Data Analysis – Analyzing and Visualizing

15-110 – Wednesday 4/15

Learning Goals

- Perform basic **analyses** on data to answer simple questions
- Identify which **visualization** is appropriate based on the **type of data**
- Use **matplotlib** to create visualizations that show the state of a dataset

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.

Analysis

Basic Data Analyses – Mean/Median/Mode

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.

Mean: `sum(lst) / len(lst)`

Median: `sorted(lst)[len(lst) // 2]`

Mode: use `mostCommonValue` algorithm with a dictionary mapping values to counts.

Bucketing Data

If you want to break numerical data into categories, you may use **buckets** to group close numerical data together.

For example, if we have a list of grade data, we might bucket them based on the 10s digit.

```
def gradesToBuckets(grades):
    buckets = { }
    for digit in range(10):
        buckets[digit] = 0
    for grade in grades:
        tens = grade // 10
        if tens == 10:
            buckets[9] += 1
        else:
            buckets[tens] += 1
    return buckets
```

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) are slightly harder. But if you create a modified version of the list that contains only those elements with that factor, you can use the same equation.

Calculating Joint Probabilities

What if we want to determine how often two features (likely across multiple columns in the data) occur together in the same data point?

This is a **joint probability**. It requires slightly more complicated code to compute the result.

```
count = 0
for i in range(len(data)):
    if meetsCondition1(data[i]) and meetsCondition2(data[i]):
        count += 1
print(count / len(data))
```

Messy Data – Duplicates

You'll also sometimes need to clean up messy data to get a proper analysis. Some of this is done in the data cleaning stage, but even cleaned data can have problems.

One potential issue is **duplicate data**, when the same data entry is included in the dataset multiple times. To detect duplicate data, check if your data has a unique ID per data point; then you can count how often each ID occurs to find the duplicates.

```
for id in dataIds:
    if dataIds.count(id) > 1:
        print("Duplicate:", id)
```

Messy Data – Missing Values

Analyses can also run into problems when there are **missing values** in some data entries. Data can be missing if some entries were not collected, and is likely to occur in surveys with optional questions.

```
for i in range(len(data)):
    if data[i] == "": # can also check for 'n/a' or 'none'
        print("Missing row:", i)
```

To deal with missing data, ask yourself: **how crucial is that data**? If it's an important part of the analysis, all entries that are missing the needed data point should be removed, and the final report should include how much data was thrown out.

If it's less important, you can substitute in a 'N/A' class for categorical data, or skip the entry for numerical data. But be careful about how missing data affects the analysis.

Messy Data – Outliers

Finally, be careful about how **outliers** can affect the results of data analysis.

Outliers are data points that are extremely different from the rest of the dataset. For example, in a dataset of daily time reports, most people might report 5-15 hours, but one person might report 100 hours.

The easiest way to detect outliers is to use **visualizations**, which we'll discuss later in the lecture. Outliers should be removed from some calculations (especially means) to avoid skewing the results. Be careful, some outlier-like data may not actually be an outlier and may reveal important information.

Other Common Statistics

Python has already implemented some of these statistics for you!

`statistics` library: https://docs.python.org/3/library/statistics.html

This library can compute mean/median/mode, but also variance, standard deviation, and more.

Example: Analyzing Ice Cream Data

We've now cleaned the ice cream dataset from last week. Let's analyze the data to answer this question: **which ice cream flavors do people like most**?

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

```
import csv
def readData(filename):
    f = open(filename, "r")
    reader = csv.reader(f)
    data = [ ]
    for row in reader:
        data.append(row)
    return data
```

Example: Total Preferences

First: how many times does each flavor occur in any of a person's preferences?

```
def getIceCreamCounts(data):
    iceCreamDict = { }
    for i in range(1, len(data)): # skip header
        for flavor in data[i]:
            if flavor not in iceCreamDict:
                iceCreamDict[flavor] = 0
            iceCreamDict[flavor] += 1
    return iceCreamDict
```

Activity: Count Top Flavors

Second: how often does each flavor occur as the **top** preference a person has?

Modify the code from before to count only the top preference (`"Flavor 1"`)

When you're done, submit your modified code here:

https://bit.ly/110-s20-flavors

Visualization

Exploration vs. Presentation

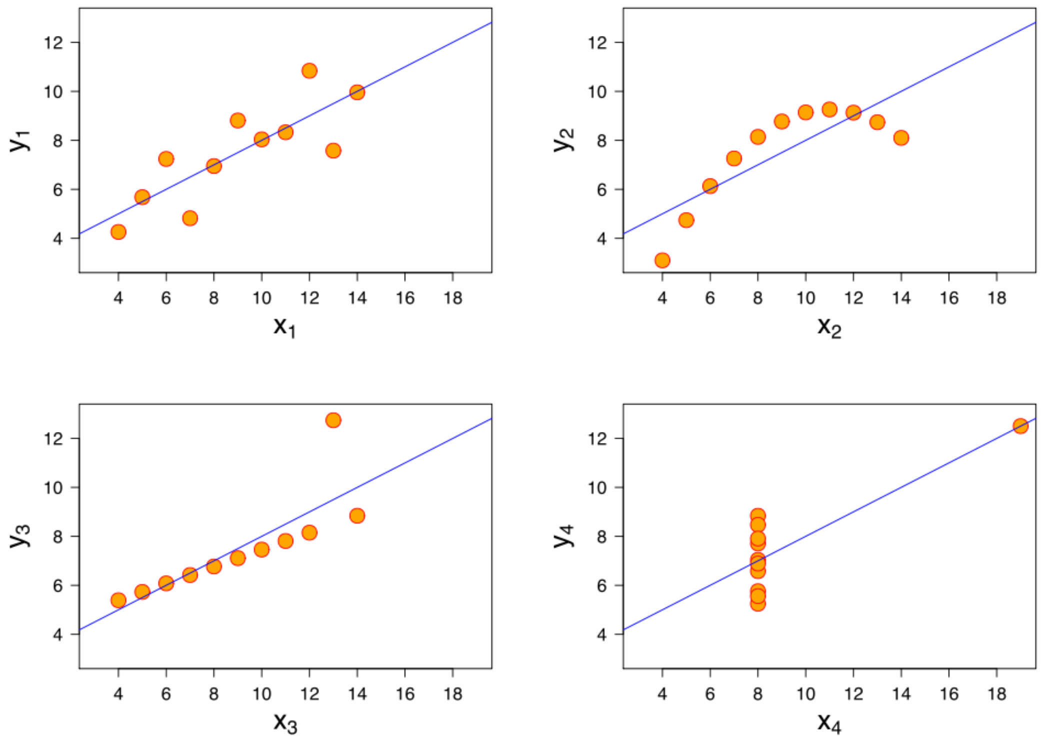
Data Visualization is the process of taking a set of data and representing it in a visual format. Whenever you've made charts or graphs in past math or science classes, you've visualized data!

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.



Visual Variables Show Differences

In visualization, we use different **visual variables** to demonstrate the differences between categories or data points.

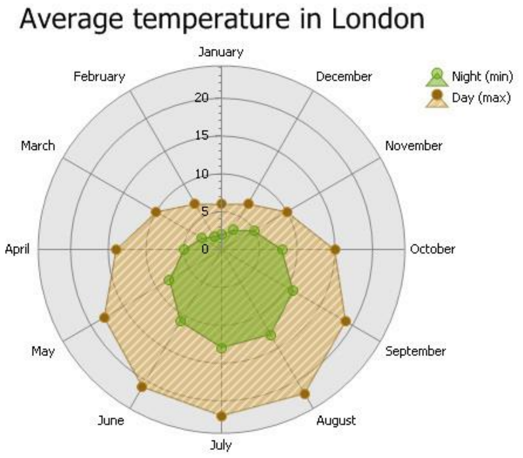
Which visual variable you use depends on the **type** of the data you're representing – categorical, ordinal, or numerical.

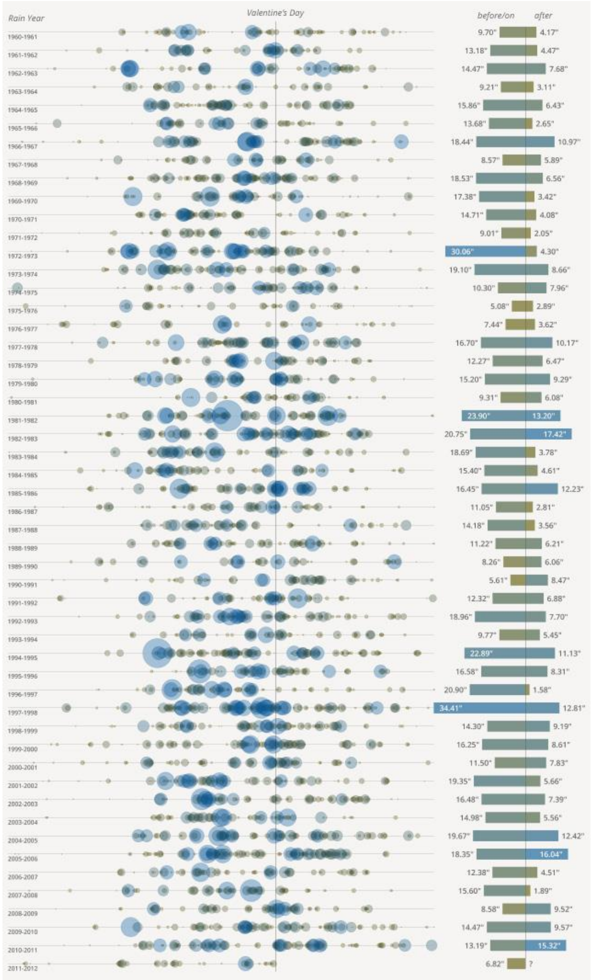
Visual Variable Options – Numerical

If you want to encode **numerical** data, you basically have two options: **position** and **size**.

Position: where something is located in the chart, as in an x,y position. Positions to the upper-right tend to be correlated with larger numbers.

Size: how large a visual element is, or how long it is in the chart. The larger the size, the bigger the number.

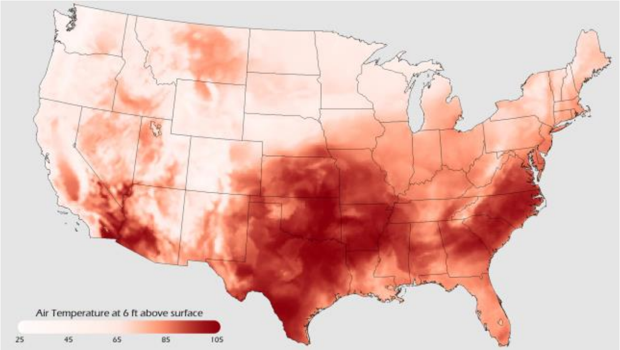




Visual Variable Options – Ordinal

For **ordinal** data, you can use position and size, but you can also use **value**.

Value: the hue of a color in the chart (from 0 RGB to 255 RGB). Hues are ordered based on the ordinal comparison.

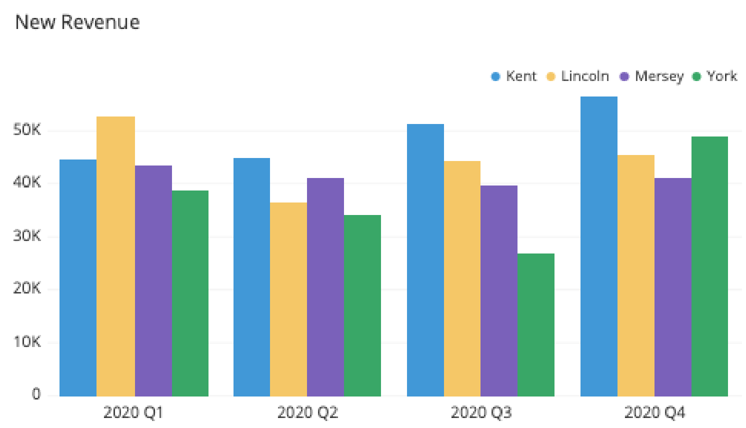


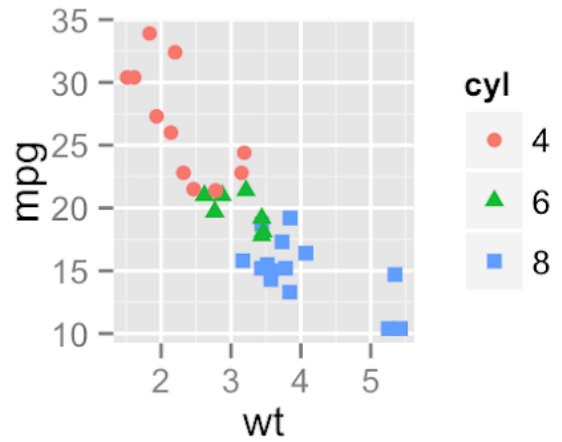
Visual Variable Options – Categorical

Categorical data can be presented using position, size, and value, but it also adds two other options: **color** and **shape**.

Color: each category can be assigned a different color (red for Cat1, blue for Cat2, pink for Cat3).

Shape: each category can be assigned a different shape (square for Cat1, circle for Cat2, triangle for Cat3).





Choosing a Visualization

There are dozens of different visualizations you can use on data.

In order to choose the best visualization for the job, consider how many **dimensions** of data you need to visualize.

We'll go over three options: one-dimensional data, two-dimensional data, and three-dimensional data.

One-Dimensional Data

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

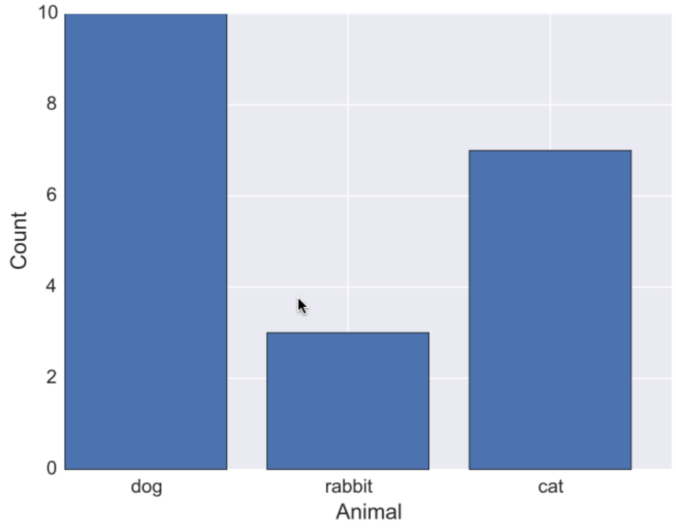
"I want to know how many of each **product type** are in my data"

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

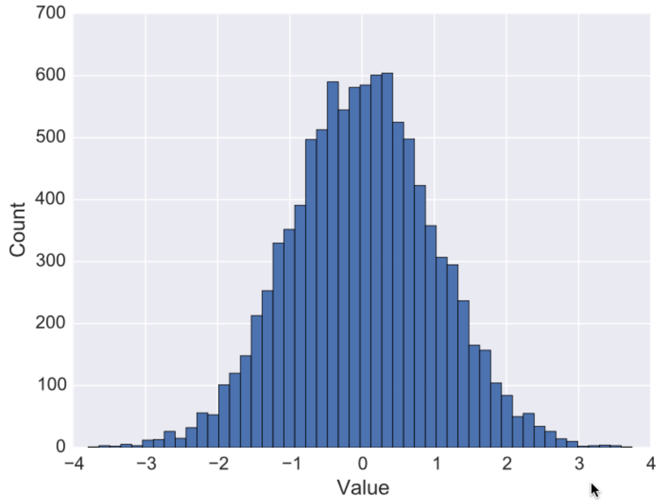
To visualize one-dimensional data, use a **histogram** or a **pie chart**.

Histograms Show Counts

For categorical or ordinal data, show counts for each type of data using bars (length = count).



For numerical data, use **bucketing** across a distribution.

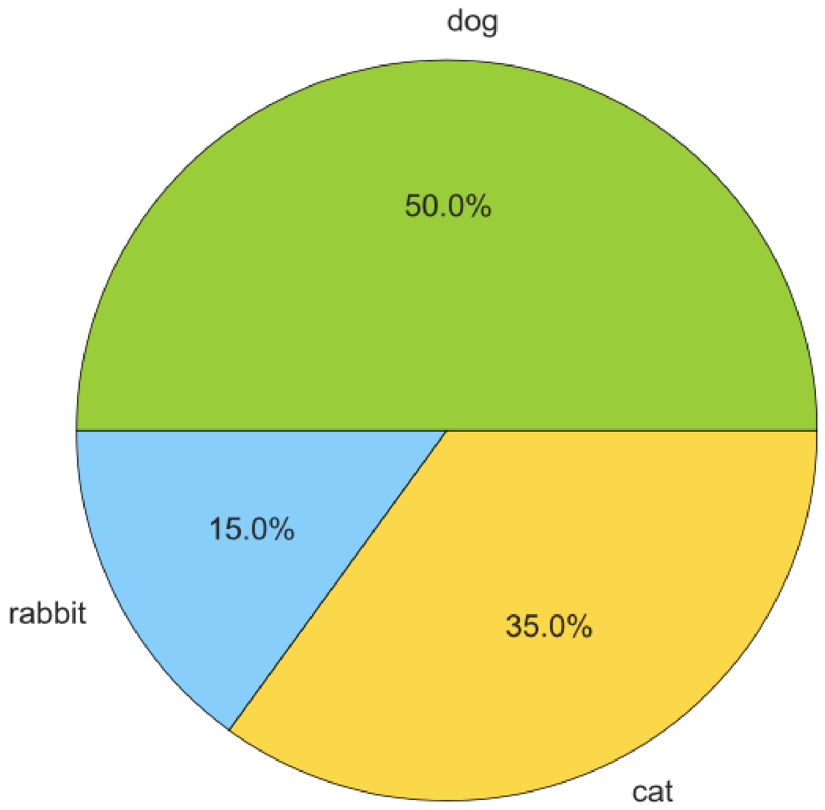


A **histogram** shows you the **shape** and the **spread** of numerical data.

Pie Charts Show Percentages

A **pie chart** shows the proportion of the data that falls into each category of the feature. The proportions should always add up to 100%.

It doesn't make sense to use a pie chart on a numerical feature unless you use **bucketing**.



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**"

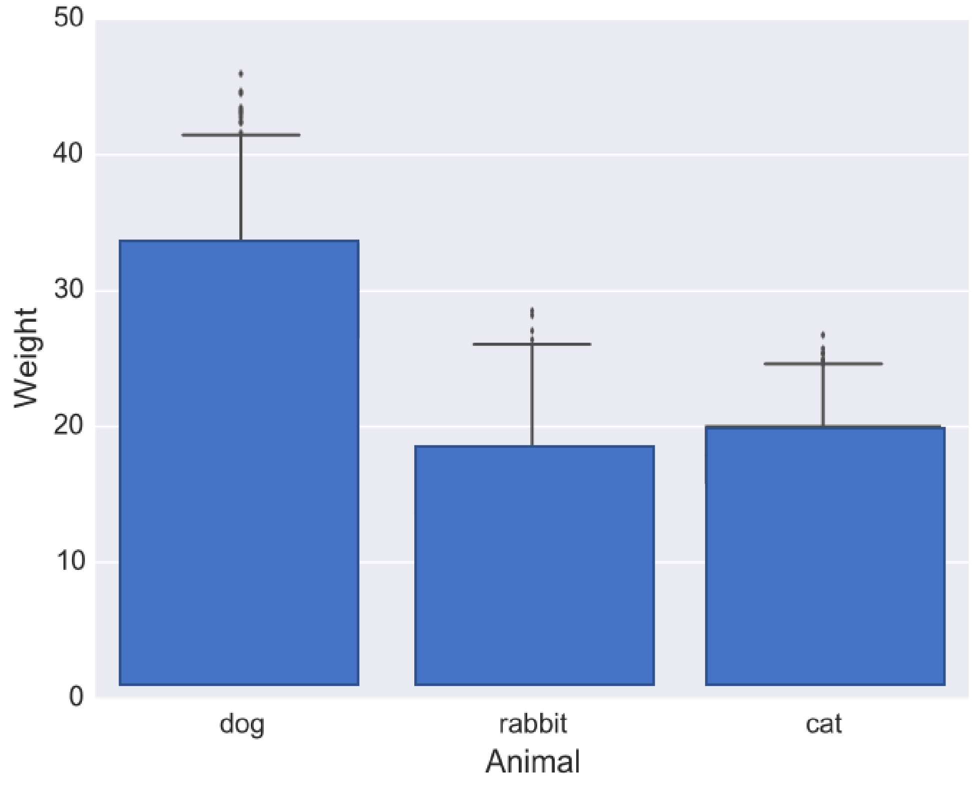
"I want to know how the **size** of the product affects **the cost of shipping**"

To visualize two-dimensional data, use a **bar chart**, a **scatter plot**, a **line plot**, or a **box-and-whiskers plot**.

Bar Charts Compare Averages

A **bar chart** compares the average results of a numerical feature across the categories of a categorical feature.

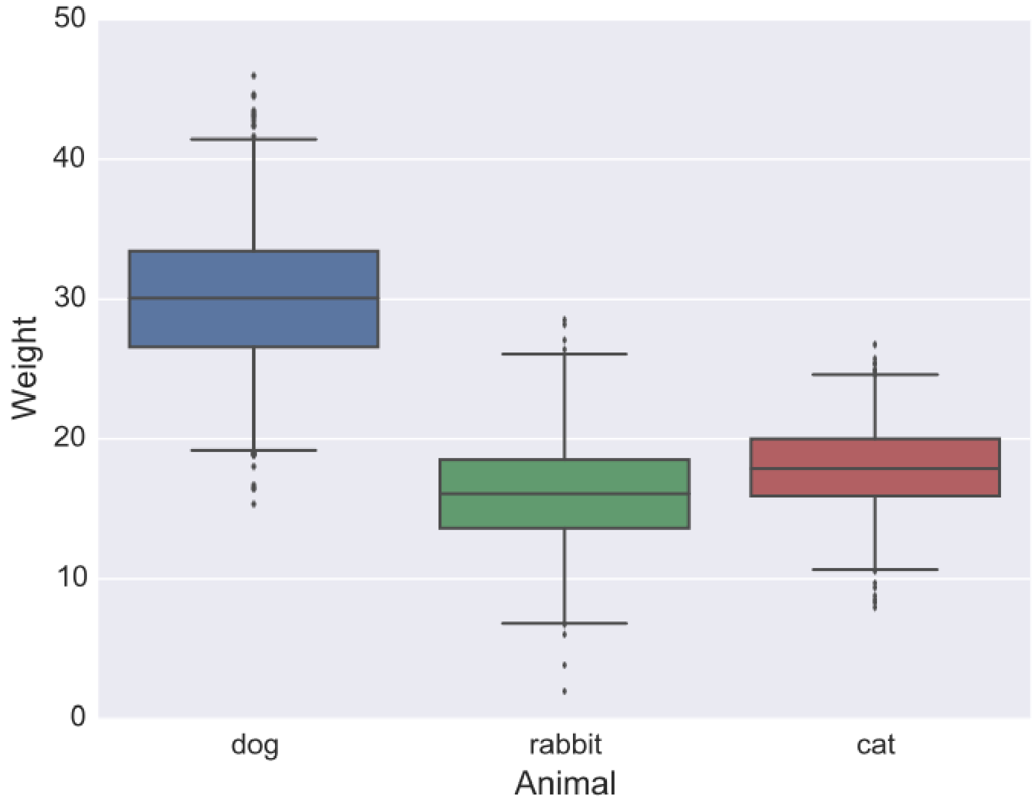
You can add error bars on a bar chart to see if two categories are significantly different.



Box-and-Whisker Plots Show Ranges

A **box-and-whisker plot** also compares averages of a numerical feature across categories of a categorical feature, but it visually provides summary statistics across the **range** of the data.

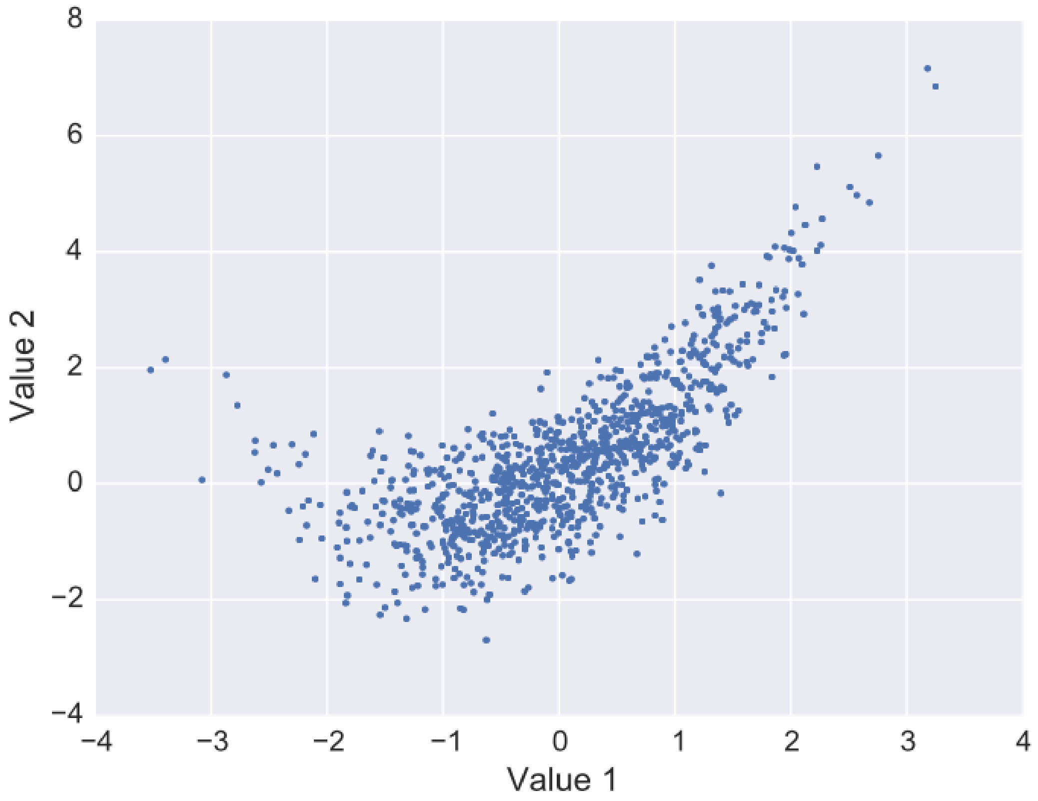
This plot is especially useful for data that is not normally distributed around the average.



Scatter Plots Show Trends

A **scatter plot** compares two numerical features by plotting every data point as a dot on the graph (with the first feature as the x axis and the second as the y axis).

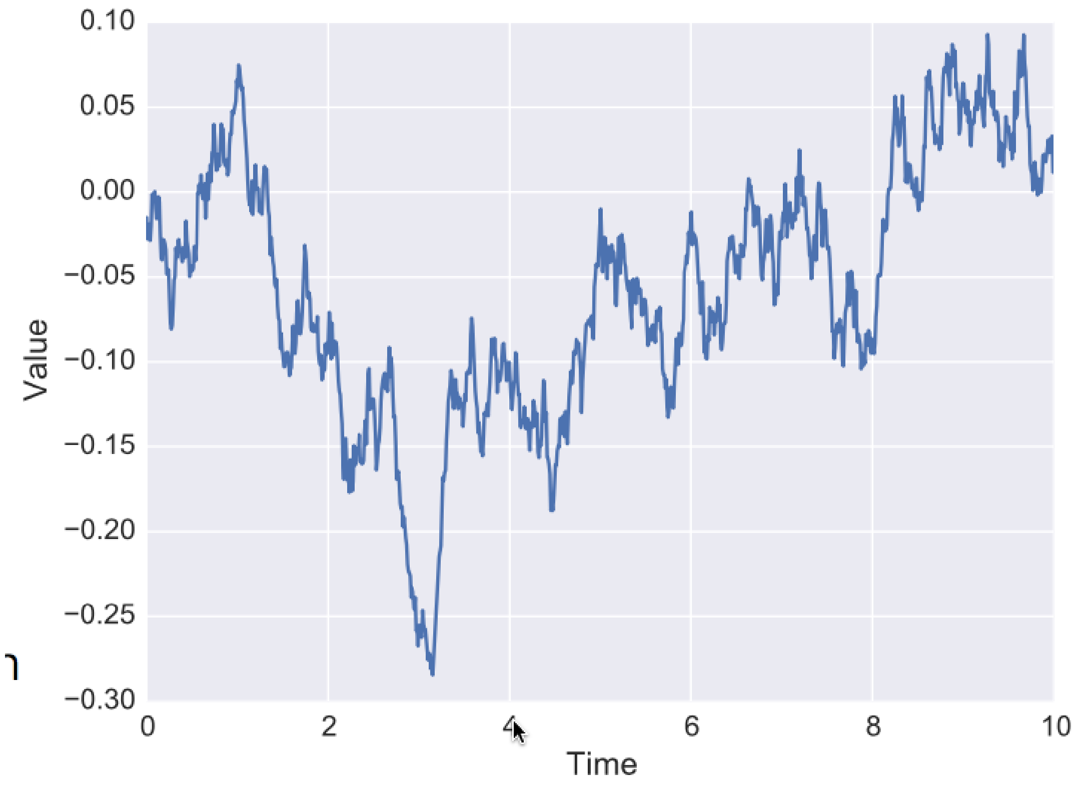
Scatter plots are useful for observing trends in data.



Line Plots Show Trends Over Time

A **line plot** uses a numerical feature that specifically measures **time** on the x axis, and a different numerical feature on the y axis.

Because there's generally one data point per timestamp, the points are connected using lines, to show a trend over time.



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**"

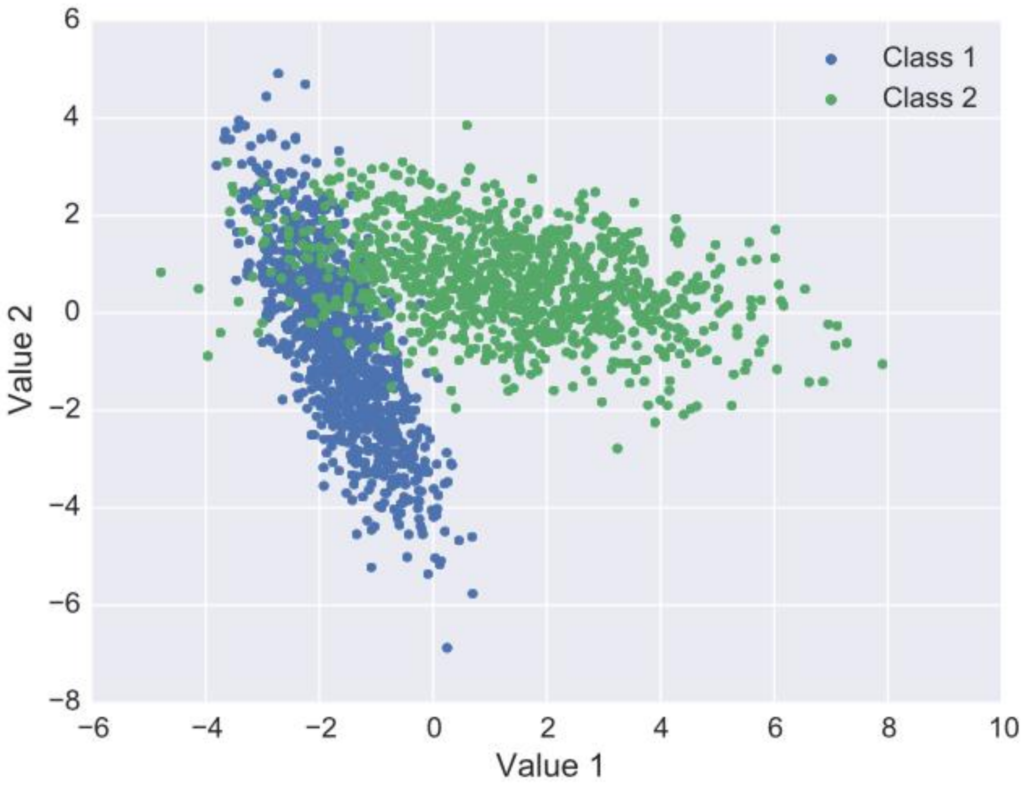
"I want to know how the **size** of the product and the **manufacturing location** affects **the cost of shipping**"

To visualize three-dimensional data, use a **colored scatter plot**, a **scatter plot matrix**, or a **bubble plot**.

Colored Scatter Plots Compare Trends

A **colored scatter plot** lets you compare two numerical features across a set of categories in a categorical feature.

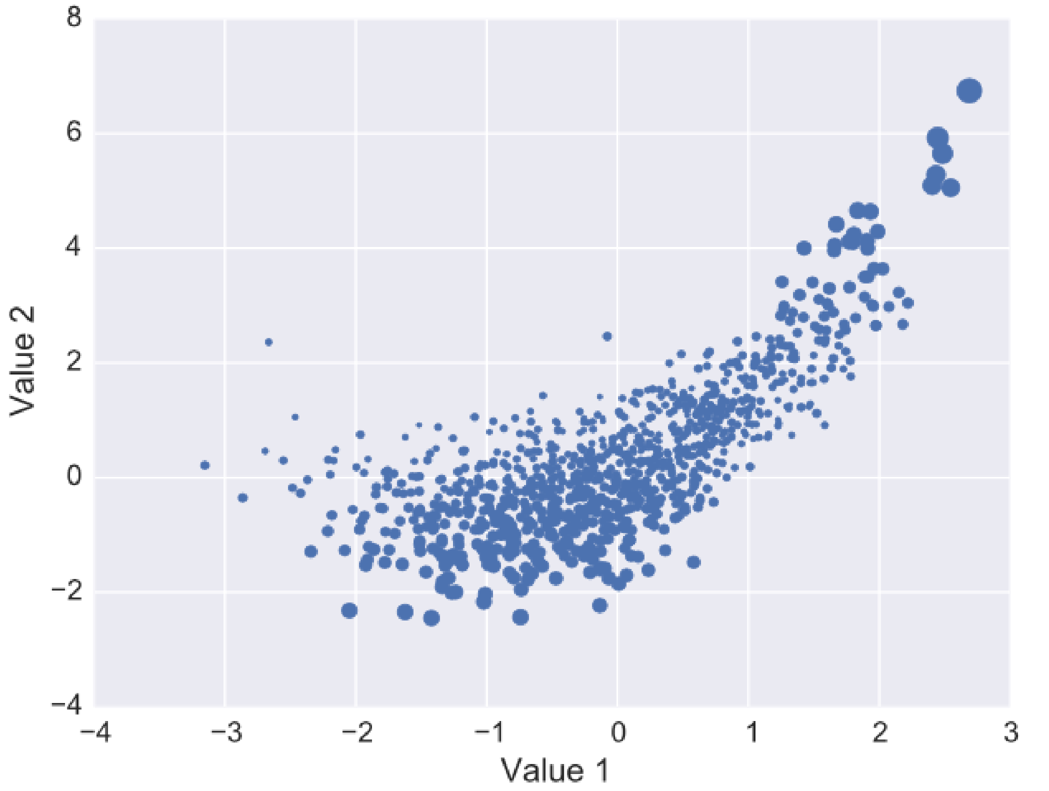
Each category has its values plotted in a scatter plot, and each category gets a different **color**. This plot makes it easy to tell which categories have different trends.



Bubble Plots Show Size

A **bubble plot** can be used to compare three numerical features. One feature is the x axis, and another is the y axis. The third feature is used to specify the **size** of the data points.

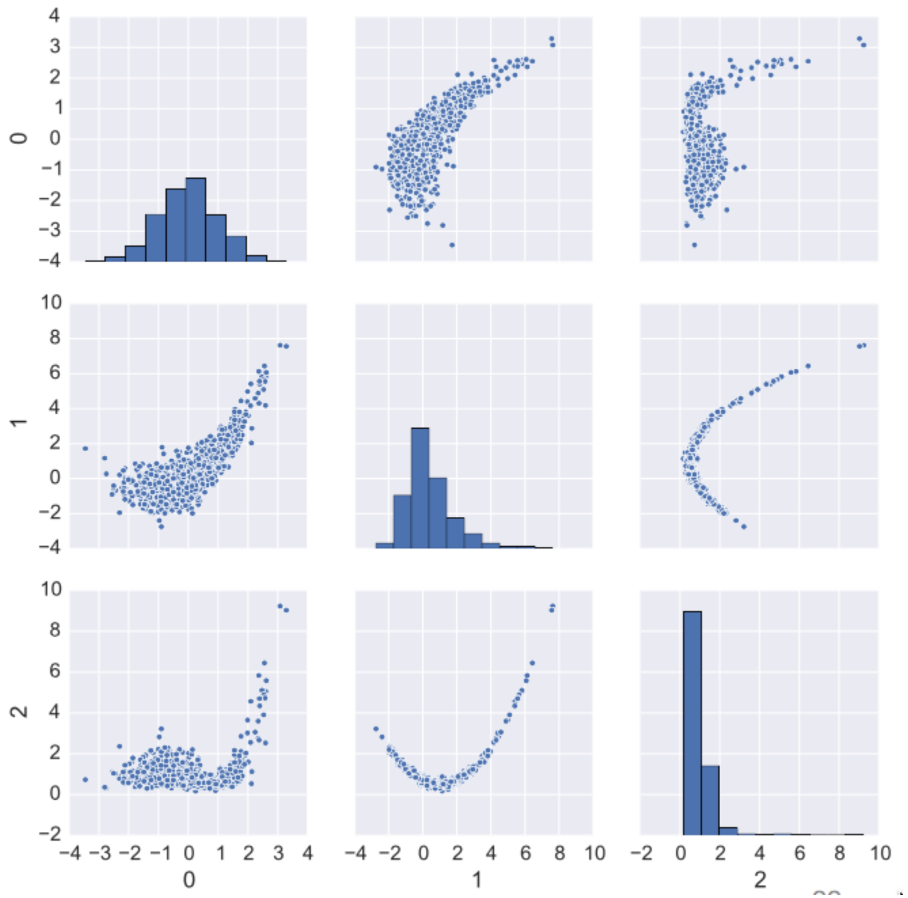
Bubble plots can get confusing when there's a lot of data points, but are useful on sparse data.



Scatter Plot Matrixes Compare Multiple Features

A **scatter plot matrix** can be used to compare three (or more) numerical features. Each **column** corresponds to one of the tree features, and each **row** corresponds to one of the three features. The graph shown in each position is then the scatter plot between the row's feature and the column's feature.

Note that graphs on the diagonal are histograms, as they compare a feature to itself.



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. We'll talk more about installing modules next week.

The matplotlib library has excellent documentation on how to make different graphs here: https://matplotlib.org/ . We'll go over the basics now.

Matplotlib Core Ideas

For every visualization you make in Matplotlib, you'll need to set up a **figure** and **axis**. This is generally done with the code:

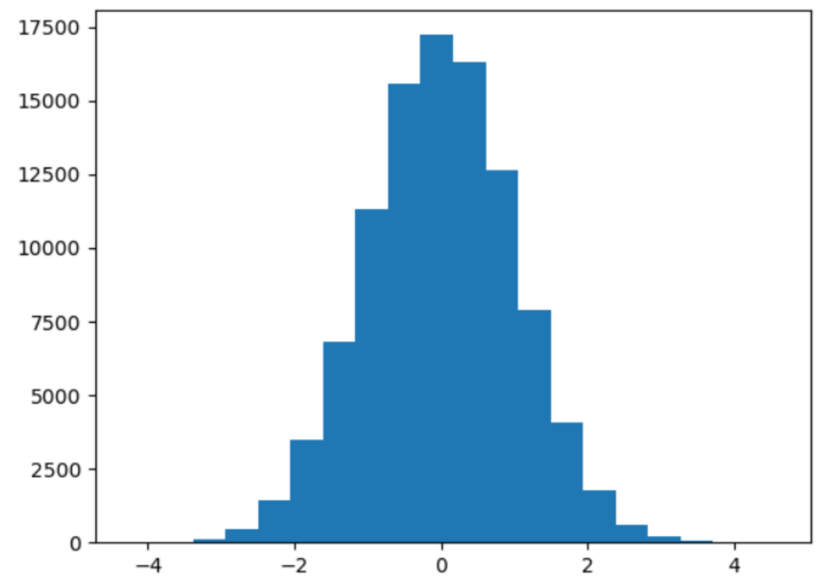
```
fig, ax = plt.subplots()
```

You can then directly add visualizations to the axis by calling different methods on `ax`, with the data as the parameter. Let's look at histograms and bar charts specifically.

Once you're done setting up the visualization, call `plt.show()` to display the chart.

