Pandas Data Structures

```
In [3]: ser = pd.Series(data = [100, 200, 300, 400, 500], index=['tom', 'bob', 'nancy', 'dan', 'eric'])

In [6]: ser
Out[6]: tom      100
        bob      200
        nancy    300
        dan      400
        eric     500
        dtype: int64

In [7]: ser.index
Out[7]: Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')

In [9]: ser[[4, 3, 1]]
Out[9]: eric    500
        dan     400
        bob     200
        dtype: int64

In [10]: ser['nancy']
Out[10]: 300

In [11]: 'bob' in ser
Out[11]: True

In [16]: ser * 2
Out[16]: tom       200
         bob       400
         nancy     600
         dan       800
         eric     1000
         dtype: int64

In [17]: ser ** 2
Out[17]: tom       10000
         bob       40000
         nancy     90000
         dan      160000
         eric     250000
         dtype: int64
```

Pandas Series

```
In [46]: d = {'one' : pd.Series([100., 200., 300.], index=['apple', 'ball', 'clock']),
              'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball', 'cerill', 'dancy'])}

In [47]: df = pd.DataFrame(d)
         df
```

Out[47]:

	one	two
apple	100.0	111.0
ball	200.0	222.0
cerill	NaN	333.0
clock	300.0	NaN
dancy	NaN	4444.0

Pandas Data Frames

```
In [48]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'])
```

Out[48]:

	one	two
dancy	NaN	4444.0
ball	200.0	222.0
apple	100.0	111.0

```
In [49]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'], columns=['two', 'five'])
```

Out[49]:

	two	five
dancy	4444.0	NaN
ball	222.0	NaN
apple	111.0	NaN

```
In [50]: df.index
Out[50]: Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')

In [51]: df.columns
Out[51]: Index(['one', 'two'], dtype='object')
```

Series

- A 1-dimensional labeled array
- Supports many data types
- Axis labels -> index
  - get and set values by index labe
- Valid argument to most NumPy methods

```
In [3]: ser = pd.Series(data = [100, 200, 300, 400, 500], index=['tom', 'bob', 'nancy', 'dan', 'eric'])

In [6]: ser

Out[6]: tom      100
        bob      200
        nancy    300
        dan      400
        eric     500
        dtype: int64

In [7]: ser.index

Out[7]: Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')

In [9]: ser[[4, 3, 1]]

Out[9]: eric    500
        dan     400
        bob     200
        dtype: int64

In [10]: ser['nancy']

Out[10]: 300

In [11]: 'bob' in ser

Out[11]: True

In [16]: ser * 2

Out[16]: tom       200
         bob       400
         nancy     600
         dan       800
         eric     1000
         dtype: int64

In [17]: ser ** 2

Out[17]: tom       10000
         bob       40000
         nancy     90000
         dan      160000
         eric     250000
         dtype: int64
```

Pandas DataFrame

- A 2-dimensional labeled data structure
- A dictionary of Series objects
- Columns can be of potentially
  - different types
  - Optionally parameters for fine-tuning:
    - index(row labels)
    - columns(column labels)

Pandas provides many constructors to create DataFrames!

```
In [46]: d = {'one' : pd.Series([100., 200., 300.], index=['apple', 'ball', 'clock']),
              'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball', 'cerill', 'dancy'])}

In [47]: df = pd.DataFrame(d)
         df
```

Out[47]:

	one	two
apple	100.0	111.0
ball	200.0	222.0
cerill	NaN	333.0
clock	300.0	NaN
dancy	NaN	4444.0

```
In [48]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'])
```

Out[48]:

	one	two
dancy	NaN	4444.0
ball	200.0	222.0
apple	100.0	111.0

```
In [49]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'], columns=['two', 'five'])
```

Out[49]:

	two	five
dancy	4444.0	NaN
ball	222.0	NaN
apple	111.0	NaN

```
In [50]: df.index
Out[50]: Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')

In [51]: df.columns
Out[51]: Index(['one', 'two'], dtype='object')
```

