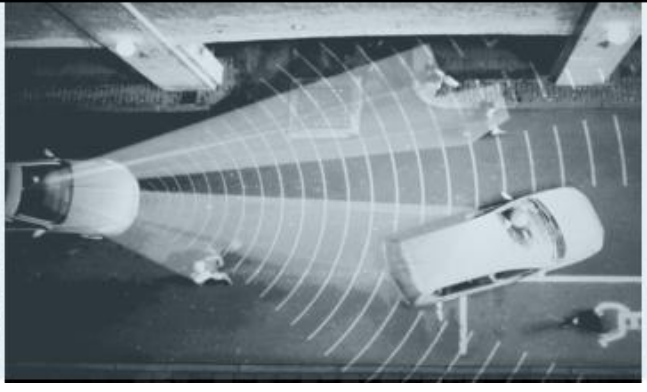
파이썬을 이용한 데이터 분석 (2)

2023. 8. 23



numpy

개요

- numpy : numerical python의 줄임말
- 여러 항목으로 구성된 데이터를 다루려면 python의 리스트나 튜플을 사용할 수 있다.
  - 그러나 다루어야 할 데이터 모두 숫자인 경우는 계산 속도를 개선하기 위해서 numpy 모듈을 사용

- 수학에서의 vector, matrix와 같은 배열(array)를 계산하는데 유리
  - 벡터(vector) : 여러 항목으로 된 1차원 배열
  - 매트릭스(matrix) : 2차원 이상의 다 차원 배열

- 데이터 형태 : ndarray (n dimentional array)를 사용
- 리스트와 달리 배열에서 각 항목은 모두 같은 타입인 숫자(정수나 소수 등)이어야 한다
- Import numpy as np

벡터화

- 기존의 리스트로부터 numpy 배열을 만들 수 있다
- 벡터화 : 하나의 숫자(스칼라)가 배열의 항목 개수만큼 자동으로 확대
  - 리스트에 숫자를 곱하면 동일한 리스트가 숫자 만큼 복사
  - numpy 배열에 숫자를 곱하면 리스트와 달리 모든 항목에 적용
- 벡터화는 곱셈 뿐만 아니라 덧셈에도 적용
- 배열 표현 : [ ]로 묶어서 표시

```
import numpy as np

list = [1, 2, 10]
print(list * 3)                 # [1, 2, 10, 1, 2, 10, 1, 2, 10]

arr = np.array(list)
print(arr * 3)                  # [ 3 6 30]
print(arr + 100)                # [101 102 110]
```

2차원 배열

- 2차원 리스트로부터 2차원 배열

```
data = [[1,2,3], [4,5,6], [7,8,9]]
print(data)                          # [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

arr2 = np.array(data)                # list를 ndarray로 변환
print(arr2)
                                     # [[1 2 3]
                                        [4 5 6]
                                        [7 8 9]]
```

- 배열의 특정 위치를 인덱싱하여
  - ➢ 아래 두 가지 액세스 방법이 가능

```
arr2[1,2]          # 6
arr2[1][2]         # 6
```

2차원 배열

- 다차원 배열에서 한 차원이 낮은 배열을 얻으려면 인자 수를 한 개 작게 주면 된다.

```
arr2[0]          # array([1, 2, 3])
arr2[2]          # array([7, 8, 9])
```

- 슬라이스(slice) : 행의 일부 또는 열의 일부를 얻고자 할 때 사용

```
arr2[:2,1:]      # array([[2, 3],
                 #        [5, 6]])
```

➢ Slice 예
- [:] 전체
- [0:n] 0번째부터 n-1번째까지, 즉 n번 항목은 포함하지 않는다.
- [:5] 0번째부터 4번째까지, 5번은 포함하지 않는다.
- [2:] 2번째부터 끝까지
- [-1] 제일 끝에 있는 항목
- [-2] 제일 끝에서 두번째 항목

arange, reshape, mean

- 일정한 범위의 숫자를 자동으로 생성할 때 사용
  - 파이썬의 range와 유사
- reshape 함수 : 배열의 차원을 생성, 변경할 때 사용

```
data = np.arange(12).reshape(4,3)
data                                    #array([[ 0, 1, 2],
                                        #       [ 3, 4, 5],
                                        #       [ 6, 7, 8],
                                        #       [ 9, 10, 11]])
```

- 이 data 배열에서 4보다 작은 항목을 모두 0으로 대체하는 코드 작성

```
data < 4      # 출력  array([[ True, True, True],
                          [ True, True, False],
                          [False, False, False],
                          [False, False, False]])
```

arange, reshape, mean

- 이 data 배열에서 4보다 작은 항목을 모두 0으로 대체하는 코드 작성

```
data[data  <  4]  =  0
data                                    # 출력       array([[ 0, 0, 0],
                                                              [ 0, 0, 5],
                                                              [ 6, 7, 8],
                                                              [ 9, 10, 11]])
```

arange, reshape, mean

- mean() : 배열에 들어 있는 모든 수의 평균을 구한다.

```
data                    # array([[ 0, 1, 2],
                                 [ 3, 4, 5],
                                 [ 6, 7, 8],
                                 [ 9, 10, 11]])
data.mean()             # 5.5
```

