)
```

```
>>> data.sample(5)
                     Rank                    Genre  ... Revenue (Millions) Metascore
Title                                               ...
Southpaw              382              Drama,Sport  ...              52.42      57.0
How to Be Single      310         Comedy,Romance  ...              46.81      51.0
Collide               376  Action,Crime,Thriller  ...               2.20      33.0
The Rise of the Krays 605              Crime,Drama  ...               6.53      90.0
The Maze Runner       372  Action,Mystery,Sci-Fi  ...             102.41      57.0

[5 rows x 11 columns]
```

Reading Data Files

- read_csv()
  - Can be used to read a lot of structured text files
  - Originally for csv files but any separator can be customly defined
  - A lot options are available.

```
pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None,
header='infer', names=NoDefault.no_default, index_col=None, usecols=None, squeeze=False,
prefix=NoDefault.no_default, mangle_dupe_cols=True, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skipinitialspace=False,
skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True,
na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None, storage_options=None)      [source]
```

Reading Data Files

- read_json()
  - JSON (Javascript Object Notation) is a very common Exchange format similar to Python dictionaries.
  - JSON files can be read from file as well as urls.
  - If the JSON file is in compressed form (.gz, .bz2, .zip, .xz), a proper decompression is applied (compression='infer' option)

```
df = pd.read_json('sample.json')
df = pd.read_json('sample.zip', compression='infer')
```

```
{
  "Name": {
    "0": "Axel",
    "1": "Alice",
    "2": "Alex"
  },
  "Age": {
    "0": 32,
    "1": 26,
    "2": 45
  }
}
```

JSON Format

Reading Data Files

- read_html()
  - HTML tables can be directly imported as a Dataframe.
  - Various options are available.
  - **pandas.read_html(*io, match='.+', flavor=None, header=None, index_col=None, skiprows=None, attrs=None, parse_dates=False, thousands=',', encoding=None, decimal='.', converters=None, na_values=None, keep_default_na=True, displayed_only=True*)**

```
html_string = """
<table>
  <thead>
    <tr>
      <th>date</th>
      <th>name</th>
      <th>year</th>
      <th>cost</th>
      <th>region</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>2020-01-01</td>
      <td>Jenny</td>
      <td>1998</td>
      <td>0.2</td>
      <td>South</td>
    </tr>
    <tr>
      <td>2020-01-02</td>
      <td>Alice</td>
      <td>1992</td>
      <td>-1.34</td>
      <td>East</td>
    </tr>
    <tr>
      <td>2020-01-03</td>
      <td>Tomas</td>
      <td>1982</td>
      <td>1.00023</td>
      <td>South</td>
    </tr>
  </tbody>
</table>
"""
```

date	name	year	cost	region
2020-01-01	Jenny	1998	0.2	South
2020-01-02	Alice	1992	-1.34	East
2020-01-03	Tomas	1982	1.00023	South

Accessing Elements:

- The elements of pandas Dataframe can be accessed either through
  - Named Row/Column indices (loc)
  - Integer Row/Column indices (iloc)
  - Mixed Row/Column indices (ix) (Deprecated)
- Slicing is also possible through
  - Named Row/Column indices (loc)
  - Integer Row/Column indices (iloc)
  - Mixed Row/Column indices (ix) (Deprecated)

Accessing Data

- Assigning the movie titles as index column allows access the records via their titles:

```
>>> data.loc['Prometheus',:]
Rank                                                        2
Genre                                     Adventure,Mystery,Sci-Fi
Description          Following clues to the origin of mankind, a te...
Director                                              Ridley Scott
Actors               Noomi Rapace, Logan Marshall-Green, Michael Fa...
Year                                                       2012
Runtime (Minutes)                                           124
Rating                                                      7.0
Votes                                                    485820
Revenue (Millions)                                       126.46
Metascore                                                 65.0
Name: Prometheus, dtype: object
```

```
>>> data.loc['Prometheus','Director']
'Ridley Scott'
```

Accessing Data

```
>>> data.head()
                     Rank                    Genre  ... Revenue (Millions) Metascore
Title                                               ...
Guardians of the Galaxy     1   Action,Adventure,Sci-Fi  ...             333.13      76.0
Prometheus                  2  Adventure,Mystery,Sci-Fi  ...             126.46      65.0
Split                       3           Horror,Thriller  ...             138.12      62.0
Sing                        4   Animation,Comedy,Family  ...             270.32      59.0
Suicide Squad               5  Action,Adventure,Fantasy  ...             325.02      40.0
```

- Slicing with named indices:

```
>>> data.loc['Prometheus':'Sing',:]
            Rank                    Genre  ... Revenue (Millions) Metascore
Title                                      ...
Prometheus     2  Adventure,Mystery,Sci-Fi  ...             126.46      65.0
Split          3           Horror,Thriller  ...             138.12      62.0
Sing           4   Animation,Comedy,Family  ...             270.32      59.0

[3 rows x 11 columns]
```

Accessing Data

```
>>> data.columns
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore'],
      dtype='object')
```

- Slicing is also possible with named columns:

```
>>> data.loc['Prometheus':'Sing','Director':'Year']
                        Director                                            Actors  Year
Title
Prometheus           Ridley Scott  Noomi Rapace, Logan Marshall-Green, Michael Fa...  2012
Split         M. Night Shyamalan  James McAvoy, Anya Taylor-Joy, Haley Lu Richar...  2016
Sing        Christophe Lourdelet  Matthew McConaughey,Reese Witherspoon, Seth Ma...  2016
```

Accessing Data

```
>>> data
                        Rank                    Genre  ... Revenue (Millions) Metascore
Title                                                  ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  ...              333.13      76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  ...              126.46      65.0
Sing                       4   Animation,Comedy,Family  ...              270.32      59.0
Suicide Squad              5  Action,Adventure,Fantasy  ...              325.02      40.0
...                      ...                       ...  ...                 ...       ...
Secret in Their Eyes     996       Crime,Drama,Mystery  ...                 NaN      45.0
Hostel: Part II          997                    Horror  ...               17.54      46.0
Step Up 2: The Streets   998       Drama,Music,Romance  ...               58.01      50.0
Search Party             999          Adventure,Comedy  ...                 NaN      22.0
Nine Lives              1000     Comedy,Family,Fantasy  ...               19.64      11.0

[1000 rows x 11 columns]
>>> data.iloc[0,1]
'Action,Adventure,Sci-Fi'
>>> data.iloc[0,:]
Rank                                                        1
Genre                                    Action,Adventure,Sci-Fi
Description          A group of intergalactic criminals are forced ...
Director                                             James Gunn
Actors               Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Year                                                      2014
Runtime (Minutes)                                          121
Rating                                                     8.1
Votes                                                   757074
Revenue (Millions)                                      333.13
Metascore                                                76.0
Name: Guardians of the Galaxy, dtype: object
```

Accessing Data

- Slicing with named indices:

```
>>> data.iloc[0:5,:]
                        Rank                      Genre  ... Revenue (Millions) Metascore
Title                                                    ...
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  ...              333.13      76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  ...              126.46      65.0
Split                      3           Horror,Thriller  ...              138.12      62.0
Sing                       4   Animation,Comedy,Family  ...              270.32      59.0
Suicide Squad              5  Action,Adventure,Fantasy  ...              325.02      40.0

[5 rows x 11 columns]
```

- Slicing is also possible with named columns:

```
>>> data.iloc[0:5,2:4]
                                                  Description          Director
Title
Guardians of the Galaxy  A group of intergalactic criminals are forced ...      James Gunn
Prometheus               Following clues to the origin of mankind, a te...      Ridley Scott
Split                    Three girls are kidnapped by a man with a diag...  M. Night Shyamalan
Sing                     In a city of humanoid animals, a hustling thea...  Christophe Lourdelet
Suicide Squad            A secret government agency recruits some of th...      David Ayer
```

Iterating over rows

- There are several ways to iterate over the rows of a dataframe:
  - Using **index** attribute of the Dataframe

```
for ind in df.index:
    print(df['column1'][ind], df['column2'][ind])
```

  - Using **loc[]** function of the Dataframe

```
for i in range(len(df)):
    print(df.loc[i,'column1'], df.loc[i,'column2'])
```

  - Using **iloc[]** function of the DataFrame

```
for i in range(len(df)):
    print(df.iloc[i,0], df.iloc[i,1])
```

  - Using **iterrows()** method of the Dataframe

```
for i,row in df.iterrows():
    print(row['column1'], row['column2'])
```

  - Using **itertuples()** method of the Dataframe

```
for row in df.itertuples(index = True, name ='Pandas'):
    print (getattr(row, 'column1'), getattr(row, 'column2'))
```

Filtering/Querying Data

- Data can be filtered and queried using Boolean indices.

```
>>> data[data['Rating'] > 8.5]
                  Rank                      Genre  ... Revenue (Millions) Metascore
Title                                              ...
Interstellar        37  Adventure,Drama,Sci-Fi  ...              187.99      74.0
The Dark Knight     55      Action,Crime,Drama  ...              533.32      82.0
Inception           81  Action,Adventure,Sci-Fi  ...              292.57      74.0
Kimi no na wa       97  Animation,Drama,Fantasy  ...                4.68      79.0
Dangal             118   Action,Biography,Drama  ...               11.15       NaN
The Intouchables   250   Biography,Comedy,Drama  ...               13.18      57.0

[6 rows x 11 columns]
```

- Logical Operators «&» for «and» «|» for «or» can be used.

