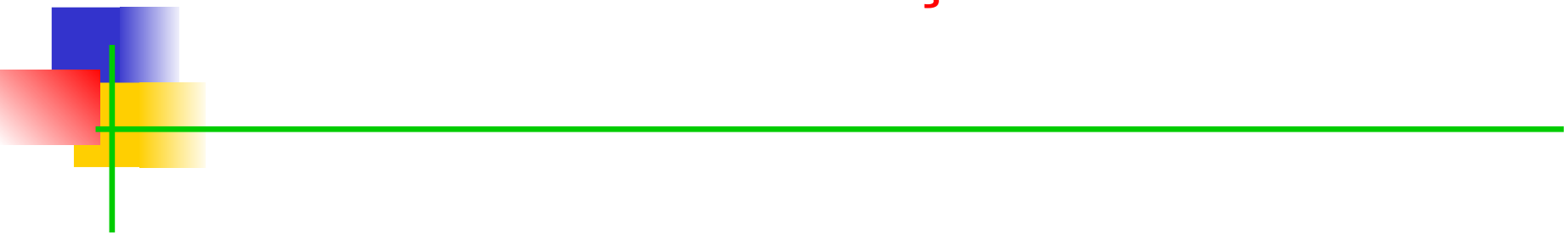
BLM442 Büyük Veri Analizine Giriş

Görselleştirme



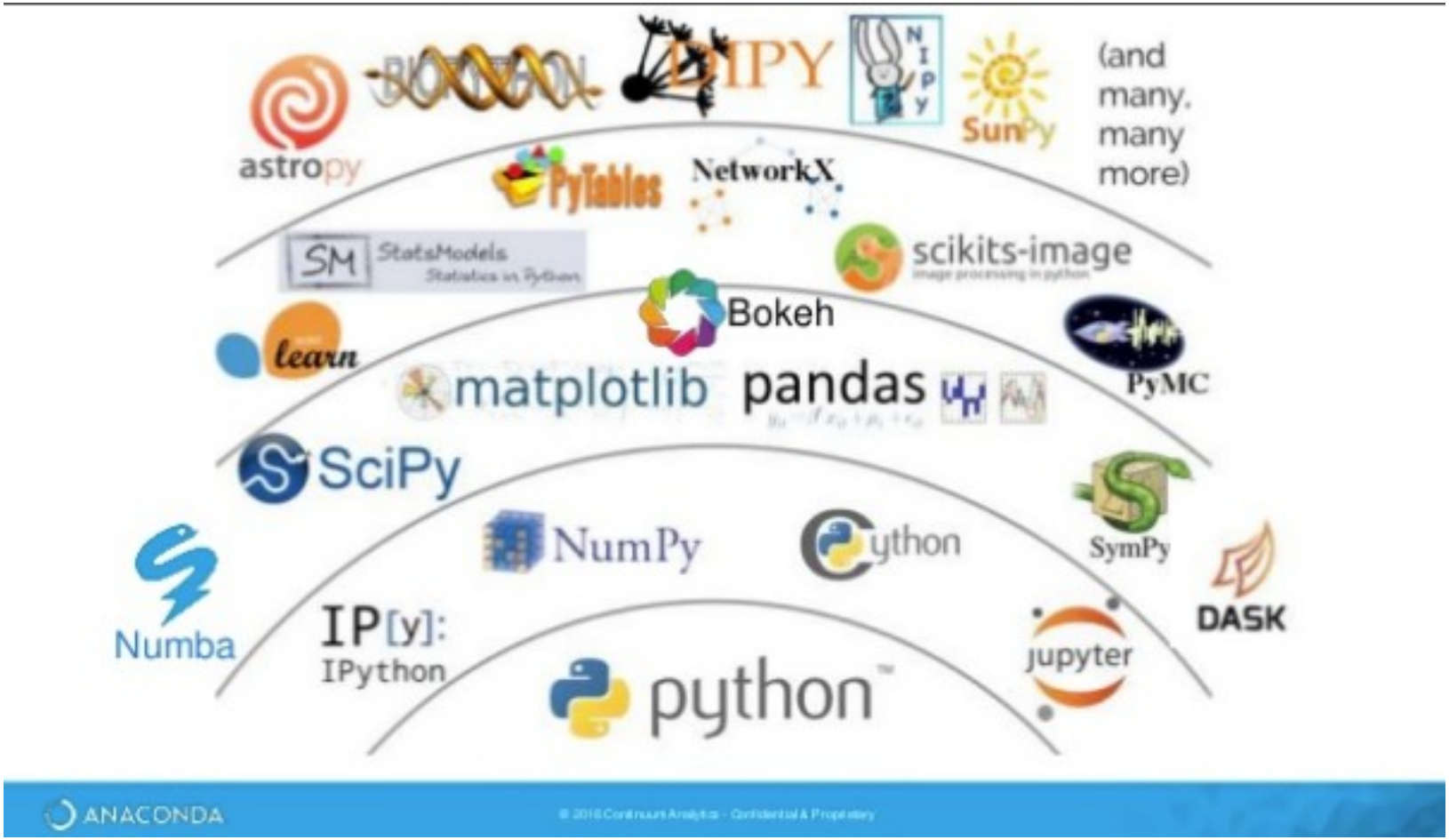
Dr. Süleyman Eken

Bilgisayar Mühendisliği
Kocaeli Üniversitesi

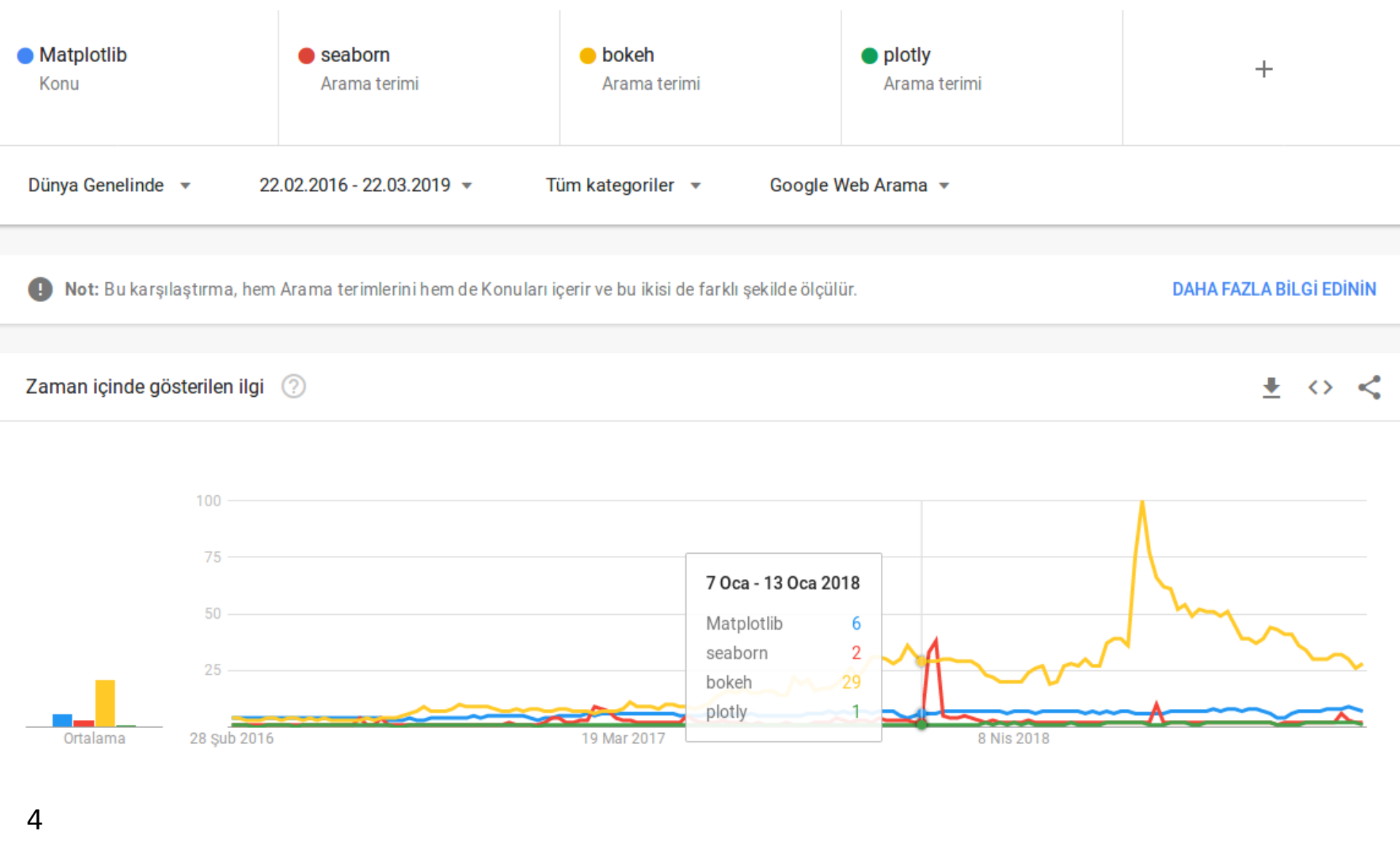
Sunum Planı

- Python'da veri görselleştirme
- ggplot
- matplotlib
- seaborn, bokeh, plotly
- Folium
- Uygulamalar

Veri bilimi için Python kütüphaneleri



Veri bilimi için görsellestirme kütüphaneleri



Python kütüphanelerini yükleme

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Verileri keşfetmek için grafikler

	description
distplot	histogram
barplot	sayısal değişken için merkezi eğilim tahmini
violinplot	kutu grafiğine (boxplot) benzer şekilde, verilerin olasılık yoğunluğunu da gösterir
jointplot	Dağılım grafiği (Scatterplot)
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot
pie	pie graphs
kde or density	density plots
area	area plots

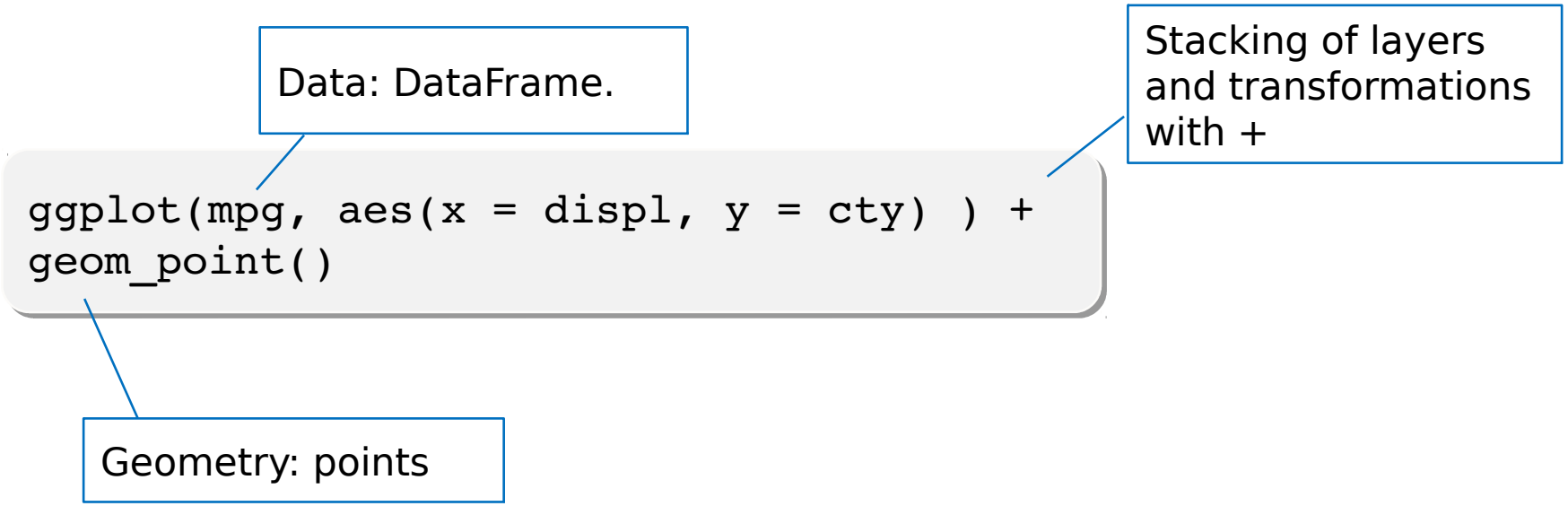
Hangi paketler / fonksiyonlar?

