Data Analysis – Analyzing and Visualizing

15-110 – Wednesday 04/21

Announcements

- **Quiz5** happening **today!**

- **Check6-1** due **Friday at noon EST**
  - Collab form: https://forms.gle/vJo5XUBHC1nthDy68

Last Time

Last week we discussed the **data analysis process** and went over several methods for **reading, representing,** and **organizing** data.

This time, we'll talk more about what we can **do** with that data once we've processed it.

Learning Goals

- Perform **basic analyses** on data, including calculating **statistics** and **probabilities**, to answer simple questions

- Choose an appropriate **visualization** to create based on the number of **dimensions** and **data types**

- Create simple **matplotlib visualizations** that show the state of a dataset using **APIs** and **examples**

Analysis

Basic Data Analyses – Statistics Library

There are many basic analyses we can run on features in data to get a sense of what the data means. You've learned about some of them already in math or statistics classes, such as **mean, median,** and **mode**.

You can implement these in Python yourself, but you don't have to! There's already a **statistics** library that does this for you.

```
import statistics

data = [41, 65, 64, 50, 45, 13, 29, 14, 7, 14]

statistics.mean(data) # 34.2
statistics.median(data) # 35.0
statistics.mode(data) # 14
```

Example: Analyzing Ice Cream Data

We've now cleaned the ice cream dataset from last week. Let's use it as a running example of how to perform analyses.

Here's a bit of code from last time to load and represent the dataset:

```
import csv
def readData(filename):
    f = open(filename, "r")
    # Semester, 3 orig, 3 cleaned, 3 categories
    data = list(csv.reader(f))
    return data
```

Example: Statistics of Ice Cream

We can start by measuring the **statistics** of the ice cream dataset.

The data is text, so we must turn it into numbers before performing analyses. Try counting the number of favorite ice creams that fall into a specific category for each person and putting those counts into a list to analyze.

The `count` method is handy here too if we narrow down the data being counted first!

```
def getFlavorCounts(data, flavor):
    counts = []
    firstCol = data[0].index("#1 category")
    for i in range(1, len(data)): # skip header
        line = data[i]
        # only include categories
        categories = line[firstCol:firstCol+3]
        count = categories.count(flavor)
        counts.append(count)
    return counts
```

```
import statistics
d = readData("all-icecream.csv")
statistics.mean(getFlavorCounts(d, "chocolate"))
```

Calculating Probabilities

You'll also often want to calculate probabilities based on your data.

In general, the probability that a certain data type occurs in a dataset is the count of how often it occurred, divided by the total number of data points.

Probability:

```
lst.count(item) / len(lst)
```

Conditional probability (the probability of something occurring given another factor) is slightly more complicated. Create a modified version of the list that contains only those elements with that factor; then you can use the same equation.

```
newLst = []
for x in lst:
    if meetsProperty(x):
        newLst.append(x)
newLst.count(item) / len(newLst)
```

Example: Probabilities of Ice Cream

To calculate ice cream probabilities, think back to the code we wrote in the machine learning lecture.

Data analysis and machine learning share a lot in common!

```
# Probability that a flavor is chosen
def getClassProb(data, flavor):
    count = 0
    for line in data:
        if line[2] == flavor:
            count += 1
    return count / len(data)

# Probability that 1st/2nd favorite is X given that
# 3rd favorite is C. Load data from CSV.
def getCondProb(data, priorFlavor, thirdFlavor, priorIndex):
    count = 0
    total = 0
    for line in data:
        if line[2] == thirdFlavor:
            total += 1 # only count entries with third flavor
            if line[priorIndex] == priorFlavor:
                count += 1
    return count / total
```

More Analysis Methods

There's plenty of other data analysis methods we could cover – bucketing, detecting outliers, dealing with missing data – but what kind of method you need will depend entirely on the context of the problem you're solving.

Sometimes we may want to investigate a dataset more broadly. For example, how many times does each individual flavor occur in any of a person's preferences?

Example: Total Preferences

Create a dictionary mapping ice cream flavors to counts and iterate through the dataset to count all the flavors. We could then use the find-best algorithm to find the most popular flavor.

```
def getIceCreamCounts(data):
    iceCreamDict = { }
    for i in range(1, len(data)): # skip header
        firstCol = data[0].index("#1 cleaned") # only cleaned flavors
        for j in range(firstCol, firstCol+3):
            flavor = data[i][j]
            if flavor not in iceCreamDict:
                iceCreamDict[flavor] = 0
            iceCreamDict[flavor] += 1
    return iceCreamDict
```

Activity: Get Flavor Probabilities

You do: how could we adapt `getIceCreamCounts` to instead calculate the **probability** that each flavor is among someone's favorites?

What if we only wanted to get the probability that a flavor is a person's **#1 favorite**?