```
>>> data[(data['Rating'] > 7.0) & (data['Director'] == 'Ridley Scott')]
                  Rank                      Genre  ... Revenue (Millions) Metascore
Title                                              ...
The Martian        103  Adventure,Drama,Sci-Fi  ...              228.43      80.0
American Gangster  471   Biography,Crime,Drama  ...              130.13      76.0
Body of Lies       738    Action,Drama,Romance  ...               39.38      57.0
```

Filtering/Querying Data

- df[df.Director.isin(['Steven Spielberg','Ridley Scott'])]

```
>>> df[df.Director.isin(['Steven Spielberg','Ridley Scott'])]
                                                  Rank                    Genre  ... Revenue (Millions) Metascore
Title                                                                            ...
Prometheus                                           2  Adventure,Mystery,Sci-Fi  ...             126.46      65.0
The Martian                                        103    Adventure,Drama,Sci-Fi  ...             228.43      80.0
The BFG                                            151  Adventure,Family,Fantasy  ...              55.47      66.0
Lincoln                                            334   Biography,Drama,History  ...             182.20      86.0
Bridge of Spies                                    384    Drama,History,Thriller  ...              72.31      81.0
Robin Hood                                         388    Action,Adventure,Drama  ...             105.22      53.0
American Gangster                                  471     Biography,Crime,Drama  ...             130.13      76.0
Exodus: Gods and Kings                             517    Action,Adventure,Drama  ...              65.01      52.0
The Counselor                                      522      Crime,Drama,Thriller  ...              16.97      48.0
A Good Year                                        531      Comedy,Drama,Romance  ...               7.46      47.0
Body of Lies                                       738      Action,Drama,Romance  ...              39.38      57.0
Indiana Jones and the Kingdom of the Crystal Skull 768  Action,Adventure,Fantasy  ...             317.01      65.0

[12 rows x 11 columns]
```

Filtering/Querying Data

```
>>> df[df.Director.isin(['Steven Spielberg','Ridley Scott']) & (df.Rating > 7.5)]
                  Rank                    Genre  ... Revenue (Millions) Metascore
Title                                            ...
The Martian        103  Adventure,Drama,Sci-Fi  ...             228.43      80.0
Bridge of Spies    384  Drama,History,Thriller  ...              72.31      81.0
American Gangster  471   Biography,Crime,Drama  ...             130.13      76.0
```

```
>>> df[df.Director.isin(['Steven Spielberg','Ridley Scott']) | (df.Rating > 9.0)]
                                                 Rank                    Genre  ... Revenue (Millions) Metascore
Title                                                                           ...
Prometheus                                          2  Adventure,Mystery,Sci-Fi  ...             126.46      65.0
The Martian                                       103    Adventure,Drama,Sci-Fi  ...             228.43      80.0
The BFG                                           151  Adventure,Family,Fantasy  ...              55.47      66.0
Lincoln                                           334   Biography,Drama,History  ...             182.20      86.0
Bridge of Spies                                   384    Drama,History,Thriller  ...              72.31      81.0
Robin Hood                                        388    Action,Adventure,Drama  ...             105.22      53.0
American Gangster                                 471     Biography,Crime,Drama  ...             130.13      76.0
Exodus: Gods and Kings                            517    Action,Adventure,Drama  ...              65.01      52.0
The Counselor                                     522      Crime,Drama,Thriller  ...              16.97      48.0
A Good Year                                       531      Comedy,Drama,Romance  ...               7.46      47.0
Body of Lies                                      738      Action,Drama,Romance  ...              39.38      57.0
Indiana Jones and the Kingdom of the Crystal Skull  768  Action,Adventure,Fantasy  ...             317.01      65.0
```

Filtering/Filling Data (where)

- Another similar command for filtering and querying data is «where».
- DataFrame.where(cond, other=nan, inplace=False, axis=None, level=None, errors='raise', try_cast=False, raise_on_error=None)
- «where» method provides a flexible way of selecting and filling data. In particular, problematic data can be fixed using the «where» function.
- There are several options to use with «where»:
  1. where with *DataFrame* condition
  2. where with *Series* condition
  3. where with *Callable* condition
  4. where with *Callable* other
  5. where with *DataFrame* other

Filtering/Filling Data (where)

- All the elements of a dataframe which don't satisfy the condition will be assigned «NaN»

```
>>> df.where(df.Director == 'Steven Spielberg')
```

Title	Rank	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
Sing	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Moana	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
The Secret Life of Pets	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Trolls	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sausage Party	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Zootopia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- Arbitrary values can be assigned.

```
>>> df.where(df.Rating > 8.0, other=0.0).sort_values('Rating',ascending=False)
```

Title	Rank	Genre	Description	Director	...	Rating
Kimi no na wa	97	Animation,Drama,Fantasy	Two strangers find themselves linked in a biza...	Makoto Shinkai	...	8.6
Koe no katachi	862	Animation,Drama,Romance	The story revolves around Nishimiya Shoko, a g...	Naoko Yamada	...	8.4
WALL·E	635	Animation,Adventure,Family	In the distant future, a small waste-collectin...	Andrew Stanton	...	8.4
Up	500	Animation,Adventure,Comedy	Seventy-eight year old Carl Fredricksen travel...	Pete Docter	...	8.3
Toy Story 3	689	Animation,Adventure,Comedy	The toys are mistakenly delivered to a day-car...	Lee Unkrich	...	8.3
Inside Out	242	Animation,Adventure,Comedy	After young Riley is uprooted from her Midwest...	Pete Docter	...	8.2
Zootopia	75	Animation,Adventure,Comedy	In a city of anthropomorphic animals, a rookie...	Byron Howard	...	8.1
How to Train Your Dragon	773	Animation,Action,Adventure	A hapless young Viking who aspires to hunt dra...	Dean DeBlois	...	8.1
Cars 2	0	0.0	0.0	0.0	...	0.0
Brave	0	0.0	0.0	0.0	...	0.0
Megamind	0	0.0	0.0	0.0	...	0.0
Kung Fu Panda 3	0	0.0	0.0	0.0	...	0.0

Query

- The query function allows for applying filtering conditions as a string. It provides more flexibility than many other techniques.

```
>>> data.query('Rating > 8')
```

Title	Rank	Genre	...	Revenue (Millions)	Metascore
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	...	333.13	76.0
La La Land	7	Comedy,Drama,Music	...	151.06	93.0
Hacksaw Ridge	17	Biography,Drama,History	...	67.12	71.0
Lion	19	Biography,Drama	...	51.69	69.0
Bahubali: The Beginning	27	Action,Adventure,Drama	...	6.50	NaN
Interstellar	37	Adventure,Drama,Sci-Fi	...	187.99	74.0
Star Wars: Episode VII - The Force Awakens	51	Action,Adventure,Fantasy	...	936.63	81.0
The Dark Knight	55	Action,Crime,Drama	...	533.32	82.0
The Prestige	65	Drama,Mystery,Sci-Fi	...	53.08	66.0
Mad Max: Fury Road	68	Action,Adventure,Sci-Fi	...	153.63	90.0
Zootopia	75	Animation,Adventure,Comedy	...	341.26	78.0
The Avengers	77	Action,Sci-Fi	...	623.28	69.0
Inglourious Basterds	78	Adventure,Drama,War	...	120.52	69.0
Inception	81	Action,Adventure,Sci-Fi	...	292.57	74.0
The Wolf of Wall Street	83	Biography,Comedy,Crime	...	116.87	75.0
Gone Girl	84	Crime,Drama,Mystery	...	167.74	79.0
Prisoners	91	Crime,Drama,Mystery	...	60.96	74.0
The Help	93	Drama	...	169.71	62.0
Kimi no na wa	97	Animation,Drama,Fantasy	...	4.68	79.0

Query

- The query function allows quick filtering of data

```
>>> data.query("Year > 2015 and Rating > 8")
                Rank                       Genre                                        Description  ...   Votes Revenue (Millions)  Metascore
Title                                                                                                ...
La La Land         7                Comedy,Drama,Music  A jazz pianist falls for an aspiring actress i...  ...  258682              151.06       93.0
Hacksaw Ridge     17           Biography,Drama,History  WWII American Army Medic Desmond T. Doss, who ...  ...  211760               67.12       71.0
Lion              19                   Biography,Drama  A five-year-old Indian boy gets lost on the st...  ...  102061               51.69       69.0
Zootopia          75  Animation,Adventure,Comedy        In a city of anthropomorphic animals, a rookie...  ...  296853              341.26       78.0
Kimi no na wa     97       Animation,Drama,Fantasy      Two strangers find themselves linked in a biza...  ...   34110                4.68       79.0
Dangal           118        Action,Biography,Drama      Former wrestler Mahavir Singh Phogat and his t...  ...   48969               11.15        NaN
Ah-ga-ssi        146         Drama,Mystery,Romance      A woman is hired as a handmaiden to a Japanese...  ...   33418                2.01       84.0
Paint It Black   479                         Drama      A young woman attempts to deal with the death ...  ...      61                 NaN       71.0
Koe no katachi   862       Animation,Drama,Romance      The story revolves around Nishimiya Shoko, a g...  ...    2421                 NaN       80.0

[9 rows x 11 columns]
```

- Some arithmetic conditions can also be applied.

