Data Science

Week 5

Data Visualization

Review

1. Basics of Statistics from the point of view of data science

2. Relationships between variables

3. Hypothesis testing

Hypothesis testing

- The first step is to quantify the size of the apparent effect by choosing a **test statistic (mean, probability…)**.
- The second step is to define a **null hypothesis**, which is a model of the system based on the assumption that the apparent effect is not real.
- The third step is to compute a ***p*-value**, which is the probability of seeing the apparent effect if the null hypothesis is true.
- The last step is to interpret the result. If the p-value is low, the effect is said to be **statistically significant**, which means that it is unlikely to have occurred by chance.

Outline

1. Basics of Data Visualization

2. One more step to application for data science
   - Visualization with parallel coordinates plot
   - Visualization for time series data

Basics of Data Visualization

- Data Visualization needs many basic skills of plotting by programming.

- The visualization project can be decomposed into serval steps of the basic plotting according to the requirements of the project.

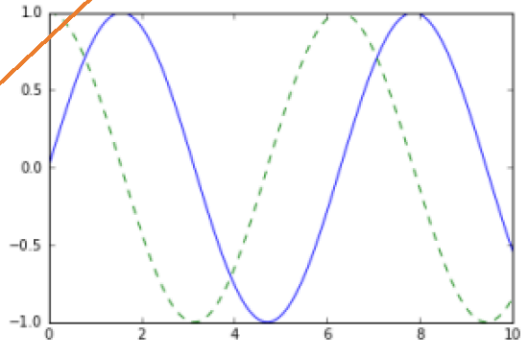
Simple line plot

We'll now take an in-depth look at the Matplotlib tool for visualization in Python in this class.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)

fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```

Resolution of data



Subplot in one figure

```
plt.figure()  # create a plot figure

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```

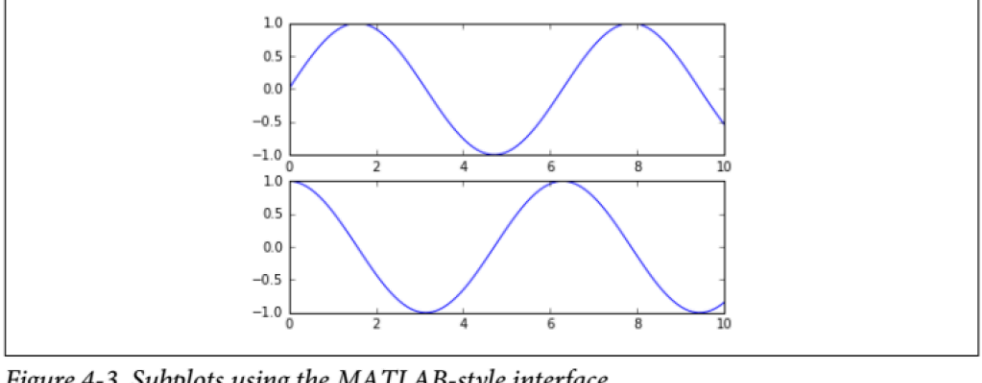


Figure 4-3. Subplots using the MATLAB-style interface

Adjusting the Plot: Line Colors

```
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

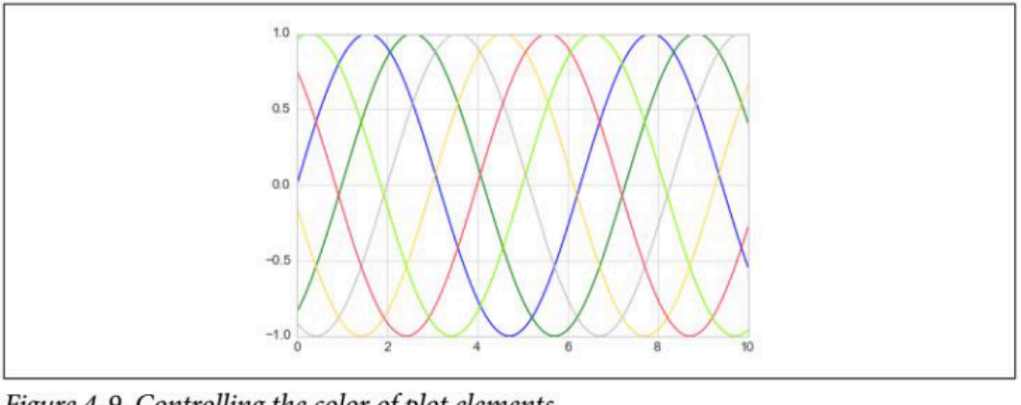


Figure 4-9. Controlling the color of plot elements

Adjusting the Plot: Styles

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```

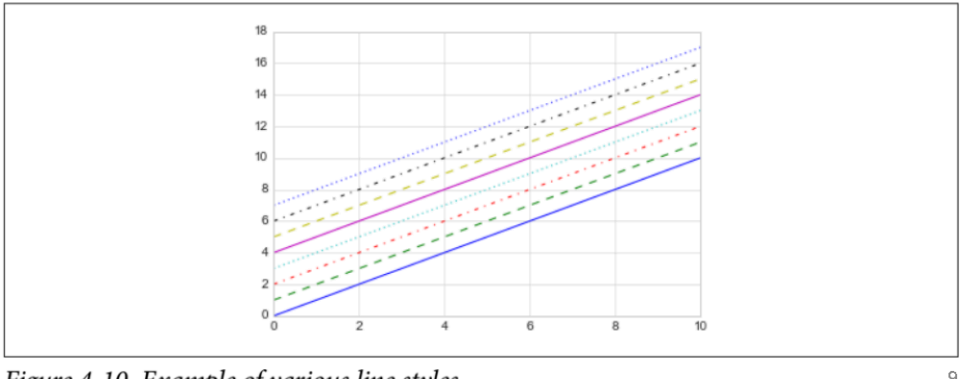


Figure 4-10. Example of various line styles

Labelling Plots

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend();
```

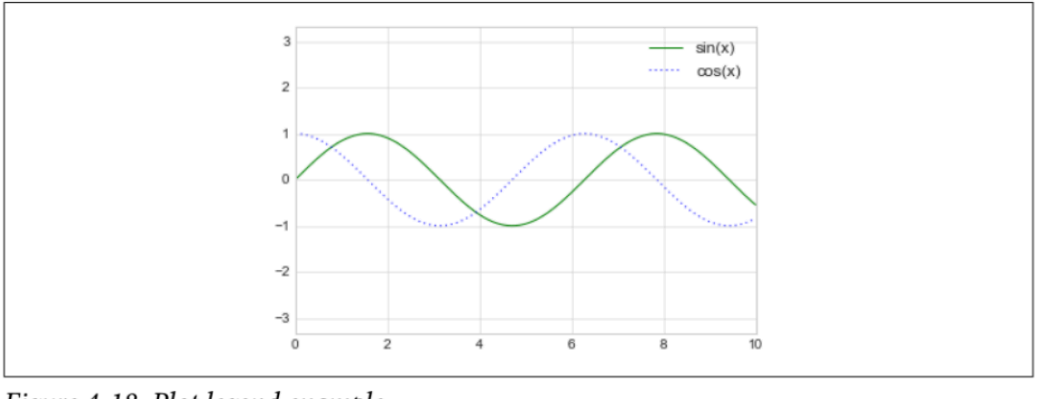


Figure 4-18. Plot legend example

Simple Scatter Plots

