Data Analysis – Analyzing and Visualizing

15-110 – Monday 11/20

Announcements

- Check6-1 was due today
  - How did it go?

- Check5/Hw5 revisions due **Tuesday at noon**

- No classes / office hours during Thanksgiving Break
  - Wed 11/22 – Sun 11/26

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

Last Time

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

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

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. There's a ton of possible flavors, so we also **hand coded** the data to group similar flavors together and reduce the number of options to analyze. This results in 6 possible categories: chocolate, coffee/tea, cookie, fruit, vanilla, and other.

Let's use this data in 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

Individual statistics methods have been implemented for us, but we may still need to **rearrange the data** to achieve the format that we need.

The data is text, so we must turn it into numbers before performing analyses. Try counting the number of times a person lists a specific flavor category as their favorite and putting those counts into a list to analyze.

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

```python
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 categorized flavors
        flavorCategories = line[firstCol:firstCol+3]
        count = flavorCategories.count(flavor)
        counts.append(count)
    return counts

import statistics
d = readData("all-icecream.csv")
print(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 the probability that an ice cream category is someone's #2 favorite assuming their #1 favorite is a second flavor, create a sublist of the #2 favorites that **only contains** entries where the #1 favorite matches the given flavor.

Then use the `count` method and the sublist length to find the probability!