```
>>> data.query("Metascore/Votes > 0.5")
                        Rank        Genre                                        Description  ... Votes Revenue (Millions)  Metascore
Title                                                                                        ...
Tracktown                338  Drama,Sport  A young, talented, and lonely long-distance ru...  ...   115                 NaN       64.0
The Headhunter's Calling 417        Drama  A headhunter whose life revolves around closin...  ...   164                 NaN       85.0
Paint It Black           479        Drama  A young woman attempts to deal with the death ...  ...    61                 NaN       71.0

[3 rows x 11 columns]
```

Query

- Many complex queries can be formed.

```
>>> data.query('Rating > 7 and Director == "Ridley Scott"')
                  Rank                    Genre  ... Revenue (Millions) Metascore
Title                                            ...
The Martian        103  Adventure,Drama,Sci-Fi  ...              228.43      80.0
American Gangster  471   Biography,Crime,Drama  ...              130.13      76.0
Body of Lies       738    Action,Drama,Romance  ...               39.38      57.0

[3 rows x 11 columns]
```

- The "not" operator can also be implemented as part of the filter in the query function.

```
>>> data.query("not (Rating < 9 and Votes < 10000000)")
             Rank                Genre                                Description  ...    Votes Revenue (Millions)  Metascore
Title                                                                             ...
The Dark Knight    55  Action,Crime,Drama  When the menace known as the Joker wreaks havo...  ...  1791916           533.32       82.0

[1 rows x 11 columns]
```

Apply

- Many complex operations over rows/columns can be applied by using the «apply» function.
- The apply function can be a standard Python function or an anonymous (lambda) function.
- Apply function can be used for both rows and columns, which is selected by the use of «axis» keyword parameter.
- Beside rows/columns, a function can be applied to all elements of the dataframe (applymap).
- General usage:
  - `def apply( self, func, axis=0, broadcast=None, raw=False, reduce=None, result_type=None, args=(), **kwds )`

Apply

- Supppose we want to add a new column which shows the short or long depending on the number of words in the description.

```
def countWords(row):
    nWords = len(row['Description'].split())
    if nWords > 20:
        return 'Long Description'
    else:
        return 'Short Description'


df['Description Type'] = df.apply(countWords, axis=1)
```

```
>>> df.iloc[:20,[0,2,3,-1]]
                                  Rank                                        Description            Director Description Type
Title
Guardians of the Galaxy              1  A group of intergalactic criminals are forced ...        James Gunn  Long Description
Prometheus                           2  Following clues to the origin of mankind, a te...       Ridley Scott  Long Description
Split                                3  Three girls are kidnapped by a man with a diag...  M. Night Shyamalan  Long Description
Sing                                 4  In a city of humanoid animals, a hustling thea...  Christophe Lourdelet  Long Description
Suicide Squad                        5  A secret government agency recruits some of th...        David Ayer  Long Description
The Great Wall                       6  European mercenaries searching for black powde...        Yimou Zhang  Long Description
La La Land                           7  A jazz pianist falls for an aspiring actress i...     Damien Chazelle  Short Description
Mindhorn                             8  A has-been actor best known for playing the ti...        Sean Foley  Long Description
The Lost City of Z                   9  A true-life drama, centering on British explor...        James Gray  Long Description
Passengers                          10  A spacecraft traveling to a distant colony pla...      Morten Tyldum  Long Description
Fantastic Beasts and Where to Find Them  11  The adventures of writer Newt Scamander in New...        David Yates  Long Description
Hidden Figures                      12  The story of a team of female African-American...     Theodore Melfi  Long Description
Rogue One                           13  The Rebel Alliance makes a risky move to steal...      Gareth Edwards  Long Description
Moana                               14  In Ancient Polynesia, when a terrible curse in...        Ron Clements  Long Description
Colossal                            15  Gloria is an out-of-work party girl forced to ...     Nacho Vigalondo  Long Description
The Secret Life of Pets             16  The quiet life of a terrier named Max is upend...        Chris Renaud  Long Description
Hacksaw Ridge                       17  WWII American Army Medic Desmond T. Doss, who ...        Mel Gibson  Long Description
Jason Bourne                        18  The CIA's most dangerous former operative is d...      Paul Greengrass  Short Description
Lion                                19  A five-year-old Indian boy gets lost on the st...        Garth Davis  Long Description
Arrival                             20  When twelve mysterious spacecraft appear aroun...     Denis Villeneuve  Long Description
```

Applymap

- Let's apply a function to all the elements.
- Suppose we want to convert all the text to the upper case.

```
def upperCase(x):
    return x.upper()

df1 = df.iloc[:,[1,2,3]]
df1 = df1.applymap(upperCase)
```

Title	Genre	Description	Director
Guardians of the Galaxy	ACTION,ADVENTURE,SCI-FI	A GROUP OF INTERGALACTIC CRIMINALS ARE FORCED ...	JAMES GUNN
Prometheus	ADVENTURE,MYSTERY,SCI-FI	FOLLOWING CLUES TO THE ORIGIN OF MANKIND, A TE...	RIDLEY SCOTT
Split	HORROR,THRILLER	THREE GIRLS ARE KIDNAPPED BY A MAN WITH A DIAG...	M. NIGHT SHYAMALAN
Sing	ANIMATION,COMEDY,FAMILY	IN A CITY OF HUMANOID ANIMALS, A HUSTLING THEA...	CHRISTOPHE LOURDELET
Suicide Squad	ACTION,ADVENTURE,FANTASY	A SECRET GOVERNMENT AGENCY RECRUITS SOME OF TH...	DAVID AYER
...	...	...	...
Secret in Their Eyes	CRIME,DRAMA,MYSTERY	A TIGHT-KNIT TEAM OF RISING INVESTIGATORS, ALO...	BILLY RAY
Hostel: Part II	HORROR	THREE AMERICAN COLLEGE STUDENTS STUDYING ABROA...	ELI ROTH
Step Up 2: The Streets	DRAMA,MUSIC,ROMANCE	ROMANTIC SPARKS OCCUR BETWEEN TWO DANCE STUDEN...	JON M. CHU
Search Party	ADVENTURE,COMEDY	A PAIR OF FRIENDS EMBARK ON A MISSION TO REUNI...	SCOT ARMSTRONG
Nine Lives	COMEDY,FAMILY,FANTASY	A STUFFY BUSINESSMAN FINDS HIMSELF TRAPPED INS...	BARRY SONNENFELD

[1000 rows x 3 columns]

Apply

- Apply function can be used for individual columns:

```
>>> df.Genre
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi
Prometheus                 Adventure,Mystery,Sci-Fi
Split                             Horror,Thriller
Sing                        Animation,Comedy,Family
Suicide Squad              Action,Adventure,Fantasy
                                              ...
Secret in Their Eyes            Crime,Drama,Mystery
Hostel: Part II                                Horror
Step Up 2: The Streets          Drama,Music,Romance
Search Party                       Adventure,Comedy
Nine Lives                    Comedy,Family,Fantasy
Name: Genre, Length: 1000, dtype: object
```

```
>>> df.Genre.apply(str.upper)
Title
Guardians of the Galaxy     ACTION,ADVENTURE,SCI-FI
Prometheus                 ADVENTURE,MYSTERY,SCI-FI
Split                             HORROR,THRILLER
Sing                        ANIMATION,COMEDY,FAMILY
Suicide Squad              ACTION,ADVENTURE,FANTASY
                                              ...
Secret in Their Eyes            CRIME,DRAMA,MYSTERY
Hostel: Part II                                HORROR
Step Up 2: The Streets          DRAMA,MUSIC,ROMANCE
Search Party                       ADVENTURE,COMEDY
Nine Lives                    COMEDY,FAMILY,FANTASY
Name: Genre, Length: 1000, dtype: object
```

Apply

- Find movies in which Chris Pratt was casted.

```
>>> df[df.Actors.apply(lambda x: 'Chris Pratt' in x)]
                        Rank                        Genre                                        Description
Title
Guardians of the Galaxy    1    Action,Adventure,Sci-Fi  A group of intergalactic criminals are forced ...
Passengers                10    Adventure,Drama,Romance  A spacecraft traveling to a distant colony pla...
The Magnificent Seven     39   Action,Adventure,Western  Seven gunmen in the old west gradually come to...
Jurassic World            86    Action,Adventure,Sci-Fi  A new theme park, built on the original site o...
The Lego Movie           385  Animation,Action,Adventure  An ordinary Lego construction worker, thought ...
Zero Dark Thirty         407      Drama,History,Thriller  A chronicle of the decade-long hunt for al-Qae...
10 Years                 697        Comedy,Drama,Romance  The night before their high school reunion, a ...

[7 rows x 11 columns]
```

- Find animation movies

```
>>> df = df[df.Genre.apply(lambda x: 'Animation' in x)]
                        Rank                       Genre                                        Description
Title
Sing                       4    Animation,Comedy,Family  In a city of humanoid animals, a hustling thea...
Moana                     14  Animation,Adventure,Comedy  In Ancient Polynesia, when a terrible curse in...
The Secret Life of Pets   16  Animation,Adventure,Comedy  The quiet life of a terrier named Max is upend...
Trolls                    24  Animation,Adventure,Comedy  After the Bergens invade Troll Village, Poppy,...
Sausage Party             41  Animation,Adventure,Comedy  A sausage strives to discover the truth about ...
Zootopia                  75  Animation,Adventure,Comedy  In a city of anthropomorphic animals, a rookie...
Kimi no na wa             97     Animation,Drama,Fantasy  Two strangers find themselves linked in a biza...
Finding Dory             120  Animation,Adventure,Comedy  The friendly but forgetful blue tang fish, Dor...
Kubo and the Two Strings 172  Animation,Adventure,Family  A young boy named Kubo must locate a magical s...
Frozen                   175  Animation,Adventure,Comedy  When the newly crowned Queen Elsa accidentally...
```

Map

- A similar function «map» can also be used for the same purpose.

