Co-funded by the
Erasmus+ Programme
of the European Union

DeCAIR

Plotting and Visualization

Prof. Gheith Abandah

Reference

- **Chapter 9: Plotting and Visualization**

- Wes McKinney, **Python for Data Analysis**: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media, 2nd Edition, 2018.
  - Material: https://github.com/wesm/pypop-book

Plotting and Visualization

- **Making informative visualizations** is one of the most important tasks in data analysis.

- It may be a part of the **exploratory process**: to help identify outliers or needed data transformations, or as a way of generating ideas for models.

Outline

9.1 A Brief **matplotlib** API Primer

- Figures and Subplots
- Colors, Markers, and Line Styles
- Ticks and Labels
- Saving Plots to File
- matplotlib Configuration

9.2 Plotting with **pandas** and **seaborn**

- Line Plots
- Bar Plots
- Histograms and Density Plots
- Scatter or Point Plots
- Facet Grids and Categorical Data

Matplotlib: MATLAB-style Scientific Visualization

- Matplotlib is a Python **plotting library** which produces publication **quality figures** in a variety of hardcopy formats.

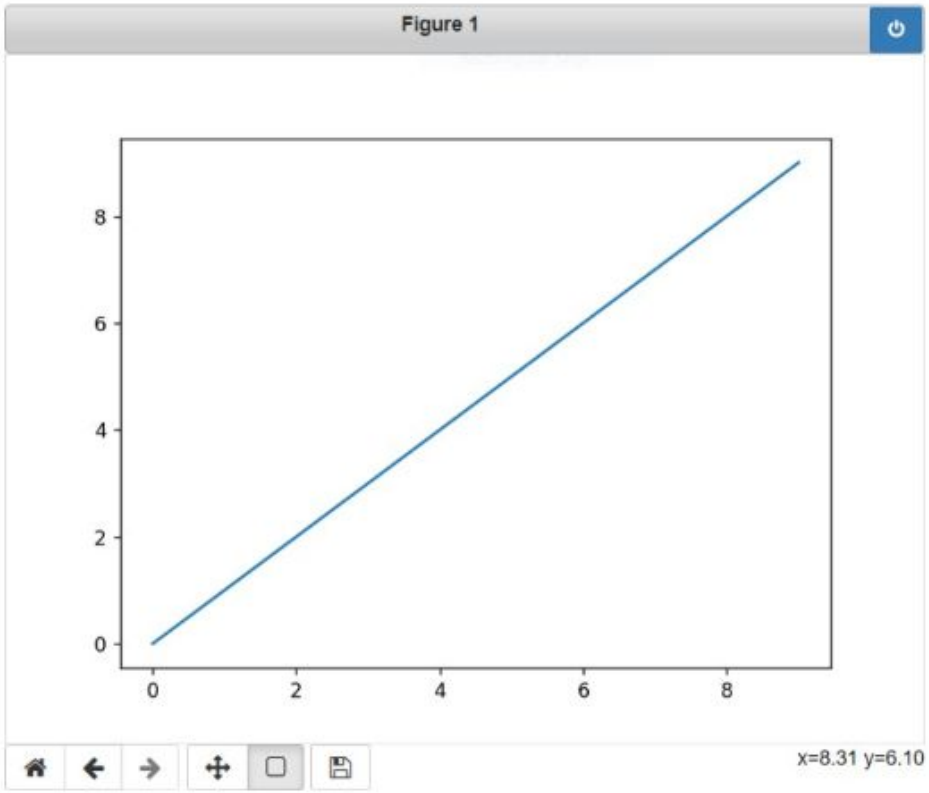
- **Website**: https://matplotlib.org/
- Also, check the **tutorial** package website: https://matplotlib.org/tutorials/introductory/pyplot.html

9.1 A Brief matplotlib API Primer

- To set up Jupyter Notebook, run **%matplotlib notebook** (**%matplotlib** in IPython).

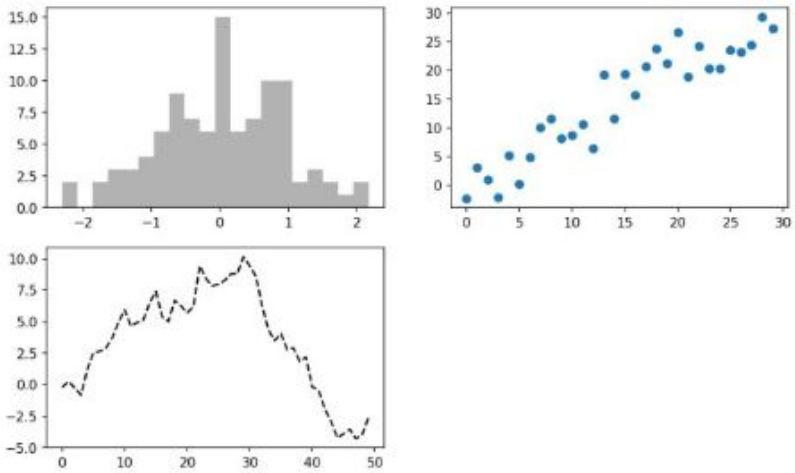
- Simple line plot:

```
import matplotlib.pyplot as plt
import numpy as np
data = np.arange(10)
plt.plot(data)
```



Figures and Subplots

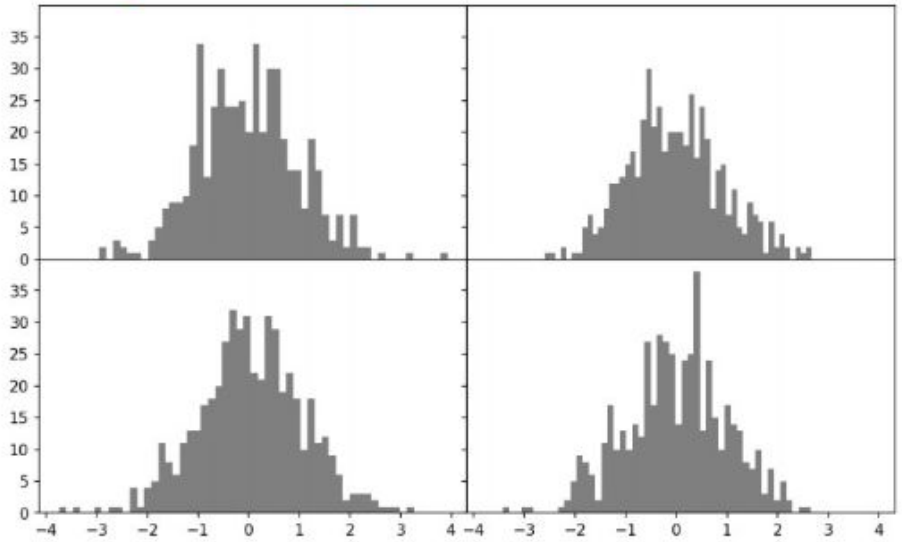
- Plots in matplotlib reside within a **Figure** object.
- You must create one or more **subplots** inside a blank figure.



```
fig = plt.figure(figsize=(4, 3))
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
plt.plot(np.random.randn(50).cumsum
    (), 'k--')
_ = ax1.hist(np.random.randn(100),
    bins=20, color='k', alpha=0.3)
ax2.scatter(np.arange(30),
    np.arange(30) + 3 *
    np.random.randn(30))
```

Figures and Subplots

- Better to create a new figure and return a NumPy **array** containing the created **subplot objects**.

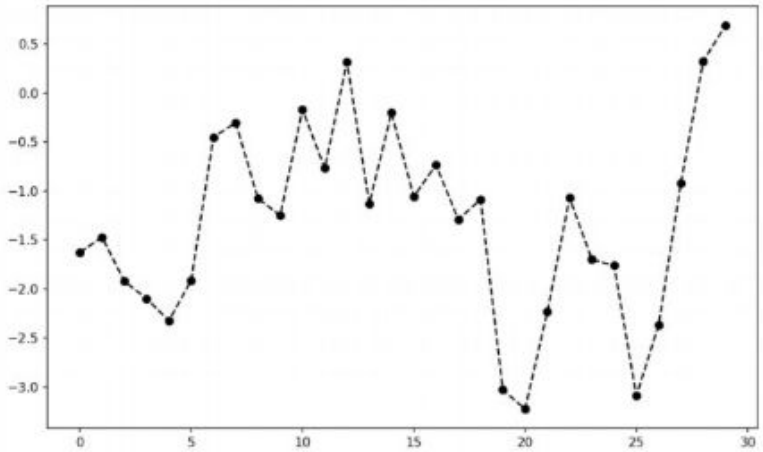


```
fig, axes = plt.subplots(2, 2,
    sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(
            np.random.randn(500),
            bins=50, color='k',
            alpha=0.5)
plt.subplots_adjust(wspace=0,
  hspace=0)
```

Colors, Markers, and Line Styles

- Specify color and style:
  - **String**
  - **Explicitly**
- Check options using `plt.plot?`

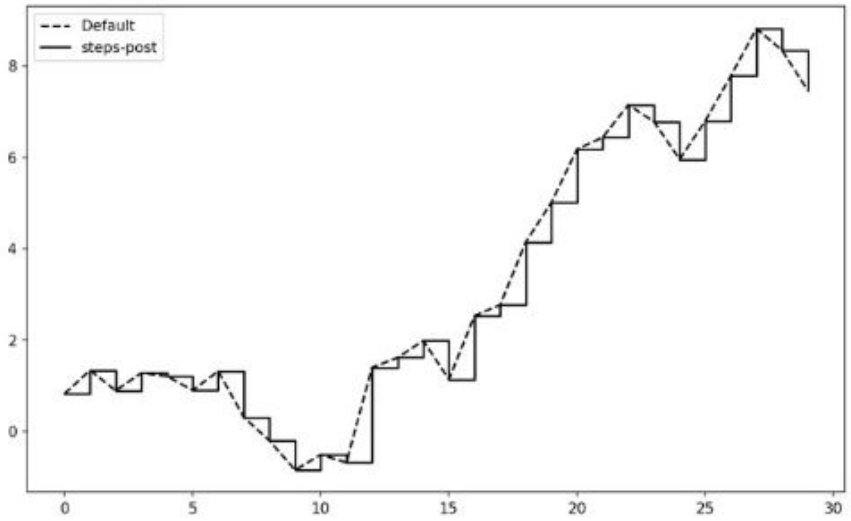
```
from numpy.random import randn
plt.plot(randn(30).cumsum(),
    'ko--')
plt.plot(randn(30).cumsum(),
    color='k', marker='o',
    linestyle='dashed')
```



Colors, Markers, and Line Styles

- You can specify the draw style:
  - **Linearly interpolated**
  - **Step**
  - **etc.**
- You can draw a **legend**.

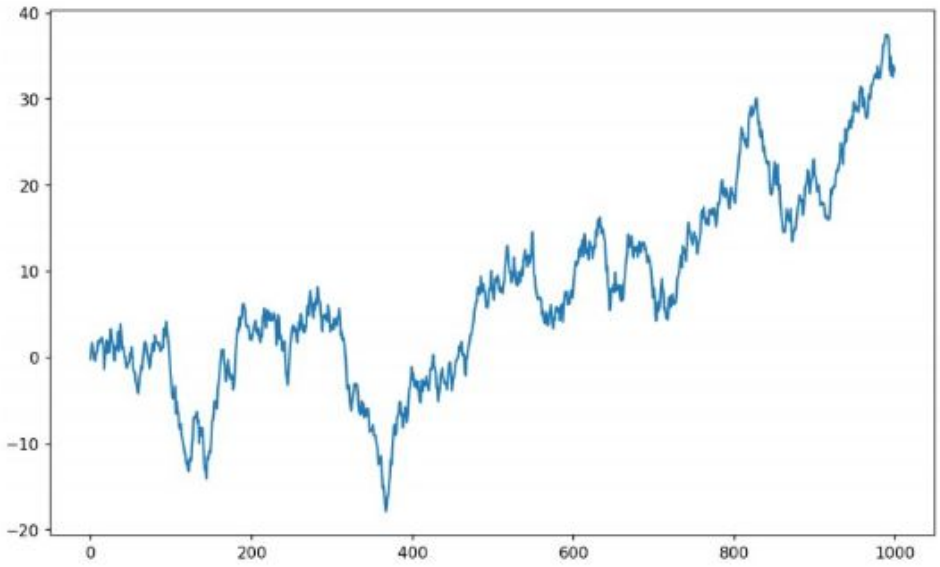
```
data = np.random.randn(30).cumsum()
plt.plot(data, 'k--',
    label='Default')
plt.plot(data, 'k-',
    drawstyle='steps-post',
    label='steps-post')
plt.legend(loc='best')
```



Ticks and Labels

- For plot decoration:
  - **Procedural** pyplot interface
  - **Object-oriented** interface

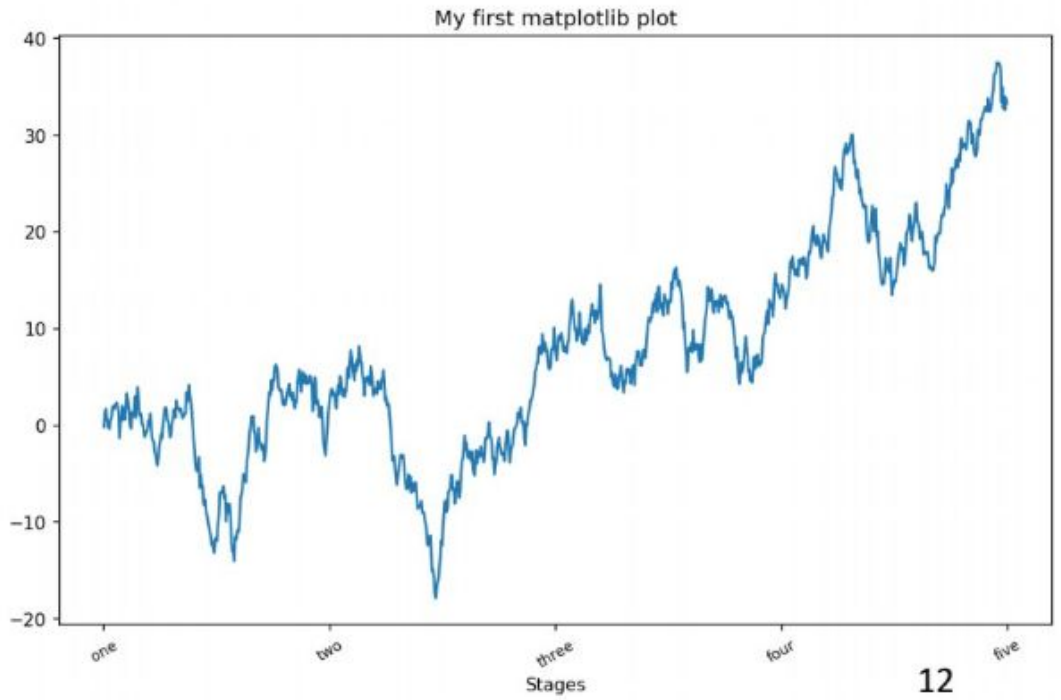
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(
                1000).cumsum())
```



Ticks and Labels

```
ticks = ax.set_xticks([0, 250, 500,
    750, 1000])
labels = ax.set_xticklabels(['one',
    'two', 'three', 'four',
    'five'], rotation=30,
    fontsize='small')
ax.set_title(
   'My first matplotlib plot')
ax.set_xlabel('Stages')
# Or:
props = {
   'title': 'My first plot',
   'xlabel': 'Stages'
}
ax.set(**props)
```

- Check matplotlib.axes



Saving Plots to File

- When saving to a file, the **file extension** specifies the **image format**.

```
plt.savefig('figpath.png', dpi=400,
            bbox_inches='tight')
```

Argument	Description
`fname`	String containing a filepath or a Python file-like object. The figure format is inferred from the file extension (e.g., `.pdf` for PDF or `.png` for PNG)
`dpi`	The figure resolution in dots per inch; defaults to 100 out of the box but can be configured
`facecolor`, `edgecolor`	The color of the figure background outside of the subplots; `'w'` (white), by default
`format`	The explicit file format to use (`'png'`, `'pdf'`, `'svg'`, `'ps'`, `'eps'`, ...)
`bbox_inches`	The portion of the figure to save; if `'tight'` is passed, will attempt to trim the empty space around the figure

matplotlib Configuration

- matplotlib comes configured with color schemes and defaults geared for publication.
- Can be customized:
  - Programmatically using the **rc** method
  - Configuration: **matplotlib/mpl-data/matplotlibrc**. Customize in your home directory as **.matplotlibrc**

```
plt.rc('figure', figsize=(10, 10))
font_options = {
    'family' : 'monospace',
    'weight' : 'bold',
    'size' : 'small'}
plt.rc('font', **font_options)
```

Outline

9.1 A Brief **matplotlib** API Primer

- Figures and Subplots
- Colors, Markers, and Line Styles
- Ticks and Labels
- Saving Plots to File
- matplotlib Configuration

9.2 Plotting with **pandas** and **seaborn**

- Line Plots
- Bar Plots
- Histograms and Density Plots
- Scatter or Point Plots
- Facet Grids and Categorical Data

9.2 Plotting with pandas and seaborn

- YouTube Video from **Kimberly Fessel**

Introduction to Seaborn | How seaborn Python works with matplotlib along with seaborn and pandas

https://youtu.be/vaf4ir8eT38

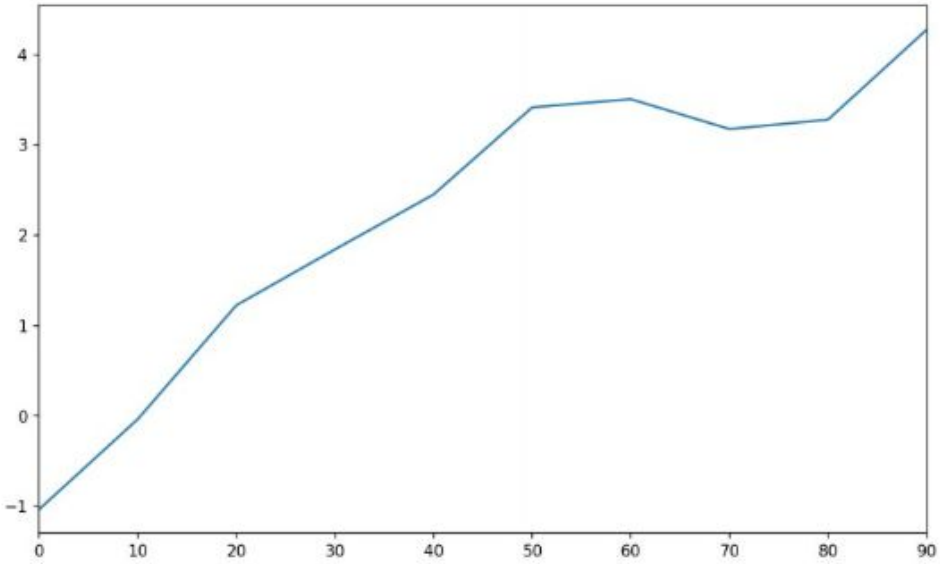
9.2 Plotting with pandas and seaborn

- **matplotlib is low-level tool**; you assemble a plot from its base components.
- Productive plotting is available through:
  - **pandas**
  - **seaborn** (https://seaborn.pydata.org/)
- Importing **seaborn** modifies the default matplotlib color schemes.

Line Plots

- Series and DataFrame have **plot** for making some basic plot types.