```
# Probability that flavor A is #2 if flavor B is #1
def getCondProb(data, flavorA, flavorB):
    flavorSubset = []
    firstCol = data[0].index("#1 category")
    for i in range(1, len(data)):
        entry = data[i]
        if entry[firstCol] == flavorB:
            flavorSubset.append(entry[firstCol+1])
    count = flavorSubset.count(flavorA)
    return count / len(flavorSubset)
```

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.

You should generally be able to derive an algorithm that matches the analysis you want to perform.

Visualization

Exploration vs. Presentation

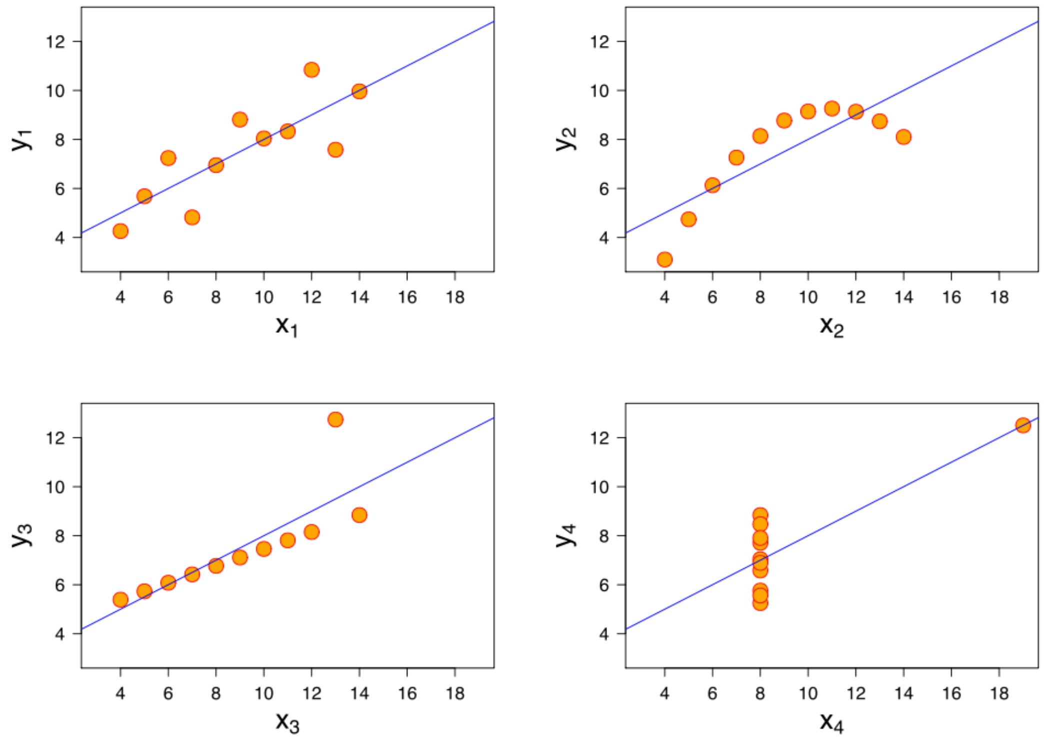
Data Visualization is the process of taking a set of data and representing it in a visual format. Whenever you've made charts or graphs in past math or science classes, you've visualized data!

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.

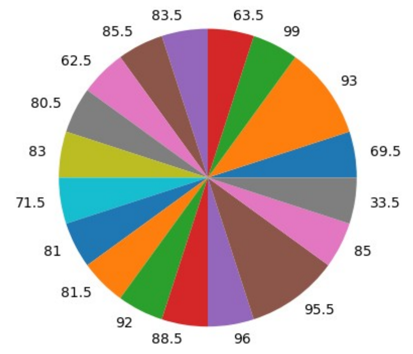
For example, the four graphs to the right all have the same mean and the same best-fit linear regression. But they tell very different stories.

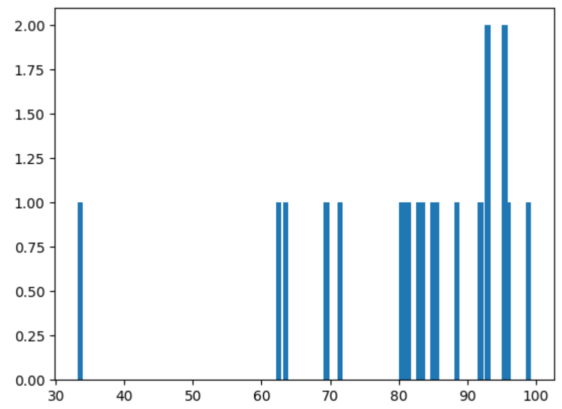


Data Presentation

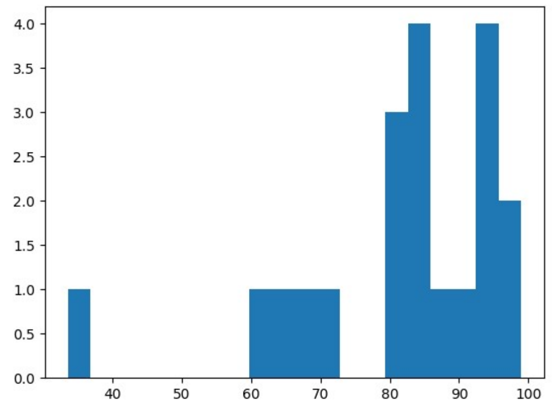
In data presentation, you've already found an interesting pattern in the data and you need to make that pattern easily visible to other people. Choosing the wrong format can lead to misinterpretations.

