

Business Process Automation with VBA and Python

Mr. Eddie Chow / 25 January 2025



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 to Data collection and Cleaning
2. ETL Data Processing
3. Machine Learning Framework
4. Introduction to Data Visualization
5. Data Visualization Tools and Technologies
6. Drawing Insights from Visualized Data
7. Best practices for presenting data visualizations

Data Collection - Reading API

Case Study - Read Weather Forecasting from HKO API

Step 1. Import the required libraries:

```
#import necessary library
import requests
import pandas as pd
```

Step 2: Load the weather forecasting data from '<https://data.weather.gov.hk/weatherAPI/opendata/>'

```
# data dict is saved in https://www.hko.gov.hk/en/weatherAPI/doc/files/HKO\_Open\_Data\_API\_Documentation.pdf
# Parameters for the API
datatype = 'fnd'
lang = 'en'
apiurl = f'https://data.weather.gov.hk/weatherAPI/opendata/weather.php?dataType={datatype}&lang={lang}'
```

```
# Fetching the JSON data
response = requests.get(apiurl)
response.raise_for_status() # Check for request errors
```

```
# Converting JSON to a DataFrame
json_data = response.json()
```

```
# Convert JSON data to DataFrame
weatherforecast_df = pd.DataFrame(json_data['weatherForecast'])
```

```
# Display the DataFrame
print(weatherforecast_df.head())
```

```
forecastDate      week          forecastWind \
0   20250115 Wednesday      North force 4 to 5.
1   20250116 Thursday     North to northeast force 4, force 5 at first.
2   20250117 Friday       Northeast force 4, east force 5 later.
3   20250118 Saturday    East to northeast force 4 to 5, occasionally f...
4   20250119 Sunday        East to northeast force 2 to 3.

                           forecastWeather \
0   Mainly cloudy with a few light rain patches at...
1   Fine. Rather cool in the morning. Very dry dur...
2   Fine. Rather cool in the morning. Very dry dur...
3   Dry with sunny periods.
4   Mainly fine and dry.

           forecastMaxtemp      forecastMintemp \
0   {'value': 20, 'unit': 'C'}  {'value': 15, 'unit': 'C'}
1   {'value': 19, 'unit': 'C'}  {'value': 13, 'unit': 'C'}
2   {'value': 19, 'unit': 'C'}  {'value': 13, 'unit': 'C'}
3   {'value': 20, 'unit': 'C'}  {'value': 15, 'unit': 'C'}
4   {'value': 21, 'unit': 'C'}  {'value': 16, 'unit': 'C'

                           forecastMaxrh      forecastMinrh \
0   {'value': 80, 'unit': 'percent'}  {'value': 40, 'unit': 'percent'}
1   {'value': 65, 'unit': 'percent'}  {'value': 35, 'unit': 'percent'}
2   {'value': 65, 'unit': 'percent'}  {'value': 30, 'unit': 'percent'}
3   {'value': 70, 'unit': 'percent'}  {'value': 40, 'unit': 'percent'}
4   {'value': 70, 'unit': 'percent'}  {'value': 40, 'unit': 'percent'

      ForecastIcon  PSR
0         51  Low
1         92  Low
2         50  Low
3         51  Low
4         50  Low
```

Data Collection - Reading API

#Continue from previous page

Step 3. Saving Weather Forecasting DataFrame to a CSV

```
filecsv_file = 'weatherForecast.csv' # Desired output CSV file  
nameweatherforcast_df.to_csv(csv_file, index=False)  
  
print(f'Data has been successfully saved to {csv_file}')
```

A	B	C	D	E	F	G	H	I	J
forecastDate	week	forecastWind	forecastWeather	forecastMaxtemp	forecastMintemp	forecastMaxrh	forecastMinrh	ForecastIcon	PSR
20250115	Wednesday	North force 4 to 5.	Mainly cloudy with a few showers.	{'value': 20, 'unit': 'C'} {'value': 15, 'unit': 'C'}	{'value': 80, 'unit': '%'} {'value': 40, 'unit': '%'}	{'value': 80, 'unit': '%'} {'value': 65, 'unit': '%'}	{'value': 35, 'unit': '%'} {'value': 30, 'unit': '%'}	51	Low
20250116	Thursday	North to northeast force 4.	Fine. Rather cool in the morning.	{'value': 19, 'unit': 'C'} {'value': 13, 'unit': 'C'}	{'value': 65, 'unit': '%'} {'value': 35, 'unit': '%'}	{'value': 65, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 50, 'unit': '%'} {'value': 30, 'unit': '%'}	92	Low
20250117	Friday	Northeast force 4, east 5.	Fine. Rather cool in the morning.	{'value': 19, 'unit': 'C'} {'value': 13, 'unit': 'C'}	{'value': 65, 'unit': '%'} {'value': 35, 'unit': '%'}	{'value': 65, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 50, 'unit': '%'} {'value': 30, 'unit': '%'}	50	Low
20250118	Saturday	East to northeast force 4.	Dry with sunny periods.	{'value': 20, 'unit': 'C'} {'value': 15, 'unit': 'C'}	{'value': 70, 'unit': '%'} {'value': 40, 'unit': '%'}	{'value': 70, 'unit': '%'} {'value': 51, 'unit': '%'}	{'value': 40, 'unit': '%'} {'value': 30, 'unit': '%'}	51	Low
20250119	Sunday	East to northeast force 4.	Mainly fine and dry.	{'value': 21, 'unit': 'C'} {'value': 16, 'unit': 'C'}	{'value': 70, 'unit': '%'} {'value': 40, 'unit': '%'}	{'value': 70, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 40, 'unit': '%'} {'value': 30, 'unit': '%'}	50	Low
20250120	Monday	East to northeast force 4.	Fine and dry.	{'value': 22, 'unit': 'C'} {'value': 16, 'unit': 'C'}	{'value': 70, 'unit': '%'} {'value': 40, 'unit': '%'}	{'value': 70, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 40, 'unit': '%'} {'value': 30, 'unit': '%'}	50	Low
20250121	Tuesday	East to northeast force 4.	Fine and dry.	{'value': 22, 'unit': 'C'} {'value': 17, 'unit': 'C'}	{'value': 70, 'unit': '%'} {'value': 40, 'unit': '%'}	{'value': 70, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 40, 'unit': '%'} {'value': 30, 'unit': '%'}	50	Low
20250122	Wednesday	East force 4, occasional rain.	Sunny periods.	{'value': 21, 'unit': 'C'} {'value': 17, 'unit': 'C'}	{'value': 75, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 75, 'unit': '%'} {'value': 51, 'unit': '%'}	{'value': 50, 'unit': '%'} {'value': 30, 'unit': '%'}	51	Low
20250123	Thursday	East force 4, occasional rain.	Sunny intervals.	{'value': 21, 'unit': 'C'} {'value': 17, 'unit': 'C'}	{'value': 75, 'unit': '%'} {'value': 50, 'unit': '%'}	{'value': 75, 'unit': '%'} {'value': 52, 'unit': '%'}	{'value': 50, 'unit': '%'} {'value': 30, 'unit': '%'}	52	Low

Combining several CSV File

This Tasks is to combining Building Information and School Net Numbers

Step 1. Import the required libraries:

```
#Loading orginal CSV file containing property data:
```

```
hkbuildingdata = pd.read_csv('hkbuildinginfo.csv', encoding='ISO-8859-1')
```

```
hkbuildingdata.head(5)
```

Step 2: #Clean up the data

```
# Check for missing values
```

```
print("\nMissing values in each column:")
```

```
print(hkbuildingdata.isnull().sum())
```

```
#Handle missing values by Dropping rows with missing values
```

```
# For columns with a high percentage of missing values, consider dropping or filling them
```

```
# Here we will drop 'block' and 'state' columns due to excessive missing values
```

```
hkbuildingdata.drop(columns=['withpre', 'rootid', 'fatherid', 'catid', 'source', 'block', 'state', 'date_dm'],  
    inplace=True, errors='ignore')
```

```
# Assuming 'floor' is the column causing the error
```

```
hkbuildingdata['floor'] = hkbuildingdata['floor'].astype(str) # Convert to string type
```

```
# Now you can safely use the .str accessor
```

```
hkbuildingdata['floor'] = hkbuildingdata['floor'].str.replace(r'^\d', "", regex=True) # Remove  
non-numeric characters
```

```
# Convert back to numeric if needed
```

```
hkbuildingdata['floor'] = pd.to_numeric(hkbuildingdata['floor'], errors='coerce') # Convert to numeric,  
coercing errors to NaN
```

	Missing values in each column:
date	0
withpre	23564
id	0
rootid	0
fatherid	0
catid	0
catname	0
catfathername	0
url_father	0
url_cat	0
source	11556
contract	0
memo	0
price	0
price_value	0
holddate	0
winloss_flag	0
winloss	0
act_area	0
area	0
arearaw	4
sq_price	0
sq_price_value	0
sq_actprice	0
sq_actprice_value	0
month	0
day	0
date_dm	0
date_y	0
block	32308
state	77003
floor	32
room	258
addr	0
Unnamed: 34	80000
	dtype: int64

Combining several CSV File

#Continue from previous page

Continue with your existing code for filling missing values and other operations

```
hkbuildingdata['floor'] = hkbuildingdata['floor'].fillna(hkbuildingdata['floor'].median()) # Fill with median for numeric data
```

Step 4: Fill missing values for 'floor' and 'room'

```
hkbuildingdata['floor'] = hkbuildingdata['floor'].fillna(hkbuildingdata['floor'].median()) # Fill with median for numeric data
```

```
hkbuildingdata['room'] = hkbuildingdata['room'].fillna(hkbuildingdata['room'].mode()[0]) # Fill with mode for categorical data
```

Step 5: Fill missing values for 'arearaw'

```
hkbuildingdata['arearaw'] = hkbuildingdata['arearaw'].fillna(hkbuildingdata['arearaw'].mean())
```

Step 6: Rename columns to more meaningful names

```
hkbuildingdata.rename(columns={'floor': 'Floor', 'arearaw': 'House Area', 'sq_price_value': 'Price Per Square', 'catname': 'Building Name', 'catfathername': 'District'}, inplace=True)
```

Step 7: Display the cleaned DataFrame and check for remaining missing values

```
print("\nCleaned DataFrame:")
```

```
print(hkbuildingdata.head())
```

```
print("\nRemaining Missing Values:")
```

```
print(hkbuildingdata.isnull().sum())
```

Cleaned DataFrame:								
	date	id	Building Name	District	url_cat	contract	memo	price
0	2020-11-27	683333	Bel Air Heights	Diamond Hill	https://data.28hse.com/en/k1/diamond-hill	2058...	\$10M	\$15221
1	2020-11-27	683332	Fa Yuen Plaza	Mong Kok	https://data.28hse.com/en/k1/mong-kok/2712_fa...	309...	\$4.28M	\$20000
2	2020-11-27	683331	Caldecott Hill	Yau Yat Tsuen	https://data.28hse.com/en/k1/yau-yat-tsuen/309...	5035...	\$11000	\$11000
3	2020-11-27	683330	Pang Ching Court	Wong Tai Sin	https://data.28hse.com/en/k1/wong-tai-sin/5035...	8404.80	\$4.9M	\$17354
4	2020-11-27	683329	Metro Harbour View	Tai Kok Tsui	https://data.28hse.com/en/k1/tai-kok-tsui/2564...	12016.81	\$7.15M	\$12016.81
					sq_actprice	sq_actprice_value	month	day
0	15220.70	Nov	27	2020	2.0	E BLOCK 2	2#/F	Room E
1	20000.00	Nov	27	2020	1.0	C	1#/F	Room C
2	11099.90	Nov	27	2020	7.0	E TOWER 1	7#/F	Room E
3	0.00	Nov	27	2020	1.0	22	1#/F	Room 22
4	17354.37	Nov	27	2020	1.0	B BLOCK 8	1#/F	Room B

Combining Building Information and School Net Numbers

#Combine Original Data with School District Data

```
schoolnetnumber_df = pd.read_csv("Primary Sch_Net Numbers By District.csv")
```

```
# Normalize the 'District' column in both DataFrames to lowercase
```

```
hkbuildingdata['District'] = hkbuildingdata['District'].str.lower()
```

```
schoolnetnumber_df['District'] = schoolnetnumber_df['District Areas'].str.lower()
```

```
# Merge schooldistrict_df with districtcode_df on the normalized 'District' column
```

```
merged_df = hkbuildingdata.merge(schoolnetnumber_df[['District', 'Sch_Net_No']], on='District', how='left')
```

```
# Create the 'District_Code' column based on the logic provided
```

```
hkbuildingdata['School_Netnumber'] = merged_df['Sch_Net_No'].fillna(0).astype(int)
```

```
hkbuildingdata
```

[27]:	date	id	Building Name	District	url_father	url_cat	contract	memo	price	price_va
0	2020-11-27	683333	Bel Air Heights	diamond hill	https://data.28hse.com/en/kl/diamond-hill	https://data.28hse.com/en/kl/diamond-hill/2058...	Agreement	20112702480015	\$10M	100000
1	2020-11-27	683332	Fa Yuen Plaza	mong kok	https://data.28hse.com/en/kl/mong-kok	https://data.28hse.com/en/kl/mong-kok/2712_fa...	Agreement	20112702470044	\$4.28M	428000
2	2020-11-27	683331	Caldecott Hill	yau yat tsuen	https://data.28hse.com/en/kl/yau-yat-tsuen	https://data.28hse.com/en/kl/yau-yat-tsuen/309...	Agreement	20112702440018	\$11M	110000
3	2020-11-27	683330	Pang Ching Court	wong tai sin	https://data.28hse.com/en/kl/wong-tai-sin	https://data.28hse.com/en/kl/wong-tai-sin/5035...	Agreement	20112702410017	\$4.9M	490000
4	2020-11-27	683329	Metro Harbour View	tai kok tsui	https://data.28hse.com/en/kl/tai-kok-tsui	https://data.28hse.com/en/kl/tai-kok-tsui/2564...	Agreement	20112702400032	\$7.15M	715000

5 rows × 29 columns

Exploratory Data Analysis(ETL) and Data Visualization

Exploratory Data Analysis and Visualization

```
#filter the data with period "2020-1-1" to "2020-12-31"
```

```
# Create a full date column from 'month', 'day', and 'date_y'
```

```
hkbuildingdata['full_date'] = pd.to_datetime(hkbuildingdata['month'] + ' ' + hkbuildingdata['day'].astype(str) + ' ' +  
    hkbuildingdata['date_y'].astype(str))
```

```
# Filter the DataFrame for the date range "2020-01-01" to "2020-12-31"
```

```
start_date = '2020-01-01'
```

```
end_date = '2020-12-31'
```

```
covidfiltered_data = hkbuildingdata[(hkbuildingdata['full_date'] >= start_date) & (hkbuildingdata['full_date'] <= end_date)]
```

```
# Display the first few rows of the filtered DataFrame
```

```
print(covidfiltered_data.head(5))
```

	date	id	Building Name	District	\
0	2020-11-27	683333	Bel Air Heights	diamond hill	
1	2020-11-27	683332	Fa Yuen Plaza	mong kok	
2	2020-11-27	683331	Caldecott Hill	yau yat tsuen	
3	2020-11-27	683330	Pang Ching Court	wong tai sin	
4	2020-11-27	683329	Metro Harbour View	tai kok tsui	

```
# Filter the DataFrame for the date range "2019-
```

```
start_date = '2019-01-01'
```

```
end_date = '2019-12-31'
```

```
filtered_data = hkbuildingdata[(hkbuildingdata['fu
```

	url_father	\
0	https://data.28hse.com/en/k1/diamond-hill	
1	https://data.28hse.com/en/k1/mong-kok	
2	https://data.28hse.com/en/k1/yau-yat-tsuen	
3	https://data.28hse.com/en/k1/wong-tai-sin	
4	https://data.28hse.com/en/k1/tai-kok-tsui	

```
# Display the first few rows of the filtered DataFr
```

```
print(filtered_data.head(5))
```

	url_cat	contract	\
0	https://data.28hse.com/en/k1/diamond-hill/2058...	Agreement	
1	https://data.28hse.com/en/k1/mong-kok/2712_fa...	Agreement	
2	https://data.28hse.com/en/k1/yau-yat-tsuen/309...	Agreement	
3	https://data.28hse.com/en/k1/wong-tai-sin/5035...	Agreement	
4	https://data.28hse.com/en/k1/tai-kok-tsui/2564...	Agreement	

Exploratory Data Analysis(ETL) and Visualization

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a scatter plot
plt.figure(figsize=(10, 6))

# Plot for 2019
plt.scatter(filtered_data['full_date'], filtered_data['Price Per Square'], color='blue', label='2019', alpha=0.6)

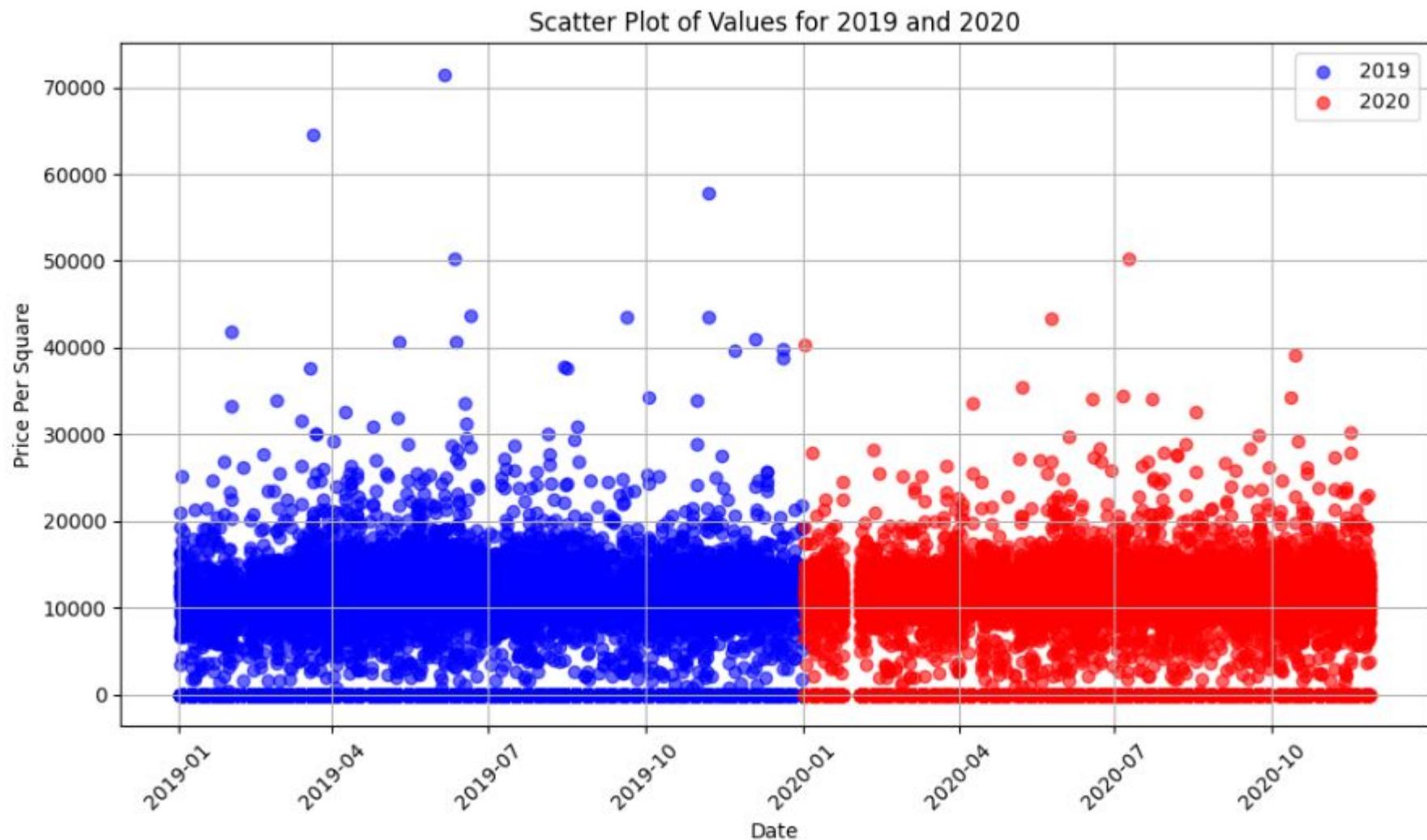
# Plot for 2020
plt.scatter(covidfiltered_data['full_date'], covidfiltered_data['Price Per Square'], color='red', label='2020', alpha=0.6)

# Adding titles and labels
plt.title('Scatter Plot of Values for 2019 and 2020')
plt.xlabel('Date')
plt.ylabel('Price Per Square')
plt.xticks(rotation=45)
plt.legend()
plt.grid()

# Show the plot
plt.tight_layout()
plt.show()
```



Exploratory Data Analysis(ETL) and Visualization



Business Process Automation with Python

Practice 4 – File Processing with DATA.GOV.HK data

- Read the CSV file
- Loop through the data and do some counting



A	B	C	D	E
Account	Expenditure Group	Subhead	Item	Actual(\$)
1 RECURRENT	PERSONNEL EMOLUMENTS	SALARIES	SALARIES	151,760,978
2 RECURRENT	PERSONNEL EMOLUMENTS	ALLOWANCES	OVERTIME	507,492
4 RECURRENT	PERSONNEL EMOLUMENTS	ALLOWANCES	ACTING ALLOWANCES	5,636,585
5 RECURRENT	PERSONNEL RELATED EXPENSES	MANDATORY PROVIDENT FUND CONTRIBUTION	CONTRIBUTION FOR RECRUITS UNDER THE NEW TERMS BEFORE CONFIRMATION	265,855
6 RECURRENT	PERSONNEL RELATED EXPENSES	CIVIL SERVICE PROVIDENT FUND CONTRIBUTION	MANDATORY CONTRIBUTION UNDER THE MANDATORY PROVIDENT FUND SCHEMES ORDINANCE	1,468,649
7 RECURRENT	PERSONNEL RELATED EXPENSES	CIVIL SERVICE PROVIDENT FUND CONTRIBUTION	GOVERNMENT VOLUNTARY CONTRIBUTION	9,534,636
8 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	CLOTHES AND UNIFORMS	21,940
9 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	CLEANING MATERIALS	107,532
10 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	OFFICE STATIONERY AND MATERIALS	672,661
11 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	GENERAL PUBLICATIONS, PERIODICALS AND JOURNALS	123,429
12 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	OFFICE FURNITURE AND EQUIPMENT (INCLUDING RENTAL)	445,709
13 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	PAPERS	53,221
14 RECURRENT	DEPARTMENTAL EXPENSES	STORES AND EQUIPMENT	INFORMATION AND COMMUNICATIONS TECHNOLOGY	3,299,560
15 RECURRENT	DEPARTMENTAL EXPENSES	LIGHT AND POWER	ELECTRICITY	422,305

- Starting notebook:

https://github.com/innoviai/ipa_courses/blob/main/Lecture%202/practice4_file_IO_gov_hk_202409.ipynb

Business Process Automation with Python

Notebook Summary

https://github.com/innovai/iqa_courses/blob/main/Lecture%202/practice2_word_count_202409.ipynb

The screenshot shows a Jupyter Notebook interface with the following structure:

- Table of contents**:
 - Python Introduction
 - 0) Displaying values
 - The last line in a code block is evaluated and displayed
 - You can also use print() method to display the values
 - 1) Primitive Data types
 - Numeric - Integer
 - Numeric - Float (with decimal points)
 - Numeric - Complex (For scientific calculations)

Common functions:
 - Text - String
 - Boolean (True or False)
 - Simple conversion
 - Simple string formatting
 - 2) Sequential Data Types
 - Motivation - 1 variable to store many values
 - Tuples (multiple values)
 - List - You can change the values
 - String - as a sequential data type
 - Set - Unique values
 - 3) Dictionary - Key and value pairs lookup
 - 4) Control Flows
 - If-else

Business Process Automation with Python

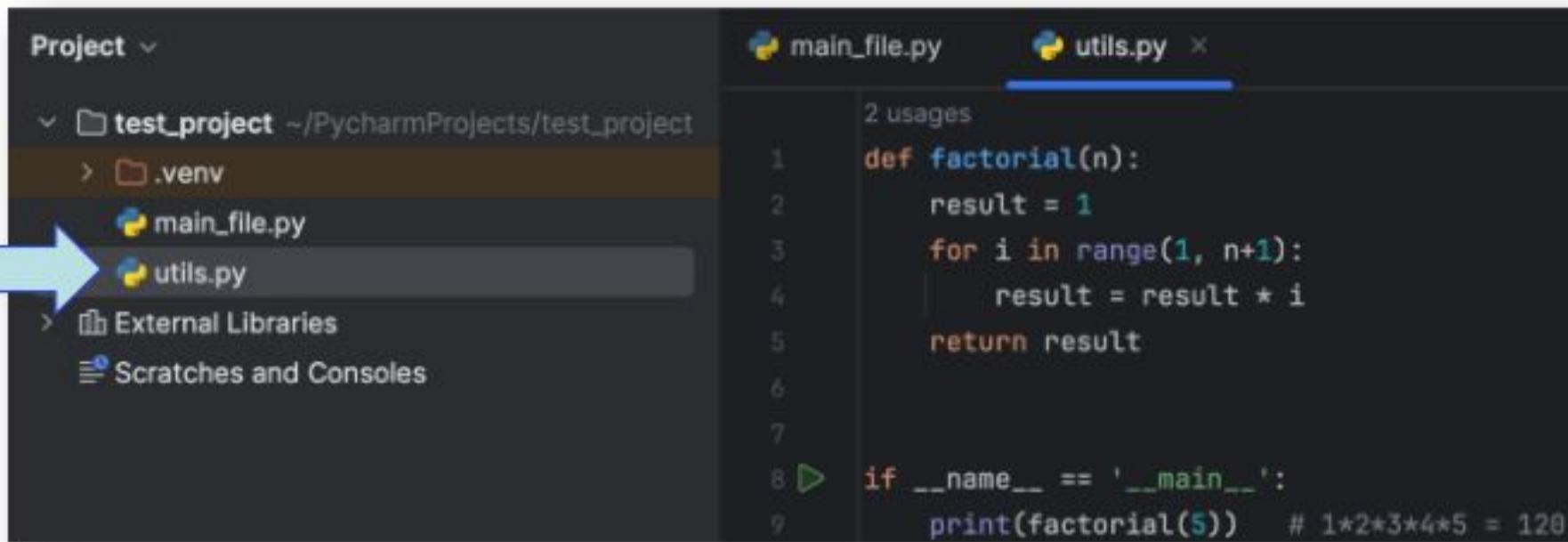
Practice 4 - Review

- It takes so much effort to read CSV files
 - Q: But this should be a common task?
 - Any instant noodles for me to “add hot water & eat”?



Business Process Automation with Python

Importing Modules



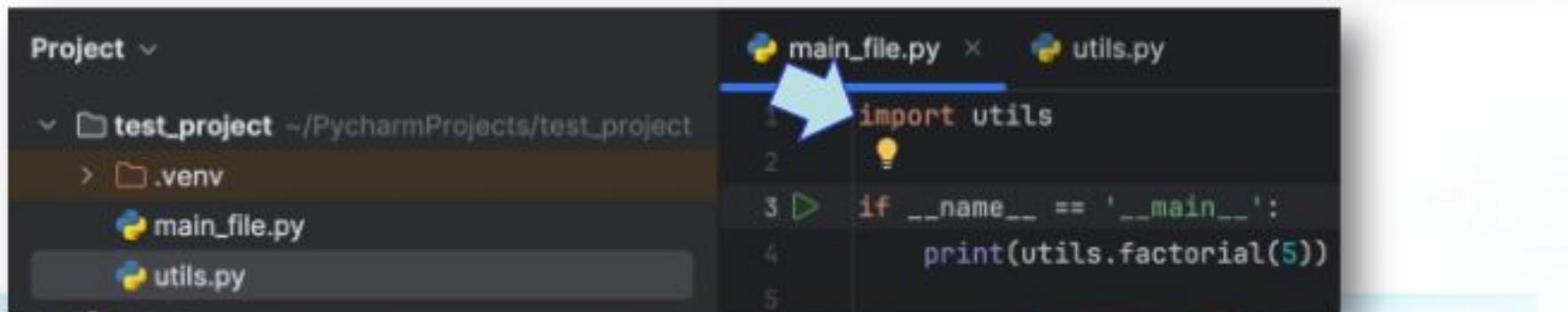
Project

- test_project (~/PycharmProjects/test_project)
 - .venv
 - main_file.py
 - utils.py
- External Libraries
- Scratches and Consoles

main_file.py

```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result

if __name__ == '__main__':
    print(factorial(5)) # 1*2*3*4*5 = 120
```



Project

- test_project (~/PycharmProjects/test_project)
 - .venv
 - main_file.py
 - utils.py

main_file.py

```
import utils

if __name__ == '__main__':
    print(utils.factorial(5))
```

Business Process Automation with Python

Python Architecture (Python runtime + Packages)

- Packages

- Built-in / published codes which can be imported to use
- e.g., **datetime** module → **date** class → **strftime** function

```
▶ import datetime  
  
dt = datetime.date(2024, 12, 25)  
print(dt.strftime('%Y-%m-%d'))  
print(dt.strftime('%d %B %Y (%A)'))
```

```
→ 2024-12-25  
25 December 2024 (Wednesday)
```



Python



Packages

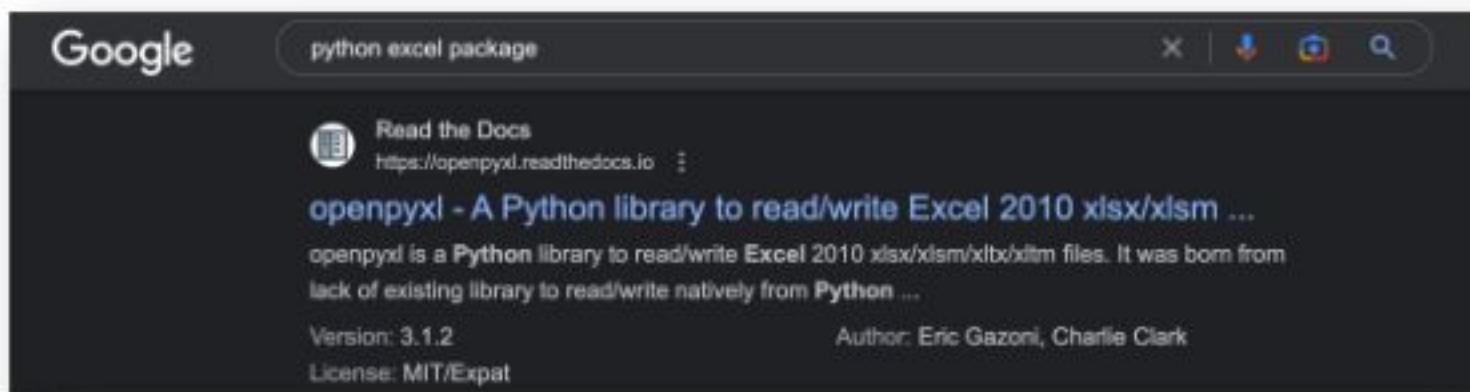


Easy programs!

Business Process Automation with Python

Packages - Installation

- Default installer – pip (Python 3.4 or later)
 - Search for “Python” + (something you need) + “Package”
 - Run “pip install xxx” in command prompt / terminal



Installation

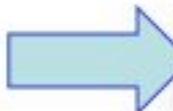
Install openpyxl using pip. It is advisable to do this in a Python virtualenv without system packages:

```
$ pip install openpyxl
```

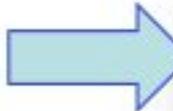
Business Process Automation with Python

Packages - Installation

- In Jupyter Notebooks
 - Add “!” → Install system-wide
 - Add “%” → Install within notebook environment



```
● !pip install pandas
C: Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (1.5.3)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.9/dist-packages (from pandas) (1.22.4)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```



```
● %pip install pandas
C: Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (1.5.3)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.9/dist-packages (from pandas) (1.22.4)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

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.

EDA: without Pandas

Say we have a small dataset of the top 50 most-streamed Spotify songs, globally, for 2019.

EDA: without Pandas

Say we have a small dataset of the top 50 most-streamed Spotify songs, globally, for 2019.

NOTE: The following music data are used purely for illustrative, educational purposes. The data, including song titles, may include explicit language. Harvard, including myself and the rest of the CS109 staff, does not endorse any of the entailed contents or the songs themselves, and we apologize if it is offensive to anyone in anyway.

EDA: without Pandas

top50.csv

Each row represents a distinct song. The columns are:

- **ID:** a unique ID (i.e., 1-50)
- **TrackName:** Name of the Track
- **ArtistName:** Name of the Artist
- **Genre:** the genre of the track
- **BeatsPerMinute:** The tempo of the song.
- **Energy:** The energy of a song - the higher the value, the more energetic.
- **Danceability:** The higher the value, the easier it is to dance to this song.
- **Loudness:** The higher the value, the louder the song.
- **Liveness:** The higher the value, the more likely the song is a live recording.
- **Valence:** The higher the value, the more positive mood for the song.
- **Length:** The duration of the song (in seconds).
- **Acousticness:** The higher the value, the more acoustic the song is.
- **Speechiness:** The higher the value, the more spoken words the song contains.
- **Popularity:** The higher the value, the more popular the song is.

EDA: without Pandas

top50.csv

TrackName	ArtistName	Genre	BeatsPer			Live			Speechi			Popularity
			Minute	Energy	Danceability	Loudness	ness	Valence	Length	Acousticness	ness	
1 Senorita	Shawn Mendes	canadian pop	117	55	76	-6	8	75	191	4	3	79
2 China	Anuel AA	reggaeton flow	105	81	79	-4	8	61	302	8	9	92
3 boyfriend (w Ariana Grande)	Ariana Grande	dance pop	190	80	40	-4	16	70	186	12	46	85
4 Beautiful Picnic	Ed Sheeran	pop	93	65	64	-8	8	55	198	12	19	86

-
-
-

Q1: What are some ways we can store this file into data structure(s) using regular Python (not the Pandas library).

EDA: without Pandas

top50.c

TrackName	ArtistName	Genre	BeatsPer	SV	Live				Speechi	Popularity	
			Minute		Energy	Danceability	Loudness	ness	Valence	Length	
1 Senorita	Shawn Mendes	canadian pop	117	55	76	-6	8	75	191	4	3
2 China	Anuel AA	reggaeton flow	105	81	79	-4	8	61	302	8	9
3 boyfriend (w Ariana Grande)	Ariana Grande	dance pop	190	80	40	-4	16	70	186	12	46
4 Beautiful Picnic	Ed Sheeran	pop	93	65	64	-8	8	55	198	12	19

- Possible Solution #1: A 2D array (i.e., matrix)

Weaknesses:

- What are the row and column names? Need separate lists for them - clumsy.
- Lists are O(N). We'd need 2 dictionaries just for column names

```
data = []
col_name
-> index
index ->
col_name
```

EDA: without Pandas

top50.csv

TrackName	ArtistName	Genre	BeatsPer			Live			Speechi			Popularity
			Minute	Energy	Danceability	Loudness	Valence	Length	Acousticness	ness	ness	
1 Senorita	Shawn Mendes	canadian pop	117	55	76	-6	8	75	191	4	3	79
2 China	Anuel AA	reggaeton flow	105	81	79	-4	8	61	302	8	9	92
3 boyfriend (w Ariana Grande)	Ariana Grande	dance pop	190	80	40	-4	16	70	186	12	46	85
4 Beautiful Picnic	Ed Sheeran	pop	93	65	64	-8	8	55	198	12	19	86

-
-
- Possible Solution #2: A list of dictionaries

list

Item 1	=	{"Track.Name": "Senorita", "Artist.Name": "Shawn Mendes", "Genre": "Canadian pop", ...}
Item 2	=	{"Track.Name": "China", "Artist.Name": "Anuel AA", "Genre": "reggaetón flow", ... }
Item 3	=	{"Track.Name": "Ariana Grande", "Artist.Name": "boyfriend", "Genre": "dance pop", ... }

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

```
f = open("../data/top50.csv", encoding = "ISO-8859-1")
column_names = f.readline().strip().split(",")[1:] # puts names in a list
cleaned_column_names = [name[1:-1] for name in column_names]
cleaned_column_names.insert(0, "ID")

dataset = []

# iterates through each line of the .csv file
for line in f:
    attributes = line.strip().split(",")

    # constructs a new dictionary for each line
    dataset.append(dict(zip(cleaned_column_names, attributes)))
```

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

Q2: Write code to print all songs (Artist and Track name) that are longer than 4 minutes (240 seconds):

```
for song in dataset:  
    if int(song["Length."]) > 240:  
        print(song["Artist.Name"], "-", song["Track.Name"], "is", song["Length."], "seconds long")
```

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

Q3: Write code to print the most popular song (artist and track) – if ties, show all ties.

```
max_score = -1
most_populars = set()
for song in dataset:
    if int(song["Popularity"]) > max_score:
        most_populars = set([str(song["Artist.Name"] + "-" + song["Track.Name"])])
        max_score = int(song["Popularity"])
    elif int(song["Popularity"]) == max_score:
        most_populars.add(str(song["Artist.Name"] + "-" + song["Track.Name"]))
print(most_populars)
```

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

Q4: Write code to print the songs (and their attributes), if we sorted by their popularity (highest scoring ones first).

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

Q4: Write code to print the songs (and their attributes), if we sorted by their popularity (highest scoring ones first).

list	
Item 1	= {“Track.Name”: “Senorita”, “Artist.Name”: “Shawn Mendes”, “Genre”: “Canadian pop”, ...}
Item 2	= {“Track.Name”: “China”, “Artist.Name”: “Anuel AA”, “Genre”: “reggaetón flow”, ... }
Item 3	= {“Track.Name”: “Ariana Grande”, “Artist.Name”: “boyfriend”, “Genre”: “dance pop”, ... }

Cumbersome to move dictionaries around in a list. Problematic even if we don't move the dictionaries.

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

Q5: How could you check for null/empty entries? This is only 50 entries.
Imagine if we had 500,000.

list	
Item 1	= {“Track.Name”: “Senorita”, “Artist.Name”: “Shawn Mendes”, “Genre”: “Canadian pop”, ...}
Item 2	= {“Track.Name”: “China”, “Artist.Name”: “Anuel AA”, “Genre”: “reggaetón flow”, ... }
Item 3	= {“Track.Name”: “Ariana Grande”, “Artist.Name”: “boyfriend”, “Genre”: “dance pop”, ... }

EDA: list of dictionaries

Possible Solution #2: A list of dictionaries

Q6: Imagine we had another table* below (i.e., .csv file). How could we combine its data with our already-existing *dataset*?

spotify_aux.			
	TrackName	ArtistName	ExplicitLanguage
1	Senorita	Shawn Mendes	TRUE
2	China	Anuel AA	FALSE
3	boyfriend (w Ariana Grande)	Ariana Grande	TRUE
4	Beautiful People	Ed Sheeran	FALSE

* 3rd column is made-up by me. Random values. Pretend they're accurate.

EDA: with Pandas!



Kung Fu Panda is property of DreamWorks and Paramount Pictures

Lecture Outline

- Exploratory Data Analysis (EDA):
 - Without Pandas (part 1) – These slides
 - With Pandas (part 2) – Mostly Jupyter Notebook
- Data concerns (part 3) – These slides
- Web Scraping with Beautiful Soup (part 4) – Mix

What / Why?

- Pandas is an *open-source* [Python library](#) (anyone can contribute)
- Allows for high-performance, easy-to-use data structures and data analysis
- Unlike NumPy library which provides multi-dimensional arrays, Pandas provides 2D table object called [DataFrame](#) (akin to a spreadsheet with column names and row labels).
- Used by *a lot* of people

EDA: with Pandas

How

- import `pandas` library (convenient to rename it)
- Use `read_csv()` function

```
import pandas as pd  
dataframe = pd.read_csv("yourfile.csv")
```

Common Panda functions

High-level viewing:

- `head()` – first N observations
- `tail()` – last N observations
- `columns()` – names of the columns
- `describe()` – statistics of the quantitative data
- `dtypes()` – the data types of the columns

Common Panda functions

Accessing/processing:

- `df["column_name"]`
- `Df.column_name`
- `.max(), .min(), .idxmax(), .idxmin()`
- `<dataframe> <conditional statement>`
- `.loc[]` – label-based accessing
- `.iloc[]` – index-based accessing
- `.sort_values()`
- `.isnull(), .notnull()`

Common Panda functions

Grouping/Splitting/Aggregating:

- `groupby()`, `.get_groups()`
- `.merge()`
- `.concat()`
- `.aggregate()`
- `.append()`

Lecture Outline

- Exploratory Data Analysis (EDA):
 - Without Pandas (part 1) – These slides
 - With Pandas (part 2) – Mostly Jupyter Notebook
- Data concerns (part 3) – These slides
- Web Scraping with Beautiful Soup (part 4) – Mix

Data Concerns

When determining if a dataset is sound to use, it can be useful to think about these four questions:

- Did it come from a trustworthy, authoritative source?
- Is the data a complete sample?
- Does the data seem correct?
- **(optional)** Is the data stored efficiently or does it have redundancies?

Data Concerns: the format

- Often times, there may not exist a single dataset that contains all of the information we are interested in.
- May need to merge existing datasets
- Important to do so in a sound and efficient format

Data Concerns: the format

For example, say we have two datasets:

Dataset 1

Top 200 most-frequent streams per day (for June 2019)

SpotifySongID, # of Streams, Date		
2789179,	42003,	06-01
3819390,	89103,	06-01
4492014,	52923,	06-02
8593013,	189145,	06-02

6,000 x 3

Dataset 2

Top 50 most streamed in 2019, so far

SpotifySongID, Artist, Track, [10 acoustic features]		
2789179,	Billie Eilish, bad guy, 3.2, 5.9, ...	
3901829,	Outkast, Elevators, 9.3, 5.1, ...	
50 x 13		

Data Concerns: the format

For example, say we have two datasets:

Dataset 1

Top 200 most-frequent streams per day (for June 2019)

SpotifySongID, # of Streams, Date

2789179,	42003,	06-01
3819390,	89103,	06-01
4492014,	52923,	06-02
8593013,	189145,	06-02

6,000 x 3

Dataset 2

Top 50 most streamed in 2019, so far

SpotifySongID, Artist, Track, [10 acoustic features]

2789179,	Billie Eilish, bad guy, 3.2, 5.9, ...
3901829,	Outkast, Elevators, 9.3, 5.1, ...

50 x 13

We are interested in determining if songs with high danceability are more popular during the weekends of June than weekdays in June. What should our merged table look like? Concerns?

Data Concerns: the format

This is wasteful, as it has 10 acoustic features, artist, and track repeated many times for each unique song.

Datasets Merged (poorly)

SpotifySongID, # of Streams, Date,Artist, Track, [10 acoustic features]

200	2789179, 06-01	42003,	Billie Eilish, bad guy, 3.2, 5.9, ...
200	3819390, 06-01	89103,	Outkast, Elevators, 9.3, 5.1, ...
200	4492014, 06-02	52923,	
200	8593013, 06-02	189145,	

6,000 x 15 □ 90,000 cells

Data Concerns: the format

Some rows may have null values for # of Streams (if the song wasn't popular in June)

Datasets Merged (better)

SpotifySongID, Artist, Track, [10 acoustic features], 06-01 Streams, 06-02 Streams

2789179,	Billie Eilish, bad guy, 3.2, 5.9, ...	42003, 42831, 43919
3901829,	Outkast, Elevators, 9.3, 5.1, ...	29109, 27193, 25982
		▪ ▪

50

50 x 70 □ 3,500 cells

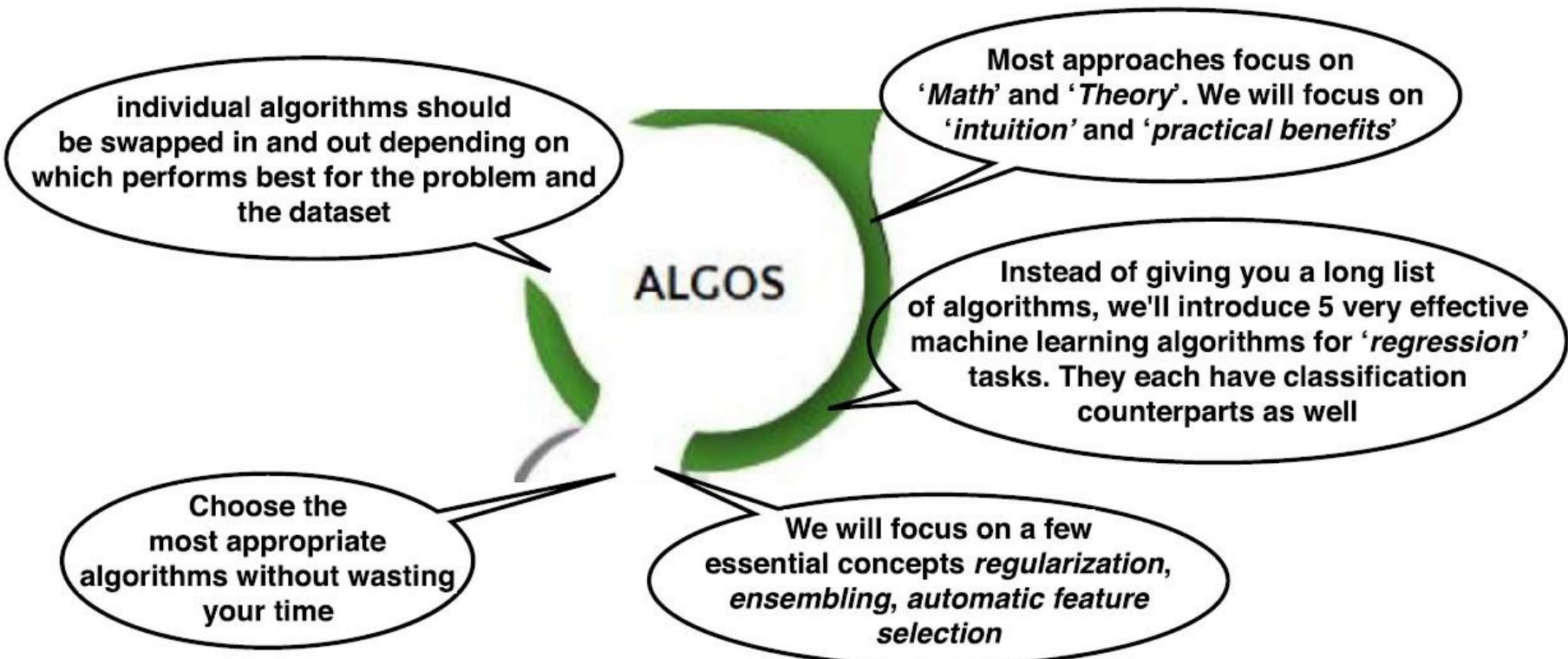
Data Concerns: the format

- Is the data correctly constructed (or are values wrong)?
- Is there redundant data in our merged table?
- Missing values?

Big Data Tools and Algorithm Exploration

Applied Machine Learning Algorithm Selection

Overview



Source: <https://elitedatascience.com/algorithm-selection>

Big Data Tools and Algorithm Exploration

Applied Machine Learning Algorithm Selection

HKUSPACE

Highly recommend skipping Linear Regression for most machine learning problems

2.0

0.5

0.0

-0.5

-1.0

-1.5

-2.5

-1

Big Data Tools and Algorithm Exploration

Applied Machine Learning Algorithm Selection

Linear Regression models are easy to interpret and understand, but they are deeply flawed and rarely perform well !

Regression is the supervised learning task for modeling and predicting continuous, numeric variables. Examples include predicting real-estate prices, stock price movements, or student test scores

Regression tasks are characterized by labeled datasets that have a numeric target variable, you have some "ground truth" value for each observation that you can use to supervise your algorithm

Simple linear regression models fit a "straight line", a *hyperplane* depending on the number of features

Simple linear regression are 1 - prone to overfitting with many input features and 2 - cannot easily represent non-linear relationships

Inspiring Your Future

Business Education @ HKUSPACE

Source: <https://elitedatascience.com/algorithm-selection>

HKUSPACE

Big Data Tools and Algorithm Exploration

Applied Machine Learning Algorithm Selection

Each tree is only trained on a random subset of observations (a process called resampling)

Each tree is only allowed to choose from a random subset of features to split on (leading to feature selection)



Random Forests

Random forests train a large number of "strong" decision trees and combine their predictions through bagging

random forests tend to perform very well right out of the box often beat many other models that take up to weeks to develop, and are the perfect "swiss-army-knife" algorithm that almost always gets good results

Source: <https://elitedatascience.com/algorithm-selection>

Big Data Tools and Algorithm Exploration

Model Training

Training sets are used to 'fit and tune' (train) your models

Split your data **before** doing anything else then separately train and test subsets of your dataset

We set up the entire modelling process to maximise performance whilst preventing overfitting

Training Set

Test Set

Train and tune your models (using cross-validation)

Don't touch this until the very end.

Test sets are put aside as "*unseen*" data to evaluate your models, splitting your data, don't touch your test set until you're ready to choose your final model

If the model performs very well on the training data but poorly on the test data, then it's overfit

Split DataSet Into Training/Test Sets

If you evaluate your model on the same data you used to train it, your model could be very overfit. A model should be judged on its ability to predict *new, unseen data*

Source: <https://elitedatascience.com/model-training>

Big Data Tools and Algorithm Exploration

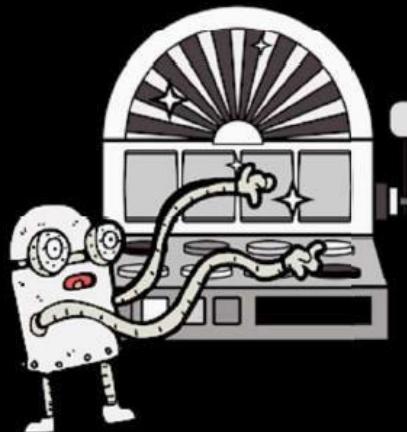
Model Training

Examples of Model parameters include regression coefficients, decision tree split locations

Model parameters are attributes that define *individual models*, and are learned directly from the training data

There are two types of parameters in machine learning algorithms: Model Parameters and hyperparameters

"Don't mind me. Just randomly pressing buttons carefully tuning level parameters."



Hyperparameters: When we talk of tuning models, we specifically mean tuning hyperparameters

Hyperparameters express "higher-level" *structural settings* for algorithms. They are decided before fitting the model because they can't be learned from the data

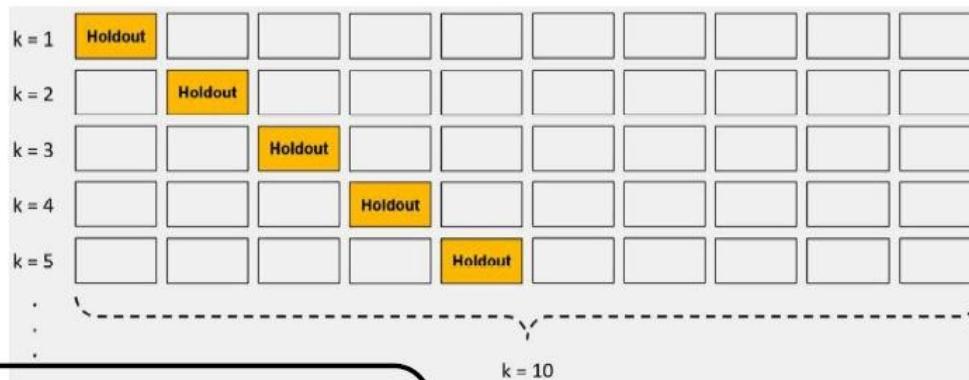
E.g. Hyperparameters "higher-level" *structural settings* are strength of the penalty used in regularized regression and the number of trees to include in a random forest

Source: <https://elitedatascience.com/model-training>

Big Data Tools and Algorithm Exploration

Model Training

1- Split your data into 10 equal parts, or “folds”, 2- Train your model on 9 folds (e.g. the first 9 folds)



10-fold cross-validation, breaks your training data into 10 equal parts (or *folds*), essentially creating 10 miniature train/test splits

Cross-validation is a tuning method for getting a reliable estimate of model performance using only training data

3 - Evaluate it on the 1 remaining "hold-out" fold, Perform steps (2) and (3) 10 times, each time holding out a different fold

Average the performance across all 10 hold-out folds, average performance across the 10 hold-out folds is your final performance estimate, also called your *cross-validated score*

Source: <https://elitedatascience.com/model-training>

Big Data Tools and Algorithm Exploration

Model Training

perform the entire cross-validation loop detailed above on each set of hyperparameter values using a high-level pseudo-code

We've split our dataset into training and test sets, and we've learned about hyperparameters and cross-validation, we're ready fit and tune our models

For each algorithm (i.e. regularized regression, random forest, etc.):

For each set of hyperparameter values to try:
Perform cross-validation using the training set.
Calculate cross-validated score.

At the end of this process, you will have a cross-validated score for each set of hyperparameter values... for each algorithm

Elastic-Net		
Penalty Ratio	Penalty Strength	CV-Score
75/25	0.01	0.63
75/25	0.05	0.64
75/25	0.10	0.67
50/50	0.01	0.62
50/50	0.05	0.63
50/50	0.10	0.66

Source: <https://elitedatascience.com/model-training>

Big Data Tools and Algorithm Exploration

Model Training

Keep the set of hyperparameter values with best cross-validated score.
Re-train the algorithm on the entire training set (without cross-validation)

By now, you'll have 1 "best" model for each algorithm that has been tuned through cross-validation. Most importantly, you've only used the training data so far

Because you've saved your test set as a truly unseen dataset, you can now use it get a reliable estimate of each models' performance. There are a variety of performance metrics you could choose from

For classification tasks, we recommend Area Under ROC Curve (AUROC). *Higher values are better*

For regression tasks, we recommend Mean Squared Error (MSE) or Mean Absolute Error (MAE). (*Lower values are better*)



"I volunteer as tribute"

like the Hunger Games... each algorithm sends its own "representatives" (i.e. model trained on the best set of hyperparameter values) to the final selection

Source: <https://elitedatascience.com/model-training>

Big Data Tools and Algorithm Exploration

Model Training Conclusion

Pick the winning model by asking: Which model had the best performance on the test set? (performance)

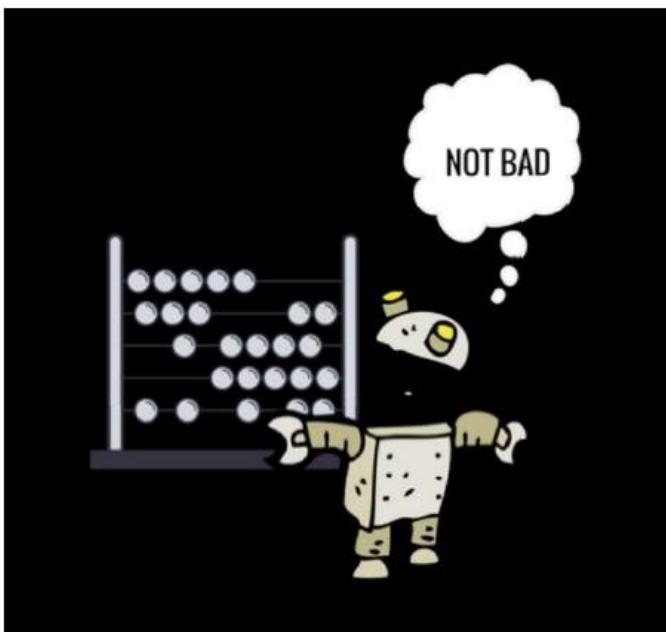
Calculate performance metrics using those predictions and the "ground truth" target variable from the test set

For each of your models, make predictions on your test set

Pick the winning model by asking: Does it perform well across various performance metrics? (robustness)

Pick the winning model by asking: Did it also have (one of) the best cross-validated scores from the training set? (consistency)

Pick the winning model by asking: Does it solve the original business problem? (win condition)



Source: <https://elitedatascience.com/model-training>

Conduct Random Forest Model training

#This code is going to train an random forest model to find top important features

#Step 1 : Import necessary library

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import plot_tree
import joblib # Import joblib for saving the model
```

#Step 2 : import data for machine learning model training

```
data = pd.read_csv('combined_building_withschool.csv')
```

#Step 3 : Machine Learning Model Implementation

Encode categorical variables

```
label_encoder = LabelEncoder()
data['District'] = label_encoder.fit_transform(data['District'])
data['School_Netnumber'] = label_encoder.fit_transform(data['School_Netnumber'])
```

Define features and target variable

```
X = data[['District', 'Floor', 'School_Netnumber']] # Features
```

```
y = data['Price Per Square'] # Target variable
```

Check the distribution of the target variable

```
print(y.value_counts())
```

```
print(X)
```

	Price Per Square			
0.00	30508			
10000.00	217			
8333.33	109			
8000.00	81			
12500.00	70			
...	...			
8554.05	1			
13764.04	1			
10443.86	1			
15421.30	1			
4672.90	1			
Name: count, Length: 31331, dtype: int64				
	District	Floor	School_Netnumber	
0	1	2.0	8	
1	16	1.0	2	
2	31	7.0	5	
3	28	1.0	7	
4	24	1.0	2	
...
79995	6	2.0	9	
79996	9	9.0	6	
79997	3	4.0	4	
79998	6	1.0	9	
79999	26	3.0	1	
[80000 rows x 3 columns]				

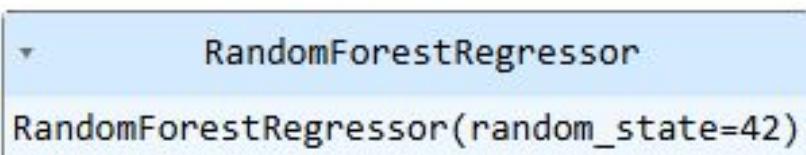
Conduct Random Forest Model training

```
# Step 3: Split the data into training and testing sets with stratification
# Ensure the target variable is continuous
print(f"Target variable type: {y.dtype}")

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 6: Display a decision tree from the Random Forest with reduced size
plt.figure(figsize=(6, 6)) # Reduced by 10%
plot_tree(rf_model.estimators_[5], feature_names=X.columns.tolist(), filled=True)
plt.title("Decision Tree from Random Forest")
plt.show()
```

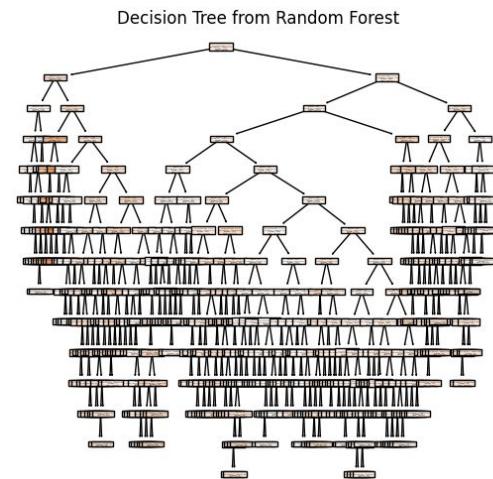


Conduct Random Forest Model training

```
# Display feature importances
importances = rf_model.feature_importances_
feature_names = X.columns.tolist()
feature_importances = pd.DataFrame({'Feature': feature_names, 'Importance':
    importances})
```

```
# Sort by importance and display the top 5 features
top_features = feature_importances.sort_values(by='Importance',
    ascending=False).head(5)
print("Top 5 Features and their Importance Scores:")
print(top_features)
```

```
# Step 5: Save the trained model to a file
model_filename = 'housing_valuation_model.joblib' # Specify the filename
joblib.dump(rf_model, model_filename) # Save the model
print(f'Model saved to {model_filename}')
```

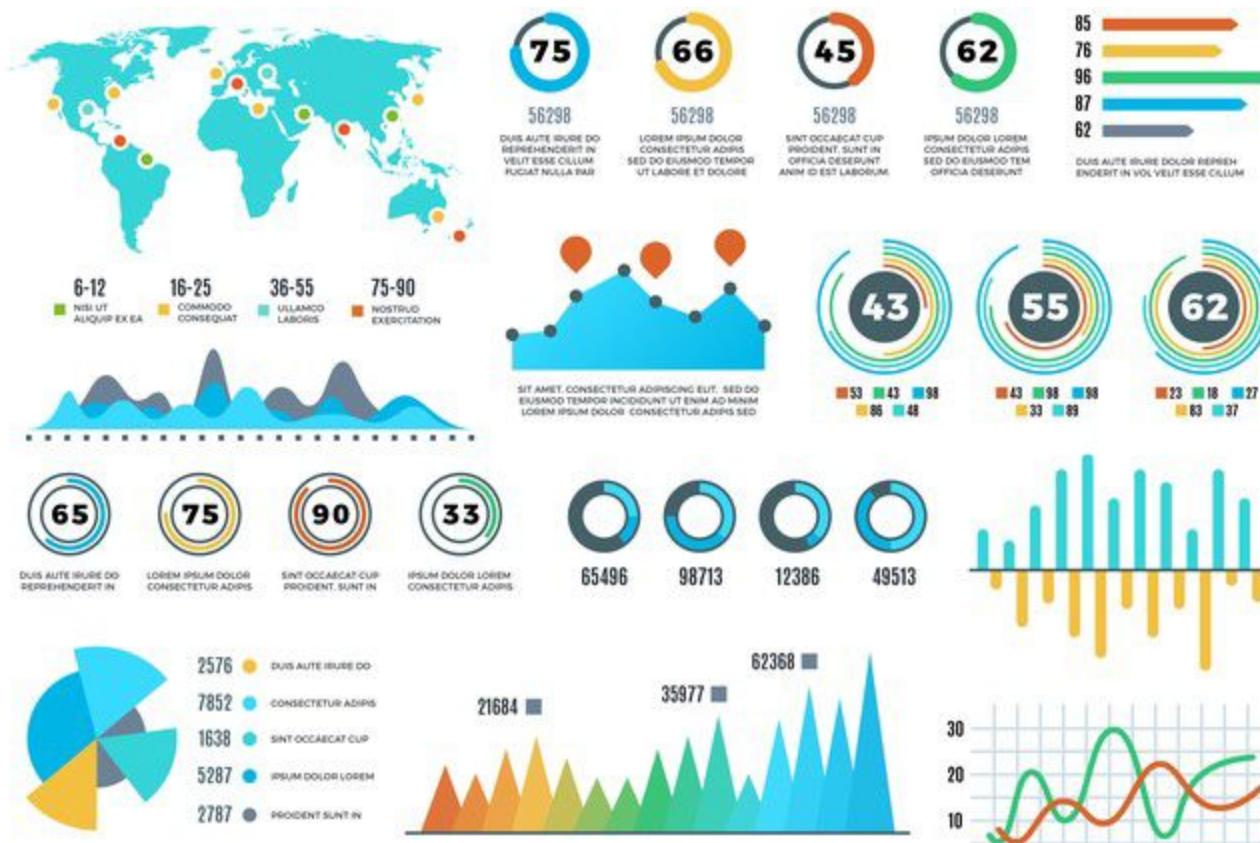


Top 5 Features and their Importance Scores:		
	Feature	Importance
0	District	0.728459
2	School_Netnumber	0.199190
1	Floor	0.072351

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.



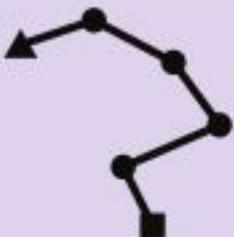
Sparkline



+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

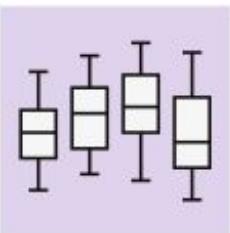


+Distributions

+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

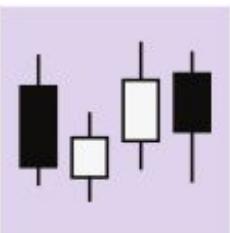


+Distributions

+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.

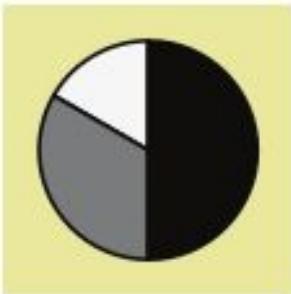
Tracking change over time is of key interest in the financial domain. One specialist chart developed for this field includes the following:



Candlestick chart ◆

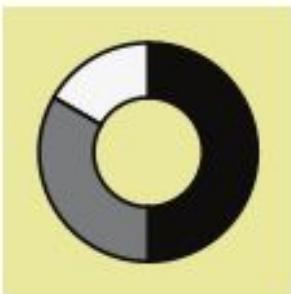
Looks like a box plot, but each box and whiskers encodes different statistics. The box ends indicate opening and closing prices, while color indicates the direction of change.

Data Visualization Roles - Part-to-whole composition



Pie chart

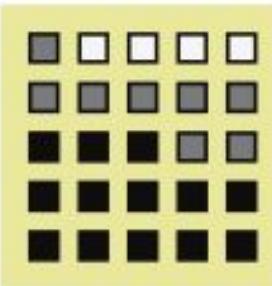
The whole is represented by a filled circle. Parts are proportional slices from that circle, one for each categorical group. Best with five or fewer slices with distinct proportions.



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).

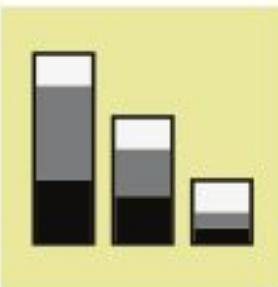
Data Visualization Roles - Part-to-whole composition



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



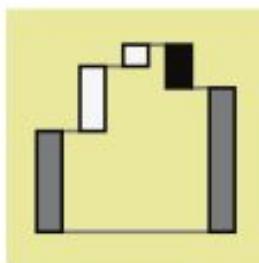
A line chart (see *Change over time*) where shaded regions are added under the line to divide the total into sub-group values.



Stream graph



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



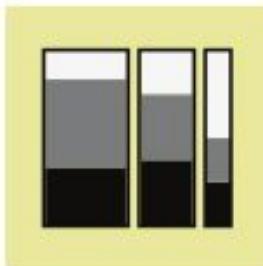
Waterfall chart



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.

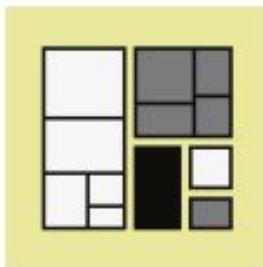
Data Visualization Roles - Part-to-whole composition

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 ◆

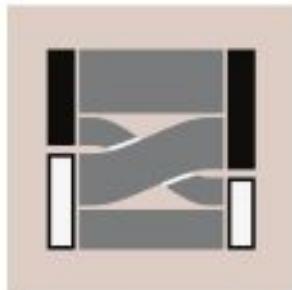
Can be thought of as a more generalized Marimekko plot. Sub-boxes do not need to have a consistent cut direction at a particular hierarchy level, and there can be more than two levels of hierarchy.

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

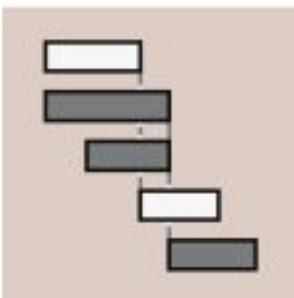
Multiple part-to-whole divisions on different dimensions are depicted as parallel stacked bars. Connecting regions show how different subgroups relate to one another between dimensions.

Data Visualization Roles - Flows and processes



Sankey diagram ◆

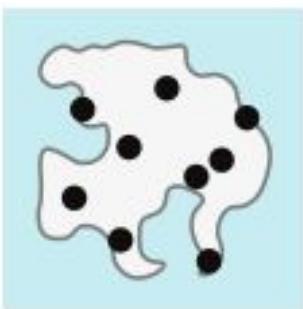
The width of the colored region shows the relative volume at each part of a process. Allows for multiple sources of inputs and outputs to be visualized.



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.

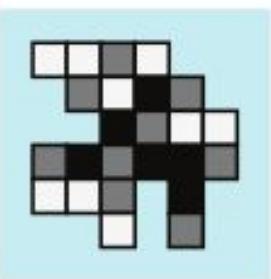


Bubble map



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



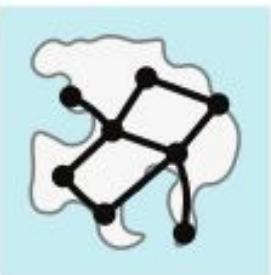
Heatmaps can be built on top of geographic areas. Sometimes seen with a hexagon-shaped grid rather than a rectangular grid. May distort the geography on its edges.



Isopleth / contour map



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



Connection map



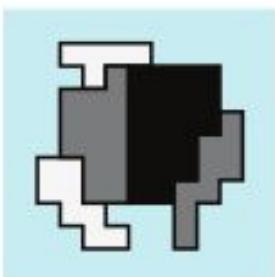
Network information and flows built on top of a geographic map.

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

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
```

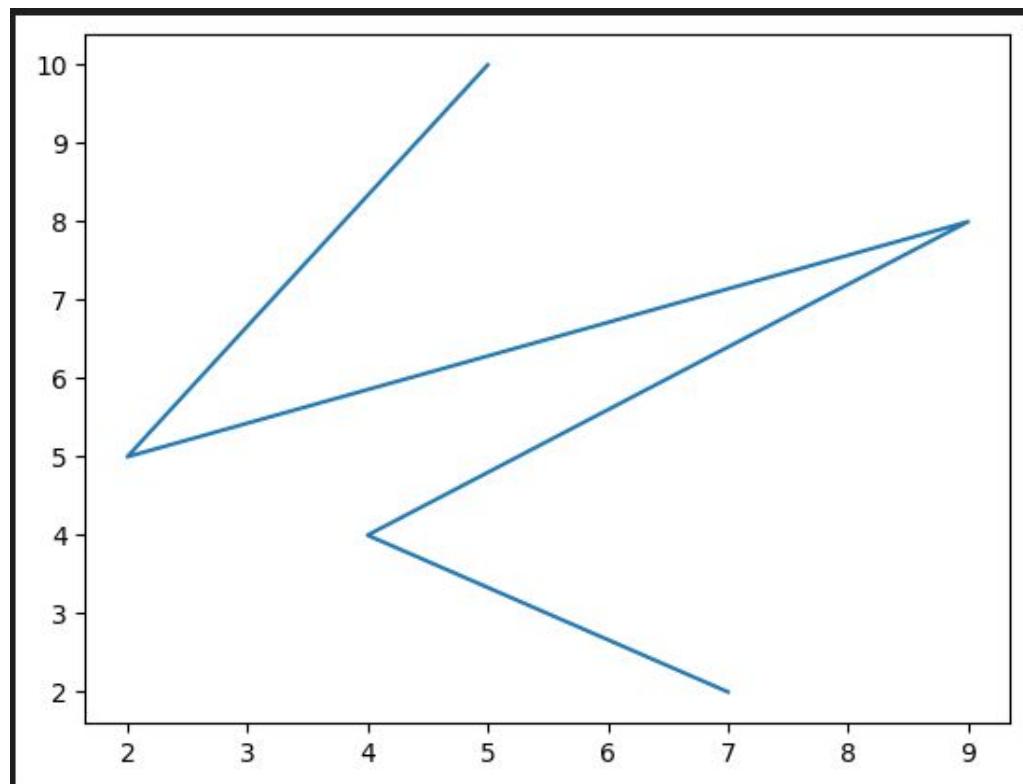
```
y = [10, 5, 8, 4, 2]
```

```
# 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
```

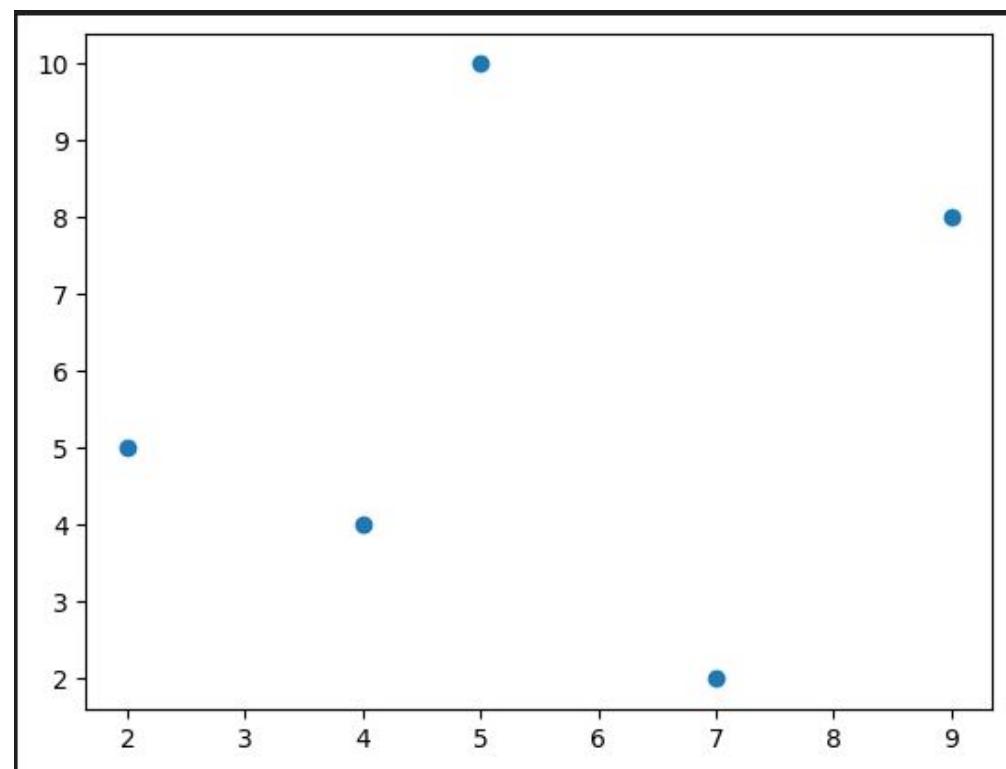
```
y = [10, 5, 8, 4, 2]
```

```
# 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
```

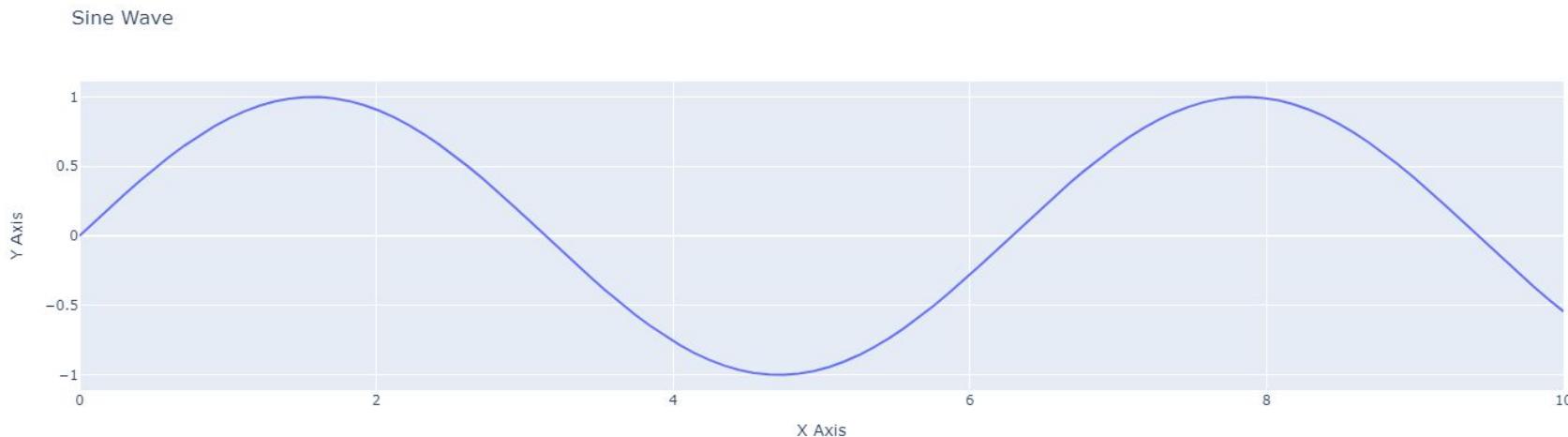
```
fig.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Sine  
Wave'))
```

```
# 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 histgram 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

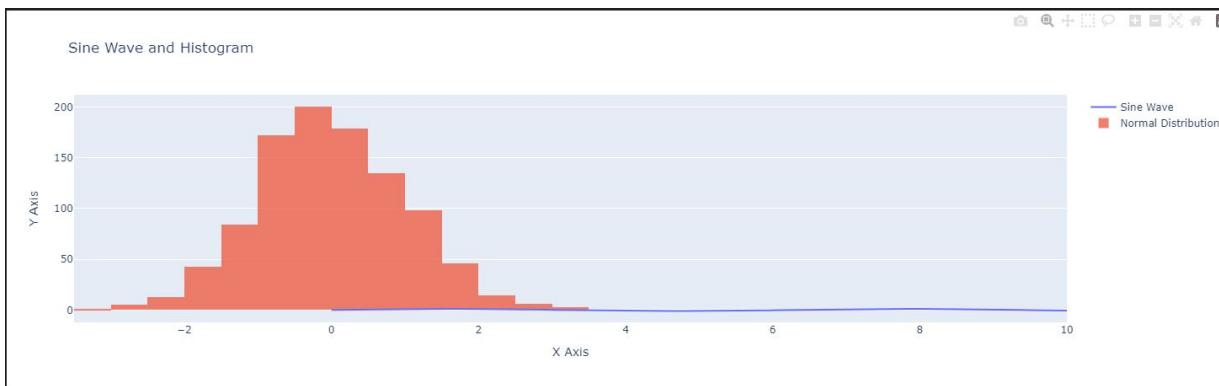
# Create a plotly figure
fig = go.Figure()
```

```
# 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
```

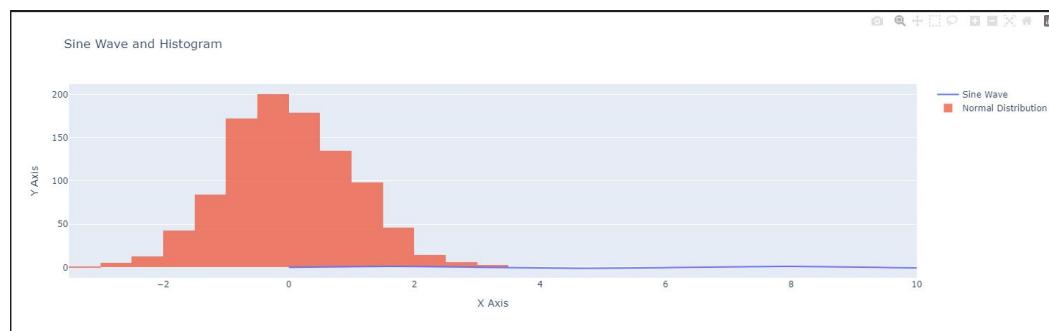
```
# Show the plot
```

```
fig.show()
```

```
# Show the plot
```

```
fig.show()
```

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

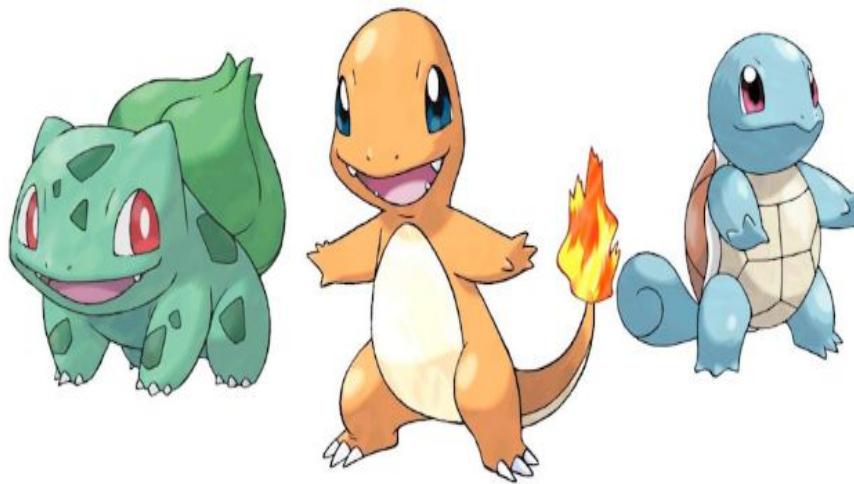


Download the data

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 bypass 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		309	...	50	65	1	False
5	Charmeleon	Fire		405	...	65	80	2	False

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

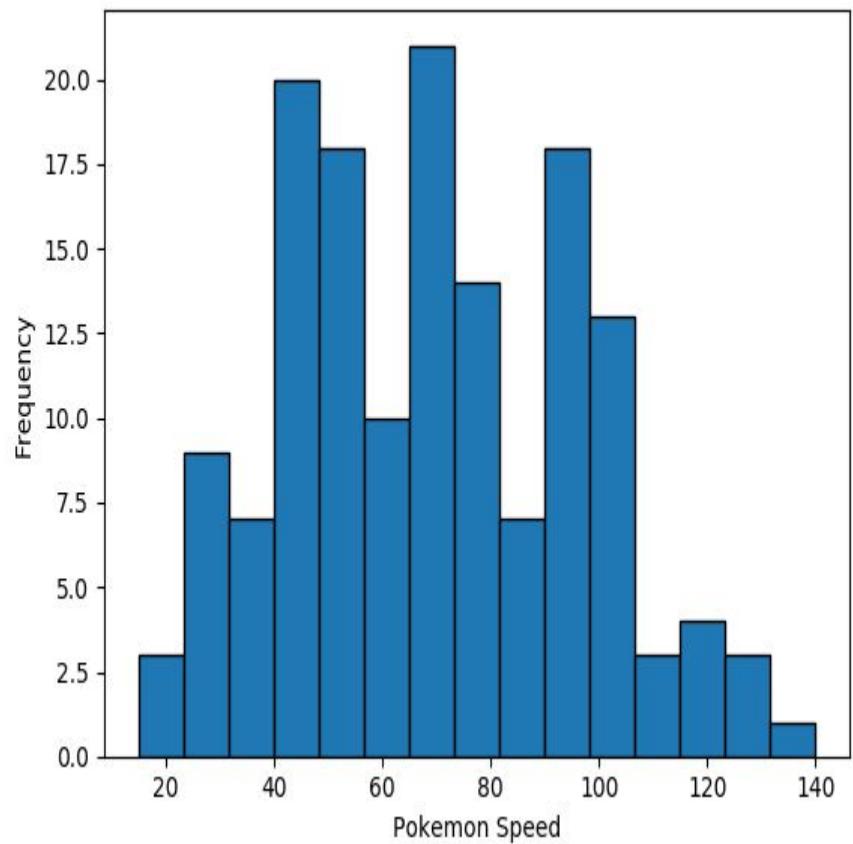
	Total	HP	Attack	...	Sp. Def	Speed	Stage
count	151.00000	151.00000	151.00000	...	151.00000	151.00000	151.00000
mean	407.07947	64.211921	72.549669	...	66.019868	68.933775	1.582781
std	99.74384	28.590117	26.596162	...	24.197926	26.746880	0.676832
min	195.00000	10.00000	5.00000	...	20.00000	15.00000	1.000000
25%	320.00000	45.00000	51.00000	...	49.00000	46.50000	1.000000
50%	405.00000	60.00000	70.00000	...	65.00000	70.00000	1.000000
75%	490.00000	80.00000	90.00000	...	80.00000	90.00000	2.000000
max	680.00000	250.00000	134.00000	...	125.00000	140.00000	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.

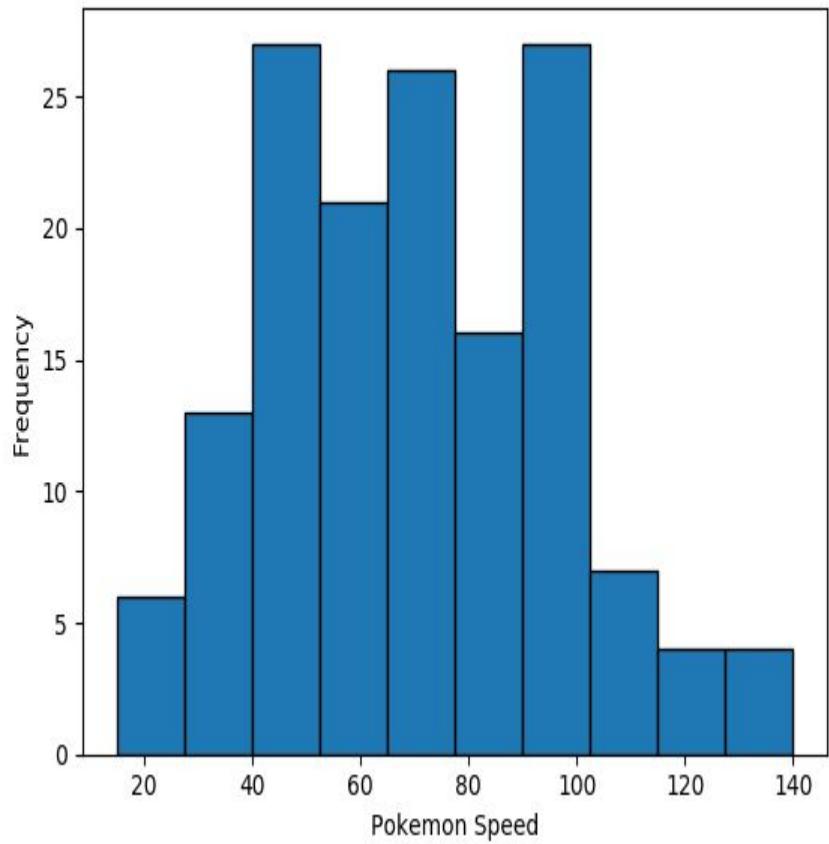
However, a good field commander never goes
into battle without first doing a reconnaissance
of the terrain...

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()
```

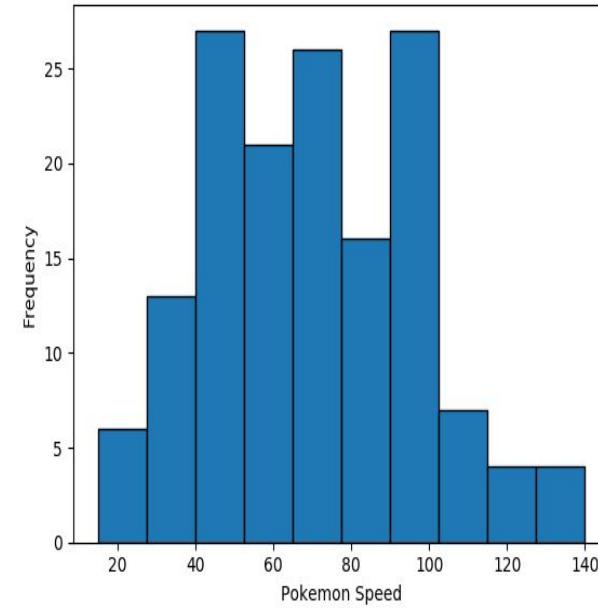
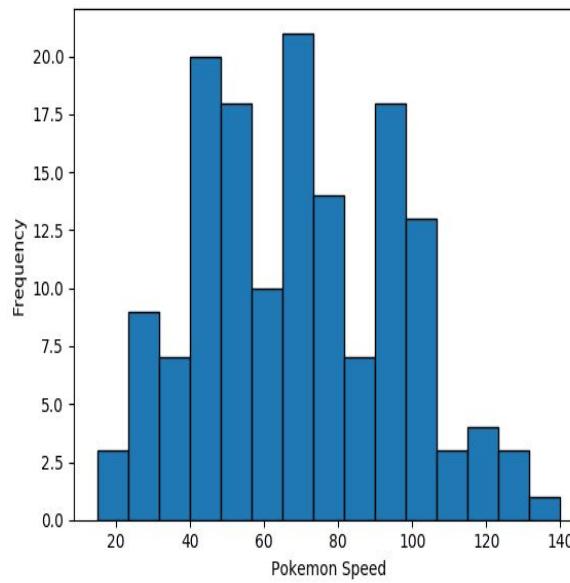


Bins

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.

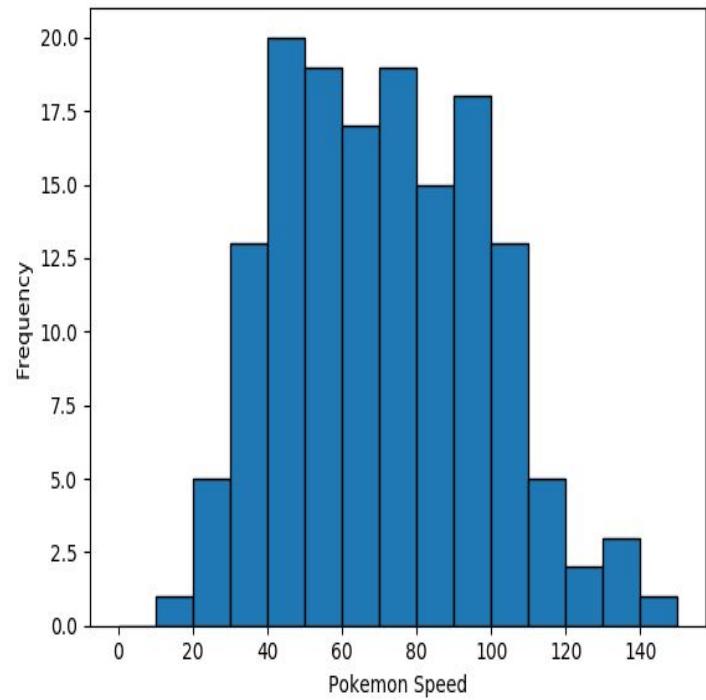


HKU

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

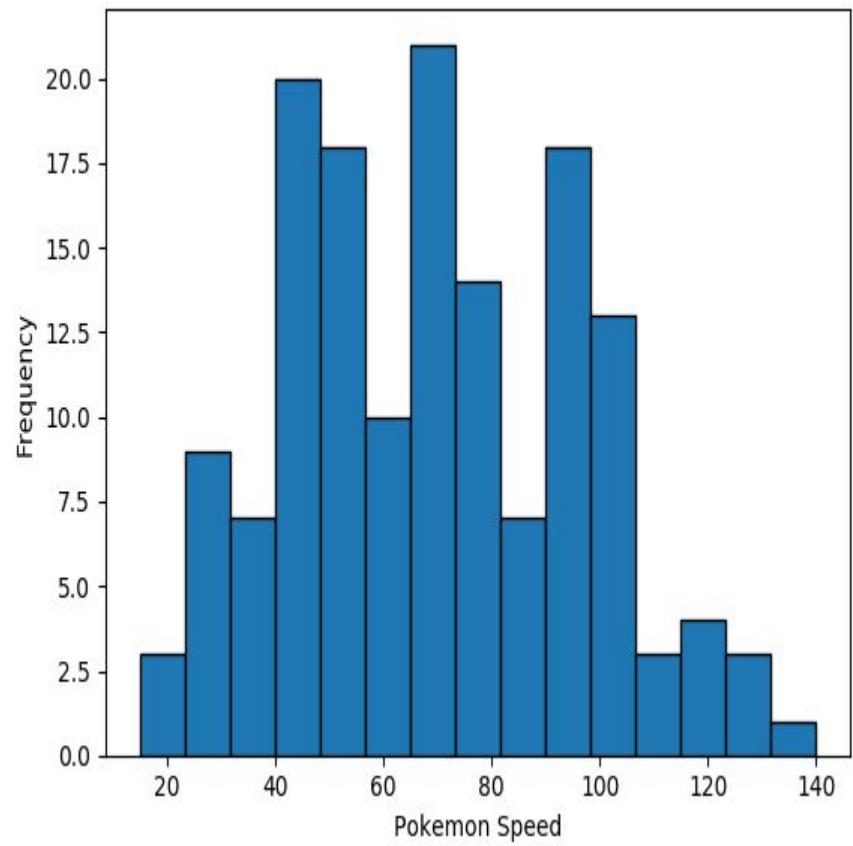
Here, I specified where the edges of the bars of the histogram are; the bin edges.

```
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.

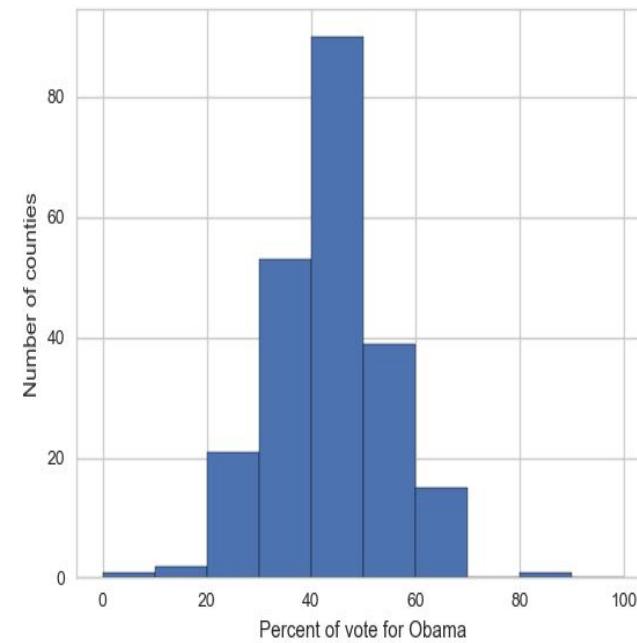
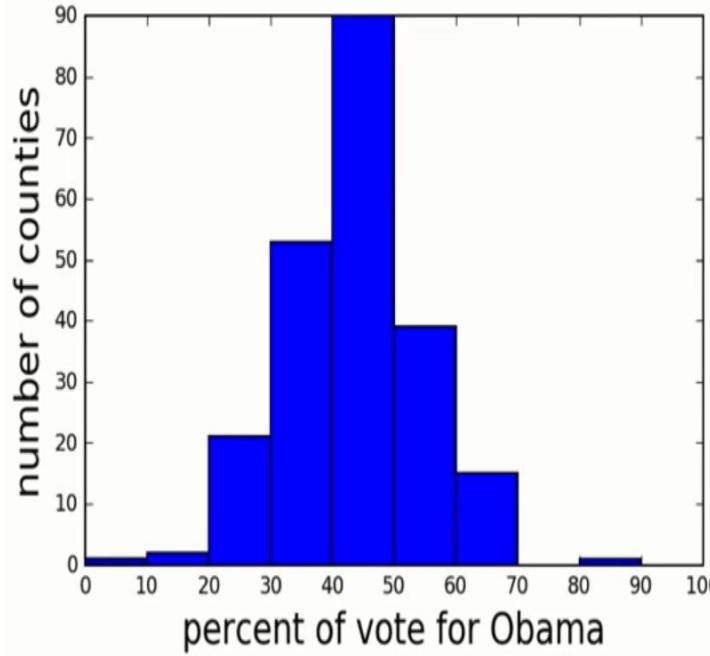
```
g = plt.hist(df1['Speed'], histtype='bar', ec='black', bins=15)
g = plt.xlabel('Pokemon Speed')
g = plt.ylabel('Frequency')
plt.show()
```



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.

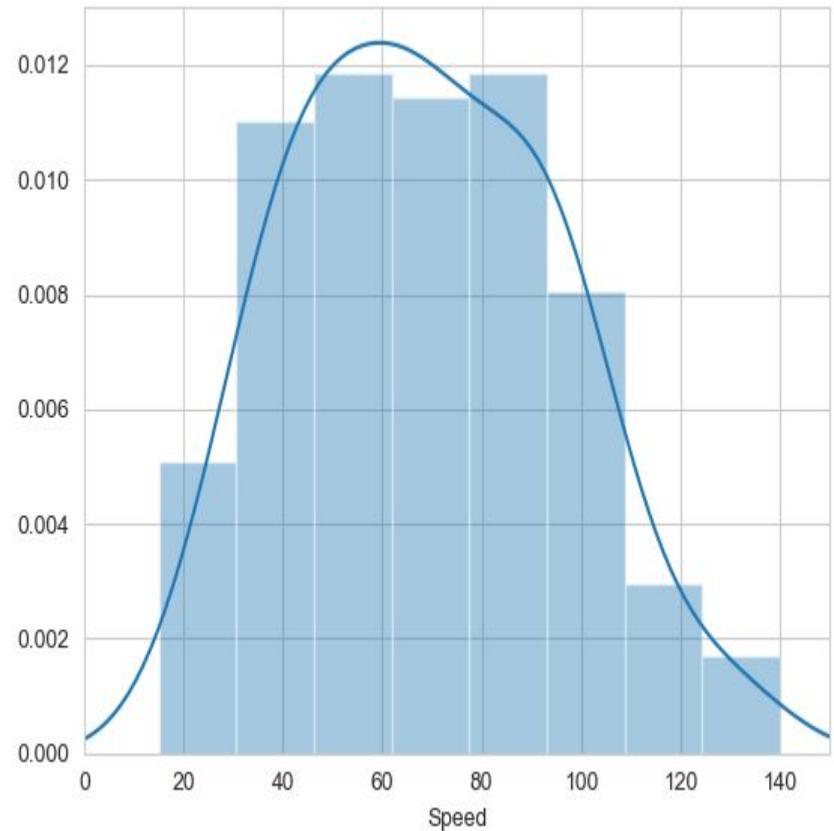
Plotting with Seaborn

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

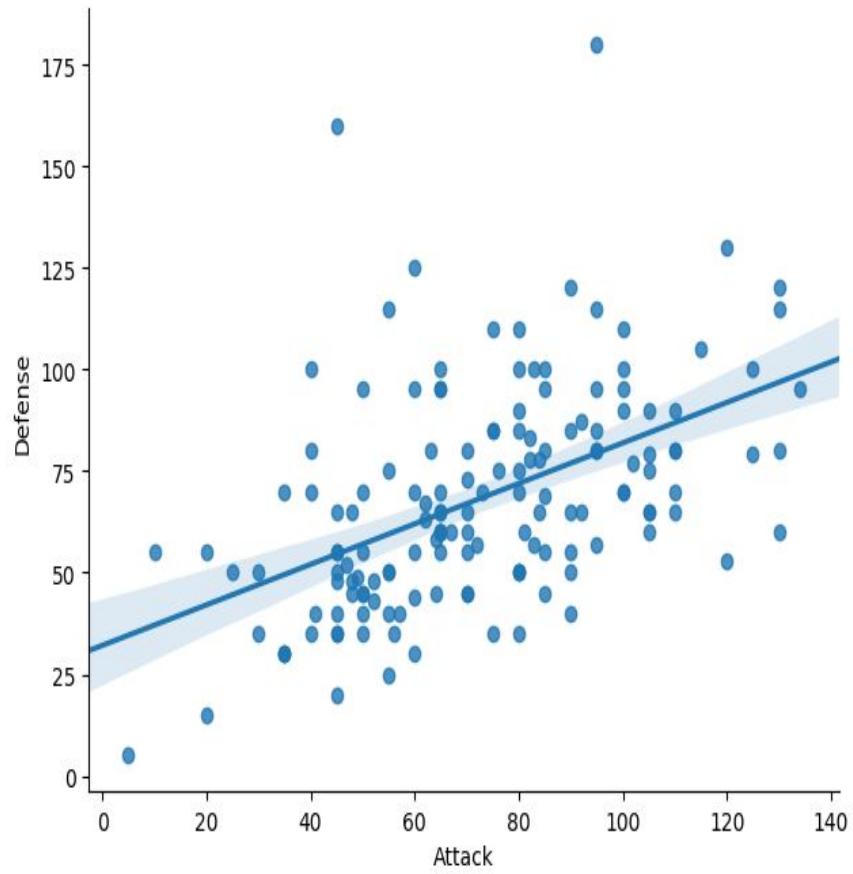
Name of variable we want on the x-axis

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



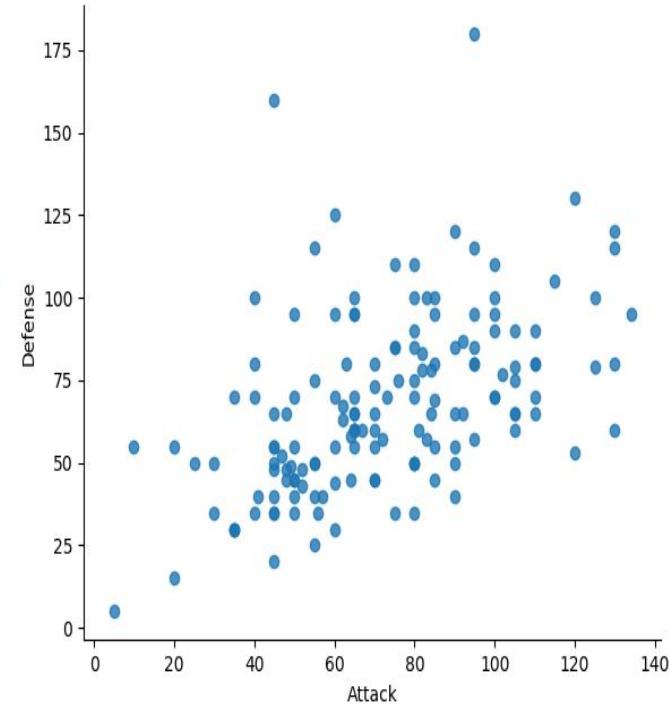
Seaborn doesn't have a dedicated scatter plot function.

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

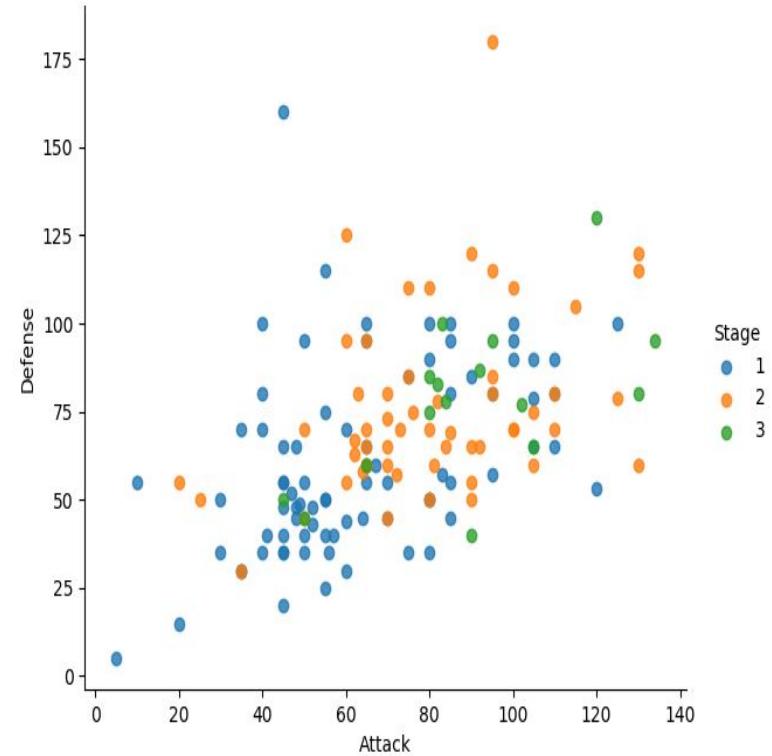
```
sns.lmplot(x='Attack', y='Defense', data=df1, fit_reg=False)
```



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)
```

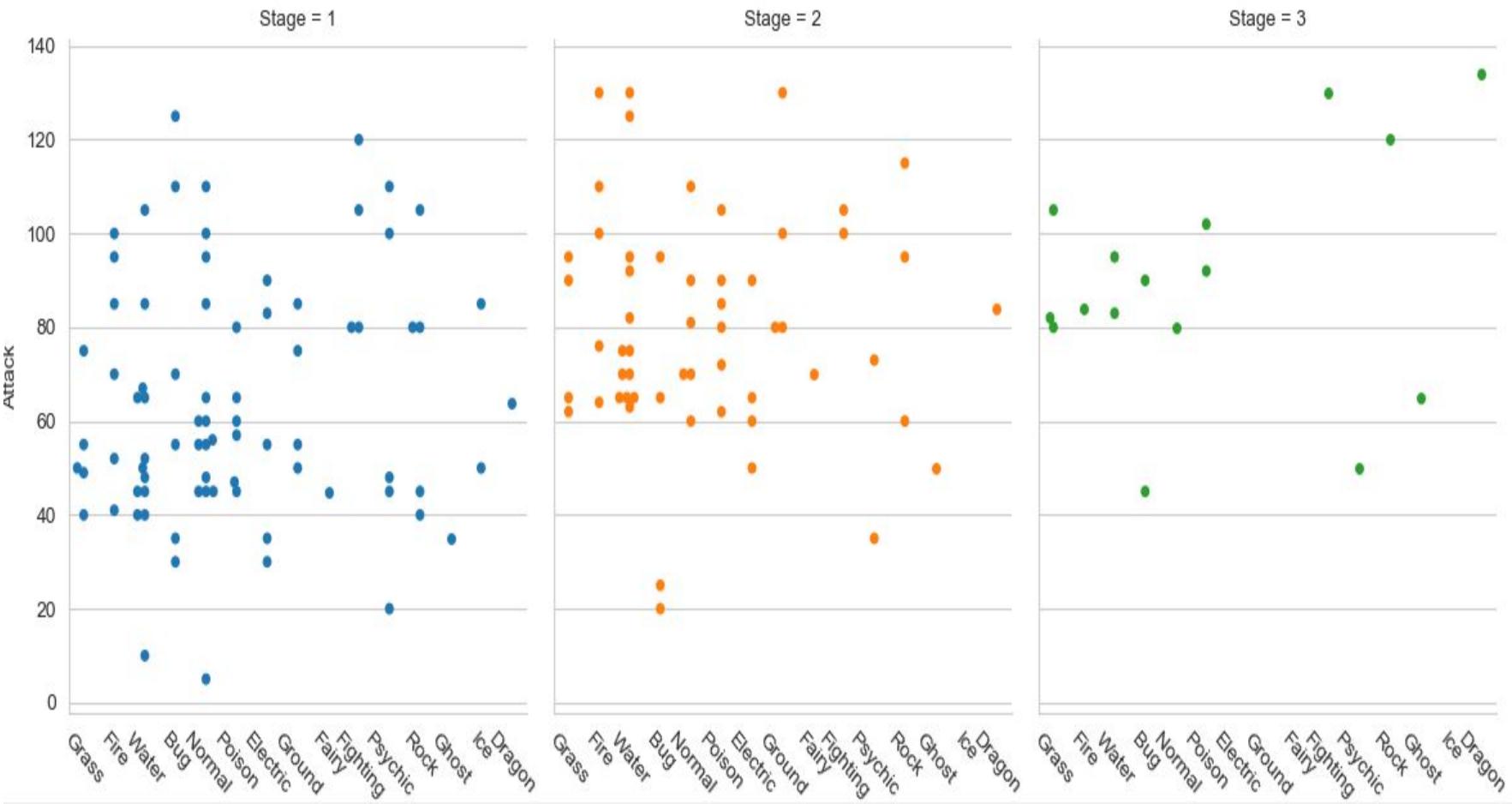
Colour by stage.

Separate by stage.

Generate using a swarmplot.

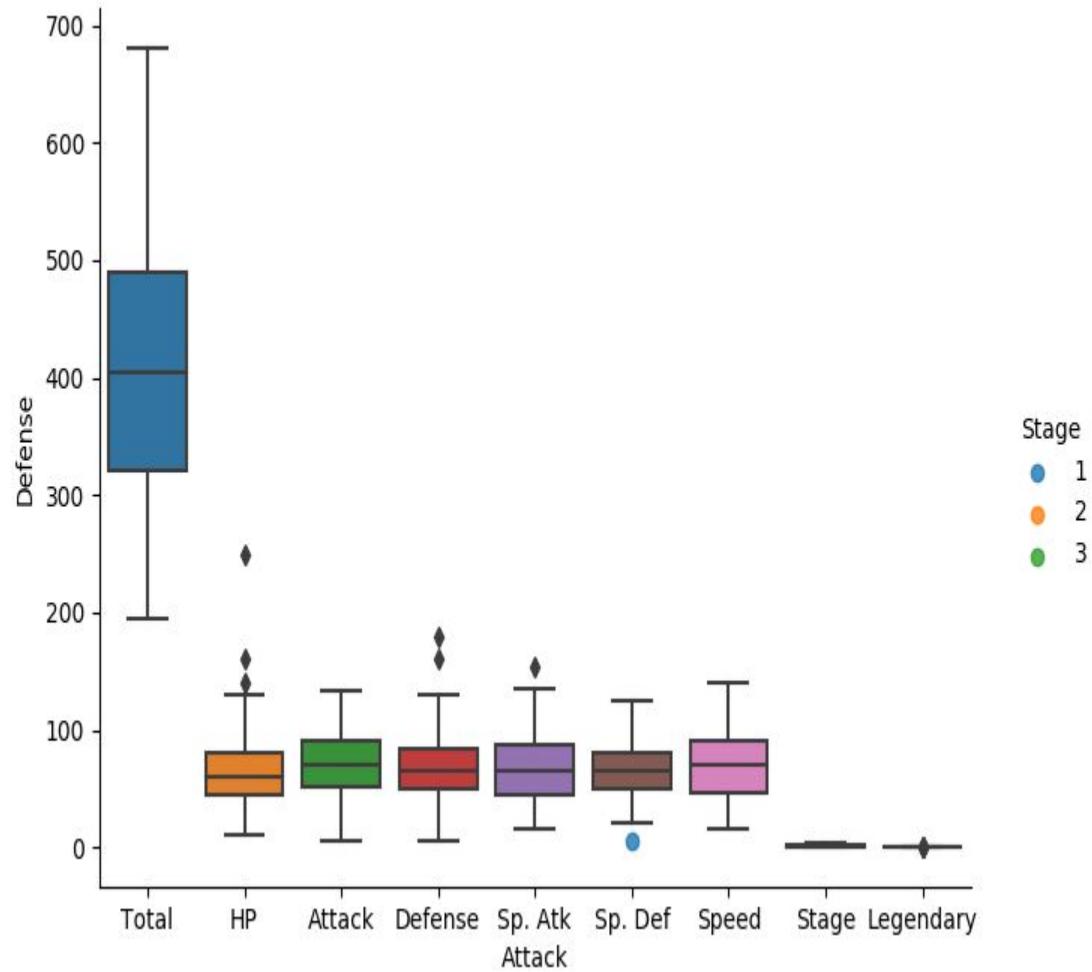
Rotate axis on x-ticks by 45 degrees.

Attack



A box plot

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

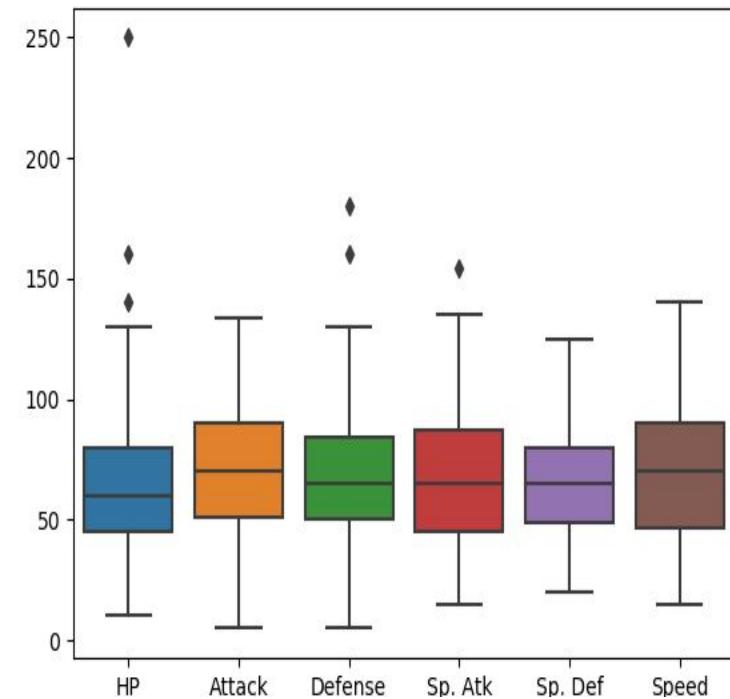


The total, stage, and legendary entries are not combat stats so we should remove them.

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.

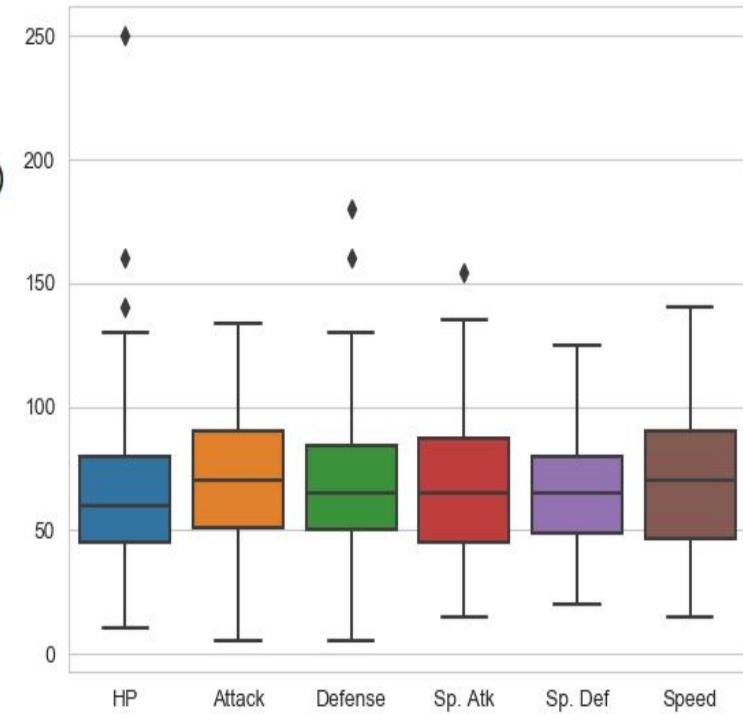
```
stats_df = df1.drop(['Total', 'Stage', 'Legendary'], axis=1)  
sns.boxplot(data=stats_df)
```



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)
```



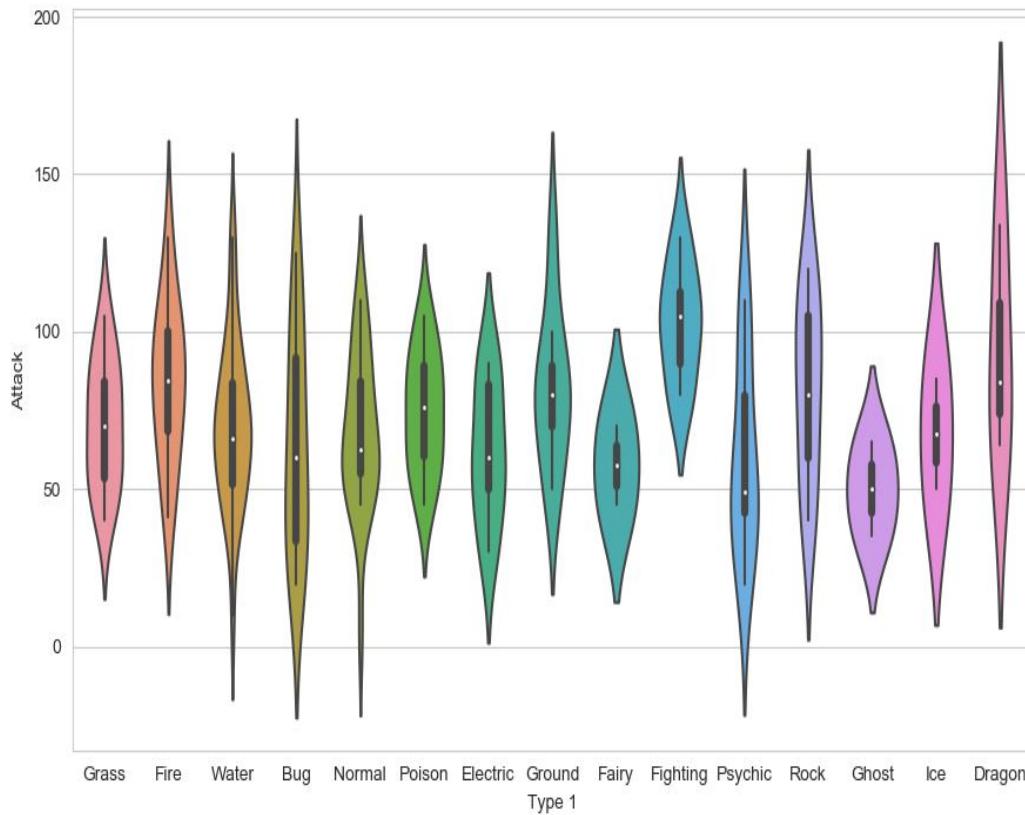
Violin plots

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

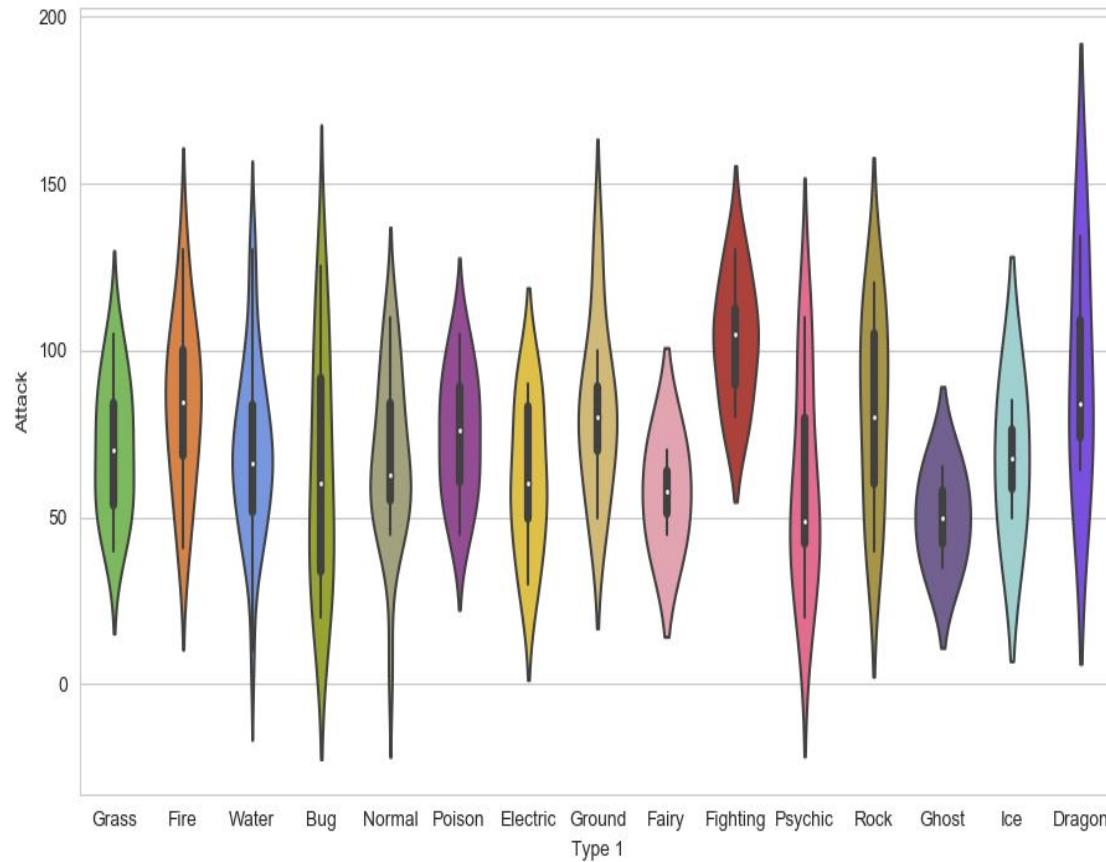
Seaborn allows us to easily set custom colour palettes by providing it with an ordered list of colour hex values.

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  
]
```

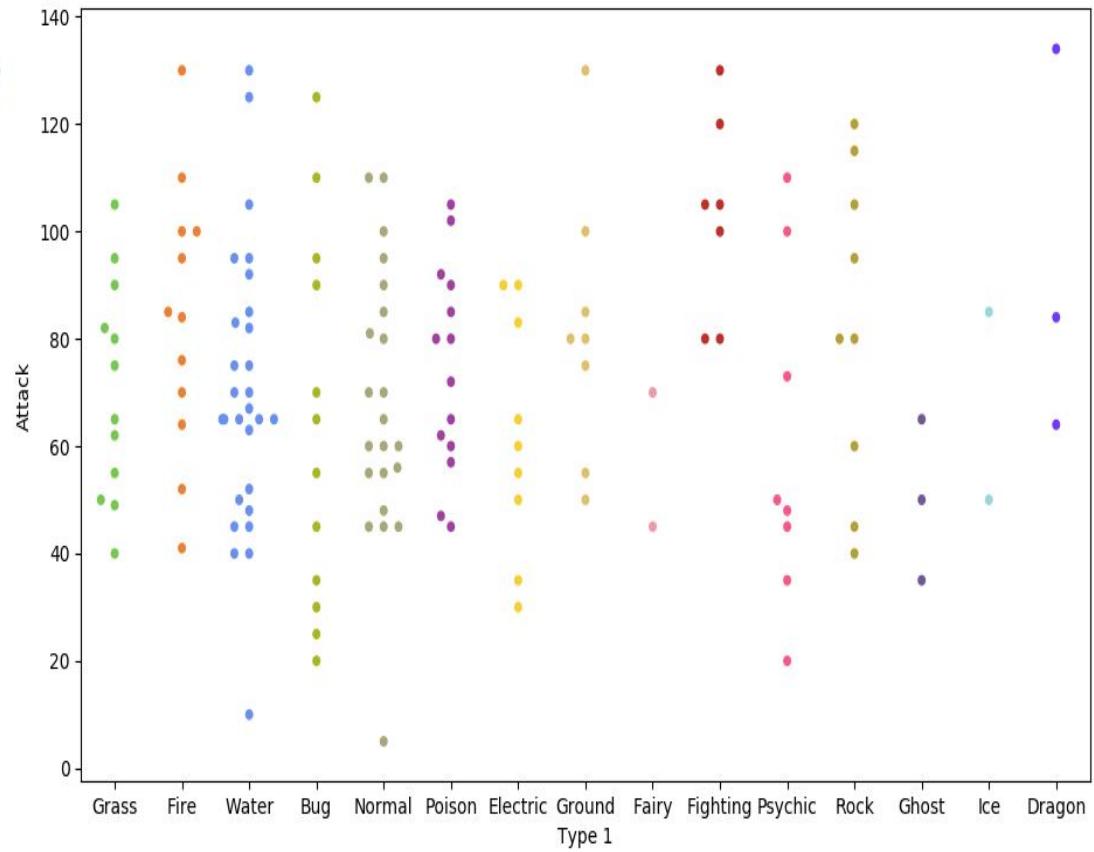
Then we just use the `palette=` function and feed in our colours list.

```
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)
```