- 배열 전체의 평균이 아니고 각 행 또는 각 열에 대해서 평균을 구할 때
  - 열(column) 기준 평균 : 속성값 axis를 0으로 지정
  - 행(row) 기준 평균 : 속성값 axis를 1로 지정

```
data.mean(axis=0)       # array([ 4.5, 5.5, 6.5])
data.mean(axis=1)       # array([ 1., 4., 7., 10.])
```

DataFrame – 개요

- pandas : panel data analysis의 줄임말
  - '구조화된 데이터 분석'이란 의미
- python에서 데이터를 편리하게 다루기 위해 table(표) 구조로 데이터를 처리하는 경우가 많다
  - 이를 위해서 피이선의 pandas 패키지를 사용
- 데이터 type : DataFrame, Series
- 데이터프레임
  - excel의 spread sheet와 같은 2차원 테이블 구조로 데이터를 다룬다

DataFrame – 생성

- DataFrame에는 숫자, 문자열, 불리언 등 임의의 타입의 데이터를 담을 수 있다.
- DataFrame은 dictionary 타입의 데이터로부터 만들 수 있다
  - 인덱싱 번호 : 자동으로 부여, 0부터 시작
  - column의 배치 : 알파벳순으로 정렬

```
import pandas as pd
import numpy as np

dic = {'city': ['서울', '부산', '대전', '대구', '광주'],
'year': [2017, 2017, 2018, 2018, 2018],
'temp': [18, 20, 19, 21, 20]}
data = pd.DataFrame(dic)
print(data)                                    ##          city temp year
                                                          0 서울 18 2017
                                                          1 부산 20 2017
                                                          2 대전 19 2018
                                                          3 대구 21 2018
                                                          4 광주 20 2018
```

DataFrame – column 순서 변경

- column의 순서를 바꾸려면
  - ➢ 원하는 순서로 된 리스트로 만들어서 인자로 주면 된다

```
data[['year', 'city', 'temp']]
```

	year	city	temp
0	2017	서울	18
1	2017	부산	20
2	2018	대전	19
3	2018	대구	21
4	2018	광주	20

DataFrame – index

- index를 임의의 이름으로 지정할 수 있다

data.index = ['a', 'b', 'c', 'd', 'e'] ; data

	city	year	temp
a	서울	2017	18
b	부산	2017	20
c	대전	2018	19
d	대구	2018	21
e	광주	2018	20

DataFrame – column 이름 변경

- column 이름을 변경할 수 있다.

```
data.columns = ['도시','연도','날씨']; data
```

	도시	연도	날씨
a	서울	2017	18
b	부산	2017	20
c	대전	2018	19
d	대구	2018	21
e	광주	2018	20

DataFrame – 특정 column 또는 row의 내용

- DataFrame에서 특정한 column(열)의 내용만 얻는 방법 : 2가지
  - column 이름을 [ ] 내의 인자로 지정하는 방법
  - column 이름을 속성 값으로 취급하여 "." 연산을 이용하는 방법

    - column 이름으로 접근 : data['연도']
    - 속성(attribute) 값으로 접근 : data.연도

- DataFrame에서 특정 row(행)을 얻는 방법 : 2 가지
  - index를 사용하는 방법 : loc()

    data.loc['b']

  - 행(row)의 위치를 지정하는 방법 : iloc()
    - ✓ ":" 사용하여 행의 범위를 지정할 수도 있다

    data.iloc[2]
    data.iloc[1:3]

DataFrame – index 변경

- index를 임의의 column으로 재배정할 수 있다.

```
data.set_index(['도시'], inplace=True)
data
```

	연도	날씨
도시		
서울	2017	18
부산	2017	20
대전	2018	19
대구	2018	21
광주	2018	20

DataFrame – 새로운 column 추가

- 새로운 컬럼을 추가하려면
  - ➢ 현재 없는 column명을 인자로 주면, 새로운 column이 자동으로 생성

```
cars = [50, 40, 20, 30, 10]
data['car'] = cars
data
```

도시	연도	날씨	car
서울	2017	18	50
부산	2017	20	40
대전	2018	19	20
대구	2018	21	30
광주	2018	20	10

DataFrame – 특정 조건에 맞는 항목 찾기

- 특정 조건에 맞는 항목을 찾고 True/False로 표시된 column 추가

```
data['high'] = data.car >= 30
data
```