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.





These three graphs all represent the same data, but some may tell a clearer story than others.



One-Dimensional Data

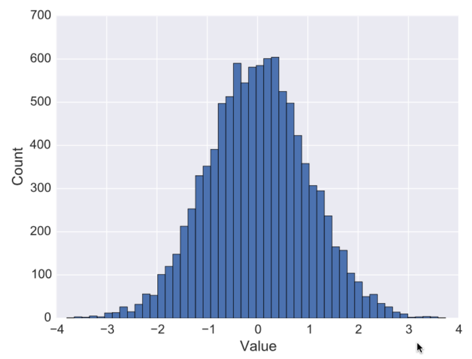
A **one-dimensional visualization** only visualizes a single feature of the dataset. For example:

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

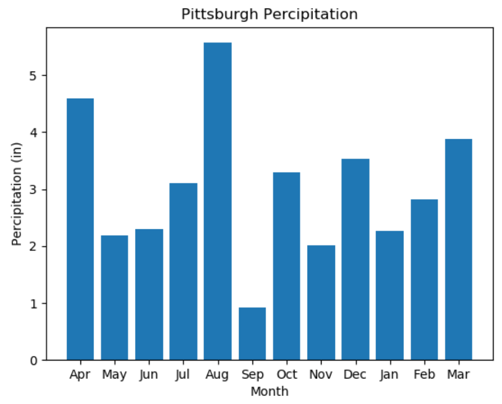
"I want to know the proportion of **people who have cats** in my data"

Charts for One-Dimensional Data

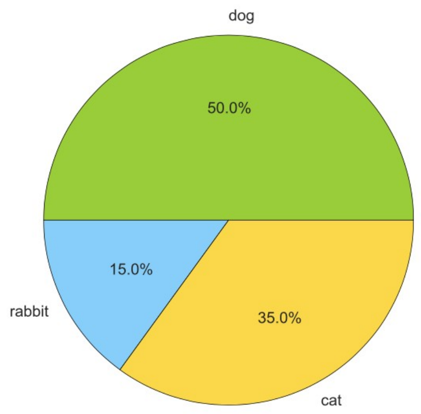
To visualize **numerical** data, use a **histogram**.



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:

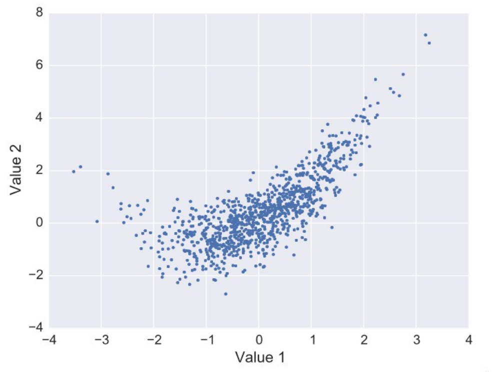
"I want to know the **cost** of each **product category** that we have"

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

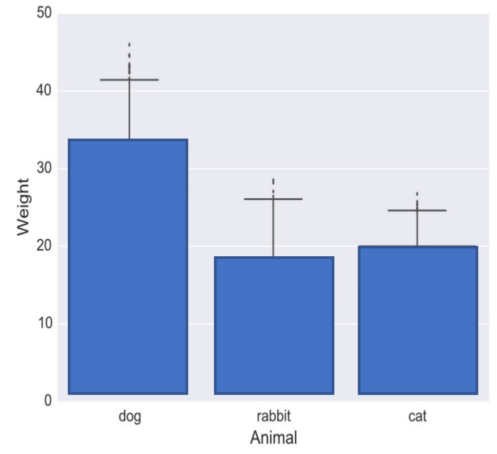
"I want to know how the **size** of the product affects **the cost of shipping**"

Charts for Two-Dimensional Data

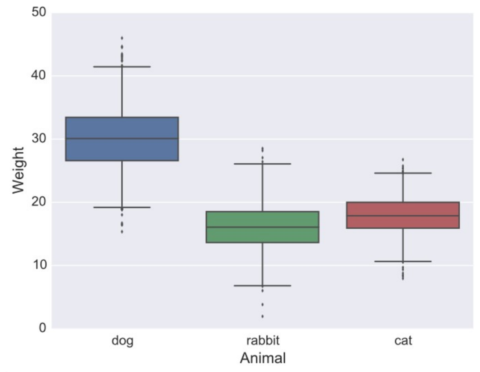
To analyze **numerical x numerical** data, use a **scatter plot**.



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:

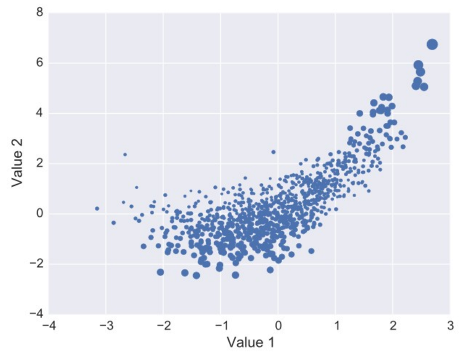
"I want to know the **cost** and the **development time** by **product category**"

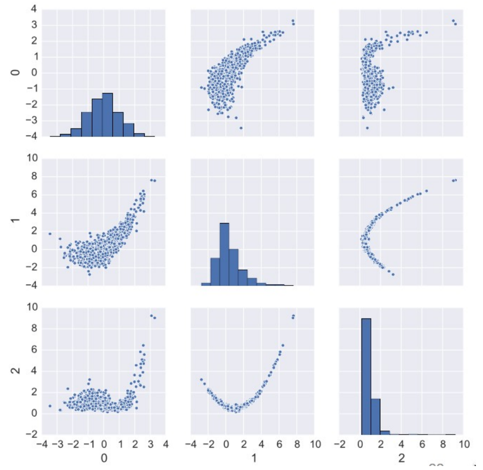
"I want to know the **weight** of the animals that people own and how much they **cost**, by **pet species**"