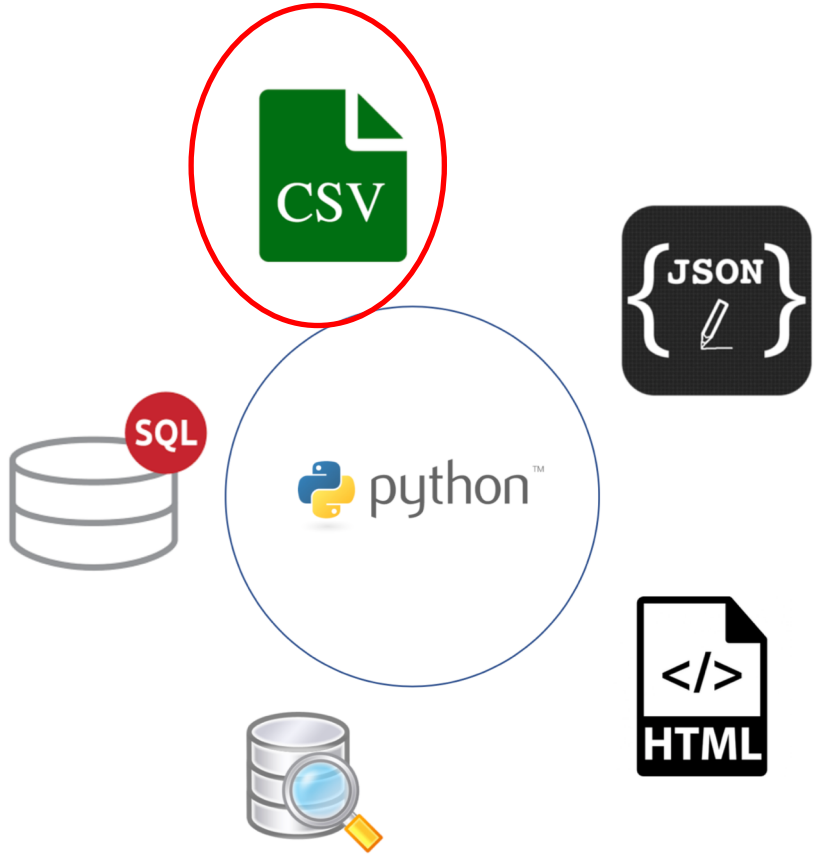
Loading Data

- Pandas can directly import data from various sources that are in different format into a **dataframe**
- The function to read from any format is:

  ‘read_<input format>`

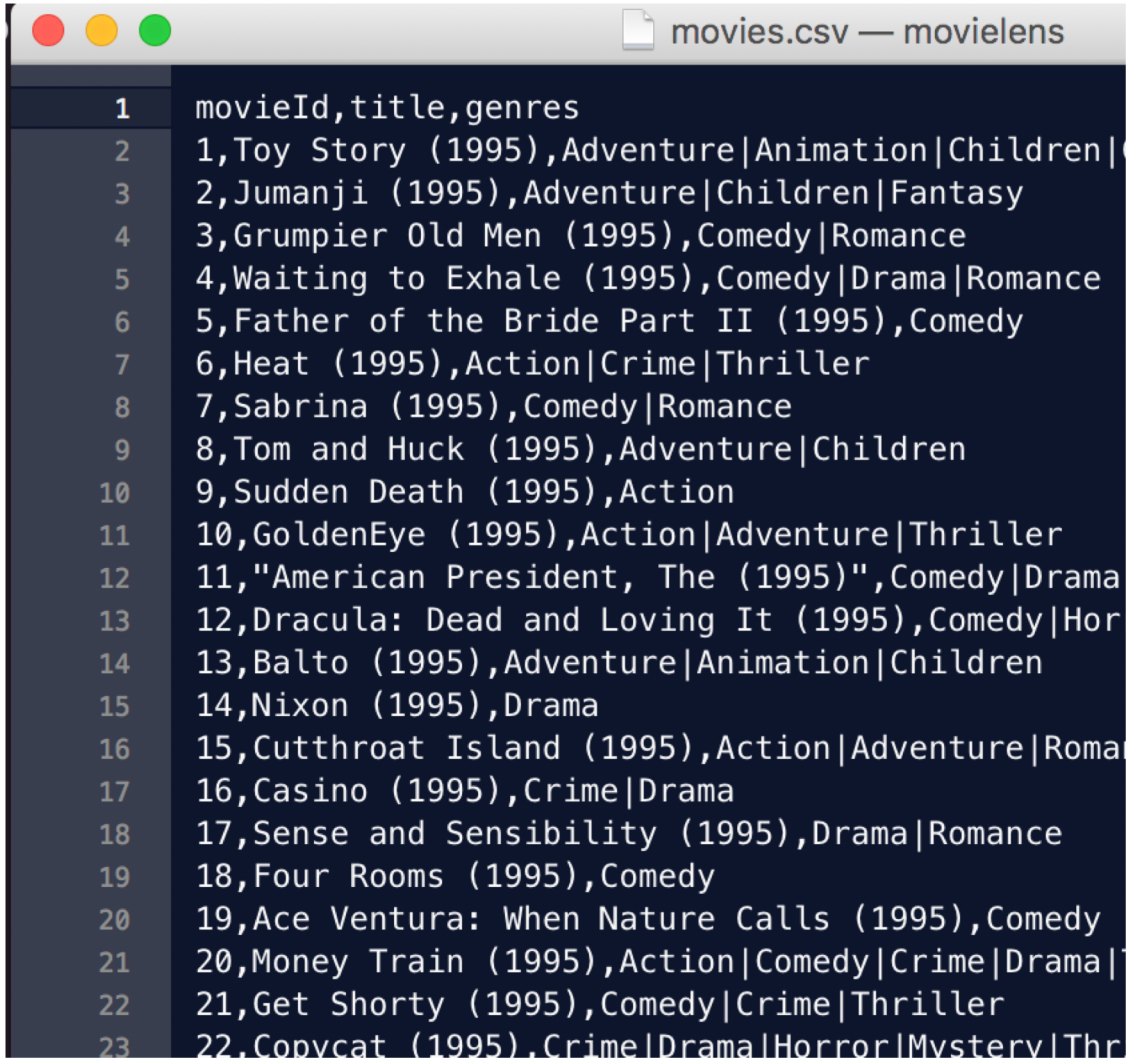
- We will usually read it from a ‘.csv’ file so we will use:

  read_csv()



read_csv()

- Input : Path to a Comma Separated File
- Output: Pandas DataFrame object containing contents of the file



Loading DataFrame

In [1]:
```
import pandas as pd
```

In [3]:
```
df=pd.read_csv('movies.csv')
```

pandas DataFrame

pandas DataFrame is a 2-dimensional labeled data structure (tables).

In [5]:
```
df
```

Out[5]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller
10	11	American President, The (1995)	Comedy|Drama|Romance
11	12	Dracula: Dead and Loving It (1995)	Comedy|Horror

Summary statistics

```
In [170]: df['rating'].describe()

Out[170]: count    26744.000000
          mean         3.133200
          std          0.664084
          min          0.500000
          25%          2.800000
          50%          3.235294
          75%          3.565217
          max          5.000000
          Name: rating, dtype: float64
```

- df['rating'].mean(); df['rating'].median(); df['rating'].std()
- df.corr()

func = min(), max(), mode(), median()

- The general syntax for calling these functions is
  - data_frame.func()
  - Frequently used optional parameter:
    - axis = 0 (rows) or 1 (columns)

mean()

- Syntax: data_frame.mean(axis={0 or 1})
  - Axis=0:Index
  - Axis = 1 : Columns
- Output: Series or DataFrame with the mean values

std()

- Syntax: data_frame.std(axis={0 or 1})
  - Axis=0:Index
  - Axis = 1 : Columns
- Output: Series or DataFrame with the Standard Deviation values
- Normalized by N-1

- Some other functions that are worth exploring:
    - Count()
    - Clip()
    - Rank()
    - Round()

Frequent Data Operations

Slice Out Columns

```
In [13]: df
```

Out[13]:

	movieId	title	genres	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.211977
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861393
4	5	Father of the Bride Part II (1995)	Comedy	3.064592
5	6	Heat (1995)	Action|Crime|Thriller	3.834930
6	7	Sabrina (1995)	Comedy|Romance	3.366484
7	8	Tom and Huck (1995)	Adventure|Children	3.142049
8	9	Sudden Death (1995)	Action	3.004924
9	10	GoldenEye (1995)	Action|Adventure|Thriller	3.430029

```
df['rating']
```

```
0     3.921240
1     3.211977
2     3.151040
3     2.861393
4     3.064592
5     3.834930
6     3.366484
7     3.142049
8     3.004924
9     3.430029
10    3.667713
11    2.619766
12    3.272416
13    3.432082
14    2.721993
15    3.787455
16    3.968573
17    3.373631
18    2.607412
19    2.880754
20    3.581689
21    3.319400
```

Filter Out Rows

```
In [17]: #select rows where rating is greater than 3
         df[df['rating']>3]
```