Visualization

Exploration vs. Presentation

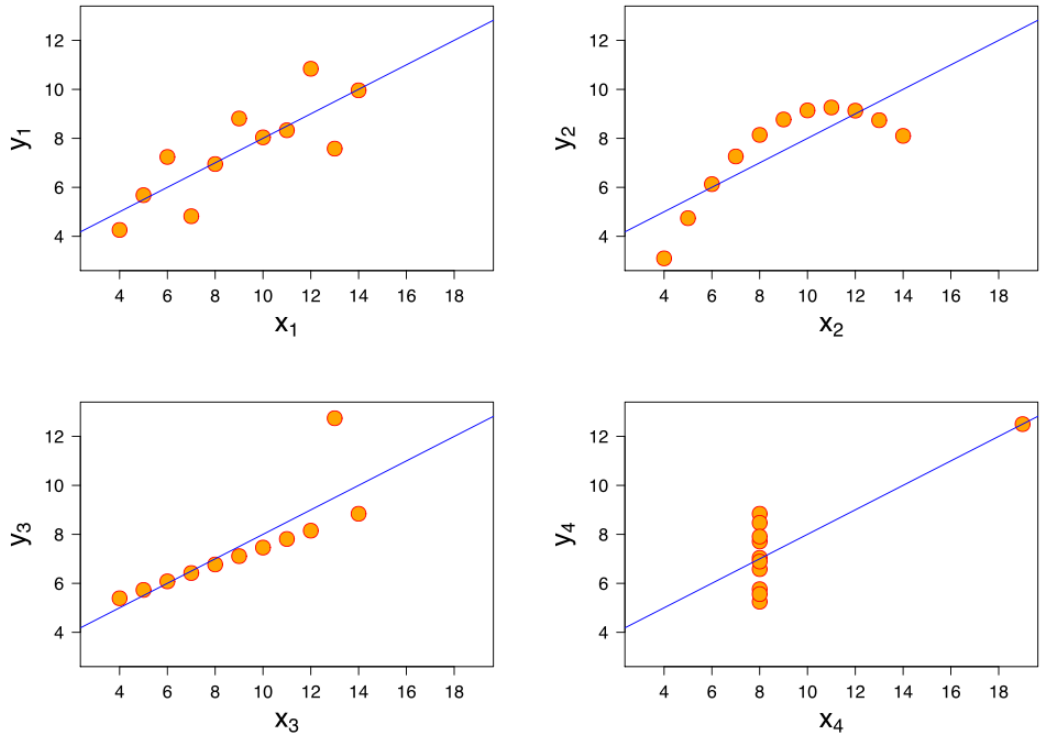
Data Visualization is the process of taking a set of data and representing it in a visual format. Whenever you've made charts or graphs in past math or science classes, you've visualized data!

Visualization is used for two primary purposes: **exploration** and **presentation**.

Data Exploration

In data exploration, charts created from data can provide information about that data beyond what is found in simple analyses alone.

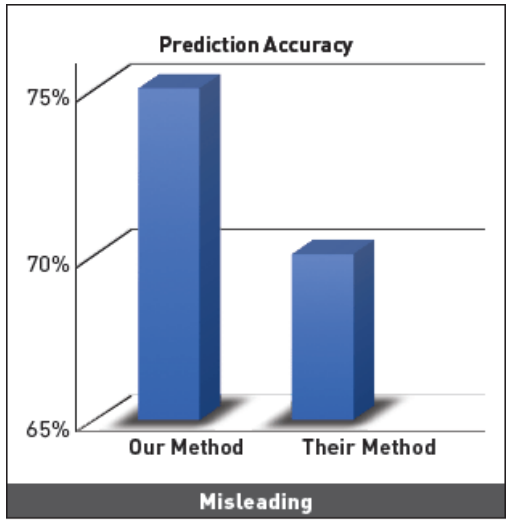
For example, the four graphs to the right all have the same mean and the same best-fit linear regression. But they tell very different stories.

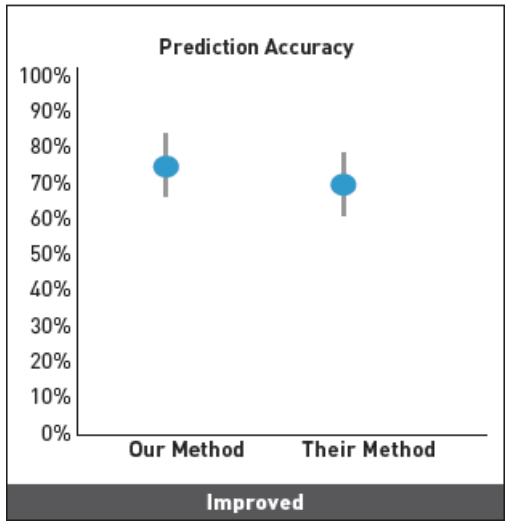


Data Presentation

In data presentation, you've already found an interesting pattern in the data and you need to make that pattern easily visible to other people.

In order to choose the best visualization for the job, consider the **type** of the data you're presenting (categorical, ordinal, or numerical), and how many **dimensions** of data you need to visualize.





https://interactions.acm.org/archive/view/july-august-2018/the-good-the-bad-and-the-biased

One-Dimensional Data

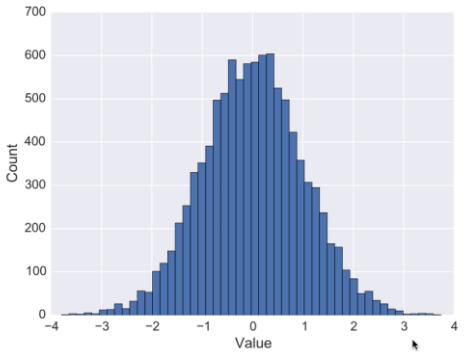
A **one-dimensional visualization** only visualizes a single feature of the dataset. For example:

"I want to know how many of each **product type** are in my data"

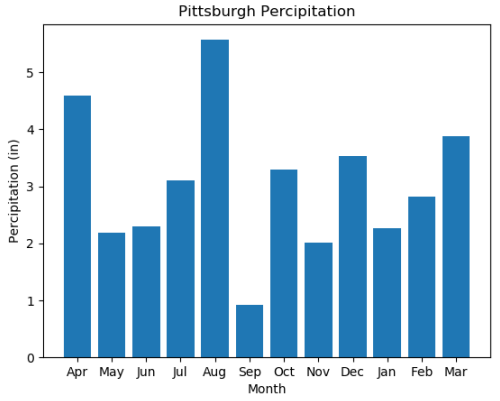
"I want to know the proportion of **people who have cats** in my data"

Charts for One-Dimensional Data

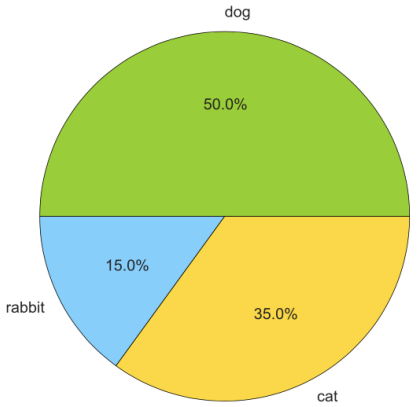
To visualize **numerical** data, use a **histogram**.