```
s = pd.Series(randn(10).cumsum(),
        index=np.arange(0, 100, 10))
s.plot()
```



Options:
`ax, use_index=False, xticks, xlim, yticks, ylim`

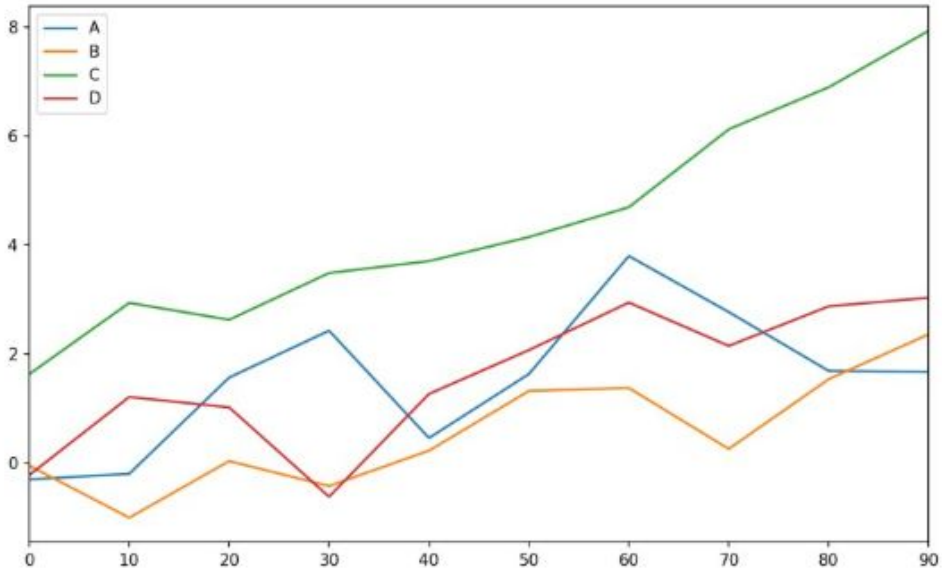
Series.plot method arguments

Argument	Description
`label`	Label for plot legend
`ax`	matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot
`style`	Style string, like `'ko--'`, to be passed to matplotlib
`alpha`	The plot fill opacity (from 0 to 1)
`kind`	Can be `'area'`, `'bar'`, `'barh'`, `'density'`, `'hist'`, `'kde'`, `'line'`, `'pie'`
`logy`	Use logarithmic scaling on the y-axis
`use_index`	Use the object index for tick labels
`rot`	Rotation of tick labels (0 through 360)
`xticks`	Values to use for x-axis ticks
`yticks`	Values to use for y-axis ticks
`xlim`	x-axis limits (e.g., `[0, 10]`)
`ylim`	y-axis limits
`grid`	Display axis grid (on by default)

Line Plots

- DataFrame plots each of its **columns** as a **different line** on the same subplot, creating a **legend** automatically.

```
df = pd.DataFrame(randn(10,
        4).cumsum(0),
        columns=['A', 'B', 'C', 'D'],
        index=np.arange(0, 100, 10))
df.plot()
```



Equivalent to `df.plot.line()`

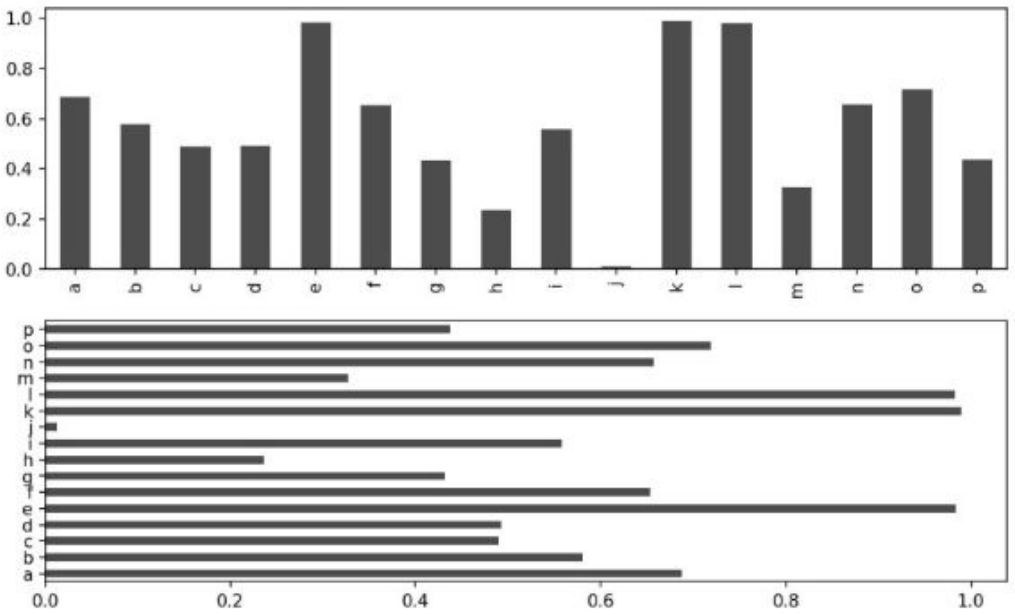
DataFrame-specific plot arguments

Argument	Description
`subplots`	Plot each DataFrame column in a separate subplot
`sharex`	If `subplots=True`, share the same x-axis, linking ticks and limits
`sharey`	If `subplots=True`, share the same y-axis
`figsize`	Size of figure to create as tuple
`title`	Plot title as string
`legend`	Add a subplot legend (`True` by default)
`sort_columns`	Plot columns in alphabetical order; by default uses existing column order

Bar Plots

- The **plot.bar** and **plot.barh** make vertical and horizontal bar plots.

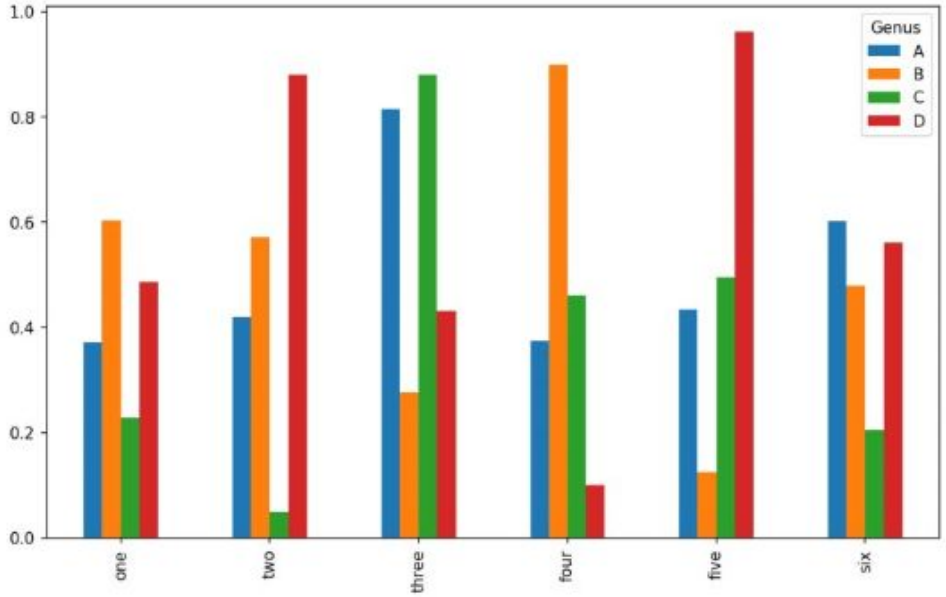
```
fig, axes = plt.subplots(2, 1)
data = pd.Series(rand(16),
    index=list('abcdefghijklmnop'))
data.plot.bar(ax=axes[0],
    color='k', alpha=0.7)
data.plot.barh(ax=axes[1],
    color='k', alpha=0.7)
```



Bar Plots

- With a DataFrame, bar plots **group** the values in **each row** together in a group in bars, side by side, for each value.

```
df.plot.bar()
```



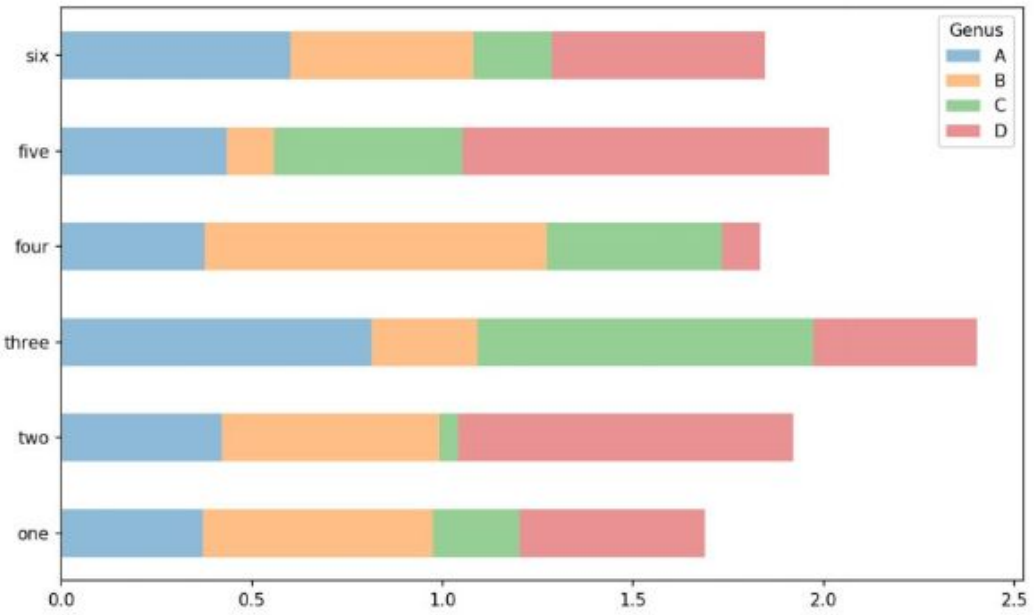

```
df
```

Genus	A	B	C	D
one	0.801554	0.094551	0.469551	0.619210
two	0.208189	0.792578	0.648303	0.260912
three	0.642697	0.847883	0.767702	0.856446
four	0.113493	0.083676	0.283905	0.023767
five	0.220087	0.573322	0.800078	0.514133
six	0.929547	0.272519	0.783754	0.007303

Bar Plots

- We create stacked bar plots from a DataFrame by passing **stacked=True**.

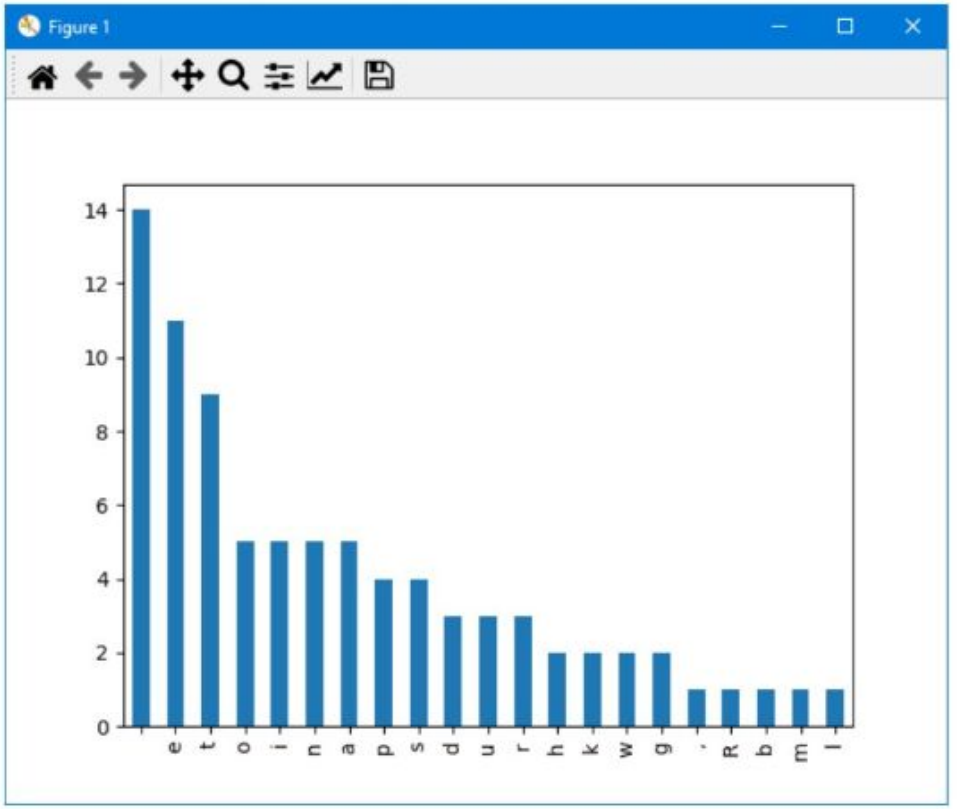
```
df.plot.barh(stacked=True,
     alpha=0.5)
```



Bar Plots

- A useful recipe for bar plots is to visualize a Series's **value frequency** `s.value_counts().plot.bar()`.

```
s = pd.Series(list('Returning to the tipping dataset used earlier in the book, suppose we wanted to make'))
s.value_counts().plot.bar()
```



Bar Plots

- **Example**: Tipping Dataset

  Make a stacked bar plot showing the **percentage** of data points for each **party size** on each **day**.

- Hint: use **crosstab**

```
tips = pd.read_csv('tips.csv')
```

بترجعلي جدول بناء على المعلومات اللي بعطيه ياها

```
party_counts = pd.crosstab(
    tips['day'], tips['size'])
```

```
party_counts
size   1   2   3   4  5  6
day
Fri    1  16   1   1  0  0
Sat    2  53  18  13  1  0
Sun    0  39  15  18  3  1
Thur   1  48   4   5  1  3
```

Bar Plots

normalization data

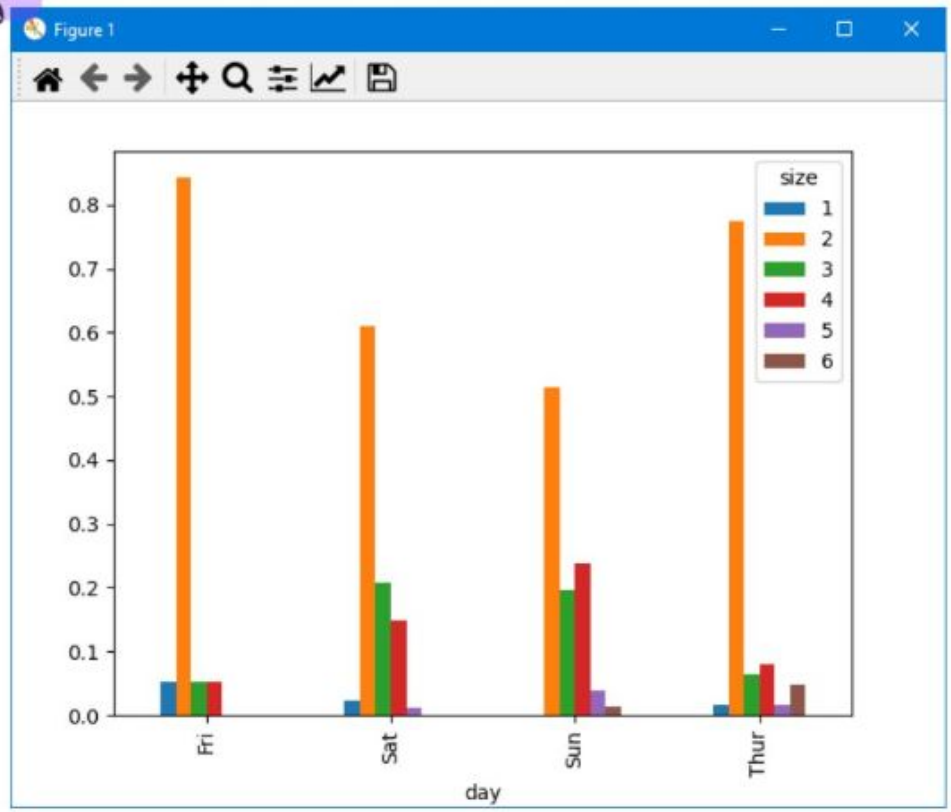
قسم كل سطر على مجموع الاعمدة اللي يهداك السطر

```
party_pcts = party_counts.div(
    party_counts.sum(1), axis=0)
```

```
party_pcts
```

size	1	2	3	4	5	6
day						
Fri	0.052632	0.842105	0.052632	0.052632	0.000000	0.000000
Sat	0.022989	0.609195	0.206897	0.149425	0.011494	0.000000
Sun	0.000000	0.513158	0.197368	0.236842	0.039474	0.013158
Thur	0.016129	0.774194	0.064516	0.080645	0.016129	0.048387

```
party_pcts.plot.bar()
```



Bar Plots (seaborn)

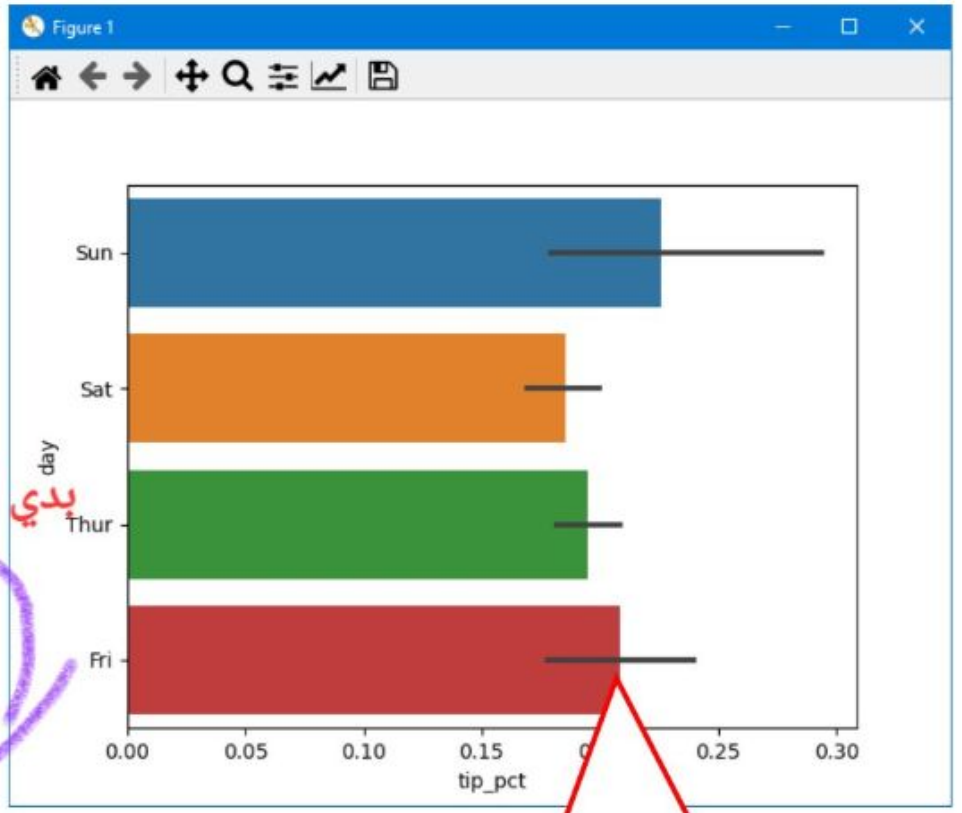
- **Example**: Use **seaborn** to visualize **tip percent**.

