Data Manipulation
with
Pandas

Introduction

- Pandas: Python Data Analysis Library
- Provides rich set of functions to process various types of data.
- Provides flexible data manipulation techniques as spreadsheets and relational databases.
- An open source, providing high-performance, easy-to-use data structures and data analysis tools
- Built on the top of Numpy.
- Integrates well with matplotlib library, which makes it very handy tool for analyzing the data.
- Part of the SciPy ecosystem (Scientific Computing Tools for Python)

Data structures

- Series : One-dimensional ndarray with axis labels (including time series).
- DataFrame: Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels.
- The primary pandas data structure

Series

- The Series is a one-dimensional array that can store various data types, including mix data types.
- The row labels in a Series are called the index.
- Any list, tuple and dictionary can be converted in to Series using 'series' method
- Like ndarrays, the length of a Series cannot be modified after definition.
- Missing data: Represented as NaN (np.nan, a float!).
- Statistical methods from ndarray have been overridden to automatically exclude missing data.

Creating a Series

```
import pandas as pd
s = pd.Series([9.1, 7.5, 8.63], index=['Vishnu', 'Akash', 'Aditya'], name='CGPA')
print(s)
```

```
Vishnu    9.10
Akash     7.50
Aditya    8.63
Name: CGPA, dtype: float64
```

Some attributes

```
import pandas as pd
s = pd.Series([9.1, 7.5, 8.63], index=['Vishnu', 'Akash', 'Aditya'], name='CGPA')
print(s.dtype)
print(s.name)
print(s.index)
```

```
float64
CGPA
Index(['Vishnu', 'Akash', 'Aditya'], dtype='object')
```

Creating a Series from List, Tuple, dictionary

```
import pandas as pd
h = ('Ram', '15-08-2010', 48, 3.2)
s = pd.Series(h)
print(s)
d = {'Name' : 'Ram', 'DoB' : '15-08-2010', 'Height' : 48, 'Weight' : 3.2}
ds = pd.Series(d)
print(ds)
f = ['Ram', '15-08-2020', 48, 3.2]
f = pd.Series(f, index = ['Name', 'DoB', 'Height', 'Weight'])
print(f)
```

```
0          Ram
1   15-08-2010
2           48
3          3.2
dtype: object
name            Ram
DoB      15-08-2010
Height           48
Weight          3.2
dtype: object
name            Ram
DoB      15-08-2020
Height           48
Weight          3.2
dtype: object
```

Accessing data

```
import pandas as pd
s = pd.Series([9.1, 7.5, 8.63], index=['Vishnu',
'Akash', 'Aditya'], name='CGPA')
print(s['Vishnu'])
print(s['Akash':'Aditya'])
print(s['Akash':])
```

```
9.1
Akash     7.50
Aditya    8.63
Name: CGPA, dtype: float64
Akash     7.50
Aditya    8.63
Name: CGPA, dtype: float64
```

Creating a View

```
import pandas as pd
s = pd.Series([9.1, 7.5, 8.63], index=['Vishnu', 'Akash', 'Aditya'], name='CGPA')
t=s['Akash':]
print(t)
t['Aditya']=9.5
print(s)
```

```
Akash     7.50
Aditya    8.63
Name: CGPA, dtype: float64
Vishnu    9.1
Akash     7.5
Aditya    9.5
Name: CGPA, dtype: float64
```

Adding two series (with automatic data alignment)

```
import pandas as pd
s = pd.Series([9.1, 7.5, 8.63], index=['Aditya', 'Bibek', 'Satya'], name='CGPA')
t = pd.Series([6, 6.3], index=['Bibek', 'Satya'], name='Height')
u=s.add(t)
print(u)
v=s.add(t, fill_value=0)
print(v)
```

```
Aditya     NaN
Bibek    13.50
Satya    14.93
dtype: float64
Aditya    9.10
Bibek    13.50
Satya    14.93
dtype: float64
```

DataFrame

- DataFrame can be used with two dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes
- DataFrame has two different index i.e. column-index and row-index.
- Columns can have different dtypes and can be added and removed,
- The most common way to create a DataFrame is by using the dictionary of equal-length list.
- Further, all the spreadsheets and text files are read as DataFrame.