To visualize **ordinal** data, use a **bar chart**.



To visualize **categorical** data, use a **pie chart**.



Two-Dimensional Data

A **two-dimensional visualization** shows how two features in the dataset relate to each other. For example:

"I want to know the **cost** of each **product category** that we have"

"I want to know the **weight** of the animals that people own, by **pet species**"

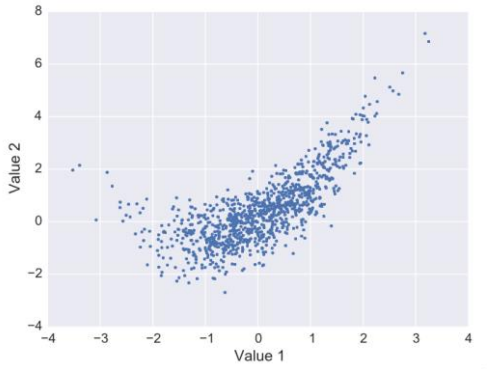
"I want to know how the **size** of the product affects **the cost of shipping**"

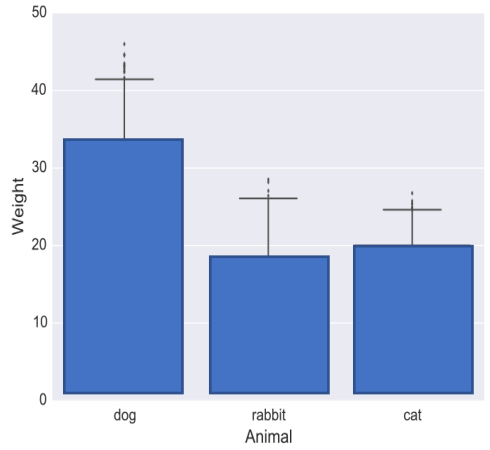
Charts for Two-Dimensional Data

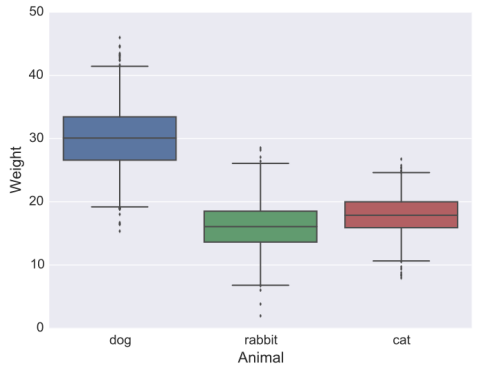
To analyze **numerical x numerical** data, use a **scatter plot**.

To analyze **numerical x ordinal/categorical** data, use a **bar chart** for averages or a **box-and-whiskers plot** for ranges.

It is difficult to analyze **ordinal/categorical x ordinal/categorical** data visually; use a table instead.







Three-Dimensional Data

A **three-dimensional** visualization tries to show the relationship between three different features at the same time. For example:

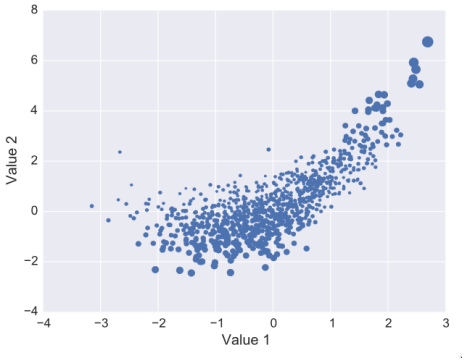
"I want to know the **cost** and the **development time** by **product category**"

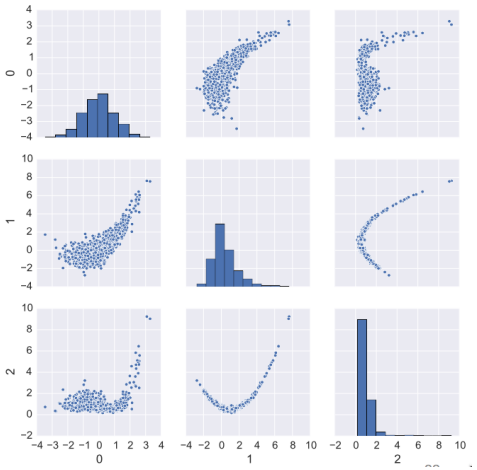
"I want to know the **weight** of the animals that people own and how much they **cost**, by **pet species**"

"I want to know how the **size** of the product and the **manufacturing location** affects **the cost of shipping**"

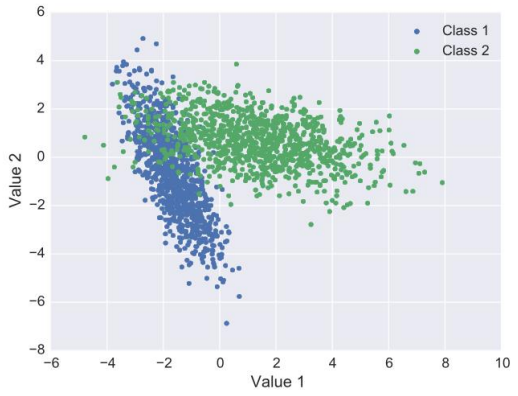
Charts for Three-Dimensional Data

To analyze **numerical x numerical x numerical** data, use a **bubble plot** to compare all three or a **scatter plot matrix** to compare all the pairs.





To analyze **numerical x numerical x ordinal/categorical** data, use a **colored scatter plot**.



Activity: Pick a Visualization

You do: for each of the problem prompts, determine the number of dimensions, then pick the **best** visualization to use based on the data types.

