Bonus Slide

New (to us) Ice Creams from the Ice Cream Survey

- Caramel Crunch
- Chocolate Coconut
- Dragon Stout
- Fruit Sorbet
- Mango Sorbet
- Salted Licorice









Unit 2: Modern CS Techniques

Data Analysis Libraries

15-110 – Wednesday 10/21

Quizlet5

Learning Goals

- Recognize how different data types - **categorical, ordinal,** and **numerical -** affect how that data can be interpreted and analyzed.

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

- Use **statistics, data management, and visualization libraries** to analyze and present information about a dataset.

Last Time

Last time 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
```

Other methods in the Statistics Library

You can also measure spread in your dataset:

```
statistics.variance(data) # 475.733333
statistics.stdev(data) # 21.811312
```

As well as relationships with data such as correlations or covariance and distributions in the data.

You can find many more methods in the official documentation: https://docs.python.org/3/library/statistics.html

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

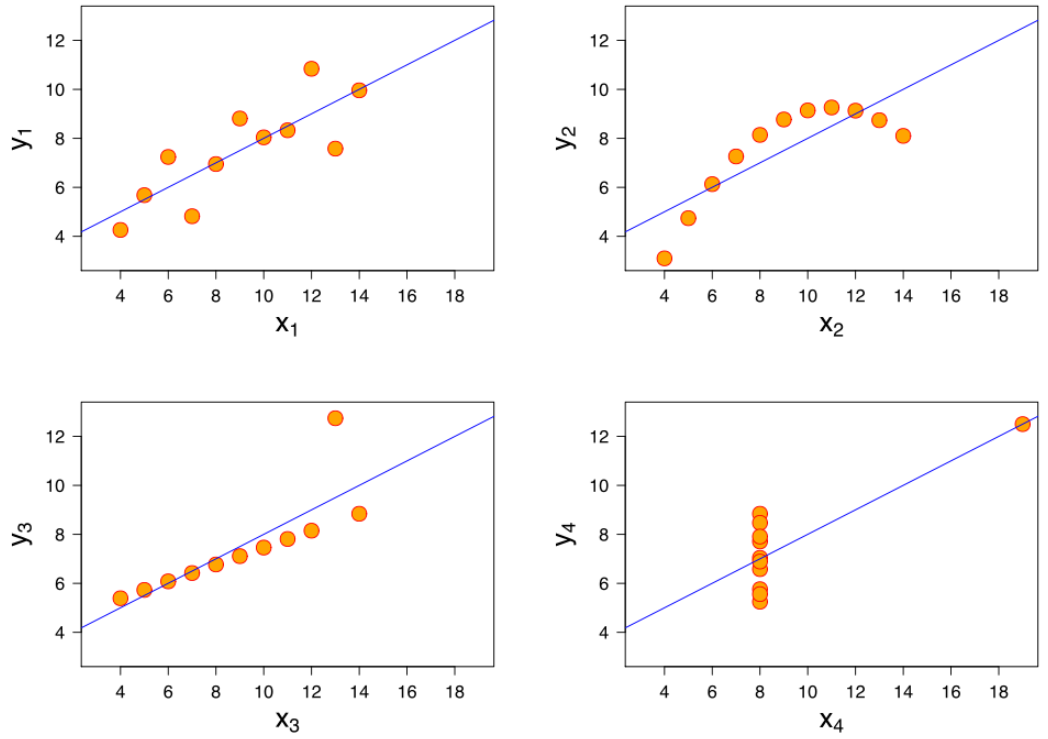
Data Visualization is the process of taking a set of data and representing it in a visual format. Whenever you've made charts or graphs in past math or science classes, you've visualized data!

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.

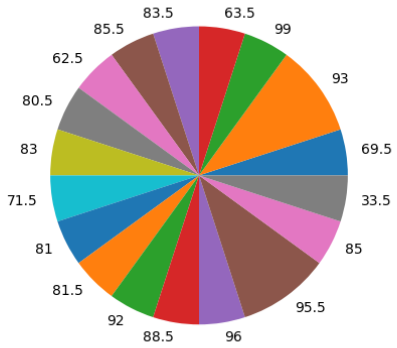
For example, the four graphs to the right all have the same mean and the same best-fit linear regression. But they tell very different stories.

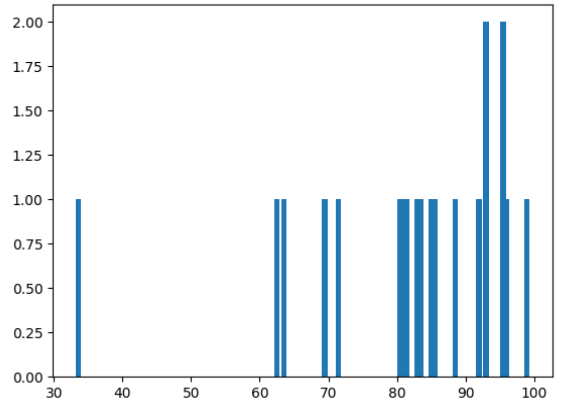


Data Presentation

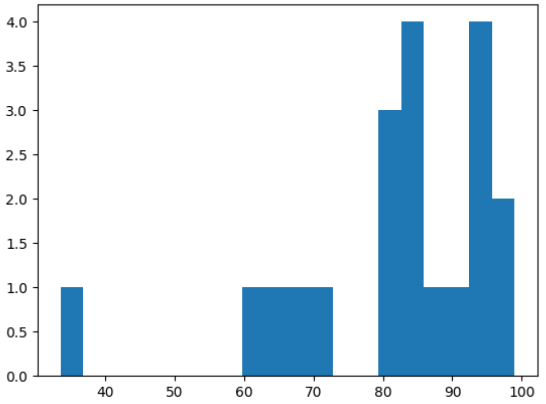
In data presentation, you've already found an interesting pattern in the data and you need to make that pattern easily visible to other people. Choosing the wrong format can lead to misinterpretations.

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

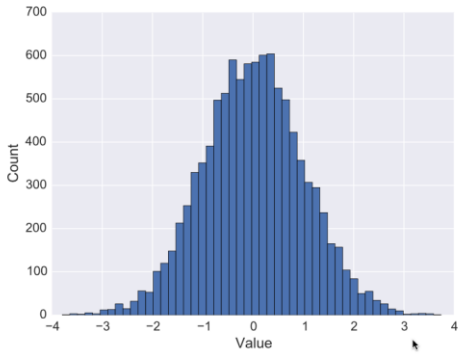
A **one-dimensional visualization** only visualizes a single feature of the dataset. For example:

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