Creating a DataFrame

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'Height': [5.5, 6, 6.5], 'Weight': [np.nan,
230., 275.]},
index=['Aditya', 'Bivek', 'Vishnu'])
print(df)
print(df.dtypes)
```

```
        Height  Weight
Aditya     5.5     NaN
Bivek      6.0   230.0
Vishnu     6.5   275.0
```

```
Height    float64
Weight    float64
dtype: object
```

Other attributes

```
print(df.shape)
print(df.size)
print(df.columns)
print(df.index)
```

```
(3, 2)
6
Index(['Height', 'Weight'], dtype='object')
Index(['Aditya', 'Bivek', 'Vishnu'], dtype='object')
```

DataFrame is by using the dictionary

```
data = { 'name' : ['AA', 'IBM', 'GOOG'],
'date' : ['2001-12-01', '2012-02-10',
'2010-04-09'], 'shares' : [100, 30, 90],
'price' : [12.3, 10.3, 32.2]}
df = pd.DataFrame(data)
print(df)
df['owner'] = 'Unknown'
print(df)
```

```
   name        date  shares  price
0    AA  2001-12-01     100   12.3
1   IBM  2012-02-10      30   10.3
2  GOOG  2010-04-09      90   32.2
   name        date  shares  price    owner
0    AA  2001-12-01     100   12.3  Unknown
1   IBM  2012-02-10      30   10.3  Unknown
2  GOOG  2010-04-09      90   32.2  Unknown
```

```python
df.index = ['one', 'two', 'three']
print(df)
df = df.set_index('name', drop=False)
print(df)
```

```
        name        date  shares  price   owner
one      AA  2001-12-01     100   12.3  Unknown
two     IBM  2012-02-10      30   10.3  Unknown
three  GOOG  2010-04-09      90   32.2  Unknown
            date  shares  price   owner
name
AA    2001-12-01     100   12.3  Unknown
IBM   2012-02-10      30   10.3  Unknown
GOOG  2010-04-09      90   32.2  Unknown
```

Accessing data

```
print(df['shares'])
print(df.loc['AA',:])
print(df.loc[:, 'name'])
print(df.loc['AA', 'shares'])
```

```
name
AA      100
IBM      30
GOOG     90
Name: shares, dtype: int64
name             AA
date     2001-12-01
shares          100
price          12.3
owner       Unknown
Name: AA, dtype: object
name
AA        AA
IBM      IBM
GOOG    GOOG
Name: name, dtype: object
100
```

Deleting any Column

```
del df['owner']
print(df)
df.drop('shares', axis = 1,inplace = True)
print(df)
df.drop(['AA', 'IBM'],axis=0, inplace=True)
print(df)
```

```
 name       date  shares  price
name
AA      AA  2001-12-01     100   12.3
IBM    IBM  2012-02-10      30   10.3
GOOG  GOOG  2010-04-09      90   32.2
     name       date  price
name
AA      AA  2001-12-01   12.3
IBM    IBM  2012-02-10   10.3
GOOG  GOOG  2010-04-09   32.2
     name       date  price
name
GOOG  GOOG  2010-04-09   32.2
```

Summing over columns and rows

```
print(df.sum())
```

```
name                       AAIBMGOOG
date      2001-12-012012-02-102010-04-09
shares                           220
price                           54.8
owner          UnknownUnknownUnknown
dtype: object
```

```
print(df.sum(axis=1))
```

```
name
AA      112.3
IBM      40.3
GOOG    122.2
dtype: float64
```

Reading files

```
import pandas as pd
casts = pd.read_csv('cast.csv', index_col=None)
print(casts.head())
```