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

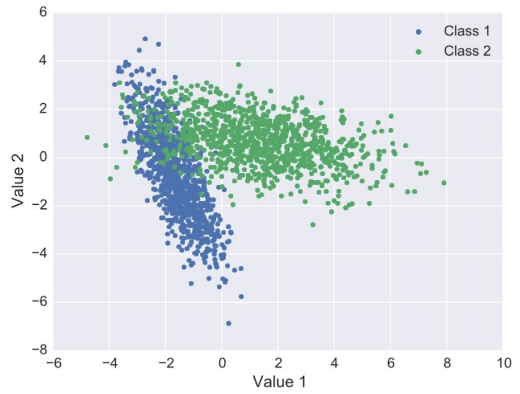
Charts for Three-Dimensional Data

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





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



Reference Tables

One-Dimensional	
Numerical	Histogram
Ordinal	Bar Chart
Categorical	Pie Chart

Two-Dimensional	Numerical	Ordinal/Categorical
Numerical	Scatter Plot	Bar Chart or Box-and-Whiskers Plot
Ordinal/Categorical	Bar Chart or Box-and-Whiskers Plot	Table

Three-Dimensional	Numerical	Ordinal/Categorical
Numerical	x Numerical: Bubble Plot or Scatter Plot Matrix x Ordinal/Categorical: Colored Scatter Plot	x Numerical: Colored Scatter Plot
Ordinal/Categorical	x Numerical: Colored Scatter Plot	Table

Activity: Pick a Visualization

You do: for each of the problem prompts, a) determine the number of dimensions, b) determine the data type, then c) pick the **best** visualization to use based on the dimensions and 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 (72, 94, etc) 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. We'll use it both to create visualizations and to learn how to learn about new libraries.

Matplotlib is **external** – you need to install it on your machine to run it. You can use Thonny's 'Manage Packages' feature to do this, or use the `pip` command:

```
pip install matplotlib
```

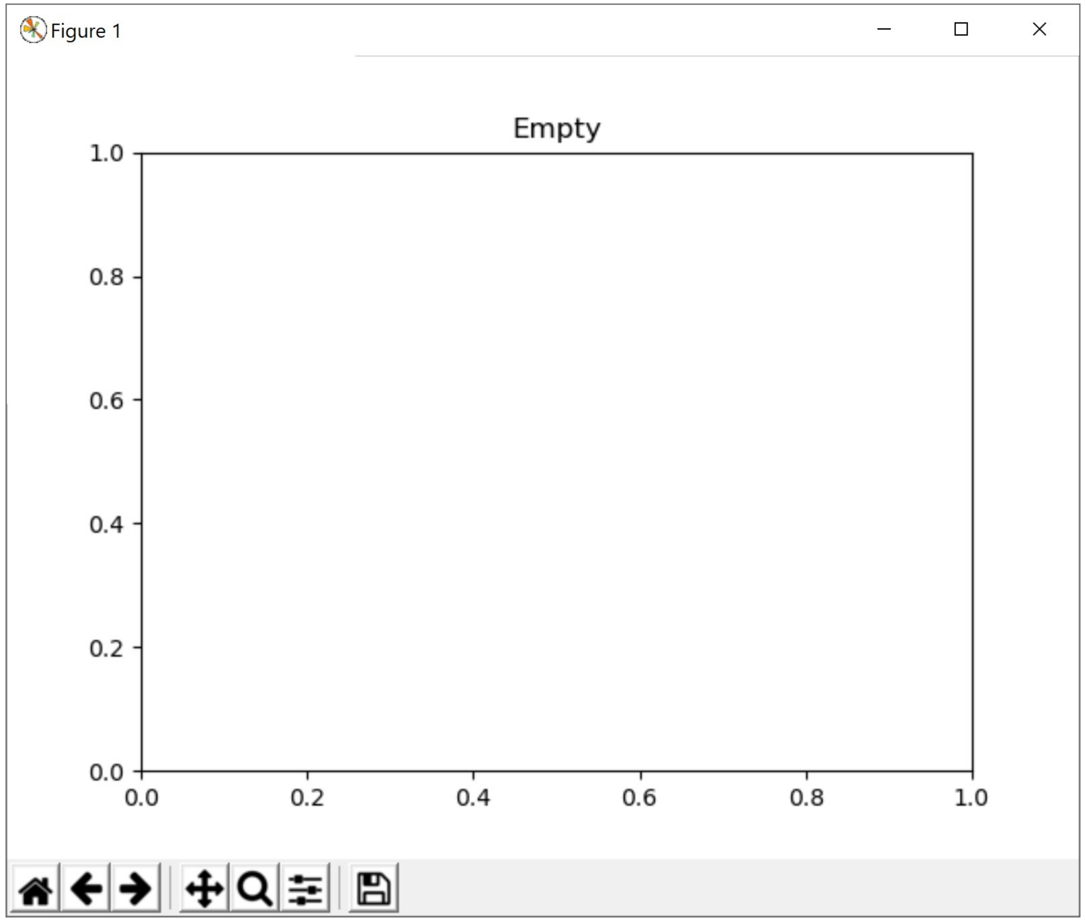
We're going to jump right into Matplotlib, but if you needed to learn without this guidance, you'd likely start with the user guide, which breaks down how the library works. Many libraries have getting-started guides like this!

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.

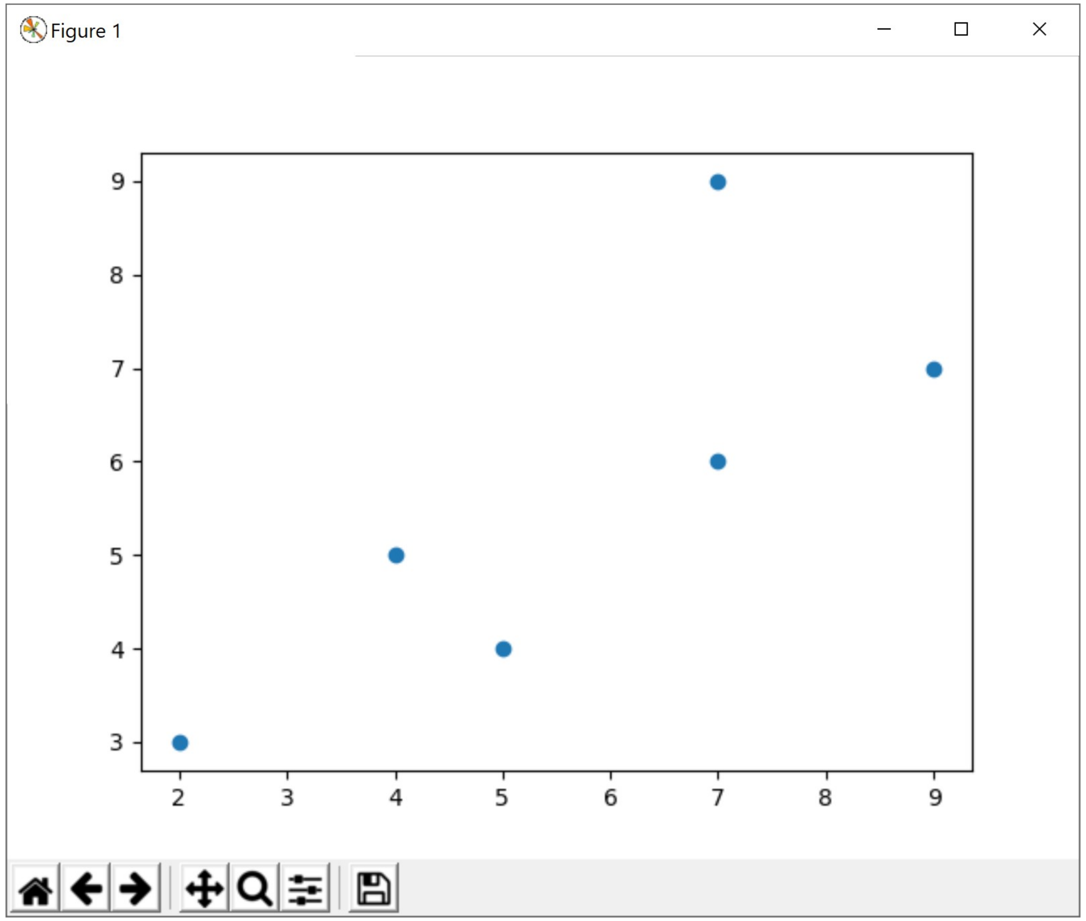
```
import matplotlib.pyplot as plt

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()
```

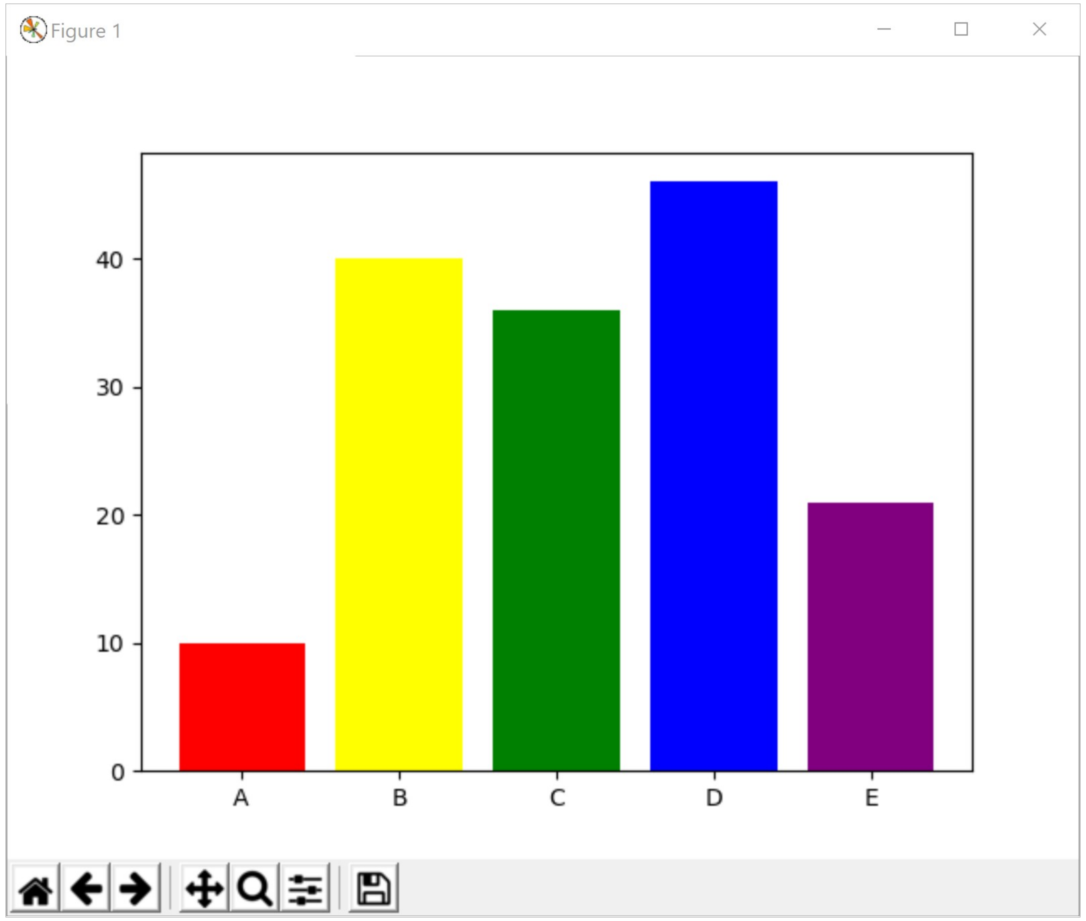


Visualization Methods have Keyword Args

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.

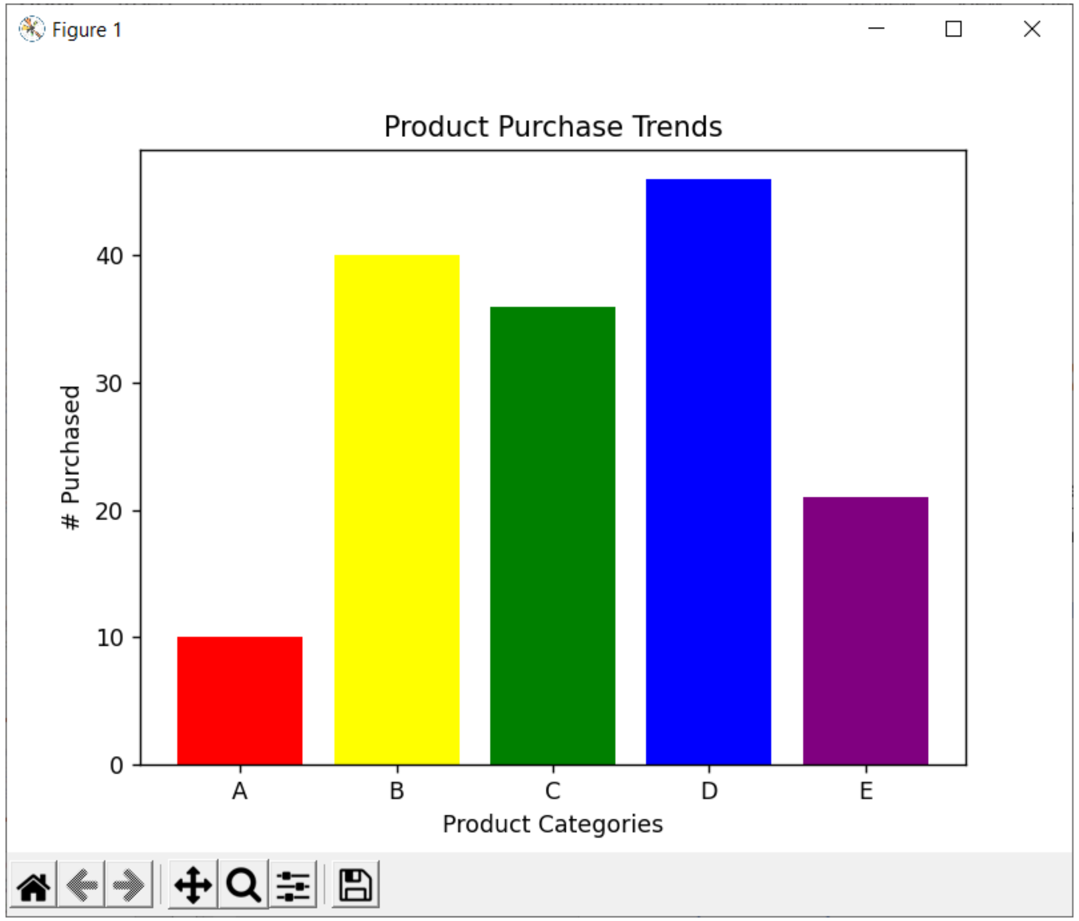
```
labels = [ "A", "B", "C", "D", "E" ]
yValues = [ 10, 40, 36, 46, 21 ]
colors = [ "red", "yellow", "green",
           "blue", "purple" ]
plt.bar(labels, yValues, color=colors)
plt.show()
```



Further Customize with Plot Methods

We can also customize the plot itself by calling more methods that add additional information to the graph! For example, we can add labels to a chart.