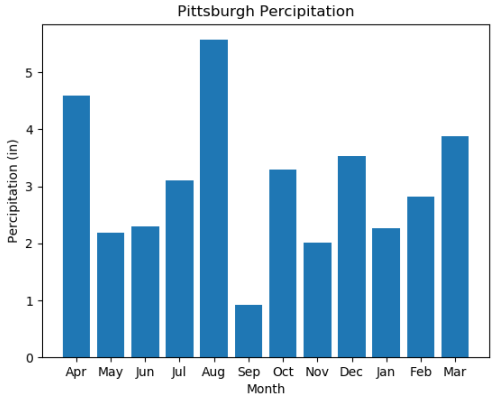
"I want to know the proportion of **people who have cats** in my data"

Charts for One-Dimensional Data

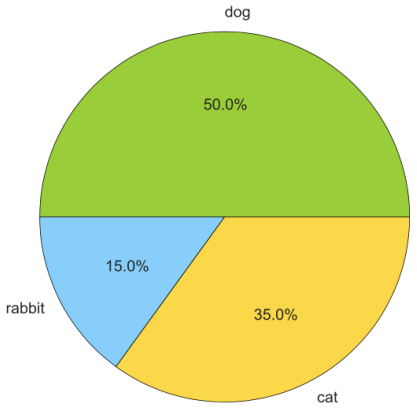
To visualize **numerical** data, use a **histogram**.



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**"

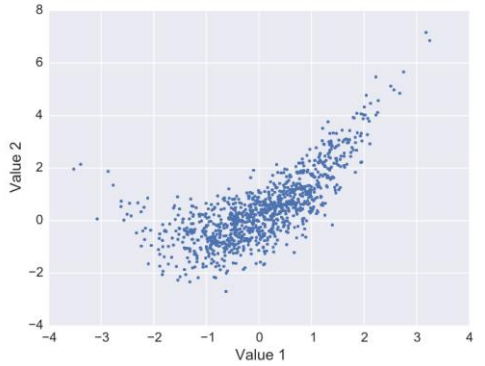
"I want to know how the **size** of the product affects **the cost of shipping**"

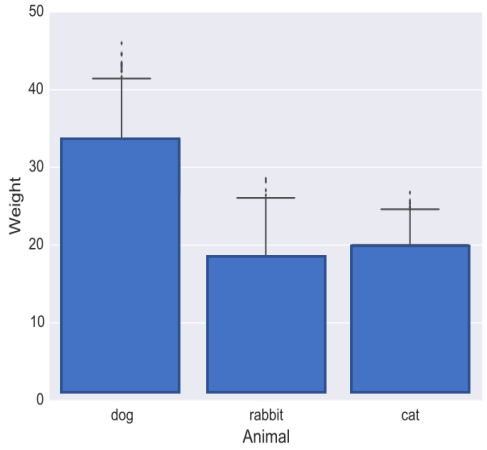
Charts for Two-Dimensional Data

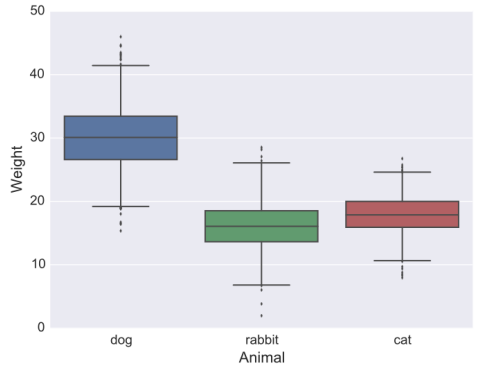
To analyze **numerical x numerical** data, use a **scatter plot**.

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**"

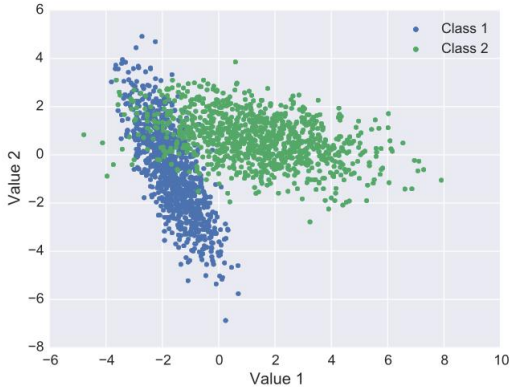
"I want to know how the **size** of the product and the **manufacturing location** affects **the cost of shipping**"

Charts for Three-Dimensional Data

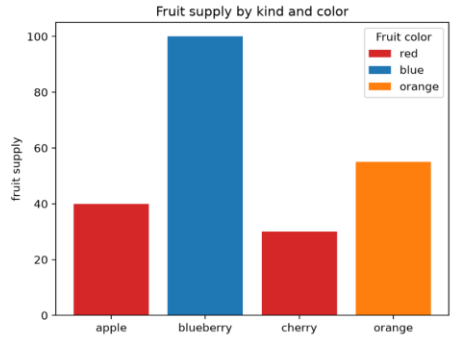
To analyze **numerical x numerical x numerical** data, use a **bubble plot** to compare all three together.



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