```
import seaborn as sns

tips['tip_pct'] = tips['tip'] /
(tips['total_bill'] - tips['tip'])
tips.head()
```

بدي اعمل عليه تحليلات

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069

```
sns.barplot(x='tip_pct', y='day',
      data=tips, orient='h')
```



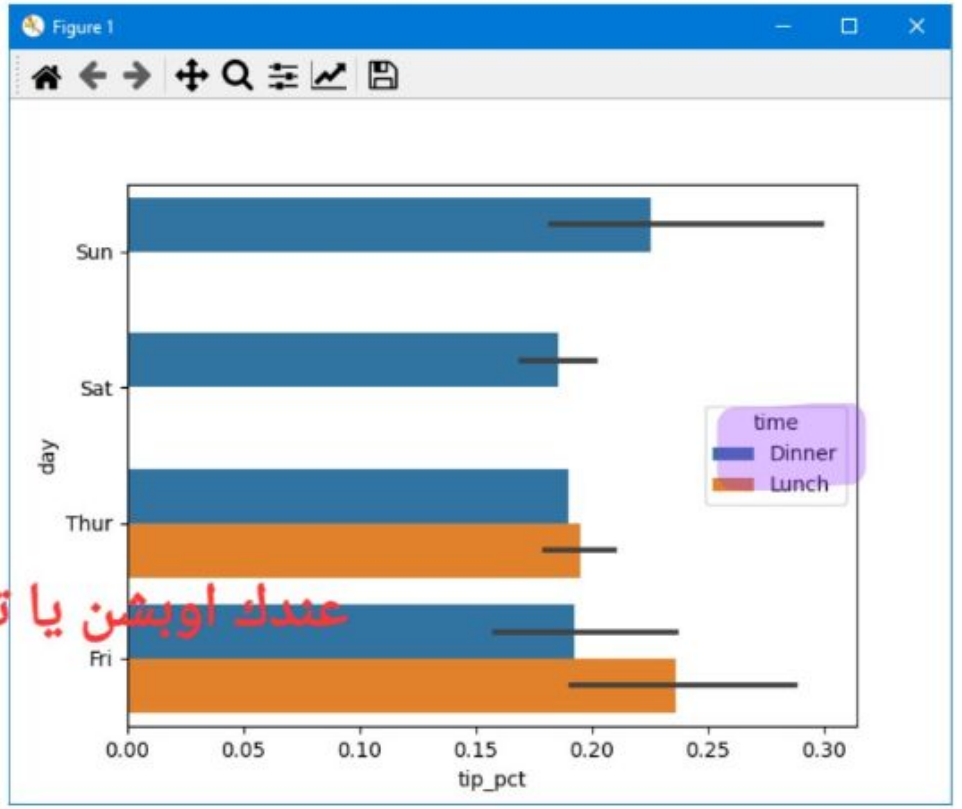
Averages with 95% confidence interval

Bar Plots (seaborn)

- **seaborn.barplot** has a **hue** option that enables us to split by an additional categorical value.

```
sns.barplot(x='tip_pct', y='day',
      hue='time', data=tips,
      orient='h')
```

عندك اوبشن يا تشتغل يوم احد وقت العشاء او يوم جمعة وقت



- [ ] import seaborn is important
- [ ] arange(0 ,100,10) ==[0,90]
- [ ] x-axis –index , y-axis–values
- [ ] rand(10,4):

اربع اعمده كل عامود من ١٠ اسطر

- [ ] cumsum(0): الجمع باتجاه الاسطر
- [ ] plot.barh h: horizontal
- [ ] fig: means object of the whole figure
- [ ] axes: array including number of subplots (object of subplots)
- [ ] series(list(Text)) : use value_counts().plot.bar() To find value frequency of each letter
- [ ]

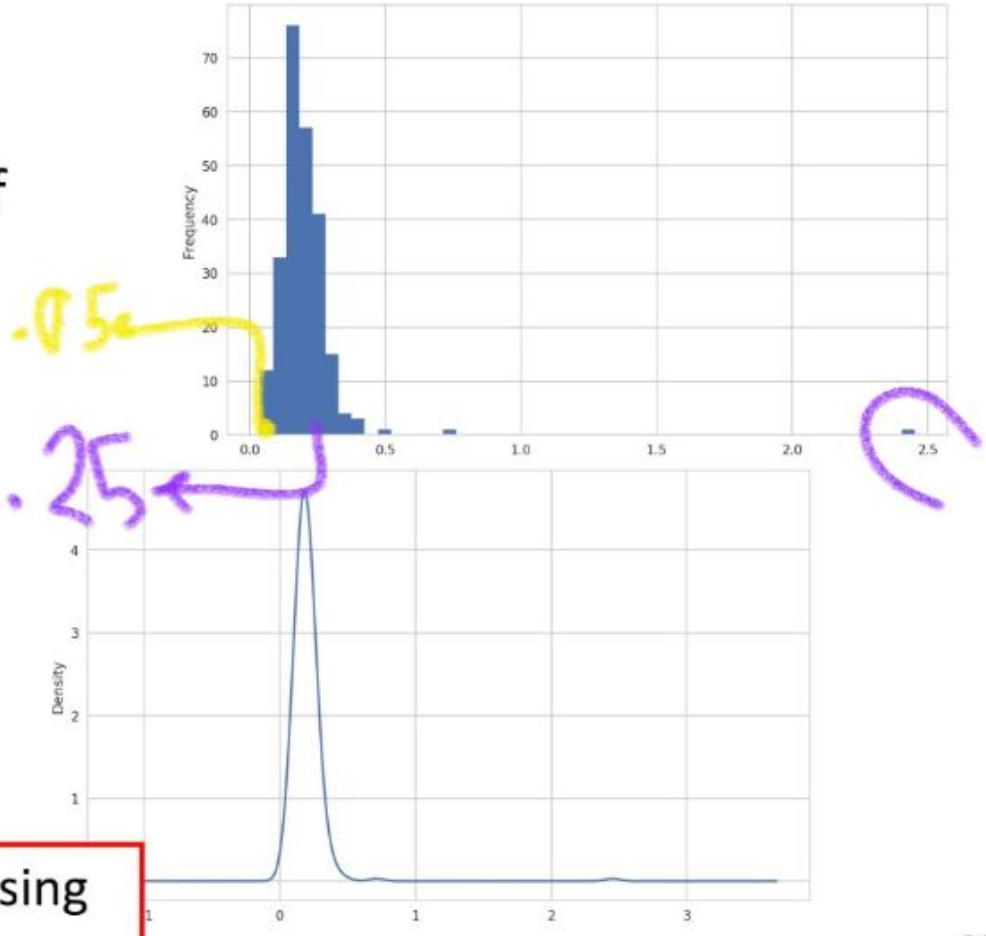
Histograms and Density Plots

- A **histogram** is a kind of bar plot that gives a discretized display of value frequency.

```
tips['tip_pct'].plot.hist(bins=50)
```

- A **density** plot is formed by computing an estimate of a CPD for the observed data.

```
tips['tip_pct'].plot.density()
```



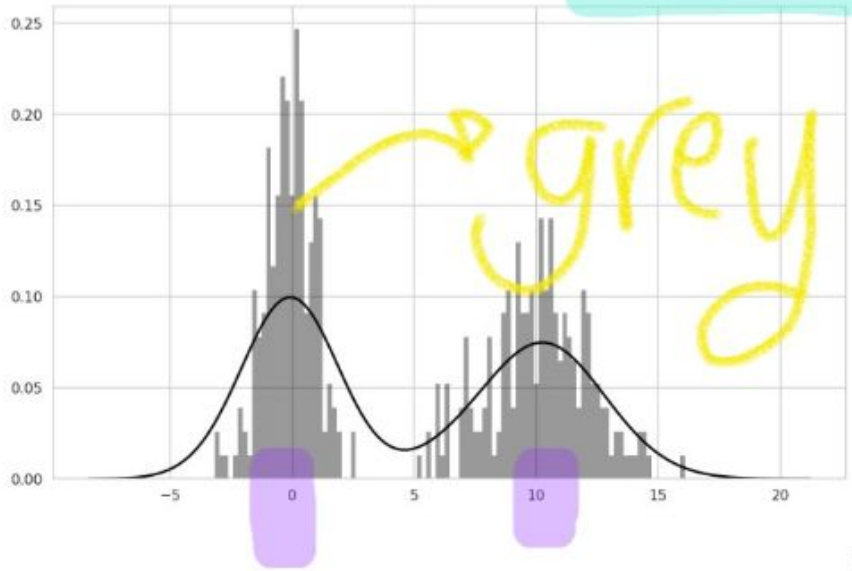
Same as kernel density estimate (KDE) using **plot.kde**

Histograms and Density Plots

- Seaborn's **displot** can plot both a **histogram** and a **continuous density estimate** simultaneously.
- **Example**: bimodal normal distributions.

```
comp1 = np.random.normal(0, 1,
                         size=200)
comp2 = np.random.normal(10, 2,
                         size=200)
```

```
values = pd.Series(np.concatenate(
                        [comp1, comp2]))
sns.displot(values, bins=100,
                    color='k', kde=True)
```



مثال: plot عدد ساعات دراسة الطالب مع علاماته

Scatter or Point Plots

- Point plots or **scatter plots** can be a useful way of **examining** the **relationship** between two one-dimensional data series.
- **Example**: macrodata.csv

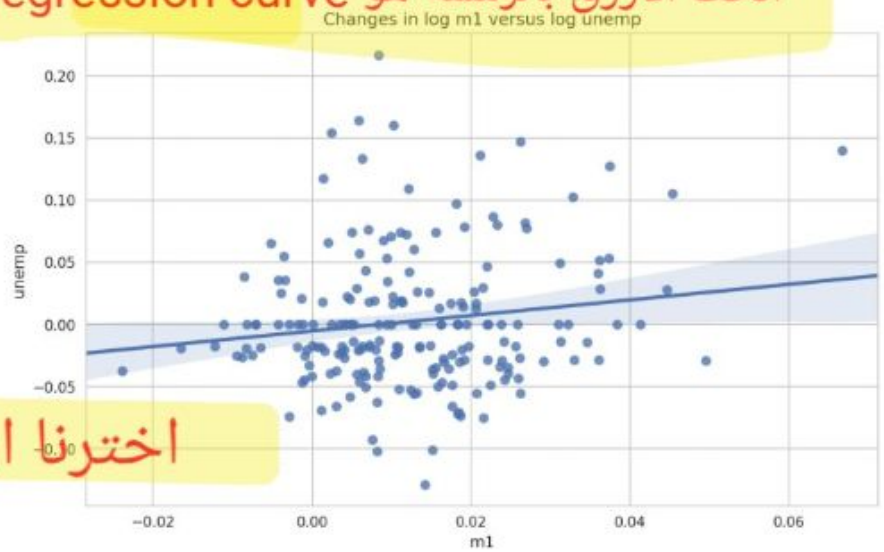
```
macro = pd.read_csv(
            'macrodata.csv')
data = macro[['cpi', 'm1',
              'tbilrate', 'unemp']]
trans_data = np.log(
                data).diff().dropna()
```

عالج القيم المتطرفة بال log

اخترنا اربع اعمدة من الداتا

```
sns.regplot('m1', 'unemp',
            data=trans_data)
plt.title('Changes in log %s versus
log %s' % ('m1', 'unemp'))
```

الخط الازرق بالرسمة هو regression curve



-diff: بحسب الفرق بين القيمة والقيمة اللي قبلها
يستخدم لتكتشف العلاقة العكسية

-dropna:

difference اول قيمة ما الها

drop لذلك بنعمللها

1- sns.regplot

2- plt.title استخدم مكتبتين

-seaborn:

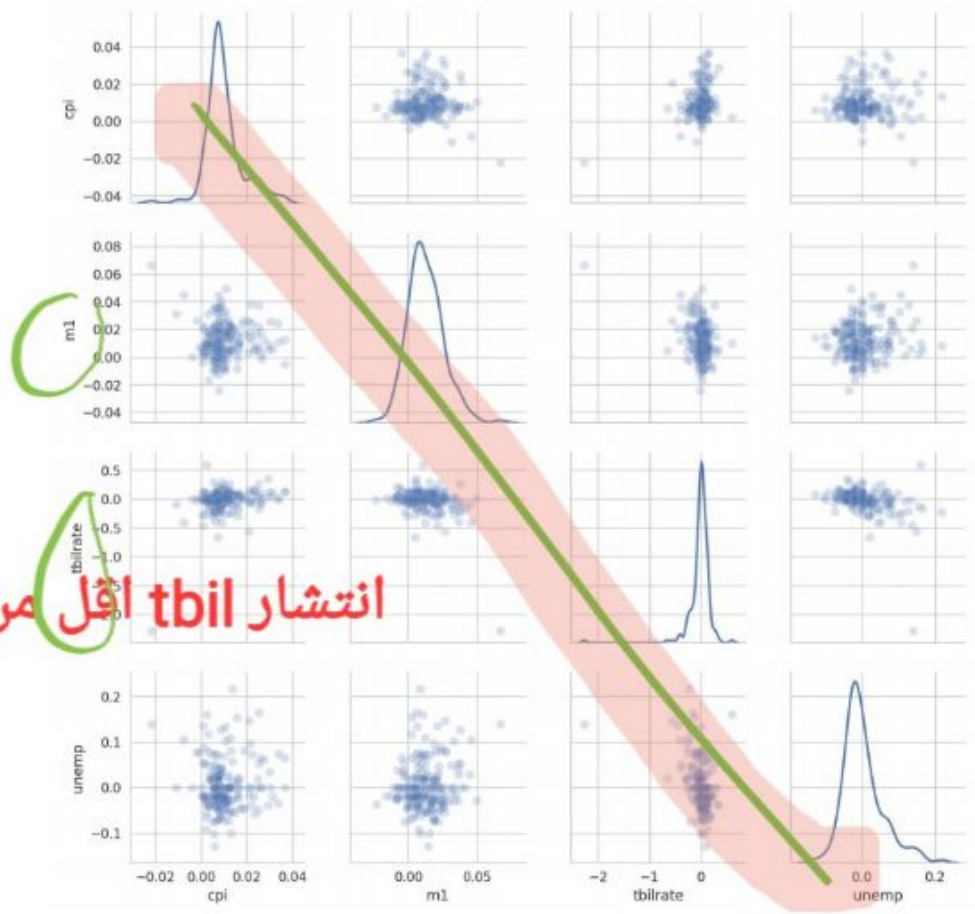
1-اعطاك الظل اللي يعبر عن الكونفيدانس انتيرفال لل regression

2- blue line : regression curve

Scatter or Point Plots

- In exploratory data analysis it's helpful to look at all the scatter plots; this is known as **scatter plot matrix**.

```
sns.pairplot(trans_data,
    diag_kind='kde',
    plot_kws={'alpha': 0.2})
```

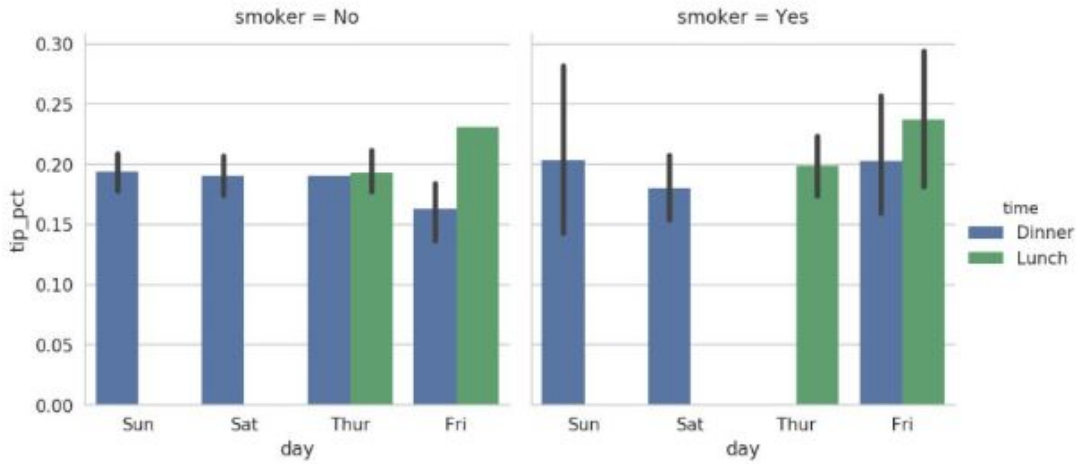


انتشار tbil اقل من انتشار m1

Facet Grids and Categorical Data

- You can use **facet grids** to visualize data with many categorical variables.
- **Example**: Compare tip percentage with smoking.

```
sns.catplot(x='day',
        y='tip_pct', hue='time',
        col='smoker', kind='bar',
        data=tips[tips.tip_pct < 1])
```



-(facet). بتعمل visula على اسطر واعمدة

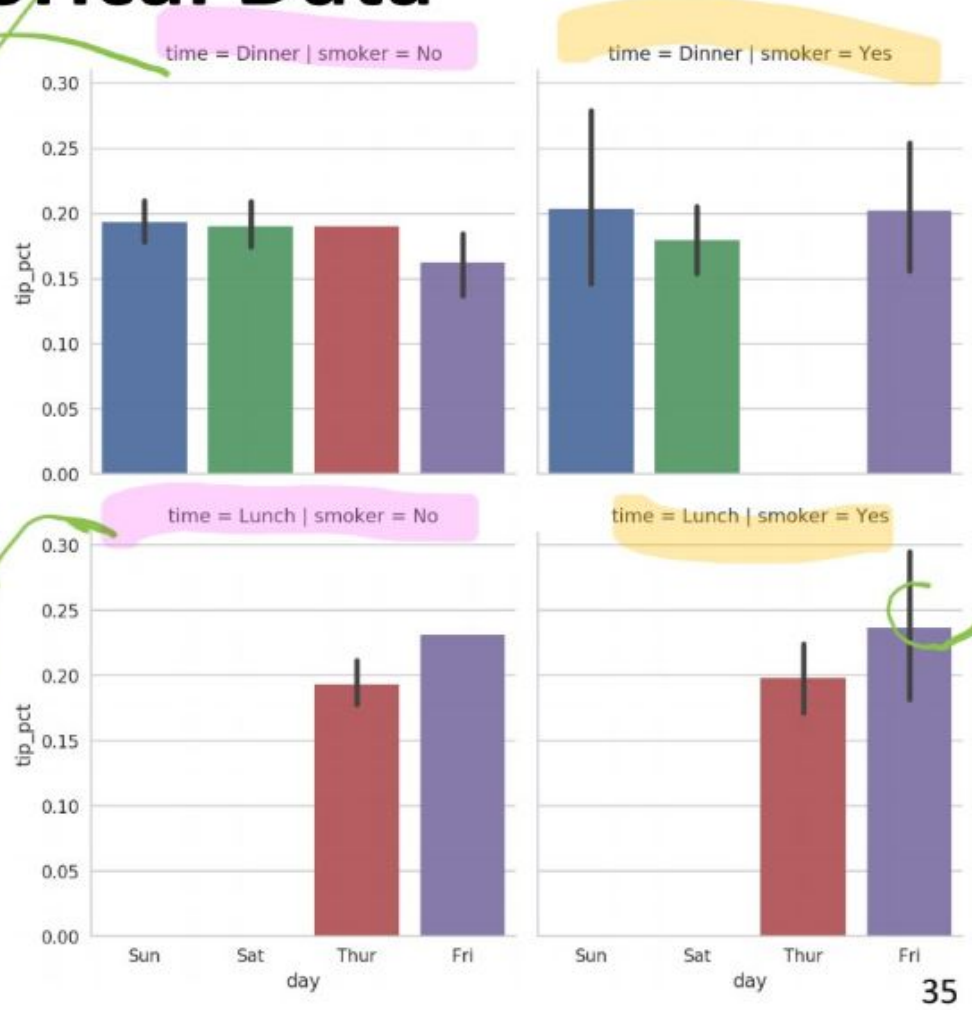
-عملت تحديد فصل الداتا حسب عامود ال smoker

Facet Grids and Categorical Data

- **Example**: Show time in a different facet.