```
>>> df.Genre
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi
Prometheus                 Adventure,Mystery,Sci-Fi
Split                              Horror,Thriller
Sing                        Animation,Comedy,Family
Suicide Squad              Action,Adventure,Fantasy
                                               ...
Secret in Their Eyes            Crime,Drama,Mystery
Hostel: Part II                                 Horror
Step Up 2: The Streets          Drama,Music,Romance
Search Party                       Adventure,Comedy
Nine Lives                    Comedy,Family,Fantasy
Name: Genre, Length: 1000, dtype: object
```

```
>>> df.Genre.map(lambda x: x.upper())
Title
Guardians of the Galaxy     ACTION,ADVENTURE,SCI-FI
Prometheus                 ADVENTURE,MYSTERY,SCI-FI
Split                              HORROR,THRILLER
Sing                        ANIMATION,COMEDY,FAMILY
Suicide Squad              ACTION,ADVENTURE,FANTASY
                                               ...
Secret in Their Eyes            CRIME,DRAMA,MYSTERY
Hostel: Part II                                 HORROR
Step Up 2: The Streets          DRAMA,MUSIC,ROMANCE
Search Party                       ADVENTURE,COMEDY
Nine Lives                    COMEDY,FAMILY,FANTASY
Name: Genre, Length: 1000, dtype: object
```

Combining Data Frames

Mapping:

```
lookup = sales_team.set_index('Company Name')['Sales Rep']
lookup
```

```
Company Name
Chimera-Chasing Casbah          Jessie Mcallister
Tangential Sheds                Jessie Mcallister
Two-Mile Grab                   Jessie Mcallister
                                      ...
Spotty Adoniram                   Hazel Dickerson
Physicochemical Impatience        Hazel Dickerson
Fierce Productivity               Hazel Dickerson
Name: Sales Rep, Length: 4725, dtype: object
```

```
order_data['Sales Rep'] = order_data['Company Name'].map(lookup)
order_data
```


```
order_data['Sales Rep'] = order_data['Company Name'].map(lookup)
order_data
```


	Order Id	Company Id	Company Name	Date	Order Value	Converted	Sales Rep
0	80EYLOKP9E762WKG	LJKS5NK6788CYMUU	Chimera-Chasing Casbah	2017-02-18	4875	1	Jessie Mcallister
1	TLEXR1HZWTUTBHPB	LJKS5NK6788CYMUU	Chimera-Chasing Casbah	2015-07-30	8425	0	Jessie Mcallister
2	839FKFW2LLX4LMBB	LJKS5NK6788CYMUU	Chimera-Chasing Casbah	2016-05-27	4837	0	Jessie Mcallister
...	...	...	...	...	...	...	...
99997	NKHFWT5I2J9LPAPG	E4K99D4JR9E40VE1	Fierce Productivity	2017-09-06	5138	0	Hazel Dickerson
99998	OF79M3H9C44UM6PO	E4K99D4JR9E40VE1	Fierce Productivity	2015-10-29	2036	0	Hazel Dickerson
99999	EHN2QR78K2RX0FQ9	E4K99D4JR9E40VE1	Fierce Productivity	2015-03-07	7183	0	Hazel Dickerson

Groupby

- Groupby function provides sql-like grouping of data as well as using aggregate functions.
- Data can be grouped on one or more columns.
- Suppose we want to get the total revenues of the directors.

```
>>> df.groupby('Director')['Revenue (Millions)'].sum()
```

```
>>> df.groupby('Director').aggregate({'Revenue (Millions)':np.sum})
                    Revenue (Millions)
Director
Aamir Khan                        1.20
Abdellatif Kechiche               2.20
Adam Leon                         0.00
Adam McKay                      438.14
Adam Shankman                   157.33
...                                ...
Xavier Dolan                      3.49
Yimou Zhang                      45.13
Yorgos Lanthimos                  8.81
Zack Snyder                     975.74
Zackary Adler                     6.53

[644 rows x 1 columns]
```

Groupby

- Groupby function can be applied over multiple columns as well.
- When grouped, grouping column becomes index.
- Suppose we want to get the total revenues for each year.

```
>>> df.groupby('Year').aggregate({'Revenue (Millions)':np.sum})
      Revenue (Millions)
Year
2006             3624.46
2007             4306.23
2008             5053.22
2009             5292.26
2010             5989.65
2011             5431.96
2012             6910.29
2013             7666.72
2014             7997.40
2015             8854.12
2016            11211.65
```

Groupby

- There 1000 movies in our data set. There are multiple titles for one director.
- Lets find the number of directors.

```
>>> df.groupby('Director').count()
                     Rank  Genre  Description  Actors  Year  Runtime (Minutes)
Director
Aamir Khan              1      1            1       1     1                  1
Abdellatif Kechiche     1      1            1       1     1                  1
Adam Leon               1      1            1       1     1                  1
Adam McKay              4      4            4       4     4                  4
Adam Shankman           2      2            2       2     2                  2
...                   ...    ...          ...     ...   ...                ...
Xavier Dolan            2      2            2       2     2                  2
Yimou Zhang             1      1            1       1     1                  1
Yorgos Lanthimos        2      2            2       2     2                  2
Zack Snyder             5      5            5       5     5                  5
Zackary Adler           1      1            1       1     1                  1

[644 rows x 10 columns]
```

Groupby

- The aggregate function applies to all columns. However a specific column can be selected for applying the aggregate function.
- Suppose we want to get the total number of votes for each director.

```
>>> df.groupby('Director')['Votes'].sum()
Director
Aamir Khan                102697
Abdellatif Kechiche       103150
Adam Leon                   1031
Adam McKay                806827
Adam Shankman             167467
                             ...
Xavier Dolan               44218
Yimou Zhang                56036
Yorgos Lanthimos          172259
Zack Snyder              2301544
Zackary Adler               1630
Name: Votes, Length: 644, dtype: int64
```

Groupby

- Multiple aggregate functions can be used simultaneously.

```
>>> df.groupby('Director')['Rating'].agg(['min','max'])
                      min  max
Director
Aamir Khan            8.5  8.5
Abdellatif Kechiche   7.8  7.8
Adam Leon             6.5  6.5
Adam McKay            6.6  7.8
Adam Shankman         5.9  6.7
...                   ...  ...
Xavier Dolan          7.0  8.1
Yimou Zhang           6.1  6.1
Yorgos Lanthimos      7.1  7.3
Zack Snyder           6.1  7.7
Zackary Adler         5.1  5.1
```

Groupby

- There is also a «sort» keyword which is True by default.

```
>>> df.groupby('Director',sort=False)['Rating'].agg(['min','max'])
                      min  max
Director
James Gunn            6.5  8.1
Ridley Scott          5.3  8.0
M. Night Shyamalan    4.2  7.3
Christophe Lourdelet  7.2  7.2
David Ayer            6.2  7.7
...                   ...  ...
Patrick Tatopoulos    6.6  6.6
Aamir Khan            8.5  8.5
Nima Nourizadeh       6.7  6.7
Billy Ray             6.2  6.2
Scot Armstrong        5.6  5.6
```

Groupby

- There are several useful functions to apply as well.
- Suppose we want to get the 5 directors whose movies voted the most.

```
>>> df.groupby('Director')['Votes'].sum().nlargest(5)
Director
Christopher Nolan    6559085
Martin Scorsese      2966524
Quentin Tarantino    2559586
David Fincher        2309652
Zack Snyder          2301544
Name: Votes, dtype: int64
```

```
>>> df.groupby('Director')['Votes'].median()
Director
Alessandro Carloni       89791.0
Andrew Stanton          466961.5
Ash Brannon               1109.0
Brad Bird               504039.0
Byron Howard            296853.0
Carlos Saldanha         173919.0
Chris Buck              451894.0
Chris Renaud            120259.0
Christophe Lourdelet     60545.0
Claude Barras             4370.0
```

Groupby-Aggregate Functions

- .agg()
- .aggregate()
- .all()
- .any()
- .apply()
- .corr()
- .corrwith()
- .count()
- .cov()
- .cumcount()
- .cummax()
- .cummin()
- .cumprod()
- .cumsum()
- .describe()
- .idxmax()
- .idxmin()
- .mad()
- .max()
- .mean()
- .median()
- .min()
- .nunique()
- .prod()
- .sem()
- .size()
- .skew()
- .std()
- .sum()
- .var()

Iterating over Groups

- There are two fundamental ways of iterating over groups created by groupby() method:

```
groups = df.groupby("X")

for name, group in groups:
    print(name)
    print(group)
    print("\n")
```

Finding Max/Min

- There are several approachs to find the maximum value in a column.
- Using the «max» command of the dataframe, the maximums of each column can be retrieved.