- graph the % of people who have gotten COVID vs. the % of people who have been vaccinated, separated by state
- graph the distribution of grades in a class
- graph the ages of pets at a shelter compared to the species of pets

Coding Visualizations with Matplotlib

Matplotlib Makes Visualizations

The **matplotlib** library can be used to generate interesting visualizations in Python.

Unlike the previous libraries we've discussed, matplotlib is **external** – you need to install it on your machine to run it. Use the `pip` command to do this.

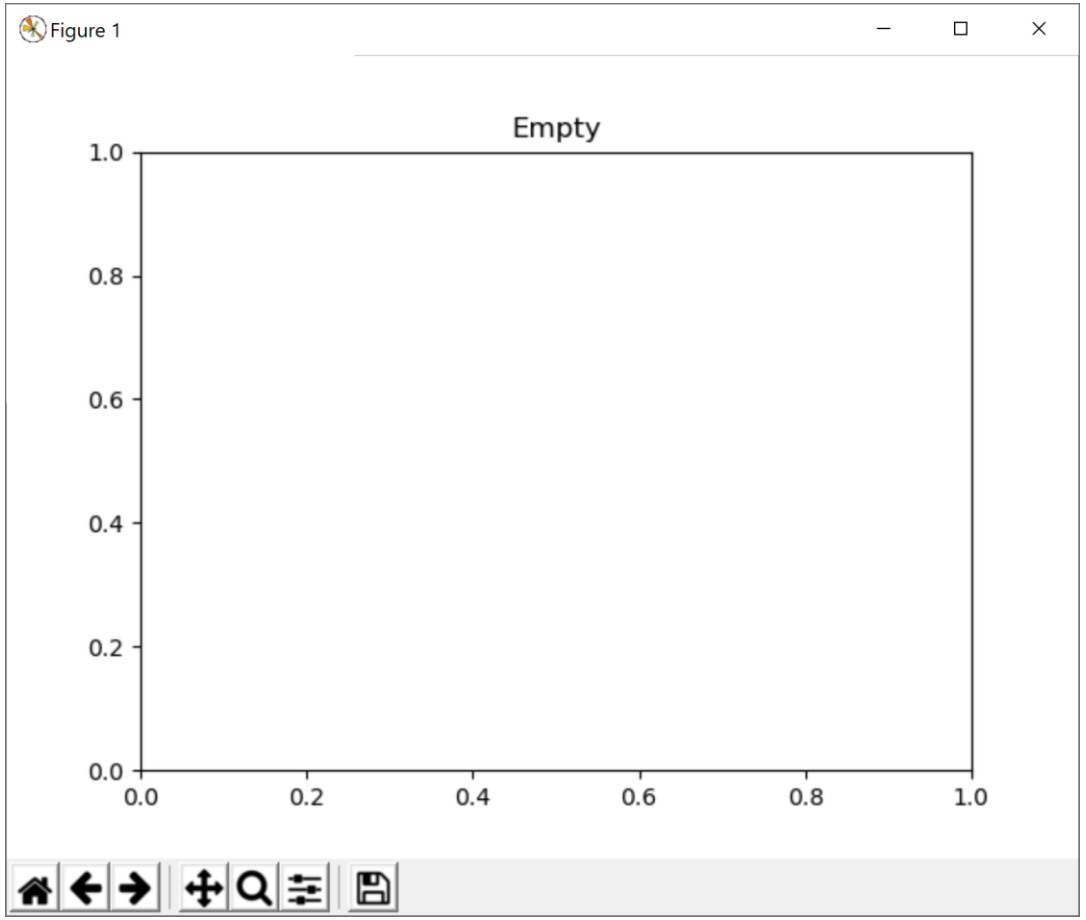
```
pip install matplotlib
```

Draw Visualizations on the Plot

Matplotlib visualizations can be broken down into several components. We'll mainly care about one: the **plot** (called `plt`). This is like Tkinter's canvas, except that we'll draw visualizations on it instead of shapes.

We can construct an (almost) empty plot with the following code. Note that matplotlib comes with built-in buttons that let you zoom, move data around, and save images.

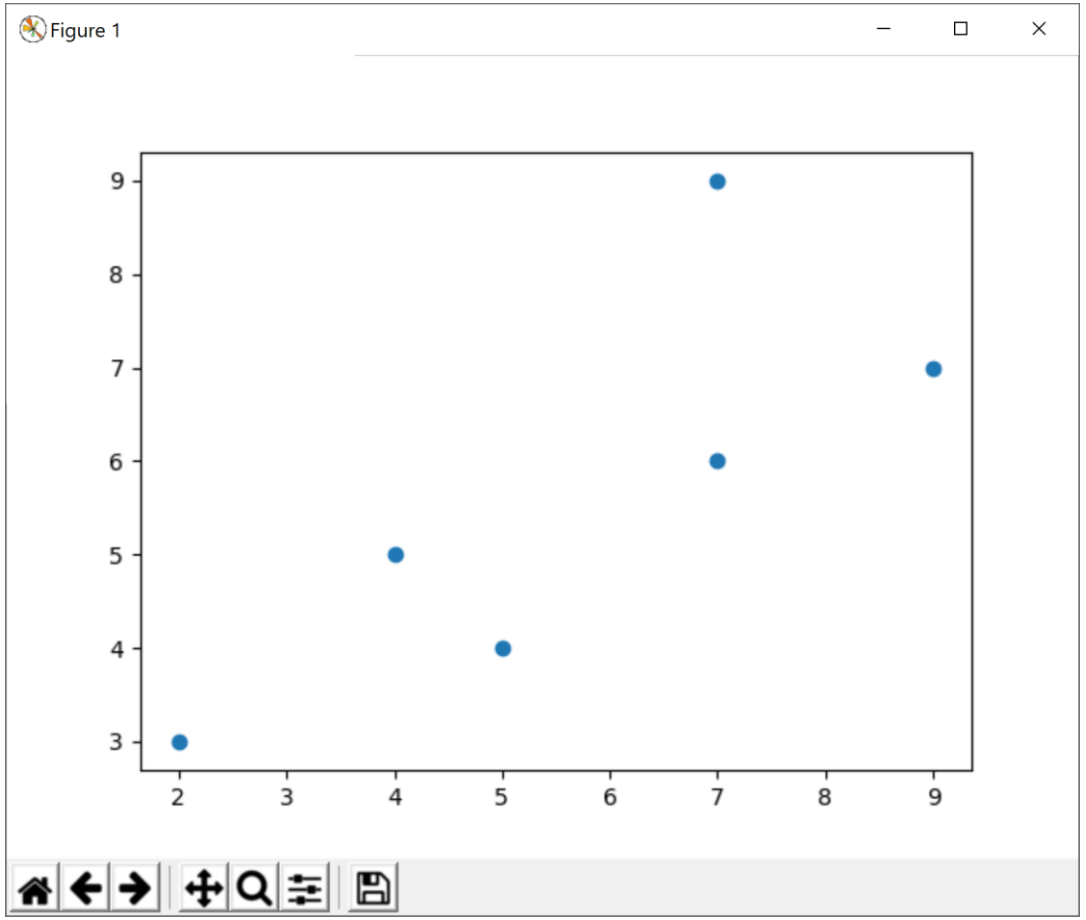
```
import matplotlib.pyplot as plt

plt.title("Empty")
plt.show()
```



