

Business Process Automation with VBA and Python

Mr. Eddie Chow / 30 November 2024



Table Of Contents

Introduction to business process automation

Business Process automation with VBA

Business process automation with Python

Introduction to project management for business process automation

➔ Development and implementation of business process automation

Final Group Presentation



Intended Learning Outcomes

1. explain the basic concepts and principles of web scraping, including the ethical considerations and legal implications involved in data extraction from websites.
2. demonstrate the ability to utilize Python libraries such as BeautifulSoup and Scrapy to efficiently extract, parse, and clean data from various web sources.
3. learn to visualize the scraped data using libraries like Matplotlib or Seaborn, enabling them to create informative graphs and charts that effectively communicate insights derived from the data.
4. apply data analysis techniques to interpret the visualized data, drawing conclusions that can inform decision-making processes in real-world scenarios, such as market analysis or competitive intelligence.



Agenda

1. Introduction of Web Scraping
2. Overview of Web Scraping
3. Scraping Tools and Technologies
4. Scraping Environment Setup
5. Introduction to HTML and CSS
6. Beautiful Soup for Web Scraping
7. Introduction to Data Visualization
8. Data Visualization Tools and Technologies
9. Drawing Insights from Visualized Data
10. Best practices for presenting data visualizations

Web Crawler

Definition

- a web crawler is a computer program that browses the World Wide Web in a methodical, automated manner. (Wikipedia)

Utilities:

- Gather pages from the Web.
- Support a search engine, perform data mining and so on.

Object:

- Text, video, image and so on.
- Link structure

Web Features of a Crawler

Should provide:

- Distributed
- Scalable
- Performance and efficiency
- Quality
- Freshness
- Extensible

Some scraping knowledge

1. HTTP: the communication protocol
2. HTML: the language in which web pages are defined
3. JS: javascript (code executing in the browser)
4. CSS: style sheets, how web pages are styled. Important, but does not contain data.
5. JPG, PNG, BMP: images, usually not interesting
6. CSV / TXT / JSON / XML: data, interesting !!!

Obtaining Data

Data can come from:

- You curate it
- Someone else provides it, all pre-packaged for you (e.g., files)
- Someone else provides an API
- Someone else has available content, and you try to take it (web scraping)

Obtaining Data – Web Scraping

Web Scraping

- Using programs to get data from online
- Often much faster than manually copying data!
- Transfer the data into a form that is compatible with your code

Obtaining Data – Web Scraping

Why scraping the web?

- Vast source of information; can combine with multiple datasets
- Companies have not provided APIs
- Automate tasks
- Keep up with sites / real-time data
- Fun!

Robots.txt

- Specified by web site owner
- Gives instructions to web robots (e.g., your code)
- Located at the top-level directory of the web server
 - E.g., <http://google.com/robots.txt>

Robots.txt

- Protocol for giving spiders ("robots") limited access to a website

www.robotstxt.org/wc/norobots.html

- Website announced its request on what can(not) be crawled
 - For a server, create a file /robots.txt
 - This file specifies access restrictions

Robots.txt

- Protocol for giving spiders ("robots") limited access to a website

www.robotstxt.org/wc/norobots.html

- Website announced its request on what can(not) be crawled
 - For a server, create a file /robots.txt
 - This file specifies access restrictions

An example of robots.txt

- No rebot should visit any URL starting with "/yoursite/temp/", except the rebot called "searchengine":
- User-agent: *
- Disallow: /yoursite/temp/
- User-agent: searchengine
- Disallow:

Web Servers

- A server maintains a long-running process (also called a daemon), which listens on a pre-specified port
- It responds to requests, which is sent using a protocol called HTTP (HTTPS is secure)
- Our browser sends these requests and downloads the content, then displays it
- 2– request was successful, 4– client error, often `page not found`; 5– server error (often that your request was incorrectly formed)

Obtaining Data – Web Scraping

HTML

- Tags are denoted by angled brackets
- Almost all tags are in pairs e.g.,
`<p>Hello</p>`
- Some tags do not have a closing tag e.g.,
`
`

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
  </head>
  <body>
    <h1>Body Title</h1>
    <p>Body Content</p>
  </body>
</html>
```

Obtaining Data – Web Scraping

HTML

- `<html>`, indicates the start of an html page
- `<body>`, contains the items on the actual webpage (text, links, images, etc)
- `<p>`, the paragraph tag. Can contain text and links
- `<a>`, the link tag. Contains a link url, and possibly a description of the link
- `<input>`, a form input tag. Used for text boxes, and other user input
- `<form>`, a form start tag, to indicate the start of a form
- ``, an image tag containing the link to an image

Obtaining Data – Web Scraping

How to Web scrape:

1. Get the webpage content

- Requests (Python library) gets a webpage for you

2. Parse the webpage content

- (e.g., find all the text or all the links on a page)
- BeautifulSoup (Python library) helps you parse the webpage.
- Documentation:

<http://crummy.com/software/BeautifulSoup>

The Big Picture Recap

Data Sources Files, APIs, Webpages (via Requests)



Data Parsing Regular Expressions, BeautifulSoup

Data Structures/Storage Traditional lists/dictionaries, PANDAS

Models Linear Regression, Logistic Regression, kNN, etc

BeautifulSoup only concerns webpage data

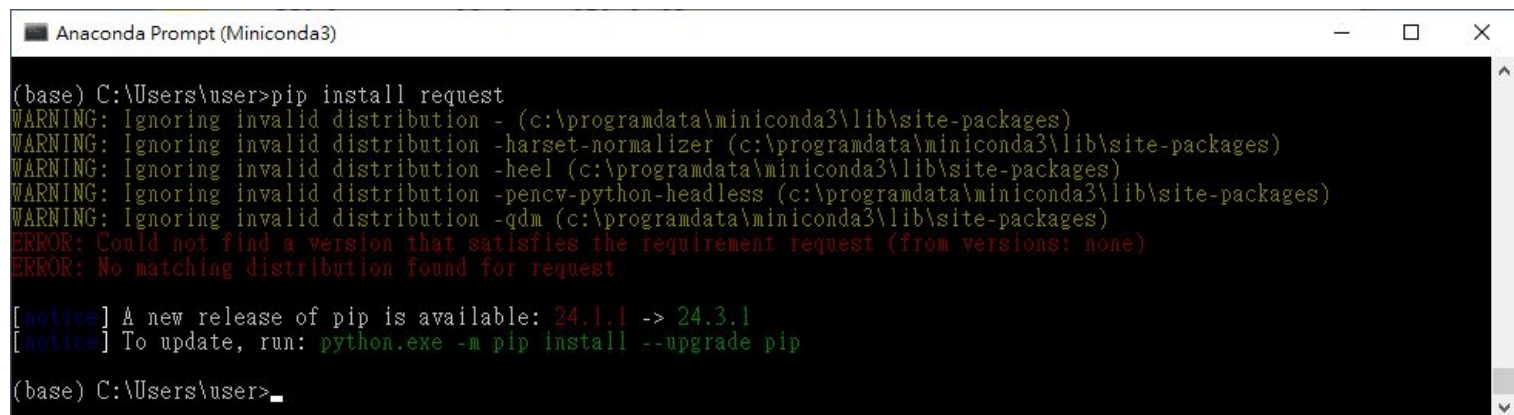
Obtaining Data – Web Scraping

1. Open tools “Anaconda Prompt” which will direct you to the command prompt.



```
Anaconda Prompt (Miniconda3)
(base) C:\Users\user>
```

2. Install python package requests by entering “pip install request”

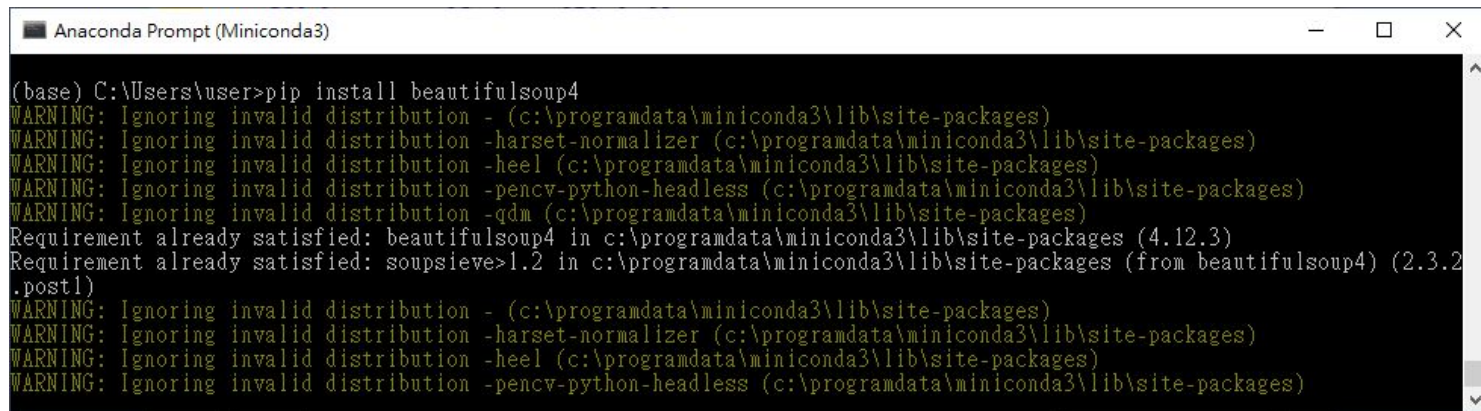


```
Anaconda Prompt (Miniconda3)
(base) C:\Users\user>pip install request
WARNING: Ignoring invalid distribution - (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -harset-normalizer (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -heel (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -qdm (c:\programdata\miniconda3\lib\site-packages)
ERROR: Could not find a version that satisfies the requirement request (from versions: none)
ERROR: No matching distribution found for request

[notice] A new release of pip is available: 24.1.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(base) C:\Users\user>_
```


Obtaining Data – Web Scraping


1. Install python package beautifulsoup4 by entering “pip install beautifulsoup4”



```
Anaconda Prompt (Miniconda3)

(base) C:\Users\user>pip install beautifulsoup4
WARNING: Ignoring invalid distribution - (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -harset-normalizer (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -heel (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -qdm (c:\programdata\miniconda3\lib\site-packages)
Requirement already satisfied: beautifulsoup4 in c:\programdata\miniconda3\lib\site-packages (4.12.3)
Requirement already satisfied: soupsieve>1.2 in c:\programdata\miniconda3\lib\site-packages (from beautifulsoup4) (2.3.2.post1)
WARNING: Ignoring invalid distribution - (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -harset-normalizer (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -heel (c:\programdata\miniconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -pencv-python-headless (c:\programdata\miniconda3\lib\site-packages)
```

2. Enter “python” in the command



```
Anaconda Prompt (Miniconda3) - python

(base) C:\Users\user>python
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Obtaining Data – Web Scraping

1. Get the webpage content

Requests (Python library) gets a webpage for you

```
import requests
```

```
url =
```

```
"https://www.nytimes.com/international/"
```

```
page = requests.get(url)
```

```
page.status_code
```

```
page.content
```

Obtaining Data – Web Scraping

1. Get the webpage content

Requests (Python library) gets a webpage for you

```
import requests
```

```
url =
```

```
"https://www.nytimes.com/  
1/"
```

```
page = requests.get(url)
```

```
page.status_code
```

```
page.content
```

Gets the status from
the webpage request.

200 means success.

404 means page not
found.

Obtaining Data – Web Scraping

1. Get the webpage content

Requests (Python library) gets a webpage for you

```
import requests
```

```
url =
```

```
"https://www.nytimes.com/international/"
```

```
page = requests.get(url)
```

```
page.status_code
```

```
page.content
```

Returns the content of the response, in bytes.

Obtaining Data – Web Scraping

2. Parse the webpage content

BeautifulSoup (Python library) helps you parse a webpage

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(page.content,
    "html.parser")

soup.title

soup.title.text
```

Obtaining Data – Web Scraping

2. Parse the webpage content

BeautifulSoup (Python library) helps you parse a webpage

```
from bs4 import BeautifulSoup
```

```
soup = BeautifulSoup(page.content,  
"html.parser")
```

```
soup.title
```

```
soup.title.text
```

Returns the full context,

including the title tag. e.g.,

```
<title data-rh="true">The New  
York Times - Breaking  
News</title>
```


Obtaining Data – Web Scraping

2. Parse the webpage content

BeautifulSoup (Python library) helps you parse a webpage

```
soup = BeautifulSoup(page.content,  
"html.parser")
```

```
soup.title
```

```
soup.title.text
```

Returns the text part of the title tag. e.g.,

The New York Times - Breaking
News

Obtaining Data – Web Scraping

BeautifulSoup

- Helps make messy HTML digestible
- Provides functions for quickly accessing certain sections of HTML content

Exempl

```
import bs4
## get bs4 object
soup = bs4.BeautifulSoup(source)
## all a tags
soup.findAll('a')
## first a
soup.find('a')
## get all links in the page
link_list = [l.get('href') for l in soup.findAll('a')]
```

Obtaining Data – Web Scraping

HTML is a tree

- You don't have to access the HTML as a tree, though;
- Can immediately search for tags/content of interest (a la previous slide)

Exempl

```
tree = bs4.BeautifulSoup(source)