```
>>> df.max()
Rank                                                             1000
Genre                                                   Thriller,War
Description          Young, up-and-coming martial artist, Bruce Lee...
Director                                                Zackary Adler
Actors               Zooey Deschanel, Joseph Gordon-Levitt, Geoffre...
Year                                                             2016
Runtime (Minutes)                                                 191
Rating                                                            9.0
Votes                                                         1791916
Revenue (Millions)                                             936.63
Metascore                                                       100.0
dtype: object
>>>
```

Finding Max/Min

- The max/min of a specific column can be retrieved through associated min/max functions.

```
>>> df['Rating'].max()
9.0
```

```
>>> df.max()['Rating']
9.0
```

```
>>> df['Rating'].min()
1.9
```

- If we want to get the max/min of a specific row instead column, then «axis» keyword is set as 1.

```
>>> df.max(axis=1)
Title
Guardians of the Galaxy    757074.0
Prometheus                 485820.0
Split                      157606.0
Sing                        60545.0
Suicide Squad              393727.0
                               ...
Secret in Their Eyes        27585.0
Hostel: Part II             73152.0
Step Up 2: The Streets      70699.0
Search Party                 4881.0
Nine Lives                  12435.0
Length: 1000, dtype: float64
```

Finding Max/Min

- The min/max of selected columns can be retrieved.

```
>>> df[['Rating','Votes']].max()
Rating           9.0
Votes      1791916.0
dtype: float64
```

- The position of minimum/maximum values of every column can also be retrieved. The position is the «index» of the min/max row.

```
>>> df[['Rating','Votes']].idxmax()
Rating    The Dark Knight
Votes     The Dark Knight
dtype: object
```

```
>>> df[['Rating','Metascore']].idxmax()
Rating       The Dark Knight
Metascore            Boyhood
dtype: object
```

Finding Max/Min

- Let's find the director of the movie which has the highest rating:

```
>>> df.loc[df['Rating'].idxmax(),'Director']
'Christopher Nolan'
```

- Let's find the director whose films have the highest revenue put together.

```
>>> df.groupby('Director')['Revenue (Millions)'].sum().idxmax()
'J.J. Abrams'
```

- The minimum counterpart of idxmin is also available.

```
>>> df.groupby('Director')['Revenue (Millions)'].sum().idxmin()
'Adam Leon'
```

Sorting

- The simplest form of sorting in Pandas is probably sorting the rows with respect to their corresponding indices.
- «sort_index()» is the function to sort the indices.
- Since our index column consists of movie titles, the result will be the alphabetically ordered movie titles:

```
>>> df.sort_index()
                      Rank                      Genre  ... Revenue (Millions) Metascore
Title                                                  ...
(500) Days of Summer   508        Comedy,Drama,Romance  ...              32.39      76.0
10 Cloverfield Lane    119        Drama,Horror,Mystery  ...              71.90      76.0
10 Years               697        Comedy,Drama,Romance  ...               0.20       NaN
12 Years a Slave       112     Biography,Drama,History  ...              56.67      96.0
127 Hours              818   Adventure,Biography,Drama  ...              18.33      82.0
...                    ...                         ...  ...                ...       ...
Zipper                 545              Drama,Thriller  ...                NaN      39.0
Zodiac                 278         Crime,Drama,History  ...              33.05      78.0
Zombieland             364     Adventure,Comedy,Horror  ...              75.59      73.0
Zoolander 2            432                      Comedy  ...              28.84      34.0
Zootopia                75  Animation,Adventure,Comedy  ...             341.26      78.0
```

Sorting

- For sorting columns «sort_values()» can be used.
- «sort_values()» has several keyword parameters, one of them is compulsory,(«by»), i.e. the column to which the sorting will be carried out.

```
>>> df.sort_values(by='Revenue (Millions)')
                          Rank                 Genre  ... Revenue (Millions) Metascore
Title                                                 ...
A Kind of Murder           232   Crime,Drama,Thriller  ...              0.00       50.0
Into the Forest            962  Drama,Sci-Fi,Thriller  ...              0.01       59.0
Love, Rosie                678        Comedy,Romance  ...              0.01       44.0
Lovesong                   322                 Drama  ...              0.01       74.0
Wakefield                   69                 Drama  ...              0.01       61.0
...                        ...                   ...  ...               ...        ...
Amateur Night              978                Comedy  ...               NaN       38.0
It's Only the End of the World 979            Drama  ...               NaN       48.0
Martyrs                    989                Horror  ...               NaN       89.0
Secret in Their Eyes       996   Crime,Drama,Mystery  ...               NaN       45.0
Search Party               999      Adventure,Comedy  ...               NaN       22.0

[1000 rows x 11 columns]
```

Sorting

- Another important keyword parameter is the «ascending» to specify the direction of sorting.
- The default value for «ascending» is «True».

```
>>> df.sort_values(by='Revenue (Millions)',ascending=False)
                                      Rank                 Genre  ... Revenue (Millions) Metascore
Title                                                             ...
Star Wars: Episode VII - The Force Awakens    51  Action,Adventure,Fantasy  ...             936.63       81.0
Avatar                                        88  Action,Adventure,Fantasy  ...             760.51       83.0
Jurassic World                                86   Action,Adventure,Sci-Fi  ...             652.18       59.0
The Avengers                                  77            Action,Sci-Fi  ...             623.28       69.0
The Dark Knight                               55      Action,Crime,Drama  ...             533.32       82.0
...                                          ...                     ...  ...                ...        ...
Amateur Night                                978                  Comedy  ...                NaN       38.0
It's Only the End of the World               979                   Drama  ...                NaN       48.0
Martyrs                                      989                  Horror  ...                NaN       89.0
Secret in Their Eyes                         996      Crime,Drama,Mystery  ...                NaN       45.0
Search Party                                 999         Adventure,Comedy  ...                NaN       22.0

[1000 rows x 11 columns]
```

Sorting

- Multiple columns can be given for sorting:

```
>>> df.sort_values(by=['Director','Revenue (Millions)'])[['Director','Revenue (Millions)']]
                                            Director  Revenue (Millions)
Title
Taare Zameen Par                          Aamir Khan                1.20
La vie d'Adèle                 Abdellatif Kechiche                2.20
Tramps                                   Adam Leon                 NaN
The Big Short                           Adam McKay                70.24
Step Brothers                           Adam McKay               100.47
...                                            ...                  ...
Watchmen                              Zack Snyder               107.50
300                                   Zack Snyder               210.59
Man of Steel                          Zack Snyder               291.02
Batman v Superman: Dawn of Justice    Zack Snyder               330.25
The Rise of the Krays                Zackary Adler                6.53

[1000 rows x 2 columns]
```

```
>>> df.Actors.str.split(',').apply(len).sort_values(ascending=False)
Title
Guardians of the Galaxy    4
The Hurt Locker            4
The Daughter               4
Pineapple Express          4
The First Time             4
                          ..
Blended                    4
Fast & Furious             4
Looper                     4
Nine Lives                 4
Antichrist                 3
Name: Actors, Length: 1000, dtype: int64
```

Data Types in Pandas

- There are 7 data types in Pandas:
  - `object` : This data type is used for strings (i.e., sequences of characters)
  - `int64` : Used for integers (whole numbers, no decimals)
  - `float64` : Used for floating-point numbers (i.e., figures with decimals/fractions)
  - `bool` : Used for values that can only be True/False
  - `datetime64` : Used for date and time values
  - `timedelta` : Used to represent the difference between datetimes
  - `category` : Used for values that take one out of a limited number of available options (categories don't have to, but can have explicit ordering)

Data Conversion in Pandas

- When we try to load data in a Pandas Dataframe, Pandas does its best to guess the data types (for assignment)
- However, it might be necessary to convert types
- There are two main methods to convert Pandas data types (typecasting):
  - <column>.astype(<desired type>)
  - conversion helper functions, like pd.to_numeric or pd.to_datetime

Data Conversion in Pandas

astype:

- astype is quick and works well with clean data and when the conversion is straight forward, e.g., from int64 to float64 (or vice versa).
- astype has to be called directly on the column that you want to convert. Like this:
  - *meals['type'] = meal['type'].astype('category')*
  - *events['date'] =events['date'].astype('datetime64')*
  - *country['population'] = country['population'].astype('int')*
- The change in types can be shown using data.dtypes command. (i.e. meals.dtype)

Data Conversion in Pandas

Conversion helper functions:

- There are three pd.to_<some_type> functions, only two of them come up frequently:
  - pd.to_numeric()
  - pd.to_datetime()
  - pd.to_timedelta()
- Their main advantage over astype, is that it is possible to specify the behavior in case a value is encountered, that can not be converted.
- Both functions accept an additional parameter errors that defines how errors should be treated.
- We could choose to ignore errors by passingerrors='ignore' , or turn the offending values into np.nan values by passing errors='coerce'.
- The default behavior is to raise errors.

Data Conversion in Pandas

pd.to_numeric():

```
ser = pd.Series(['Geeks', 11, 22.7,
33])

pd.to_numeric(ser, errors ='coerce')
```

```
0     NaN
1    11.0
2    22.7
3    33.0
dtype: float64
```

```
0     0.0
1    99.0
2    30.0
3    28.0
4     8.0
5    90.0
6    55.0
7    41.0
8    12.0
9    36.0
Name: Number, dtype: float64
```

```
pd.to_numeric(ser, downcast ='signed')
```

```
0     0
1    99
2    30
3    28
4     8
5    90
6    55
7    41
8    12
9    36
Name: Number, dtype: int8
```

Data Conversion in Pandas

pd.to_datetime():

- The method converts a string into a datetime format.
- A simple call to_datetime is like:
- pd.to_datetime(meals['date of meal'])
- Pandas will then guess the format and try to parse the date from the Input.