```
                  title  year      name   type                character     n
0        Closet Monster  2015  Buffy #1  actor                  Buffy 4  31.0
1       Suuri illusioni  1985    Homo $  actor                   Guests  22.0
2   Battle of the Sexes  2017   $hutter  actor          Bobby Riggs Fan  10.0
3  Secret in Their Eyes  2015   $hutter  actor          2002 Dodger Fan   NaN
4            Steve Jobs  2015   $hutter  actor  1988 Opera House Patron   NaN
```

```python
titles = pd.read_csv('titles.csv', index_col =None)
print(titles.tail())
```

```
                        title  year
49995                   Rebel  1970
49996                Suzanne  1996
49997                   Bomba  2013
49998  Aao Jao Ghar Tumhara  1984
49999             Mrs. Munck  1995
```

```
a=pd.read_csv('cast.csv', usecols= ['title','year'])
print(a.head(6))
```

```
                     title  year
0           Closet Monster  2015
1          Suuri illusioni  1985
2       Battle of the Sexes  2017
3    Secret in Their Eyes  2015
4               Steve Jobs  2015
5  Straight Outta Compton  2015
```

Row and column selection

```
t = titles['title']
print(t.head(3))
```

```
0              The Rising Son
1     The Thousand Plane Raid
2           Crucea de piatra
Name: title, dtype: object
```

Filter Data

- Data can be filtered by providing some boolean expression in DataFrame.

```
movies90 = titles[ (titles['year']>=1990) & (titles['year']<2000) ]
print(movies90.head(4))
```

```
                     title  year
0           The Rising Son  1990
2         Crucea de piatra  1993
12  Poka Makorer Ghar Bosoti  1996
19        Maa Durga Shakti  1999
```

Sorting

- In filtering operation, the data is sorted by index i.e. by default ‘sort_index’ operation is used

```
macbeth = titles[ titles['title'] == 'Macbeth'].sort_values('year')
print(macbeth.head())
```

```
         title  year
4226   Macbeth  1913
17166  Macbeth  1997
25847  Macbeth  1998
9322   Macbeth  2006
11722  Macbeth  2013
```

Null values

- ‘isnull’ command returns the true value if any row of has null values.

```
c = casts
print(c['n'].isnull().head())
```

```
0    False
1    False
2    False
3     True
4     True
Name: n, dtype: bool
```

- To display the rows with null values, the condition must be passed in the DataFrame

```
print(c[c['n'].isnull()].head(3))
```

```
                      title  year     name   type                character   n
3    Secret in Their Eyes  2015  $hutter  actor           2002 Dodger Fan NaN
4              Steve Jobs  2015  $hutter  actor  1988 Opera House Patron NaN
5  Straight Outta Compton  2015  $hutter  actor              Club Patron NaN
```

- df.isna().any() returns a boolean value for each column.
- If there is at least one missing value in that column, the result is True.

```
print(c.isna().any())
```

```
title        False
year         False
name         False
type         False
character    False
n            True
dtype: bool
```

- df.isna().sum() returns the number of missing values in each column.

```
print(c.isna().sum())
```

```
title           0
year            0
name            0
type            0
character       0
n           28966
dtype: int64
```

Handling Missing Values

- Drop missing values
- Replace missing values

Drop missing values

- We can drop a row or column with missing values using dropna() function. how parameter is used to set condition to drop.

- how=’any’ : drop if there is any missing value
- how=’all’ : drop if all values are missing

- Furthermore, using thresh parameter, we can set a threshold for missing values in order for a row/column to be dropped.

- c.dropna(axis=0, inplace=True)
- print(c.head())

```
                                             title  ...     n
0                                   Closet Monster  ...  31.0
1                                  Suuri illusioni  ...  22.0
2                              Battle of the Sexes  ...  10.0
8   Lapis, Ballpen at Diploma, a True to Life Journey  ...   9.0
10                           When the Man Went South  ...   8.0