## get html root node
root_node = tree.html

## get head from root using contents
head = root_node.contents[0]

## get body from root
body = root_node.contents[1]

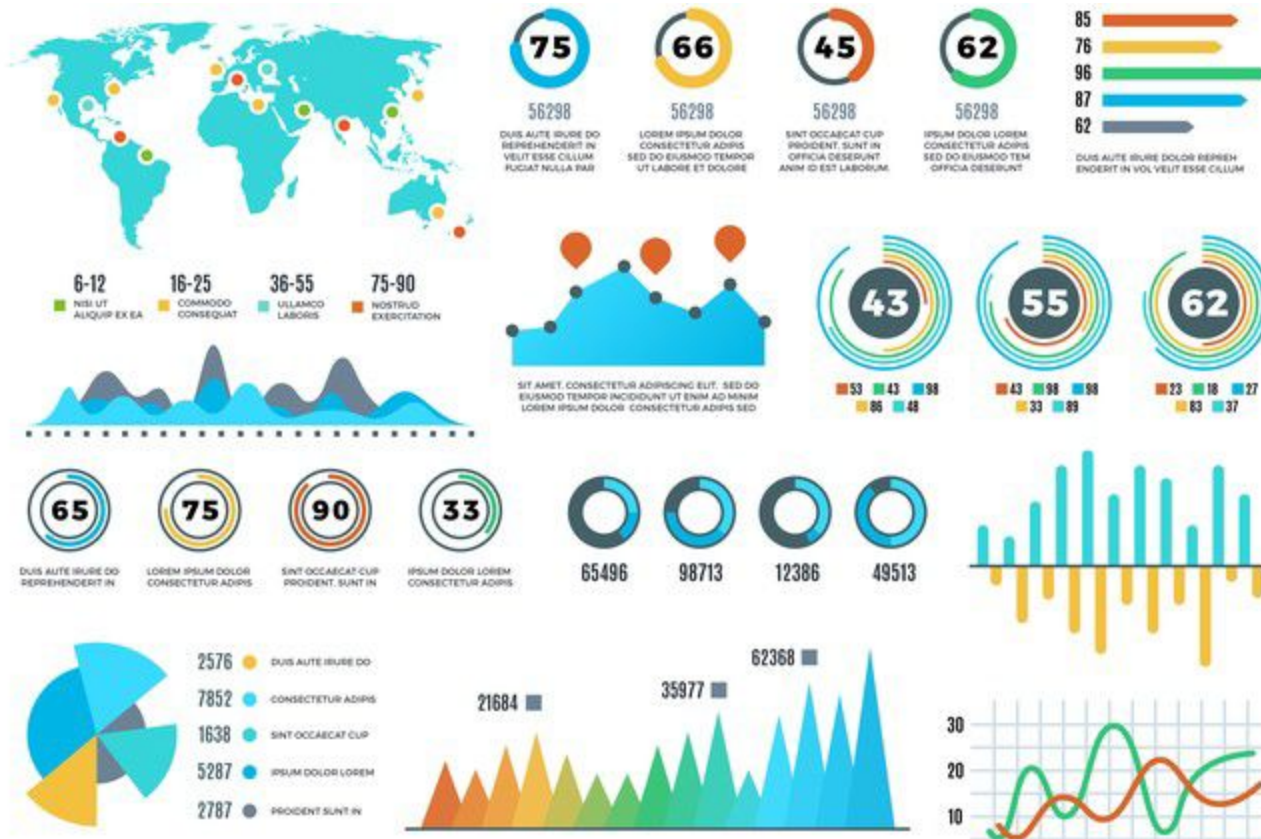
## could directly access body
tree.body
```



Data Visualizations

1. describe the contemporary trends of process automation and explain the opportunities and challenges of business process automation;
2. outline key steps in process automation project management and illustrate the significance of each step;
3. apply computational tools to implement process automation;
4. discuss the development of process automation and practical cases for business.

Data Visualizations



Data Visualizations

- Data visualization is the practice of translating information into a visual context, such as a map or graph, to make data easier for the human brain to understand and pull insights from.
- The main goal of data visualization is to make it easier to identify patterns, trends and outliers in large data sets.
- The term is often used interchangeably with others, including information graphics, information visualization and statistical graphics.
- Data visualization is one of the steps of the data science process, which states that after data has been collected, processed and modeled, it must be visualized for conclusions to be made.
- Data visualization is also an element of the broader data presentation architecture (DPA) discipline, which aims to identify, locate, manipulate, format and deliver data in the most efficient way possible.



Benefits of Data Visualizations

- The ability to absorb information quickly, improve insights and make faster decisions;
- An increased understanding of the next steps that must be taken to improve the organization;
- An improved ability to maintain the audience's interest with information they can understand;
- An easy distribution of information that increases the opportunity to share insights with everyone involved;
- Eliminate the need for data scientists since data is more accessible and understandable; and
- An increased ability to act on findings quickly and, therefore, achieve success with greater speed and less mistakes.

Data Visualization Roles - Change over time



Line chart



+Comparisons

Most common chart type for showing change over time. A point is plotted for each time period from left to right; each point's vertical position indicates the feature's value. Points are connected by line segments to emphasize progression across time.



Sparkline



+Comparisons

A miniature line chart with little to no labeling, designed to be placed alongside text or in tables. Provides a high-level overview without attracting too much attention. Can also be seen in a sparkbar form, or miniature bar chart (see below).

Data Visualization Roles - Change over time



Connected scatter plot



+Relationships

Shows change over time across two numeric variables (see scatter plot in *Relationships*). Line segments still connect points across time, but they may not consistently go from left to right like in a line chart.



Bar chart

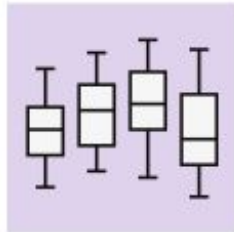


+Distributions

+Comparisons

Each time period is associated with a bar; each bar's value is represented in its height above (or below) a zero-baseline. Works best when there aren't too many time periods to show.

Data Visualization Roles - Change over time



Box plot

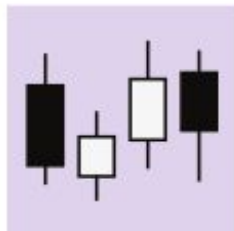


+Distributions

+Comparisons

Each time period is associated with a box and whiskers; each set of box and whiskers shows the range of the most common data values. Best when there are multiple recordings for each time period and a distribution of values needs to be plotted.

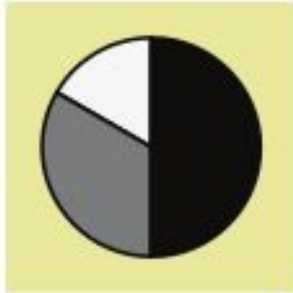
Tracking change over time is of key interest in the financial domain. One specialist chart developed for this field includes the following:



Candlestick chart ◆

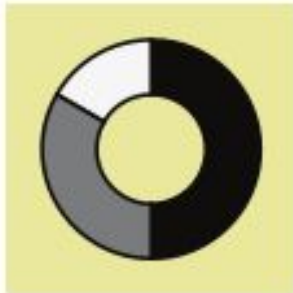
Looks like a box plot, but each box and whiskers encodes different statistics. The box ends indicate opening and closing prices, while color indicates the direction of change.

Data Visualization Roles - Part-to-whole composition



Pie chart ●

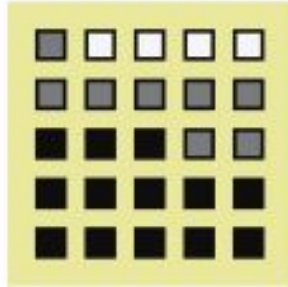
The whole is represented by a filled circle. Parts are proportional slices from that circle, one for each categorical group. Best with five or fewer slices with distinct proportions.



Doughnut chart ●

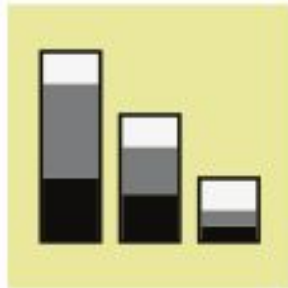
A pie chart with a hole in the center. This central area can be used to show a relevant single numeric value. Sometimes used as an aesthetic alternative to a standard progress bar (see stacked bar chart below).

Data Visualization Roles - Part-to-whole composition



Waffle chart / grid plot

Squares laid out in a (typically) 10 x 10 grid; each square represents one percent of the whole. Squares are colored based on categorical group size.



Stacked bar chart

A bar chart (see *Change over time* or *Distributions*) where each bar has been divided into multiple sub-bars to show a part-to-whole breakdown. A single stacked bar can be used as an alternative to the pie or doughnut chart; people tend to make more precise judgments of length over area or angle.

Data Visualization Roles - Part-to-whole composition



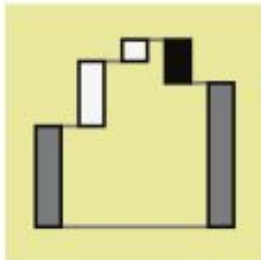
Stacked area chart ●

A line chart (see *Change over time*) where shaded regions are added under the line to divide the total into sub-group values.



Stream graph ◆

Modified version of the stacked area chart where areas are stacked around a central axis. Highlights relative changes instead of exact values.

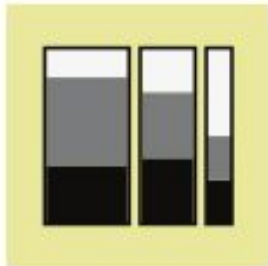


Waterfall chart ◆

Augments a change over time with a part-to-whole decomposition. Bars on the ends depict values at two time points, and lengths of intermediate floating bars' show the decomposition of the change between points.

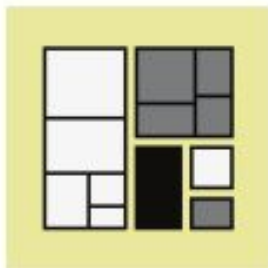
Data Visualization Roles - Part-to-whole composition

Certain part-to-whole compositions follow a hierarchical form. In these cases, each part can be divided into finer parts on lower levels. Here are a couple of more specialized chart types for visualizing this type of data:



Mosaic plot / Marimekko chart ■

Can be thought of as a stacked bar divided on both axes. A box is divided on one axis based on one categorical variable, then each sub-box is divided in the other axis based on a second categorical variable.



Treemap ◆

Can be thought of as a more generalized Marimekko plot. Sub-boxes do not need to have a consistent cut direction at a particular hierarchy level, and there can be more than two levels of hierarchy.

Data Visualization Roles - Flows and processes



Funnel chart

Seen in business contexts, showing how people encounter a product and eventually become users or customers. One bar is plotted for each stage, whose lengths reflect the number of users. Connecting regions emphasize connections in stages and give the chart type's namesake shape.



Parallel sets chart

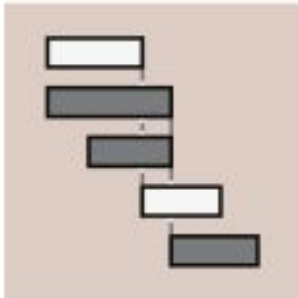
Multiple part-to-whole divisions on different dimensions are depicted as parallel stacked bars. Connecting regions show how different subgroups relate to one another between dimensions.

Data Visualization Roles - Flows and processes



Sankey diagram ◆

The width of the colored region shows the relative volume at each part of a process. Allows for multiple sources of inputs and outputs to be visualized.



Gantt chart ■

Used for project scheduling, breaking them down into individual tasks. Each task is associated with a bar, providing a timeline for when each task should begin and end.

Data Visualization Roles - Geographical data



Scatter map ●

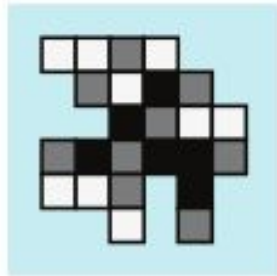
Scatter plot built on top of a geographical map, using geographic coordinates as point positions.



Bubble map ●

Bubble chart built on top of a geographic map, where point size is an indicator of value. Can also be used to group together points in a scatter map if they are too dense.

Data Visualization Roles - Geographical data



2-d histogram ●

Heatmaps can be built on top of geographic areas. Sometimes seen with a hexagon-shaped grid rather than a rectangular grid. May distort the geography on its edges.



Isopleth / contour map ◆

2-d density curve built on top of a geographic map.



Connection map ◆

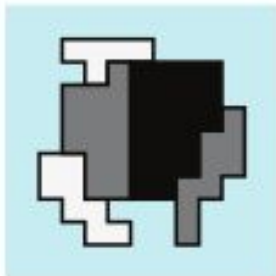
Network information and flows built on top of a geographic map.

Data Visualization Roles - Geographical data



Choropleth ●

Similar to a heatmap, but colors are assigned to geopolitical regions rather than an arbitrary grid. Values are often in the form of rates or ratios to avoid distortion due to population density.



Cartogram ◆

Geopolitical regions sized by value. This necessarily requires distortion in shapes and topology.



Data Visualization Tools

- Tableau
- Infogram
- ChartBlocks
- D3.js
- Google Charts
- Fusion Charts
- Chart.js



Visualization using Programming

- Python
 - matplotlib
 - seaborn
 - plotly
 - pylab
- R
 - graphics
 - ggplot2



Exploratory data analysis (EDA)

Exploring your data is a crucial step in data analysis. It involves:

Organising the data set

Plotting aspects of the data set

Maybe producing some numerical summaries; central tendency and spread, etc.

“Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone.”

- John Tukey.

Exploratory Data Analysis (EDA)

Why?

- EDA encompasses the “*explore* data” part of the data science process
- EDA is crucial but often overlooked:
 - If your data is bad, your results will be bad
 - Conversely, understanding your data well can help you create smart, appropriate models

Exploratory Data Analysis (EDA)

What?

1. Store data in data structure(s) that will be convenient for exploring/processing
(Memory is fast. Storage is slow)
2. Clean/format the data so that:
 - Each row represents a single object/observation/entry
 - Each column represents an attribute/property/feature of that entry
 - Values are numeric whenever possible
 - Columns contain atomic properties that cannot be further decomposed*

* Unlike food waste, which can be composted.

Please consider composting food scraps.

Exploratory Data Analysis (EDA)

What? (continued)

3. Explore **global** properties: use histograms, scatter plots, and aggregation functions to summarize the data
4. Explore **group** properties: group like-items together to compare subsets of the data (are the comparison results reasonable/expected?)

This process transforms your data into a format which is easier to work with, gives you a basic overview of the data's properties, and likely generates several questions for you to follow-up in subsequent analysis.

Exercise 1

Starting notebook:

https://github.com/innoviai/ipa_courses/blob/main/Lecture%203/python_data_visulisation_start.ipynb

Step

1. *Open “Anaconda Prompt” and type “python”*
2. *Import library “matplotlib” by entering code “import matplotlib”*

Exercise 1

Step 2

```
from matplotlib import pyplot as plt
```

```
# x-axis values
```

```
x = [5, 2, 9, 4, 7]
```

```
# Y-axis values
```

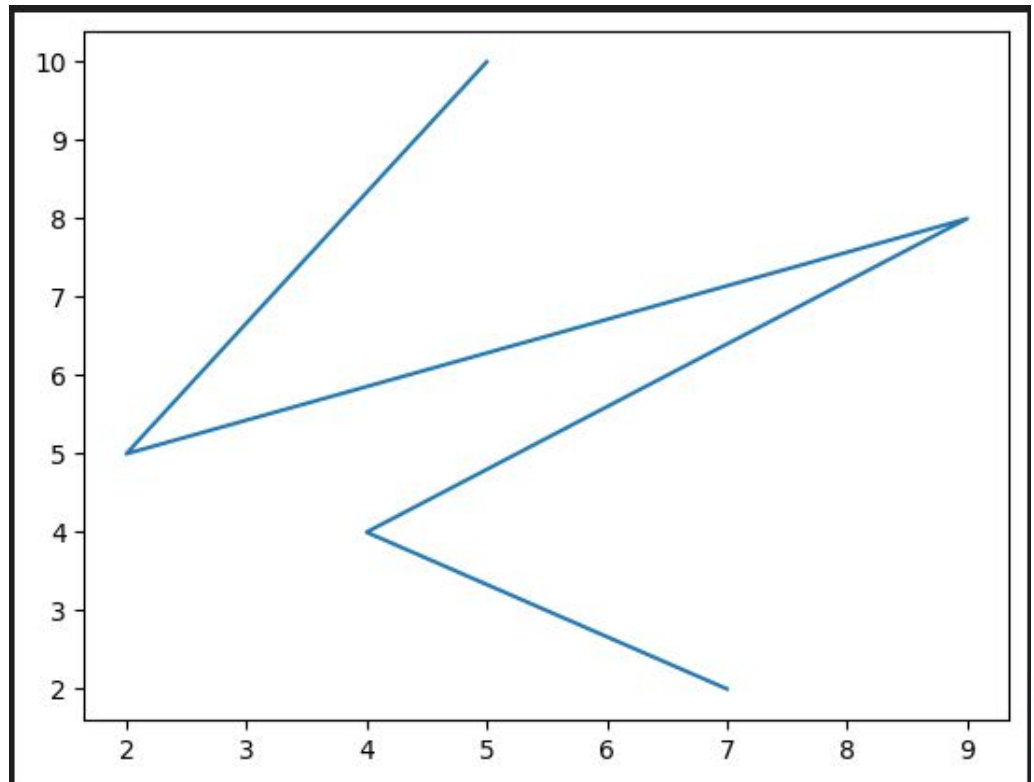
```
y = [10, 5, 8, 4, 2]
```