Add Visualizations with Methods

There are lots of built-in methods that let you construct different types of visualizations. For example, to make a scatterplot use `plt.scatter(xValues, yValues)`.

```
x = [2, 4, 5, 7, 7, 9]
y = [3, 5, 4, 6, 9, 7]
plt.scatter(x, y)
plt.show()
```

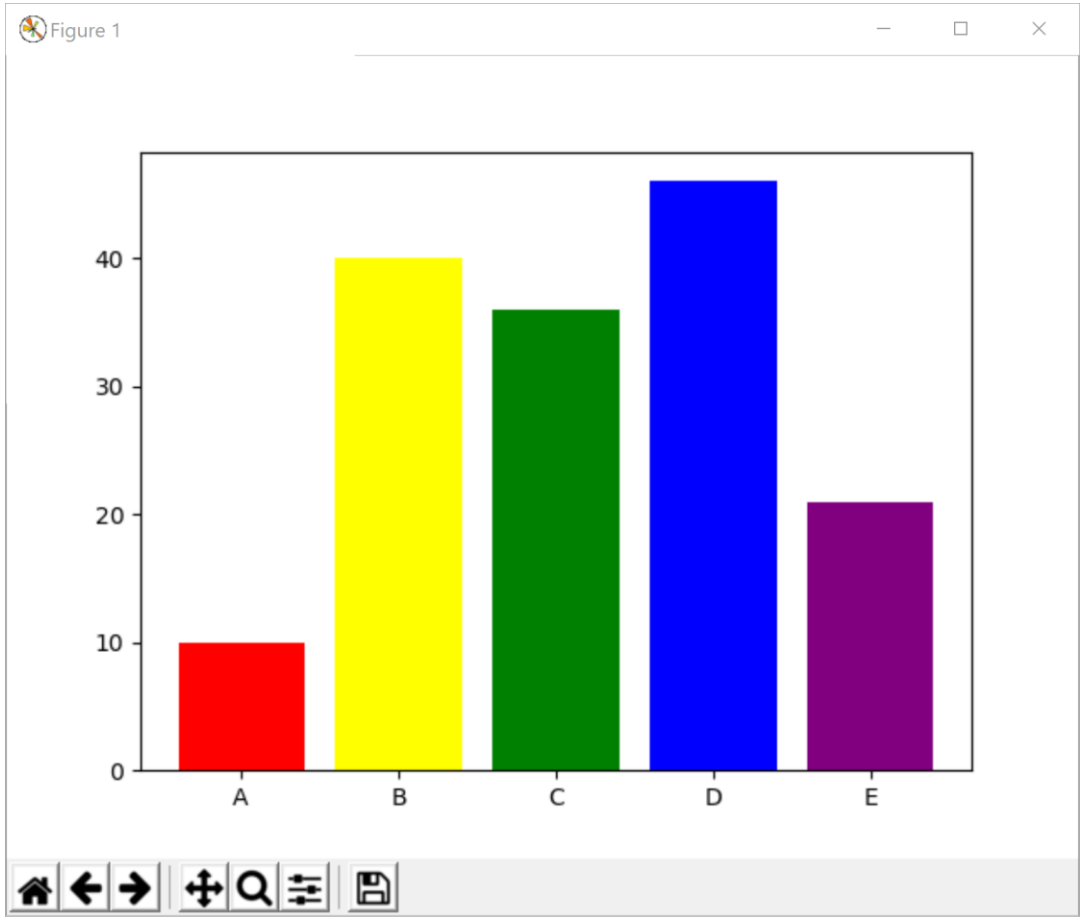


Visualization Methods have Keyword Args

You can **customize** how a visualization looks by adding **keyword arguments**. We used these in Tkinter to optionally change a shape's color or outline; in Matplotlib we can use them to add labels, error bars, and more.

For example, we might want to create a bar chart (with `plt.bar`) with a unique color for each bar. Use the keyword argument `color` to set the colors.

```
labels = [ "A", "B", "C", "D", "E" ]
yValues = [ 10, 40, 36, 46, 21 ]
colors = [ "red", "yellow", "green",
           "blue", "purple" ]
plt.bar(labels, yValues, color=colors)
plt.show()
```



Don't Memorize – Use the Website!

There are a *ton* of visualizations you can draw in Matplotlib, and hundreds of ways to customize them. It isn't productive to try to memorize all of them.

Instead, use the documentation! Matplotlib's website is very well organized and has tons of great examples: https://matplotlib.org/

When you want to create a visualization, start by searching the **API** and the **pre-built examples** to find which methods might do what you need.

API Example

For example – how can we add x-axis and y-axis labels to the bar chart?