To analyze **numerical x ordinal/categorical x ordinal/categorical** data, use a **colored bar chart**.



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 x Ordinal/Categorical: Colored Scatter Plot	x Numerical: Colored Scatter Plot x Ordinal/Categorical: Colored Bar Chart
Ordinal/Categorical	x Numerical: Colored Scatter Plot x Ordinal/Categorical: Colored Bar Chart	x Numerical: Colored Bar Chart

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 in a state vs. the % of people who have been vaccinated in the state, where states are coded as using an outreach campaign vs. not

- 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. 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!

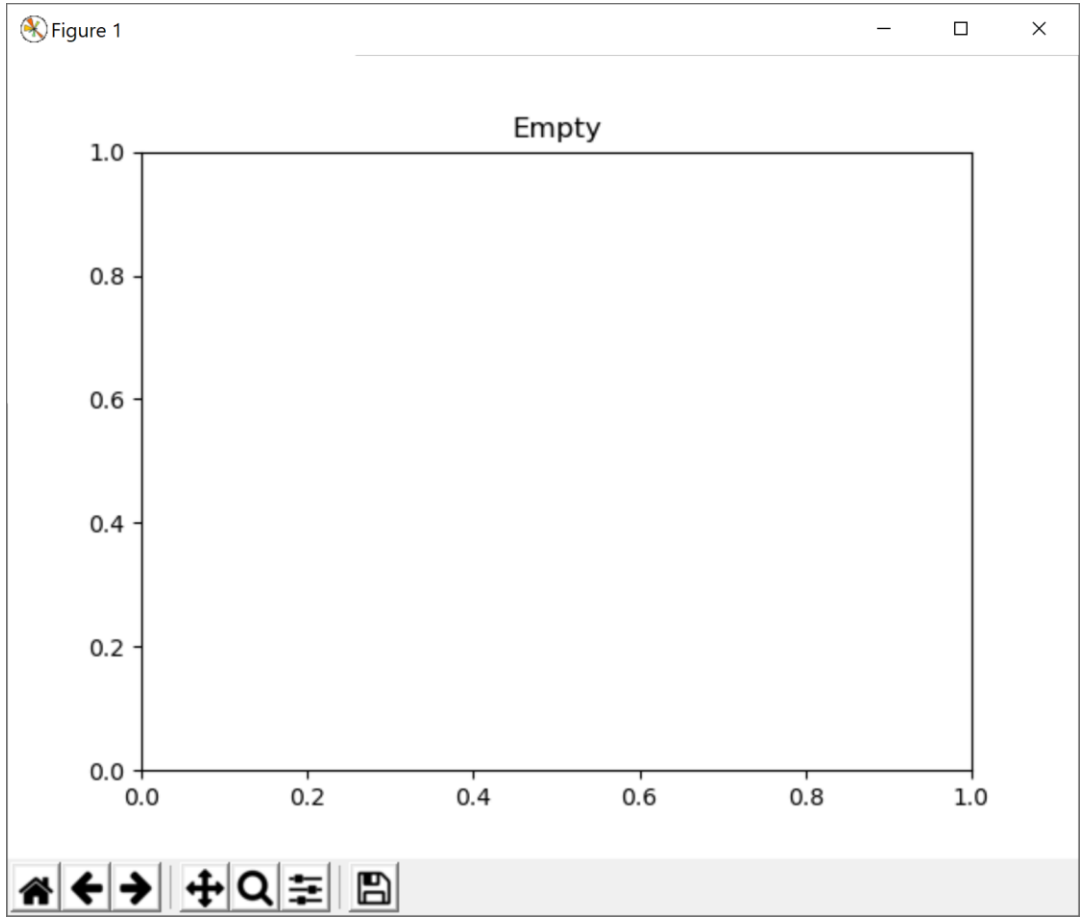
Draw Visualizations on the Plot

Matplotlib visualizations can be broken down into several components. We'll mainly care about one: the **plot** (called `plt`). This is a canvas that we can draw visualizations on.

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

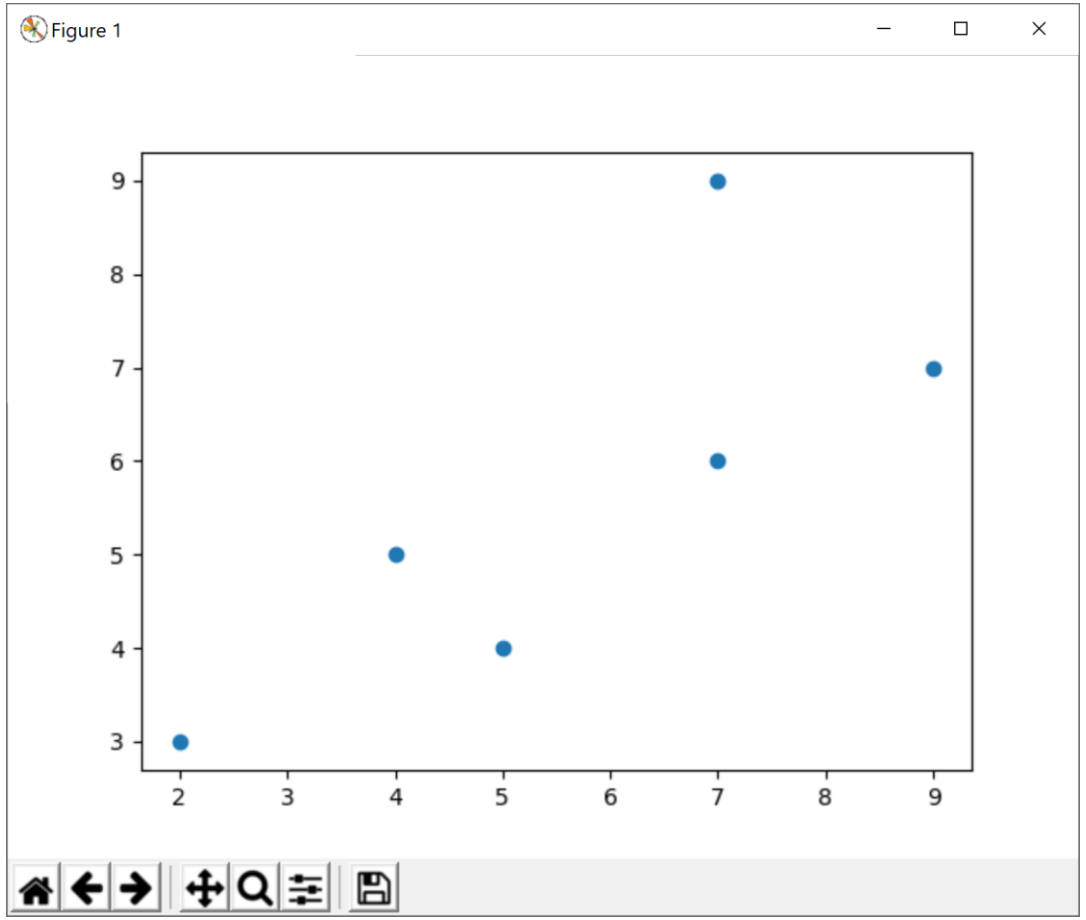
plt.title("Empty")
plt.show()
```