```
sns.catplot(x='day',
      y='tip_pct', row='time',
      col='smoker', kind='bar',
      data=tips[tips.tip_pct < 1])
```

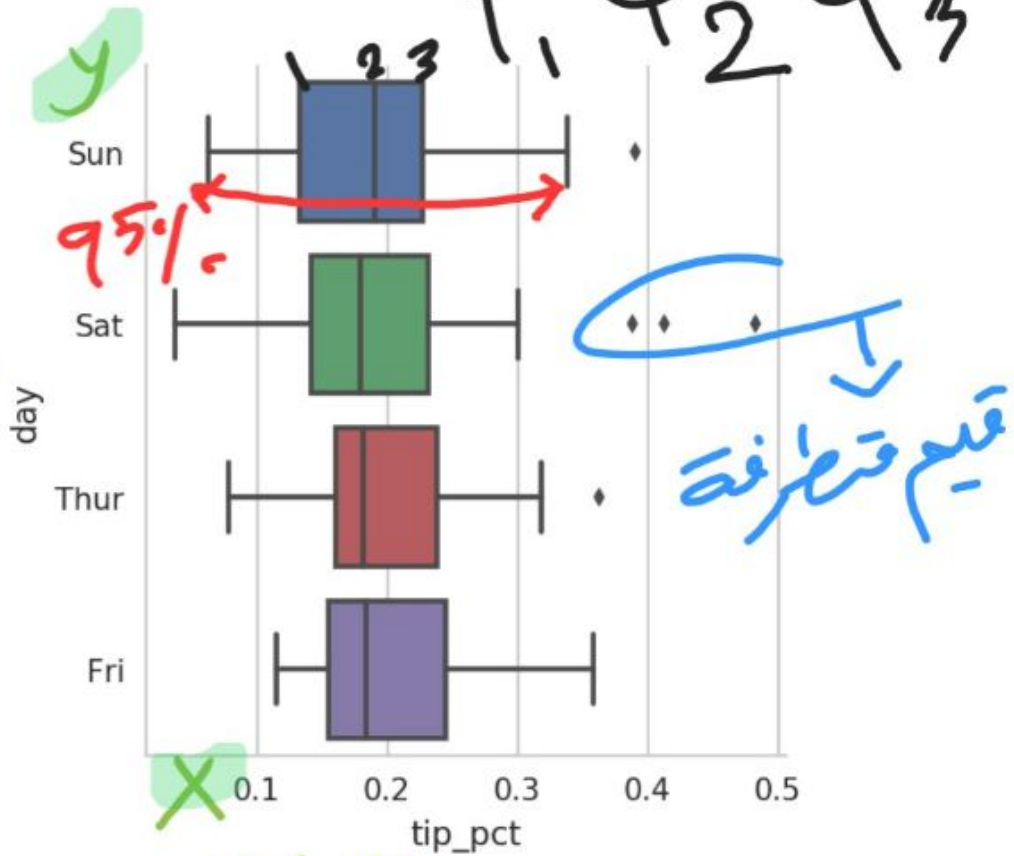
Automatic title



Facet Grids and Categorical Data

- **Example**: Draw a box plot to show the median, quartiles, and outliers.

```
sns.catplot(x='tip_pct',
      y='day', kind='box',
      data=tips[tips.tip_pct < 0.5])
```



Homework

- Solve the homework on **data wrangling and visualization**.

Summary

9.1 A Brief **matplotlib** API Primer

- Figures and Subplots
- Colors, Markers, and Line Styles
- Ticks and Labels
- Saving Plots to File
- matplotlib Configuration

9.2 Plotting with **pandas** and **seaborn**

- Line Plots
- Bar Plots
- Histograms and Density Plots
- Scatter or Point Plots
- Facet Grids and Categorical Data

Co-funded by the
Erasmus+ Programme
of the European Union

DeCAIR

Data Aggregation and Group Operations

Prof. Gheith Abandah

Reference

- **Chapter 10: Data Aggregation and Group Operations**

- Wes McKinney, **Python for Data Analysis**: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media, 2nd Edition, 2018.
  - Material: https://github.com/wesm/pypop-book

Data Aggregation and Group Operations

- **Categorizing** a dataset and **applying a function to each group**, whether an aggregation or transformation, is often a critical component of a data analysis workflow.
  - Split a pandas object **into pieces** using one or more keys
  - Calculate **group summary** statistics
  - Apply **within-group transformations** or other manipulations
  - Compute **pivot tables** and cross-tabulations

Outline

- 10.1 GroupBy Mechanics
- 10.2 Data Aggregation
- 10.3 Apply: General split-apply-combine
- 10.4 Pivot Tables and Cross-Tabulation

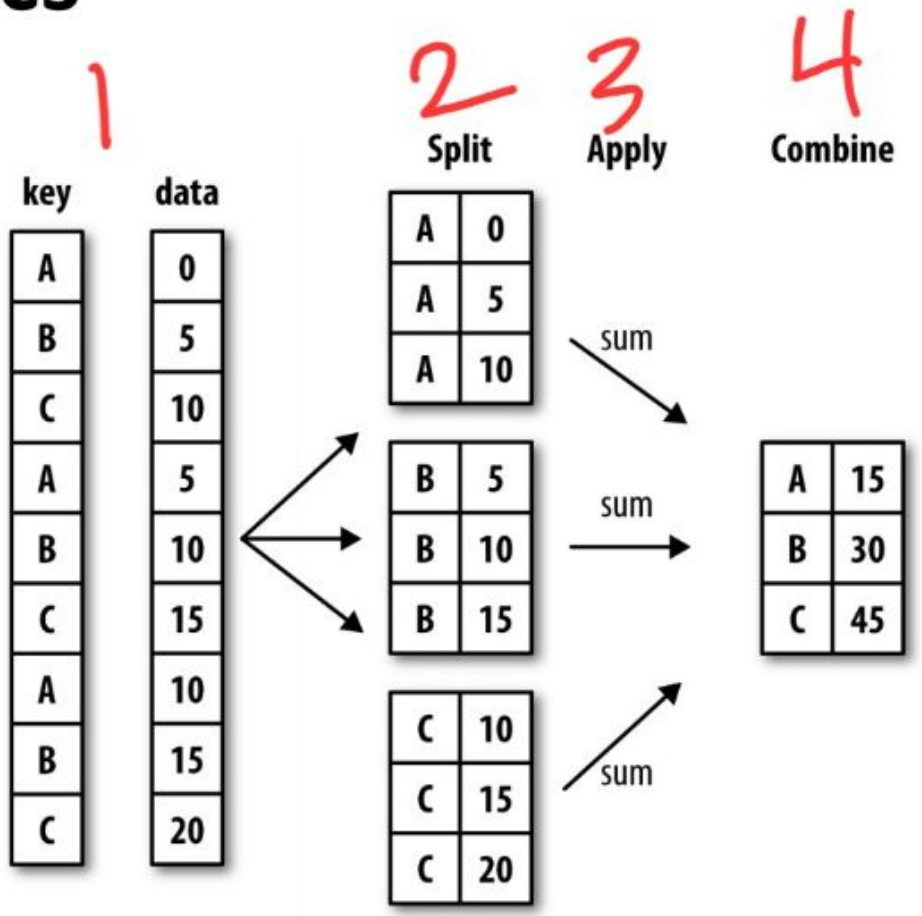
Outline

- 10.1 GroupBy Mechanics
- 10.2 Data Aggregation
- 10.3 Apply: General split-apply-combine
- 10.4 Pivot Tables and Cross-Tabulation

- Iterating Over Groups
- Selecting a Column or Subset of Columns
- Grouping with Dicts and Series
- Grouping with Functions

10.1 GroupBy Mechanics

- Group operations involve **split-apply-combine** sub-operations.
- **Grouping key**
  - **A list** or array of values of same length as the values grouped
  - A value indicating a **column name** in a DataFrame
  - A **dict or Series** giving a correspondence between the values on the axis being grouped and the group names
  - A **function**



10.1 GroupBy Mechanics

Series

- **Example**: Group by a column
- Compute the **mean** of the **data1** using the labels from **key1**.
- The **groupby** method gives an object that can apply some operation to each of the groups.

```
df
  key1 key2     data1     data2
0    a  one  1.303569  1.411498
1    a  two  0.792029 -1.116429
2    b  one -0.422705  0.589257
3    b  two -0.654579 -0.533492
4    a  one  0.567334 -1.029506
```

```
grouped = df['data1'].groupby(
                      df['key1'])
grouped
<pandas.core.groupby.SeriesGrou
pBy object at 0x7faa315[illegible]7390>

grouped.mean()
key1
a    0.746672
b   -0.537585
grouped.size()
key1
a    3
b    2
```

The values and the keys can be default, one, or multiple columns.

```
Out[6]:
  key1 key2     data1     data2
0    a  one  0.664614  1.641934
1    a  two -0.218797  2.417729
2    b  one -0.473717  0.204944
3    b  two  0.008221  0.861692
4    a  one -0.117392 -0.128778

In [7]: type(grouped)
Out[7]: pandas.core.groupby.generic.SeriesGroupBy

In [8]: grouped
Out[8]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000002A88A9A4550>

In [9]: grouped.mean()
Out[9]:
key1
a    0.109475
b   -0.232748
Name: data1, dtype: float64

In [10]: type(grouped.mean())
Out[10]: pandas.core.series.Series
```

Iterating Over Groups

- The **GroupBy** object **supports iteration**.
- Or the pieces can be put in a `dict`.

```
pieces = dict(list(df.groupby(
                    'key1')))
pieces['b']
```

```
      data1     data2 key1 key2
2 -0.519439  0.281746    b  one
3 -0.555730  0.769023    b  two
```

```
for (k1, k2), group in df.groupby(
                ['key1', 'key2']):
    print((k1, k2))
    print(group)
```

k1 k2

```
('a', 'one')
      data1     data2 key1 key2
0 -0.204708  1.393406    a  one
4  1.965781  1.246435    a  one
…
```

بتطلب اشي من هاد الجروب ← group[]

Selecting a Column or Subset of Columns

dataFrame

- Indexing a GroupBy object created from a DataFrame with a column name or array of column names has the effect of column **subsetting** for aggregation.

```
df.groupby(['key1', 'key2'])
                    [['data2']].mean()
                data2   col
key1 key2
a    one    0.190996
b    one    0.589257
     two   -0.533492
s_grouped = df.groupby(['key1',
                        'key2'])['data2']
```

Hierarchical index

استخدمت data2 بدون دبل براكيتس بالتالي انت بتحصل على سيريز هون

- Can get a **Series** when a single column name is passed.

```
In [13]: df.groupby(['key1', 'key2']).mean()
Out[13]:
                data1     data2
key1 key2
a    one    0.273611  0.756578
     two   -0.218797  2.417729
b    one   -0.473717  0.204944
     two    0.008221  0.861692

In [14]: df.groupby(['key1', 'key2'])[['data2']].mean()
Out[14]:
                data2
key1 key2
a    one    0.756578
     two    2.417729
b    one    0.204944
     two    0.861692

In [15]: df.groupby(['key1', 'key2'])[['data2']].mean()
```

```
In [15]: df.groupby(['key1', 'key2'])['data2'].mean()
Out[15]:
key1  key2
a     one     0.756578
      two     2.417729
b     one     0.204944
      two     0.861692
Name: data2, dtype: float64

In [16]: type(df.groupby(['key1', 'key2'])['data2'].mean())
Out[16]: pandas.core.series.Series

In [17]: type(df.groupby(['key1', 'key2'])[['data2']].mean())
Out[17]: pandas.core.frame.DataFrame

In [18]:
```

بعد ما يطبق الدكشينري بعمل grouping بناء على الاعمدة ax=1

Grouping with Dicts and Series

- Grouping information may exist in a **dictionary** or a **series**.
- **Example**: Group by mapping:

```
mapping = {'a': 'red', 'b': 'red',
    'c': 'blue', 'd': 'blue', 'e':
    'red', 'f' : 'orange'}
```

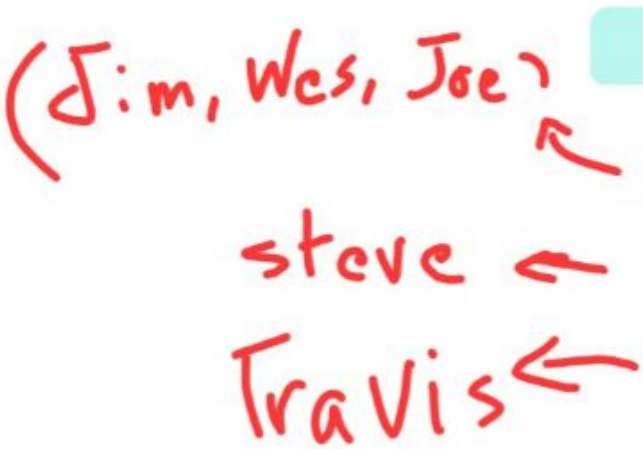
```
by_column = people.groupby(mapping,
                           axis=1)

by_column.sum()
          blue       red
Joe   1.148819  2.478529
Steve -3.498286  0.783905
Wes   -0.505604  0.442407
Jim    1.265539 -2.598872
Travis  0.691711  1.255747
```

لانه ما حددنا ال key صار by default على الاندكس بس حددناله انه الاندكس للاعمدة Ax=1

```
          b         c         d         e
49  1.131019  0.017800  1.746317
Steve  0.313654  0.694288 -2.342447 -1.155839 -0.224036
Wes    1.863238       NaN       NaN -0.505604 -1.420831
Jim   -0.539711 -0.887612  1.136920  0.128619 -1.171549
Travis -0.919159  1.603411 -0.469481  1.161192  0.571495
```

Grouping with Functions

- Any **function** passed as a group key will be **called once per index value**.
- **Example**: Group by length of index.




```
people.groupby(len).sum()
```

	a	b	c	d	e
3	-0.080110	1.248237	2.267939	-0.359185	-0.846063
5	0.313654	0.694288	-2.342447	-1.155839	-0.224036
6	-0.919159	1.603411	-0.469481	1.161192	0.571495

Outline

- 10.1 GroupBy Mechanics
- 10.2 Data Aggregation
- 10.3 Apply: General split-apply-combine
- 10.4 Pivot Tables and Cross-Tabulation

10.2 Data Aggregation

- Aggregations refer to any data transformation that produces **scalar values from arrays**.
- You can use your **own** aggregation **function**.

```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

- Some **other methods** also work.

Fast

Table 10-1. Optimized groupby methods

Function name	Description
count	Number of non-NA values in the group
sum	Sum of non-NA values
mean	Mean of non-NA values
median	Arithmetic median of non-NA values
std, var	Unbiased (n – 1 denominator) standard deviation and variance
min, max	Minimum and maximum of non-NA values
prod	Product of non-NA values
first, last	First and last non-NA values

```
grouped.describe()
```

Column-Wise and Multiple Function Application

Multi Function

- Pandas allows you to aggregate using a **different function** depending on the column, or **multiple functions** at once.