- Standart grafikler (ör. Çizgi grafik, çubuk grafik, dağılım grafiği):
  - Matplotlib, Seaborn, ggplot, Altair, ...
- Tematik haritalar
  - Folium, Basemap, Cartopy, İris,…
- Diğer görselleştirmeler
  - Bokeh (etkileşimli parseller), plotly,…

ggplot

- En popüler R paketinden (ggplot2)'ne göre tasarlanmıştır.
- Grafik dilbilgisine dayalı (Wilkinson, 2005)
- Grafikler bu gramerlere göre oluşturulmuştur:
  - **data**
  - **mapping / aestetics**
  - **geoms**
  - **stats**
  - **scales**
  - **coord**
  - **Facets**
- Ggplot'da doğal olarak pandas DataFrames kullanılır.

ggplot ve qplot



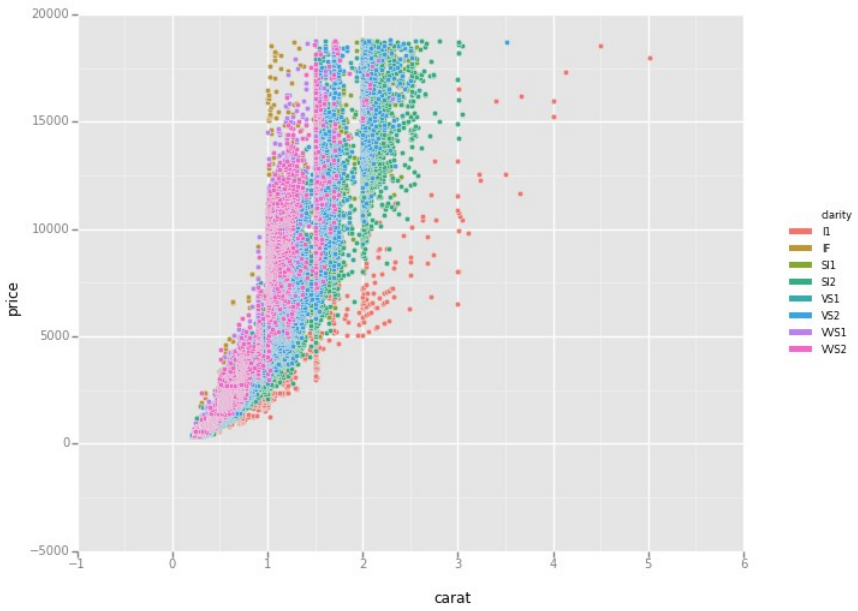
Shortcut function: qplot (quick plot):

```
qplot(diamonds.carat, diamonds.price)
```

Aesthetics

Mapping of data to visual attributes of geometric objects:

- Position: **x**, **y**
- Color: **color**
- Shape: **shape**

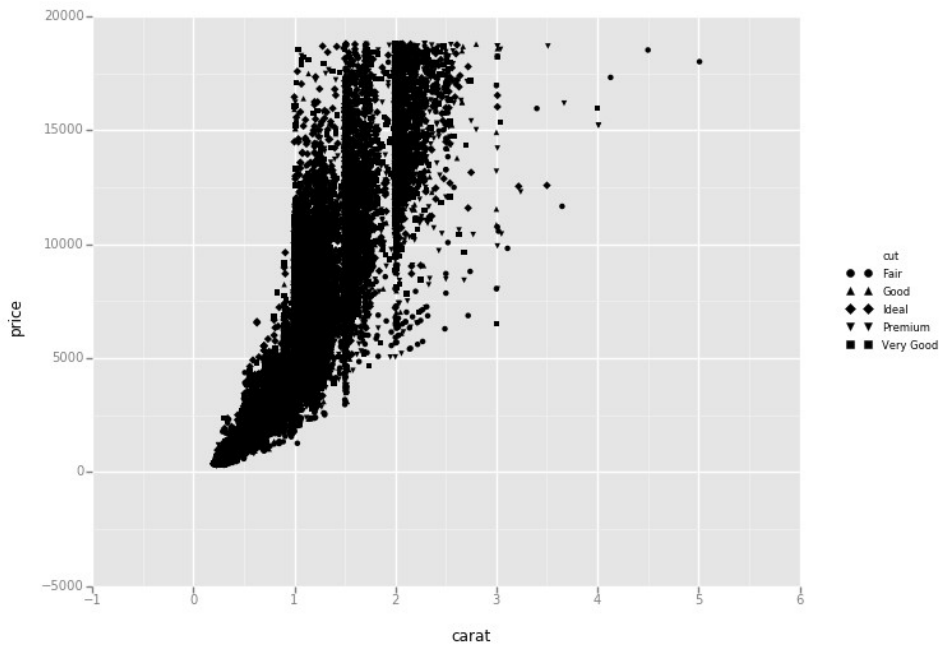



```
ggplot(aes(x='carat', y='price', color='clarity'), diamonds) +
geom_point()
```

Aesthetics

Mapping of data to visual attributes of geometric objects:

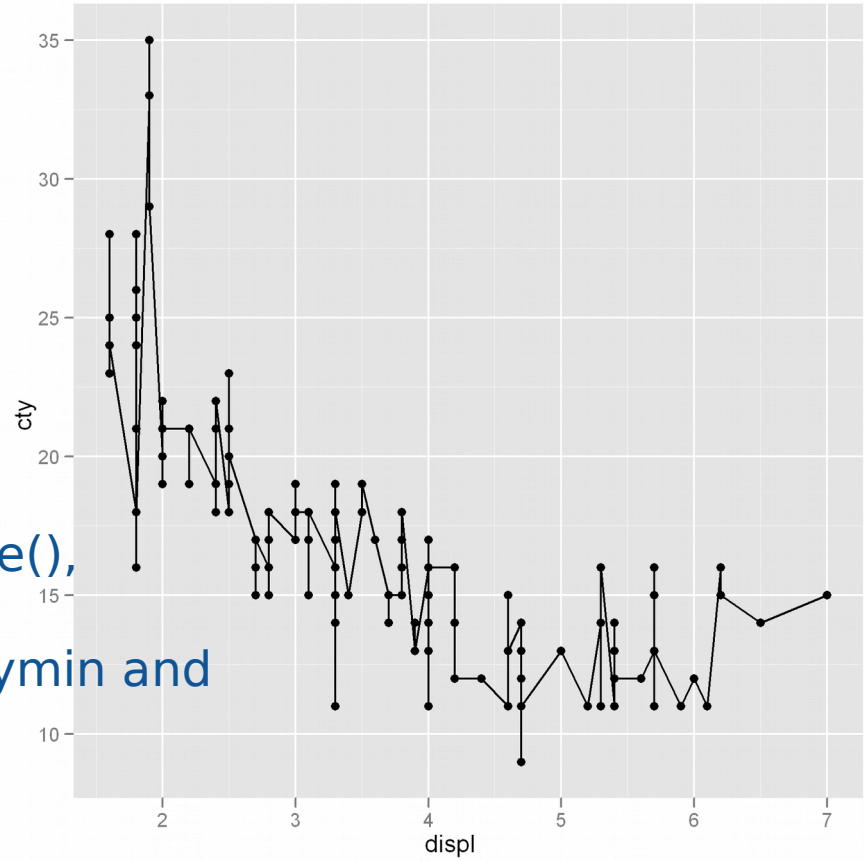
- Position: **x**,**y**
- Color: **color**
- Shape: **shape**




```
ggplot(aes(x='carat', y='price', shape="cut"), diamonds) +
geom_point()
```

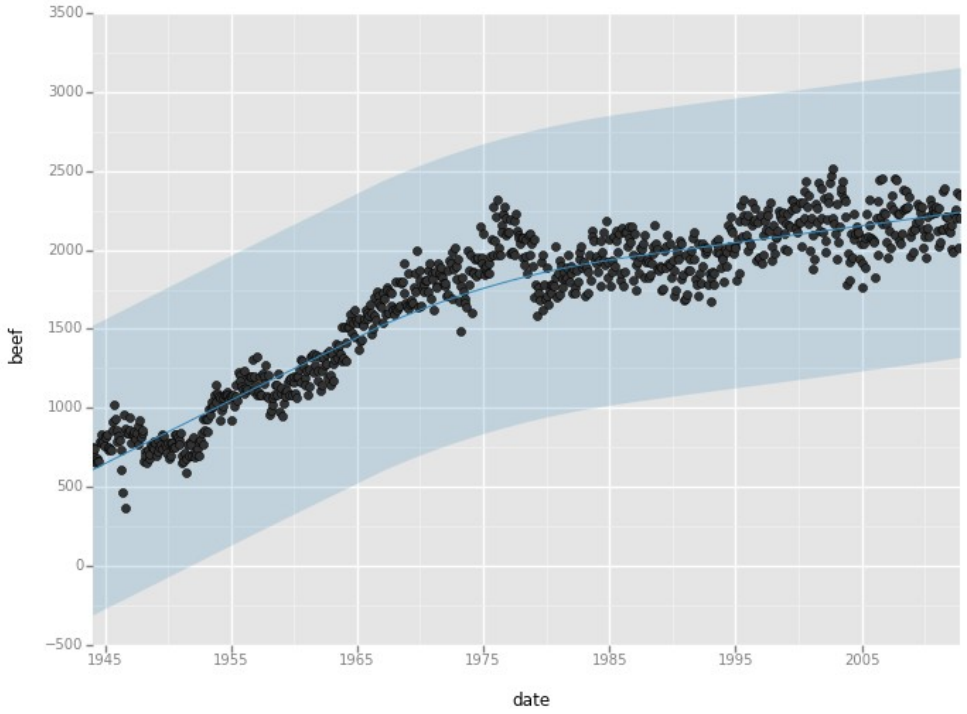
Geom

- Geometric objects:
  - Points, lines, polygons, …
  - Functions start with “geom_”
- Also margins:
  - geom_errorbar(), geom_pointrange(), geom_linerange().
  - Note: they require the aesthetics ymin and ymax.



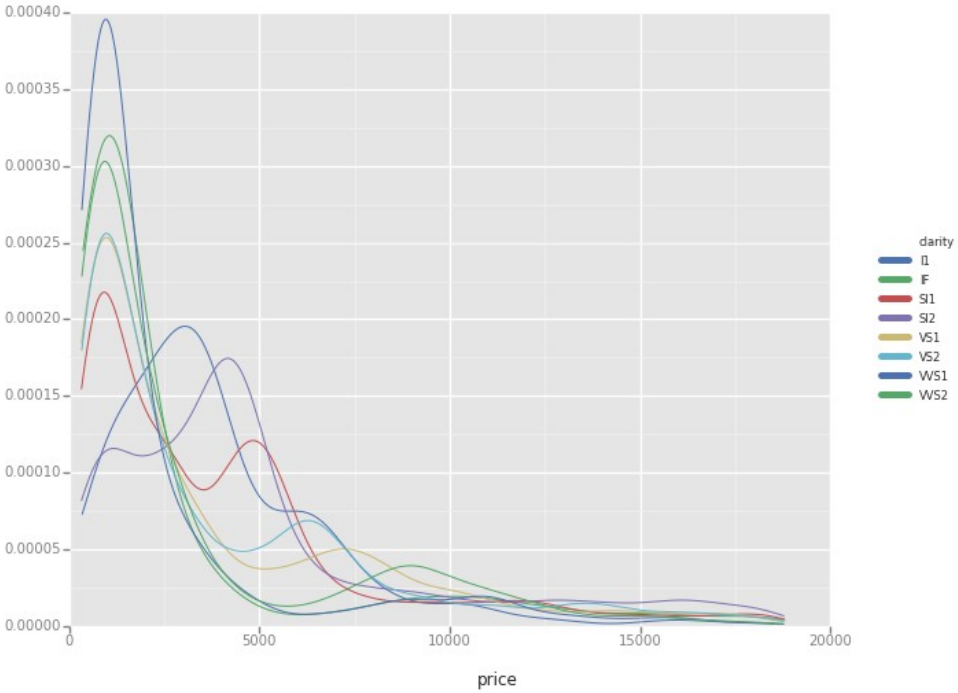

```
ggplot(mpg, aes(x = displ, y = cty)) +
geom_point() + geom_line()
```

stat_smooth




```
ggplot(aes(x='date', y='beef'), data=meat) + geom_point() + \
    stat_smooth(method='loess')
```