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()
```

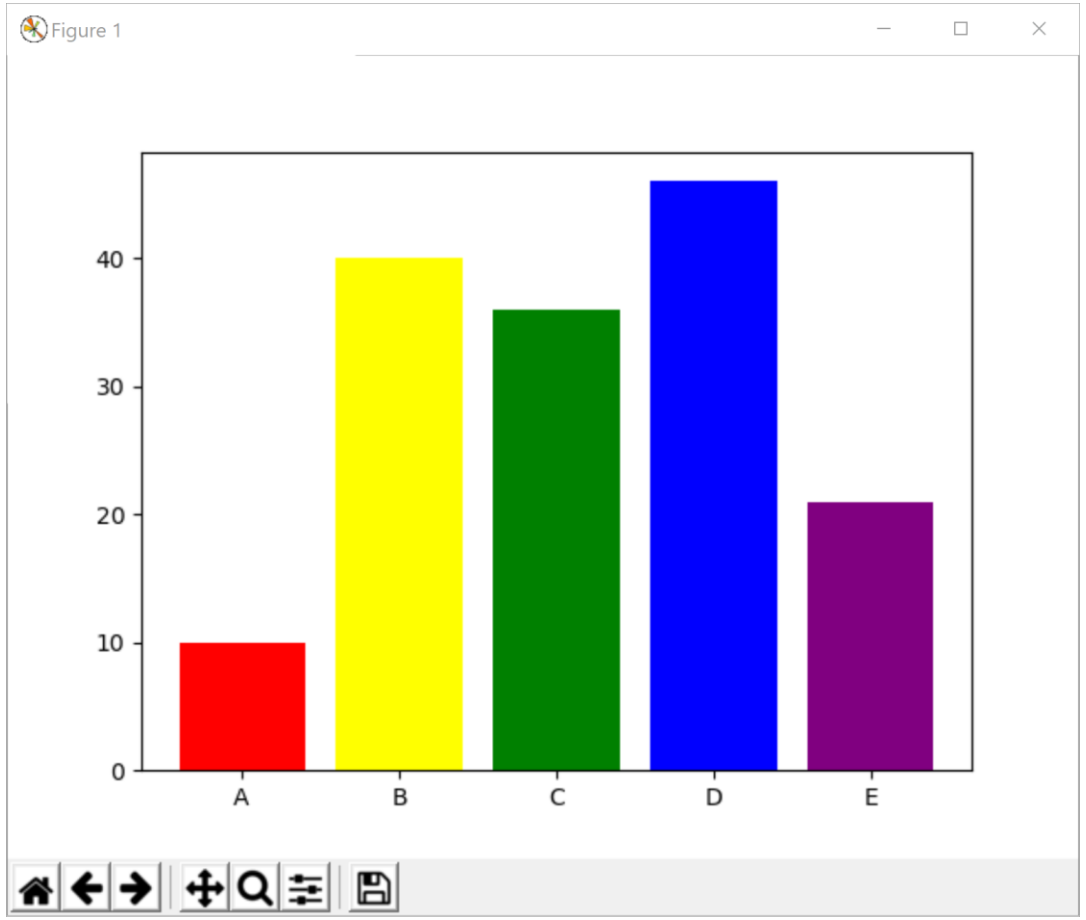


Visualization Methods have Keyword Args

You can **customize** how a visualization looks by adding **keyword arguments**. These are optional arguments that are added to a function call with `keyword=value`. 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 ]
lst = [ "red", "yellow", "green",
        "blue", "purple" ]
plt.bar(labels, yValues, color=lst)
plt.show()
```