```
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color='black');
```

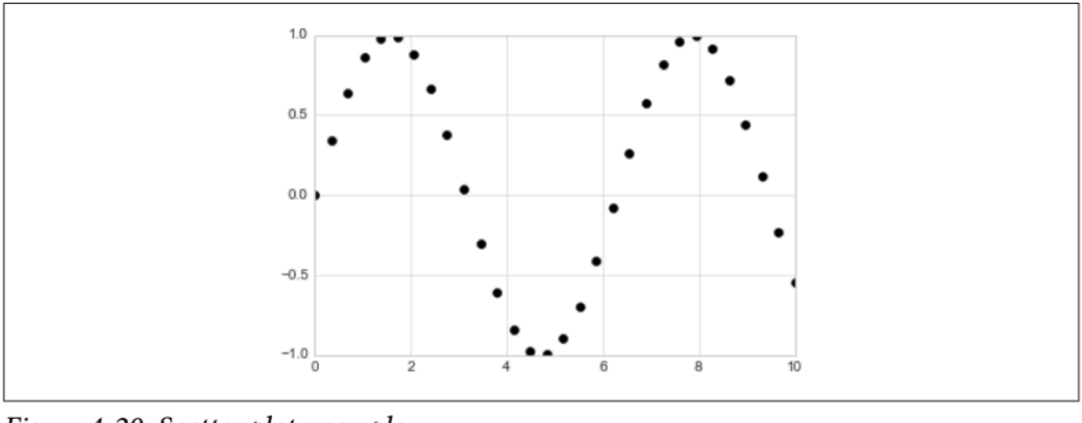


Figure 4-20. Scatter plot example

Point-line style in Scatter Plots

```
plt.plot(x, y, '-ok');   # line (-), circle marker (o), black (k)
```

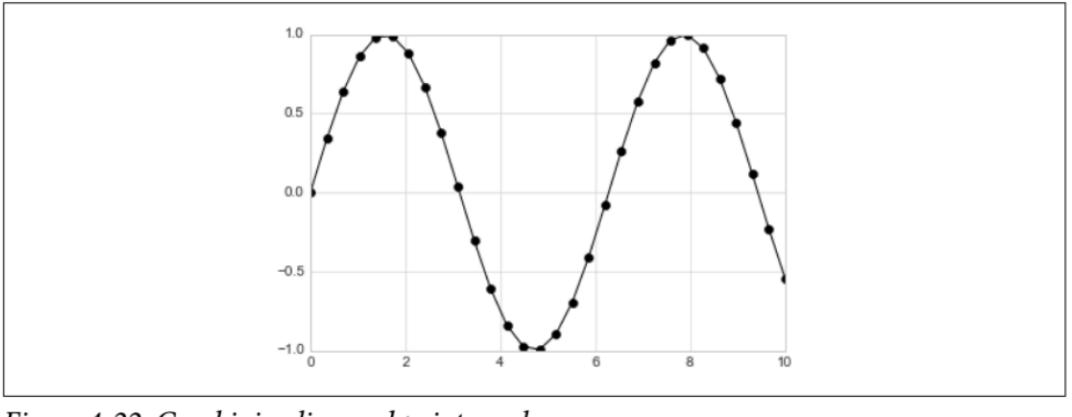


Figure 4-22. Combining line and point markers

Size, color, and transparency in Scatter Plots

```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar();  # show color scale
```

Check the arguments by yourself

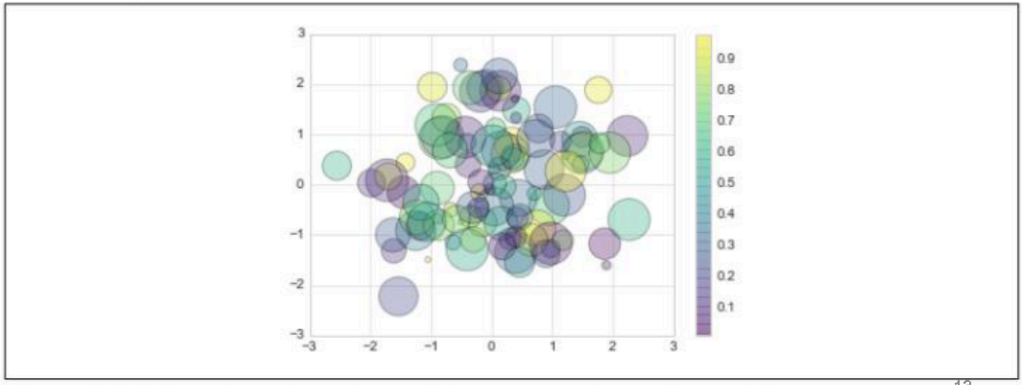


Figure 4-25. Changing size, color, and transparency in scatter points

Visualizing Errors

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np

x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50)

plt.errorbar(x, y, yerr=dy, fmt='.k');
```

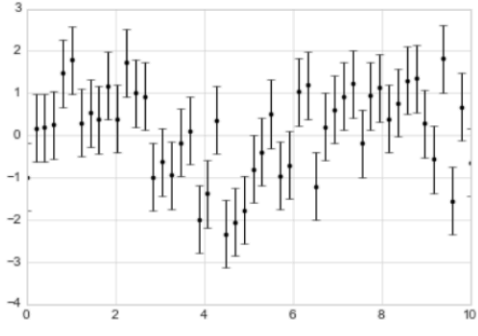


Figure 4-27. An errorbar example

Customizing error bars

```
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
             ecolor='lightgray', elinewidth=3, capsize=0);
```

Check the arguments by yourself

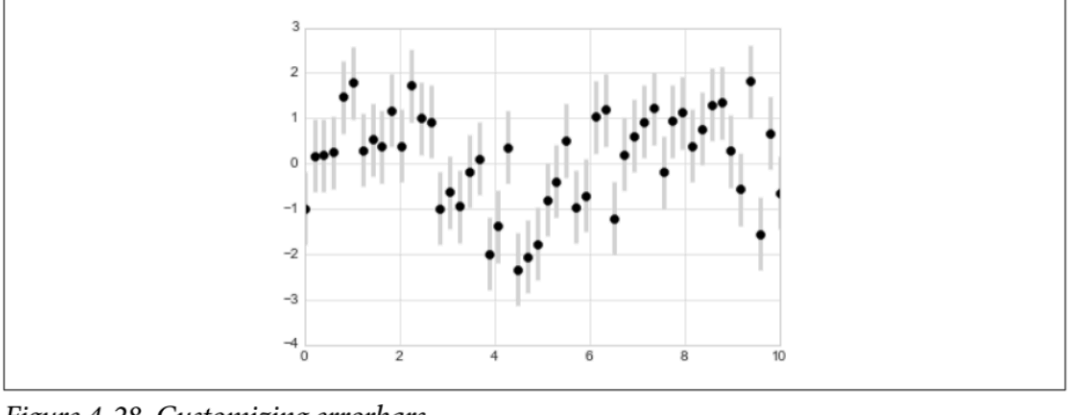


Figure 4-28. Customizing errorbars

Contour Plots

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np

def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)


x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)


plt.contour(X, Y, Z, colors='black');
```

Visualizing 3D data by 2D plot

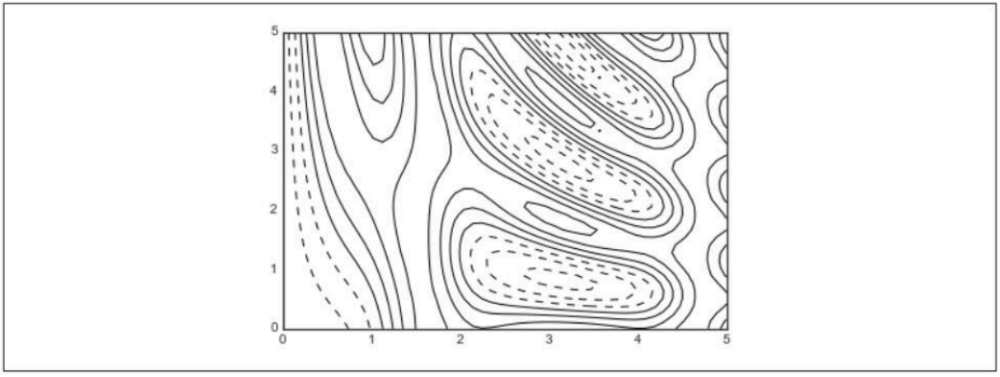


Figure 4-30. Visualizing three-dimensional data with contours

Visualizing 3D data with color contours

