andyly

March 27, 2022

# 1 Where to focus a marketing campaign?

## 1.1 Background

You are a data analyst at a crowdfunding site. For the next quarter, your company will be running a marketing campaign. The marketing manager wants to target those segments that have donated the most in the past year. She turned to you to help her with her upcoming meeting with the CEO.

## 1.2 The data

You have access to the following information:

### Historic crowdfunding donations

- "category" - "Sports", "Fashion", "Technology", etc.
- "device" - the type of device used.
- "gender" - gender of the user.
- "age range" - one of five age brackets.
- "amount" - how much the user donated in Euros.

[120]:
```
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv('./data/crowdfunding.csv')
df.head()
```

[120]:

| | category | device | gender | age | amount |
|---|---|---|---|---|---|
| 0 | Fashion | iOS | F | 45-54 | 61.0 |
| 1 | Sports | android | M | 18-24 | 31.0 |
| 2 | Technology | android | M | 18-24 | 39.0 |
| 3 | Technology | iOS | M | 18-24 | 36.0 |
| 4 | Sports | android | M | 18-24 | 40.0 |

## 1.3 Challenge

Create a **single** visualization that the marketing manager can use to explore the data. Include:

1. What are the top three categories in terms of total donations?

2. What device type has historically provided the most contributions?
3. What age bracket should the campaign target?

### 1.4 Judging criteria

This is a community-based competition. The top 5 most upvoted entries will win.

The winners will receive DataCamp merchandise.

### 1.5 Checklist before publishing

- Rename your workspace to make it descriptive of your work. N.B. you should leave the notebook name as notebook.ipynb.
- Remove redundant cells like the judging criteria, so the workbook is focused on your answers.
- Check that all the cells run without error.

### 1.6 Time is ticking. Good luck!

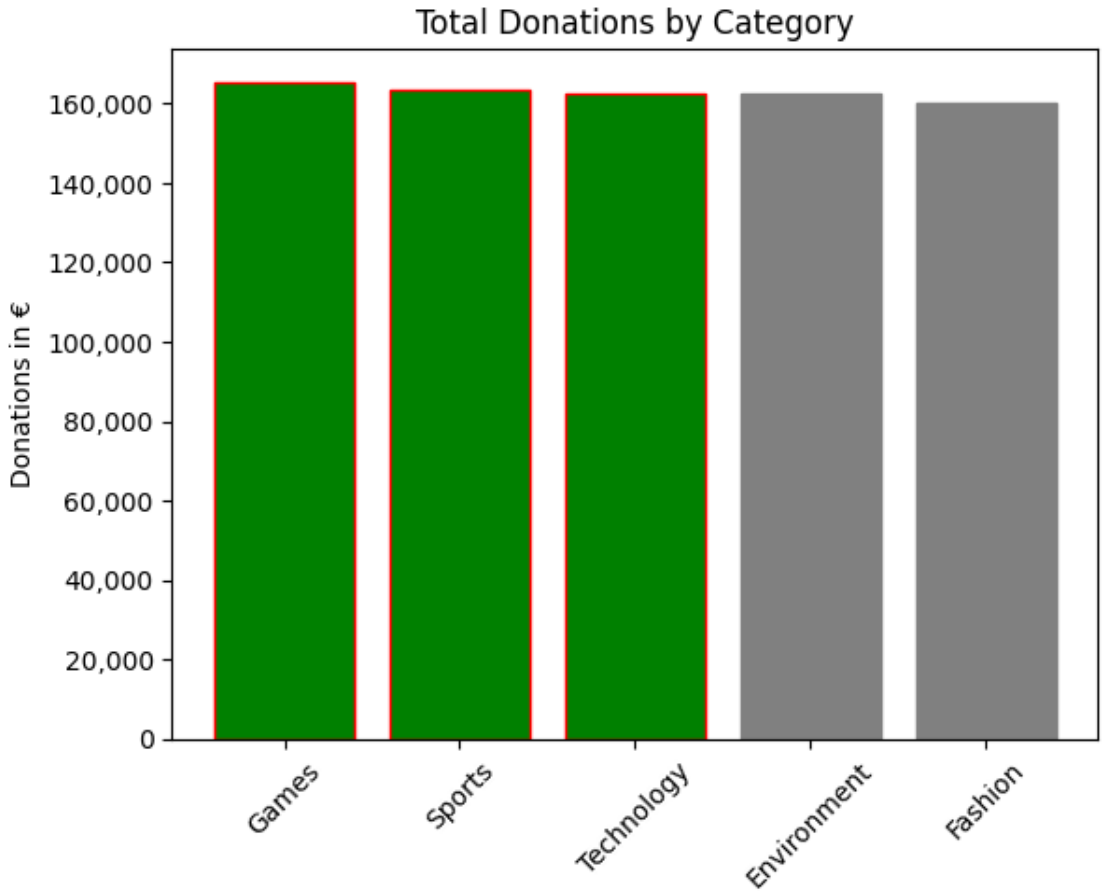
[121]:
```
# 1. What are the top three categories in terms of total donations?
# Separated by small margins
df.category.unique()
categories = df.groupby(['category']).sum().sort_values(by=['amount'],␣
↪ascending=False)
# The top 3 donations by category are: Games, Sports, and Technology
categories['percentages'] = round(categories / sum(categories.amount) * 100, 2)
categories
```