```
IN:
print(pd.to_datetime('2019-8-1'))
print(pd.to_datetime('2019/8/1'))
print(pd.to_datetime('8/1/2019'))
print(pd.to_datetime('Aug, 1 2019'))
print(pd.to_datetime('Aug - 1 2019'))
print(pd.to_datetime('August - 1 2019'))
print(pd.to_datetime('2019, August - 1'))
print(pd.to_datetime('20190108'))
```

```
OUT:
2019-08-01 00:00:00
2019-08-01 00:00:00
2019-08-01 00:00:00
2019-08-01 00:00:00
2019-08-01 00:00:00
2019-08-01 00:00:00
2019-08-01 00:00:00
2019-01-08 00:00:00
```

Data Conversion in Pandas

pd.to_datetime():

- A custom format could also be defined using the string formatting options:
  - pd.to_datetime('20190108',format='%Y%d%m’).
- One additional noteworthy parameter when working with custom formats is the option exact=False.
  - print(pd.to_datetime('yolo 20190108', format='%Y%d%m', exact=False)) will work, while it would fail without the exact parameter. With exact=False Pandas tries to match the pattern anywhere in the date string.

Data Conversion in Pandas

Accessor Functions:

- Accessor methods are highly specialized functions that acts as an interface to methods specific to the type you are trying to access.
- Those methods are highly specialized. They serve one job and one job only. However, they are excellent and extremely concise for that particular job.
- There are three different accessors:
  - dt
  - str
  - cat
- All of the methods are accessed by calling .<accessor>.method on the column of choice, like this: invoices['Date of Meal'].dt.date

Data Conversion in Pandas

Accessor - dt:

```
IN:
invoices['Date of Meal'].dt.weekday_name

OUT:
0          Thursday
1            Monday
```

```
IN:
invoices['Date of Meal'].dt.days_in_month

OUT:
0         31
1         31
          ..
49972     30
49973     31
Name: Date of Meal, Length: 49974, dtype: int64
```

```
IN:
invoices['Date of Meal'].dt.month_name()

OUT:
0           January
1           October
            ...
49972     September
49973        August
Name: Date of Meal, Length: 49974, dtype: object
```

```
IN:
invoices['Date of Meal'].dt.week

OUT:
0          5
1         44
          ..
49972     36
49973     34
Name: Date of Meal, Length: 49974, dtype: int64
```

nanosecond,**microsecond**,**second**,**minute**,**hour** ,**day**, **week**, **month**, **quarter**, **year** gets the integer of the corresponding frequency.

Data Conversion in Pandas

Accessor - str:

```
IN:
invoices['Type of Meal'].str.lower()

OUT:
0          breakfast
1             dinner
              ...
49972      breakfast
49973         dinner
Name: Type of Meal, Length: 49974, dtype: object
```

```
IN:
invoices['Type of Meal'].str.endswith('ast')

OUT:
0           True
1          False
           ...
49972       True
49973      False
Name: Type of Meal, Length: 49974, dtype: bool
```

```
IN:
invoices['Type of Meal'].str.upper()

OUT:
0          BREAKFAST
1             DINNER
              ...
49972      BREAKFAST
49973         DINNER
Name: Type of Meal, Length: 49974, dtype: object
```

```
IN:
invoices['Type of Meal'].str.repeat(2)

OUT:
0          BreakfastBreakfast
1                DinnerDinner
                    ...
49972      BreakfastBreakfast
49973            DinnerDinner
Name: Type of Meal, Length: 49974, dtype: object
```

Data Conversion in Pandas

Accessor - str:

```
>>> data.head().Director
Title
Guardians of the Galaxy                 James Gunn
Prometheus                             Ridley Scott
Split                                M. Night Shyamalan
Sing                               Christophe Lourdelet
Suicide Squad                             David Ayer
Name: Director, dtype: object
>>> data.head().Director.str.upper()
Title
Guardians of the Galaxy                 JAMES GUNN
Prometheus                             RIDLEY SCOTT
Split                                M. NIGHT SHYAMALAN
Sing                               CHRISTOPHE LOURDELET
Suicide Squad                             DAVID AYER
Name: Director, dtype: object
```

Data Conversion in Pandas

Accessor - cat:

cat provides access to a couple of categorial operations, like:

- `ordered` lets you know if the column is ordered

```
IN:
invoices['Type of Meal'].cat.ordered

OUT:
False
```

- `categories` to return the categories

```
IN:
invoices['Type of Meal'].cat.categories

OUT:
Index(['Breakfast', 'Dinner', 'Lunch'], dtype='object')
```

- `codes` for quick conversion of the category into its numerical representation

```
IN:
invoices['Type of Meal'].cat.codes

OUT:
0         0
1         1
2         2
3         2
4         2
         ..
49969     1
49970     2
49971     1
49972     0
49973     1
Length: 49974, dtype: int8
```

Combining Data Frames

Concatenation:

- Concatenating comes in handy when you have similar data (structurally and in terms of content) spread out across multiple files.
- You can concatenate data vertically (i.e., stack the data on top of each other) or horizontally (i.e., stack the data next to each other).
- Implemented using pd.concat() function
- A special case of «concat» can be found as «append». However, it does not provide a benefit over concatenating.

Combining Data Frames

Concatenation:

- Parameters of pd.concat function:

- `axis` : `0` for vertical, `1` for horizontal. `axis` defaults to `0`
- `join` : `'inner'` for the intersection, `'outer'` for the union of indices of the non-concatenating axis. When we use `axis=0` and `join='inner'` we will consider only overlapping columns. When using `axis=1` and `join='inner'` we consider only overlapping indices. In the case of `outer` non-overlapping columns/indices will be filled with `nan` values. `join` defaults to `outer`
- `ignore_index` : `True` to ignore preexisting indices and instead use labels from 0 to n-1 for the resulting DataFrame. `ignore_index` defaults to `False`
- `keys` : If we provide a list (has to be the same length as the number of DataFrames) a hierarchical index will be constructed. `keys` defaults to `None`. Use `keys` for example, to add the source of the data. Best used in combination with names.
- `names` : Assuming that you provide keys, the names will be used to label the resulting hierarchical index. `names` defaults to `None`.

Concat

- Suppose we have one dataframe for each year.
- We can combine two or more years into one.

```
>>> movies_2010 = df[df['Year'] == 2010]
>>> movies_2011 = df[df['Year'] == 2011]
```

```
>>> movies_10_11 = pd.concat([movies_2010,movies_2011])
>>> movies_10_11['Year']
Title
Despicable Me                  2010
Tangled                        2010
Megamind                       2010
Toy Story 3                    2010
How to Train Your Dragon       2010
Cars 2                         2011
Rio                            2011
Name: Year, dtype: int64
```

Combining Data Frames

Merging:

- Merging, as opposed to concatenating DataFrames together, allows us to combine two DataFrames in a more traditional SQL-query kind of way.
- When merging DataFrames, most of the time you want some information from one source and another piece of information from another source.
- Whereas when concatenating your DataFrames are structurally and in terms of content quite similar, and you want to combine them into one unified DataFrame.
- Merging two DataFrames in Pandas is done with pd.merge.

Combining Data Frames

Merging:

- Parameters of pd.merge function:

- `left/right` : the left, respectively right, DataFrame you want to merge
- `how` : `'left'`, `'right'`, `'outer'`, `'inner'`. `how` defaults to `'inner'`. See below a schematic overview of what each of them does. We will discuss specific examples a little bit later.

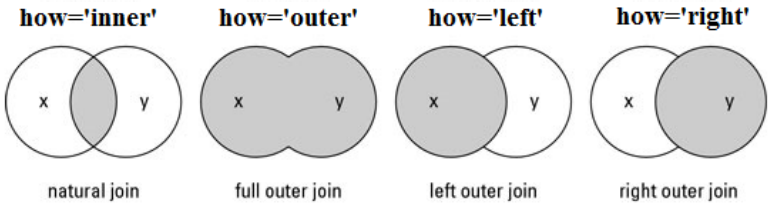


Image from DataScienceMadeSimple

- `left_index/right_index` : If `True`, use the index from the left/right DataFrame to merge on. `left_index/right_index` defaults to `False`
- `on` : Column name(s) to merge on. Column name(s) have to exist in both the left and right DataFrame. If not passed and `left_index` and `right_index` are `False`, **the intersection of the columns in both DataFrames will be inferred to join on.**
- `left_on/right_on` : Column name(s) from the left/right DataFrame to join on. **Typical use case:** Keys you are joining on are differently labeled in your DataFrames. E.g., what is `location_id` in your left DataFrame, might be `_id` in your right DataFrame. In this case, you would do `left_on='location_id', right_on='_id'`.
- `suffixes` : A tuple of string suffixes to apply to overlapping columns. `suffixes` defaults to `('_x', '_y')`. I like to use `('_base', '_joined')`.

Merge

- Suppose we have two dataframes with some overlapping index.