Go to the plot API:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html

Search 'label' and you'll soon find the functions `xlabel` and `ylabel`. You can click on the function to find more information. The page describes what the function does, what the required arguments are, and what it returns.

Note that the **keyword arguments** will be listed with default values. That's how we know they're optional.

`matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)` [sou

Set the label for the x-axis.

Parameters:

xlabel : str

The label text.

labelpad : float, default: `rcParams["axes.labelpad"]` (default: `4.0`)

Spacing in points from the axes bounding box including ticks and tick labels. If None, the previous value is left as is.

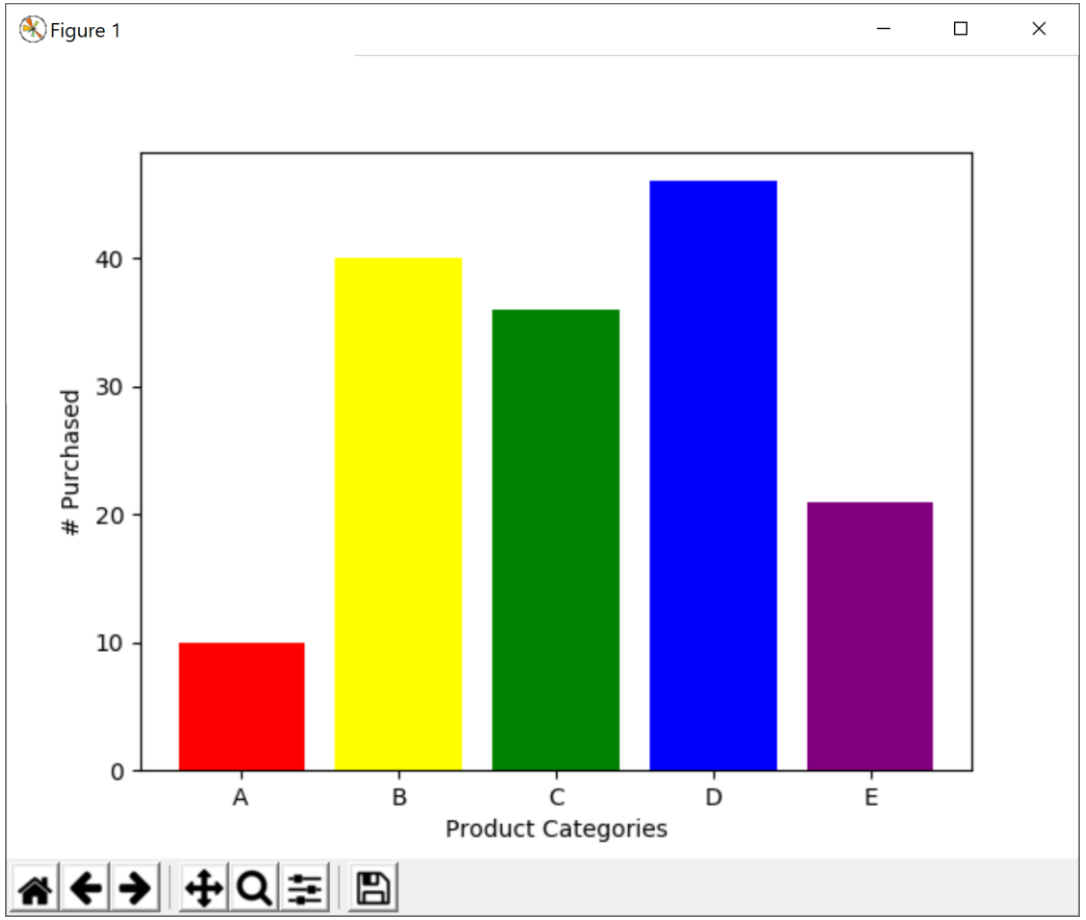
loc : {'left', 'center', 'right'}, default: `rcParams["xaxis.labellocation"]` (default: `'center'`)

The label position. This is a high-level alternative for passing parameters *x* and *horizontalalignment*.

If nothing obvious shows up, you can do a broader internet search of 'matplotlib x-axis label', which will often point you to the right place.

Adding xlabel and ylabel

```
labels = [ "A", "B", "C", "D", "E" ]
yValues = [ 10, 40, 36, 46, 21 ]
colors = [ "red", "yellow", "green",
           "blue", "purple" ]
plt.bar(labels, yValues, color=colors)

plt.xlabel("Product Categories")
plt.ylabel("# Purchased")

plt.show()
```