도시	연도	날씨	car	high
서울	2017	18	50	True
부산	2017	20	40	True
대전	2018	19	20	False
대구	2018	21	30	True
광주	2018	20	10	False

DataFrame – 특정 column 삭제

- drop() : 특정 컬럼을 삭제
  - ➢ 인자 = 1 : column(열)을 기준으로 삭제

```
data.drop(['car',  'high'], 1)
```

도시	연도	날씨
서울	2017	18
부산	2017	20
대전	2018	19
대구	2018	21
광주	2018	20

Series

- Series 객체 : column이 하나뿐인 DataFrame
  - ➢ 즉, DataFrame의 특수한 형태라 할 수 있다

```
region = pd.Series(['서울', '부산', '대전', '대구', '광주'],
                                index=['1','2','3','4','5'])
print(region)

##
      1 서울
      2 부산
      3 대전
      4 대구
      5 광주
```

➢ 시각화 – matplotlib

matplotlib 개요

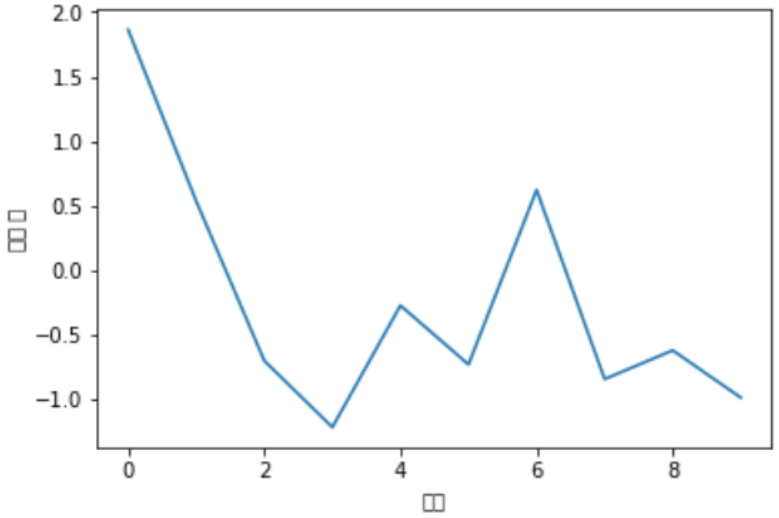
- 파이썬의 시각화 라이브러리
- 시각화 기능
  - 라인 플롯(line plot)
  - 스캐터 플롯(scatter plot)
  - 컨투어 플롯(contour plot)
  - 서피스 플롯(surface plot)
  - 바 차트(bar chart)
  - 히스토그램(histogram)
  - 박스 플롯(box plot)
- matplotlib를 사용한 시각화 예제를 보려면 갤러리 웹사이트를 방문
  - http://matplotlib.org/gallery.html

시각화 – matplotlib

matplotlib – 간단한 예

- 10개의 랜덤한 숫자를 만들고 이들을 선으로 연결하는 코드
  - %matplotlib inline : graph를 현재 jupyter notebook 화면에 직접 그린다

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

n = 10
y = np.random.randn(n)

plt.plot(range(n), y)
plt.xlabel('시간')
plt.ylabel('랜덤 값')
```



시각화 – matplotlib

matplotlib – 한글 처리 코드

- matplotlib에서 한글이 보이도록 하려면 다음의 코드를 먼저 실행

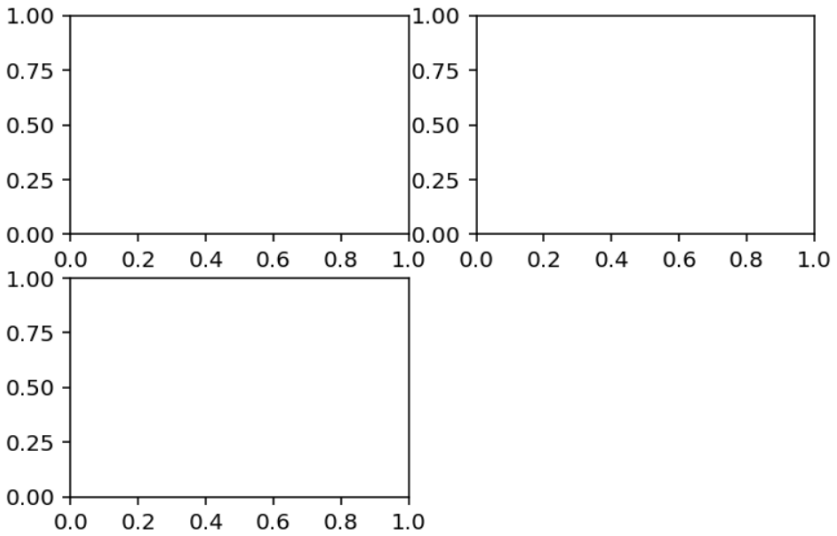
```
import platform
from matplotlib import font_manager, rc
import matplotlib

# '-' 부호가 제대로 표시되게 하는 설정
matplotlib.rcParams['axes.unicode_minus'] = False

# 윈도우
if platform.system() == 'Windows':
path = "c:₩Windows₩Fonts₩malgun.ttf"
font_name = font_manager.FontProperties(fname=path).get_name()
rc('font', family=font_name)

# 맥
elif platform.system() == 'Darwin':
rc('font', family='AppleGothic')

# 리눅스
elif platform.system() == 'Linux':
rc('font', family='NanumBarunGothic')
```

시각화 – matplotlib

Graph 생성

- 2*2 격자에서 1, 2, 3 번째 그림 객체를 얻는 방법

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```

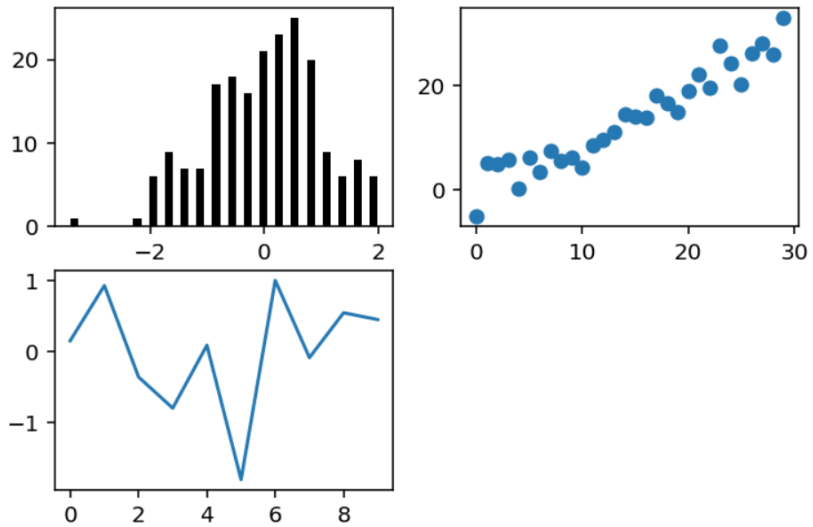


시각화 – matplotlib

Graph 생성

- Histogram, scatter plot, plot 그래프 생성

```
ax1.hist(np.random.randn(200), bins=20, color='k', rwidth=0.5)
ax2.scatter(np.arange(30), np.arange(30) + 3*np.random.randn(30))
ax3.plot(np.arange(10),np.random.randn(10))
fig
```

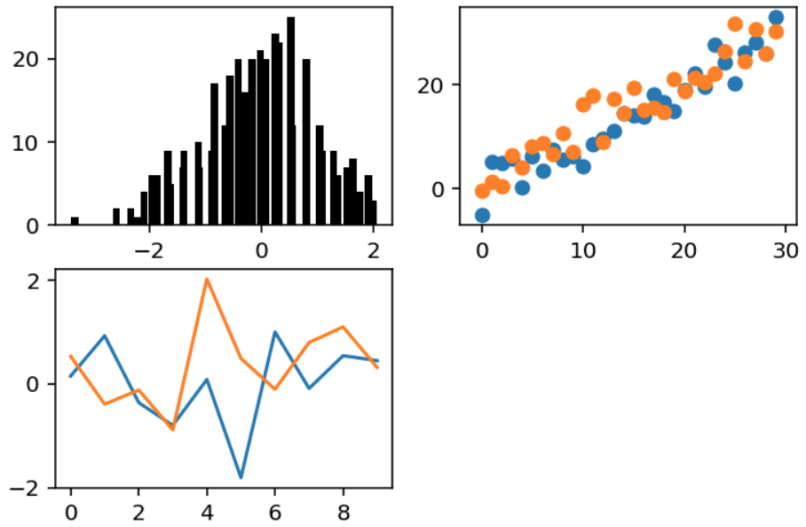


시각화 – matplotlib

Graph 생성

- 같은 코드를 두 번 실행하면 동일 그래프에 계속 겹쳐서 그려진다