```
# Function to plot
```

```
plt.plot(x, y)
```

```
# function to show the plot
```

```
plt.show()
```



Exercise 1

Step 4

#scatter

x-axis values

x = [5, 2, 9, 4, 7]

Y-axis values

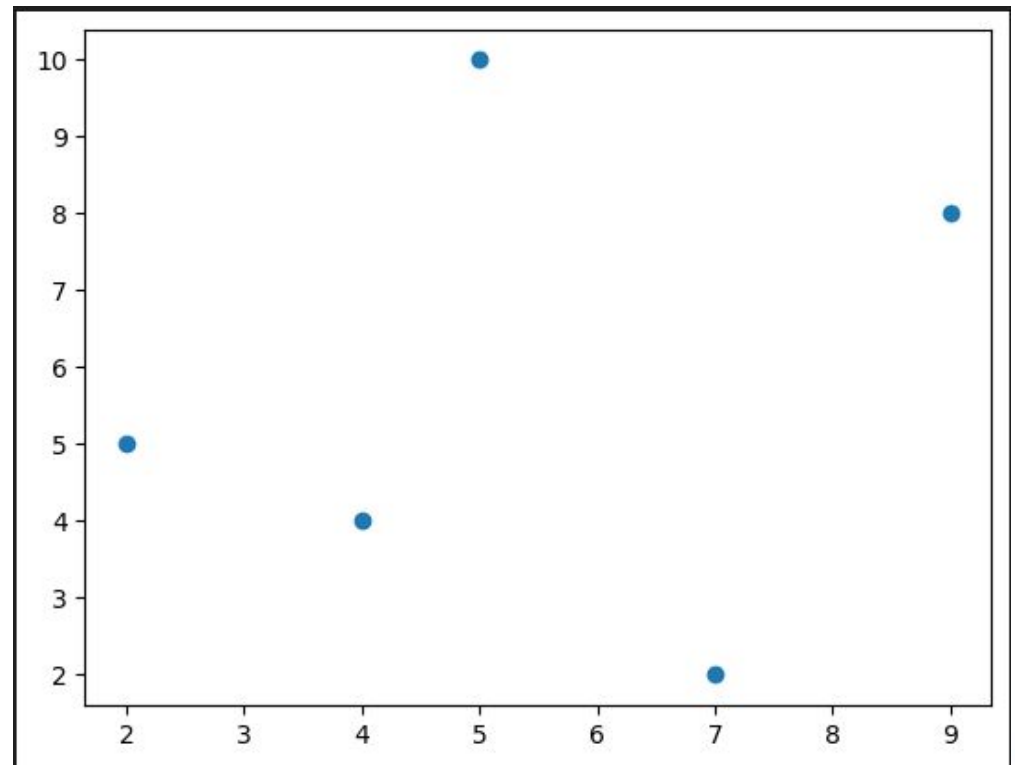
y = [10, 5, 8, 4, 2]

Function to plot scatter

plt.scatter(x, y)

function to show the plot

plt.show()



Exercise 1

Step 5

pip install plotly numpy

#using plotly library and display lines chart

import numpy as np

import plotly.graph_objects as go

Generate sample data

x = np.linspace(0, 10, 100) # 100 points from
0 to 10

y = np.sin(x) # y is the sine of x

Create a plotly figure

fig = go.Figure()

Add a line plot

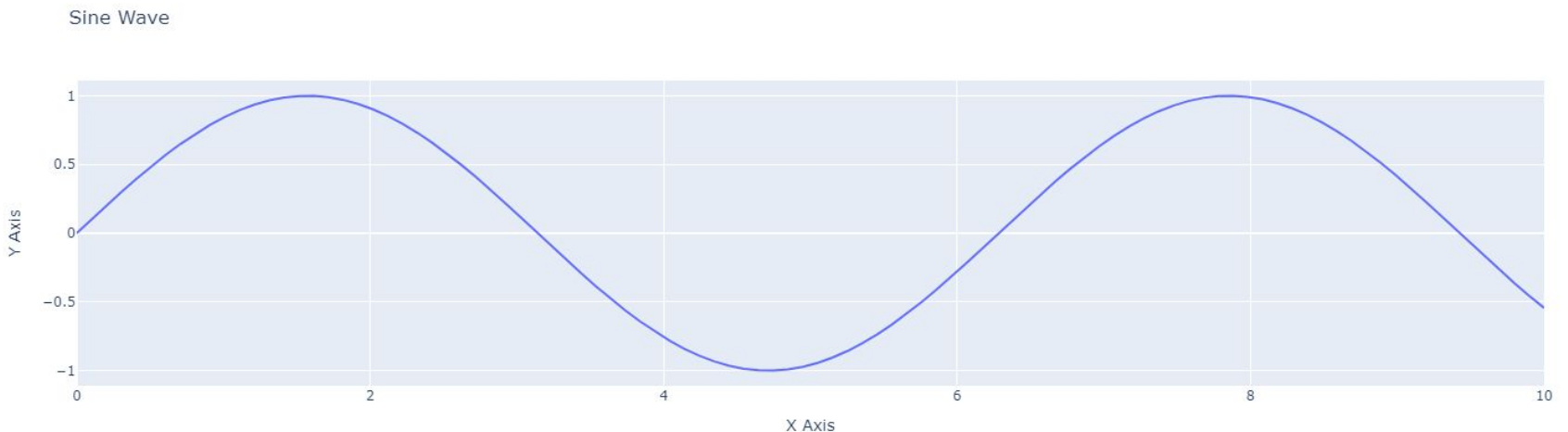
fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Sine
Wave'))

Customize layout

fig.update_layout(title='Sine Wave',
xaxis_title='X Axis',
yaxis_title='Y Axis')

Show the plot

fig.show()



Exercise 1

Step 6

#using plotly library and display histogram chart

import numpy as np

import plotly.graph_objects as go

Generate sample data for the sine wave

x = np.linspace(0, 10, 100) # 100 points from 0 to 10

y = np.sin(x) # y is the sine of x

Generate sample data for the histogram

data = np.random.normal(loc=0, scale=1,
size=1000) # 1000 random values from a
normal distribution

Create a plotly figure

fig = go.Figure()

Add a line plot for the sine wave

fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Sine Wave'))

Add a histogram

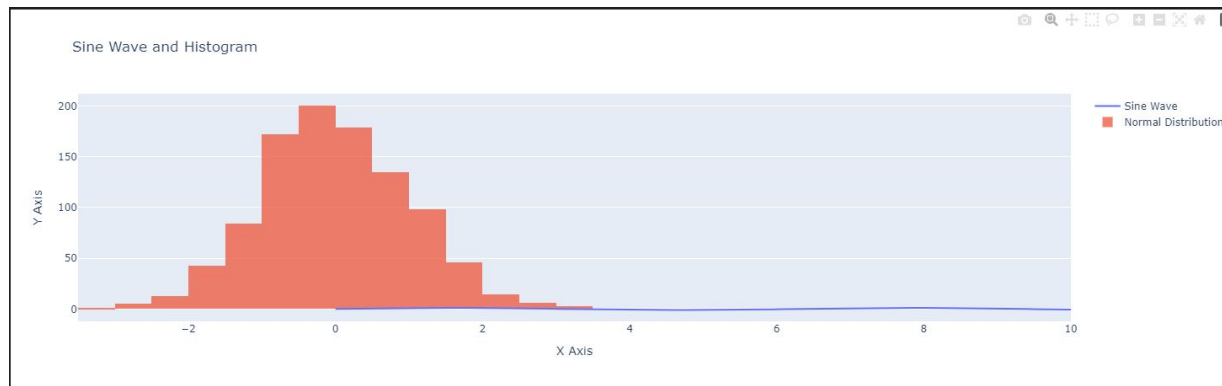
fig.add_trace(go.Histogram(x=data, name='Normal Distribution', opacity=0.75, nbinsx=30))

Customize layout

fig.update_layout(title='Sine Wave and Histogram',
xaxis_title='X Axis',
yaxis_title='Y Axis',
barmode='overlay') # Overlay histograms

Show the plot

fig.show()



Exercise 1

Step 7

```
#using plotly, display histogram chart and download chart
import numpy as np
import plotly.graph_objects as go
```

```
# Generate sample data for the sine wave
x = np.linspace(0, 10, 100) # 100 points from 0 to 10
y = np.sin(x)               # y is the sine of x
```

```
# Generate sample data for the histogram
data = np.random.normal(loc=0, scale=1, size=1000) # 1000
random values from a normal distribution
```

```
# Create a plotly figure
fig = go.Figure()
```

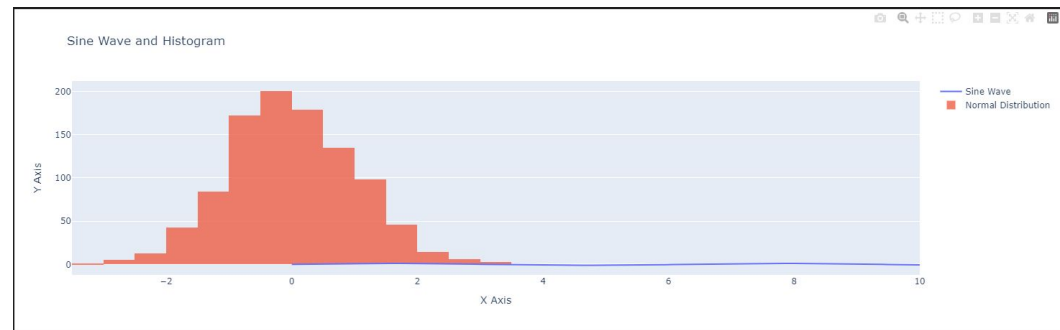
```
# Add a line plot for the sine wave
fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Sine
Wave'))
```

```
# Add a histogram
fig.add_trace(go.Histogram(x=data, name='Normal
Distribution', opacity=0.75, nbinsx=30))
```

```
# Customize layout
fig.update_layout(title='Sine Wave and Histogram',
                  xaxis_title='X Axis',
                  yaxis_title='Y Axis',
                  barmode='overlay') # Overlay histograms
```

```
# Show the plot
fig.show()
# Show the plot
fig.show()
```

```
# Save the figure as a PNG file
fig.write_image('sine_wave_histogram.png')
```



Download the data

Download the Pokemon dataset from:

https://github.com/innoviai/ipa_courses/blob/main/Lecture%203/pokemon_dataset.csv

Save the pokemon dataset file in a location you'll remember.



Reading in the data

First we import the Python packages we are going to use.
Then we use Pandas to load in the dataset as a data frame.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

```
df1 = pd.read_csv("filepath/to/where/data/is/stored/pokemon_dataset.csv",
index_col=0, encoding="ISO-8859-1")
```

NOTE: The argument `index_col` argument states that we'll treat the first column of the dataset as the ID column.

NOTE: The encoding argument allows us to by pass an input error created by special characters in the data set.

Examine the data set

```
print(df1.head())
```

	Name	Type 1	Type 2	Total	...	Sp. Def	Speed	Stage	Legendary
#					...				
1	Bulbasaur	Grass	Poison	318	...	65	45	1	False
2	Ivysaur	Grass	Poison	405	...	80	60	2	False
3	Venusaur	Grass	Poison	525	...	100	80	3	False
4	Charmander	Fire	NaN	309	...	50	65	1	False
5	Charmeleon	Fire	NaN	405	...	65	80	2	False

```
print(df1.describe())
```

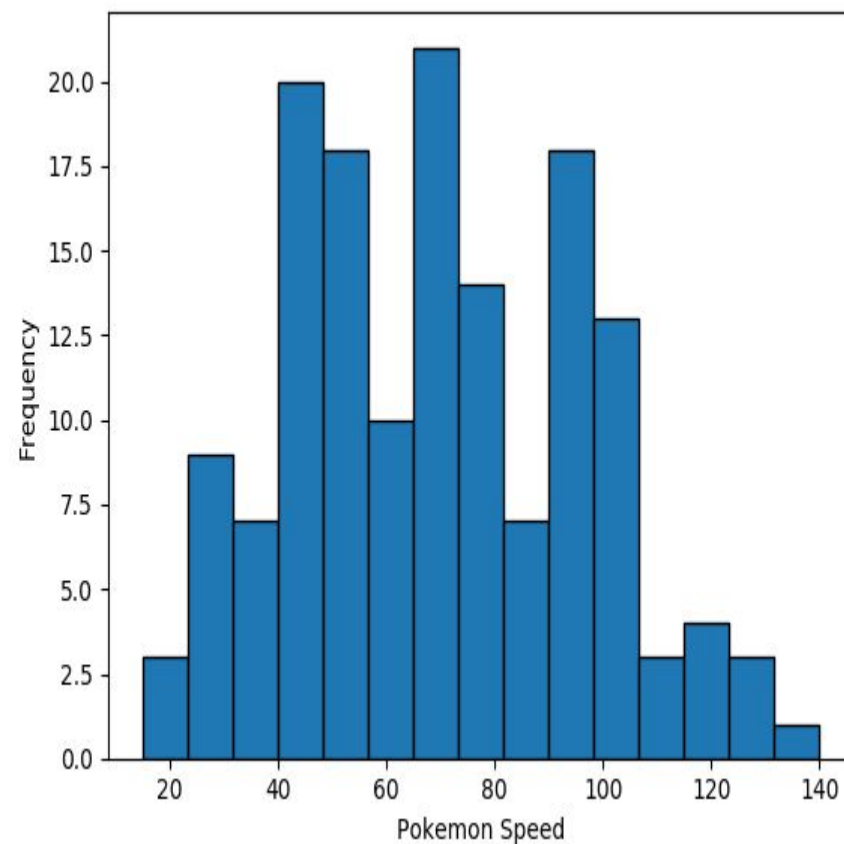
	Total	HP	Attack	...	Sp. Def	Speed	Stage
count	151.000000	151.000000	151.000000	...	151.000000	151.000000	151.000000
mean	407.07947	64.211921	72.549669	...	66.019868	68.933775	1.582781
std	99.74384	28.590117	26.596162	...	24.197926	26.746880	0.676832
min	195.000000	10.000000	5.000000	...	20.000000	15.000000	1.000000
25%	320.000000	45.000000	51.000000	...	49.000000	46.500000	1.000000
50%	405.000000	60.000000	70.000000	...	65.000000	70.000000	1.000000
75%	490.000000	80.000000	90.000000	...	80.000000	90.000000	2.000000
max	680.000000	250.000000	134.000000	...	125.000000	140.000000	3.000000

We could spend time staring at these numbers, but that is unlikely to offer us any form of insight.

We could begin by conducting all of our statistical tests.

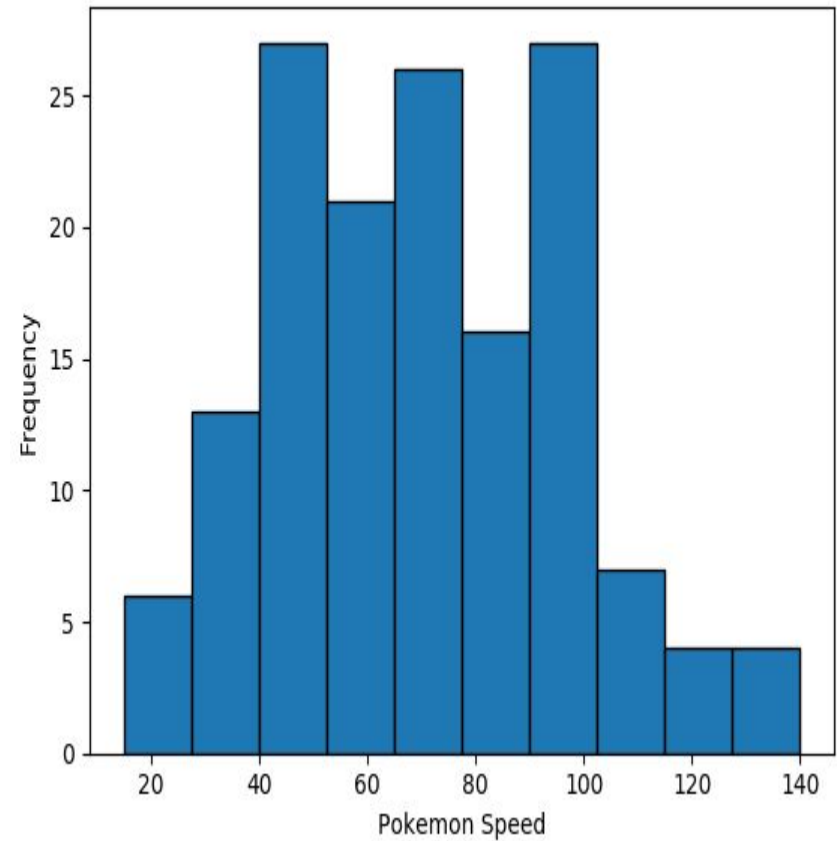
However, a good field commander never goes into battle without first doing a reconnaissance of the terrain...

This is exactly what EDA is for...



Plotting a histogram in Python

