Bubble Chart

1. Bubble Chart adalah tipe grafik yang memberikan tampilan 3 dimensi dari data yang berbentuk gelembung.
2. Grafik gelembung adalah variasi dari grafik scatter chart dimana titik data diganti dengan gelembung dan dimensi tambahan dari data yang dipresentasikan dalam ukuran gelembung.
3. Grafik gelembung tidak menggunakan kategori sumbu-sumbu horizontal dan vertikal yang merupakan sumbu nilai.

Kapan Menggunakan Bubble Chart

1. Bubble Chart biasanya digunakan untuk membandingkan dan menunjukkan hubungan antara label atau lingkaran yang dikategorikan, dengan menggunakan posisi dan proporsi.
2. Gambaran keseluruhan dari bubble chart dapat digunakan untuk menganalisa pola atau korelasi.
3. Namun yang perlu diperhatikan dalam penggunaan bubble chart adalah jika terlalu banyak bubble dapat membuat grafik sulit dibaca, sehingga bubble chart memiliki kapasitas ukuran data yang terbatas.

In [1]:

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

browser_market_share = {
    'browsers': ['firefox', 'chrome', 'safari', 'edge', 'ie', 'opera'],
    'market_share': [8.61, 69.55, 8.36, 4.12, 2.76, 2.43],
    'color': ['#5A69AF', '#579E65', '#F9C784', '#FC944A', '#F24C00', '#00B825
}


class BubbleChart:
    def __init__(self, area, bubble_spacing=0):
        """
        Setup for bubble collapse.

        Parameters
        ----------
        area : array-like
            Area of the bubbles.
        bubble_spacing : float, default: 0
            Minimal spacing between bubbles after collapsing.

        Notes
        -----
        If "area" is sorted, the results might look weird.
        """
        area = np.asarray(area)
        r = np.sqrt(area / np.pi)

        self.bubble_spacing = bubble_spacing
        self.bubbles = np.ones((len(area), 4))
        self.bubbles[:, 2] = r
        self.bubbles[:, 3] = area
        self.maxstep = 2 * self.bubbles[:, 2].max() + self.bubble_spacing
        self.step_dist = self.maxstep / 2

        # calculate initial grid layout for bubbles
        length = np.ceil(np.sqrt(len(self.bubbles)))
```

```
        grid = np.arange(length) * self.maxstep
        gx, gy = np.meshgrid(grid, grid)
        self.bubbles[:, 0] = gx.flatten()[:len(self.bubbles)]
        self.bubbles[:, 1] = gy.flatten()[:len(self.bubbles)]

        self.com = self.center_of_mass()

    def center_of_mass(self):
        return np.average(
            self.bubbles[:, :2], axis=0, weights=self.bubbles[:, 3]
        )

    def center_distance(self, bubble, bubbles):
        return np.hypot(bubble[0] - bubbles[:, 0],
                        bubble[1] - bubbles[:, 1])

    def outline_distance(self, bubble, bubbles):
        center_distance = self.center_distance(bubble, bubbles)
        return center_distance - bubble[2] - \
            bubbles[:, 2] - self.bubble_spacing

    def check_collisions(self, bubble, bubbles):
        distance = self.outline_distance(bubble, bubbles)
        return len(distance[distance < 0])

    def collides_with(self, bubble, bubbles):
        distance = self.outline_distance(bubble, bubbles)
        idx_min = np.argmin(distance)
        return idx_min if type(idx_min) == np.ndarray else [idx_min]

    def collapse(self, n_iterations=50):
        """
        Move bubbles to the center of mass.