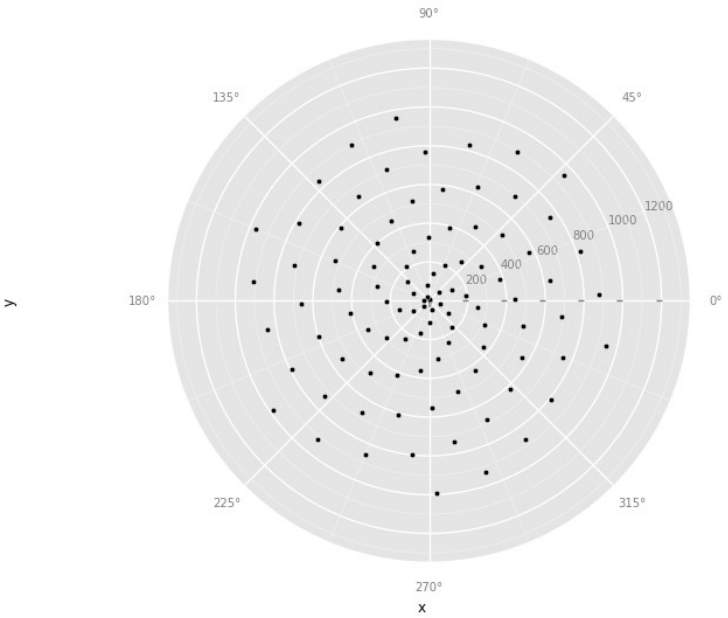
stat_density




```
ggplot(aes(x='price', color='clarity'), data=diamonds) + stat_density()
```

Coord

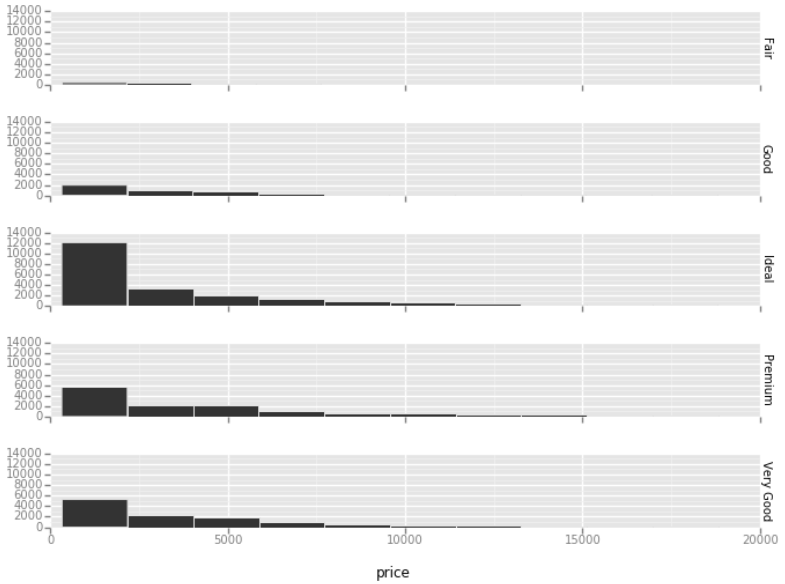
- A chart is drawn in a coordinate system. This can be transformed.
- A pie chart has a polar coordinate system.




```
df= pd.DataFram e({"x":np.arange(100)})
df['y'] = df.x * 10 # polar coords
p = ggplot(df,aes(x= 'x',y= 'y'))+ geom _point()+ coord_polar()print(p)
```

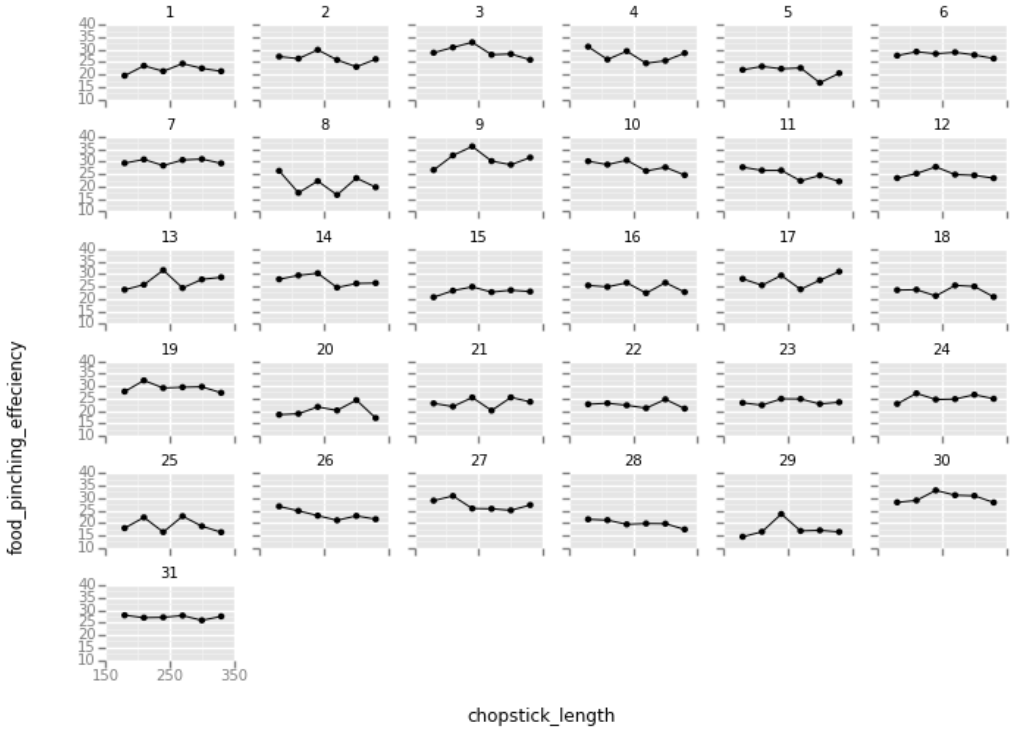
Facets

- With facets, small multiples are created.
- Each facet shows a subset of the data.



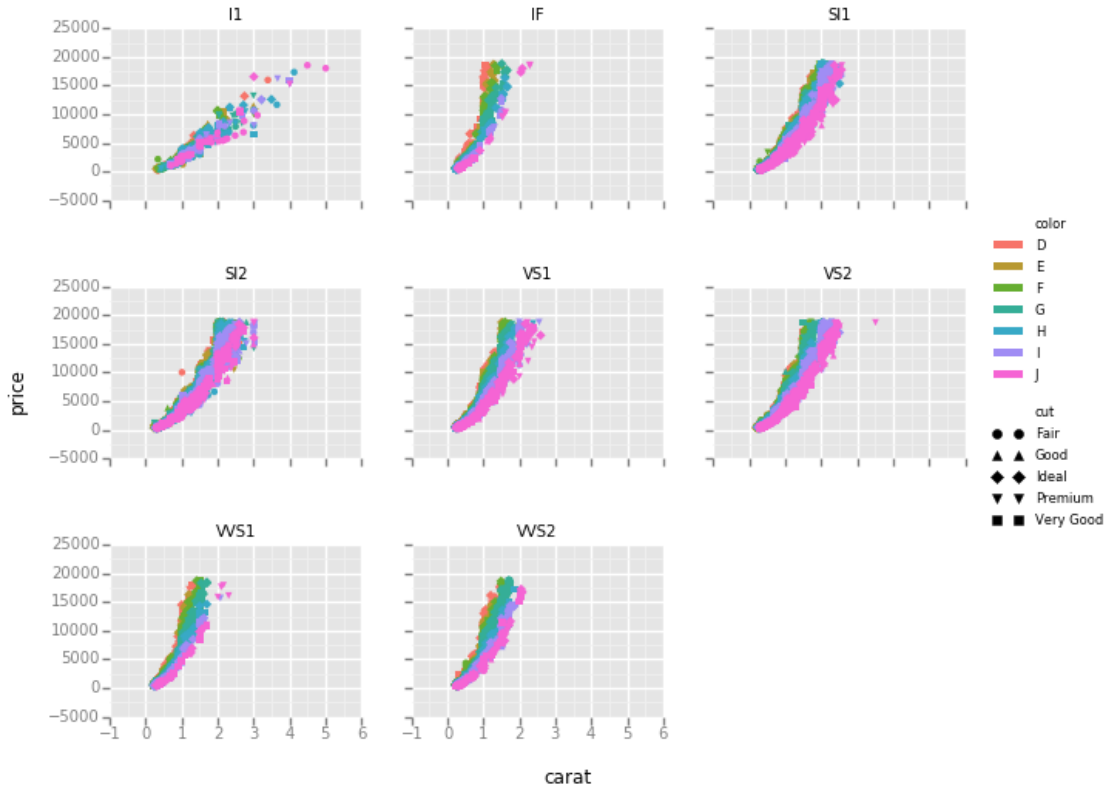

```
ggplot(diamonds, aes(x='price')) + \
geom_histogram() + \
facet_grid("cut")
```

Facets örnek




```
ggplot(chopsticks, aes(x='chopstick_length',
y='food_pinching_effeciency')) + \
geom_point() + \
geom_line() + \
scale_x_continuous(breaks=[150, 250, 350]) + \
facet_wrap("individual")
```

Facets örnek 2




```
ggplot(diamonds, aes(x="carat", y="price", color="color",
shape="cut")) + geom_point() + facet_wrap("clarity")
```


18

ggplot tips

- You can annotate plots

```
ggplot(mtcars, aes(x='mpg')) + geom_histogram() + \
      xlab("Miles per Gallon") + ylab("# of Cars")
```

- Assign a plot to a variable, for instance g:

```
g = ggplot(mpg, aes(x = displ, y = cty)) +
geom_point()
```

- The function save saves the plot to the desired format:

```
g.save(“myimage.png”)
```

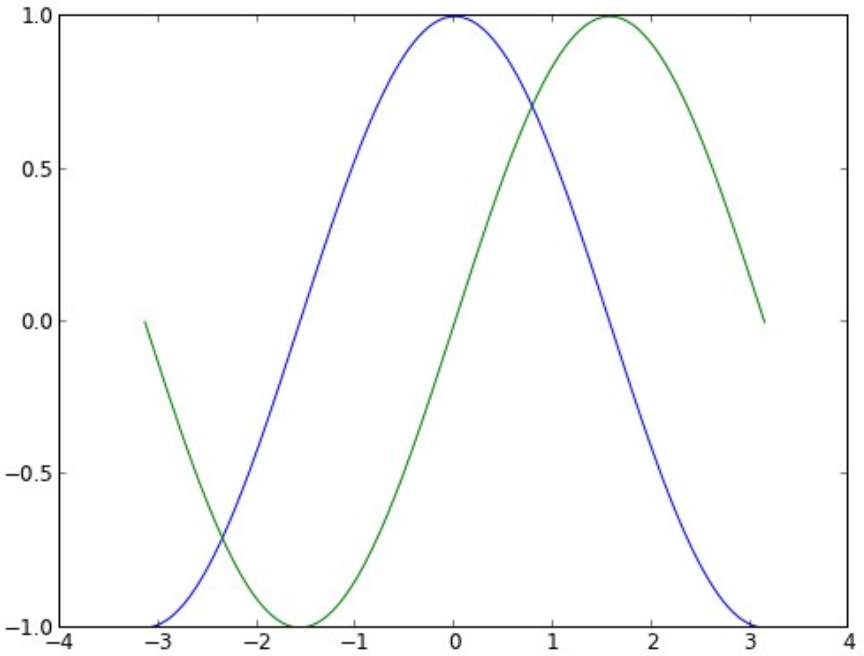
matplotlib

- matplotlib, 2D grafikler için en çok kullanılan Python paketidir.
- IPython and the pylab mode
  - Bu mode Matlab / Mathematica benzeri işlevselliğe sahip etkileşimli matplotlib oturumlarına izin verir. Daha cok etkilesimli olmayan OO (object oriented) arayuzu onerilmektedir.
- pyplot
  - Bu mode, matplotlib nesne yönelimli çizim kütüphanesine uygun bir arayüz sağlar.
- Matplotlib, her tür özelliği özelleştirmenize izin veren bir dizi varsayılan ayar ile birlikte gelir. Matplotlib'deki hemen hemen her özelliğin varsayılan ayarlarını kontrol edebilirsiniz: şekil boyutu ve dpi, çizgi genişliği, renk ve stil, eksenler, eksen ve ızgara özellikleri, metin ve yazı tipi özellikleri vb.