Out[17]:

	movieId	title	genres	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.211977
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040
4	5	Father of the Bride Part II (1995)	Comedy	3.064592
5	6	Heat (1995)	Action|Crime|Thriller	3.834930
6	7	Sabrina (1995)	Comedy|Romance	3.366484
7	8	Tom and Huck (1995)	Adventure|Children	3.142049
8	9	Sudden Death (1995)	Action	3.004924

Insert New Column

```
In [ ]: df['rating^2']=df['rating']**2
```

```
In [20]: df
```

Out[20]:

	movieId	title	genres	rating	rating^2
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240	15.376120
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.211977	10.316795
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040	9.929056
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861393	8.187572
4	5	Father of the Bride Part II (1995)	Comedy	3.064592	9.391722
5	6	Heat (1995)	Action|Crime|Thriller	3.834930	14.706691
6	7	Sabrina (1995)	Comedy|Romance	3.366484	11.333215
7	8	Tom and Huck (1995)	Adventure|Children	3.142049	9.872475

Add a New Row

```
In [ ]:
```

```
In [36]: df
```

26737	131250	No More School (2000)	Comedy	4.000
26738	131252	Forklift Driver Klaus: The First Day on the Jo...	Comedy|Horror	4.000
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.000
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.000
26741	131258	The Pirates (2014)	Adventure	2.500
26742	131260	Rentun Ruusu (2001)	(no genres listed)	3.000
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.000

26744 rows × 5 columns

```
In [ ]: df.loc[26744]=[131267, 'my new movie', 'Fantasy|Comedy', 2.5,2.5**2]
```

```
In [30]: df
```

26738	131252	Forklift Driver Klaus: The First Day on the Jo...	Comedy|Horror	4.000000	16.000000
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.000000	16.000000
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.000000	16.000000
26741	131258	The Pirates (2014)	Adventure	2.500000	6.250000
26742	131260	Rentun Ruusu (2001)	(no genres listed)	3.000000	9.000000
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.000000	16.000000
26744	131267	my new movie	Fantasy|Comedy	2.500000	6.250000

26745 rows × 5 columns

Update a row

```
In [ ]:
```

```
In [39]: df
```

Out[39]:

	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Ch
1	2	Jumanji (1995)	Adv
2	3	Grumpier Old Men (1995)	
3	4	Waiting to Exhale (1995)	Co
4	5	Father of the Bride Part II (1995)	
5	6	Heat (1995)	
6	7	Sabrina (1995)	
7	8	Tom and Huck (1995)	
8	9	Sudden Death (1995)	
9	10	GoldenEye (1995)	A

```
In [24]: df.loc[1]=['2','My new movie','Comedy', 5,25]
```

```
In [37]: df
```

Out[37]:

	movieId	title	genres	rating	rating^2
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240	15.376120
1	2	My new movie	Comedy	5.000000	25.000000
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040	9.929056
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861393	8.187572
4	5	Father of the Bride Part II (1995)	Comedy	3.064592	9.391722
5	6	Heat (1995)	Action|Crime|Thriller	3.834930	14.706691
6	7	Sabrina (1995)	Comedy|Romance	3.366484	11.333215
7	8	Tom and Huck (1995)	Adventure|Children	3.142049	9.872475

Delete a Row

```
In [32]: df.drop(df.index[[26744]])
```

26737	131250	No More School (2000)	Comedy	4.000000	16.000000
26738	131252	Forklift Driver Klaus: The First Day on the Jo...	Comedy|Horror	4.000000	16.000000
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.000000	16.000000
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.000000	16.000000
26741	131258	The Pirates (2014)	Adventure	2.500000	6.250000
26742	131260	Rentun Ruusu (2001)	(no genres listed)	3.000000	9.000000
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.000000	16.000000

26744 rows × 5 columns

Delete a Column

```
In [ ]: del df['rating^2']
```

```
In [42]: df
```

Out[42]:

	movieId	title	genres	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.211977
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861393
4	5	Father of the Bride Part II (1995)	Comedy	3.064592
5	6	Heat (1995)	Action|Crime|Thriller	3.834930
6	7	Sabrina (1995)	Comedy|Romance	3.366484
7	8	Tom and Huck (1995)	Adventure|Children	3.142049
8	9	Sudden Death (1995)	Action	3.004924