[5 rows x 6 columns]
```

Replacing missing values

- fillna() function of Pandas conveniently handles missing values.
- Replace missing values with a scalar: c.fillna(2)
- fillna() can also be used on a particular column: c[‘n’].fillna(1)
- Using method parameter, missing values can be replaced with the values before or after them.
- c.fillna(axis=0, method=‘ffill’)

String operations

- Various string operations can be performed using ‘.str.’ option
- h=t[t['title'].str.startswith("Maa ")].head(3)
- print(h)

```
 title  year
19     Maa Durga Shakti  1999
3046      Maa Aur Mamta  1970
7470  Maa Vaibhav Laxmi  1989
```

Method	Description
`cat()`	Concatenate strings
`split()`	Split strings on delimiter
`rsplit()`	Split strings on delimiter working from the end of the string
`get()`	Index into each element (retrieve i-th element)
`join()`	Join strings in each element of the Series with passed separator
`get_dummies()`	Split strings on the delimiter returning DataFrame of dummy variables
`contains()`	Return boolean array if each string contains pattern/regex
`replace()`	Replace occurrences of pattern/regex/string with some other string or the return value of a callable given the occurrence
`repeat()`	Duplicate values (`s.str.repeat(3)` equivalent to `x * 3`)

`pad()`	Add whitespace to left, right, or both sides of strings
`center()`	Equivalent to `str.center`
`ljust()`	Equivalent to `str.ljust`
`rjust()`	Equivalent to `str.rjust`
`zfill()`	Equivalent to `str.zfill`
`wrap()`	Split long strings into lines with length less than a given width
`slice()`	Slice each string in the Series
`slice_replace()`	Replace slice in each string with passed value
`count()`	Count occurrences of pattern
`startswith()`	Equivalent to `str.startswith(pat)` for each element
`endswith()`	Equivalent to `str.endswith(pat)` for each element
`findall()`	Compute list of all occurrences of pattern/regex for each string

`findall()`	Compute list of all occurrences of pattern/regex for each string
`match()`	Call `re.match` on each element, returning matched groups as list