```
plt.contour(X, Y, Z, 20, cmap='RdGy');
```

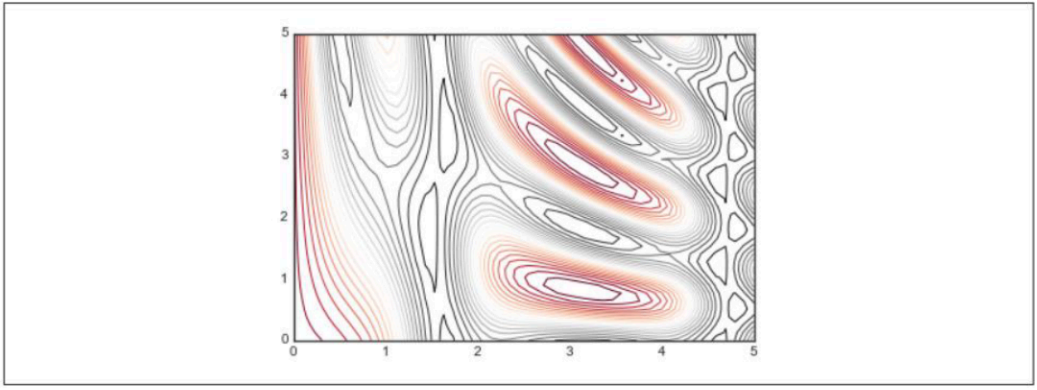


Figure 4-31. Visualizing three-dimensional data with colored contours

Visualizing 3D data with filled contours

```
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar();
```

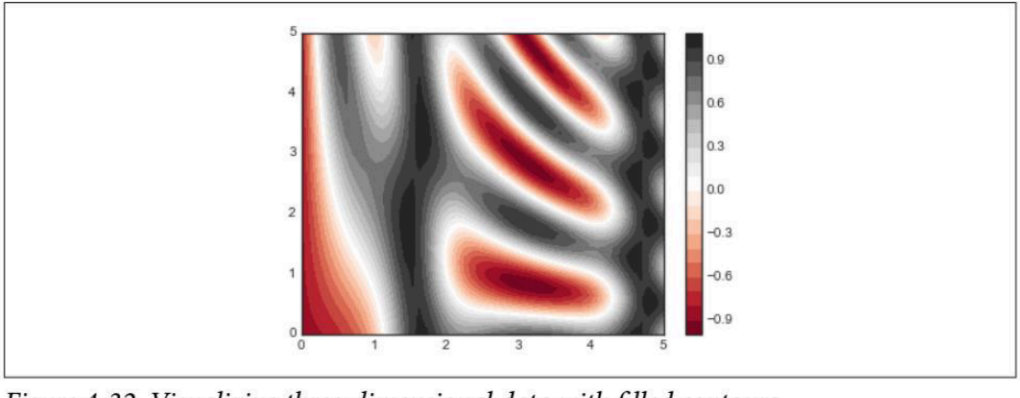


Figure 4-32. Visualizing three-dimensional data with filled contours

Labelling contours

```
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)

plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
           cmap='RdGy', alpha=0.5)
plt.colorbar();
```

Check the arguments by yourself

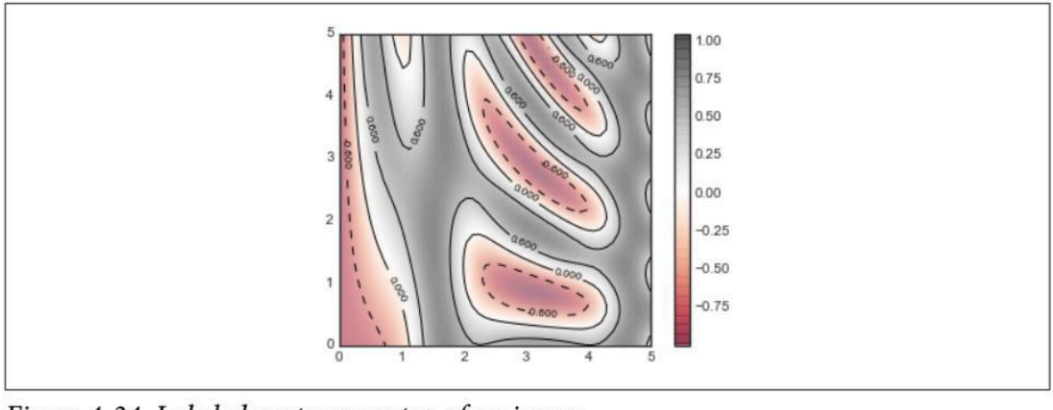


Figure 4-34. Labeled contours on top of an image

Distribution: Histogram (slide in week 4)

- One of the best ways to describe a variable is to report the values that appear in the dataset and how many times each value appears. This description is called the distribution of the variable.
- The most common representation of a distribution is a histogram, which is a graph that shows the frequency of each value. In this context, “frequency” means the number of times the value appears.

Python code for Histogram (slide in week 4)

- In Python, an efficient way to compute frequencies is with a dictionary. Given a sequence of values, t:

```
 8    hist = {}
 9    t = [1, 2, 2, 3, 5]
10    for x in t:
11        hist[x] = hist.get(x, 0) + 1
```

- The result is a dictionary that maps from values to frequencies.

Histograms and Binnings

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

data = np.random.randn(1000)

plt.hist(data);
```

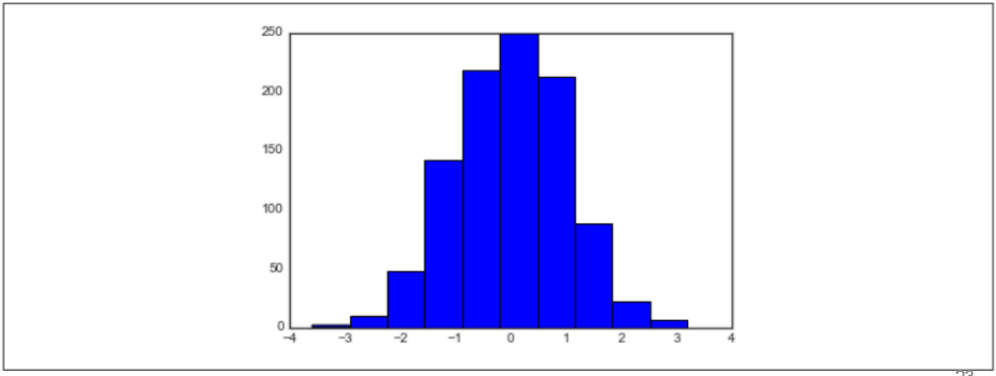


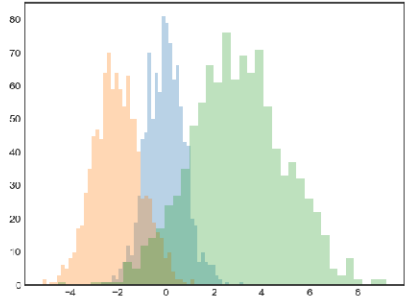
Figure 4-35. A simple histogram

Multiple Histograms

- The plt.hist has more information on other customization options available.
- We find this combination of histtype='stepfilled' along with some transparency alpha to be very useful when comparing histograms of several distributions