Example:
```
grouped.agg({'tip' : np.max,
             'size' : 'sum'})
```

```
tips = pd.read_csv(
                'examples/tips.csv')
tips['tip_pct'] = tips['tip'] /
                  tips['total_bill']
grouped = tips.groupby(['day',
                        'smoker'])
functions = ['count', 'mean',
             'max']
result = grouped['tip_pct',
       'total_bill'].agg(functions)
```

```
result
              Tip_pct                     total_bill
                count      mean       max count      mean   max
day  smoker
Fri  No          4 0.151650 0.187735     4 18.420000 22.75
     Yes        15 0.174783 0.263480    15 16.813333 40.17
Sat  No         45 0.158048 0.291990    45 19.661778 48.33
     Yes        42 0.147906 0.325733    42 21.276667 50.81
Sun  No         57 0.160113 0.252672    57 20.506667 48.17
     Yes        19 0.187250 0.710345    19 24.120000 45.35
Thur No         45 0.160298 0.266312    45 17.113111 41.19
     Yes        17 0.163863 0.241255    17 19.190588 43.11
```

Outline

- 10.1 GroupBy Mechanics
- 10.2 Data Aggregation
- 10.3 Apply: General split-apply-combine
- 10.4 Pivot Tables and Cross-Tabulation

10.3 Apply: General split-apply-combine

- The **most general-purpose** GroupBy method is **apply**.
- **apply** applies the function to each group.
- **agg** aggregates each column for each group, so you end up with one value per column per group.
- **Example**: Filling missing values with group-specific values

```
data
Ohio          0.922264
New York     -2.153545
Vermont            NaN
Florida      -0.375842
Oregon        0.329939
Nevada             NaN
California    1.105913
Idaho              NaN
```

East

West

Example: Filling Missing Values with Group-Specific Values

```
group_key = ['East'] * 4 +
            ['West'] * 4
```

```
data.groupby(group_key).mean()
```

```
East   -0.535707
West    0.717926
```

```
fill_mean = lambda g:
                g.fillna(g.mean())
```

```
data.groupby(group_key).apply(
                              fill_mean)
```

```
Ohio         0.922264
New York    -2.153545
Vermont     -0.535707
Florida     -0.375842
Oregon       0.329939
Nevada       0.717926
California   1.105913
Idaho        0.717926
```

Outline

- 10.1 GroupBy Mechanics
- 10.2 Data Aggregation
- 10.3 Apply: General split-apply-combine
- 10.4 Pivot Tables and Cross-Tabulation

Pivot Tables

- DataFrame has **pivot_table** that performs groupby operations and adds partial totals (**margins**).

```
tips.pivot_table('tip_pct',
    index=['time', 'smoker'],
    columns='day', aggfunc='mean',
    margins=True)
```

margins

day		Fri	Sat	Sun	Thur	All
time	smoker					
Dinner	No	0.139622	0.158048	0.160113	0.159744	0.158653
	Yes	0.165347	0.147906	0.187250	NaN	0.160828
Lunch	No	0.187735	NaN	NaN	0.160311	0.160920
	Yes	0.188937	NaN	NaN	0.163863	0.170404
All		0.169913	0.153152	0.166897	0.161276	0.160803

Cross-Tabulation

- A **cross-tabulation** (**crosstab**) is a special case of a pivot table that computes **group frequencies**.

```
pd.crosstab([tips.time, tips.day],
            tips.smoker,
            margins=True)
```

smoker		No	Yes	All
time	day			
Dinner	Fri	3	9	12
	Sat	45	42	87
	Sun	57	19	76
	Thur	1	0	1
Lunch	Fri	1	6	7
	Thur	44	17	61
All		151	93	244

Summary

- 10.1 GroupBy Mechanics
- 10.2 Data Aggregation
- 10.3 Apply: General split-apply-combine
- 10.4 Pivot Tables and Cross-Tabulation

Co-funded by the
Erasmus+ Programme
of the European Union

DeCAIR

Time Series

Prof. Gheith Abandah

Reference

- **Chapter 11: Time Series**

- Wes McKinney, **Python for Data Analysis**: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media, 2nd Edition, 2018.
  - Material: https://github.com/wesm/pypop-book

Time Series

- Time series data is an **important** form of structured data in **many** different **fields**, such as finance, economics, ecology, neuroscience, and physics.
- Can be of **fixed frequency** or **irregular**.
- **Types**:
  - **Timestamps**, specific instants in time
  - **Fixed periods**, such as the month January 2007 or the full year 2010
  - **Intervals of time**, indicated by a start and end timestamp.
  - Experiment or **elapsed time**

Outline

11.1 Date and Time Data Types and Tools

- Converting Between String and Datetime

11.2 Time Series Basics

- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting

- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

11.1 Date and Time Data Types and Tools

- The Python **datetime**, **time**, and **calendar** modules support date and time data, as well as calendar-related functionality.
- **datetime** stores both the date and time down to the microsecond.
- **timedelta** represents the temporal difference between two datetime objects.

```
from datetime import datetime
now = datetime.now()
now
datetime.datetime(2020, 12, 16, 14,
                  5, 52, 72973)
now.year, now.month, now.day
(2020, 12, 16)
delta = datetime(2020, 12, 31, 1) -
            datetime(2020, 1, 1)0
delta
datetime.timedelta(365, 3600)
delta.days, delta.seconds
(365, 3600)
```

11.1 Date and Time Data Types and Tools

- You can **add** (or **subtract**) `timedeltas` to a datetime object to yield a new shifted object.

```
from datetime import timedelta
start = datetime(2020, 12, 16)
start + timedelta(10)
datetime.datetime(2020, 12, 26,
                  0, 0)
start - 2 * timedelta(10)
datetime.datetime(2020, 11, 26,
                  0, 0)
```

Table 11-1. Types in datetime module

Type	Description
`date`	Store calendar date (year, month, day) using the Gregorian calendar
`time`	Store time of day as hours, minutes, seconds, and microseconds
`datetime`	Stores both date and time
`timedelta`	Represents the difference between two `datetime` values (as days, seconds, and microseconds)
`tzinfo`	Base type for storing time zone information

اذا اعطيت timedelta

رقم واحد = بقصد فيه الايام

رقمين = ايام وثواني

٣ ارقام = ايام وثواني واجزاء من الثانية

Converting Between String and Datetime

- You can format **datetime** objects and pandas **Timestamp** objects as strings using **str** or the **strftime** method.

- You can use the same format codes to convert strings to dates using **datetime.strptime**.

```
stamp = datetime(2020, 1, 3)
str(stamp)
'2020-01-03 00:00:00'
stamp.strftime('%Y-%m-%d')
'2020-01-03'
```

```
value = '2020-01-03'
datetime.strptime(value,
                  '%Y-%m-%d')
datetime.datetime(2011, 1, 3, 0, 0)
```

'2020-01-03 00:00:00' :
default form for date .
sorted descending.

stamp.strftime('%Y-%m-%d'):
you formating date using format you need.

Datetime format specification (ISO C89 compatible)

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month [01, 12]
%d	Two-digit day [01, 31]
%H	Hour (24-hour clock) [00, 23]
%I	Hour (12-hour clock) [01, 12]
%M	Two-digit minute [00, 59]
%S	Second [00, 61] (seconds 60, 61 account for leap seconds)
%w	Weekday as integer [0 (Sunday), 6]

Datetime format specification (ISO C89 compatible) – cont.

Type	Description
%U	Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are "week 0"
%W	Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are "week 0"
%z	UTC time zone offset as +HHMM or -HHMM; empty if time zone naive
%F	Shortcut for %Y-%m-%d (e.g., 2012-4-18)
%D	Shortcut for %m/%d/%y (e.g., 04/18/12)

Converting Between String and Datetime

stamp.strftime('%Y-%m-%d') بعطيه شكل الفورم

- The **parser.parse** method can parse dates of **common** date formats.

- For our date format, you can pass **dayfirst=True**.

default = False

```
from dateutil.parser import parse
parse('2020-01-03')
datetime.datetime(2020, 1, 3, 0, 0)
parse('Jan 31, 1997 10:45 PM')
datetime.datetime(1997, 1, 31,
                  10pm ← 22, 45)
parse('6/12/2020', dayfirst=True)
datetime.datetime(2020, 12, 6,
                  0, 0)
```

Converting Between String and Datetime

- The pandas **to_datetime** method parses many date representations including standard date formats quickly.
- It also handles **missing values**.
- pandas stores timestamps using NumPy's **datetime64** data type at the nanosecond resolution.

```
datestrs = ['2011-07-06 12:00:00',
            '2011-08-06 00:00:00']
idx = pd.to_datetime(datestrs +
                     [None])
idx
DatetimeIndex(['2011-07-06
12:00:00', '2011-08-06 00:00:00',
'NaT'], dtype='datetime64[ns]',
freq=None)
```

new index

Outline

11.1 Date and Time Data Types and Tools
- Converting Between String and Datetime

11.2 Time Series Basics
- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting
- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

11.2 Time Series Basics

- pandas **Series** can be **indexed** by timestamps.
- The datetime objects is put in a **DatetimeIndex**.

- pandas stores scalar values of datetime objects as **Timestamp** objects.

```
ts.index[0]
Timestamp('2011-01-02 00:00:00')
```

```
dates = [datetime(2011, 1, 2),
         datetime(2011, 1, 5),
         datetime(2011, 1, 7)]
ts = pd.Series(np.random.randn(3),
               index=dates)
ts
```

index

```
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
```

```
ts.index
DatetimeIndex(['2011-01-02', '2011-
01-05', '2011-01-07'],
dtype='datetime64[ns]', freq=None)
```

Indexing, Selection, Subsetting

- Time series behaves like any other pandas.Series when you are **indexing** and **selecting** data based on **label**.

- You can also pass a **string** that is interpretable as a date.

- Slicing:

Equivalent to:
`ts['1/5/2011':'1/7/2011']`

```
stamp = ts.index[2]
ts[stamp]
-0.5194387150567381

ts['1/7/2011']
-0.5194387150567381
ts['20110107']
-0.5194387150567381
ts[datetime(2011, 1, 5):]
2011-01-05   0.478943
2011-01-07  -0.519439
```

اذا بدك شهر كامل او سنة كاملة

Indexing, Selection, Subsetting

- For long time series, **select slices** of data by passing a **year** or only a **year and month**.

من هون إلى ١٠٠٠ يوم

- The **truncate** method slices a Series between two dates.

```
len(longer_ts.truncate(after='12/31/2020'))
```

```
366
```

٢٠٢٠ كبيسة

```
longer_ts = pd.Series(
    np.random.randn(1000),
    index=pd.date_range(
        '1/1/2020', periods=1000))
len(longer_ts)
```

```
1000
```

2021 مش سنة كبيسة

```
len(longer_ts['2021'])
```

```
365
```

```
len(longer_ts['2021-5'])
```

```
31
```

اي شيء بعد 2020- 12-31 ما بدي ياه بالتالي برجعلي

سنة ٢٠٢٠ وهي كبيسة لهيك طولها ٣٦٦
Indexing, Selection, Subsetting

- **DataFrames** can be indexed by time series as well.

بدي ١٠٠ value لكل اربعا ب سنة 2000

100 row, 4 col

```
dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')
long_df = pd.DataFrame(np.random.randn(100, 4), index=dates,
                       columns=['Colorado', 'Texas', 'New York', 'Ohio'])
long_df.loc['5-2001']
```

	Colorado	Texas	New York	Ohio
2001-05-02	0.981586	0.056159	-1.787511	0.348340
2001-05-09	0.758415	-1.591688	1.103367	0.945877
2001-05-16	-1.797193	-1.660692	-0.415911	0.521095
2001-05-23	0.675365	-0.169244	0.174330	-0.448221
2001-05-30	-0.699178	0.892690	1.535503	0.350494

```
[100 rows x 4 columns]

In [10]: long_df['2023-05']
<ipython-input-10-ef63ede8f2ff>:1: FutureWarning: Indexing a DataFrame with a datetimelike index u
 single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a fut
rsion. Use `frame.loc[string]` instead.
  long_df['2023-05']
Out[10]:
            Colorado     Texas  New York      Ohio
2023-05-03 -0.766223  0.683694 -1.106138  0.982921
2023-05-10 -0.443832  0.587776 -1.088769  0.448773
2023-05-17 -0.396149 -0.059591 -1.318055  0.130527
2023-05-24  0.112922 -0.478922 -1.554891  0.442007
2023-05-31  0.036924  0.450320  0.799786  0.499715

In [11]:
```

Time Series with Duplicate Indices

- Duplicates in time series index are **allowed**.

```
dates = pd.DatetimeIndex(
    ['1/1/2000', '1/2/2000',
     '1/2/2000', '1/2/2000',
     '1/3/2000'])
dup_ts = pd.Series(np.arange(5),
                   index=dates)
dup_ts
2000-01-01  0
2000-01-02  1
2000-01-02  2
2000-01-02  3
2000-01-03  4
```

```
dup_ts.index.is_unique
False
dup_ts['1/3/2000'] # not duplicated
4
dup_ts['1/2/2000'] # duplicated
2000-01-02  1
2000-01-02  2
2000-01-02  3

grouped = dup_ts.groupby(level=0)
grouped.count()
2000-01-01  1
2000-01-02  3
2000-01-03  1
```

Outline

11.1 Date and Time Data Types and Tools
- Converting Between String and Datetime

11.2 Time Series Basics
- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting
- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

11.3 Date Ranges, Frequencies, and Shifting

- pandas has a full suite of standard time series frequencies and tools for:
  - **generating** fixed-frequency date **ranges**.
  - **inferring frequencies**,
  - **resampling**.

Generating Date Ranges

- pandas **date_range** can generate time series index by specifying:
  - Start and end
  - Start or end and periods
- The **freq** option allows you to specify wide range of frequencies:
  - H, min, S
  - D, B, M, BM, MS, BMS
  - W-SUN, W-MON, …
  - AS-JAN, AS-FEB, …

BM: اخر يوم عمل في الشهر

اول يوم عمل في الشهر

```
pd.date_range('2020-12-16',
              '2020-12-19')
DatetimeIndex(['2020-12-16',
 '2020-12-17', '2020-12-18'
 '2020-12-19'],
 dtype='datetime64[ns]', freq='D')
pd.date_range(start='2012-04-01',
              periods=20)
pd.date_range(end='2012-06-01',
              periods=20)
pd.date_range('2020-01-01',
              '2020-12-01', freq='MS')
```

default day

```
In [4]: pd.date_range(start='2023-05-03', periods=10)
Out[4]:
DatetimeIndex(['2023-05-03', '2023-05-04', '2023-05-05', '2023-05-06',
               '2023-05-07', '2023-05-08', '2023-05-09', '2023-05-10',
               '2023-05-11', '2023-05-12'],
              dtype='datetime64[ns]', freq='D')

In [5]: pd.date_range(end='2023-12-31', periods=10)
Out[5]:
DatetimeIndex(['2023-12-22', '2023-12-23', '2023-12-24', '2023-12-25',
               '2023-12-26', '2023-12-27', '2023-12-28', '2023-12-29',
               '2023-12-30', '2023-12-31'],
              dtype='datetime64[ns]', freq='D')

In [6]:
```

MS: month start

```
In [6]: pd.date_range(start='2023-05-03', periods=10, freq='H')
Out[6]:
DatetimeIndex(['2023-05-03 00:00:00', '2023-05-03 01:00:00',
               '2023-05-03 02:00:00', '2023-05-03 03:00:00',
               '2023-05-03 04:00:00', '2023-05-03 05:00:00',
               '2023-05-03 06:00:00', '2023-05-03 07:00:00',
               '2023-05-03 08:00:00', '2023-05-03 09:00:00'],
              dtype='datetime64[ns]', freq='H')

In [7]:
```

default of 'M' is last day of month .

```
n [7]: pd.date_range(start='2023-05-03', periods=10, freq='B')
ut[7]:
atetimeIndex(['2023-05-03', '2023-05-04', '2023-05-05', '2023-05-08',
              '2023-05-09', '2023-05-10', '2023-05-11', '2023-05-12',
              '2023-05-15', '2023-05-16'],
             dtype='datetime64[ns]', freq='B')

n [8]: pd.date_range(start='2023-05-03', periods=10, freq='M')
ut[8]:
atetimeIndex(['2023-05-31', '2023-06-30', '2023-07-31', '2023-08-31',
              '2023-09-30', '2023-10-31', '2023-11-30', '2023-12-31',
              '2024-01-31', '2024-02-29'],
             dtype='datetime64[ns]', freq='M')

n [9]: pd.date_range(start='2023-05-03', periods=10, freq='MS')
ut[9]:
atetimeIndex(['2023-06-01', '2023-07-01', '2023-08-01', '2023-09-01',
              '2023-10-01', '2023-11-01', '2023-12-01', '2024-01-01',
              '2024-02-01', '2024-03-01'],
             dtype='datetime64[ns]', freq='MS')

n [10]:
```

B:business day

```
In [11]: pd.date_range(start='2023-05-03', periods=10, freq='A')
Out[11]:
DatetimeIndex(['2023-12-31', '2024-12-31', '2025-12-31', '2026-12-31',
               '2027-12-31', '2028-12-31', '2029-12-31', '2030-12-31',
               '2031-12-31', '2032-12-31'],
              dtype='datetime64[ns]', freq='A-DEC')

In [12]:
```


نهاية السنوات --- اخر يوم في السنوات(A)

```
In [12]: pd.date_range(start='2023-05-03', periods=10, freq='A-JAN')
Out[12]:
DatetimeIndex(['2024-01-31', '2025-01-31', '2026-01-31', '2027-01-31',
               '2028-01-31', '2029-01-31', '2030-01-31', '2031-01-31',
               '2032-01-31', '2033-01-31'],
              dtype='datetime64[ns]', freq='A-JAN')

In [13]: pd.date_range(start='2023-05-03', periods=10, freq='AS-JAN')
Out[13]:
DatetimeIndex(['2024-01-01', '2025-01-01', '2026-01-01', '2027-01-01',
               '2028-01-01', '2029-01-01', '2030-01-01', '2031-01-01',
               '2032-01-01', '2033-01-01'],
              dtype='datetime64[ns]', freq='AS-JAN')

In [14]: pd.date_range(start='2023-05-03', periods=10, freq='AS-FEB')
Out[14]:
DatetimeIndex(['2024-02-01', '2025-02-01', '2026-02-01', '2027-02-01',
               '2028-02-01', '2029-02-01', '2030-02-01', '2031-02-01',
               '2032-02-01', '2033-02-01'],
              dtype='datetime64[ns]', freq='AS-FEB')

In [15]:
```

Frequencies and Date Offsets

- Frequencies in pandas are composed of a **base frequency** and a **multiplier**.

```
from pandas.tseries.offsets import
            Hour, Minute
pd.date_range('2000-01-01',
  '2000-01-03 23:59', freq='4h')


Hour(2) + Minute(30)
<150 * Minutes>
pd.date_range('2000-01-01',
      periods=10, freq='1h30min')
```

```
In [16]: pd.date_range(start='2023-05-03', periods=10, freq='4h')
Out[16]:
DatetimeIndex(['2023-05-03 00:00:00', '2023-05-03 04:00:00',
               '2023-05-03 08:00:00', '2023-05-03 12:00:00',
               '2023-05-03 16:00:00', '2023-05-03 20:00:00',
               '2023-05-04 00:00:00', '2023-05-04 04:00:00',
               '2023-05-04 08:00:00', '2023-05-04 12:00:00'],
              dtype='datetime64[ns]', freq='4H')

In [17]: pd.date_range(start='2023-05-03', periods=10, freq='1h30min')
Out[17]:
DatetimeIndex(['2023-05-03 00:00:00', '2023-05-03 01:30:00',
               '2023-05-03 03:00:00', '2023-05-03 04:30:00',
               '2023-05-03 06:00:00', '2023-05-03 07:30:00',
               '2023-05-03 09:00:00', '2023-05-03 10:30:00',
               '2023-05-03 12:00:00', '2023-05-03 13:30:00'],
              dtype='datetime64[ns]', freq='90T')

In [18]:
```

shift:

اذا اكتشفت انه ال values عند times معينة هي خطأ او مخربطة

Shifting (Leading and Lagging) Data

- **Shifting** refers to moving data backward and forward through time. Both Series and DataFrame have a shift method that leaves the **index unmodified**.

```
ts = pd.Series(np.random.randn(4),
   index=pd.date_range('1/1/2020',
   periods=4, freq='M'))
ts
2020-01-31    0.137923
2020-02-29    0.887474
2020-03-31   -0.090994
2020-04-30    0.412466
```

```
ts.shift(2)
2020-01-31         NaN
2020-02-29         NaN
2020-03-31    0.137923
2020-04-30    0.887474
```

```
ts.shift(-2)
2020-01-31   -0.090994
2020-02-29    0.412466
2020-03-31         NaN
2020-04-30         NaN
```

Shifting (Leading and Lagging) Data

- If the **frequency** is **known**, it can be passed to shift to **advance** the **timestamps** instead of advancing the data.

shift على month

```
ts.shift(2, freq='M')
2000-03-31 -0.066748
2000-04-30  0.838639
2000-05-31 -0.117388
2000-06-30 -0.517795
```

```
ts.shift(3, freq='D')
```

```
ts.shift(1, freq='90T')
```

```
In [20]: ts.shift(1, freq='90T')
Out[20]:
2020-01-31 01:30:00    0.736727
2020-02-29 01:30:00    1.056822
2020-03-31 01:30:00   -1.149580
2020-04-30 01:30:00   -0.588029
dtype: float64

In [21]:
```

shift 1 hour and 30 min = 90T

```
In [21]: ts.shift(3, freq='90T')
Out[21]:
2020-01-31 04:30:00    0.736727
2020-02-29 04:30:00    1.056822
2020-03-31 04:30:00   -1.149580
2020-04-30 04:30:00   -0.588029
dtype: float64

In [22]: ts.shift(-2, freq='90T')
Out[22]:
2020-01-30 21:00:00    0.736727
2020-02-28 21:00:00    1.056822
2020-03-30 21:00:00   -1.149580
2020-04-29 21:00:00   -0.588029
dtype: float64

In [23]:
```