Simple plot

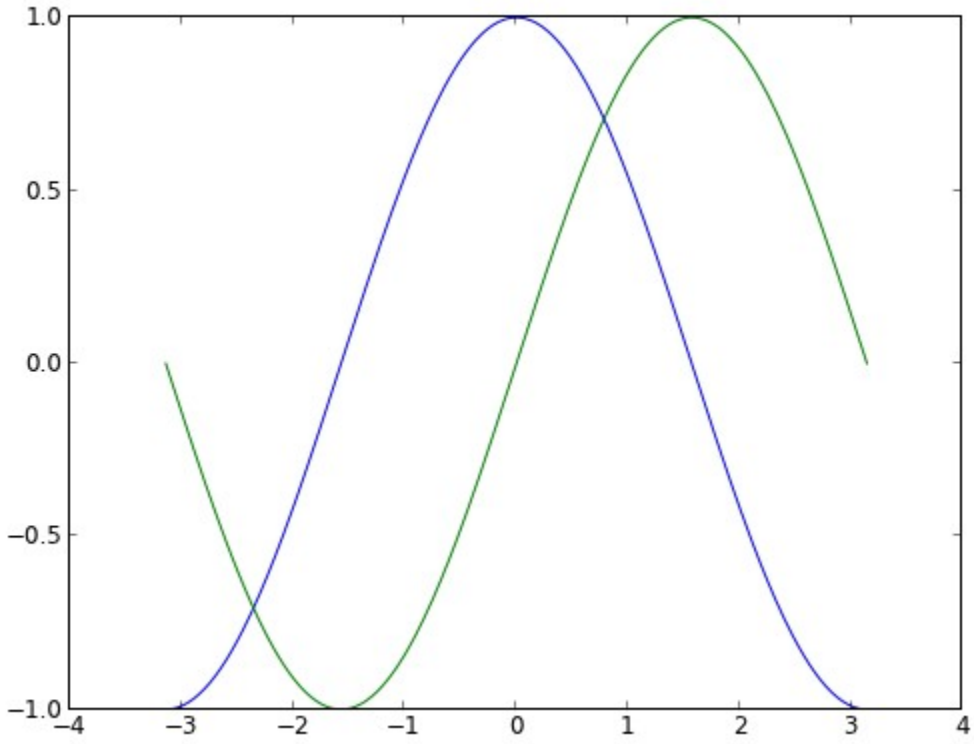
- import numpy as np
- import matplotlib.pyplot as plt

- X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
- C,S = np.cos(X), np.sin(X)
-
- plt.plot(X,C)
- plt.plot(X,S)
-
- plt.show()



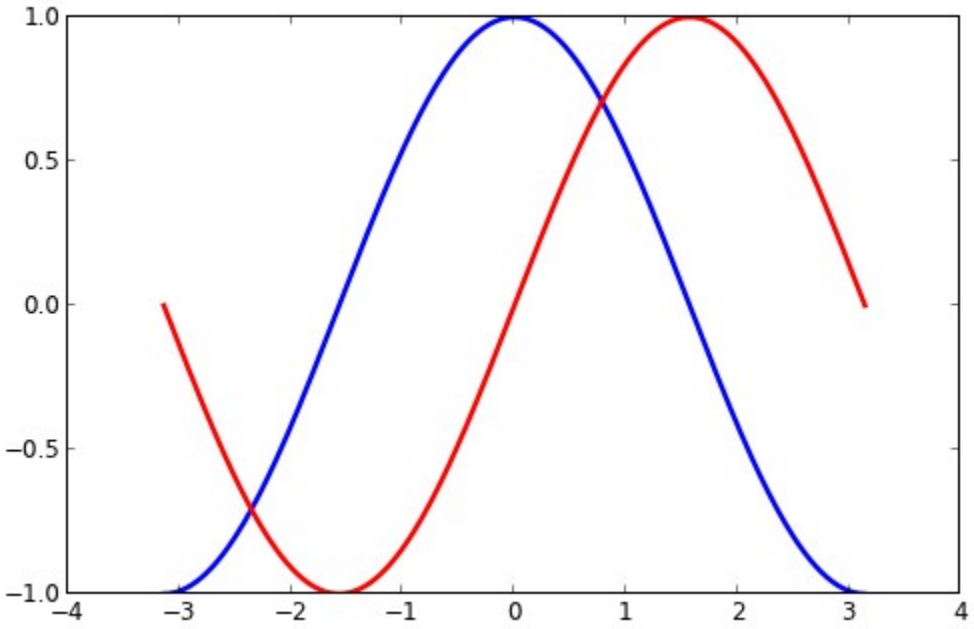
Instantiating defaults

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
# Create a new figure of size 8x6 points, using 100 dots per
plt.figure(figsize=(8,6), dpi=80)
# Create a new subplot from a grid of 1x1
plt.subplot(111)
X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C,S = np.cos(X), np.sin(X)
# Plot cosine using blue color with a continuous line of width
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
# Plot sine using green color with a continuous line of width
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")
# Set x limits
plt.xlim(-4.0,4.0)
# Set x ticks
plt.xticks(np.linspace(-4,4,9,endpoint=True))
# Set y limits
plt.ylim(-1.0,1.0)
# Set y ticks
plt.yticks(np.linspace(-1,1,5,endpoint=True))
# Save figure using 72 dots per inch
# savefig("../figures/exercice_2.png",dpi=72)
# Show result on screen
plt.show()
```



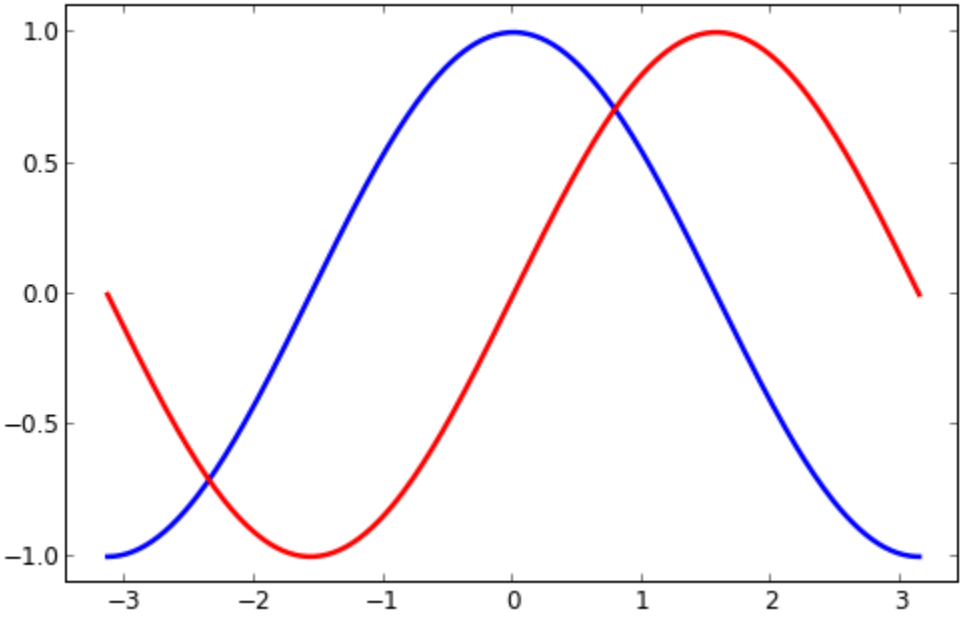
Changing colors and line widths

- ...
- plt.figure(figsize=(10,6), dpi=80)
- plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
- plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
- ...



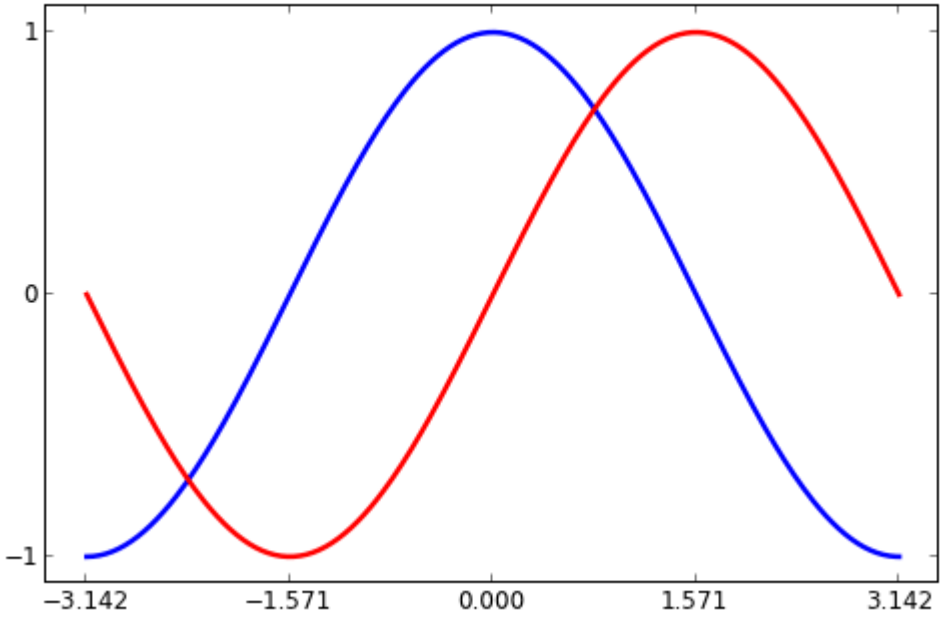
Setting limits

- ...
- plt.xlim(X.min()*1.1, X.max()*1.1)
- plt.ylim(C.min()*1.1, C.max()*1.1)
- ...



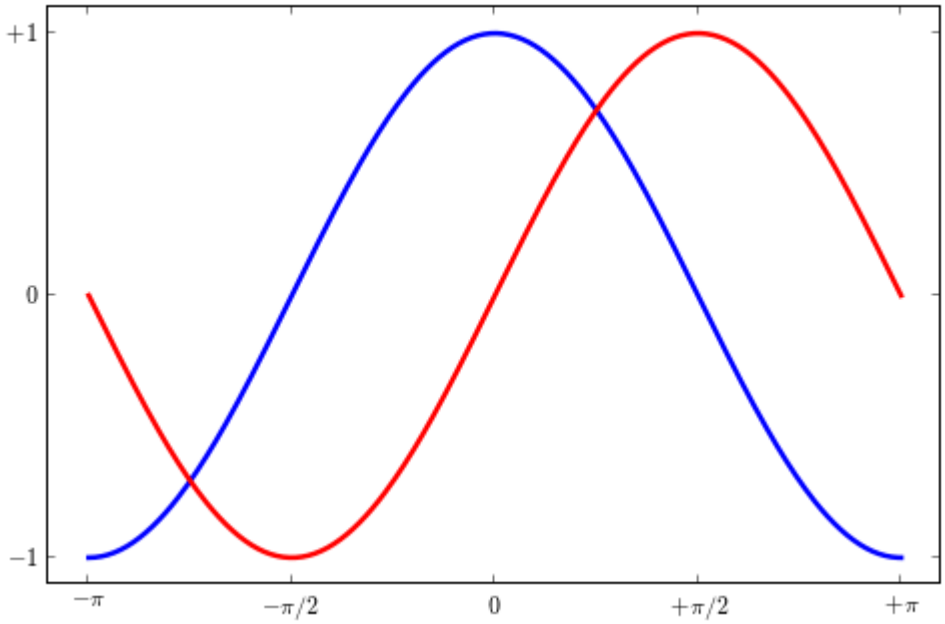
Setting ticks

- ...
- plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
- plt.yticks([-1, 0, +1])
- ...



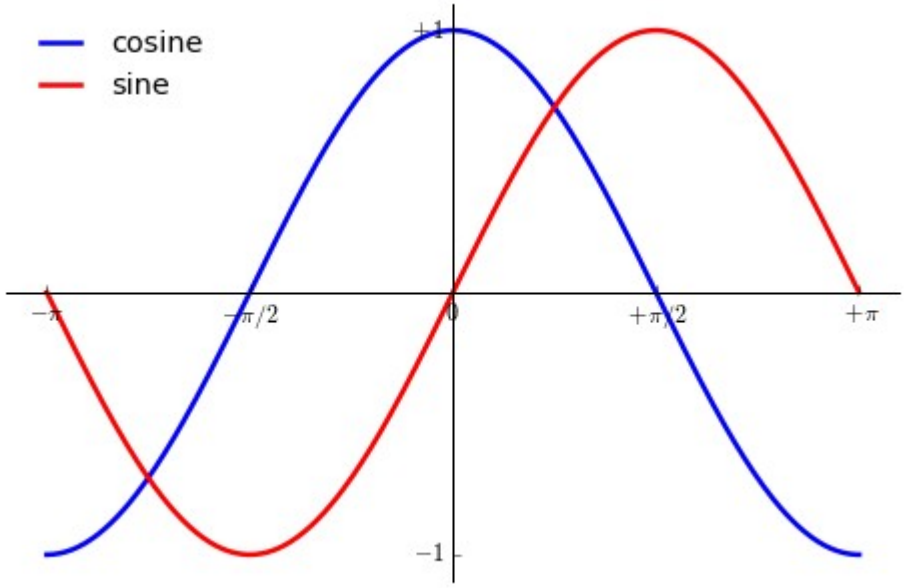
Setting tick labels

- ...
- plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
- [r'$-\pi$', r'$-\pi/2$', r'0', r'$+\pi/2$', r'$+\pi$'])
-
- plt.yticks([-1, 0, +1],
- [r'-1', r'0', r'$+1$'])
- ...



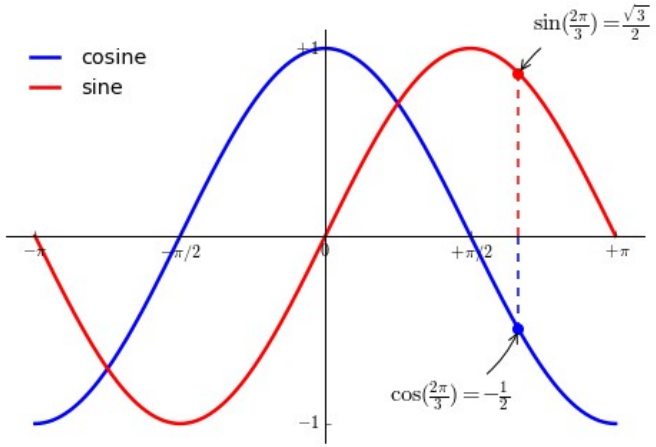
Adding a legend

- ...
- plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosine")
- plt.plot(X, S, color="red", linewidth=2.5, linestyle="-", label="sine")
-
- plt.legend(loc='upper left', frameon=False)
- ...

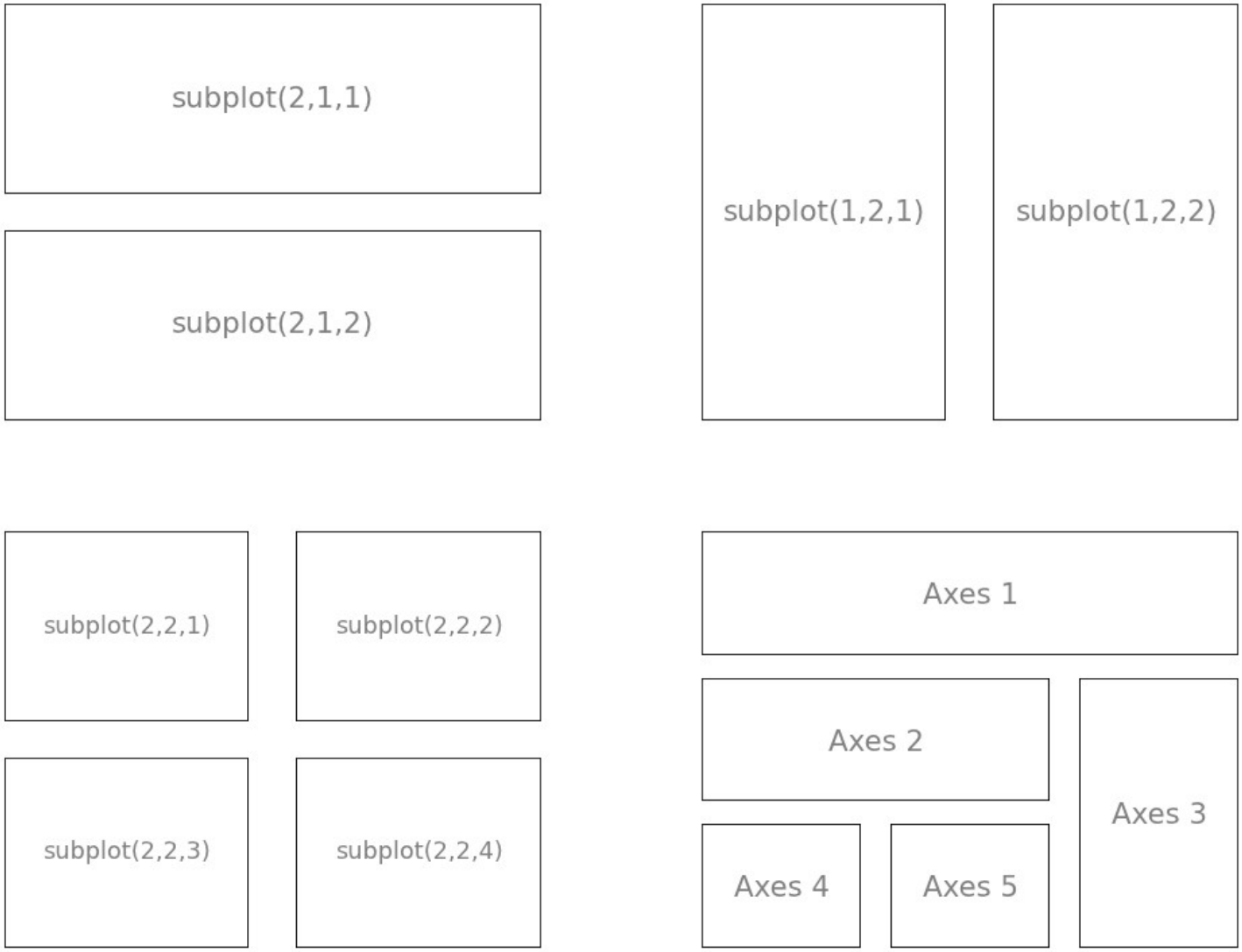


Annotate some points

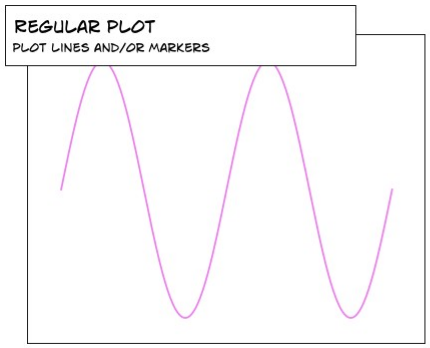
```
...