```
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.title("Product Purchase Trends")

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/

You can use the strategies we discussed in the Libraries and Documentation lecture to explore this library as needed.

Sidebar: Plot vs Axis

When browsing the documentation, you may notice some new approaches to creating graphs that use

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

and call 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 or 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.

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**.

Start by looking up how to make a pie chart in the plot API:

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

Example: Reformat the Data

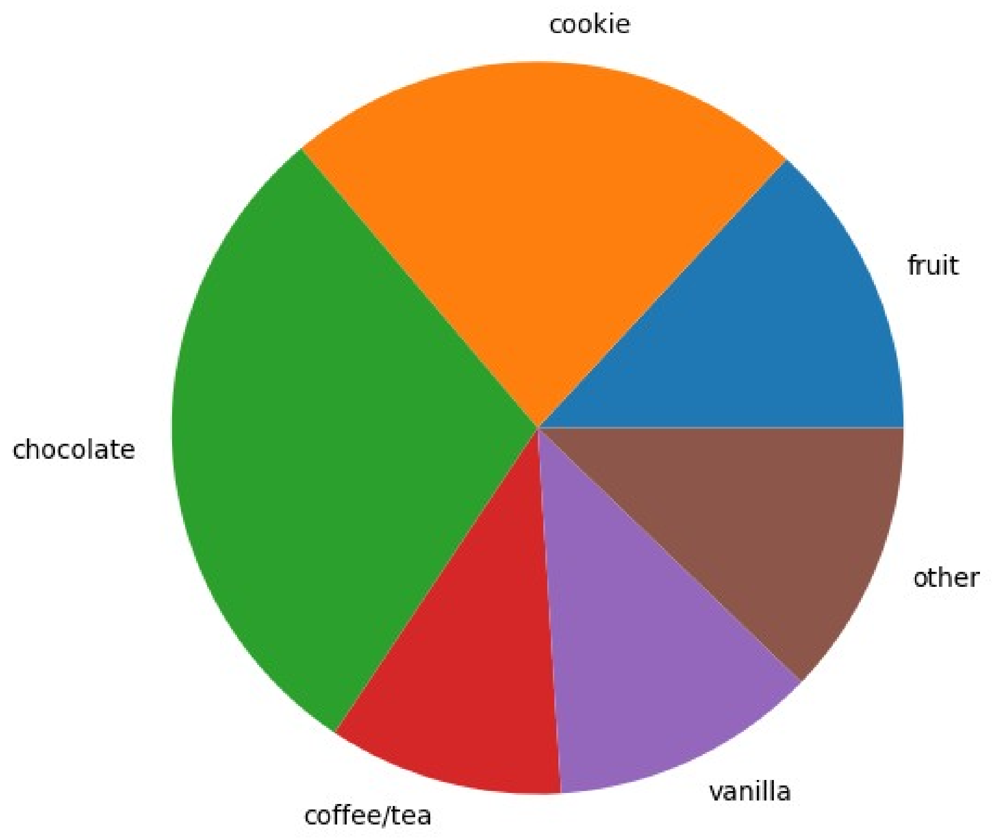
A pie chart requires a list `x` that holds the size of each portion. We can optionally provide a list labels with the labels. Let's find all the flavor categories in the dataset and create a list of all the #1 categories so we can find their counts easily.

```
data = readData("all-icecream.csv")
firstCol = data[0].index("#1 category")
numberOneData = []
flavors = []
for i in range(1, len(data)):
    flavor = data[i][firstCol]
    numberOneData.append(flavor)
    if flavor not in flavors: # haven't seen this one yet
        flavors.append(flavor)

counts = []
for flavor in flavors:
    counts.append(numberOneData.count(flavor)) # get each flavor's count
```

Example: Create the Pie Chart

Now we can combine it all together into one pie chart!

```
import matplotlib.pyplot as plt

plt.pie(counts, 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