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```



Two-Dimensional Histograms and Binnings

```
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 10000).T

plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```

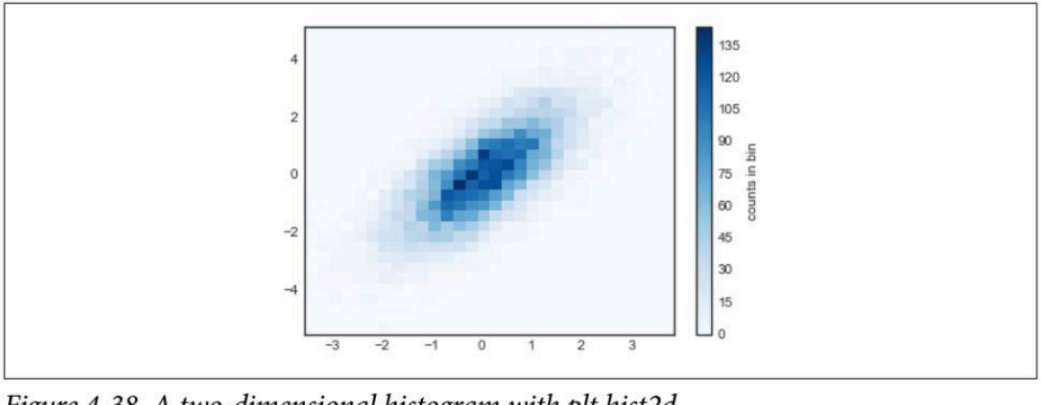
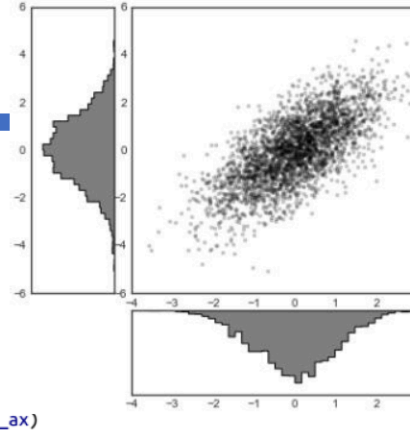


Figure 4-38. A two-dimensional histogram with plt.hist2d

Visualizing multidimensional distributions



```
# Create some normally distributed data
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 3000).T

# Set up the axes with gridspec
fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)
main_ax = fig.add_subplot(grid[:-1, 1:])
y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)

# scatter points on the main axes
main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)

# histogram on the attached axes
x_hist.hist(x, 40, histtype='stepfilled',
            orientation='vertical', color='gray')
x_hist.invert_yaxis()

y_hist.hist(y, 40, histtype='stepfilled',
            orientation='horizontal', color='gray')
y_hist.invert_xaxis()
```

Customizing Plot Legends

```
import numpy as np

x = np.linspace(0, 10, 1000)
fig, ax = plt.subplots()
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')
ax.axis('equal')
leg = ax.legend();
```

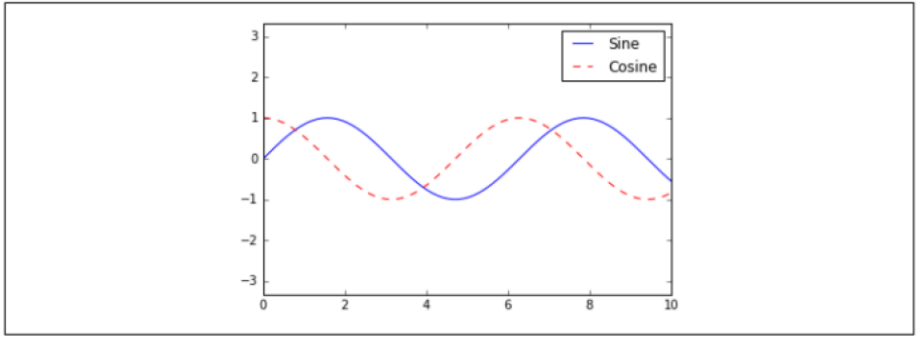


Figure 4-41. A default plot legend

Legend for Size of Points

```
import pandas as pd
cities = pd.read_csv('data/california_cities.csv')

# Extract the data we're interested in
lat, lon = cities['latd'], cities['longd']
population, area = cities['population_total'], cities['area_total_km2']

# Scatter the points, using size and color but no label
plt.scatter(lon, lat, label=None,
            c=np.log10(population), cmap='viridis',
            s=area, linewidth=0, alpha=0.5)
plt.axis(aspect='equal')
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.colorbar(label='log$_{10}$(population)')
plt.clim(3, 7)

# Here we create a legend:
# we'll plot empty lists with the desired size and label
for area in [100, 300, 500]:
    plt.scatter([], [], c='k', alpha=0.3, s=area,
                label=str(area) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, title='City Area')

plt.title('California Cities: Area and Population');
```

Check the arguments by yourself

Legend for Size of Points

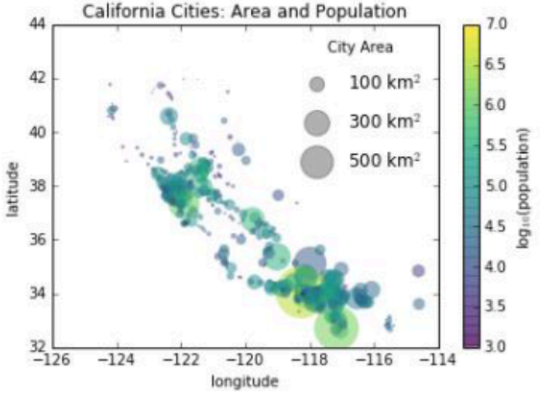


Figure 4-47. Location, geographic size, and population of California cities

Colorbars

```
x = np.linspace(0, 10, 1000)
I = np.sin(x) * np.cos(x[:, np.newaxis])