```
>>> df1 = df.iloc[:10,:]
>>> df2 = df.iloc[6:14,:]
>>> df1
                        Rank                     Genre  ... Revenue (Millions) Metascore
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi  ...              333.13      76.0
Prometheus                 2  Adventure,Mystery,Sci-Fi  ...              126.46      65.0
Split                      3           Horror,Thriller  ...              138.12      62.0
Sing                       4   Animation,Comedy,Family  ...              270.32      59.0
Suicide Squad              5  Action,Adventure,Fantasy  ...              325.02      40.0
The Great Wall             6  Action,Adventure,Fantasy  ...               45.13      42.0
La La Land                 7        Comedy,Drama,Music  ...              151.06      93.0
Mindhorn                   8                    Comedy  ...                 NaN      71.0
The Lost City of Z         9  Action,Adventure,Biography  ...               8.01      78.0
Passengers                10   Adventure,Drama,Romance  ...              100.01      41.0
```

```
>>> df2
                                  Rank                     Genre  ... Revenue (Millions) Metascore
Title                                                             ...
La La Land                           7        Comedy,Drama,Music  ...              151.06      93.0
Mindhorn                             8                    Comedy  ...                 NaN      71.0
The Lost City of Z                   9  Action,Adventure,Biography  ...               8.01      78.0
Passengers                          10   Adventure,Drama,Romance  ...              100.01      41.0
Fantastic Beasts and Where to Find Them  11  Adventure,Family,Fantasy  ...              234.02      66.0
Hidden Figures                      12   Biography,Drama,History  ...              169.27      74.0
Rogue One                           13   Action,Adventure,Sci-Fi  ...              532.17      65.0
Moana                               14  Animation,Adventure,Comedy  ...              248.75      81.0

[8 rows x 11 columns]
```

Merge

- Selecting the type or merge as «outer», merge behaves like «concat» function.
- However, common rows (having the same index) will take place only once.

```
>>> pd.merge(df1,df2,how='outer')
    Rank                        Genre  ... Revenue (Millions) Metascore
0      1     Action,Adventure,Sci-Fi  ...              333.13       76.0
1      2    Adventure,Mystery,Sci-Fi  ...              126.46       65.0
2      3             Horror,Thriller  ...              138.12       62.0
3      4     Animation,Comedy,Family  ...              270.32       59.0
4      5    Action,Adventure,Fantasy  ...              325.02       40.0
5      6    Action,Adventure,Fantasy  ...               45.13       42.0
6      7          Comedy,Drama,Music  ...              151.06       93.0
7      8                      Comedy  ...                 NaN       71.0
8      9  Action,Adventure,Biography  ...                8.01       78.0
9     10     Adventure,Drama,Romance  ...              100.01       41.0
10    11    Adventure,Family,Fantasy  ...              234.02       66.0
11    12     Biography,Drama,History  ...              169.27       74.0
12    13     Action,Adventure,Sci-Fi  ...              532.17       65.0
13    14  Animation,Adventure,Comedy  ...              248.75       81.0
```

Merge

- Selecting the type or merge as «inner», merge behaves like intersection.
- However, it does more than that. Columns will be merged if it is not shared between dataframes.

```
>>> pd.merge(df1,df2,how='inner')
   Rank                        Genre  ... Revenue (Millions) Metascore
0     7          Comedy,Drama,Music  ...             151.06      93.0
1     8                       Comedy  ...                NaN      71.0
2     9  Action,Adventure,Biography  ...               8.01      78.0
3    10     Adventure,Drama,Romance  ...             100.01      41.0

[4 rows x 11 columns]
```

- Another important keyword for merge is the «on» which determines the column on which the merge is applied.
- The default behaviour is the merge on the indices of the dataframes.

Merge

- In this example, there are non-overlapping columns.

```
>>> df1
                                    Genre                                  Description  Runtime (Minutes)  Rating
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi  A group of intergalactic criminals are forced ...                121     8.1
Prometheus                  Adventure,Mystery,Sci-Fi  Following clues to the origin of mankind, a te...                124     7.0
Split                              Horror,Thriller  Three girls are kidnapped by a man with a diag...                117     7.3
Sing                         Animation,Comedy,Family  In a city of humanoid animals, a hustling thea...                108     7.2
Suicide Squad               Action,Adventure,Fantasy  A secret government agency recruits some of th...                123     6.2
The Great Wall              Action,Adventure,Fantasy  European mercenaries searching for black powde...                103     6.1
La La Land                       Comedy,Drama,Music  A jazz pianist falls for an aspiring actress i...                128     8.3
Mindhorn                                    Comedy  A has-been actor best known for playing the ti...                 89     6.4
The Lost City of Z       Action,Adventure,Biography  A true-life drama, centering on British explor...                141     7.1
Passengers                  Adventure,Drama,Romance  A spacecraft traveling to a distant colony pla...                116     7.0
>>> df2
                                        Runtime (Minutes)  Rating   Votes  Revenue (Millions)
Title
La La Land                                            128     8.3  258682              151.06
Mindhorn                                               89     6.4    2490                 NaN
The Lost City of Z                                    141     7.1    7188                8.01
Passengers                                            116     7.0  192177              100.01
Fantastic Beasts and Where to Find Them               133     7.5  232072              234.02
Hidden Figures                                        127     7.8   93103              169.27
Rogue One                                             133     7.9  323118              532.17
Moana                                                 107     7.7  118151              248.75
```

```
>>> pd.merge(df1,df2)
                    Genre                                  Description  Runtime (Minutes)  Rating   Votes  Revenue (Millions)
0         Comedy,Drama,Music  A jazz pianist falls for an aspiring actress i...                128     8.3  258682              151.06
1                     Comedy  A has-been actor best known for playing the ti...                 89     6.4    2490                 NaN
2  Action,Adventure,Biography  A true-life drama, centering on British explor...                141     7.1    7188                8.01
3     Adventure,Drama,Romance  A spacecraft traveling to a distant colony pla...                116     7.0  192177              100.01
```

Melt

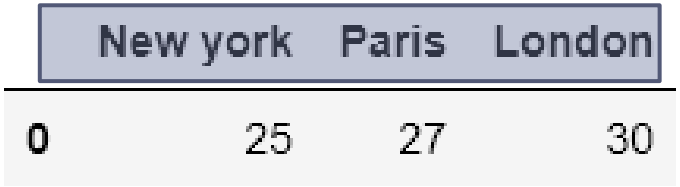
- Most of the time, we don't get data in the exact form we want.
- For example, sometimes we might have data in columns which we might need in rows.
- Suppose our dataset is organized such that every genre is listed as columns and movie titles have 0s or 1s if that genre is included in that title.

```
>>> df
                     Title  Rating   Votes  Revenue (Millions)  Mystery  Sci-Fi  ...  Adventure  War  Action  Musical  History  Biography
0    Guardians of the Galaxy     8.1  757074              333.13        0       1  ...          1    0       1        0        0          0
1                 Prometheus     7.0  485820              126.46        1       1  ...          1    0       0        0        0          0
2                      Split     7.3  157606              138.12        0       0  ...          0    0       0        0        0          0
3                       Sing     7.2   60545              270.32        0       0  ...          0    0       0        0        0          0
4             Suicide Squad     6.2  393727              325.02        0       0  ...          1    0       1        0        0          0
..                      ...     ...     ...                 ...      ...     ...  ...        ...  ...     ...      ...      ...        ...
995    Secret in Their Eyes     6.2   27585                 NaN        1       0  ...          0    0       0        0        0          0
996          Hostel: Part II     5.5   73152               17.54        0       0  ...          0    0       0        0        0          0
997   Step Up 2: The Streets     6.2   70699               58.01        0       0  ...          0    0       0        0        0          0
998             Search Party     5.6    4881                 NaN        0       0  ...          1    0       0        0        0          0
999               Nine Lives     5.3   12435               19.64        0       0  ...          0    0       0        0        0          0

[1000 rows x 24 columns]
```

Melt

- What «melt» does is basically convert a column to a value in the associated row.



	New york	Paris	London
0	25	27	30

```
df.melt()
```

	variable	value
0	New york	25
1	Paris	27
2	London	30

variable and value are default column names

Note how the columns become row-wise data

Melt

- «melt» usually increases the number of rows.
- The column value repeats in each row.

	New york	Paris	London
0	25	27	30
1	27	22	31
2	23	24	33
3	25	26	29
4	29	28	25

```
df.melt()
```

	variable	value
0	New york	25
1	New york	27
2	New york	23
3	New york	25
4	New york	29
5	Paris	27
6	Paris	22
7	Paris	24
8	Paris	26
9	Paris	28
10	London	30
11	London	31
12	London	33
13	London	29
14	London	25

Melt

	New york	Paris	London
0	25	27	30

```
df.melt(var_name='city', value_name='temperature')
```

	city	temperature
0	New york	25
1	Paris	27
2	London	30

Melt

- If we want to turn only some of the columns into rows, pass the columns to keep as a list (even if it is a single value) to id_vars.
- id_vars stands for identity variables.

```
temperatures.melt(id_vars=['city'],
                  var_name='date',
                  value_name='temperature').sample(5)
```

	city	day1	day2	day3	day4	day5
0	New York	23	22	26	23	27
1	London	25	21	25	21	26
2	Paris	27	25	24	22	27
3	Berlin	26	26	27	26	24
4	Amsterdam	24	23	23	27	28

	city	date	temperature
20	New York	day5	27
10	New York	day3	26
15	New York	day4	23
13	Berlin	day3	27
5	New York	day2	22

Melt

- Now, let's get back to our movie data. Suppose we have the movie data as follows.
- This is not particularly a great structure to have data in.
- We might like it better if we had a dataframe with only one column Genre and we can have multiple rows repeated for the same movie.