t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)], color ='blue', linewidth=1.5, linestyle="--")
plt.scatter([t,],[np.cos(t),], 50, color ='blue')

plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)], color ='red', linewidth=1.5, linestyle="--")
plt.scatter([t,],[np.sin(t),], 50, color ='red')

plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
...
```

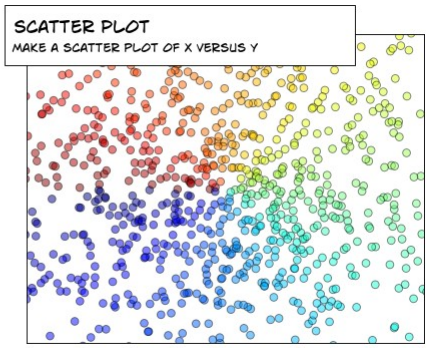


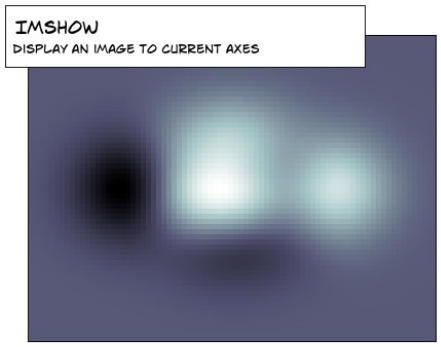
Subplots

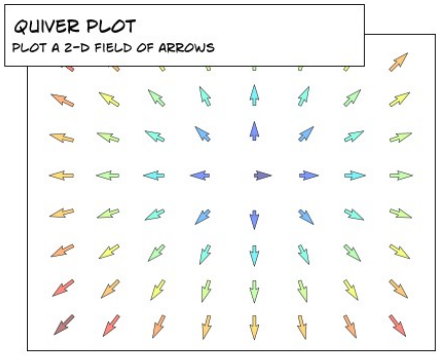

subplot(2,1,1)
subplot(2,1,2)
subplot(1,2,1)
subplot(1,2,2)
subplot(2,2,1)
subplot(2,2,2)
subplot(2,2,3)
subplot(2,2,4)
Axes 1
Axes 2
Axes 3
Axes 4
Axes 5

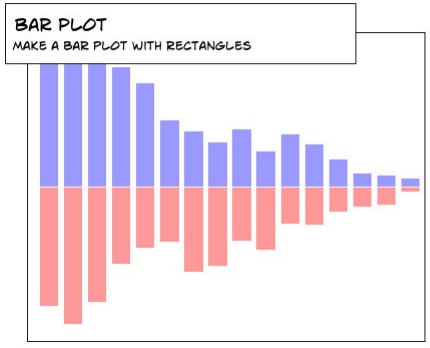
Diğer tipler

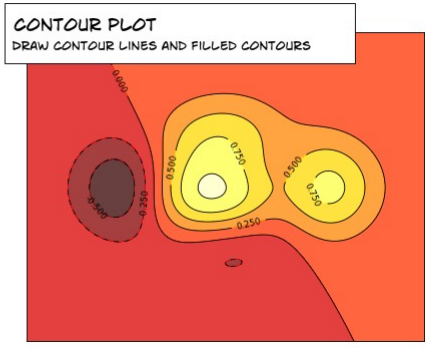


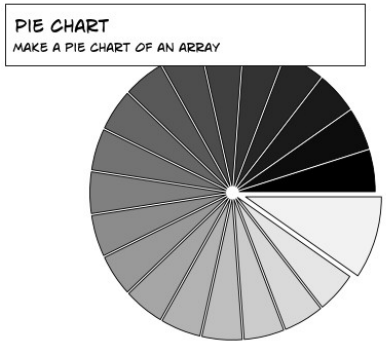


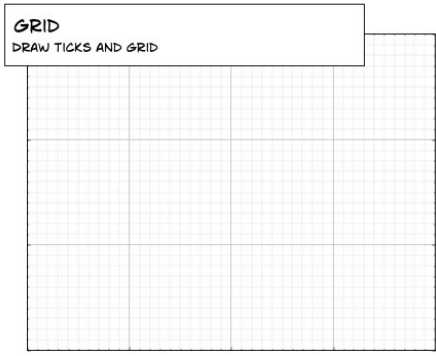




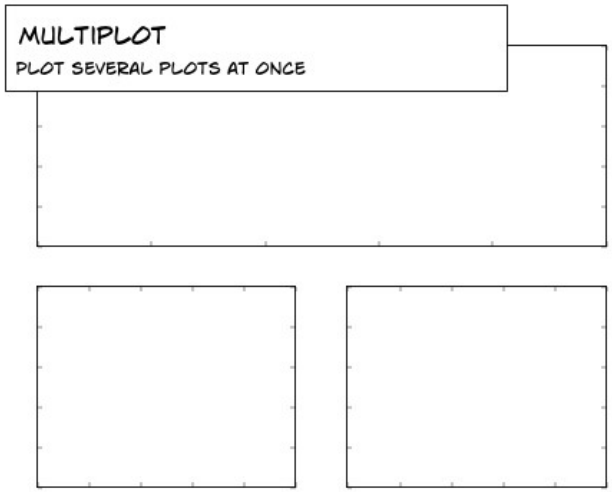


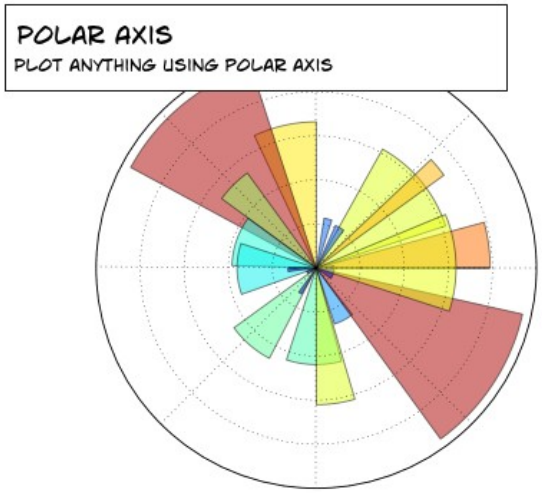


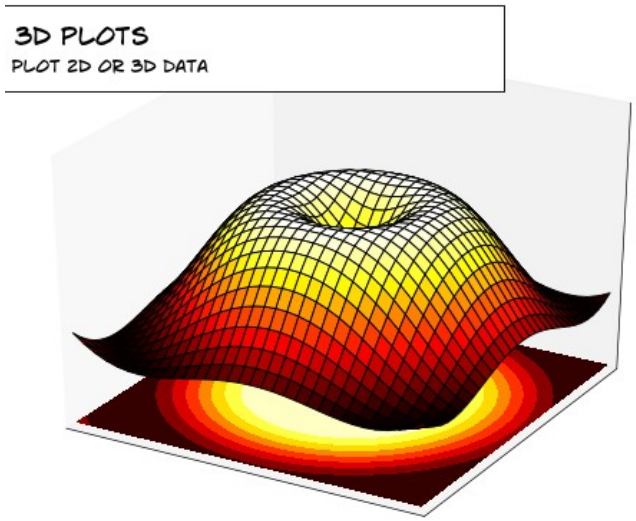


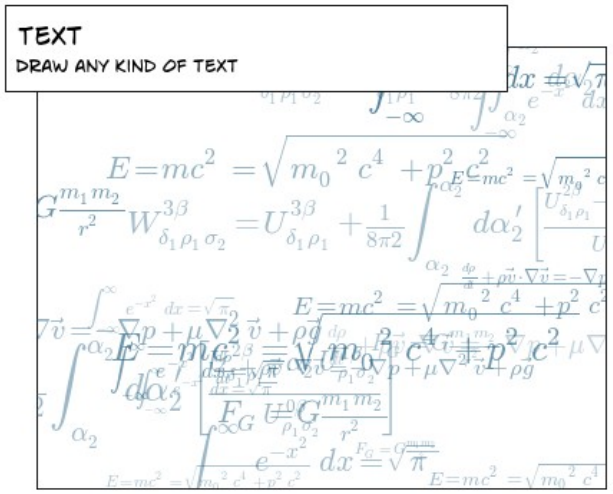


Diğer tipler



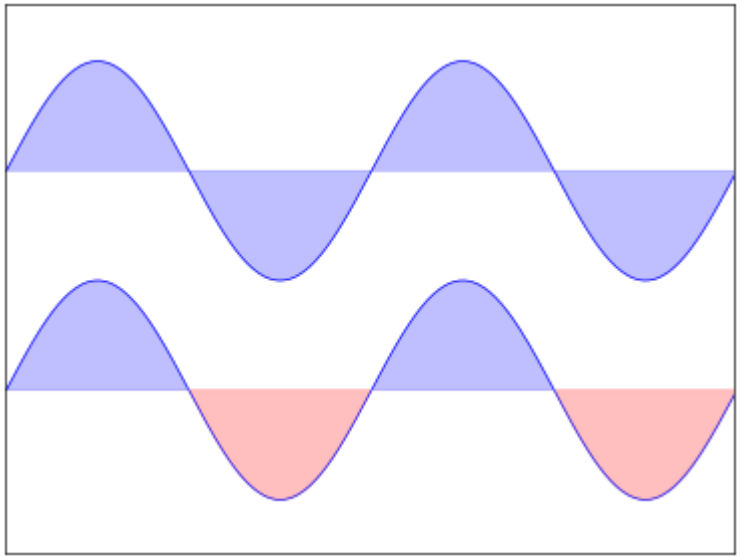






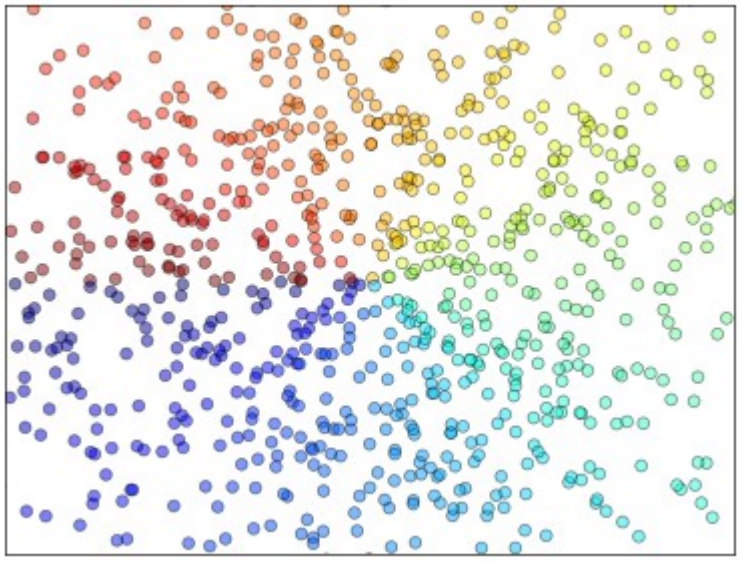
Regular Plots

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

n = 256
X = np.linspace(-np.pi,np.pi,n,endpoint=True)
Y = np.sin(2*X)

plt.axes([0.025,0.025,0.95,0.95])

plt.plot (X, Y+1, color='blue', alpha=1.00)
plt.fill_between(X, 1, Y+1, color='blue', alpha=.25)

plt.plot (X, Y-1, color='blue', alpha=1.00)
plt.fill_between(X, -1, Y-1, (Y-1) > -1, color='blue', alpha=.25)
plt.fill_between(X, -1, Y-1, (Y-1) < -1, color='red',  alpha=.25)

plt.xlim(-np.pi,np.pi), plt.xticks([])
plt.ylim(-2.5,2.5), plt.yticks([])
# savefig('../figures/plot_ex.png',dpi=48)
plt.show()
```