plt.imshow(I)
plt.colorbar();
```

- a colorbar is a separate axes that can provide a key for the meaning of colors in a plot.
- simplest colorbar can be created with the plt.colorbar function

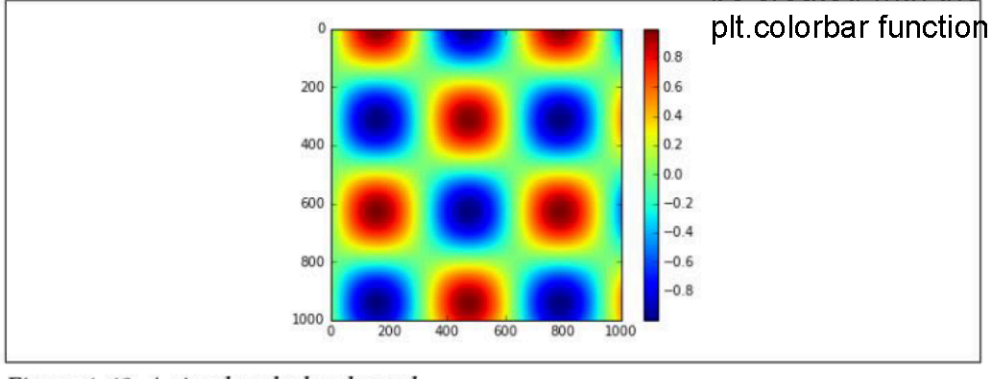


Figure 4-49. A simple colorbar legend

Text

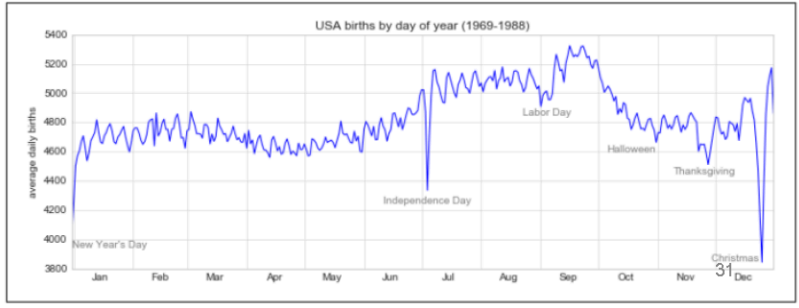
```
fig, ax = plt.subplots(figsize=(12, 4))
births_by_date.plot(ax=ax)

# Add labels to the plot
style = dict(size=10, color='gray')

ax.text('2012-1-1', 3950, "New Year's Day", **style)
ax.text('2012-7-4', 4250, "Independence Day", ha='center', **style)
ax.text('2012-9-4', 4850, "Labor Day", ha='center', **style)
ax.text('2012-10-31', 4600, "Halloween", ha='right', **style)
ax.text('2012-11-25', 4450, "Thanksgiving", ha='center', **style)
ax.text('2012-12-25', 3850, "Christmas ", ha='right', **style)

# Label the axes
ax.set(title='USA births by day of year (1969-1988)',
       ylabel='average daily births')

# Format the x axis with centered month labels
ax.xaxis.set_major_locator(mpl.dates.MonthLocator())
ax.xaxis.set_minor_locator(mpl.dates.MonthLocator(bymonthday=15))
ax.xaxis.set_major_formatter(plt.NullFormatter())
ax.xaxis.set_minor_formatter(mpl.dates.DateFormatter('%h'));
```



Annotation

```
fig, ax = plt.subplots()

x = np.linspace(0, 20, 1000)
ax.plot(x, np.cos(x))
ax.axis('equal')

ax.annotate('local maximum', xy=(6.28, 1), xytext=(10, 4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.annotate('local minimum', xy=(5 * np.pi, -1), xytext=(2, -6),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="angle3,angleA=0,angleB=-90"));
```

Check the arguments by yourself

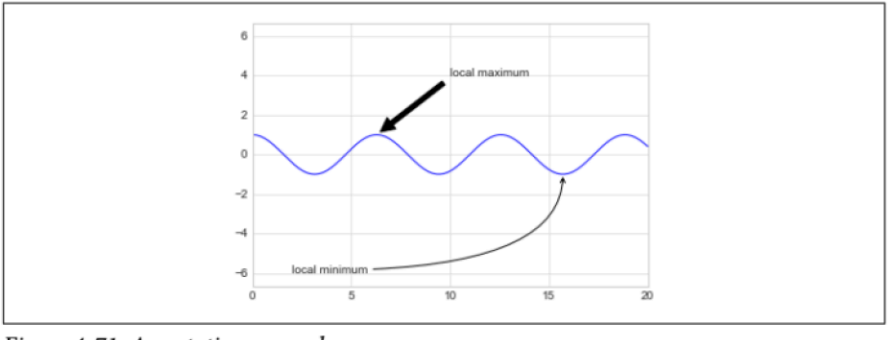


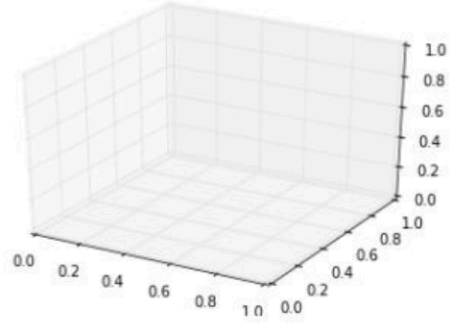
Figure 4-71. Annotation examples

Three-Dimensional Plotting

```
from mpl_toolkits import mplot3d

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection='3d')
```



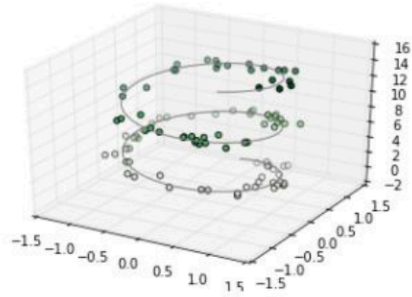
We enable three-dimensional plots by importing the mplot3d toolkit, included with the main Matplotlib installation

Three-Dimensional Points and Lines

```
ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Greens');
```



Check the arguments by yourself

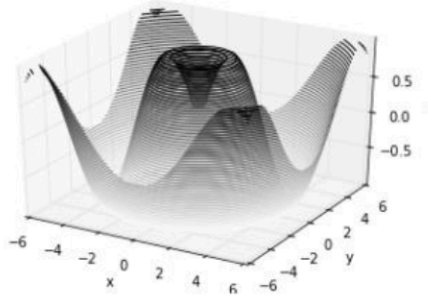
Three-Dimensional Contour Plot

```
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```



View angle for a three-dimensional plot

```
ax.view_init(60, 35)
fig
```

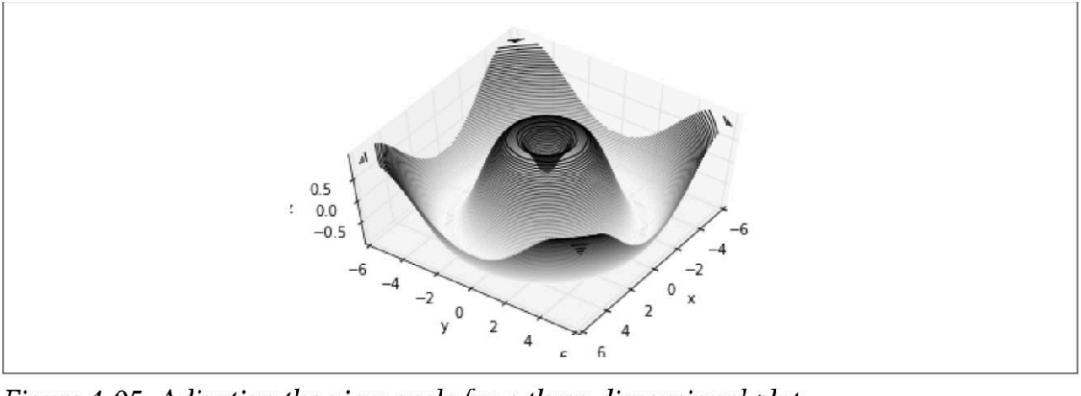


Figure 4-95. Adjusting the view angle for a three-dimensional plot

Colorful three-dimensional plot

```
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
ax.set_title('surface');
```

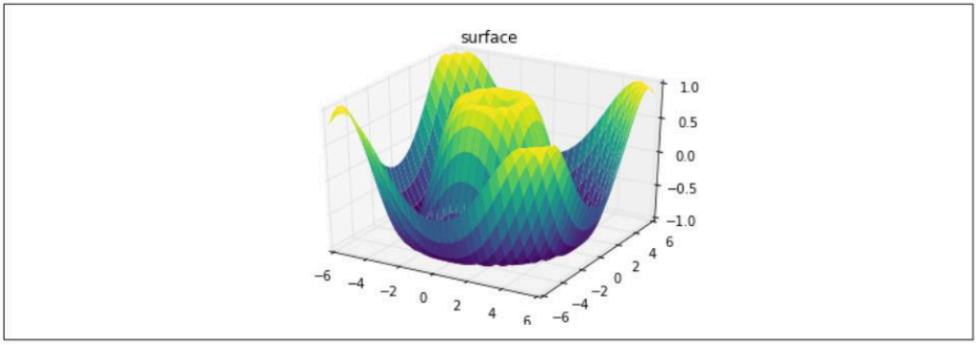


Figure 4-97. A three-dimensional surface plot

Outline

1. Basics of Data Visualization

2. One more step to application for data science
   - Visualization with parallel coordinates plot
   - Visualization for time series data (stock data)

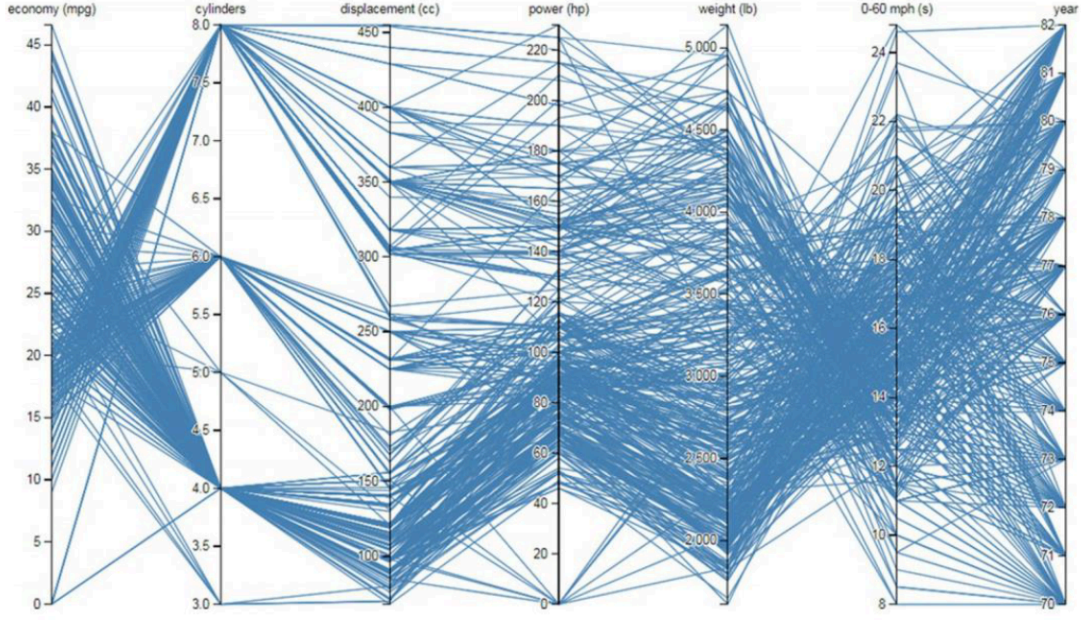
Good visualization -1

- Data analysis and data visualization are very important tools for engineers, analysts, policy-makers, and decision makers.
- Usually, a successful figure requires a lot of work, customization, attention to detail, and refinements.
- This leads naturally to the question of visualization quality.
- What is a good visualization?

Good visualization -2

- In the context of big-data, the amount of data to be plotted is very large.
- Therefore, in order to be able to extract meaningful messages, one needs to resort to preprocessing techniques, in order to extract some meaningful structure from the data.
- Another important concept relating to visualization is beauty.
  - novel, informative, and efficient

Example of a parallel coordinates plot



Dataset with passenger car characteristics

- A small data set with passenger car characteristics from the 70s and 80s is used to showcase parallel coordinates.
- Variables that describe the cars include miles per gallon (mpg), number of cylinders, displacement (cc), power (hp), weight (lb), time to reach 60miles per hour, and year of construction.
- The data set contains information for 408 cars ~~and be downloaded from https://bl.ocks.org/jasondavies/1341281~~ (Parallel Coordinates, 2018)

Parallel coordinates plot

- Parallel coordinates is a well-researched visualization that allows the analyst to discover predominant patterns.
- Parallel coordinates can visualize effectively a few thousand observations, each having up to 10 or 15 variables depending on the data.
- when used interactively, parallel coordinates can reveal outliers, clusters, and relationships

Parallel coordinates plot can:

- Focus on any given observation or subspace. Selection queries are easily generated by brushing.
- Examine the relationship between the variables in the data set. Parallel lines between adjacent axes show positive correlation and intersecting lines a negative correlation.
- Uncover clusters or patterns in the data or in a selected subspace. This can be made easier if opacity and color are used.

Selection Query in Parallel coordinates plot

An observation, or a car in this particular case, is represented by a point/coordinate on each of the parallel axes.

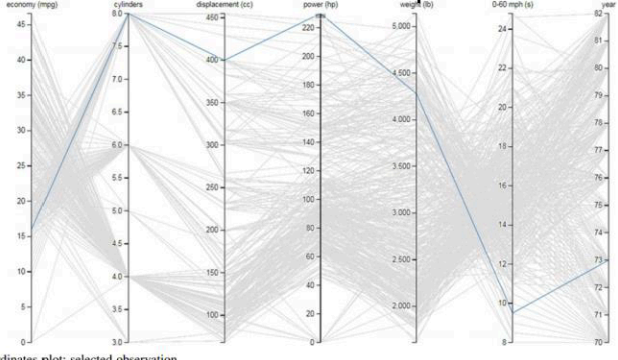


FIG. 3 Parallel coordinates plot: selected observation.

In Fig. 3, the car that has the highest horsepower (hp), approximately 230hp, is selected and shown as a blue line.

The rest of the cars/lines are shown in gray. It is easy to inspect the selected car's characteristics in relation to the rest of the data set.

For example, the selected car was built in 1973 (last variable on the right); it was one of the heaviest and at the same time one of the fastest in the data set.

Parallel coordinates can:

- Focus interactively on any given observation or subspace. Selection queries are easily generated by brushing.
- Examine the relationship between the variables in the data set. Parallel lines between adjacent axes show positive correlation and intersecting lines a negative correlation.
- Uncover clusters or patterns in the data or in a selected subspace. This can be made easier if opacity and color are used.

Relationships between adjacent variables

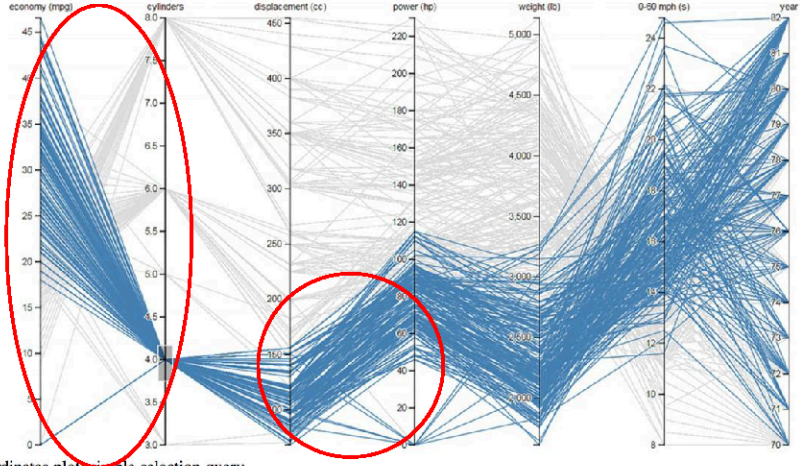
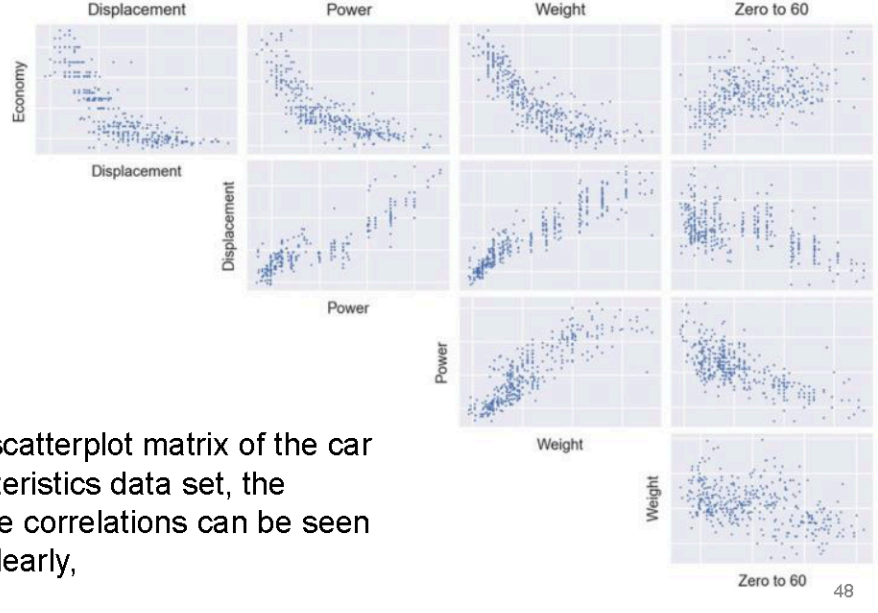


FIG. 4 Parallel coordinates plot: simple selection query.

- Relationships between adjacent variables are revealed by studying how lines are arranged between the two adjacent axes.
- If the lines are parallel to each other, regardless of slope, then there is a positive correlation between the two variables. In this Figure, this is the relationship between displacement and power for the selected four-cylinder cars.
- Negative correlation is shown as lines that cross each other. For example, in this Figure, this is the relationship between fuel economy and the number of cylinders.

Scatterplot matrix of the pairwise relationships in the car data set



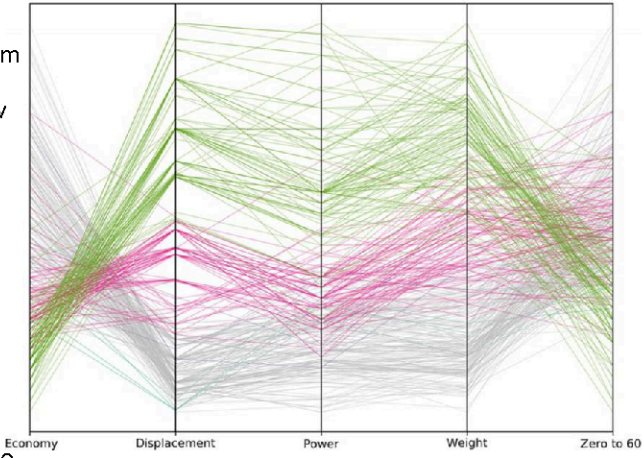
- In the scatterplot matrix of the car characteristics data set, the pairwise correlations can be seen more clearly,

Parallel coordinates can:

- Focus interactively on any given observation or subspace. Selection queries are easily generated by brushing.
- Examine the relationship between the variables in the data set. Parallel lines between adjacent axes show positive correlation and intersecting lines a negative correlation.
- Uncover clusters or patterns in the data or in a selected subspace. This can be made easier if opacity and color are used.

Uncover clusters or patterns

1. Cars have been color-coded, based on the number of cylinders variable (which is otherwise not included in the plot).
2. At least three distinct patters emerge from the display.
3. Cars with a small displacement have low power and weight and are slower, but more fuel-efficient.
4. On the contrary, cars with high displacement have more power and are faster, despite weighting more.
5. A third category of cars, those with six cylinders, lie in between the four- and eight-cylinder cars.
6. Unlike other high-dimensional visualizations, parallel coordinates can visualize side by side categorical, and numeric variables.
7. However, nonnumeric variables concentrate many lines through the same point, something that makes the plot harder to comprehend.



Using library for the Complicated Visualization

- Many famous visualization methods have been developed in libraries.
- E.g. the following code example can be used for parallel coordinates plotting to generate the picture in page 41 (Example of a parallel coordinates plot).

```
# example source code for Data Science Week-5

import plotly.express as px
from plotly.offline import plot
import pandas as pd
import plotly.io as pio
pio.renderers.default = 'browser'

# load data
df = pd.read_csv('car.csv', index_col=0, parse_dates=True)
# plot the parallel coordinates
fig = px.parallel_coordinates(df)
plot(fig)
```

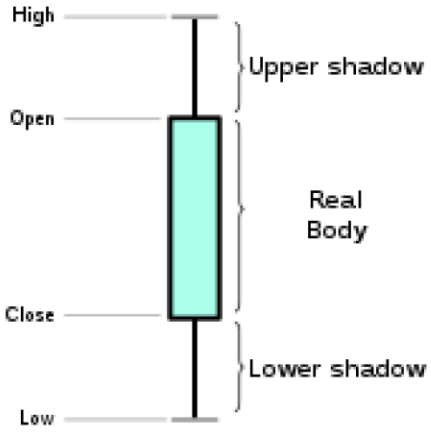
Outline

1. Basics of Data Visualization

2. One more step to application for data science
   - Visualization with parallel coordinates plot
   - Visualization for time series data

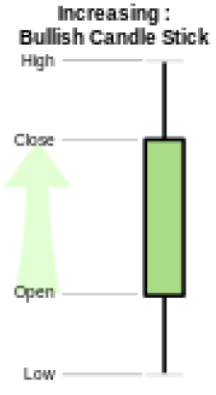
Candlestick for stock data visualization

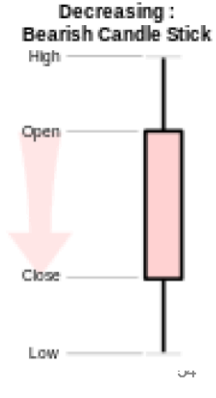
- A candlestick chart (also called Japanese candlestick chart) is a style of financial chart used to describe price movements of a security, derivative, or currency.
- Each "candlestick" typically shows one day, thus a one-month chart may show the 20 trading days as 20 candlesticks.
- It is similar to a bar chart in that each candlestick represents all four important pieces of information for that day: open and close in the thick body; high and low in the “candle wick”. Being densely packed with information, it tends to represent trading patterns over short periods of time, often a few days or a few trading sessions.



Color of Candlestick

- The area between the open and the close is called the *real body*, price excursions above and below the real body are *shadows* (also called *wicks*). Wicks illustrate the highest and lowest traded prices of an asset during the time interval represented. The body illustrates the opening and closing trades.
- If the asset closed higher than it opened, the body is hollow or unfilled, with the opening price at the bottom of the body and the closing price at the top. If the asset closed lower than it opened, the body is solid or filled, with the opening price at the top and the closing price at the bottom.
- Thus, the color of the candle represents the price movement relative to the prior period's close and the "fill" (solid or hollow) of the candle represents the price direction of the period in isolation (solid for a higher open and lower close; hollow for a lower open and a higher close). A black (or red) candle represents a price action with a lower closing price than the prior candle's close. A white (or green) candle represents a higher closing price than the prior candle's close.





Stock data

- Stock data can be download from (AAPL.csv)
- https://finance.yahoo.com/quote/AAPL/history/

- The file includes trading date, Open price, High price, Low price, Close price and trading volume

AAPL.csv

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	10/21/2019	59.38	60.2475	59.33	60.1275	59.53678	87247200
3	10/22/2019	60.29	60.55	59.905	59.99	59.40064	82293600
4	10/23/2019	60.525	60.81	60.305	60.795	60.19773	75828800
5	10/24/2019	61.1275	61.2	60.4525	60.895	60.29674	69275200
6	10/25/2019	60.79	61.6825	60.72	61.645	61.03938	73477200
7	10/28/2019	61.855	62.3125	61.68	62.2625	61.65081	96572800

Example source code for stock data visualization

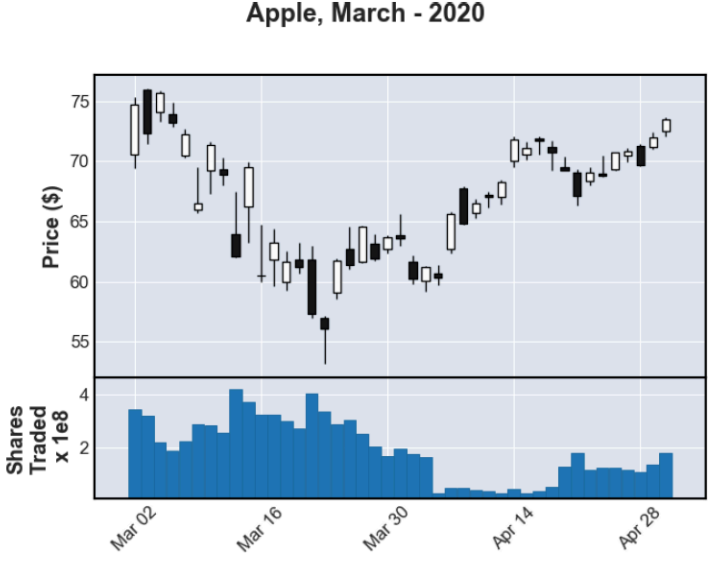
```
# example source code for Data Science Week-5
# import lib
import pandas as pd
import mplfinance as fplt

# Load the data
apple_df = pd.read_csv('AAPL.csv', index_col=0, parse_dates=True)

# Filter data to keep only March-2020 data
# into dataframe which we'll utilize for plotting.
dt_range = pd.date_range(start="2020-03-01", end="2020-04-30")
apple_df = apple_df[apple_df.index.isin(dt_range)]
apple_df.head()

# Plot the Candlestick Chart
fplt.plot(  apple_df,
            type='candle',
            title='Apple, March - 2020',
            ylabel='Price ($)',
            volume=True,
            ylabel_lower='Shares\nTraded' )
```

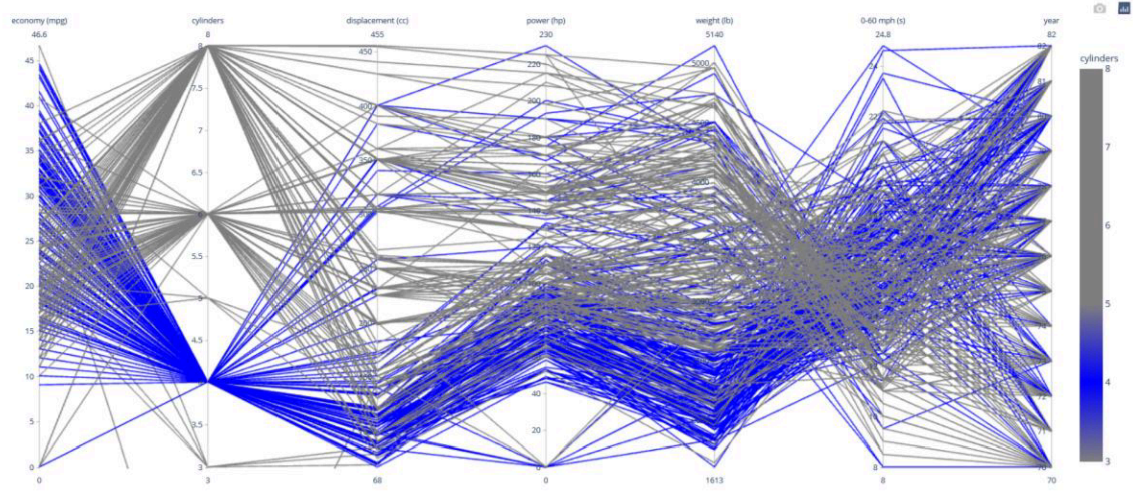
Result of stock data visualization



Excersis 1

- Extend the example source code of Parallel coordinates to generate the picture shown in the next page. The data of cars with 4 cylinders are marked by blue lines, and other data are visualized by gray lines.

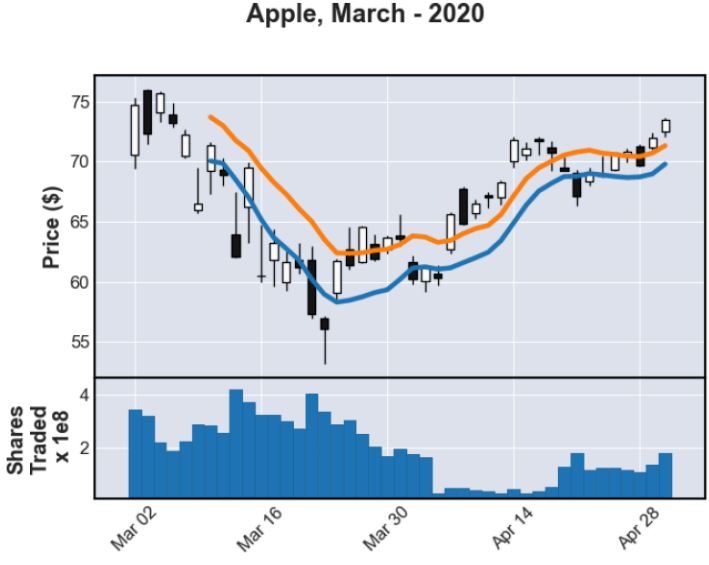
For Excersis 1



Excersis 2

Extend the source code for Candlestick to generate the picture shown in the next page. In additional to the price Candlestick chart and trading volume chart, the moving average of daily low price in the past 7 trading days and the moving average of daily high price in the past 7 trading days are needed to be plotted in the Candlestick chart area.

For Excersis 2



Summary

1. Basics of Data Visualization

2. One more step to application for data science
   - Visualization with parallel coordinates plot
   - Visualization for time series data (stock data)

Reference

- Downey, A., 2014. *Think stats: exploratory data analysis*. " O'Reilly Media, Inc.".
- VANDERPLAS, Jake. Python data science handbook: essential tools for working with data. " O'Reilly Media, Inc.", 2016.
- Antoniou, Constantinos, Loukas Dimitriou, and Francisco Pereira, eds. Mobility patterns, big data and transport analytics: tools and applications for modeling. Elsevier, 2018.
- Candlestick, https://en.wikipedia.org/wiki/Candlestick_chart