        Parameters
        ----------
        n_iterations : int, default: 50
            Number of moves to perform.
        """
        for _i in range(n_iterations):
            moves = 0
            for i in range(len(self.bubbles)):
                rest_bub = np.delete(self.bubbles, i, 0)
                # try to move directly towards the center of mass
                # direction vector from bubble to the center of mass
                dir_vec = self.com - self.bubbles[i, :2]

                # shorten direction vector to have length of 1
                dir_vec = dir_vec / np.sqrt(dir_vec.dot(dir_vec))

                # calculate new bubble position
                new_point = self.bubbles[i, :2] + dir_vec * self.step_dist
                new_bubble = np.append(new_point, self.bubbles[i, 2:4])

                # check whether new bubble collides with other bubbles
                if not self.check_collisions(new_bubble, rest_bub):
                    self.bubbles[i, :] = new_bubble
                    self.com = self.center_of_mass()
                    moves += 1
                else:
                    # try to move around a bubble that you collide with
                    # find colliding bubble
                    for colliding in self.collides_with(new_bubble, rest_bub)
                        # calculate direction vector
                        dir_vec = rest_bub[colliding, :2] - self.bubbles[i, :
```

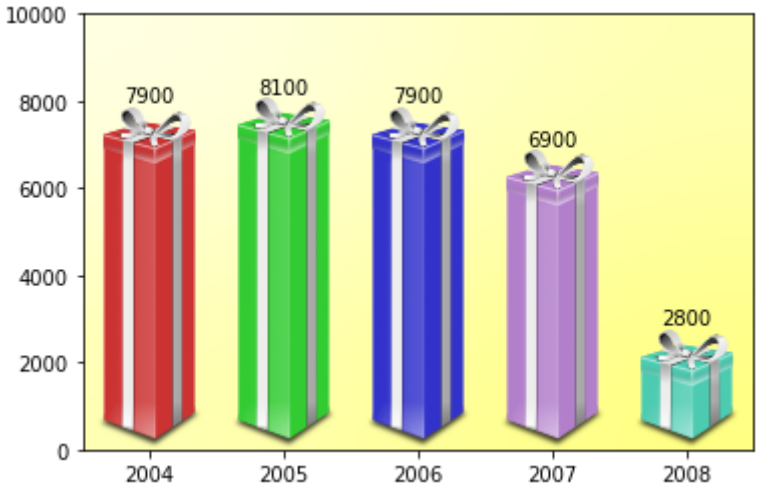
```
                        dir_vec = dir_vec / np.sqrt(dir_vec.dot(dir_vec))
                        # calculate orthogonal vector
                        orth = np.array([dir_vec[1], -dir_vec[0]])
                        # test which direction to go
                        new_point1 = (self.bubbles[i, :2] + orth *
                                      self.step_dist)
                        new_point2 = (self.bubbles[i, :2] - orth *
                                      self.step_dist)
                        dist1 = self.center_distance(
                            self.com, np.array([new_point1]))
                        dist2 = self.center_distance(
                            self.com, np.array([new_point2]))
                        new_point = new_point1 if dist1 < dist2 else new_poir
                        new_bubble = np.append(new_point, self.bubbles[i, 2:4
                        if not self.check_collisions(new_bubble, rest_bub):
                            self.bubbles[i, :] = new_bubble
                            self.com = self.center_of_mass()

            if moves / len(self.bubbles) < 0.1:
                self.step_dist = self.step_dist / 2

    def plot(self, ax, labels, colors):
        """
        Draw the bubble plot.

        Parameters
        ----------
        ax : matplotlib.axes.Axes
        labels : list
            Labels of the bubbles.
        colors : list
            Colors of the bubbles.
        """
        for i in range(len(self.bubbles)):
            circ = plt.Circle(
                self.bubbles[i, :2], self.bubbles[i, 2], color=colors[i])
            ax.add_patch(circ)
            ax.text(*self.bubbles[i, :2], labels[i],
                    horizontalalignment='center', verticalalignment='center')


bubble_chart = BubbleChart(area=browser_market_share['market_share'],
                           bubble_spacing=0.1)

bubble_chart.collapse()

fig, ax = plt.subplots(subplot_kw=dict(aspect="equal"))
bubble_chart.plot(
    ax, browser_market_share['browsers'], browser_market_share['color'])
ax.axis("off")
ax.relim()
ax.autoscale_view()
ax.set_title('Browser market share')

plt.show()
```

Ribbon Box

In [2]:

```
import numpy as np

from matplotlib import cbook, colors as mcolors
from matplotlib.image import AxesImage
import matplotlib.pyplot as plt
from matplotlib.transforms import Bbox, TransformedBbox, BboxTransformTo


class RibbonBox:

    original_image = plt.imread(
        cbook.get_sample_data("Minduka_Present_Blue_Pack.png"))
    cut_location = 70
    b_and_h = original_image[:, :, 2:3]
    color = original_image[:, :, 2:3] - original_image[:, :, 0:1]
    alpha = original_image[:, :, 3:4]
    nx = original_image.shape[1]

    def __init__(self, color):
        rgb = mcolors.to_rgba(color)[:3]
        self.im = np.dstack(
            [self.b_and_h - self.color * (1 - np.array(rgb)), self.alpha])

    def get_stretched_image(self, stretch_factor):
        stretch_factor = max(stretch_factor, 1)
        ny, nx, nch = self.im.shape
        ny2 = int(ny*stretch_factor)
        return np.vstack(
            [self.im[:self.cut_location],
             np.broadcast_to(
                 self.im[self.cut_location], (ny2 - ny, nx, nch)),
             self.im[self.cut_location:]])


class RibbonBoxImage(AxesImage):
    zorder = 1

    def __init__(self, ax, bbox, color, *, extent=(0, 1, 0, 1), **kwargs):
        super().__init__(ax, extent=extent, **kwargs)
        self._bbox = bbox
        self._ribbonbox = RibbonBox(color)
        self.set_transform(BboxTransformTo(bbox))

    def draw(self, renderer, *args, **kwargs):
        stretch_factor = self._bbox.height / self._bbox.width
```

```
        ny = int(stretch_factor*self._ribbonbox.nx)
        if self.get_array() is None or self.get_array().shape[0] != ny:
            arr = self._ribbonbox.get_stretched_image(stretch_factor)
            self.set_array(arr)

        super().draw(renderer, *args, **kwargs)


def main():
    fig, ax = plt.subplots()

    years = np.arange(2004, 2009)
    heights = [7900, 8100, 7900, 6900, 2800]
    box_colors = [
        (0.8, 0.2, 0.2),
        (0.2, 0.8, 0.2),
        (0.2, 0.2, 0.8),
        (0.7, 0.5, 0.8),
        (0.3, 0.8, 0.7),
    ]

    for year, h, bc in zip(years, heights, box_colors):
        bbox0 = Bbox.from_extents(year - 0.4, 0., year + 0.4, h)
        bbox = TransformedBbox(bbox0, ax.transData)
        ax.add_artist(RibbonBoxImage(ax, bbox, bc, interpolation="bicubic"))
        ax.annotate(str(h), (year, h), va="bottom", ha="center")

    ax.set_xlim(years[0] - 0.5, years[-1] + 0.5)
    ax.set_ylim(0, 10000)

    background_gradient = np.zeros((2, 2, 4))
    background_gradient[:, :, :3] = [1, 1, 0]
    background_gradient[:, :, 3] = [[0.1, 0.3], [0.3, 0.5]]  # alpha channel
    ax.imshow(background_gradient, interpolation="bicubic", zorder=0.1,
              extent=(0, 1, 0, 1), transform=ax.transAxes, aspect="auto")

    plt.show()


main()
```



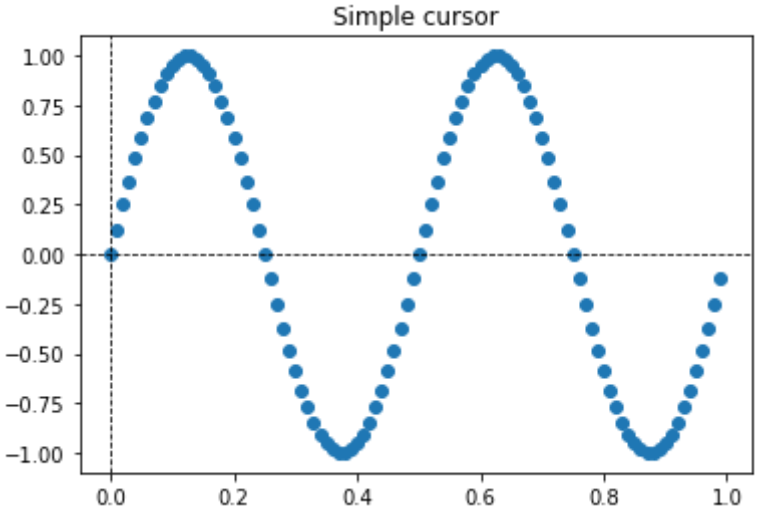
Cross Hair Cursor

In [3]:

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