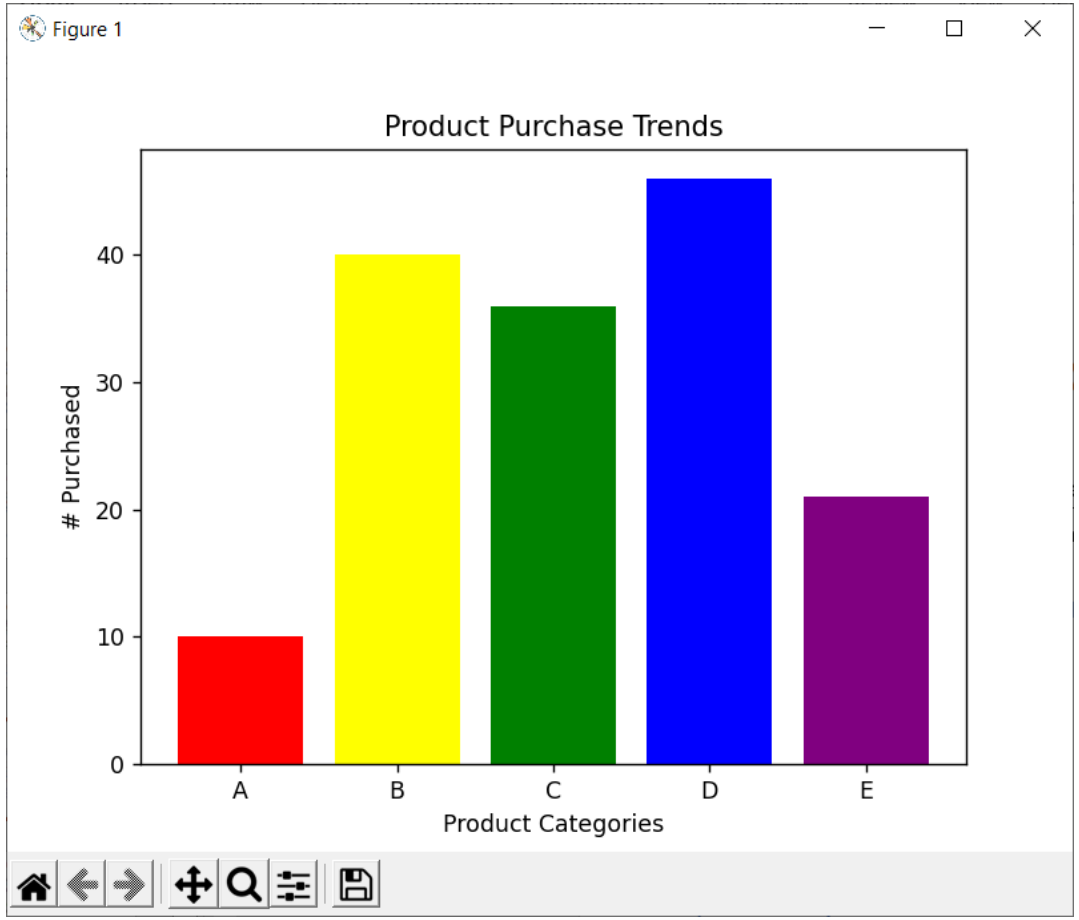
Further Customize with Plot Methods

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.

Data Analysis with Pandas

Pandas Makes Data Manageable

Pandas is a Python library for working with tabular data (think spreadsheets or tables) in code. It is the standard tool in Python for loading, cleaning, transforming, and analyzing data.