Group By and Aggregate

```
df.groupby('year')['rating'].agg(['mean'])
#or simply
#df.groupby('year')['rating'].mean()
```

year	mean
1891	3.000000
1893	3.375000
1894	3.071429
1895	3.125000
1896	3.183036
1898	3.850000
1899	3.625000
1900	3.166667
1901	5.000000

```
df.groupby('year')['rating'].agg(['mean']).reset_index()
#or simply
#df.groupby('year')['rating'].mean()
```

	year	mean
0	1891	3.000000
1	1893	3.375000
2	1894	3.071429
3	1895	3.125000
4	1896	3.183036
5	1898	3.850000
6	1899	3.625000
7	1900	3.166667
8	1901	5.000000

Transform

```
In [96]: df['y_average']=df.groupby('year')['rating'].transform('mean')
```

```
In [97]: df
```

:

	movieId	title	genres	rating	main_genre	year	y_average
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240	Adventure	1995	3.111147
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.211977	Adventure	1995	3.111147
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151040	Comedy	1995	3.111147
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861393	Comedy	1995	3.111147
4	5	Father of the Bride Part II (1995)	Comedy	3.064592	Comedy	1995	3.111147
5	6	Heat (1995)	Action|Crime|Thriller	3.834930	Action	1995	3.111147

Ranges (pd. cut)

```
In [100]: df['rating_range']=pd.cut(df['rating'], [0,3.5,5])
```

```
In [102]: df.groupby('rating_range').agg('count')
```

Out[102]:

rating_range	movieId	title	genres	rating	main_genre	year	y_average
(0.0, 3.5]	19285	19285	19285	19285	19285	19285	19285
(3.5, 5.0]	7437	7437	7437	7437	7437	7437	7437

Merge

```
yearly_values= df.groupby('year')['rating']\
        .agg(['sum','mean','std']).reset_index()

yearly_values
```

	year	sum	mean	std
0	1891	3.000000	3.000000	NaN
1	1893	3.375000	3.375000	NaN
2	1894	6.142857	3.071429	0.505076
3	1895	6.250000	3.125000	1.237437
4	1896	6.366071	3.183036	0.359867
5	1898	19.250000	3.850000	1.140175
6	1899	3.625000	3.625000	NaN
7	1900	3.166667	3.166667	NaN
8	1901	5.000000	5.000000	NaN

```
In [109]: df.merge(yearly_values, on='year')
```

title	genres	rating	main_genre	year	sum	mean	std
Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921240	Adventure	1995	1468.461185	3.111147	0.590844
hanji (1995)	Adventure|Children|Fantasy	3.211977	Adventure	1995	1468.461185	3.111147	0.590844
umpier Old Men (1995)	Comedy|Romance	3.151040	Comedy	1995	1468.461185	3.111147	0.590844
Waiting to hale (1995)	Comedy|Drama|Romance	2.861393	Comedy	1995	1468.461185	3.111147	0.590844
ather of the Bride Part II (1995)	Comedy	3.064592	Comedy	1995	1468.461185	3.111147	0.590844
Heat (1995)	Action|Crime|Thriller	3.834930	Action	1995	1468.461185	3.111147	0.590844
rina (1995)	Comedy|Romance	3.366484	Comedy	1995	1468.461185	3.111147	0.590844
n and Huck (1995)	Adventure|Children	3.142049	Adventure	1995	1468.461185	3.111147	0.590844

```
In [ ]:
```

Pivot Tables

In [103]:
```
## PIVOT TABLES
df.pivot_table('rating',index='year', columns="main_genre")
```

Out[103]:

main_genre	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary
year								
1891	3.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1893	3.375000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1894	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.714286
1895	NaN	NaN	NaN	NaN	NaN	2.250000	NaN	4.000000
1896	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.437500
1898	4.333333	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1899	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1900	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1901	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.000000
1902	NaN	3.738189	NaN	NaN	NaN	NaN	NaN	NaN
1903	NaN	NaN	NaN	NaN	NaN	NaN	3.340909	NaN
1905	NaN	NaN	NaN	NaN	NaN	2.500000	NaN	NaN
1909	NaN	NaN	NaN	NaN	NaN	2.750000	3.000000	2.500000
1910	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Pivot table (fill NA values)

In [110]:
```
## PIVOT TABLES
df.pivot_table('rating',index='year', columns="main_genre", fill_value=0)
```

Out[110]:

main_genre	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary
year								
1891	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1893	3.375000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1894	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.714286
1895	0.000000	0.000000	0.000000	0.000000	0.000000	2.250000	0.000000	4.000000
1896	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.437500
1898	4.333333	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1899	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1900	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1901	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000