```
class Cursor:
    """
    A cross hair cursor.
    """
    def __init__(self, ax):
        self.ax = ax
        self.horizontal_line = ax.axhline(color='k', lw=0.8, ls='--')
        self.vertical_line = ax.axvline(color='k', lw=0.8, ls='--')
        # text location in axes coordinates
        self.text = ax.text(0.72, 0.9, '', transform=ax.transAxes)

    def set_cross_hair_visible(self, visible):
        need_redraw = self.horizontal_line.get_visible() != visible
        self.horizontal_line.set_visible(visible)
        self.vertical_line.set_visible(visible)
        self.text.set_visible(visible)
        return need_redraw

    def on_mouse_move(self, event):
        if not event.inaxes:
            need_redraw = self.set_cross_hair_visible(False)
            if need_redraw:
                self.ax.figure.canvas.draw()
        else:
            self.set_cross_hair_visible(True)
            x, y = event.xdata, event.ydata
            # update the line positions
            self.horizontal_line.set_ydata(y)
            self.vertical_line.set_xdata(x)
            self.text.set_text('x=%1.2f, y=%1.2f' % (x, y))
            self.ax.figure.canvas.draw()


x = np.arange(0, 1, 0.01)
y = np.sin(2 * 2 * np.pi * x)

fig, ax = plt.subplots()
ax.set_title('Simple cursor')
ax.plot(x, y, 'o')
cursor = Cursor(ax)
fig.canvas.mpl_connect('motion_notify_event', cursor.on_mouse_move)
```

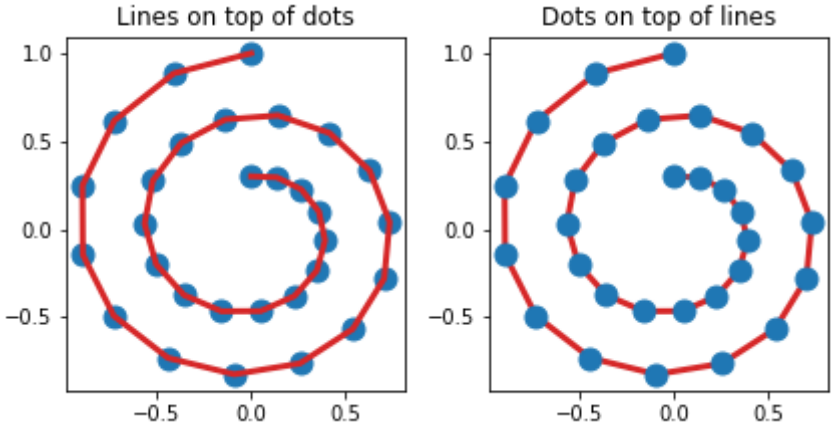
Out[3]: 5



Zorder

In [4]:
```
import matplotlib.pyplot as plt
```

```
import numpy as np

r = np.linspace(0.3, 1, 30)
theta = np.linspace(0, 4*np.pi, 30)
x = r * np.sin(theta)
y = r * np.cos(theta)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3.2))

ax1.plot(x, y, 'C3', lw=3)
ax1.scatter(x, y, s=120)
ax1.set_title('Lines on top of dots')

ax2.plot(x, y, 'C3', lw=3)
ax2.scatter(x, y, s=120, zorder=2.5)  # move dots on top of line
ax2.set_title('Dots on top of lines')

plt.tight_layout()
```



In [ ]: