```
# === plit_drills.ipynb ===

%matplotlib notebook

#%%
# Import Dependencies
import numpy as np
import matplotlib.pyplot as plt


#%%
# What kinds of plots match the datasets below?
# Create a plot for each of the datasets above, making
# certain to provide each chart with a title and labels


#%%
# DATASET 1
# ------------------------------------------------------------------------
# This dataset is ideal for a bar chart since the programmer is provided with
# nothing but a list of strings - gym names - and a list of integers - gym
# memberships - that should be compared against one-another
# ------------------------------------------------------------------------
gyms = ["Crunch", "Planet Fitness", "NY Sports Club", "Rickie's Gym"]
members = [49, 92, 84, 53]


#%%
x_axis = np.arange(0, len(gyms))
tick_locations = []
for x in x_axis:
    tick_locations.append(x)

plt.title("NYC Gym Popularity")
plt.xlabel("Gym Name")
plt.ylabel("Number of Members")

plt.xlim(-0.75, len(gyms)-.25)
plt.ylim(0, max(members) + 5)

plt.bar(x_axis, members, facecolor="red", alpha=0.75, align="center")

# ------------------------------------------------------------------------
# so as to ensure the graph is as aesthetically pleasing as possible, the tick
# locations for the X axis are modified so that they fall in the center of
# their associated bar when the bars are aligned to the edge of the chart. The
# limits of the X and Y axes are then also modified to ensure there is some
# separation between the bars and the edge of the chart.
# ------------------------------------------------------------------------
plt.xticks(tick_locations, gyms)
plt.show()


#%%
# DATASET 2
```

```
# ------------------------------------------------------------------------
# this dataset fits a line chart best since the values within the lists
# change over time in relation to one-another
# ------------------------------------------------------------------------
x_lim = 2 * np.pi
x_axis = np.arange(0, x_lim, 0.1)
sin = np.sin(x_axis)


#%%
plt.title("Sin from 0 to 2$\pi$")
plt.xlabel("Real Numbers from 0 to 2$\pi$")
plt.ylabel("sin(x)")

plt.hlines(0, 0, x_lim, alpha=0.2)
plt.xlim(0, x_lim)
plt.ylim(-1.25, 1.25)

# ------------------------------------------------------------------------
# There is not as much aesthetic editing that needs to be done with this chart
# other than adding a horizontal line to the chart where the Y axis is equal
# to 0 so that it is easy to tell when a value is positive or negative
# ------------------------------------------------------------------------
plt.plot(x_axis, sin, marker="o", color="red", linewidth=1)
plt.show()


#%%
# DATASET 3
# ------------------------------------------------------------------------
# dataset obviously fits that which would be used for a pie chart, the only
# thing that differentiates it from the first is the inclusion of the "colors"
# list and "explode" tuple. Still, since pie charts are helpful when comparing
# parts of a whole, the pie chart provides a different perspective than the
# bar chart from earlier
# ------------------------------------------------------------------------

gyms = ["Crunch", "Planet Fitness", "NY Sports Club", "Rickie's Gym"]
members = [49, 92, 84, 53]
colors = ["yellowgreen", "red", "lightcoral", "lightskyblue"]
explode = (0, 0.05, 0, 0)


#%%
# ------------------------------------------------------------------------
# note how the axes are being set to "equal" so that the pie chart is circular
# and that the parameter of `autopct=%1.1%%` is passed into the `plt.pie()`
# method so as to convert the values within the "members" list into percentages
# with a single decimal point
# ------------------------------------------------------------------------
plt.title("NYC Gym Popularity")
plt.pie(members, explode=explode, labels=gyms, colors=colors,
        autopct="%1.1f%%", shadow=True, startangle=90)
plt.axis("equal")
plt.show()
```

```
111
112
113 #%%
114 # DATASET 4
115 # ------------------------------------------------------------------------
116 # dataset compares the relationship between two lists with unique values.
117 # Because of this, a scatter plot is the ideal method through which to
118 # visualize the relationship
119 # ------------------------------------------------------------------------
120 x_axis = np.arange(0, 10, 0.1)
121 times = []
122 for x in x_axis:
123     times.append(x * x + np.random.randint(0, np.ceil(max(x_axis))))
124
125
126 #%%
127 # ------------------------------------------------------------------------
128 # Scatter plots require very little in the way of aesthetic styling and, as
129 # such, the chart really only needs to be drawn in order to look pleasing
130 # ------------------------------------------------------------------------
131 plt.title("Running Time of FakeSort for Sample Input Sizes")
132 plt.xlabel("Length of Input Array")
133 plt.ylabel("Time to Sort (s)")
134
135 plt.scatter(x_axis, times, marker="o", color="red")
136 plt.show()
137
138
139
140
```