```
>>> df
                        Title  Rating   Votes  Revenue (Millions)    Genre  Flag
0      Guardians of the Galaxy     8.1  757074              333.13  Fantasy     0
1                   Prometheus     7.0  485820              126.46  Fantasy     0
2                        Split     7.3  157606              138.12  Fantasy     0
3                         Sing     7.2   60545              270.32  Fantasy     0
4                Suicide Squad     6.2  393727              325.02  Fantasy     1
...                        ...     ...     ...                 ...      ...   ...
19995     Secret in Their Eyes     6.2   27585                 NaN   Comedy     0
19996           Hostel: Part II     5.5   73152               17.54   Comedy     0
19997   Step Up 2: The Streets     6.2   70699               58.01   Comedy     0
19998             Search Party     5.6    4881                 NaN   Comedy     1
19999               Nine Lives     5.3   12435               19.64   Comedy     1

[20000 rows x 6 columns]
```

Melt

- We can convert genre columns as rows.

```
df = pd.melt(df,id_vars = ['Title','Rating','Votes','Revenue (Millions)'],
                value_vars = list(genre_set),
                var_name = 'Genre',
                value_name ='Flag')
```

- id_vars: List of vars we want in the current form only.
- value_vars: List of vars we want to melt/put in the same column
- var_name: name of the column for value_vars
- value_name: name of the column for value of value_vars

```
>>> df
                         Title  Rating   Votes  Revenue (Millions)    Genre  Flag
0      Guardians of the Galaxy     8.1  757074              333.13  Fantasy     0
1                   Prometheus     7.0  485820              126.46  Fantasy     0
2                        Split     7.3  157606              138.12  Fantasy     0
3                         Sing     7.2   60545              270.32  Fantasy     0
4                 Suicide Squad     6.2  393727              325.02  Fantasy     1
...                         ...     ...     ...                 ...      ...   ...
19995     Secret in Their Eyes     6.2   27585                 NaN   Comedy     0
19996           Hostel: Part II     5.5   73152               17.54   Comedy     0
19997   Step Up 2: The Streets     6.2   70699               58.01   Comedy     0
19998             Search Party     5.6    4881                 NaN   Comedy     1
19999               Nine Lives     5.3   12435               19.64   Comedy     1

[20000 rows x 6 columns]
```

Pivot_table

- «pivot_table» does the opposite of «melt»

```
>>> df
                        Title  Rating   Votes  Revenue (Millions)    Genre  Flag
0      Guardians of the Galaxy     8.1  757074              333.13  Fantasy     0
1                   Prometheus     7.0  485820              126.46  Fantasy     0
2                        Split     7.3  157606              138.12  Fantasy     0
3                         Sing     7.2   60545              270.32  Fantasy     0
4                Suicide Squad     6.2  393727              325.02  Fantasy     1
...                        ...     ...     ...                 ...      ...   ...
19995     Secret in Their Eyes     6.2   27585                 NaN   Comedy     0
19996          Hostel: Part II     5.5   73152               17.54   Comedy     0
19997   Step Up 2: The Streets     6.2   70699               58.01   Comedy     0
19998             Search Party     5.6    4881                 NaN   Comedy     1
19999               Nine Lives     5.3   12435               19.64   Comedy     1

[20000 rows x 6 columns]
```

```
df = df.pivot_table(index=['Title','Rating','Votes',
                           'Revenue (Millions)'], columns='Genre',
                            values='Flag', aggfunc='sum').reset_index()
```

```
>>> df
Genre                  Title  Rating   Votes  Revenue (Millions)  Action  Adventure  ...  Romance  Sci-Fi  Sport  Thriller  War  Western
0      (500) Days of Summer     7.7  398972               32.39       0          0  ...        1       0      0         0    0        0
1       10 Cloverfield Lane     7.2  192968               71.90       0          0  ...        0       0      0         0    0        0
2                  10 Years     6.1   19636                0.20       0          0  ...        1       0      0         0    0        0
3           12 Years a Slave     8.1  486338               56.67       0          0  ...        0       0      0         0    0        0
4                  127 Hours     7.6  294010               18.33       0          1  ...        0       0      0         0    0        0
..                       ...     ...     ...                 ...     ...        ...  ...      ...     ...    ...       ...  ...      ...
867          Zero Dark Thirty     7.4  226661               95.72       0          0  ...        0       0      0         1    0        0
868                    Zodiac     7.7  329683               33.05       0          0  ...        0       0      0         0    0        0
869                Zombieland     7.7  409403               75.59       0          1  ...        0       0      0         0    0        0
870              Zoolander 2     4.7   48297               28.84       0          0  ...        0       0      0         0    0        0
```

Pivot

- A very similar function to «pivot_table» is the «pivot» function.

```
dfn = df.pivot(index='lon', columns='lat', values='Height')
```

```
>>> df
    lon  lat  Height
0    26   40     803
1    26   41     710
2    26   42     659
3    26   43     548
4    27   40     977
5    27   41     443
6    27   42     556
7    27   43     667
8    28   40     650
9    28   41     740
10   28   42     832
11   28   43     943
12   29   40     561
13   29   41     639
14   29   42     721
15   29   43     832
```

```
>>> dfn
lat    40    41    42    43
lon
26    803   710   659   548
27    977   443   556   667
28    650   740   832   943
29    561   639   721   832
```

Crosstab

- Crosstab provides a handy interface when we need to count the frequencies for groups formed by 3+ features.

```
>>> pd.crosstab(df.Director,df.Year)
```

Year	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Director											
Aamir Khan	0	1	0	0	0	0	0	0	0	0	0
Abdellatif Kechiche	0	0	0	0	0	0	0	1	0	0	0
Adam Leon	0	0	0	0	0	0	0	0	0	0	1
Adam McKay	1	0	1	0	1	0	0	0	0	1	0
Adam Shankman	0	1	0	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
Xavier Dolan	0	0	0	0	0	0	0	0	1	0	1
Yimou Zhang	0	0	0	0	0	0	0	0	0	0	1
Yorgos Lanthimos	0	0	0	1	0	0	0	0	0	1	0
Zack Snyder	1	0	0	1	0	1	0	1	0	0	1
Zackary Adler	0	0	0	0	0	0	0	0	0	1	0

```
[644 rows x 11 columns]
```

Crosstab

- There are several keyword parameters which allow creating quick tables.
- In the example below, we examine the titles by director over the years including the total number of titles by director or year.

```
>>> pd.crosstab(df.Director,df.Year,margins=True, margins_name="Total").sort_values(by='Total',ascending=False)
```

Year	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	Total
Director												
Total	44	53	52	51	60	63	64	91	98	127	297	1000
Ridley Scott	1	1	1	0	1	0	1	1	1	1	0	8
M. Night Shyamalan	1	0	1	0	1	0	0	1	0	1	1	6
Michael Bay	0	1	0	1	0	1	0	1	1	0	1	6
David Yates	0	1	0	1	1	1	0	0	0	0	2	6
...	...	...	...	...	...	...	...	...	...	...	...	...
Ilya Naishuller	0	0	0	0	0	0	0	0	0	1	0	1
Ido Fluk	0	0	0	0	0	0	0	0	0	0	1	1
Hugo Gélin	0	0	0	0	0	0	0	0	0	0	1	1
Hope Dickson Leach	0	0	0	0	0	0	0	0	0	0	1	1
Jon Kasdan	0	0	0	0	0	0	1	0	0	0	0	1

[645 rows x 12 columns]

Explode

- The explode() function is used to transform each element of a list-like to a row, replicating the index values.
- The column to explode should be str or list/tüple
- For our movie data, the genre is given as csv.

```
>>> df['Genre']
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi
Prometheus                 Adventure,Mystery,Sci-Fi
Split                              Horror,Thriller
Sing                        Animation,Comedy,Family
Suicide Squad              Action,Adventure,Fantasy
                                               ...
Secret in Their Eyes            Crime,Drama,Mystery
Hostel: Part II                                Horror
Step Up 2: The Streets          Drama,Music,Romance
Search Party                       Adventure,Comedy
Nine Lives                   Comedy,Family,Fantasy
Name: Genre, Length: 1000, dtype: object
```

```
>>> df['Genre'].str.split(',').explode()
Title
Guardians of the Galaxy       Action
Guardians of the Galaxy    Adventure
Guardians of the Galaxy       Sci-Fi
Prometheus                 Adventure
Prometheus                   Mystery
                                 ...
Search Party               Adventure
Search Party                  Comedy
Nine Lives                    Comedy
Nine Lives                    Family
Nine Lives                   Fantasy
Name: Genre, Length: 2555, dtype: object
```

References

1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). How to Think Like a Computer Scientist: Learning with Python (3nd edition).
2 Pilgrim, M. (2014). Dive into Python 3 by. Free online version: DiveIntoPython3.org ISBN: 978-1430224150.
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
5 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
6 John D. Hunter (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95.
7 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
8 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
9 http://www.geeksforgeeks.org
10 https://docs.python.org/3/tutorial/
11 http://www.python-course.eu
12 https://developers.google.com/edu/python/
13 http://learnpythonthehardway.org/book/
14 https://pythonbasics.org/
15 https://pandas.pydata.org/
16 https://towardsdatascience.com/