```
ax1.hist(np.random.randn(200),  bins=20, color='k', rwidth=0.5)
ax2.scatter(np.arange(30),  np.arange(30)  + 3*np.random.randn(30))
ax3.plot(np.arange(10),np.random.randn(10))
fig
```

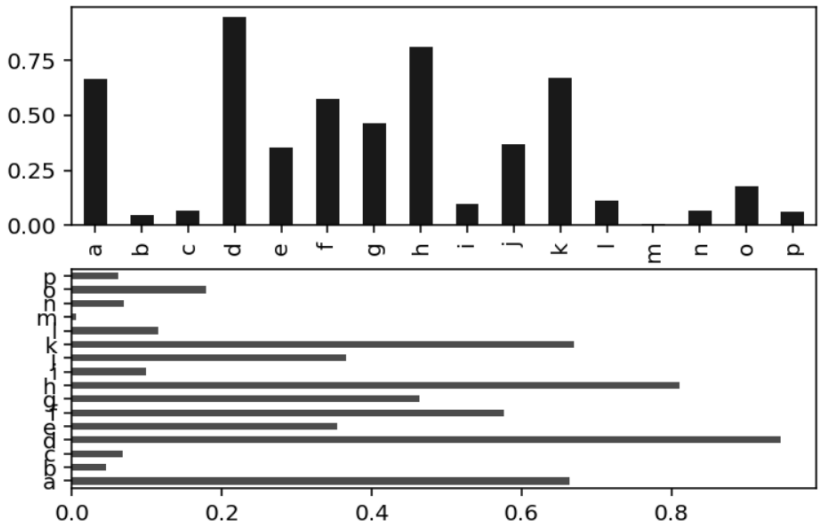


시각화 – matplotlib

Graph 생성

- bar graph를 subgraph를 이용하여 그리는 예

```
fig, axes = plt.subplots(2, 1)
data = Series(np.random.rand(16), index=list('abcdefghijklmnop'))
data.plot(kind='bar', ax=axes[0], color='k', alpha=0.9)
data.plot(kind='barh', ax=axes[1], color='k', alpha=0.7)
```



시각화 – matplotlib

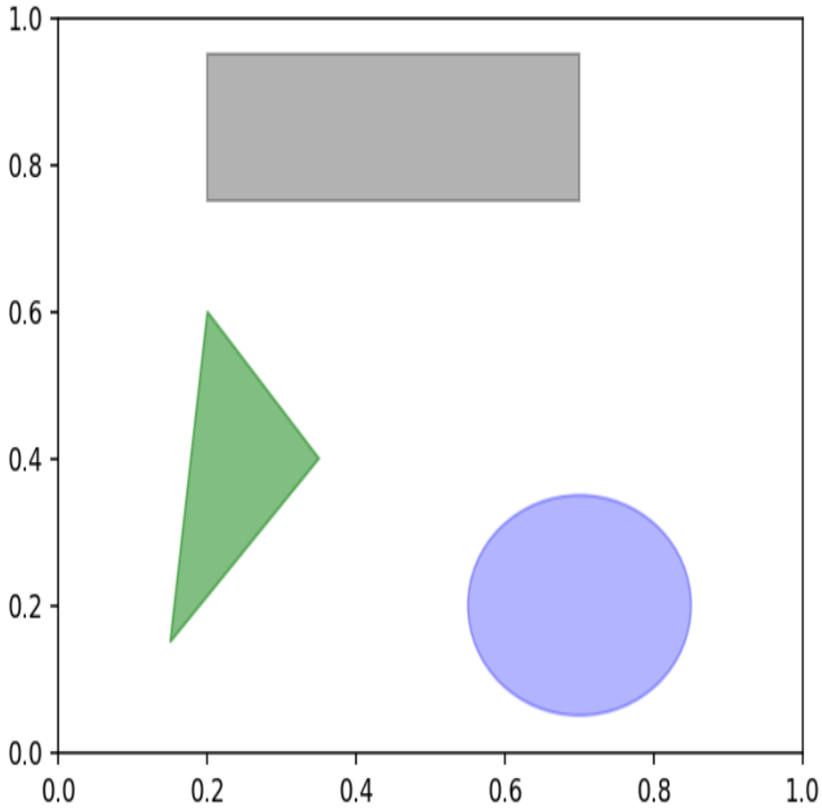
Graph 생성

- 일반 도형을 그리는 예
  - Rectangle : 사각형의 시작점 좌표, 가로 길이, 세로 길이
  - Circle : 원의 중심 좌표, 반지름
  - Polygon : 각 꼭지점의 좌표

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

rect = plt.Rectangle((0.2, 0.75), 0.5, 0.2, color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]], color='g', alpha=0.5)

ax.add_patch(rect)                    ## ax subplot에 도형을 추가
ax.add_patch(circ)
ax.add_patch(pgon)
```