[121]:

| category | amount | percentages |
|---|---|---|
| Games | 165483.0 | 20.33 |
| Sports | 163528.0 | 20.09 |
| Technology | 162731.0 | 19.99 |
| Environment | 162376.0 | 19.95 |
| Fashion | 159952.0 | 19.65 |

[122]:
```
# fix these colors
five_cats = ['Games', 'Sports', 'Technology', 'Environment', 'Fashion']
five_percs = [165483, 163528, 162731, 162376, 159952]
ranks = [1, 2, 3, 4, 5]
colors = ['red' if i > 162730 else 'grey' for i in five_percs]

plt.bar(five_cats, five_percs, color = ['green' if i > 162730 else 'grey' for i␣
↪in five_percs], edgecolor=colors)

current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels(['{:,.0f}'.format(x) for x in current_values])
plt.xticks(rotation = 45)
plt.ylabel("Donations in €")
```

```
plt.title("Total Donations by Category")
plt.show()
```



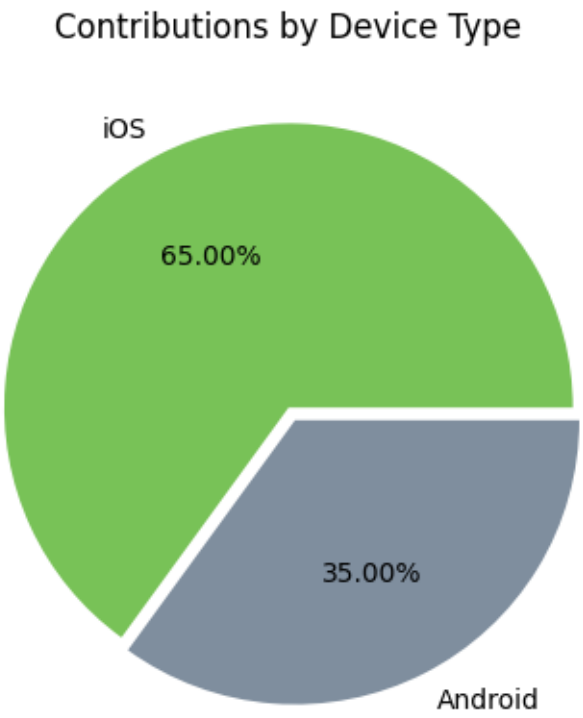

[123]:
```
# bubble won't work because you don't have both qualitative x and y
# how do i exagerrate the values to show greater differences to plot
```

[124]:
```
# 2. What device type has historically provided the most contributions?

df.device.nunique()
devices = df.groupby(['device']).sum()
# create a new column with the percentages of total contributions
devices['percentages'] = round(devices / sum(devices.amount) * 100, 2)
devices
```

[124]:

| device | amount | percentages |
|---|---|---|
| android | 283545.0 | 34.83 |
| iOS | 530525.0 | 65.17 |

[125]:
```
# bold the labels to make it pop more
perc = [65, 35]
company_colors = ['#78C257', '#7f8e9e']
plt.pie(perc, explode= (0, 0.05), labels = ["iOS", "Android"], colors =␣
↪company_colors, autopct='%.2f%%')
plt.title("Contributions by Device Type")
plt.show()
```




[126]:
```
grouped_ages = df.groupby(['age']).sum().sort_values(by=['age', 'amount'])
print(grouped_ages)
```

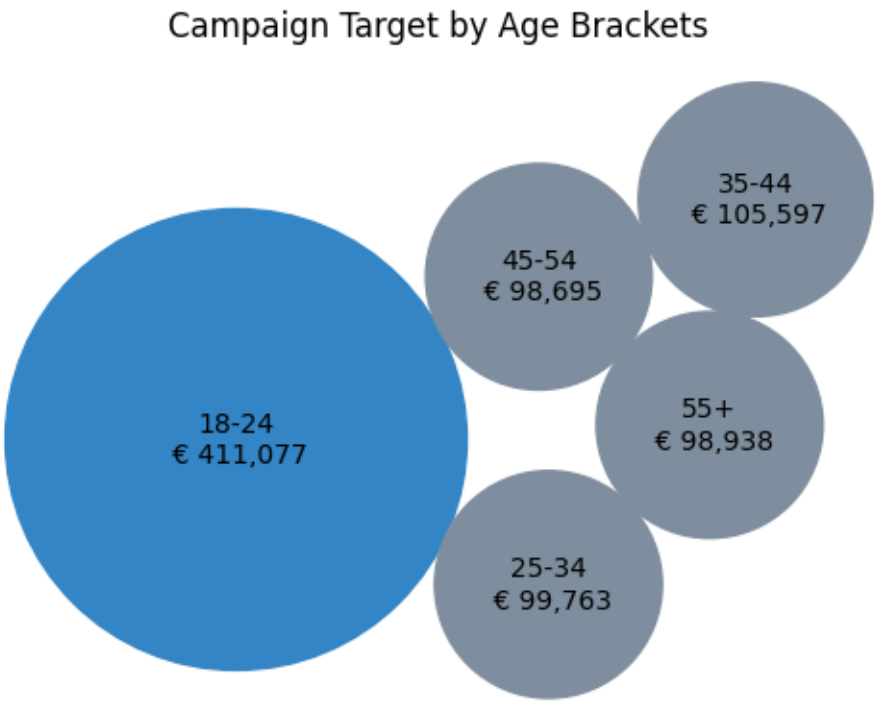
```
         amount
age
18-24  411077.0
25-34   99763.0
35-44  105597.0
45-54   98695.0
55+     98938.0
```

[127]:
```
# 3. What age bracket should the campaign target?
camp_lib = {
    'age_brackets': ['18-24', '25-34', '35-44', '45-54', '55+'],
```

```
    'amount': [411077, 99763, 105597, 98695, 98938],
    'color': ['#3385c6', '#7f8e9e', '#7f8e9e', '#7f8e9e', '#7f8e9e']
        }

combine = ['18-24\n € 411,077', '25-34\n € 99,763', '35-44\n € 105,597',␣
 ↪'45-54\n € 98,695', '55+\n € 98,938']

class BubbleChart:
    def __init__(self, area, bubble_spacing=0):
        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

        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):

        for i in range(n_iterations):
            moves=0
            for i in range(len(self.bubbles)):
                rest_bub=np.delete(self.bubbles, i, 0)
                dir_vec = self.com - self.bubbles[i, :2]

                dir_vec = dir_vec / np.sqrt(dir_vec.dot(dir_vec))

                new_point = self.bubbles[i, :2] + dir_vec * self.step_dist
                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()
                    moves += 1
                else:
                    for colliding in self.collides_with(new_bubble, rest_bub):
                        dir_vec = rest_bub[colliding, :2] - self.bubbles[i, :2]
                        dir_vec = dir_vec / np.sqrt(dir_vec.dot(dir_vec))
                        orth = np.array([dir_vec[1], -dir_vec[0]])

                        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_point2
                        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):
        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 = camp_lib['amount'], bubble_spacing = 0.1)

bubble_chart.collapse()

fig, ax = plt.subplots(subplot_kw=dict(aspect='equal'))
bubble_chart.plot(
    ax, combine, camp_lib['color'])
ax.axis('off')
ax.relim()
ax.autoscale_view()
ax.set_title("Campaign Target by Age Brackets")

plt.show()
```




[128]:
```
# I want to know what percentage the 18-24 age bracket dominated by (over 50.5␣
 ↪percent of this dataset)
```

```
df['age'].value_counts(normalize=True)
```

[128]:
```
18-24    0.505325
35-44    0.129538
45-54    0.122568
55+      0.121745
25-34    0.120825
Name: age, dtype: float64
```

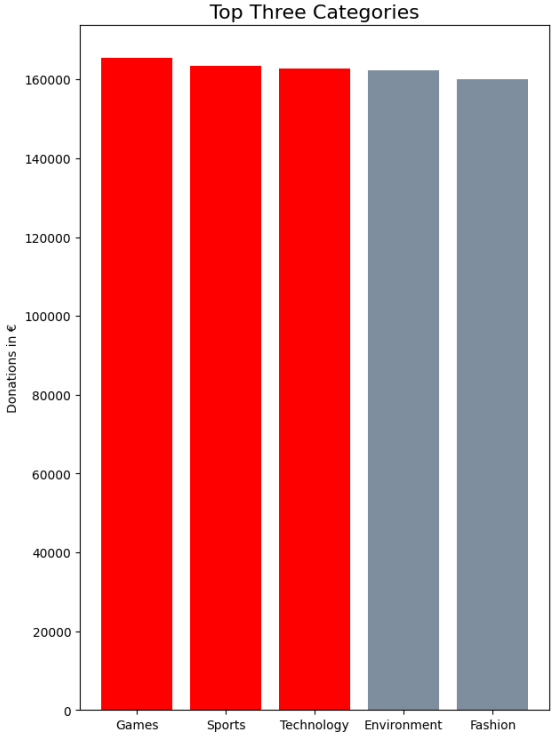
[129]:
```
df['age'].describe()
```

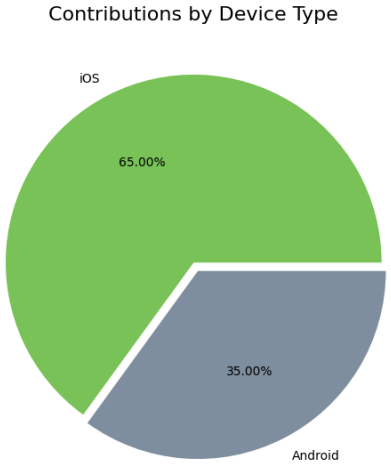
[129]:
```
count     20658
unique        5
top       18-24
freq      10439
Name: age, dtype: object
```

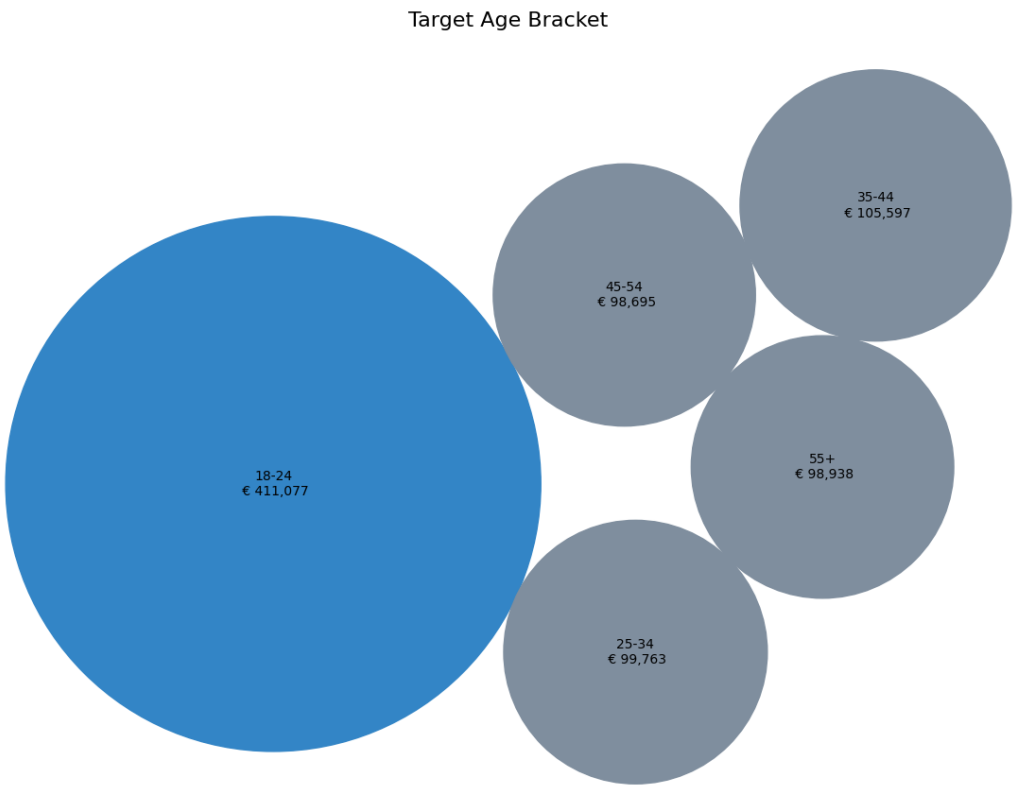
[130]:
```
# now to combine the 3 plots into a single visualization
```

[131]:
```
fig, ax = plt.subplots(1, 2, figsize = (15, 10))
fig.suptitle("Analyzing customer segments for a marketing campaign",␣
 ↪fontsize=24)

ax[0].bar(five_cats, five_percs, color = ['red' if i > 162730 else '#7f8e9e'␣
 ↪for i in five_percs])
ax[0].set_ylabel('Donations in €')
ax[0].set_title('Top Three Categories', fontsize = 16)

ax[1].pie(perc, explode= (0, 0.05), labels = ["iOS", "Android"], colors =␣
 ↪company_colors, autopct='%.2f%%')
ax[1].set_title('Contributions by Device Type', fontsize = 16)

fig, ax = plt.subplots(subplot_kw=dict(aspect='equal'), figsize=(12, 10))
bubble_chart.plot(
    ax, combine, camp_lib['color'])
ax.axis('off')
ax.relim()
ax.autoscale_view()
ax.set_title("Target Age Bracket", fontsize = 16)

fig.tight_layout()
plt.show()
```

## Analyzing customer segments for a marketing campaign




Target Age Bracket
18-24
€ 411,077
45-54
€ 98,695
35-44
€ 105,597
55+
€ 98,938
25-34
€ 99,763