`extract()`	Call `re.search` on each element, returning DataFrame with one row for each element and one column for each regex capture group
`extractall()`	Call `re.findall` on each element, returning DataFrame with one row for each match and one column for each regex capture group
`len()`	Compute string lengths
`strip()`	Equivalent to `str.strip`
`rstrip()`	Equivalent to `str.rstrip`
`lstrip()`	Equivalent to `str.lstrip`
`partition()`	Equivalent to `str.partition`
`rpartition()`	Equivalent to `str.rpartition`

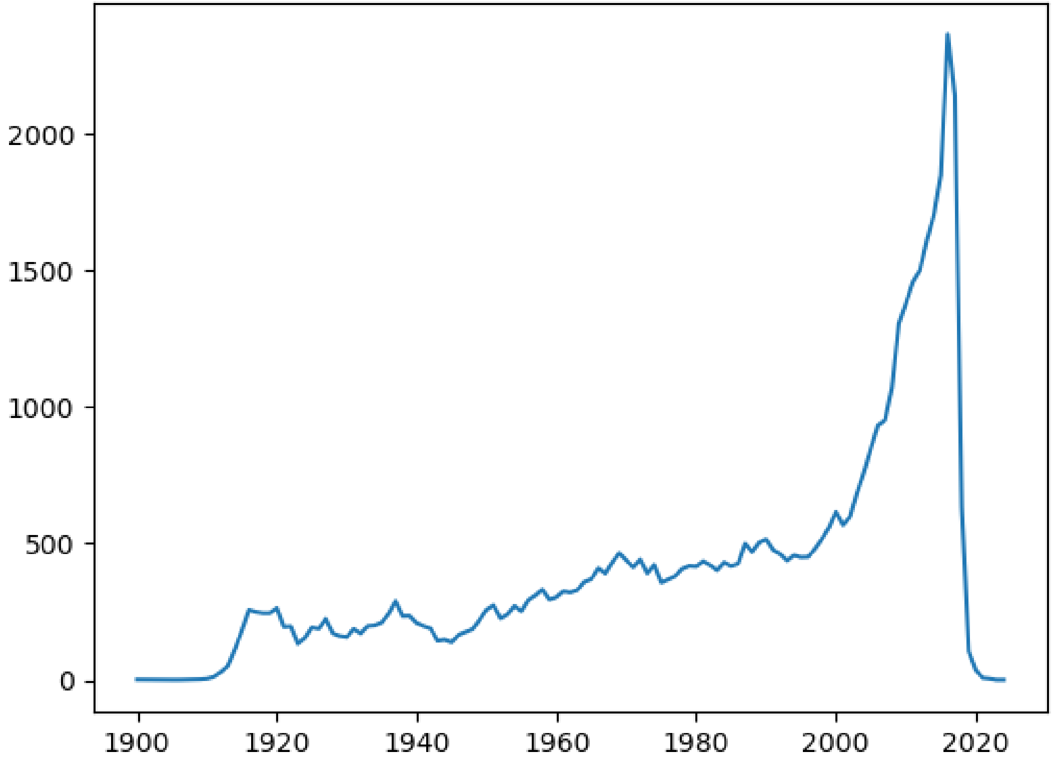
`casefold()`	Equivalent to `str.casefold`
`upper()`	Equivalent to `str.upper`
`find()`	Equivalent to `str.find`
`rfind()`	Equivalent to `str.rfind`
`index()`	Equivalent to `str.index`
`rindex()`	Equivalent to `str.rindex`
`capitalize()`	Equivalent to `str.capitalize`
`swapcase()`	Equivalent to `str.swapcase`
`normalize()`	Return Unicode normal form. Equivalent to `unicodedata.normalize`
`translate()`	Equivalent to `str.translate`
`isalnum()`	Equivalent to `str.isalnum`
`isalpha()`	Equivalent to `str.isalpha`

`isalpha()`	Equivalent to `str.isalpha`
`isdigit()`	Equivalent to `str.isdigit`
`isspace()`	Equivalent to `str.isspace`
`islower()`	Equivalent to `str.islower`
`isupper()`	Equivalent to `str.isupper`
`istitle()`	Equivalent to `str.istitle`
`isnumeric()`	Equivalent to `str.isnumeric`
`isdecimal()`	Equivalent to `str.isdecimal`

- Total number of occurrences can be counted using ‘value_counts()’ option.
- In following code, total number of movies are displayed base on years.
- t['year'].value_counts().head()

Plots

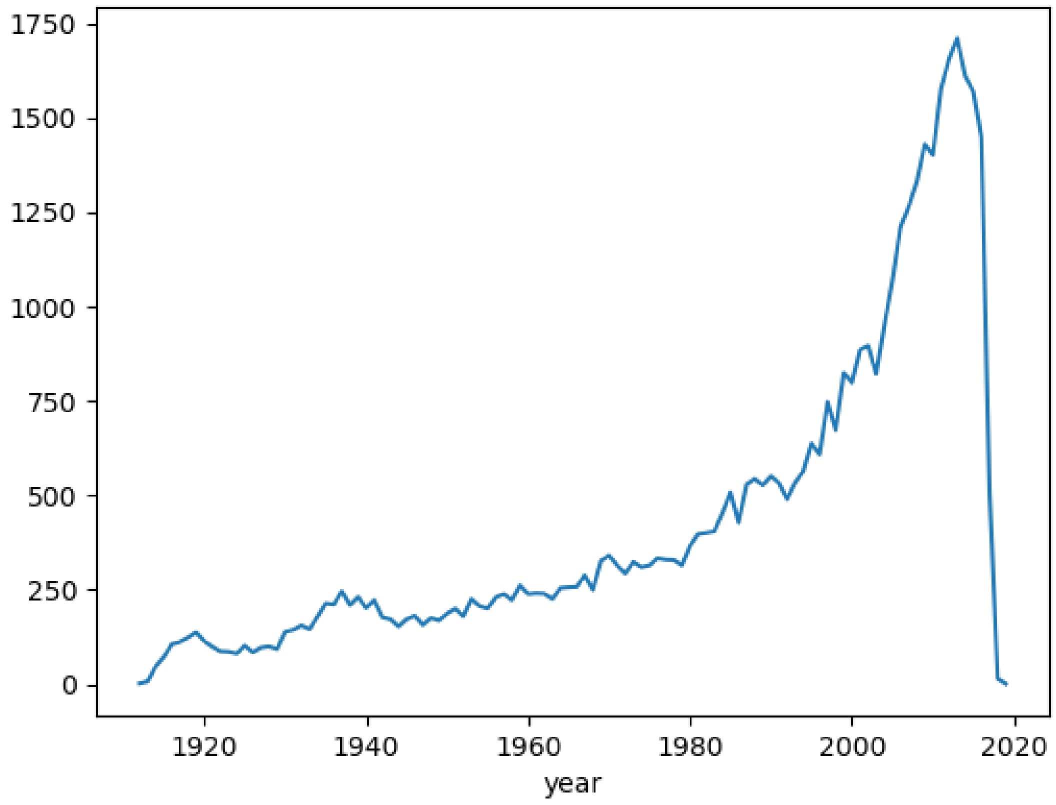
- import matplotlib.pyplot as plt
- t = titles
- p = t['year'].value_counts()
- p.sort_index().plot()
- p.show()



Grouping

- Groupby with column-names

- c = casts
- cg = c.groupby(['year']).size()
- cg.plot()
- plt.show()



groupby option can take multiple parameters for grouping

- c = casts
- cf = c[c['name'] == 'Aaron Abrams']
- ct=cf.groupby(['year', 'title']).size().head()
- print(ct)

```
year  title
2003  The In-Laws                 1
2004  Resident Evil: Apocalypse    1
      Siblings                    1
2005  Cinderella Man              1
      Sabah                       1
dtype: int64
```

- grouping based on maximum ratings in a year;
- c.groupby(['year']).n.max().head()
- To check the mean rating each year,
- c.groupby(['year']).n.mean().head()

Unstack

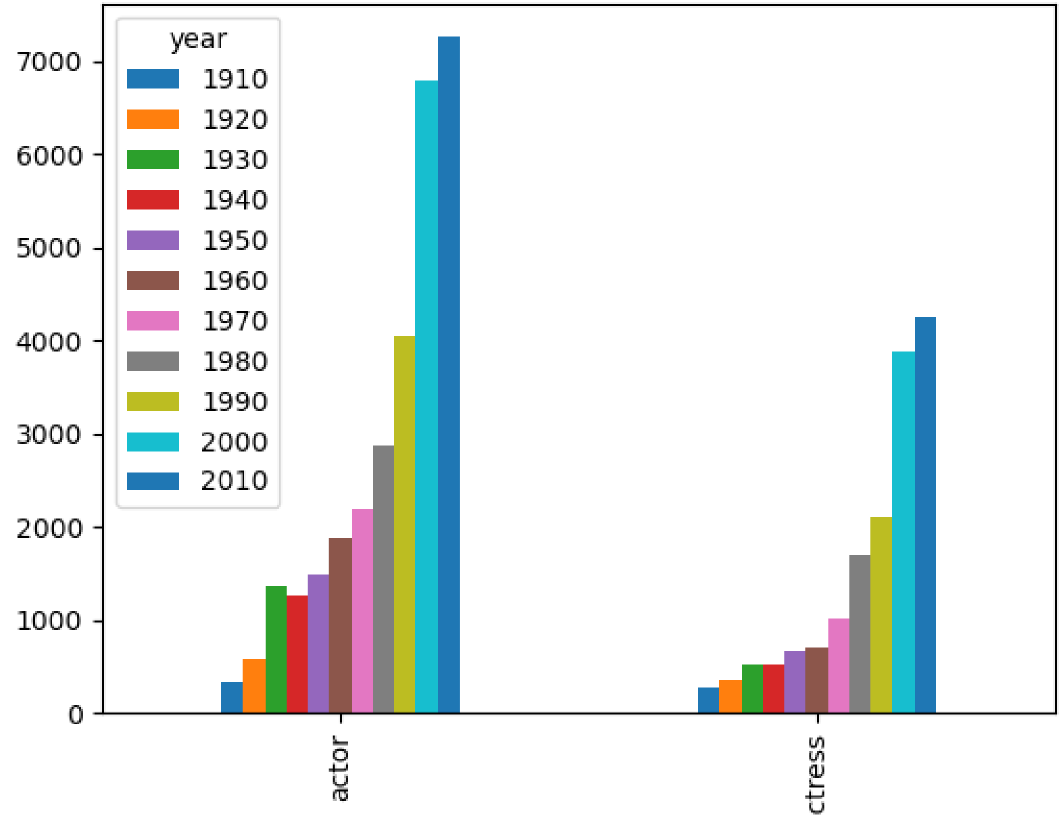
- we want to compare and plot the total number of actors and actresses in each decade.
- we need to group the data based on ‘type’
- c = casts
- c_decade = c.groupby(['type', c['year']//10*10]).size()
- print(c_decade)

```
type     year
actor    1910     340
         1920     590
         1930    1364
         1940    1253
         1950    1490
         1960    1879
         1970    2191
         1980    2874
         1990    4051
         2000    6787
         2010    7259
actress  1910     267
         1920     353
         1930     511
         1940     528
         1950     665
         1960     702
         1970    1015
         1980    1686
         1990    2115
         2000    3872
         2010    4243
dtype: int64
```

- us=c_decade.unstack()
- print(us)

```
year    1910  1920  1930  1940  1950  1960  1970  1980  1990
2000  2010
type
actor    340   590  1364  1253  1490  1879  2191  2874  4051
6787  7259
actress   267   353   511   528   665   702  1015  1686  2115
3872  4243
```

- us.plot(kind='bar')
- plt.show()



Time series

- A series of time can be generated using ‘date_range’ command.
- ‘periods’ is the total number of samples;
- freq = ‘M’ represents that series must be generated based on ‘Month’.
- By default, pandas consider ‘M’ as end of the month.
- Use ‘MS’ for start of the month.

- rng = pd.date_range('2011-03-01 10:15', periods = 10, freq = 'M')
- print(rng)

```
DatetimeIndex(['2011-03-31 10:15:00', '2011-04-30 10:15:00',
               '2011-05-31 10:15:00', '2011-06-30 10:15:00',
               '2011-07-31 10:15:00', '2011-08-31 10:15:00',
               '2011-09-30 10:15:00', '2011-10-31 10:15:00',
               '2011-11-30 10:15:00', '2011-12-31 10:15:00'],
              dtype='datetime64[ns]', freq='M')
```

Support for time zone representation, converting to another time zone, and converting between time span representations.

Categoricals

- Similar to categorical variables used in statistics.
- Practical for saving memory and sorting data.
- “Examples are gender, social class, blood type, country affiliation”
- s = pd.Series(["a","b","c","a"], dtype="category")
- print(s)

dtype: category
Categories (3, object): [a, b, c]

References

- https://pandas.pydata.org/pandas-docs/stable/user_guide/
- GitHub awesome-pandas
- Pandas Guide by Meher Krishna Patel
- Manipulating and analyzing data with pandas by Céline Comte