시각화 – matplotlib

Graph 생성

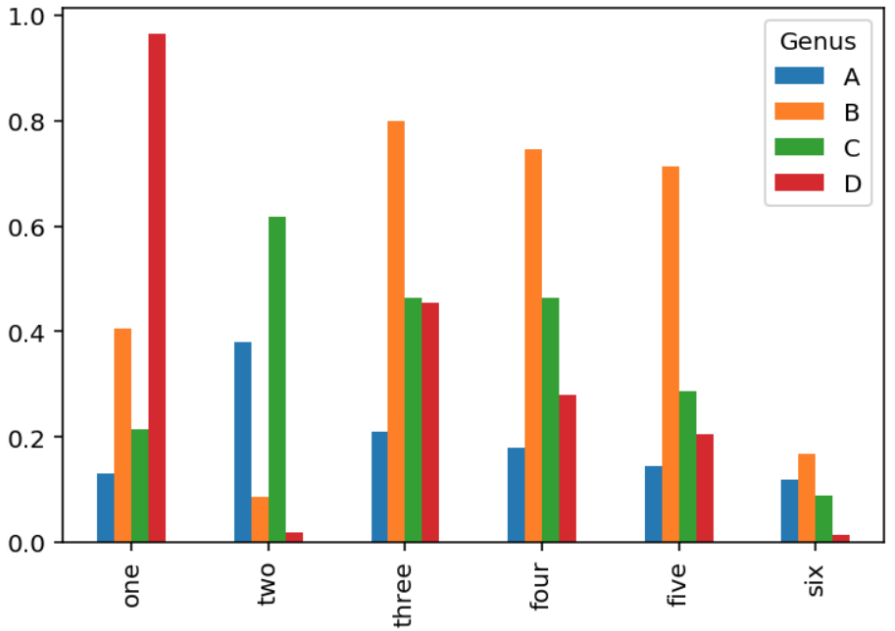


시각화 – matplotlib

Graph 생성

- DataFrame으로부터 bar graph를 그리는 예

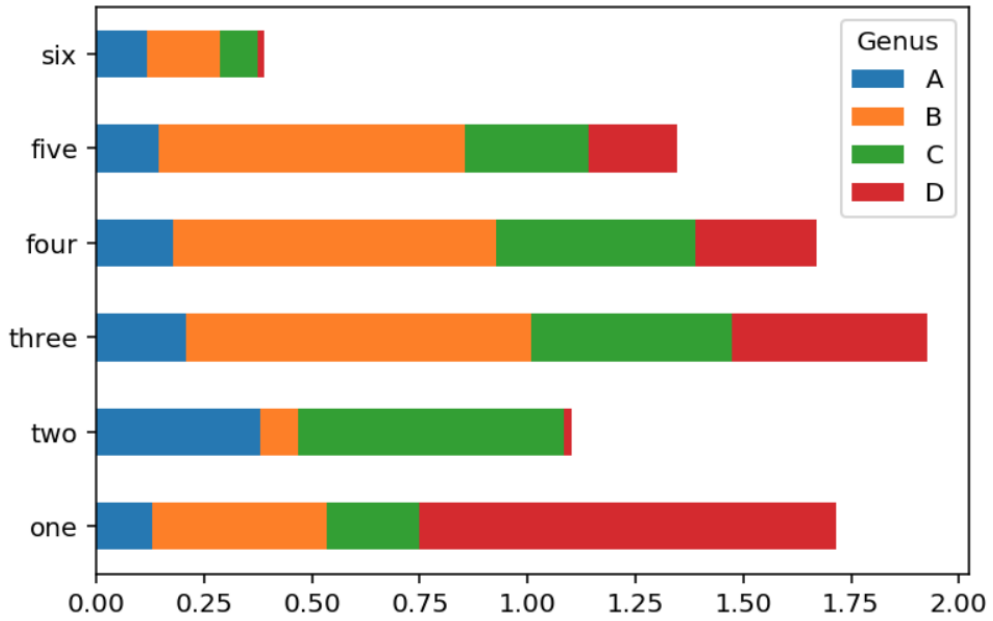
```
df = DataFrame(np.random.rand(6, 4),
               index=['one', 'two', 'three', 'four', 'five', 'six'],
               columns=pd.Index(['A', 'B', 'C', 'D'],
               name='Genus'))
df.plot(kind='bar')
```



시각화 – matplotlib

Graph 생성

df.plot(kind='barh', stacked=True)



Data visualization - matplotlib

- Use:
  - %matplotlib inline magic command (once Jupyter is open)
  - import matplotlib.pyplot as plt
- Basic template
  - Create a new figure
    - fig = plt.figure()
    - fig = plt.figure(figsize = (12,8))
  - Add subplots (if necessary)
    - ax1 = fig.add_subplot(2,1,1) # 2x1 arrangement, first figure
    - ax2 = fig.add_subplot(2,1,2)
  - Create plot (plt or ax1...axN methods)
  - Label, annotate, format plot
  - Copy or save plot