[-2] 3h

لرجوع
تتأخر
الساعة

4:30
h min

Outline

11.1 Date and Time Data Types and Tools

- Converting Between String and Datetime

11.2 Time Series Basics

- Indexing, Selection, Subsetting
- Time Series with Duplicate Indices

11.3 Date Ranges, Frequencies, and Shifting

- Generating Date Ranges
- Frequencies and Date Offsets
- Shifting (Leading and Lagging) Data

Co-funded by the
Erasmus+ Programme
of the European Union

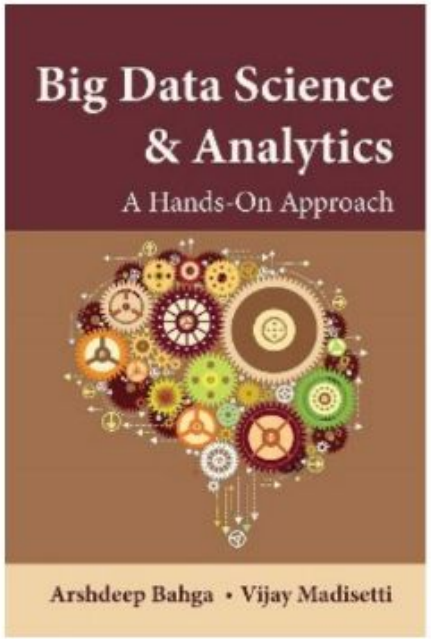
DeCAIR

Introduction to Big Data

Prof. Gheith Abandah

Reference

- Chapter 1: **Introduction to Big Data**



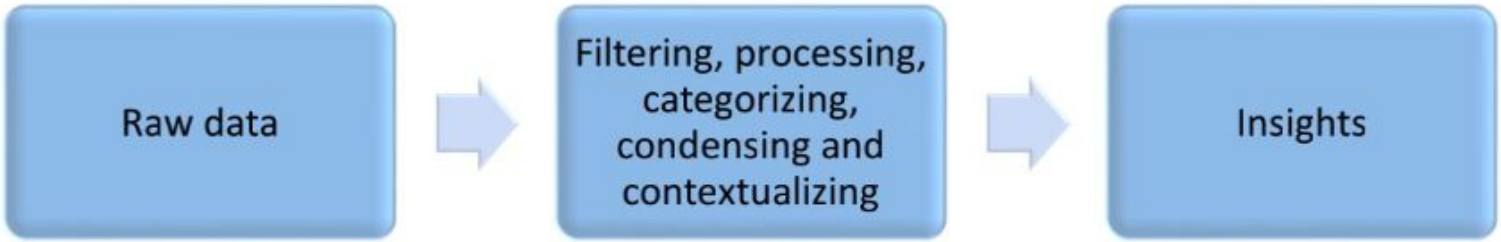
- Arshdeep Bahga and Vijay Madisetti, **Big Data Science and Analytics: A Hands-On Approach**, 2019.
  - Web site: http://www.hands-on-books-series.com/

Outline

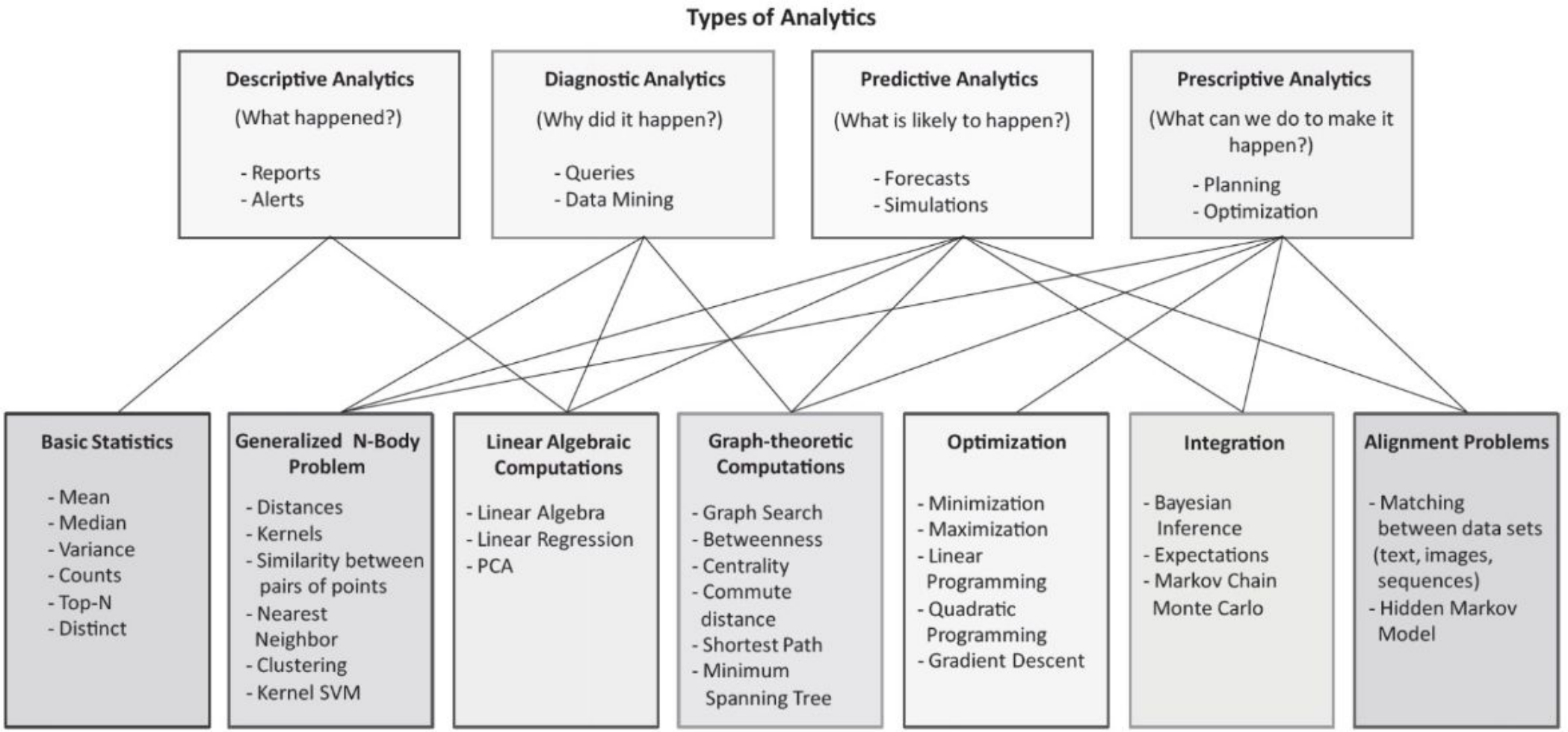
- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

What is Analytics?

- Processes, technologies, frameworks and algorithms to extract meaningful **insights from data**.



- **Goals** of the analytics task:
  - To **predict** something
  - To find **patterns** in the data
  - To find **relationships** in the data


Types of Analytics
Descriptive Analytics
(What happened?)
- Reports
- Alerts
Diagnostic Analytics
(Why did it happen?)
- Queries
- Data Mining
Predictive Analytics
(What is likely to happen?)
- Forecasts
- Simulations
Prescriptive Analytics
(What can we do to make it happen?)
- Planning
- Optimization
Basic Statistics
- Mean
- Median
- Variance
- Counts
- Top-N
- Distinct
Generalized N-Body Problem
- Distances
- Kernels
- Similarity between pairs of points
- Nearest Neighbor
- Clustering
- Kernel SVM
Linear Algebraic Computations
- Linear Algebra
- Linear Regression
- PCA
Graph-theoretic Computations
- Graph Search
- Betweenness
- Centrality
- Commute distance
- Shortest Path
- Minimum Spanning Tree
Optimization
- Minimization
- Maximization
- Linear Programming
- Quadratic Programming
- Gradient Descent
Integration
- Bayesian Inference
- Expectations
- Markov Chain Monte Carlo
Alignment Problems
- Matching between data sets (text, images, sequences)
- Hidden Markov Model
Computational Giants of Massive Data Analysis

Outline

- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

What is Big Data?

- Big data is defined as collections of datasets whose **volume**, **velocity** or **variety** is **so large** that it is difficult to store, manage, process and analyze the data using traditional databases and data processing tools.
- **Every minutes**:
  - **Facebook** users share nearly **4.16 million** pieces of content
  - **Twitter** users send nearly **300,000** tweets
  - **Instagram** users like nearly **1.73 million** photos
  - **YouTube** users upload **300 hours** of new video content
  - **Apple** users download nearly **51,000** apps
  - **Skype** users make nearly **110,000** new calls
  - **Amazon** receives **4300** new visitors
  - **Uber** passengers take **694** rides
  - **Netflix** subscribers stream nearly **77,000** hours of video

What is Big Data?

- **Big data analytics** deals with **collection**, **storage**, **processing** and **analysis** of this massive-scale data.
- **Specialized tools** and frameworks are required for big data analysis.
- Big data tools and frameworks have **distributed** and **parallel processing architectures** and can leverage the storage and computational resources of a large cluster of machines.
- Big data analytics involves several **steps**:
  - data **cleansing**
  - data **munging** (or wrangling)
  - data **processing** and **visualization**

What is Big Data?

- Some **examples** of big data are listed as follows:
  - **Data** generated by **social networks** including text, images, audio and video data
  - **Click-stream data** generated by web applications such as e-Commerce to analyze user behavior
  - **Machine sensor data** collected from sensors embedded in industrial and energy systems for monitoring their health and detecting failures
  - **Healthcare data** collected in electronic health record (EHR) systems
  - **Logs** generated by web applications
  - **Stock markets data**
  - **Transactional data** generated by banking and financial applications

Outline

- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

Characteristics of Big Data

1. **Volume**: Big data is a form of data whose volume is so large that it would not fit on a single machine
2. **Velocity**: Data arrives at very high velocities.
3. **Variety**: Big data comes in different forms such as structured, unstructured or semi-structured, including text data, image, audio, video and sensor data.
4. **Veracity**: Veracity refers to how accurate is the data. To extract value from the data, the data needs to be cleaned to remove noise.
5. **Value**: Refers to the usefulness of data for the intended purpose.

Outline

- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

Domain Specific Examples of Big Data

1. **Web**
   - Web analytics
   - Performance monitoring
   - Ad targeting & analytics
   - Content recommendations

2. **Financial**
   - Credit risk modeling
   - Fraud detection

3. **Healthcare**
   - Epidemiological surveillance
   - Patient similarity-based decision intelligence application
   - Adverse drug events prediction
   - Detecting claim anomalies
   - Evidence-based medicine
   - Real-time health monitoring

Domain Specific Examples of Big Data

4. Internet of Things

- Intrusion detection
- Smart parking
- Smart roads
- Structural health monitoring
- Smart irrigation

5. Environment

- Weather monitoring
- Air pollution monitoring
- Noise pollution monitoring
- Forest fire detection
- River floods detection
- Water quality monitoring

Domain Specific Examples of Big Data

6. Logistics & Transportation

- Real-time fleet tracking
- Shipment monitoring
- Remote vehicle diagnostics
- Route generation & scheduling
- Hyper-local delivery
- Cab/taxi aggregators

7. Industry

- Machine diagnosis & prognosis
- Risk analysis of industrial operations
- Production planning and control

Domain Specific Examples of Big Data

8. Retail

- Inventory management
- Customer recommendations
- Store layout optimization
- Forecasting demand

Outline

- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

Analytics Flow for Big Data

1. **Data Collection**
2. **Data Preparation**
   - Corrupt records, missing values, duplicates, inconsistent abbreviations, inconsistent units, typos and incorrect spellings, incorrect formatting
3. **Analysis**
4. **Visualization**

Data preparation → Filtering → بيتخدم عشان آخد داتا بيكون ليها حسب key معين

Basic statistics → Distinct → بيجيب القيم ال unique

Clustring → لازم تكون الداتا مقسمة إلى subgroups وكل جروب فيه داتا متشابهة مع بعض وكل جروب مختلف عن داتا جروب آخر

Classification → example pcR test for covid-19 virus

---

regression → معناها بتستخرج وتتنبأ ب Feature

---

recommendation:

→ Item-based → بترشحلك بوست على أكثر اشيا بتدخلها وتهتم فيها

→ Collaborative → 2 users like common things so if of users watch something interested, he will forward it to the other user [Netflix]

Dimensionality reduction: استخدامه اذا كان عندك داتا ال دايمنشن داتها صعب و كبير

مرات تستخدم كوسيلة ومرات بيكون هو الهدف الأصلي

pageRank → يعتمد على → محتوياتها
→ علاقتها بالصفحات التانية

connected components : كل اوبجكت له كومبوننت مختلف
عن اوبجكت آخر معه, أو زيادة عنه

Text analysis ⇒ Like reviews, comments

Text mining ⇒ get informations from internet sources

Association Rules ⇒ مثلاً اوجد علاقة بين اسعار الذهب
واسعار سيارات تيسلا
[ممكن يكون في علاقة وممكن لا]

real time analysis ⇒ وهيا الداتا اللي تيجي لبدي اعمل
عليها بنفس الوقت analysis

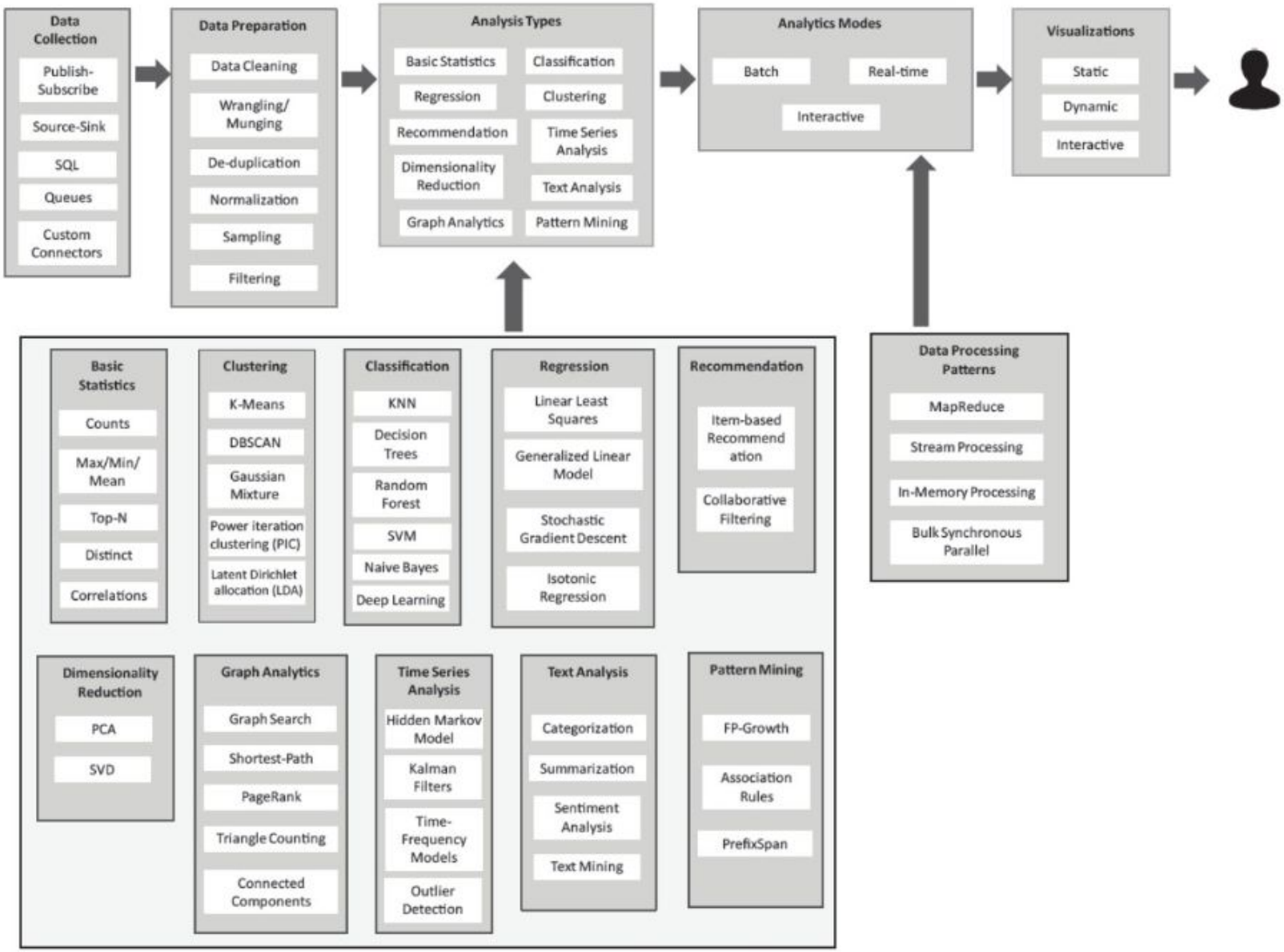
Batch ⇒ history بتحلل ال

Interactive ⇒ ممكن اليوزر بأي لحظة
يطلب عمليات على الداتا
والمطلوب تنفذ بنفس الوقت أو بوقت قصير