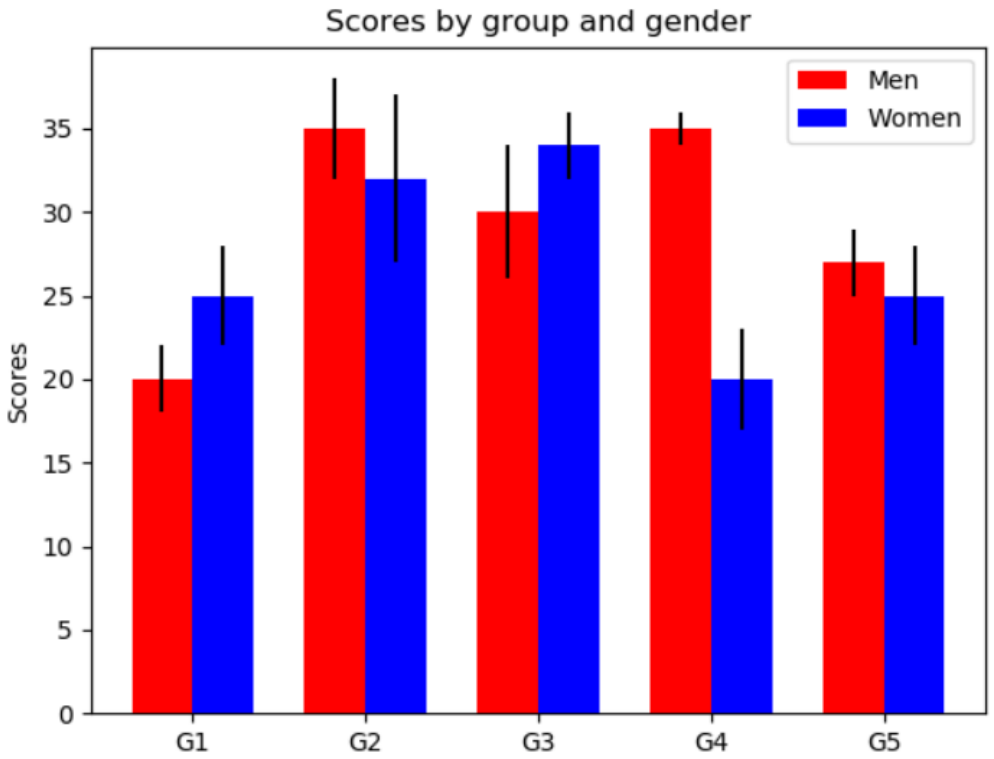
Example Example

Alternatively, you can browse the Matplotlib examples page to find visualizations and features that might prove useful:

https://matplotlib.org/stable/gallery/index.html

Perhaps we're interested in using grouped bar charts to show the breakdown between products purchased in different countries. The example code provides a starting place for which functions to use.

Try copying the example code into your editor and running it. Then try changing some things to see how the results are affected.



Plot vs Axis

You might have noticed that the grouped bar chart example looks slightly different than the code we've written so far. It sets

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

and calls methods on `ax` for the rest of the code.

This is an alternate way to write code in Matplotlib. Instead of drawing on the plot, break the plot into one of more axes with `plt.subplots`, then draw directly on the axis.

This is mainly useful if you want to draw more than one visualization in a single window. For the visualizations we'll do in this class, `plt` will work fine.

Going from Example to Our Own Code

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

labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 34, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]

x = np.arange(len(labels)) # the label locations
width = 0.35 # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, men_means,
                width, label='Men')
rects2 = ax.bar(x + width/2, women_means,
                width, label='Women')

...
```

```
labels = [ "A", "B", "C", "D", "E" ]
yValuesA = [ 10, 40, 36, 46, 21 ]
yValuesB = [ 20, 45, 35, 62, 32 ]

xValuesA = []
xValuesB = []
w = 0.35
for i in range(len(labels)):
    xValuesA.append(i - w / 2)
    xValuesB.append(i + w / 2)

plt.bar(xValuesA, yValuesA, width=w)
plt.bar(xValuesB, yValuesB, width=w)

plt.xlabel("Product Categories")
plt.ylabel("# Purchased")

plt.show()
```

Sidebar: Other Resources

It can also be helpful to search online for other projects that have used the same module, to find examples of how to set it up. Many people have written helpful tutorials for this exact purpose.

Two standard resources for finding help are **StackOverflow**, a site where people can ask questions about code and get answers from other developers, and **GitHub**, a site where people post open-source projects for others to use and contribute to.

IMPORTANT: whenever you copy code from online, make sure to **cite it** the same way you would cite a paragraph of text in an essay. You can do this by putting a comment above the copied code that includes a link to the URL you got the code from.

Example: Visualizing Ice Cream

Let's use Matplotlib to visualize how popular the ice cream flavors were. We're visualizing counts of categorical data, so we can use a **pie chart**. Get the data by calling `getIceCreamCounts` from before.

```
data = readData("all-icecream.csv")
d = getIceCreamCounts(data)
```

Look up how to make a pie chart in the plot API:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html#matplotlib.pyplot.pie

Example: Reformat the Data

A pie chart requires a list `x` that holds the size of each portion. We can optionally provide a list labels with the labels.

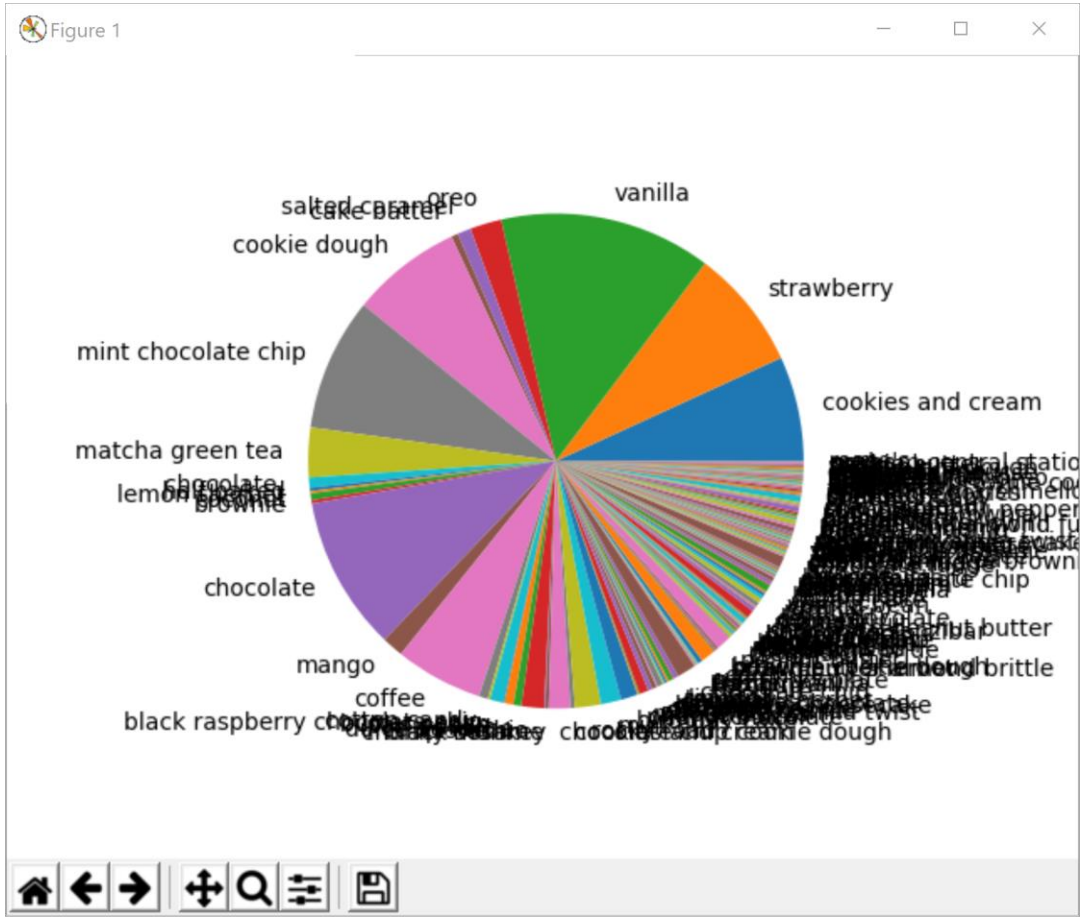
Iterate over the dictionary to break it down into portion values and labels.