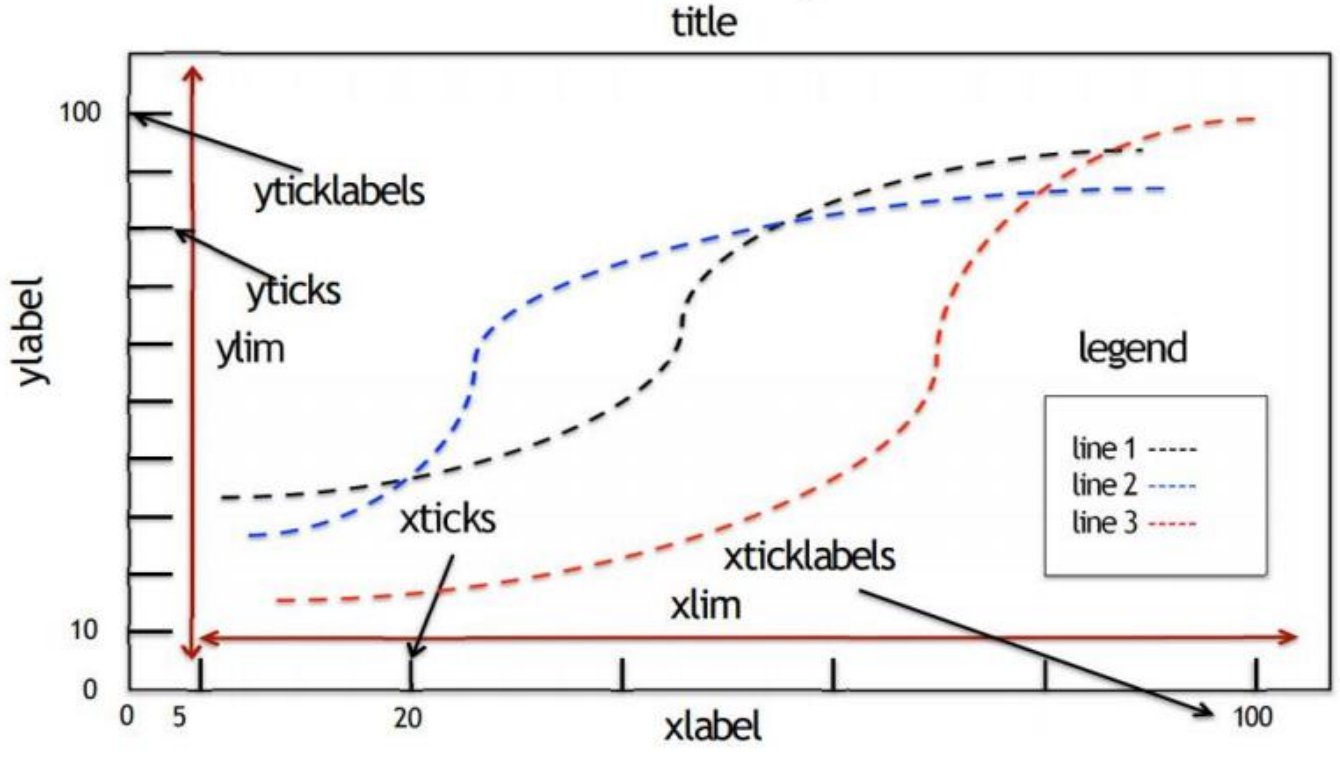
Matplotlib - Common plot types

- Line plots – trends:
  - plt.plot (x, y, '-')
- Scatter plots – comparison between lots of data
  - plt.plot (x, y, '.')
- Bar plots – comparison between few data
  - Bar (horizontal): plt.barh (x, y, width)
  - Column (vertical): plt.bar (x, y, width)
- Histogram plots – single distributions
  - plt.hist (x, bins)
- Boxplots – one or more distributions
  - plt.boxplot (x)

Matplotlib - Colors, Markers, and Line Styles

- All specified as special string characters in plot call
- Colors - Many plot types
  - Basic colors: g(reen), r(ed), b(lue), (blac)k, m(agenta), y(ellow), c(yan), w(hite)
  - For more, see http://matplotlib.org/api/colors_api.html
- Markers and Line Styles - Mostly relate to plt.plot
  - Markers: ., o, +, * (star), 1, 2, 3, 4 (triangles), s(quare), D(iamond)
  - Line styles: solid (-), dashed (--), dotted (:), dash-dot (-.)
  - linewidth keyword (float value)
- Usage
  - Style string: Combines all three (e.g., 'k.', 'g--', 'ro-')
  - Separate keyword arguments: color, linestyle, marker