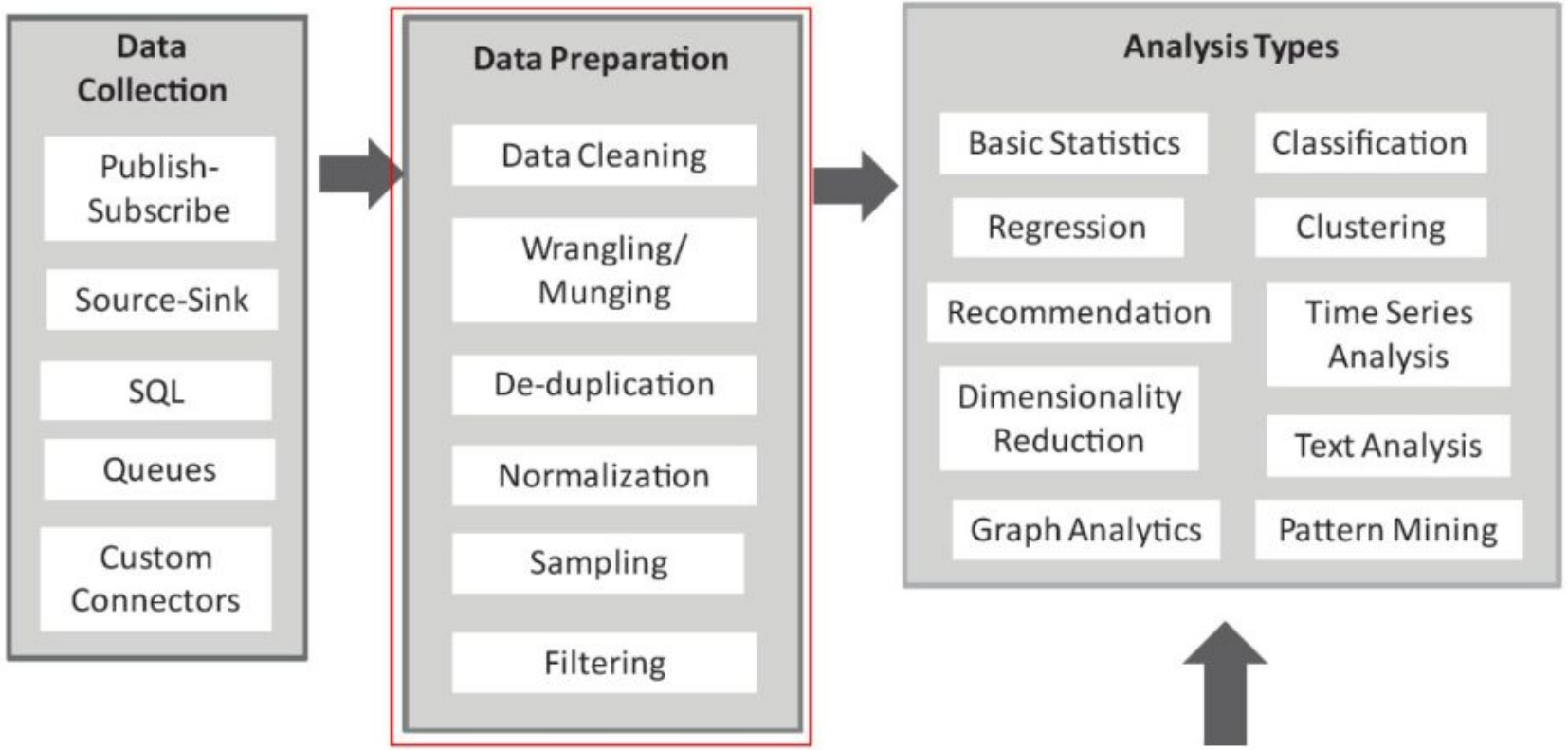
Dynamic : عندك graph بتتغير وبتتبدل

static : graph ثابتة

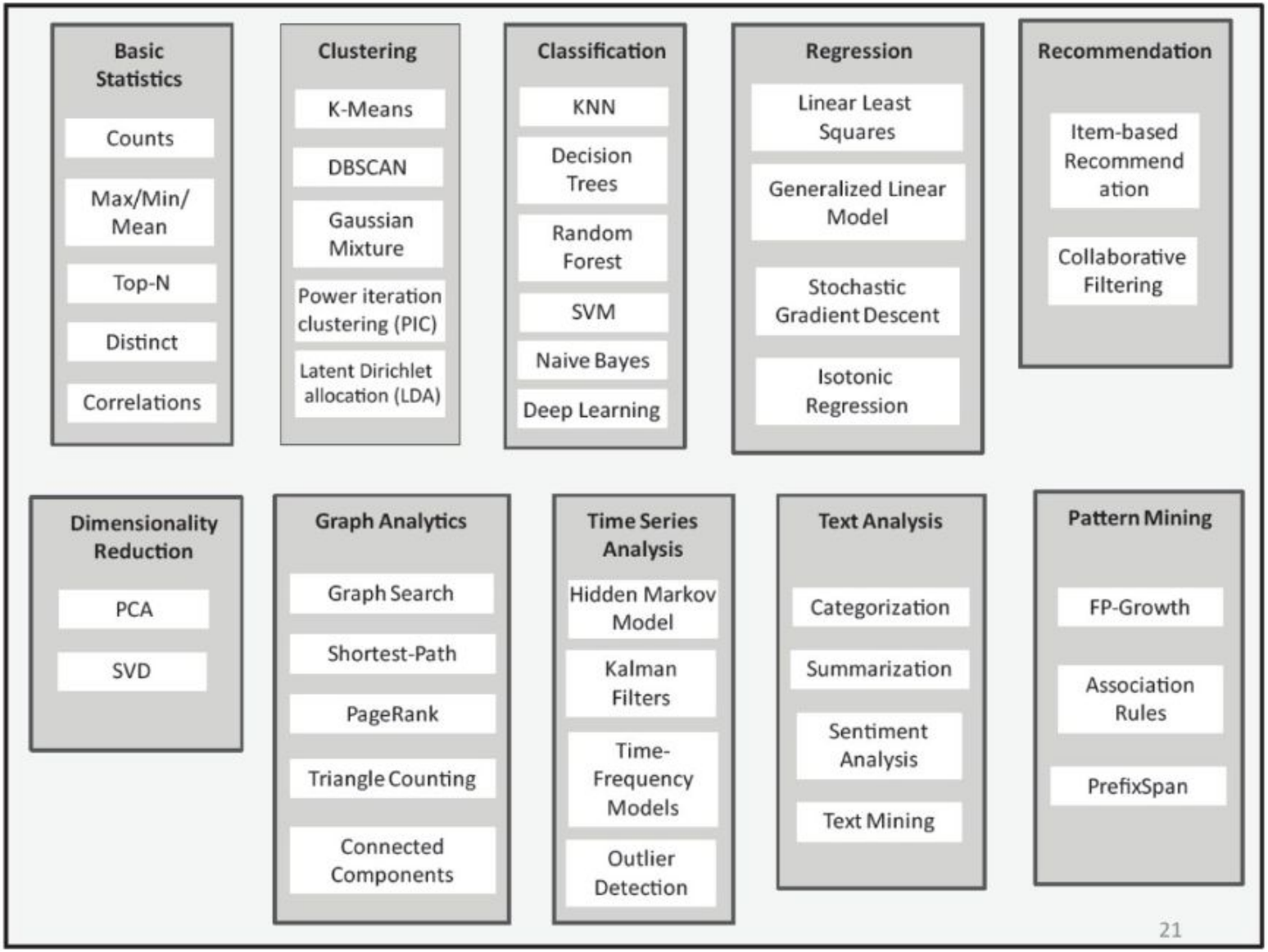
interactive : لما طالب اليوزر بتغير
الـ graph


Data Collection
Publish-Subscribe
Source-Sink
SQL
Queues
Custom Connectors
Data Preparation
Data Cleaning
Wrangling/ Munging
De-duplication
Normalization
Sampling
Filtering
Analysis Types
Basic Statistics
Classification
Regression
Clustering
Recommendation
Time Series Analysis
Dimensionality Reduction
Text Analysis
Graph Analytics
Pattern Mining
Analytics Modes
Batch
Real-time
Interactive
Visualizations
Static
Dynamic
Interactive
Basic Statistics
Counts
Max/Min/ Mean
Top-N
Distinct
Correlations
Clustering
K-Means
DBSCAN
Gaussian Mixture
Power iteration clustering (PIC)
Latent Dirichlet allocation (LDA)
Classification
KNN
Decision Trees
Random Forest
SVM
Naive Bayes
Deep Learning
Regression
Linear Least Squares
Generalized Linear Model
Stochastic Gradient Descent
Isotonic Regression
Recommendation
Item-based Recommend ation
Collaborative Filtering
Data Processing Patterns
MapReduce
Stream Processing
In-Memory Processing
Bulk Synchronous Parallel
Dimensionality Reduction
PCA
SVD
Graph Analytics
Graph Search
Shortest-Path
PageRank
Triangle Counting
Connected Components
Time Series Analysis
Hidden Markov Model
Kalman Filters
Time-Frequency Models
Outlier Detection
Text Analysis
Categorization
Summarization
Sentiment Analysis
Text Mining
Pattern Mining
FP-Growth
Association Rules
PrefixSpan

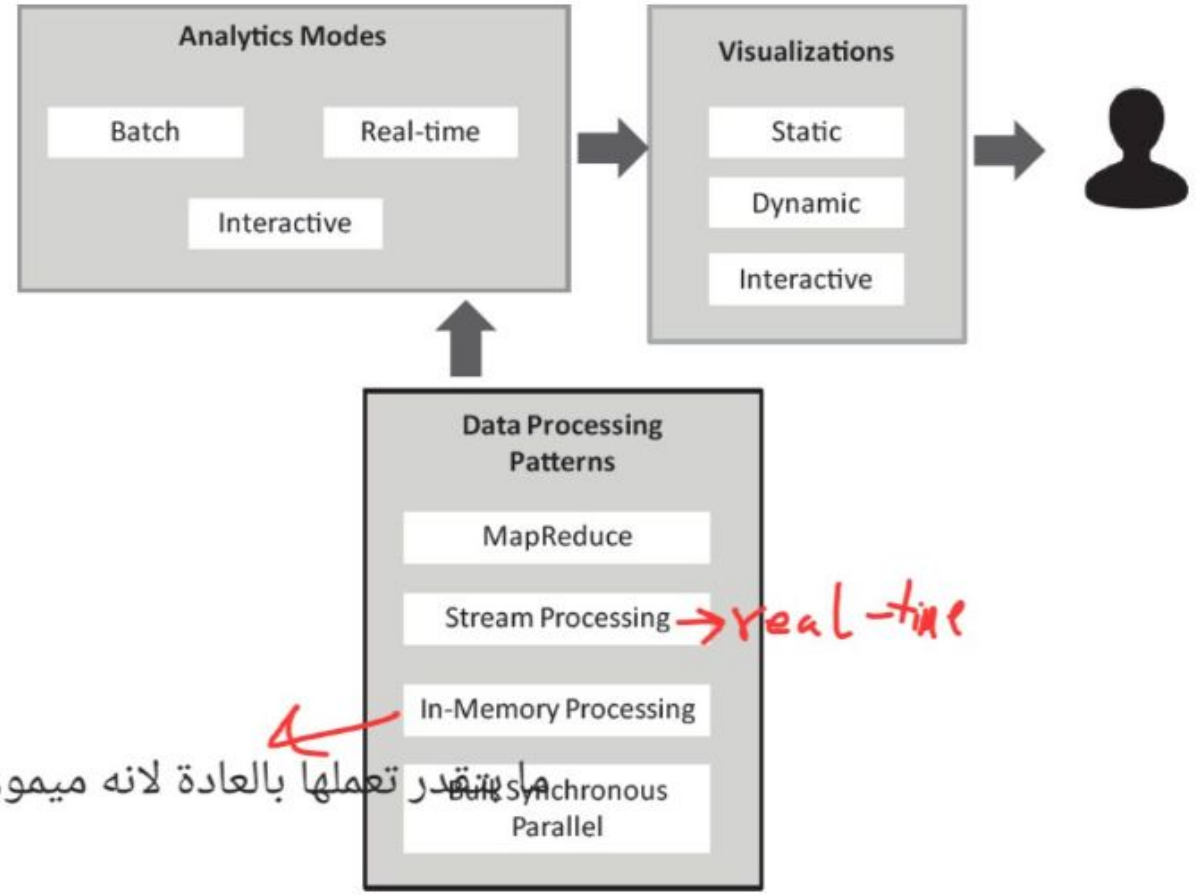
Data Collection, Preparation and Analysis



Analysis Types



Analytics and Visualization Modes

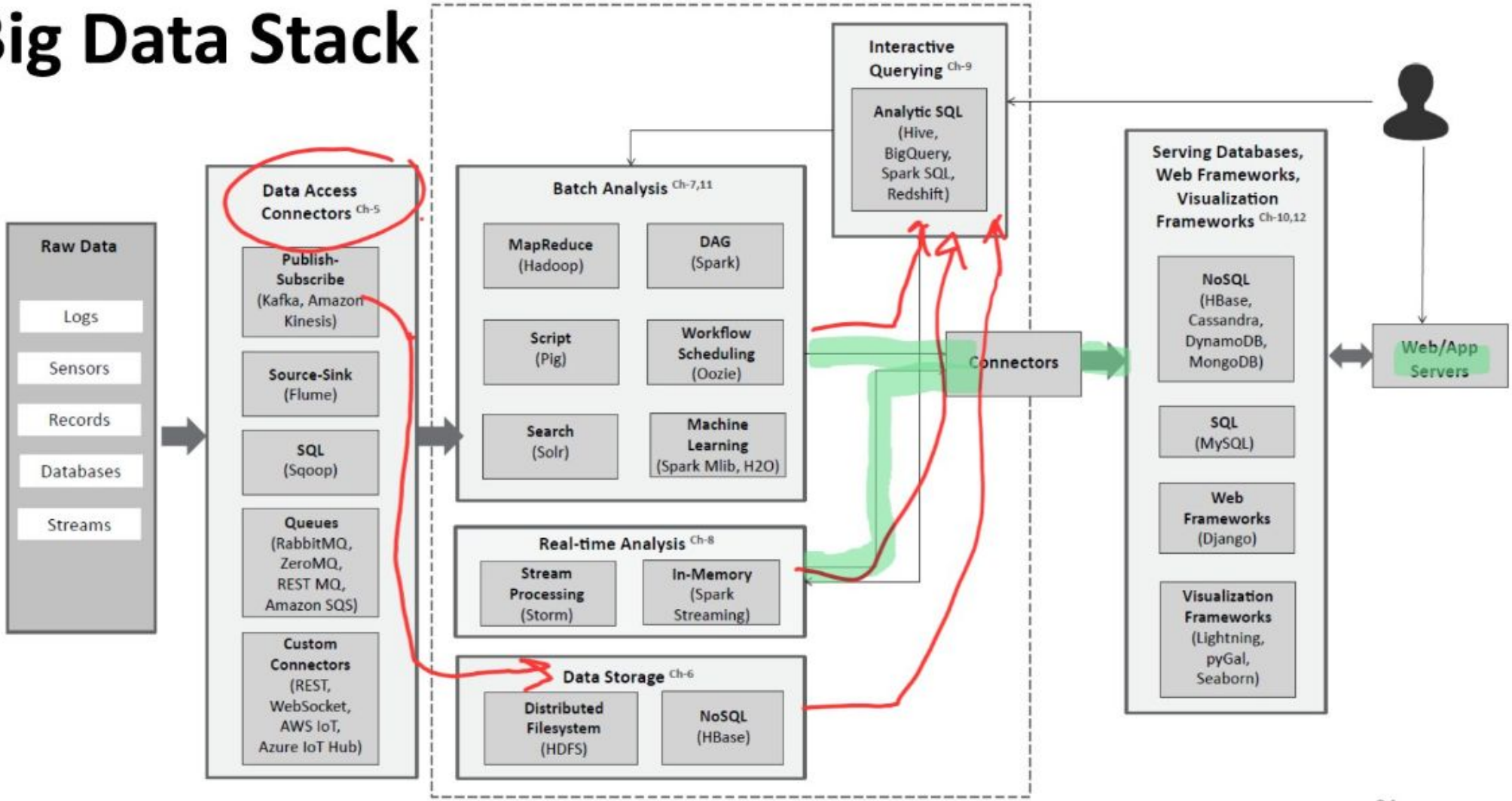


ما بنقدر تعملها بالعادة لانه ميموري الكومبيوتر محدودة

Outline

- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

Big Data Stack



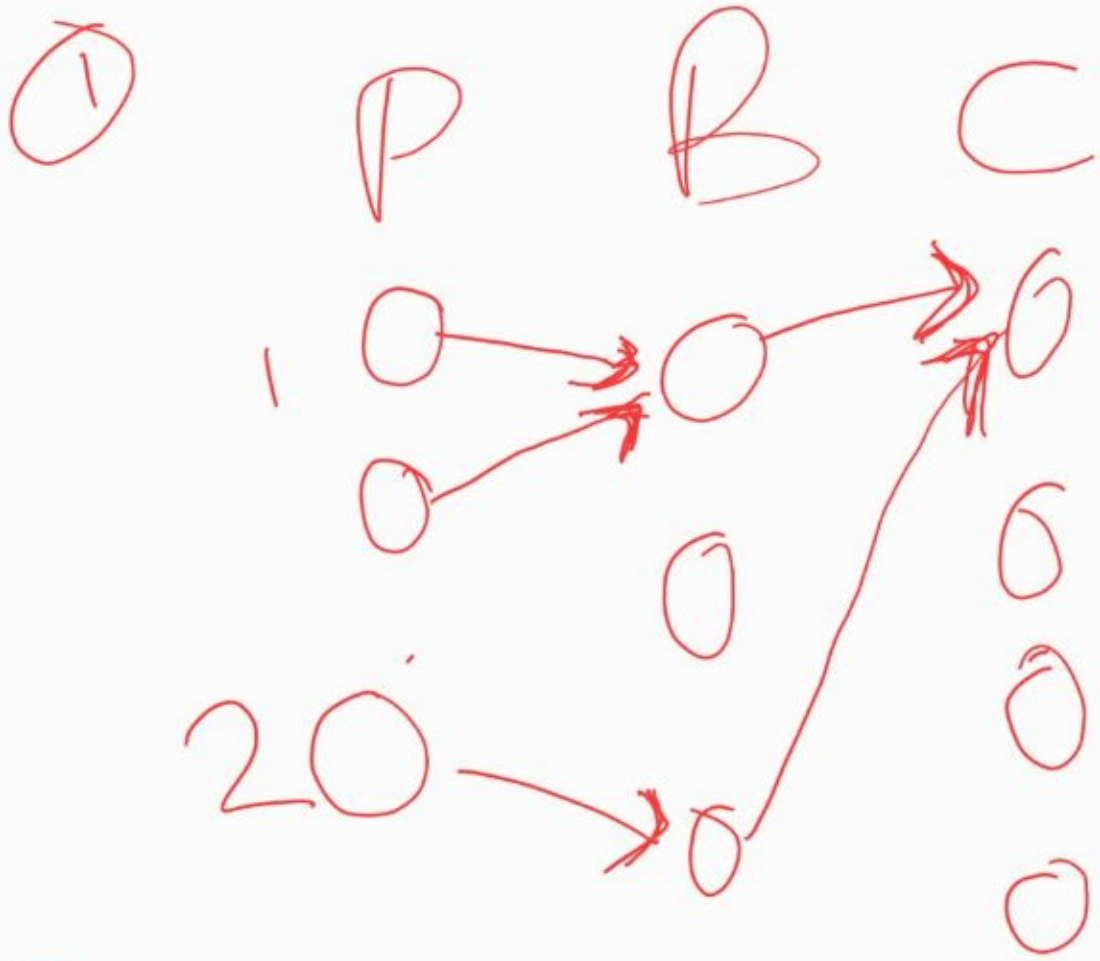
اذا كنا مهتمين بال freq و ال time

1. Raw Data Sources

1. **Logs** generated by web applications and servers for performance monitoring
2. **Transactional data** generated by applications such as eCommerce, banking and financial
3. **Social media data** generated by social media platforms
4. **Databases**: structured data residing in relational databases
5. **Sensor data** generated by Internet of Things (IoT) systems → جسور و سدود
6. **Clickstream data** generated by web applications which can be used to analyze browsing patterns of the users
7. **Surveillance data**: Sensor, image and video data generated by surveillance systems
8. **Healthcare data** generated by Electronic Health Record (EHR) and other healthcare applications
9. **Network data** generated by network devices such as routers and firewalls

2. Data Access Connectors

1. **Publish-Subscribe Messaging** is a communication model that involves publishers, brokers and consumers. E.g., Apache Kafka and Amazon Kinesis.
2. **Source-Sink Connectors** allow efficiently collecting, aggregating and moving data from various sources into a centralized data. E.g., Apache Flume.
3. **Database Connectors** can be used for importing data from relational database management systems into big data storage. E.g., Apache Sqoop.



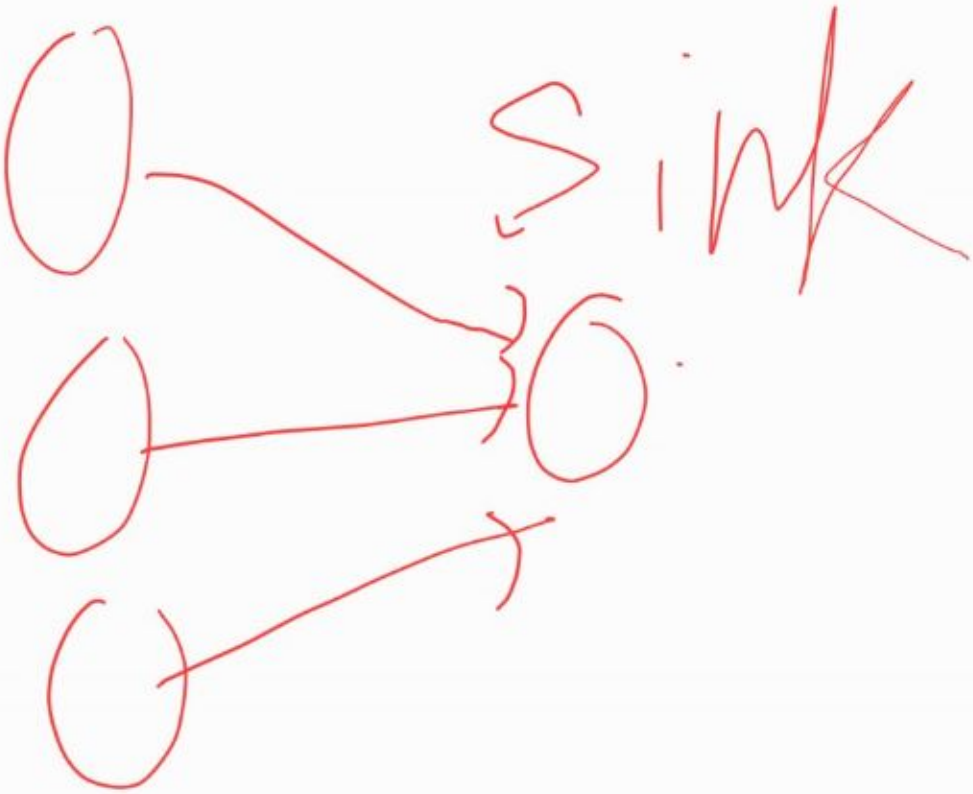
Brokers

Consumers

بتعامل مع ببليشرر محدد معين

بعمل سبسكريبشن مع ال ببليشر اللي مهتم فيهم عن طريق
توصيله مع البرزكرس الناتجه عن الببليشر اللي اختاره

2



2. Data Access Connectors

بتتوزع الداتا على queue

4. **Messaging Queues** are useful for push-pull messaging where the producers push data to the queues and the consumers pull the data from the queues. E.g., RabbitMQ, ZeroMQ, RestMQ and Amazon SQS.

5. **Custom Connectors** can be built based on the source of the data and the data collection requirements. Some examples of custom connectors include custom connectors for collecting data from social networks, and connectors for Internet of Things (IoT). E.g., REST, WebSocket, MQTT, and IoT connectors such as AWS IoT and Azure IoT Hub.

3. Data Storage

- The data storage block in the big data stack stores the data collected from the raw data sources using the data access connectors

الداتا بدها تتقسم لملفات والملفات بدها تتخزن وما بوسعو على جهاز واحد لذلك بنحتاج distributed system

ems, e.g., Hadoop Distributed File System (HDFS)

2. **Non-relational (NoSQL) databases**

لانه الداتا بيس الاصلية عليها كتير overhead ولا تصلح للبيغ داتا

4. Batch Analytics Frameworks

1. **Hadoop-MapReduce** is a framework for distributed batch processing of big data. Its programming model is used to develop batch analysis jobs which are executed in Hadoop clusters.
2. **Pig** is a high-level data processing language which makes it easy for developers to write data analysis scripts which are translated into MapReduce programs by the Pig compiler.
3. **Oozie** is a workflow scheduler system that allows managing Hadoop jobs. With Oozie, you can create workflows which are a collection of actions (such as MapReduce jobs).

4. Batch Analytics Frameworks

4. **Apache Spark** is an open-source cluster computing framework for data analytics. Spark includes various high-level tools for data analysis such as Spark Streaming for streaming jobs, Spark SQL for analysis of structured data, MLlib, and GraphX for graph processing.
5. **Apache Solr** is a scalable and open-source framework for searching data.
6. **Machine Learning**: Spark MLlib is the Spark's machine learning library which provides implementations of various machine learning algorithms. H2O is an open-source predictive analytics framework which provides implementations of various machine learning algorithms.

5. Real-time Analytics Frameworks

1. **Apache Storm** is a framework for distributed and fault-tolerant real-time computation. Storm can be used for real-time processing of streams of data. Storm can consume data from a variety of sources such as publish-subscribe messaging frameworks (such as Kafka or Kinesis), messaging queues (such as RabbitMQ or ZeroMQ) and other custom connectors.
2. **Spark Streaming** is a component of Spark which allows analysis of streaming data such as sensor data, click stream data, and web server logs. The streaming data is ingested and analyzed in micro-batches. Spark Streaming enables scalable, high throughput and fault-tolerant stream processing.

6. Interactive Querying Systems

1. **Spark SQL** enables interactive querying and is useful for querying structured and semi-structured data using SQL-like queries.
2. **Apache Hive** is a data warehousing framework built on top of Hadoop. It provides an SQL-like query language called Hive Query Language, for querying data residing in HDFS.
3. **Amazon Redshift** is a fast, massive-scale managed data warehouse service. It specializes in handling queries on datasets of sizes up to a petabyte or more parallelizing the SQL queries across all resources in the Redshift cluster.
4. **Google BigQuery** is a service for querying massive datasets. It allows querying datasets using SQL-like queries.

7. Serving Databases, Web & Visualization Frameworks

بنعمل تحليلات وبنوضعها متل الملخصات لحتى يسهل عالناس يشوفوها سواء باستخدام web ,or interactive

st widely used Relational Database Management System (RDBMS) and is a good choice to be used as a serving database for data analytics applications where the data is structured.

2. **Amazon DynamoDB** is a fully-managed, scalable, high-performance NoSQL database service. It is an excellent choice for a serving database for data analytics applications as it allows storing and retrieving any amount of data and the ability to scale up or down the provisioned throughput.

يعتبر ك اوبشن لل non sql databases

7. Serving Databases, Web & Visualization Frameworks

3. **Cassandra** is a scalable, highly available, fault tolerant open-source non-relational database system.
4. **MongoDB** is a document oriented non-relational database system. It is powerful, flexible and highly scalable database designed for web applications and is a good choice for a serving database for data analytics applications.
1. **Django** is an open-source web application framework for developing web applications in Python. It is based on the Model-Template-View architecture and provides a separation of the data model from the business rules and the user interface.

7. Serving Databases, Web & Visualization Frameworks

1. **Lightning** is a framework for creating web-based interactive visualizations.
2. **Pygal** is an easy-to-use Python charting library which supports charts of various types.
3. **Seaborn** is a Python visualization library for plotting attractive statistical plots.

Summary

- What is Analytics?
- What is Big Data?
- Characteristics of Big Data
- Domain Specific Examples of Big Data
- Analytics Flow for Big Data
- Big Data Stack

موضوع الدرس ________ اليوم ________ التاريخ / /

← بنحتاج cloud computing ضخمة لحتى نتعامل مع ال Bigdata
ما بنقدر نستخدم لابتوبات
← لا ،بنحتاج special Tools و stack of apps بتقدر انها
تخزن وتحلل وتستلم ال Big data

Mapreduce patterns: نتعلم ال mapreduce algorithim وهي وجهة ما باال B.D analysis

وهي نوع متخصص من parallel computing (code)
"تستخدم multiple nodes لحتى نتعامل مع الداتا الكبيرة وتقدر
تعمل تحليلات عليها.

→ load leveling with quelles.

مثال: في مادة المنيرالين في units for fetch & issue وفيا
في المتوسف بنعمل unit queue بنحط فيها instructions
ال unit الي بعدها.

← وال quelles بتحسن ال performance.
← مثال: عندي داتا بتوصلك من عدة اماكن "كاميرات"
ولازم تحصل لمكان وتتخزن.

← بس تيجي ال producer من ال quelle ← بتصير ال
Consumer يشتغل ← queue الي ما خرجت الـ ...

← بتحتاج cloud computing ضخمة لحتى تتعامل مع ال Bigdata
ما بتقدر تستخدم لابتوبات
← لانه بتحتاج special Tools و stack of apps بتقدر انها
تخزن وتحلل وتنظم ال Big data

Mapreduce patterns: تتعلم ال mapreduce وهي مهمة جداً بال B.D analysis
algorithim

وهي نوع متخصص من parallel computing (code)
" تستخدم multiple nodes لحتى تتعامل مع الداتا الكبيرة وتقدر
تعمل تحليلات عليها.

→ load leveling with quelles.

مثلا: في مادة الميزانين في units for fetch & issue وفي
في المنتصف بنعمل unit queue بتحطها instructions
لل unit اللي بعدها.

← وال quelles بتحسن ال performance.
← مثلا: عدادي داتا بتوصلك من عدة اماكن "كاميرات"
ولازم توصل لمكان وتتخزن.

← بس تيجي ال producer من ال quelle ← بصير ال
consumer ← queue وما يفرق عن الشغل
تشتغل

موضوع الدرس _____ اليوم _____ التاريخ / /

* لو الداتا بتتحرك ببركة نخزن منها بالسيورى نسخ
لانه السيورى اسرع بالتخزين

* لكن لو الداتا كبرت شبطل توصل بالسيورى بتعطل اعطال بار
hard disk

Multi consumers بتقدر تخلص شغل أكبر.

* كيف يمكن نعمل high reliability عند استخدام Multi Consumers ؟
يعنى لو واحد من الـ consumers تعطل
التانيين يقدروا يكملوا شغلهم عادى

Big data

* مثلا عشرة :- لازم يكون عندك multi nodes
عشان لو واحد من الـ nodes اتعطل يقدر يعطى شغله
لـ node تانى عشان يكمل الشغل

طبعا الـ nodes يكونوا مشغولين بنسبة 60% تقريبا عشان لو
حصل تعطل يقدر يكمل شغل node تانية

موضوع الدرس ______ اليوم ______ التاريخ / /

*لازم يكون مش ترتيبات عشان مش one part of queue الا يأخذه أحد غير consumer

choose data → hide data for other consumers → read by chosen consumer

في حالة تعطلة الـ node الي مختاره الشغل ← unhide
بقية الـ consumers شوفوها لما ضموها

* leader Election
How to choose leader from nodes "group of nodes"?
by some protocol

*When we need Election?
1) at beggining
2) new node Join
3) when current leader is dead.

التاريخ / / اليوم ________ موضوع الدرس ________

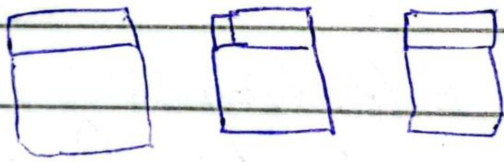
* Elect by the "largest ID"

"Bully algo : the biggest & highest node put her self at leader position

milli seconds — ← مابتاخد وقت كتير ← لأنه local network والمسج بتتفرق .

← الـ leader مش لازم يتعطل بسرعة و يموت

⇒ sharding : multiple storage system.



~~partitioning~~
partitioning data across multiple storage nodes

→ High throughput ⇒ instead of reading & writing from one node or hard disk, you can do it on multiple nodes

→ realiability : each piece of data stored multi times

Asure "
Amazon "the same data shared ... data

موضوع الدرس ______ اليوم ______ التاريخ / /

scaling up : adding nodes, if you predict that your data may get bigger size

• shard key : math operation بتعملها على data attribute تبجين

لتحدد some key "hashed number" لحد الك وين تخزن هاي الداتا "shard اي كا"

موضوع الدرس ________ اليوم ________ التاريخ / /

lecture 22

* بحاجة لـ hash function ليساعدنا نلاقي الداتا بأي shard موجودة

→ CAP: Consistency, Availability, Partition

consistent system: عندك element بأكثر من مكان ممكن تعدل عليه في node والـ node الثانية لما طلبوا التعديل فتظل القيمة مثل ما هي

لازم لما تعدّل تضمن انه كل اماكن وجود الـ element عدلت عليه بالقيمة الجديدة.

available: هل هو متاح حالياً اقرأ واكتب عليه ولا بدي استنى الـ previous writes يوصلوا لكل مكان بعدها يصير متاح

partition: may some nodes leave the connected system.

بالتالي السيستم لازم يظل شغال ويكمل بالرغم من هذا انقطاع

موضوع الدرس _____ اليوم _____ التاريخ / /

available

*Eventual consistency: التعديل حيث وتشوفه

some nodes

وال node التانية لسا وصلهم التعديل لكن بعدهم بكمان

يكون وصلهم بالتالي بصير كل ال nodes متغيين ال data

Strong consistency:

All updated are immediately available to all clients

متاحين توفر ال available اهم التعديلات تنتشر.

*Banks & their services

⇒Bloom filter: ممكن تجاوب نعم/لا عن وجود

ال data وبزمن قصير

" بدور ع كتلة مخزنة عندي أو لا "

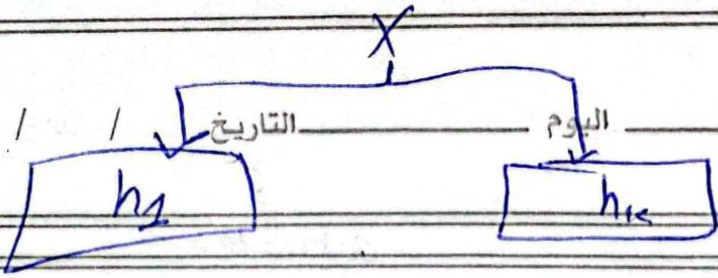
Uses an array of m-bit "initially set to 0's"

X: timestamp فيديو video ال

place الي بتدور

date عليه

موضوع الدرس ____ اليوم ____ التاريخ / /



بعمل set لـ ___ k bits لحتى أخلي هذا الـ element معمله وعندي

العملية :

* بحسب الـ hash func بعطيني وين لازم يكونوا الـ 1's
بعدها اذا لقيته في أماكن الـ bits اللي اعطاني ياها في 1's معناها الـ element موجود

* It might report "False positive"

* بنختار عدد m-bit كبير كفاية وعدد من الـ hash functions بحيث نسبة الـ F.P تكون قليلة

True positive ⇒ Bloom حكى انه موجود ودورنا عليه بالـ hardisk وطلع موجود برضه

False positive ⇒ Bloom حكى موجود لكن هو بالـ hardisk غير موجود.

موضوع الدرس _____ اليوم _____ التاريخ / /

* materialized views

لازم لا يتخزنه أعلى update for data لا في الداتا بتغير

لو غيرت على الداتا بعد ساعتين او اكثر من العملية بتقدر materialized View تضم هاي العملية، وتحسبها من غير ما تعمل

* lambda architecture.

queries بتقدر تشغلها على Big data وبطلع فيها materialize View ونقل عليهم

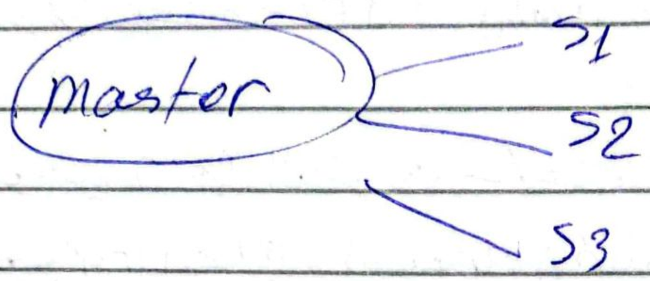
speed layer → بتاخد بعين الاعتبار المعلومات اللي "updated" تغيرت.

Batch view → history data طيب لانه بتشغل على

موضوع الدرس ________ اليوم ________ التاريخ / /

scheduler - Agent - supervisor

(old)



S: server

مشكلة :

* في حال تعطل ال master ما في مين يحلينا الي تعطل

* بخرب كل السيستم بوقع failed

(new) skheduler: بقسم ال query الكبير الى مجموعة من ال tasks

Agent: بوخذ من ال schduler التعليمات و بحكي لل node تاعته تنفذ.

state store: بتخزن فيه التاسكات الي بيبعتها

ما بشتغل بس براقب

supervisor: براقب اذا في task وال node Failed ما عملت شغلها

و بطلب من ال schedul انه يوزعها على node تانية

موضوع الدرس ________ اليوم ______ التاريخ / /

web services : طريقة حتى تتعامل عن dynamic data

client → بقدر يسأل عن Big data → عن طريق request للـ webserv

*abstraction : عليك فقط تعلم شو تمرر للـ web serv وشو بيرجعلك

*gatkeeper : لـ المعلومات كانت بدها authentication بقدر يوصل ب اسم وكلمة سر [حساب]

* اولاً يتأكد انه الـ client
* ثانياً بيسمحله يوصل ويسأل الـ webserv

authentication : تأكيد من هو الشخص
authorization : تحديد الـ services اللي مسموله يستخدمهم

موضوع الدرس _____ اليوم _____ التاريخ / / _____

Consensus الاتفاق

* Agreement: لازم الكل يتفقوا

Valid = مثلاً الاتفاق على تاريخ للامتحان
عن الدروس والطلاب

Termination: حتى لو في nodes ما وافقوا والعتروا: برضه لازم terminate ونوصل لقرار نهائي

→ paxos protocol

* proposer: act as coordinator
Accepter: الناس اللي لازم يتفقوا
learner = الناس اللي لازم يتخبروا بهاد القرار

موضوع الدرس _____ اليوم _____ التاريخ / / /

محاضرة 23

Map reduce patterns.

* الآوتبوت اللي بطلع من الـ map بيجي merged مع بعض and sorted
* وكل reducer بياخد داتا متشابهة مع بعضها

Hadoop ⇒ بحاول يعطي لكل map كمية متساوية من الداتا

failover ⇒ بقدر يكتشف انه في عندي داتا معالجة وغير تاسك ما اكتملت بالكامل بقدر يوزعها على اي node اخر

"map"	input	output
map	(k, V)	(k, V)
sort	(k, V)	(k, list(V))

بجمع المشتركين بـ k وحدد مع بعض القيم

reduce	(k, List(V))	(k, V)

Numerical Summarization

#compute count

* total number of time each page visited in 20

*Mapper : found pages "url" visited in 2014

+reducer : recieve "key value" pairs grouped by same key ⇒ compute count

#Code

from mrjob.job import MRJob

*compute maximam

mapper : (month, page) visited in 2014

Two reduce

① listofpg ← يعني JO

② page JO ← Visits عدد ال
list JL

Refer to slides

yeild k[0] [42]
↓ ↓
month url