```
flavors = []
portions = []
for flavor in d:
    flavors.append(flavor)
    portions.append(d[flavor])
```

Example: Create the Pie Chart

Now we can combine it all together into one pie chart! However, we have a problem...

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

data = readData("all-icecream.csv")
d = getIceCreamCounts(data)

flavors = []
portions = []
for flavor in d:
    flavors.append(flavor)
    portions.append(d[flavor])

plt.pie(portions, labels=flavors)
plt.show()
```



Clean the Data

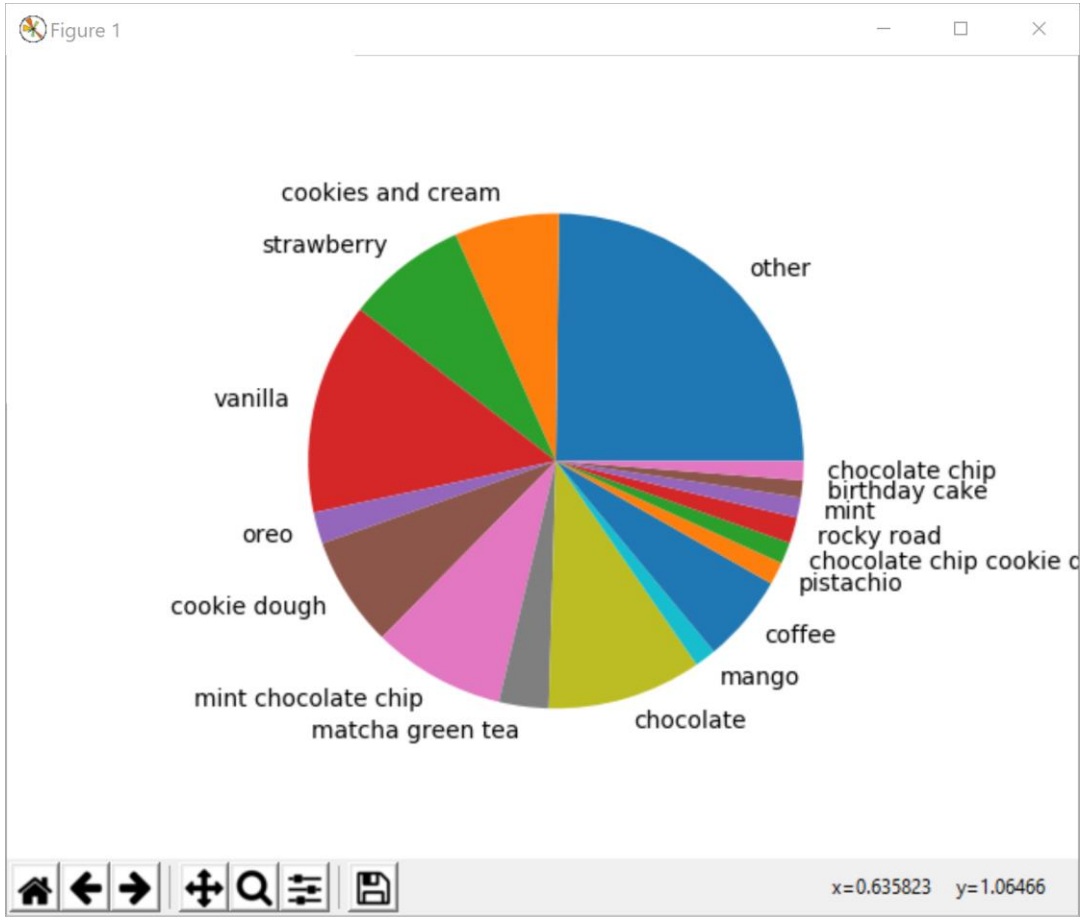
There are too many one-off flavors to make a nice pie chart.

We can handle this by imposing a cutoff on the number of people who liked a flavor. Go back through the data and add anything that doesn't meet the cutoff to an 'other' category.

```
def combineUncommon(d, cutoff):
    newD = { "other" : 0 }
    for flavor in d:
        if d[flavor] >= cutoff:
            newD[flavor] = d[flavor]
        else:
            newD["other"] += d[flavor]
    return newD
```

Improved Pie Chart

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

data = readData("all-icecream.csv")
d = getIceCreamCounts(data)
d = combineUncommon(d, 15)

flavors = []
portions = []
for flavor in d:
    flavors.append(flavor)
    portions.append(d[flavor])

plt.pie(portions, labels=flavors)

plt.show()
```



Learning Goals

- Perform **basic analyses** on data, including calculating **statistics** and **probabilities**, to answer simple questions
- Choose an appropriate **visualization** to create based on the number of **dimensions** and **data types**
- Create simple **matplotlib visualizations** that show the state of a dataset using **APIs** and **examples**
- **Feedback: http://bit.ly/110-s21-feedback**