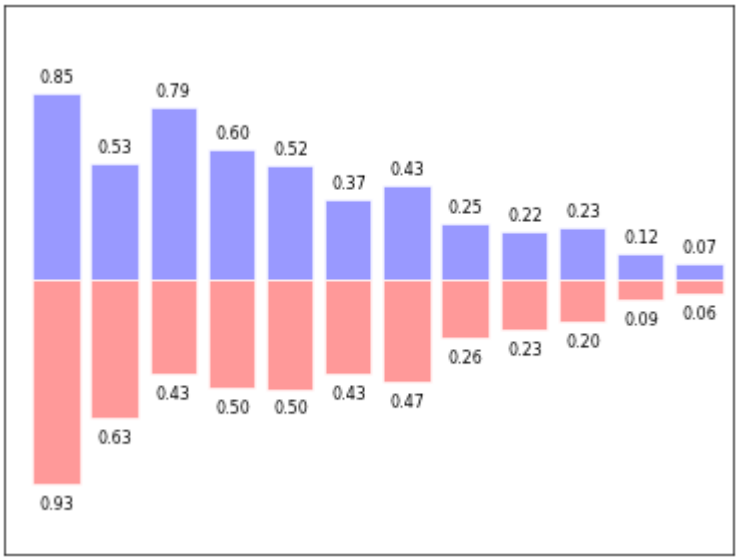
Scatter Plots

- …
- n = 1024
- X = np.random.normal(0,1,n)
- Y = np.random.normal(0,1,n)
-
- plt.scatter(X,Y)
- plt.show()
- ...



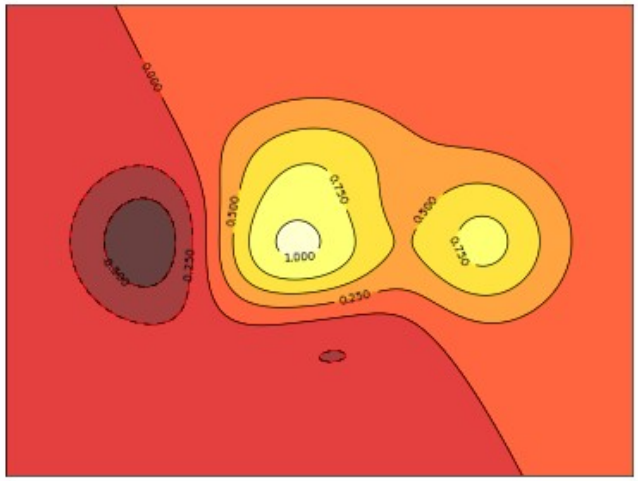
Bar Plots

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

n = 12
X = np.arange(n)
Y1 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)
Y2 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)

plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')

for x,y in zip(X,Y1):
    plt.text(x+0.4, y+0.05, '%.2f' % y, ha='center', va= 'bottom')

plt.ylim(-1.25,+1.25)
plt.show()
```



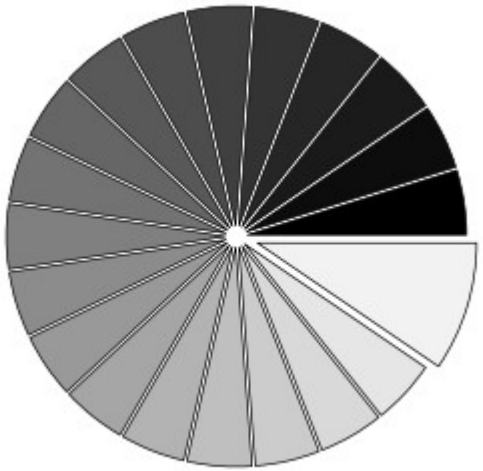
Contour Plots

- import numpy as np
- import matplotlib.pyplot as plt
-
- def f(x,y): return (1-x/2+x**5+y**3)*np.exp(-x**2-y**2)
-
- n = 256
- x = np.linspace(-3,3,n)
- y = np.linspace(-3,3,n)
- X,Y = np.meshgrid(x,y)
-
- plt.contourf(X, Y, f(X,Y), 8, alpha=.75, cmap='jet')
- C = plt.contour(X, Y, f(X,Y), 8, colors='black', linewidth=.5)
- plt.show()
-



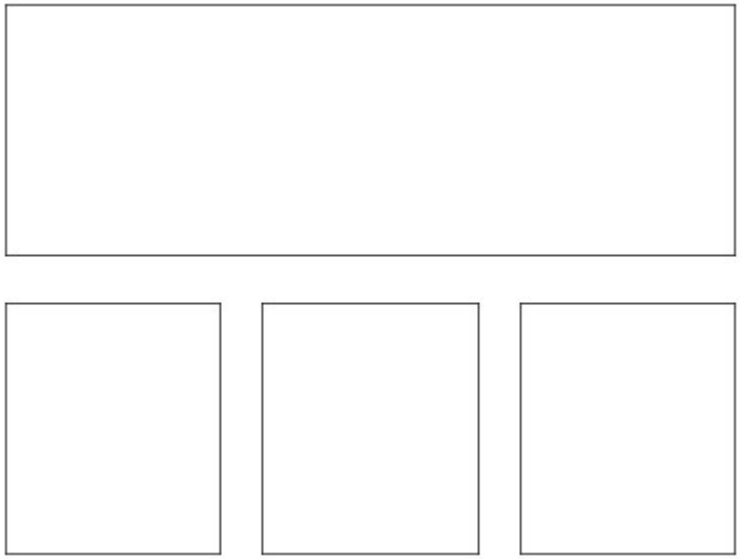
Pie Charts

- import numpy as np
- import matplotlib.pyplot as plt
-
- n = 20
- Z = np.random.uniform(0,1,n)
- plt.pie(Z)
- plt.show()



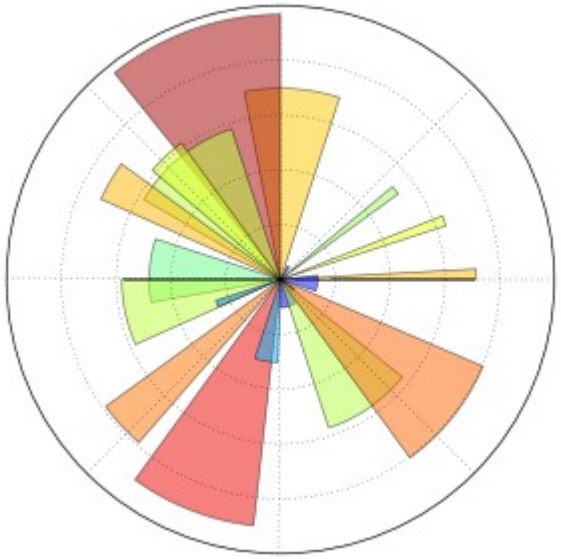
Multi Plots

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

fig = plt.figure()
fig.subplots_adjust(bottom=0.025, left=0.025, top = 0.975, right=0.975)

plt.subplot(2,1,1)
plt.xticks([]), plt.yticks([])

plt.subplot(2,3,4)
plt.xticks([]), plt.yticks([])

plt.subplot(2,3,5)
plt.xticks([]), plt.yticks([])

plt.subplot(2,3,6)
plt.xticks([]), plt.yticks([])

# plt.savefig('../figures/multiplot_ex.png',dpi=48)
plt.show()
```