Pandas 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 pandas
```

Pandas has a user guide, which breaks down how the library works.

DataFrames

Pandas mostly works with a structure called a **DataFrame**, a spreadsheet table. DataFrames act like 2D lists, but it's just as easy to access data by **column** as it is by **row.**

You can build a DataFrame directly from a 2D list, or load one from a CSV/Excel/JSON file.

When we print a DataFrame, there are no commas between values, and row/column indexes are shown directly.

Some familiar properties still apply, e.g., `len(df)` gives the number of rows.

```
import pandas as pd

# different ways to create a data frame
df = pd.DataFrame([ [1, 2, 3],
                    [4, 5, 6] ]) # from a 2D list

df1 = pd.read_csv('data.csv') # from a csv
df2 = pd.read_excel('data.xlsx') # from an xlsx file
df3 = pd.read_json('data.json') # from a json file

print(df)
#    0  1  2
# 0  1  2  3
# 1  4  5  6

print(len(df)) # 2
```

DataFrame Columns

Where DataFrames get interesting is that you don't need to refer to columns by index. You can give them names instead, just like you would in a spreadsheet with a header!

We can do this by adding a keyword argument when we create a dataframe. For example, `columns = [...]` is a list of column names.

Column names are also loaded automatically when a table is loaded from a CSV/Excel file.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])

print(df)
#   Course Number                          Course Name  # Units
# 0        15-110                 Principles of Computing       10
# 1        15-112       Fundamentals of Programming and CS       12
# 2        15-251            Great Ideas in Theoretical CS       12
```

Exploring a DataFrame

While you could always print the DataFrame to see the whole content of it, there are methods in `pandas` that allow you to explore it in different ways.

This particularly matters when your data gets too big (think thousands of rows or more).

`df.head()`: preview the first 5 rows, useful for a quick sanity check

`df.info()`: see column types and how many non-null values each column has

`df.describe()`: get summary statistics (mean, min, max) for numeric columns

`df.shape`: check the size of the DataFrame as (rows, columns)

`df.columns`: list the column names

DataFrame Indexing

We can index into a DataFrame by column with an index operation, by providing the column name. For example, if we index by `"Course Number"` in `df`, we'll get the values in that column of the table.

Note that when the values are displayed, they are paired with their row indexes, and the column name and type.

We use `.iloc` with the row's position to index into a specific row.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])

print(df["Course Number"])
# 0    15-110
# 1    15-112
# 2    15-251
# Name: Course Number, dtype: str

print(df.iloc[0]["Course Number"]) # 15-110
print(df.iloc[0]["Course Name"]) # Principles of Computing
```

Adding columns and rows

We can use indexing to add a new column to a DataFrame. Just set the DataFrame index to the column directly equal to a list of values. Same syntax applies to replace a column rather than adding a new one.

Important: the list must be the same length as the DataFrame, so that the values can be matched up.

To add a new row, we can use `.loc` with the position right after the last row.

Important: the new row needs a value for every column in the DataFrame, in the same order the columns appear.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])

names = [ "Rivers", "Kosbie", "Ada" ]
df["Instructor1"] = names # adding a new column

# adding a new row
df.loc[len(df)] = ["15-213", "Intro to Computer Systems", 12, "Railing"]
```

Let's put these new skills into practice!

Exploring a Penguin dataset with pandas and matplotlib



On our course website we have uploaded a `penguin.csv` dataset about the **Palmer Penguins.**

It includes physical measurements (bill length/depth, flipper length, body mass) and sex for ~340 penguins across three species on islands in Antarctica, collected as part of the Palmer Station Long Term Ecological Research program.

1. Step 1: Download the dataset from our course website and create a Python file in the same directory
2. Step 2: Using pandas, load the dataset, explore and understand its structure
3. Step 3: Using pandas, find the average and standard deviation of the body mass measurements
4. Step 4 : Using matplotlib, plot a histogram of the Body Mass Distribution
    - Step 4.1 Add the average and standard deviation to the histogram as reference lines
5. Step 5: Using pandas, group the penguins by species and find their average body mass; plot this as a bar chart using matplotlib

Dataset source: https://allisonhorst.github.io/palmerpenguins/

Learning Goals

- Recognize how different data types - **categorical, ordinal,** and **numerical -** affect how that data can be interpreted and analyzed.

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

- Use **statistics, data management, and visualization libraries** to analyze and present information about a dataset.