Histogram Example - Numerical

```
import matplotlib.pyplot as plt
import random

# Generate a normal distribution
data = []
for i in range(100000):
    data.append(random.normalvariate(0, 1))

# Set up the plot
fig, ax = plt.subplots()

# Set # of bins with the 'bins' arg
ax.hist(data, bins=20)

plt.show()
```



Bar Chart Example - Categorical

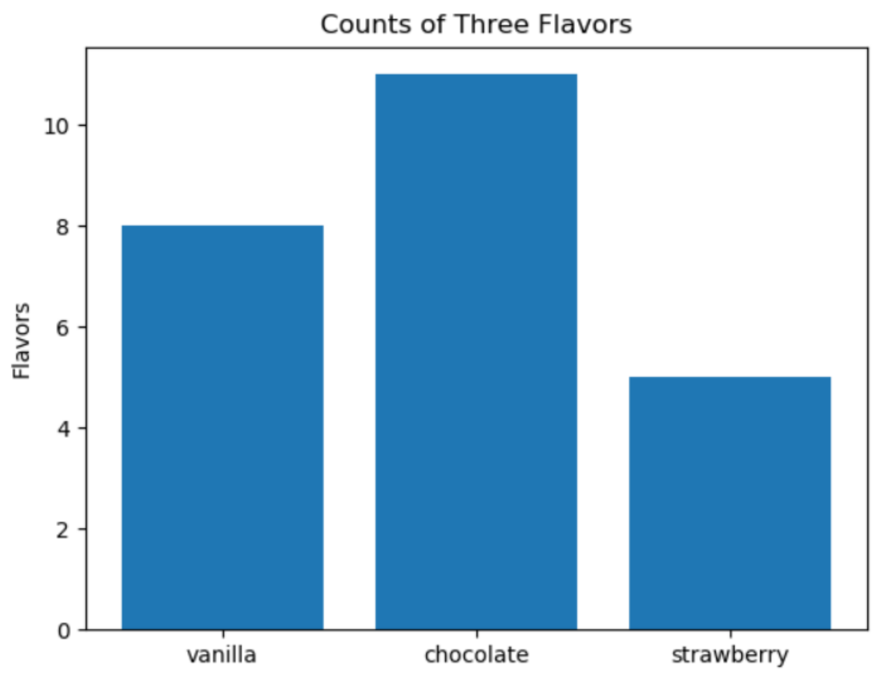
Let's use our ice cream data to make a nice categorical histogram (which will be formed using bar charts). We'll graph the counts of the three classic flavors: vanilla, chocolate, and strawberry.

First, process the data to get those counts:

```
data = readData("icecream.csv")
d = getIceCreamCounts(data)
flavors = [ "vanilla", "chocolate", "strawberry" ]
counts = [ d["vanilla"], d["chocolate"], d["strawberry"] ]
```

Bar Chart Example - Categorical

```
import matplotlib.pyplot as plt

# Set up the plot
fig, ax = plt.subplots()

# Set up the bars
ind = range(len(counts))
rects1 = ax.bar(ind, counts)

# Add labels
ax.set_ylabel('Flavors')
ax.set_title('Counts of Three Flavors')
ax.set_xticks(ind)
ax.set_xticklabels(flavors)

plt.show()
```



Advanced Bar Chart

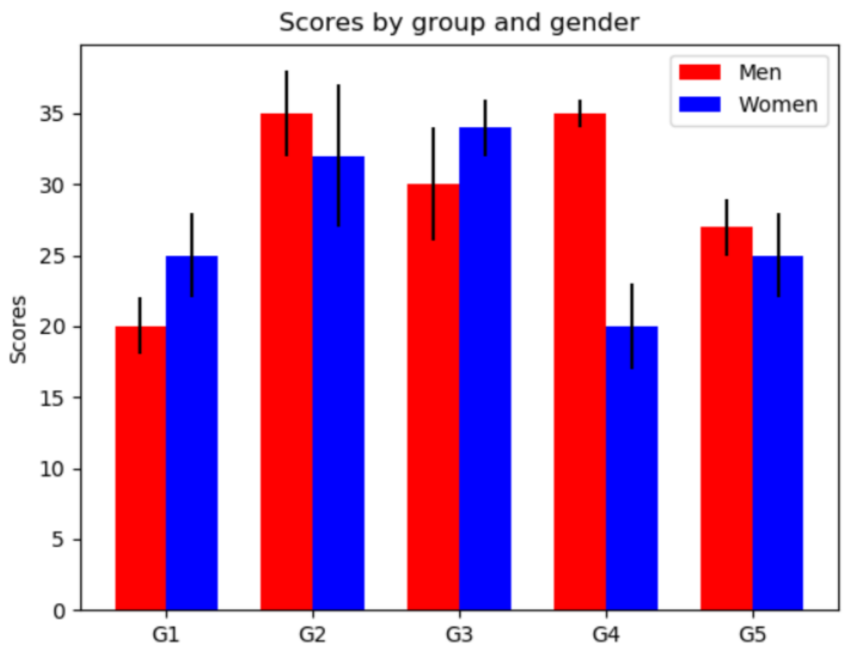
We can make our visualizations more advanced by adding side-by-side bars, and using the other matplotlib features to add data to the chart.

For example, let's write a bit of matplotlib code to compare averages and standard deviations across an arbitrary data set.

```
menMeans = [20, 35, 30, 35, 27]
menStd = [2, 3, 4, 1, 2]
womenMeans = [25, 32, 34, 20, 25]
womenStd = [3, 5, 2, 3, 3]
```

Bar Chart Matplotlib

```
# From matplotlib website
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()

# Using numpy arrays lets us do useful operations
mensInd = np.arange(5)
width = 0.35 # the width of the bars
womensInd = mensInd + width

rects1 = ax.bar(mensInd, menMeans, width,
                color='r', yerr=menStd)
rects2 = ax.bar(womensInd, womenMeans, width,
                color='b', yerr=womenStd)

# Labels and titles
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(mensInd + width / 2)
ax.set_xticklabels(['G1', 'G2', 'G3', 'G4', 'G5'])
ax.legend([rects1[0], rects2[0]], ['Men', 'Women'])

plt.show()
```



Learning Goals

- Perform basic **analyses** on data to answer simple questions
- Identify which **visualization** is appropriate based on the **type of data**
- Use **matplotlib** to create visualizations that show the state of a dataset