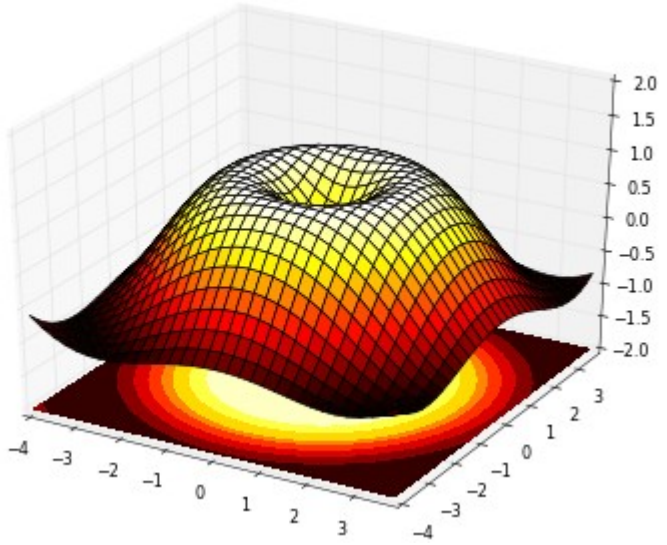
Polar Axis

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

plt.axes([0,0,1,1])

N = 20
theta = np.arange(0.0, 2*np.pi, 2*np.pi/N)
radii = 10*np.random.rand(N)
width = np.pi/4*np.random.rand(N)
bars = plt.bar(theta, radii, width=width, bottom=0.0)

for r,bar in zip(radii, bars):
    bar.set_facecolor( cm.jet(r/10.))
    bar.set_alpha(0.5)

plt.show()
```



3D Plots

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')

plt.show()
```



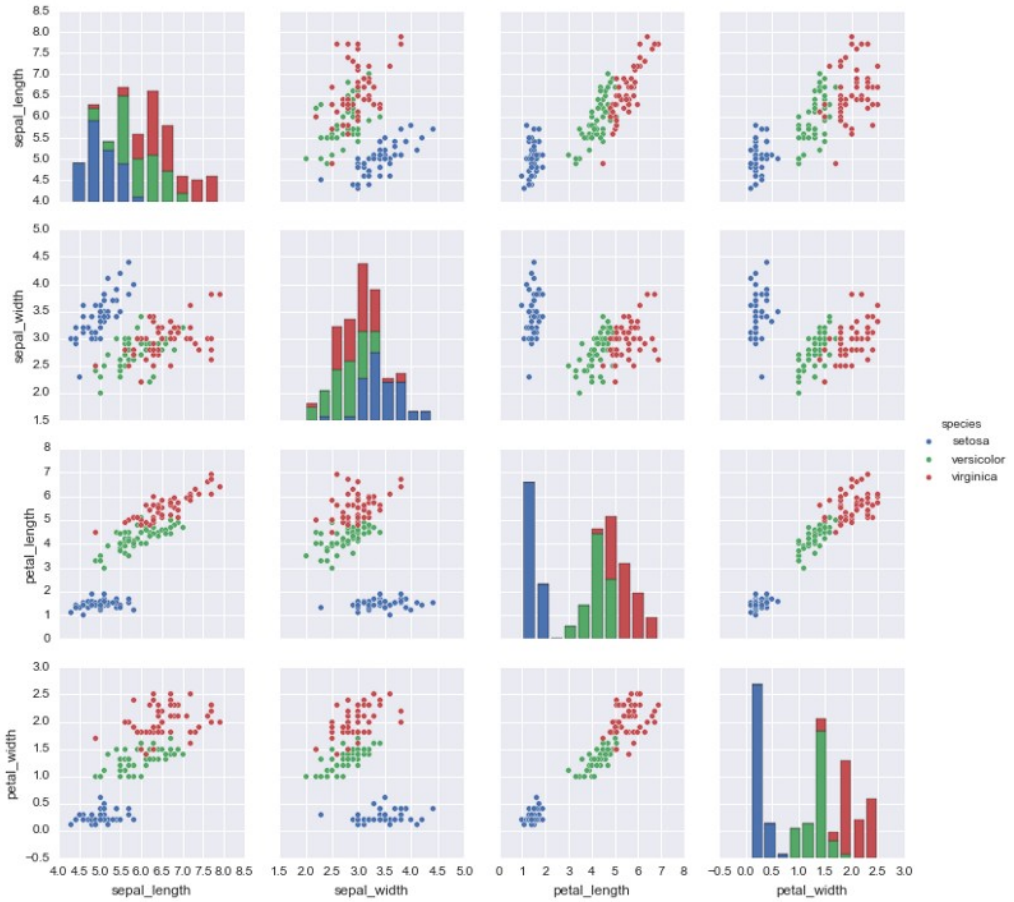
https://www.labri.fr/perso/nrougier/teaching/matplotlib/

matplotlib

- Kütüphane oldukça düşük seviyelidir
- yani gelişmiş görselleştirme seviyelerine ulaşmak için daha fazla kod yazmanız gerekecek
- ve genellikle daha üst seviye araçlar kullanmaktansa daha fazla çaba harcayacaksınız.
- Görselleştirmeyi daha da kolaylaştıracak bazı ek kütüphaneler de vardır.
  - Seaborn, Bokeh, Plotly

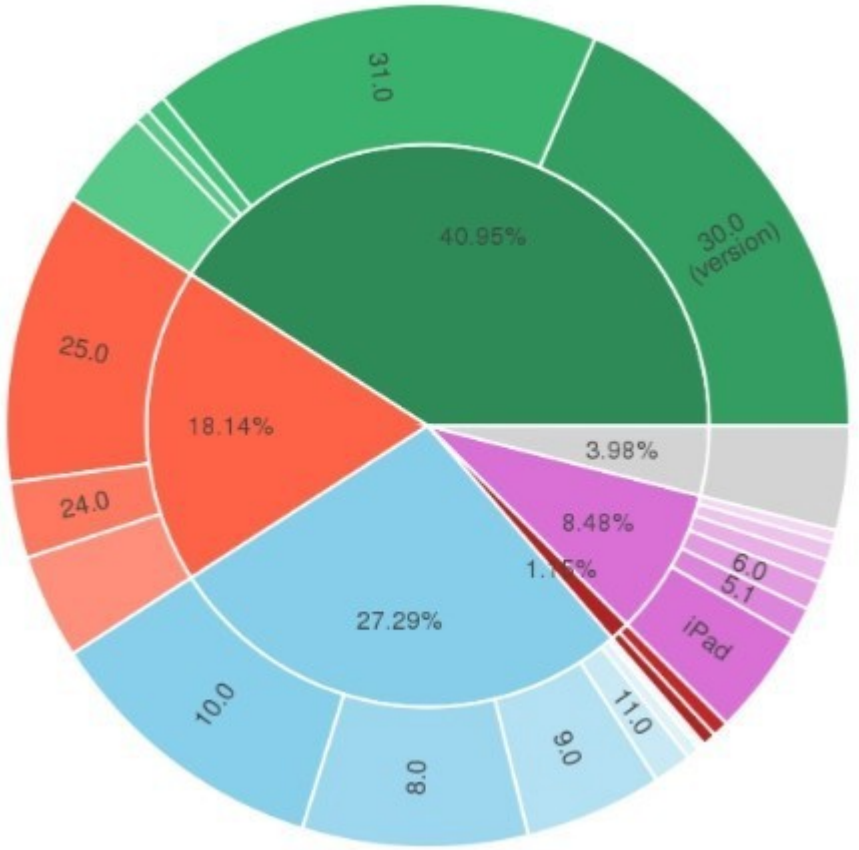
Seaborn

- Seaborn çoğunlukla istatistiksel modellerin görselleştirilmesine odaklanır.
- Bu tür görselleştirmeler, verileri özetleyen ancak yine de genel dağılımları gösteren ısı haritalarını içerir.
- Seaborn, Matplotlib'i temel alır ve buna çok bağlıdır.



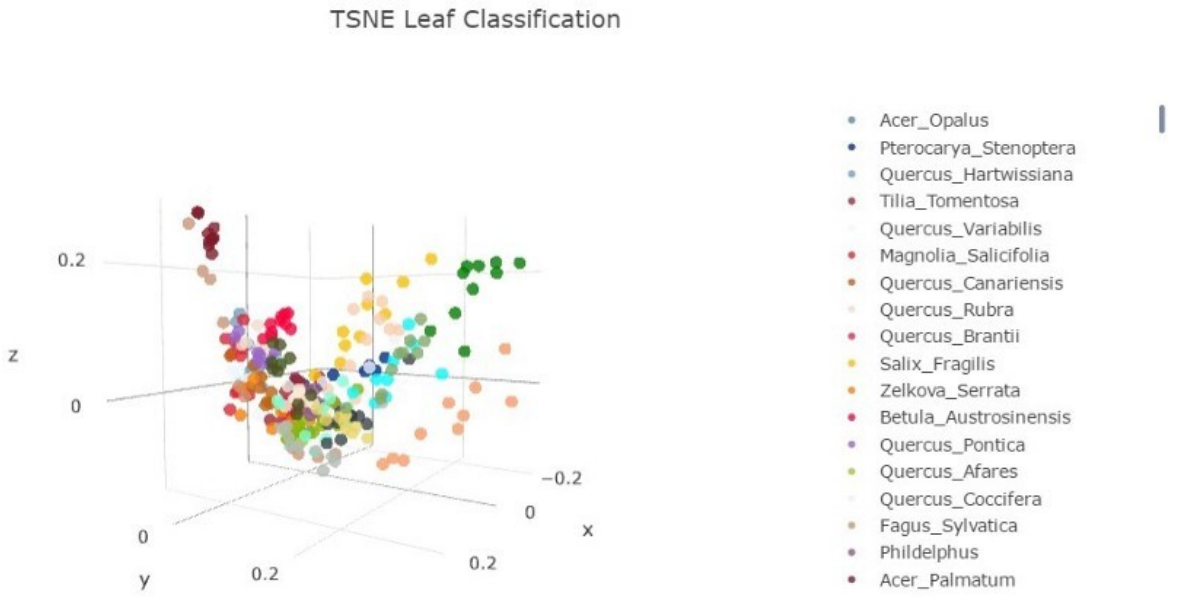
Bokeh

- Önceki kütüphanenin aksine, bu Matplotlib'den bağımsızdır.
- Bokeh'in ana odağı etkileşimdir ve sunumunu Data-Driven Documents (d3.js) tarzında modern tarayıcılar aracılığıyla yapar.



Plotly

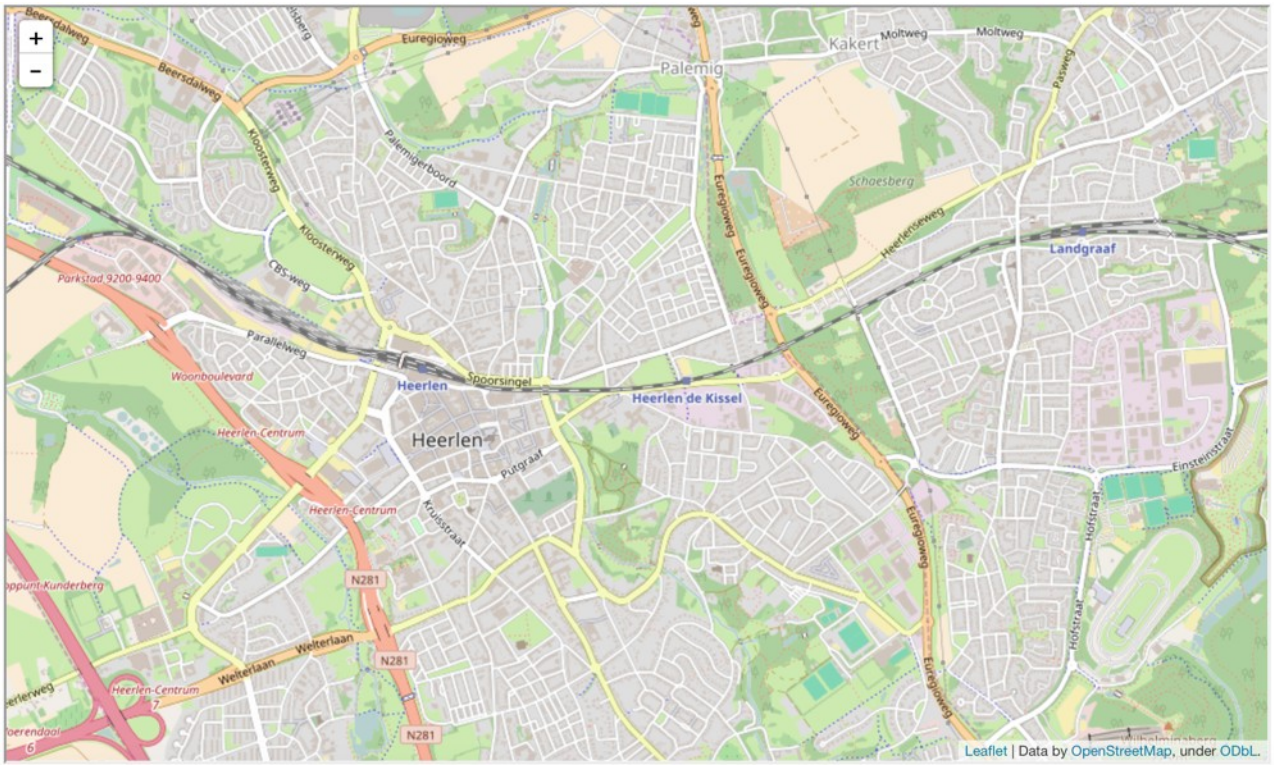
- Çevrimiçi analiz ve veri görselleştirme aracı
- Plotly hem kolayca veri grafik oluşturmanızı sağlıyor hemde verileri analiz etmenizde eğitimlerle yol gösteriyor. Kullanımı oldukça kolay ve eğitim modülleri de oldukça zevkli bu aracı denemenizi öneriyoruz.
- https://plot.ly/learn/



Folium: tematik haritalar

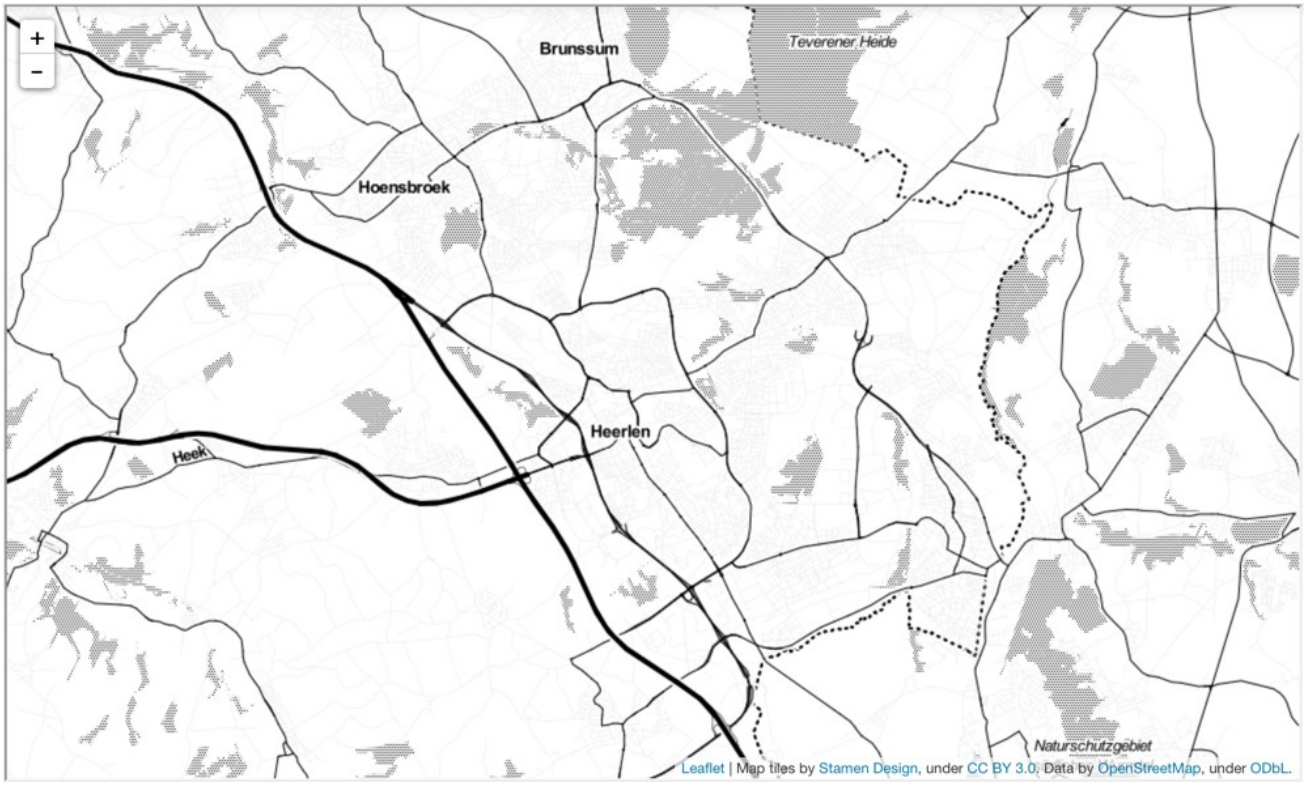
- A thematic map is a visualization where statistical information with a spatial component is shown.
- Other libraries are: Basemap, Cartopy, Iris
- Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the Leaflet.js library.
- Manipulate your data in Python, then visualize it in on a Leaflet map via Folium.

Basic maps



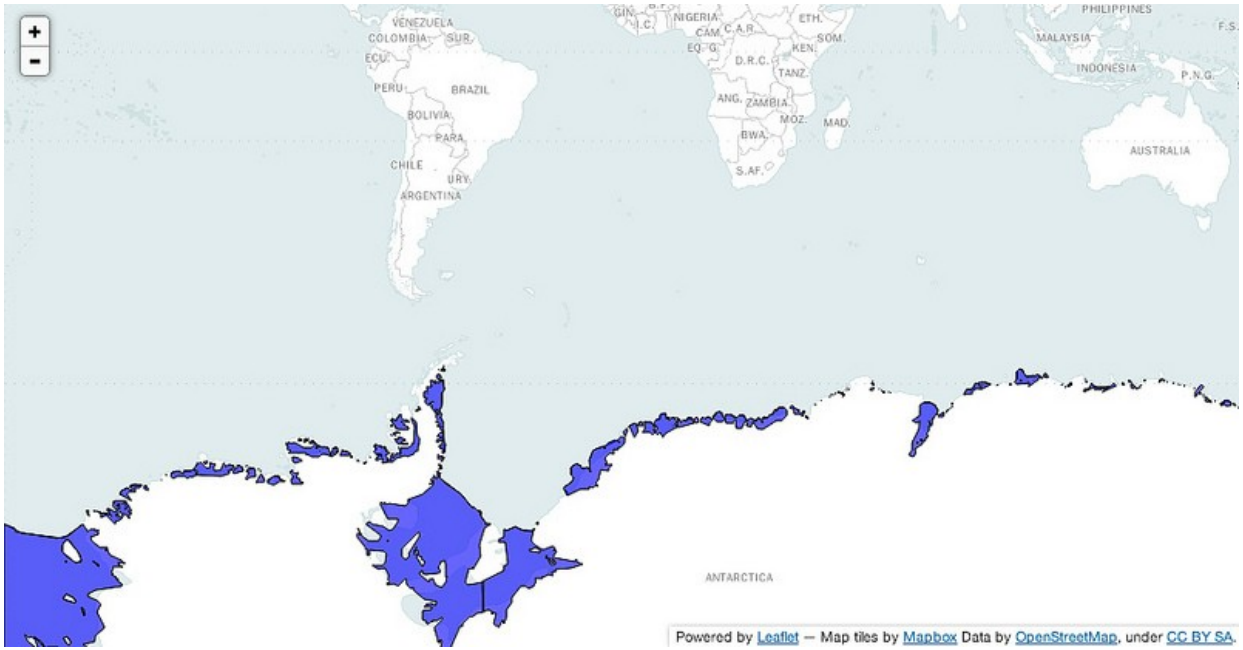

```
folium.Map(location=[50.89, 5.99], zoom_start=14)
```

Basic maps



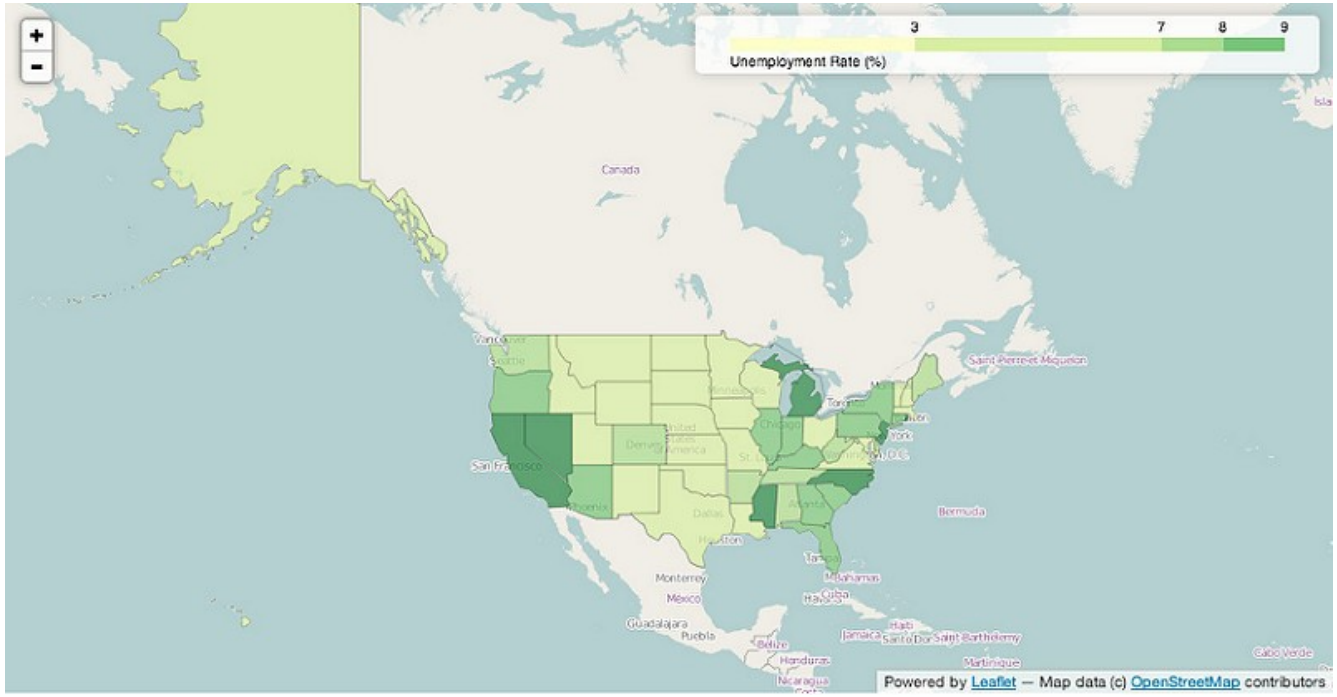

```
folium.Map(location=[50.89, 5.99], zoom_start=14, tiles='Stamen Toner')
```

ggplot GeoJSON/TpoJSON Overlays




```
ice_map = folium.Map(location=[-59, -11], tiles='Mapbox Bright', zoom_start=2)
ice_map.geo_json(geo_path=geo_path)
ice_map.geo_json(geo_path=topo_path, topojson='objects.antarctic_ice_shelf')
ice_map.create_map(path='ice_map.html')
```

Choropleth maps




```
map = folium.Map(location=[48, -102], zoom_start=3)
map.choropleth(geo_path=state_geo, data=state_data,
               columns=['State', 'Unemployment'], key_on='feature.id',
               fill_color='YlGn', fill_opacity=0.7, line_opacity=0.2,
               legend_name='Unemployment Rate (%)')
```

Uygulama

- matplotlib

  https://github.com/matplotlib/AnatomyOfMatplotlib

- seaborn

  Piazza resources icinde

  seaborn-tutorial-for-beginners.ipynb