Formatting plots



Formatting plots

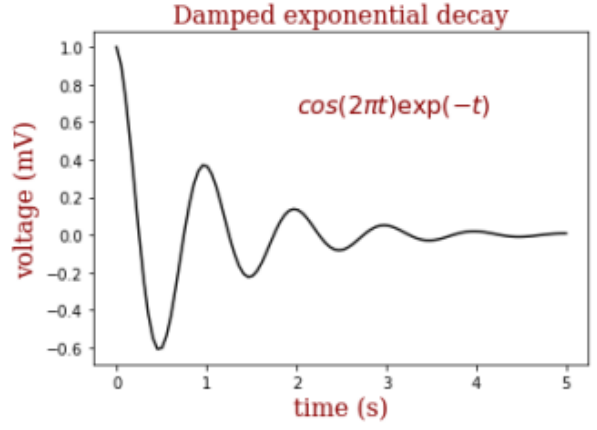
- Title
  - title('Title')
- Axis labels
  - xlabel ('Time'), ylabel ('Price)
- Axis limits
  - xlim([0,10]0, ylim
- Ticks
  - xticks([0,60,70,80,90,100]), yticks
- Tick labels – combine with ticks for text labels
  - xticklabels(['F','D','C','B','A']), yticklebals
- Lagends
  - List of labels for each series: legend(('one','two','three'))
  - Use legend()
  - Location keyword: loc = 'best', 1-10 (upper right, left, center, etc.)

Annotating plots

- Text
  - text(x, y, text, fontsize)
  - arrow(x, y, dx, dy) # draws arrow from (x,y) to (x+dx, y+dy)
  - annotate (text, xy, xytext) # annotate the xy point with text positioned at xytext
- shapes
  - Rectangles, circles, polygons
  - Location, size, color, transparency (alpha)

Matplotlib - Example(1)

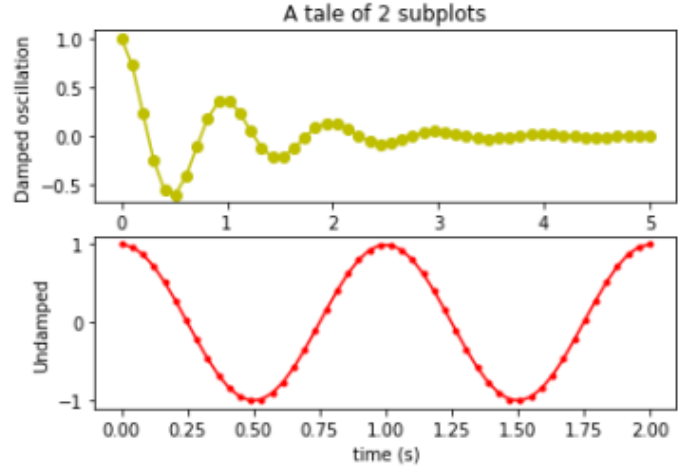
In [27]:
```
x = np.linspace(0.0,5.0,100)
y = np.cos(2*np.pi*x) * np.exp(-x)

plt.plot (x,y,'k')
plt.title('Damped exponential decay', fontdict=font)
plt.text(2, 0.65, r'$cos(2 \pi t) \exp(-t)$', fontdict=font)

plt.xlabel('time (s)', fontdict=font)
plt.ylabel('voltage (mV)', fontdict=font)

plt.subplots_adjust(left=0.15)
```



Matplotlib - Example(2)

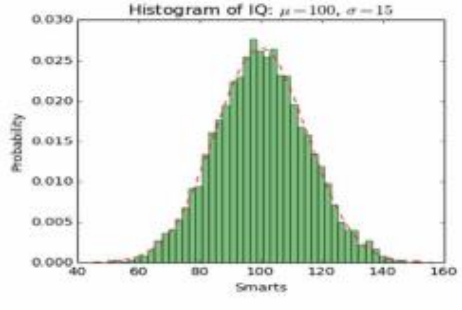
In [39]:
```
x1 = np.linspace(0.0,5.0)
x2 = np.linspace(0.0,2.0)
y1 = np.cos(2*np.pi*x1) * np.exp(-x1)
y2 = np.cos(2* np.pi* x2)

plt.subplot(2, 1, 1)
plt.plot(x1,y1,'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2)
plt.plot(x2, y2,'r.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
```

Out [39]: Text(0, 0.5, 'Undamped')

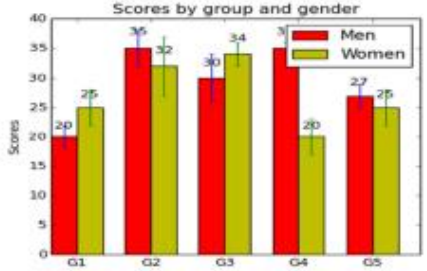


Many more examples...

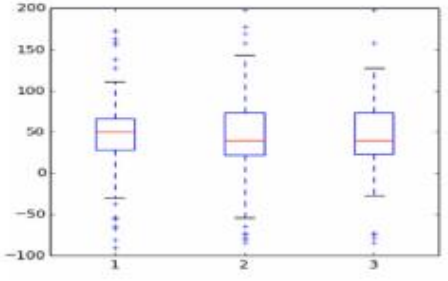
Histogram



Bar Chart (with error bars and legend)



Boxplots



Scatter + Histogram