```
g = plt.hist(df1['Speed'], histtype='bar', ec='black',)  
g = plt.xlabel('Pokemon Speed')  
g = plt.ylabel('Frequency')  
plt.show()
```

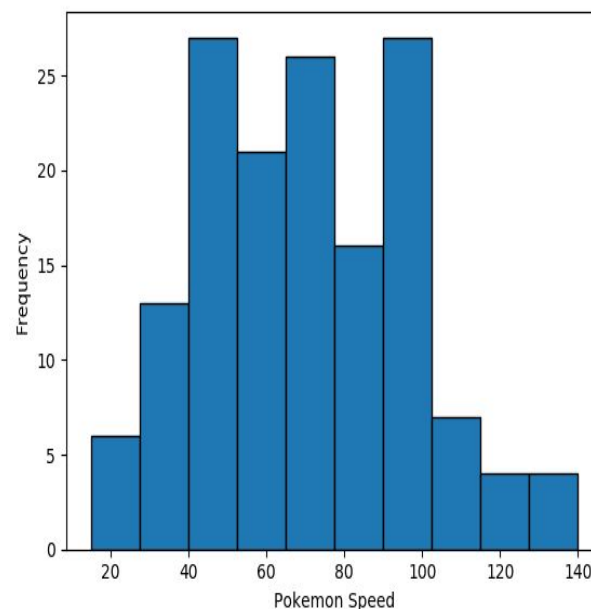
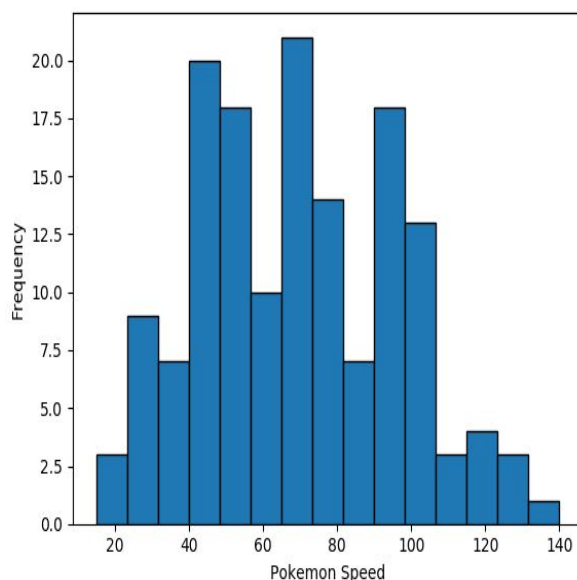


Bins

You may have noticed the two histograms we've seen so far look different, despite using the **exact** same data.

This is because they have different bin values.

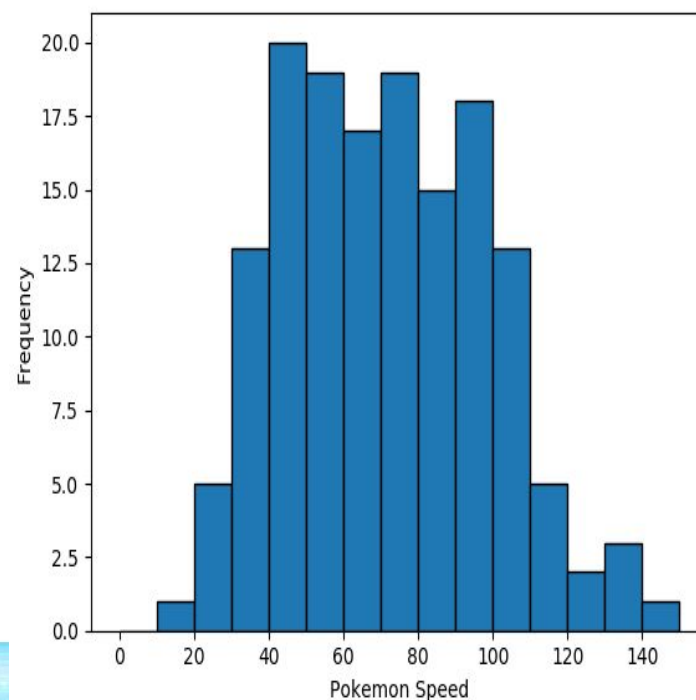
The left graph used the default bins generated by `plt.hist()`, while the one on the right used bins that I specified.



There are a couple of ways to manipulate bins in `matplotlib`.

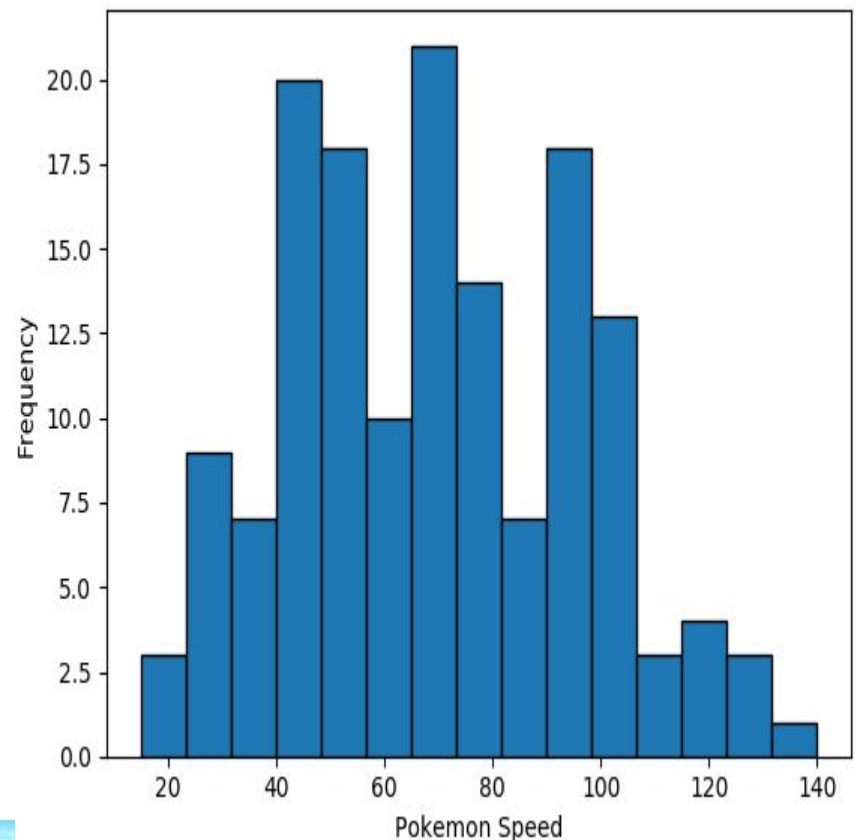
Here, I specified where the edges of the bars of the histogram are; the bin edges.

```
bin_edges = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
g = plt.hist(df1['Speed'], histtype='bar', ec='black', bins=bin_edges)
g = plt.xlabel('Pokemon Speed')
g = plt.ylabel('Frequency')
plt.show()
```



You could also specify the number of bins, and Matplotlib will automatically generate a number of evenly spaced bins.

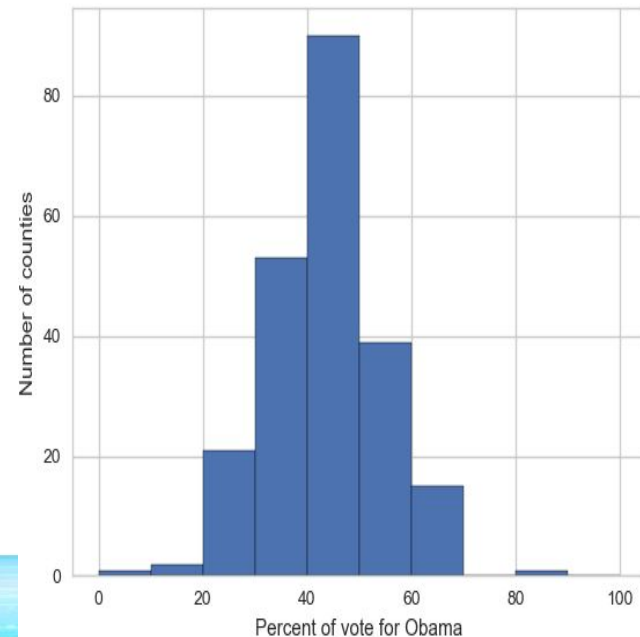
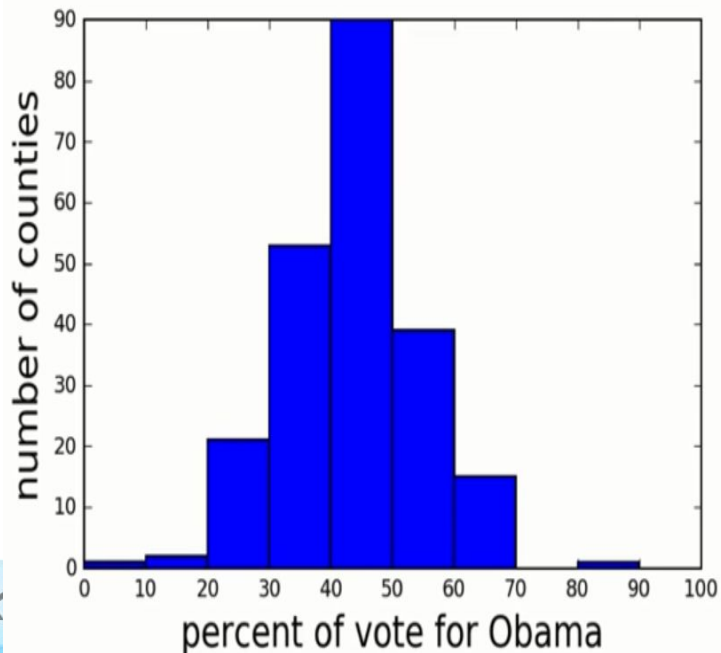
```
g = plt.hist(df1['Speed'], histtype='bar', ec='black', bins=15)
g = plt.xlabel('Pokemon Speed')
g = plt.ylabel('Frequency')
plt.show()
```



Matplotlib is a powerful, but sometimes unwieldy, Python library.

Seaborn provides a high-level interface to Matplotlib and makes it easier to produce graphs like the one on the right.

Some IDEs incorporate elements of this “under the hood” nowadays.





Benefits of Seaborn

Seaborn offers:

- Using default themes that are aesthetically pleasing.
- Setting custom colour palettes.
- Making attractive statistical plots.
- Easily and flexibly displaying distributions.
- Visualising information from matrices and DataFrames.

The last three points have led to Seaborn becoming the exploratory data analysis tool of choice for many Python users.



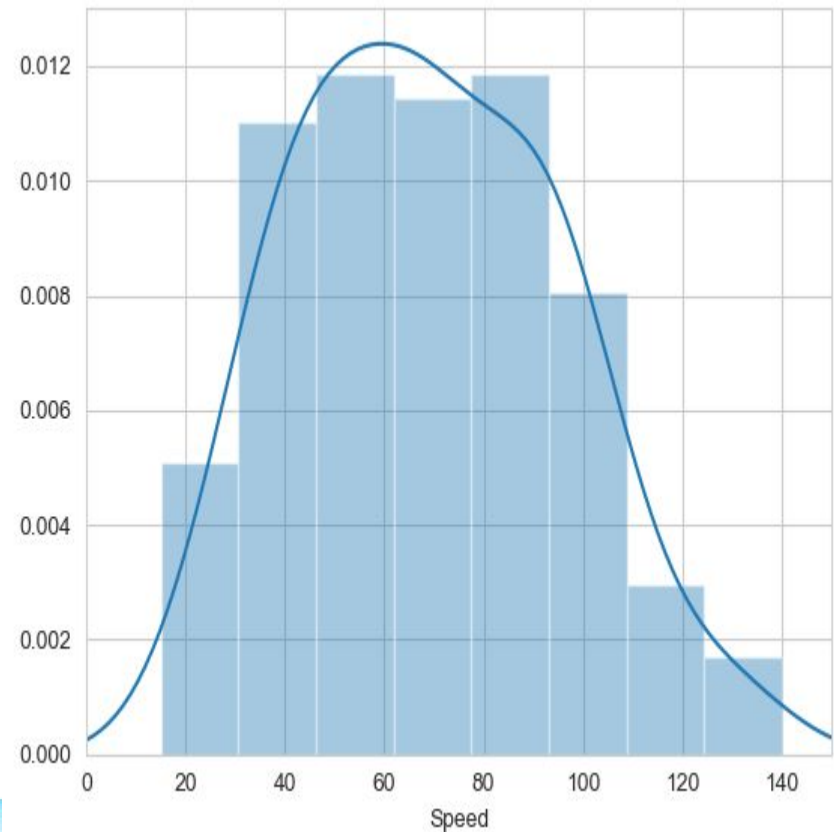
Plotting with Seaborn

One of Seaborn's greatest strengths is its diversity of plotting functions.
Most plots can be created with one line of code.
For example....

Histograms

Allow you to plot the distributions of numeric variables.

```
sns.set_style()
sns.distplot(df1.Speed)
```



Other types of graphs: Creating a scatter plot

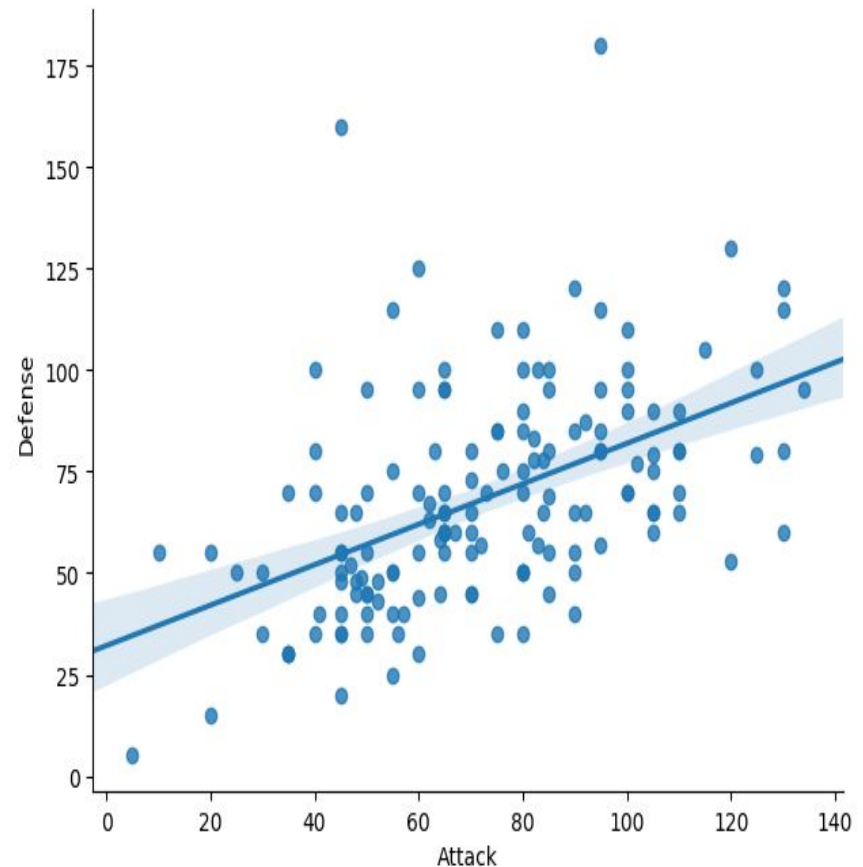
Name of variable we want on the x-axis

Name of our dataframe fed to the “data=” command

```
sns.lmplot(x='Attack', y='Defense', data=df1)
```

Seaborn “linear model plot” function for creating a scatter graph

Name of variable we want on the y-axis



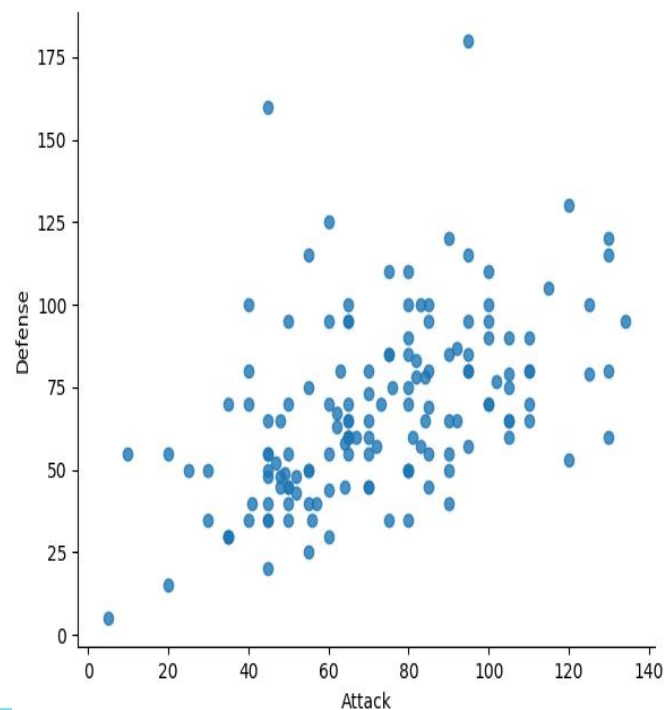
Seaborn doesn't have a dedicated scatter plot function.

We used Seaborn's function for fitting and plotting a regression line; hence `lmplot()`

However, Seaborn makes it easy to alter plots.

To remove the regression line, we use the `fit_reg=False` command

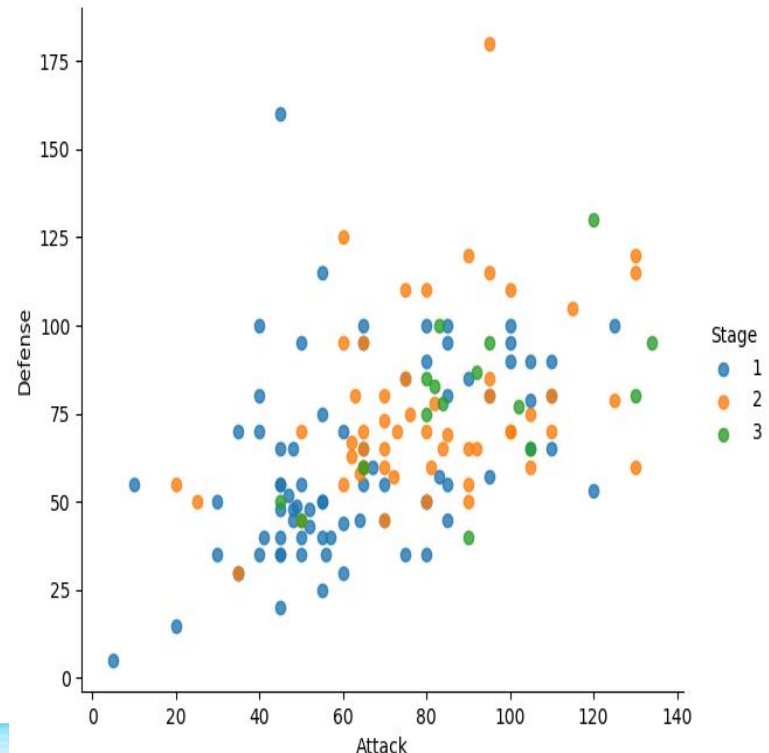
```
sns.lmplot(x='Attack', y='Defense', data=df1, fit_reg=False)
```



The hue function

Another useful function in Seaborn is the `hue` function, which enables us to use a variable to colour code our data points.

```
sns.lmplot(x='Attack', y='Defense', data=df1,  
           fit_reg=False,  
           hue='Stage')
```



Factor plots

Make it easy to separate plots by categorical classes.

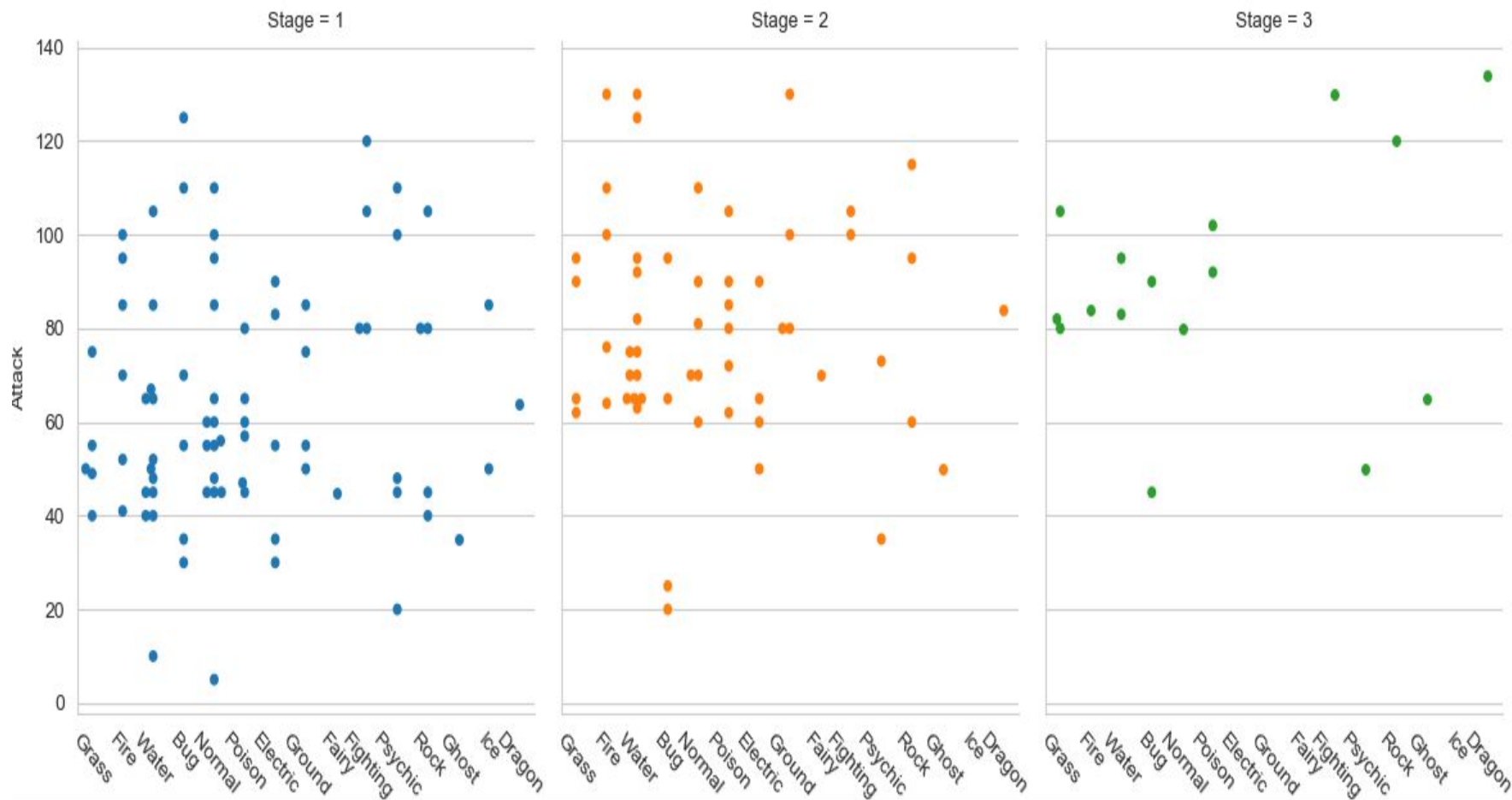
```
g = sns.factorplot(x='Type 1',  
                  y='Attack',  
                  data=df1,  
                  hue='Stage',  
                  col='Stage',  
                  kind='swarm')  
g.set_xticklabels(rotation=-45)
```

Colour by stage.

Separate by stage.

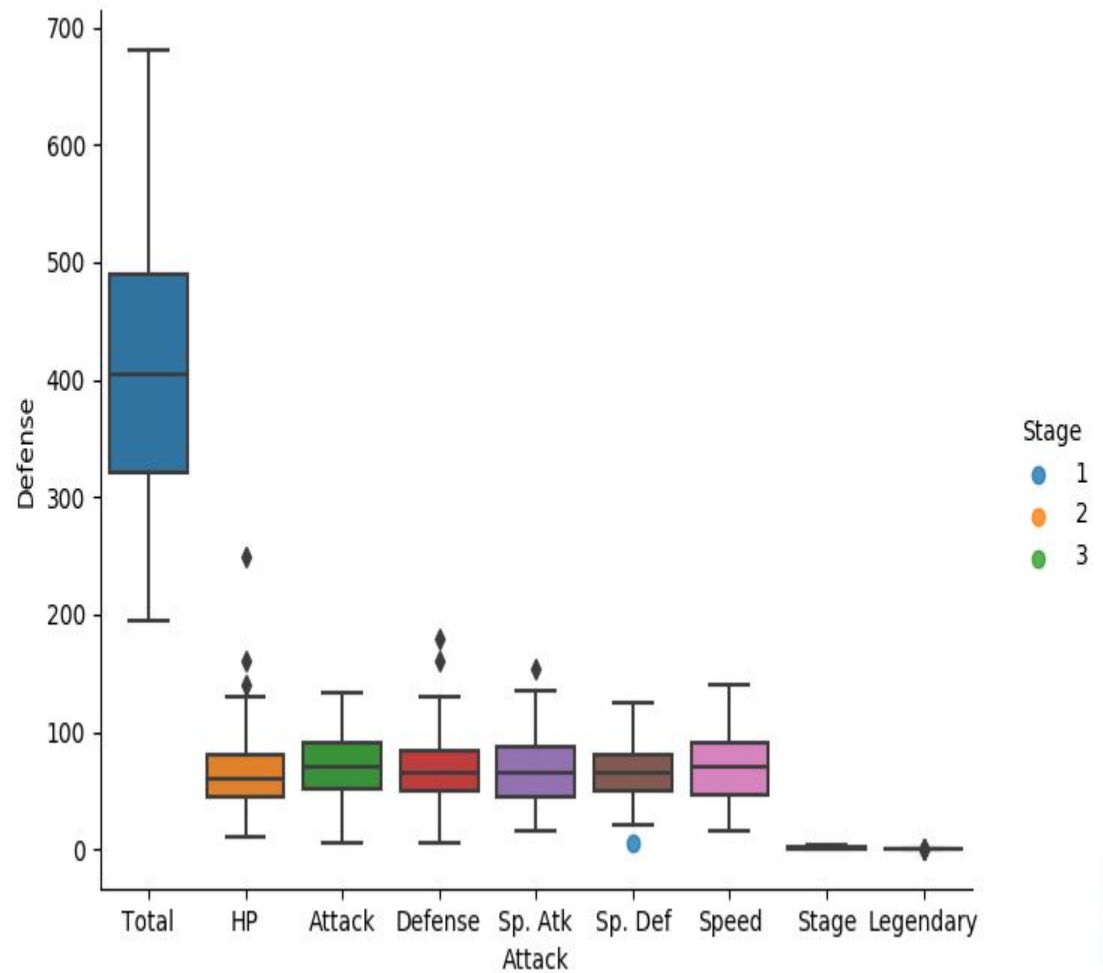
Generate using a swarmplot.

Rotate axis on x-ticks by 45 degrees.



A box plot

```
sns.boxplot(data=df1)
```

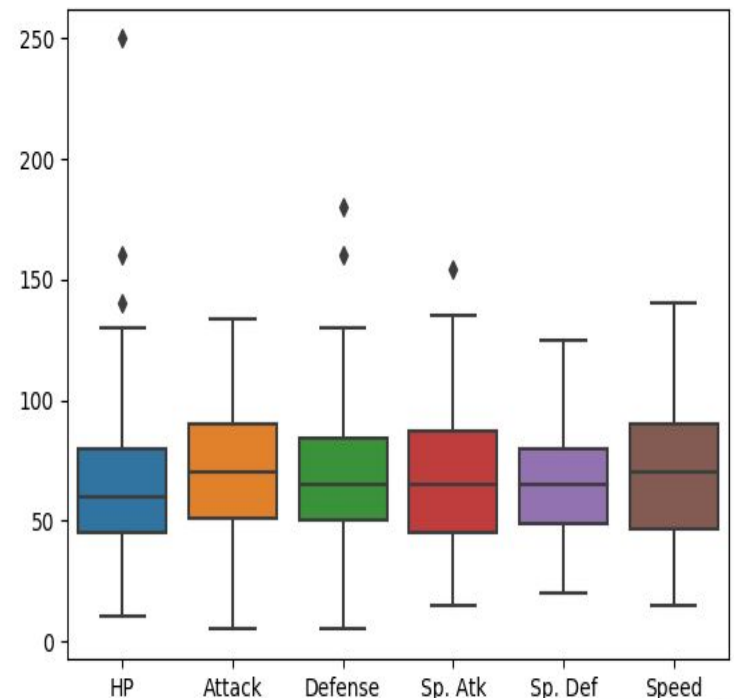


The `total`, `stage`, and `legendary` entries are not combat stats so we should remove them.

Pandas makes this easy to do, we just create a new dataframe

We just use Pandas' `.drop()` function to create a dataframe that doesn't include the variables we don't want.

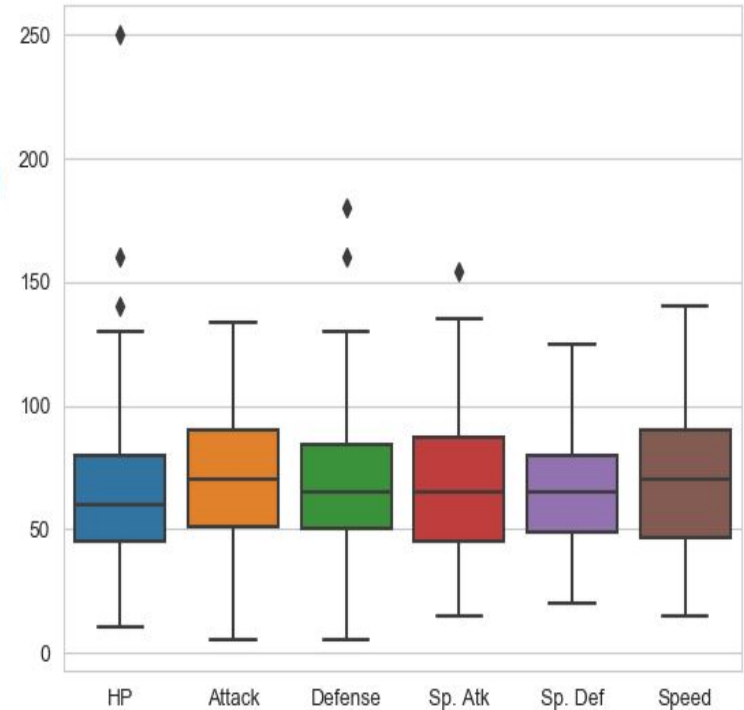
```
stats_df = df1.drop(['Total', 'Stage', 'Legendary'], axis=1)
sns.boxplot(data=stats_df)
```



Seaborn's theme

Seaborn has a number of themes you can use to alter the appearance of plots. For example, we can use “whitegrid” to add grid lines to our boxplot.

```
stats_df = df1.drop(['Total', 'Stage', 'Legendary'], axis=1)
sns.set_style('whitegrid')
sns.boxplot(data=stats_df)
```





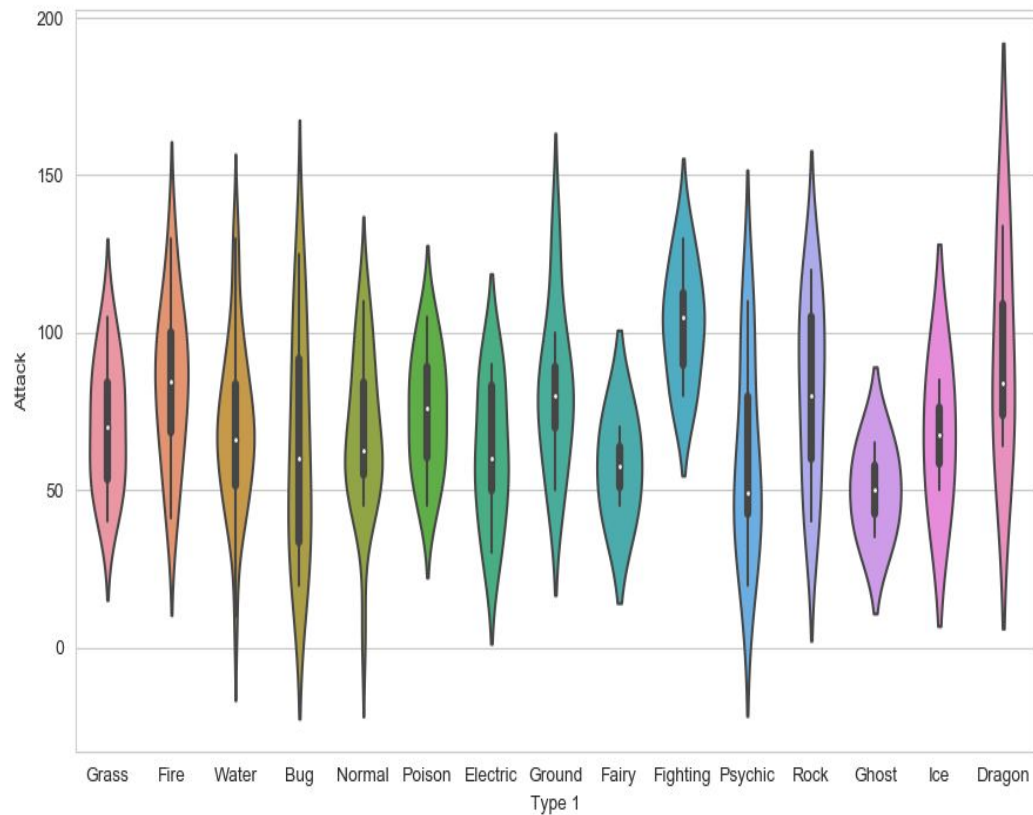
Violin plots

Violin plots are useful alternatives to box plots.

They show the distribution of a variable through the thickness of the violin.

Here, we visualise the distribution of `attack` by Pokémon's primary type:


```
sns.violinplot(x='Type 1', y='Attack', data=df1)
```



- Dragon types tend to have higher Attack stats than Ghost types, but they also have greater variance. But there is something not right here....
- The colours!

Seaborn's colour palettes

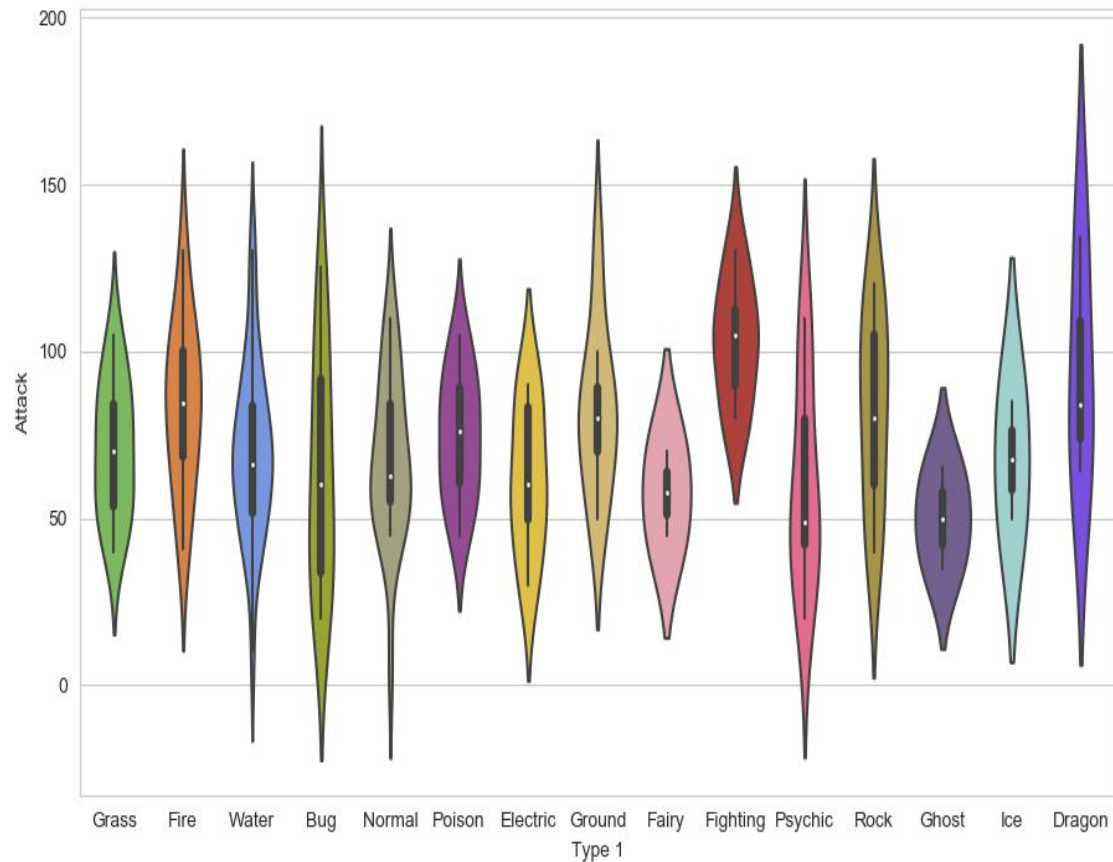
Seaborn allows us to easily set custom colour palettes by providing it with an ordered list of colour hex values.

We first create our colours list.

```
type_colors = ['#78C850', # Grass
               '#F08030', # Fire
               '#6890F0', # Water
               '#A8B820', # Bug
               '#A8A878', # Normal
               '#A040A0', # Poison
               '#F8D030', # Electric
               '#E0C068', # Ground
               '#EE99AC', # Fairy
               '#C03028', # Fighting
               '#F85888', # Psychic
               '#B8A038', # Rock
               '#705898', # Ghost
               '#98D8D8', # Ice
               '#7038F8', # Dragon
               ]
```

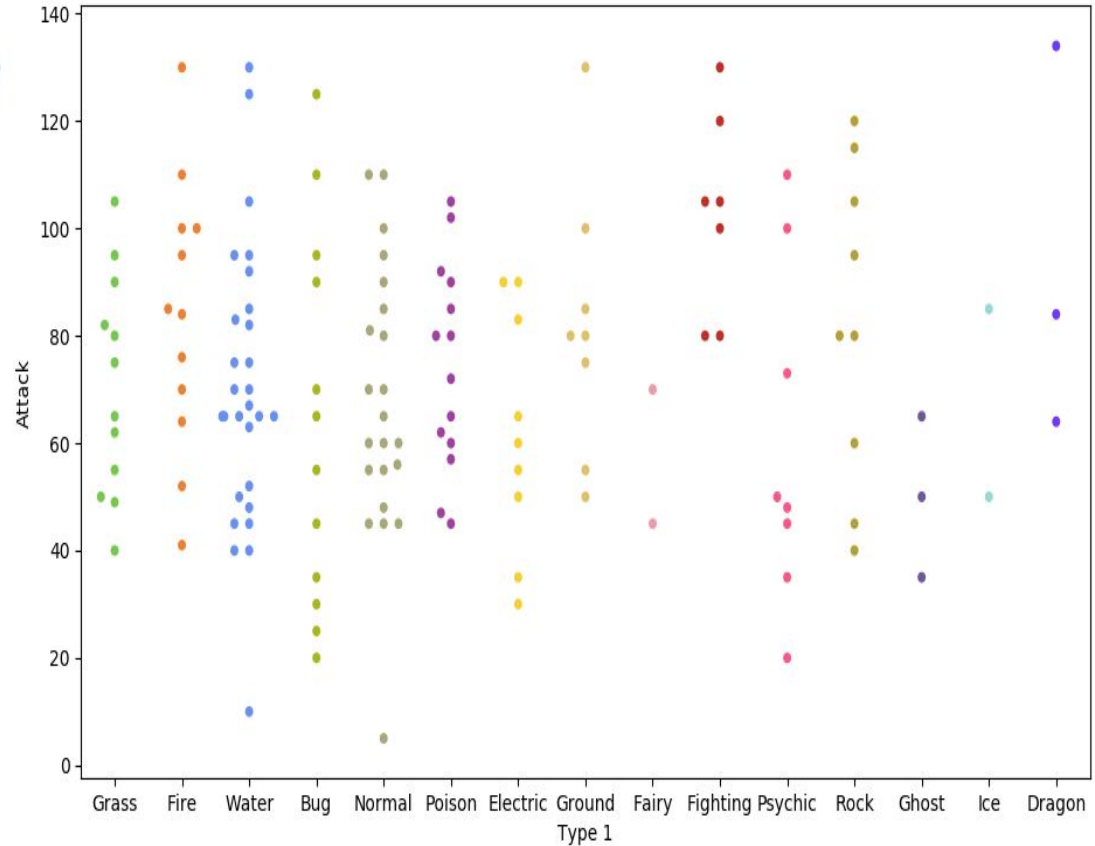
Then we just use the `palette=` function and feed in our colours list.

```
sns.violinplot(x='Type 1', y='Attack', data=df1,  
               palette=type_colors)
```



Because of the limited number of observations, we could also use a swarm plot. Here, each data point is an observation, but data points are grouped together by the variable listed on the x-axis.

```
sns.swarmplot(x='Type 1', y='Attack',  
              data=df1,  
              palette=type_colors)
```



Overlapping plots

Both of these show similar information, so it might be useful to overlap them.

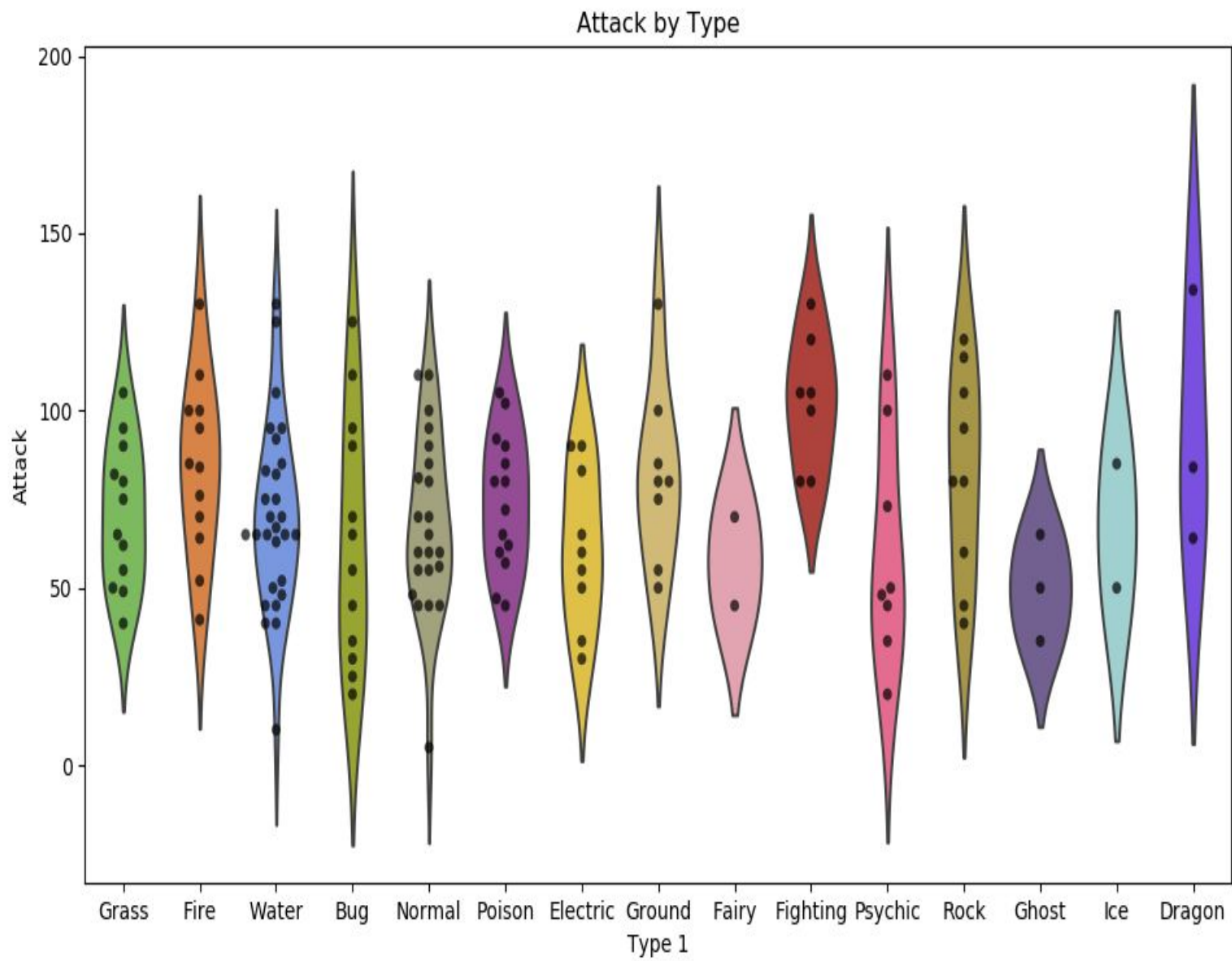
```
plt.figure(figsize=(10,6))
sns.violinplot(x='Type 1',
              y='Attack',
              data=df1,
              inner=None,
              palette=type_colors)
sns.swarmplot(x='Type 1',
              y='Attack',
              data=df1,
              color='k',
              alpha=0.7)
plt.title('Attack by Type')
```

Set size of print canvas.

Remove bars from inside the violins

Make bars black and slightly transparent

Give the graph a title



Data wrangling with Pandas

What if we wanted to create such a plot that included all of the other stats as well?
In our current dataframe, all of the variables are in different columns:

```
print(df1.head())
```

	Name	Type 1	Type 2	Total	...	Sp. Def	Speed	Stage	Legendary
#					...				
1	Bulbasaur	Grass	Poison	318	...	65	45	1	False
2	Ivysaur	Grass	Poison	405	...	80	60	2	False
3	Venusaur	Grass	Poison	525	...	100	80	3	False
4	Charmander	Fire	NaN	309	...	50	65	1	False
5	Charmeleon	Fire	NaN	405	...	65	80	2	False

If we want to visualise all stats, then we'll have to "melt" the dataframe.

```
stats_df = df1.drop(['Total', 'Stage', 'Legendary'], axis=1)
melted_df = pd.melt(stats_df,
                    id_vars=["Name", "Type 1", "Type 2"],
                    var_name="Stat")
print(melted_df.head())
```

We use the `.drop()` function again to re-create the dataframe without these three variables.

The dataframe we want to melt.

The variables to keep, all others will be melted.

A name for the new, melted, variable.

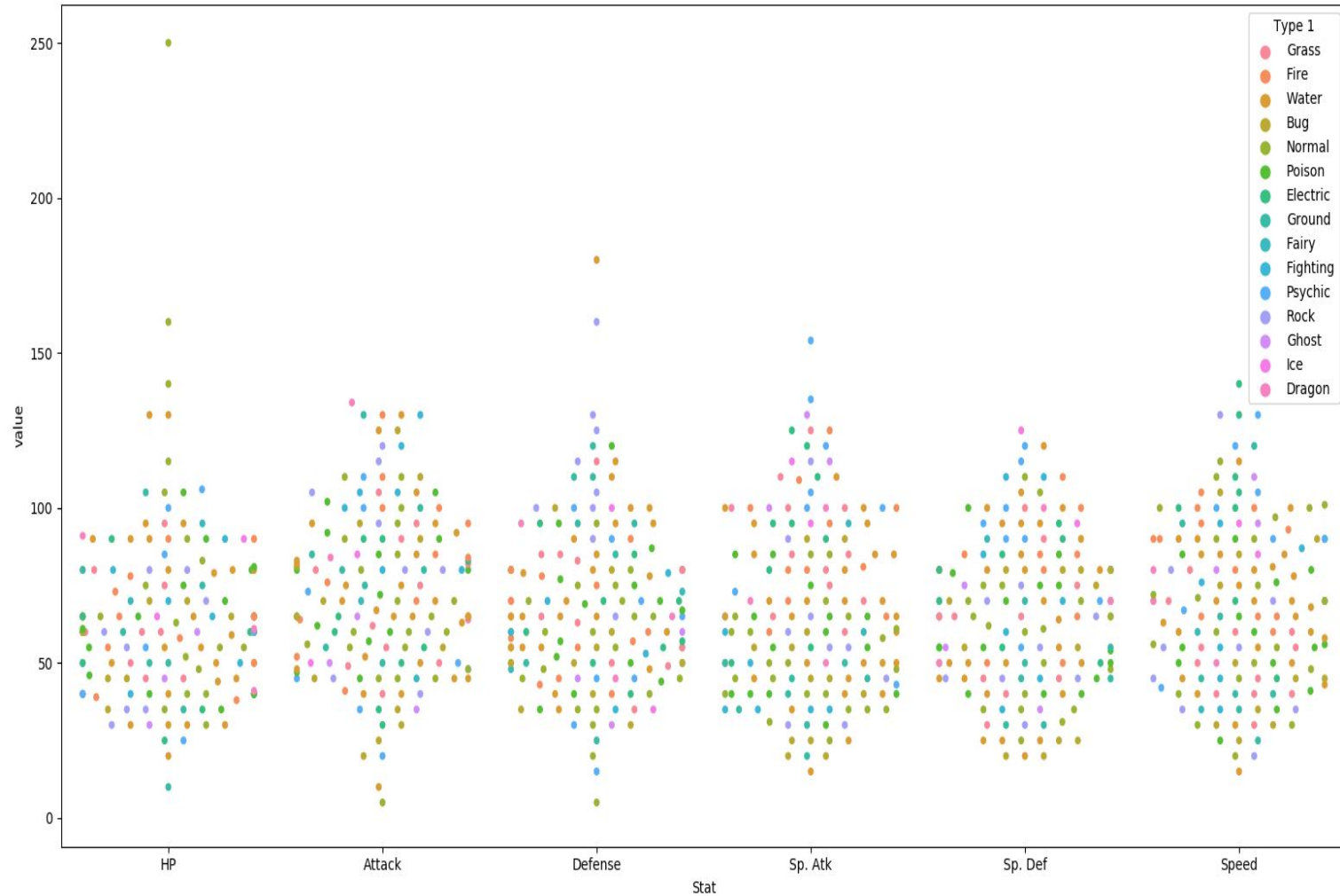
```
In [6]: runfile('C:/Users/lb690/Google Dri
pokemon_tutorial.py', wdir='C:/Users/lb690
```

	Name	Type 1	Type 2	Stat	value
0	Bulbasaur	Grass	Poison	HP	45
1	Ivysaur	Grass	Poison	HP	60
2	Venusaur	Grass	Poison	HP	80
3	Charmander	Fire	NaN	HP	39
4	Charmeleon	Fire	NaN	HP	58

- All 6 of the stat columns have been "melted" into one, and the new Stat column indicates the original stat (HP, Attack, Defense, Sp. Attack, Sp. Defense, or Speed).
- It's hard to see here, but each pokemon now has 6 rows of data; hence the `melted_df` has 6 times more rows of data.

```
print( stats_df.shape )      (151, 9)
print( melted_df.shape )    (906, 5)
```

```
sns.swarmplot(x='Stat', y='value', data=melted_df,  
             hue='Type 1')
```



This graph could be made to look nicer with a few tweaks.

```
plt.figure(figsize=(10,6))
sns.swarmplot(x='Stat',
              y='value',
              data=melted_df,
              hue='Type 1',
              split=True,
              palette=type_colors)
plt.ylim(0, 260)
plt.legend(bbox_to_anchor=(1, 1), loc=2)
```

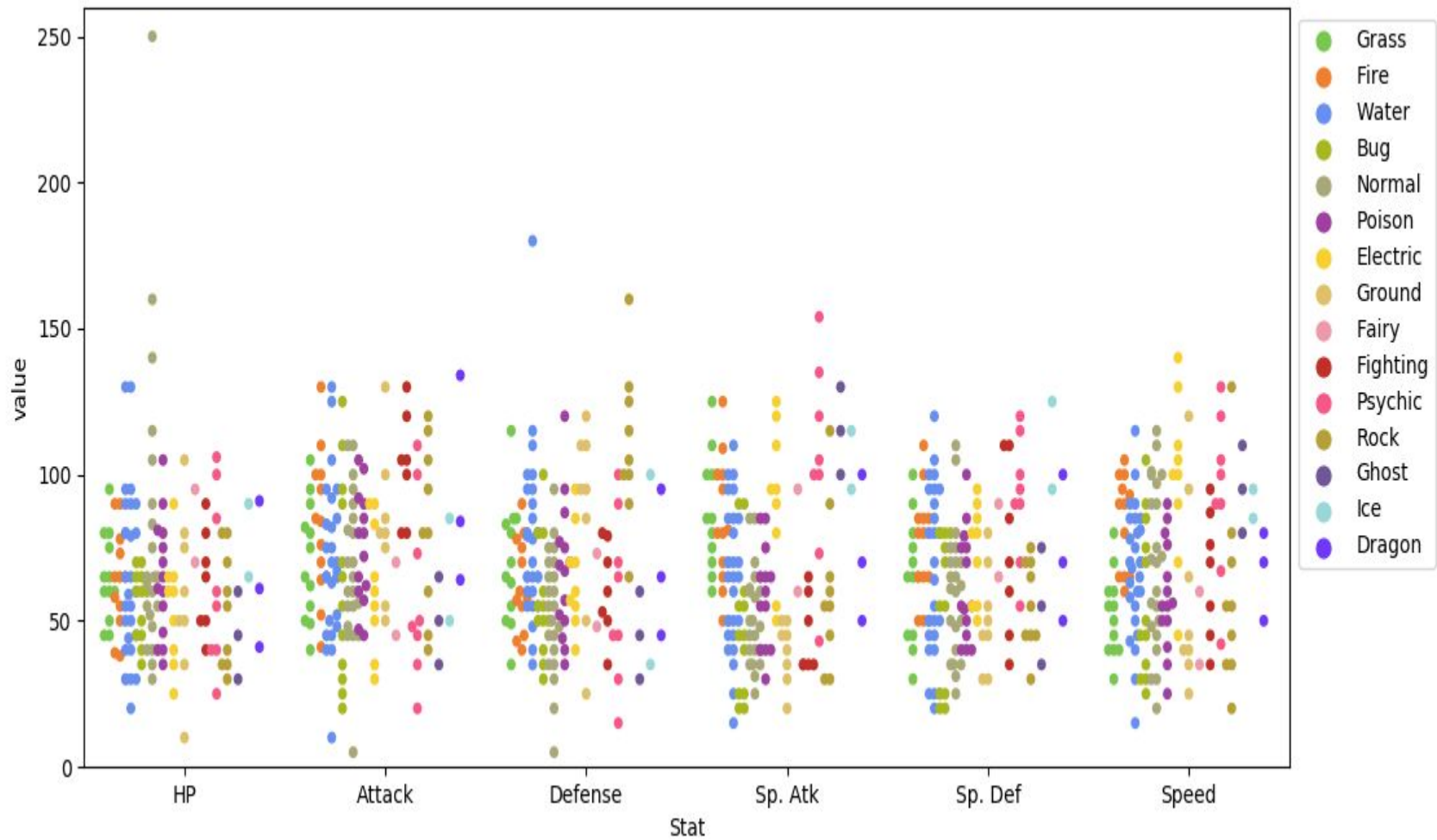
Enlarge the plot.

Separate points by hue.

Use our special Pokemon colour palette.

Adjust the y-axis.

Move the legend box outside of the graph and place to the right of it..



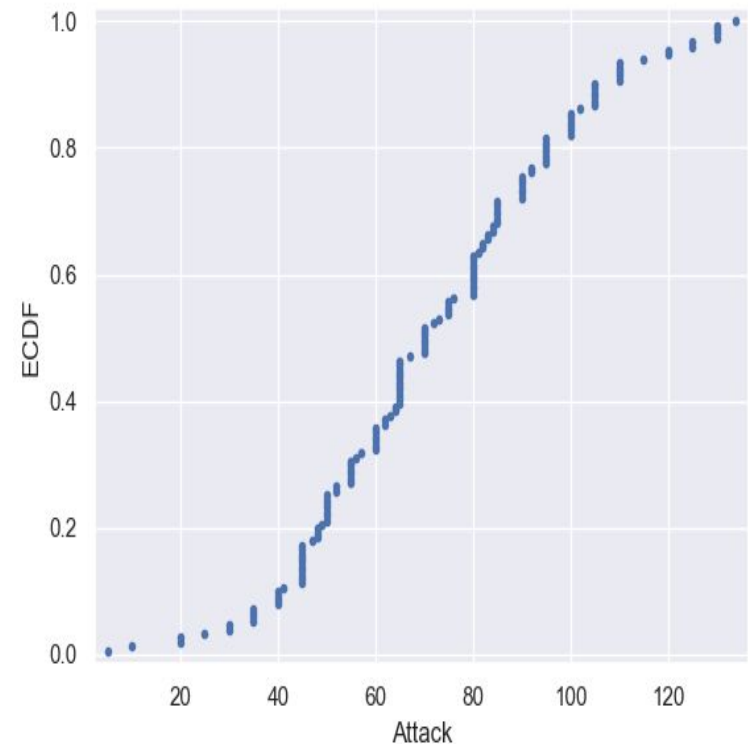
Plotting all data: Empirical cumulative distribution functions (ECDFs)

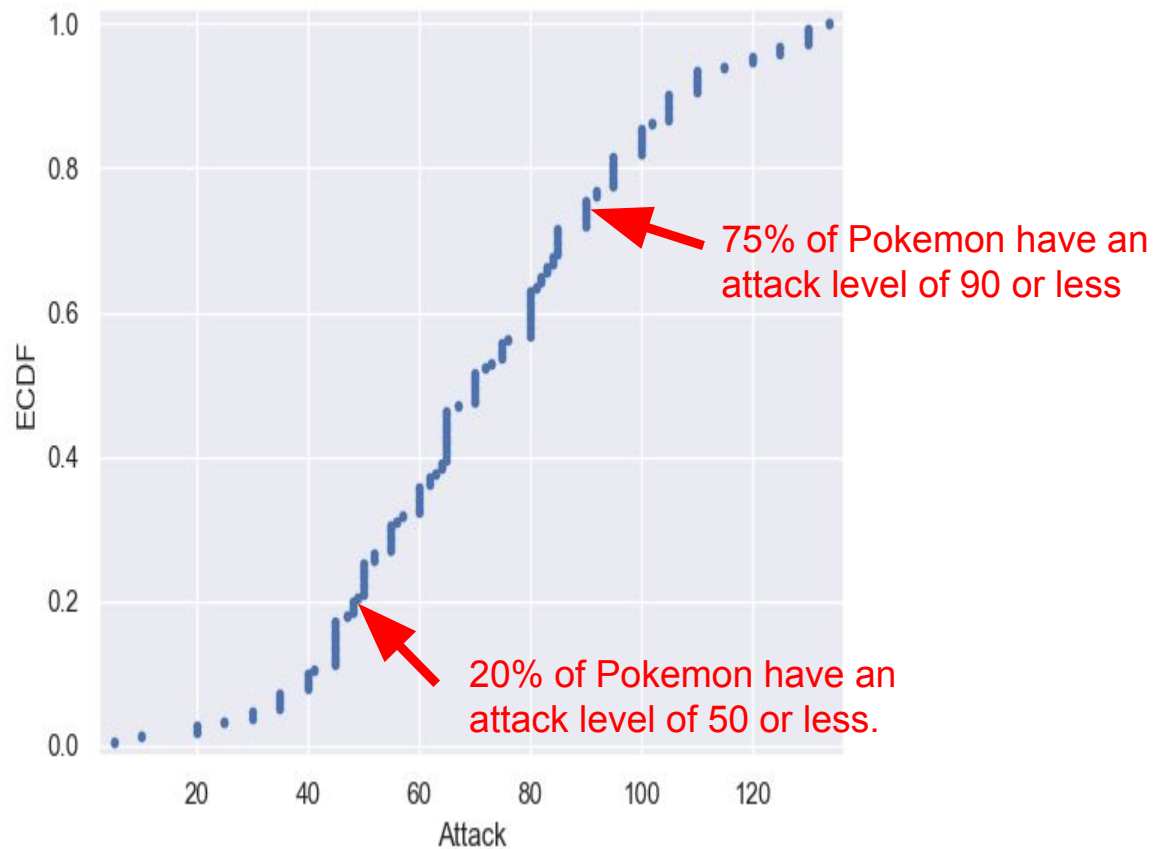
An alternative way of visualising a distribution of a variable in a large dataset is to use an ECDF.

Here we have an ECDF that shows the percentages of different attack strengths of pokemon.

An *x-value* of an ECDF is the quantity you are measuring; i.e. attacks strength.

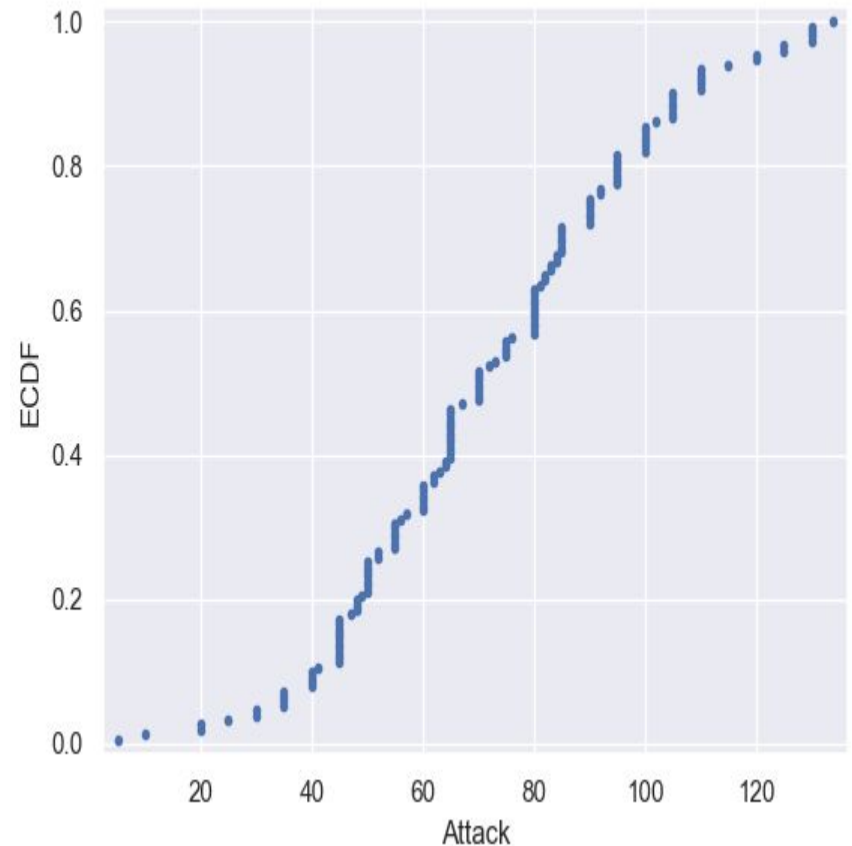
The *y-value* is the fraction of data points that have a value smaller than the corresponding x-value. For example...





Plotting an ECDF

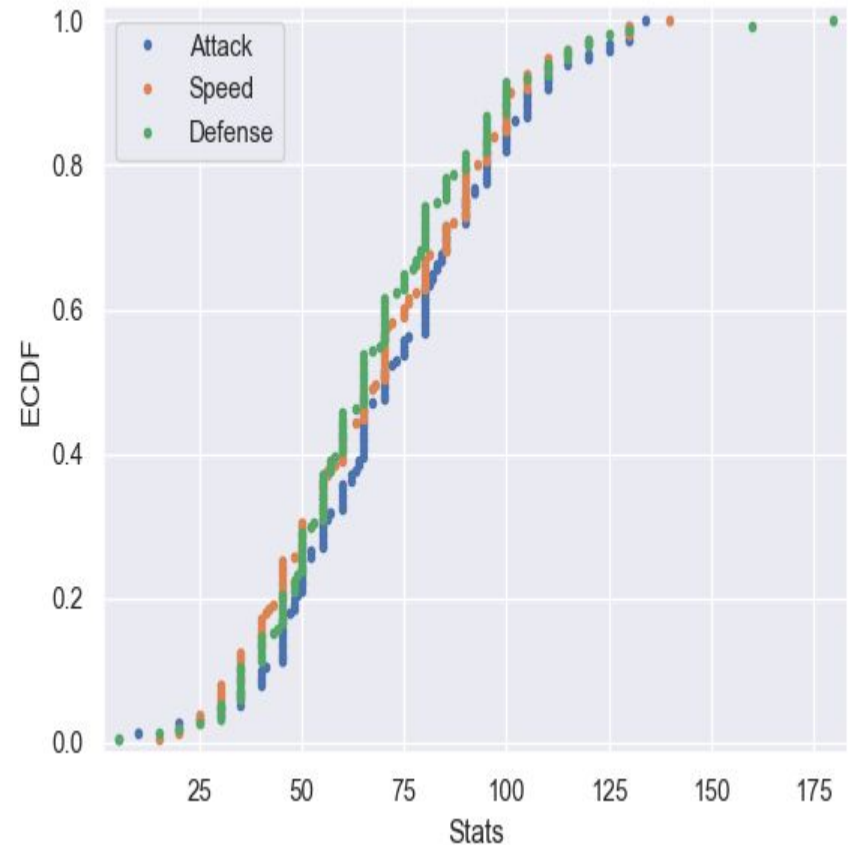
```
x = np.sort(df1['Attack'])
y=np.arange(1, len(x)+1)/len(x)
g = plt.plot(x, y, marker='.', linestyle='none')
g = plt.xlabel('Attack')
g = plt.ylabel('ECDF')
plt.margins(0.02)
plt.show()
```



You can also plot multiple ECDFs on the same plot.

As an example, here we have an ECDF for Pokemon attack, speed, and defence levels.

We can see here that defence levels tend to be a little less than the other two.





The usefulness of ECDFs

It is often quite useful to plot the ECDF first as part of your workflow.

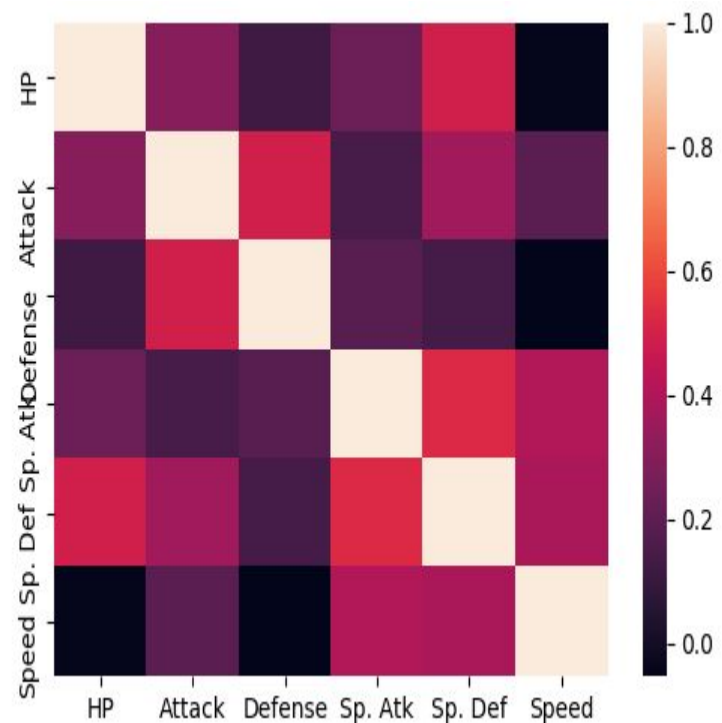
It shows all the data and gives a complete picture as to how the data are distributed.

Heatmaps

Useful for visualising matrix-like data.

Here, we'll plot the correlation of the `stats_df` variables

```
corr = stats_df.corr()  
sns.heatmap(corr)
```

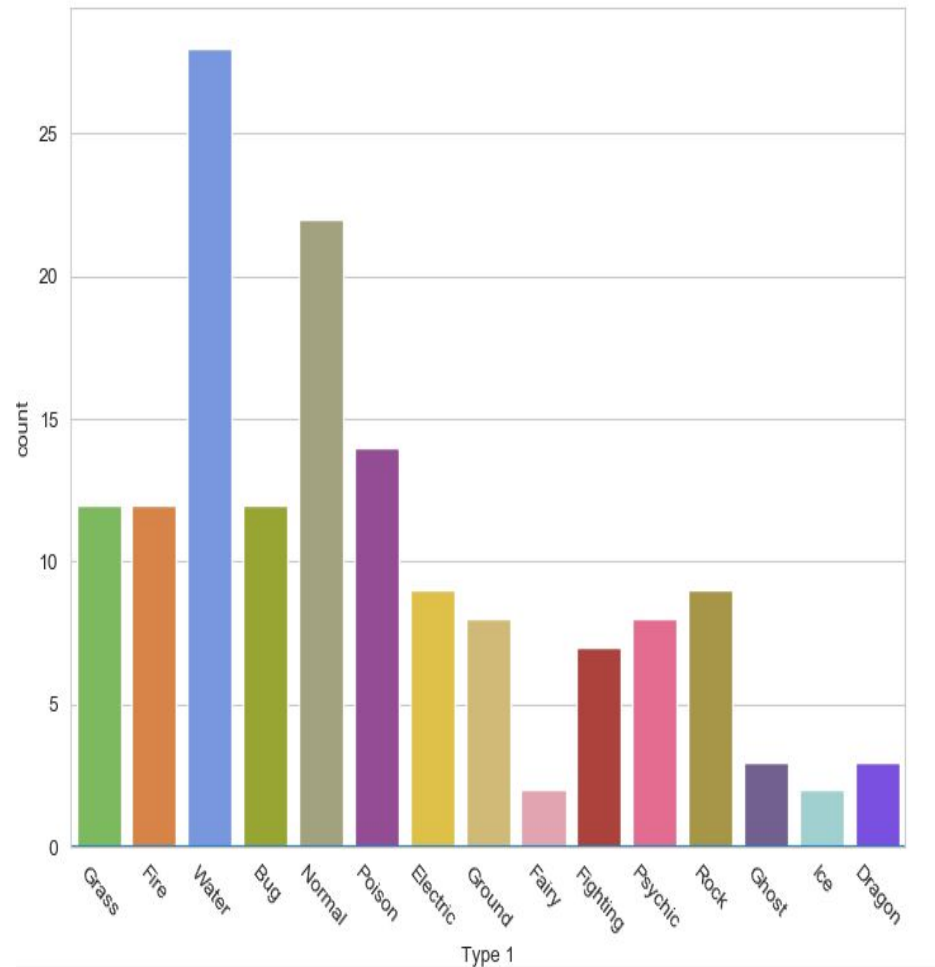


Bar plot

Visualises the distributions of categorical variables.

```
sns.countplot(x='Type 1', data=df1,  
               palette=type_colors)  
plt.xticks(rotation=-45)
```

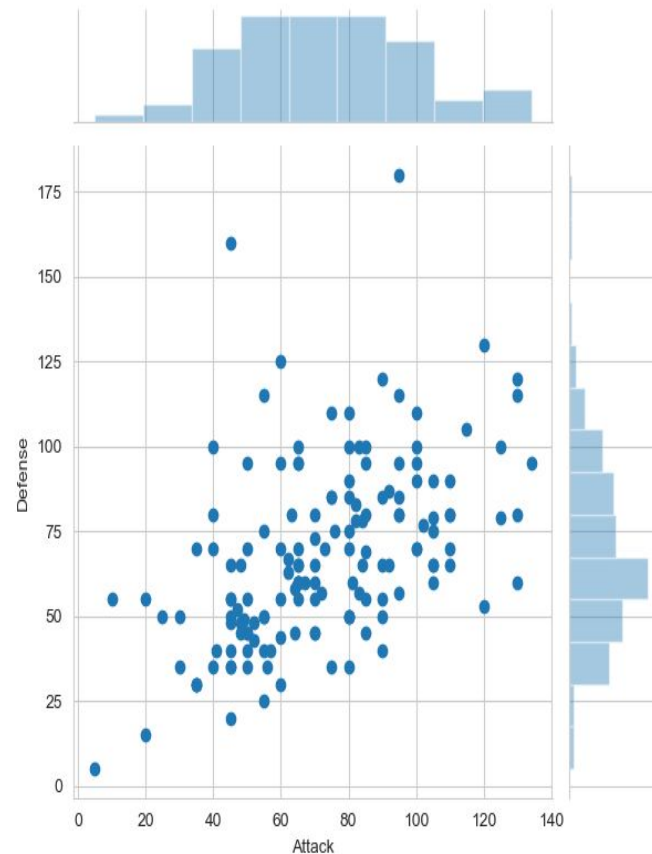
Rotates the
x-ticks 45
degrees



Joint Distribution Plot

Joint distribution plots combine information from scatter plots and histograms to give you detailed information for bi-variate distributions.

```
sns.jointplot(x='Attack',  
              y='Defense',  
              data=df1)
```





HKUSPACE
香港大學專業進修學院
HKU School of Professional and Continuing Education

THANK YOU