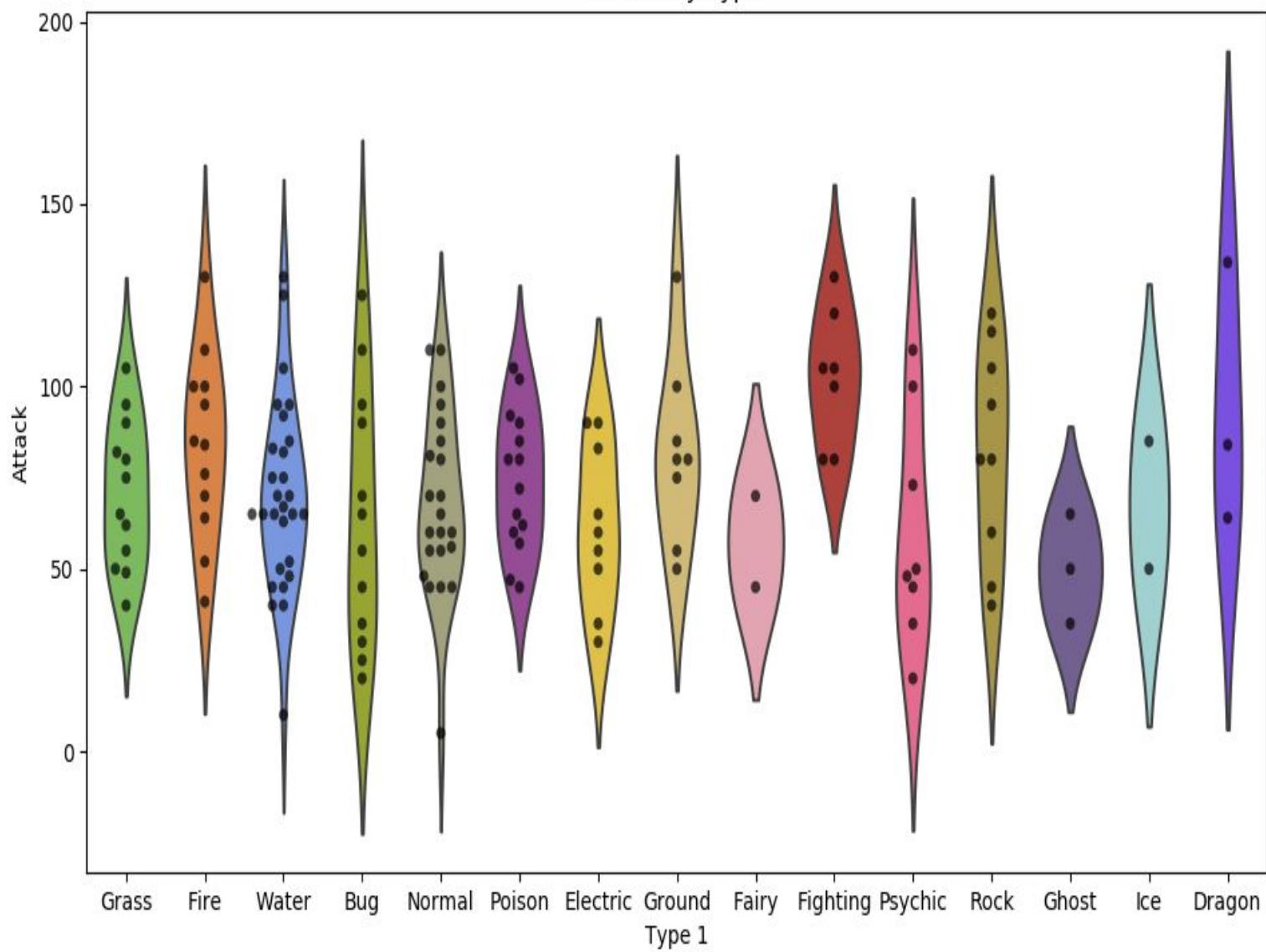


Overlapping plots

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

```
plt.figure(figsize=(10,6)) ← Set size of print canvas.  
sns.violinplot(x='Type 1',  
                 y='Attack',  
                 data=df1,  
                 inner=None, ← Remove bars from inside the violins  
                 palette=type_colors)  
sns.swarmplot(x='Type 1',  
              y='Attack',  
              data=df1,  
              color='k', ← Make bars black and slightly transparent  
              alpha=0.7)  
plt.title('Attack by Type') ← Give the graph a title
```

Attack by Type



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	45	1	False
2	Ivysaur	Grass	Poison	405	...	80	60	60	2	False
3	Venusaur	Grass	Poison	525	...	100	80	80	3	False
4	Charmander	Fire	NaN	309	...	50	65	65	1	False
5	Charmeleon	Fire	NaN	405	...	65	80	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.

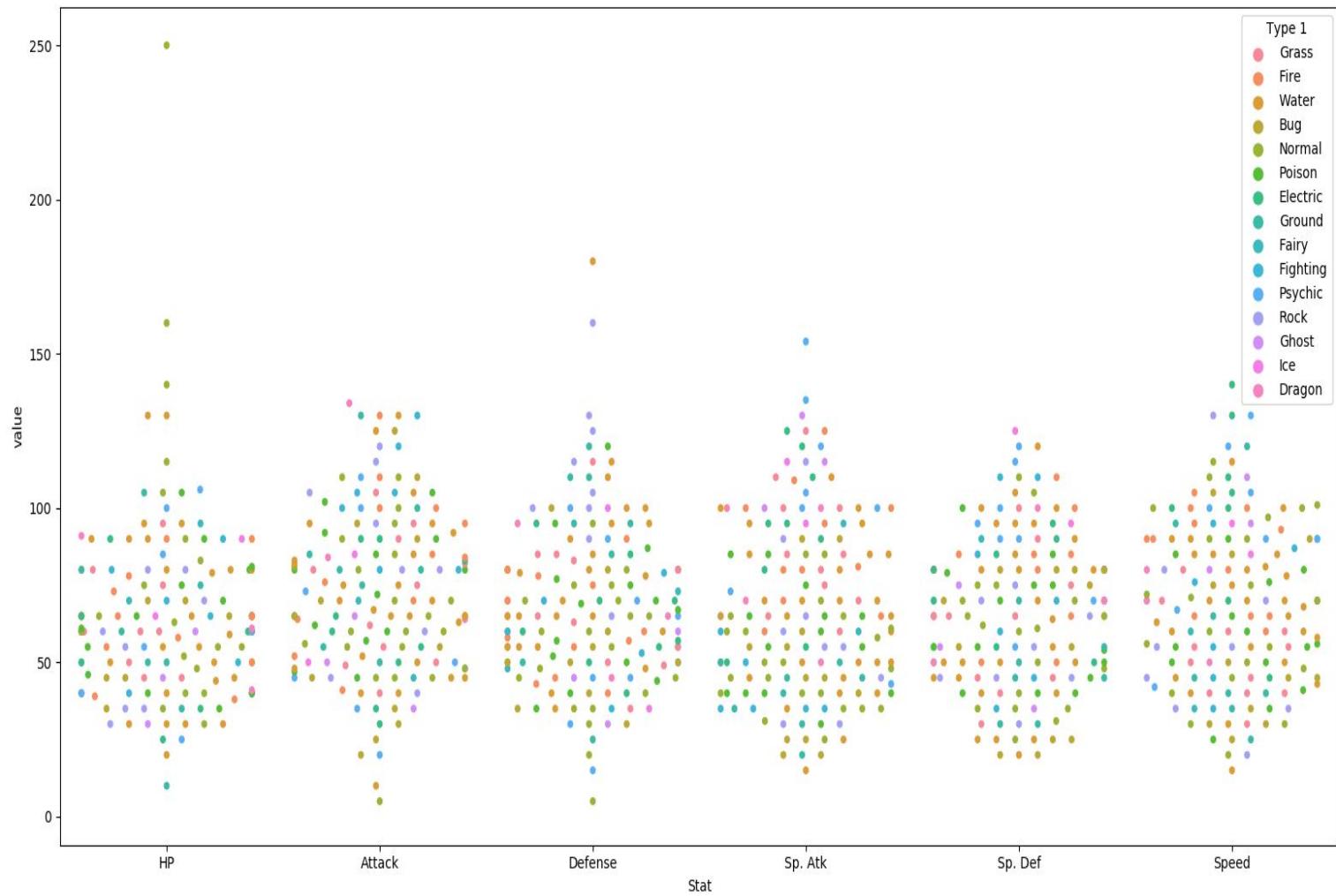
```
In [6]: runfile('C:/Users/lb690/Google Dri  
pokemon_tutorial.py', wdir='C:/Users/lb690')
```

	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.

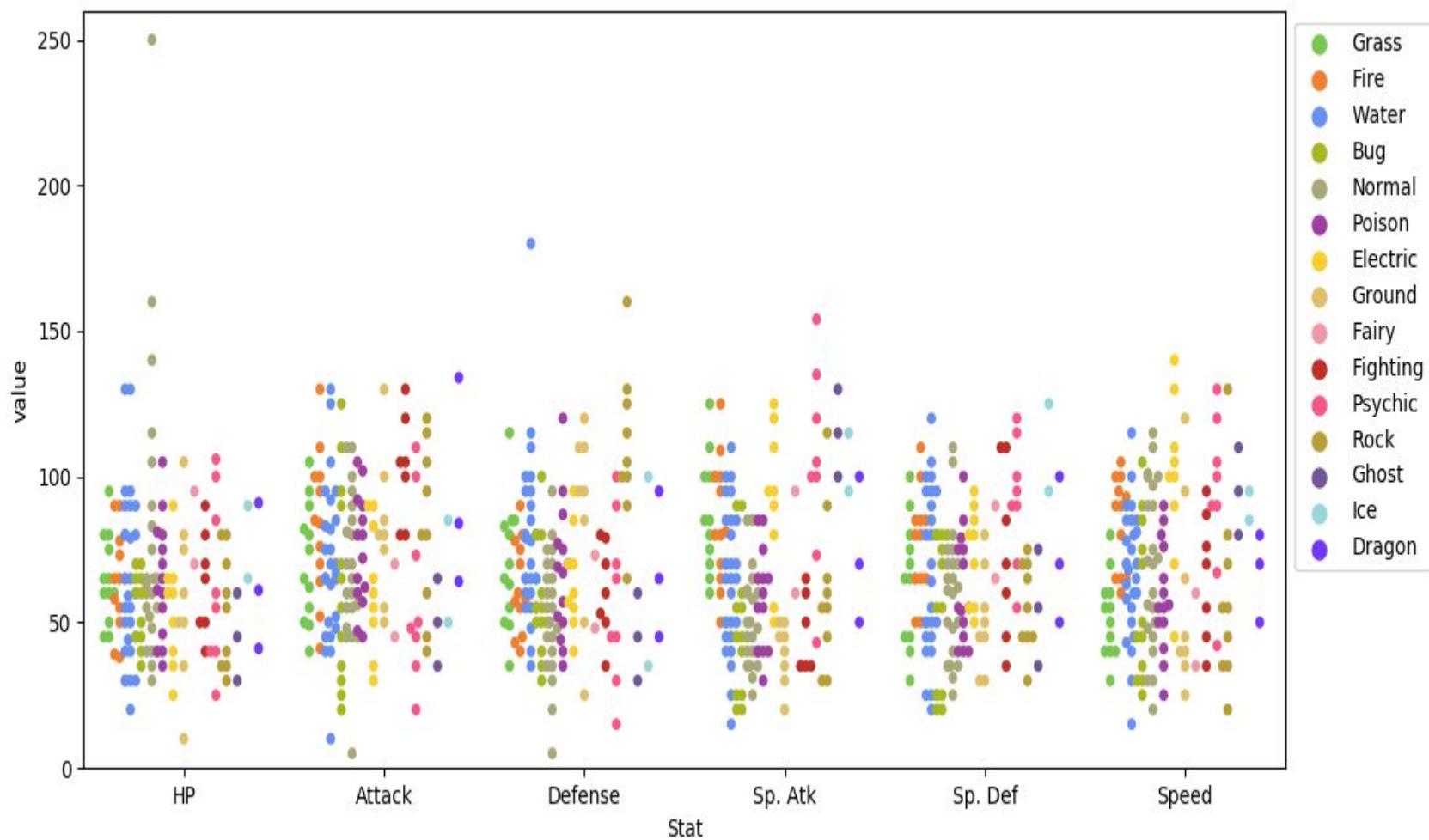
```
print( stats_df.shape )          (151, 9)
print( melted_df.shape )         (906, 5)
```

```
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)) ← Enlarge the plot.  
sns.swarmplot(x='Stat',  
               y='value',  
               data=melted_df,  
               hue='Type 1',  
               split=True,  
               palette=type_colors) ← Separate points by hue.  
plt.ylim(0, 260) ← Use our special Pokemon colour palette.  
plt.legend(bbox_to_anchor=(1, 1), loc=2) ← Adjust the y-axis.  
                                         Move the legend box outside  
                                         of the graph and place to the  
                                         right of it..
```



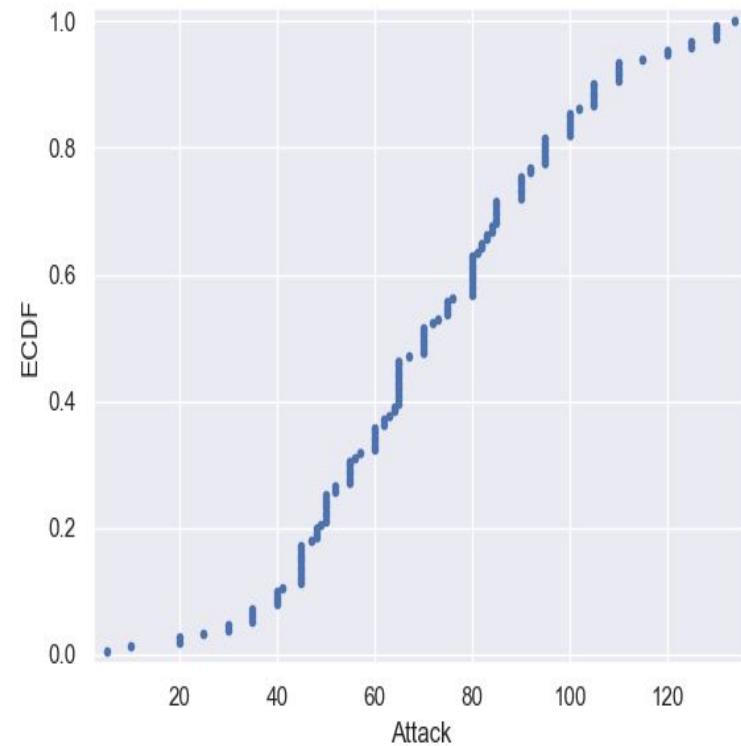
Plotting all data: Empirical cumulative distribution functions (ECDFs)

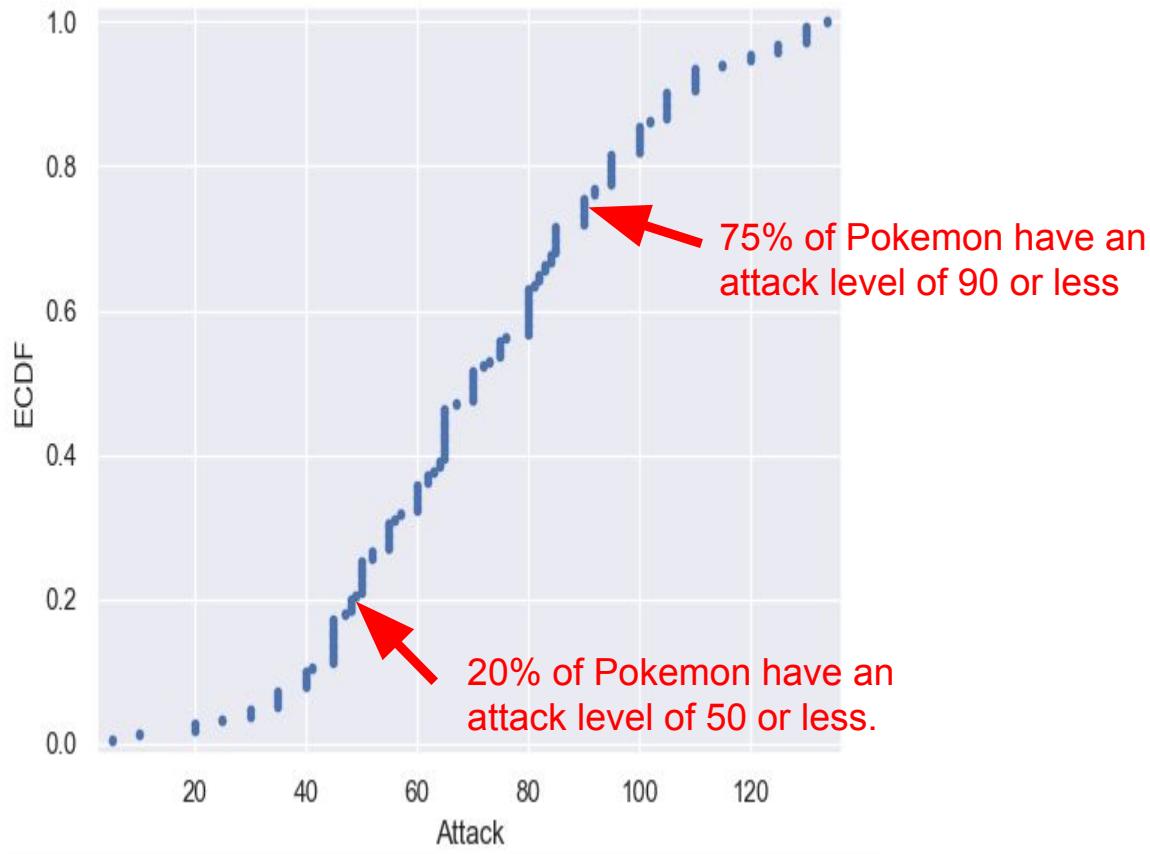
An alternative way of visualising a distribution of a variable in a large dataset is to use an ECDF.

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.

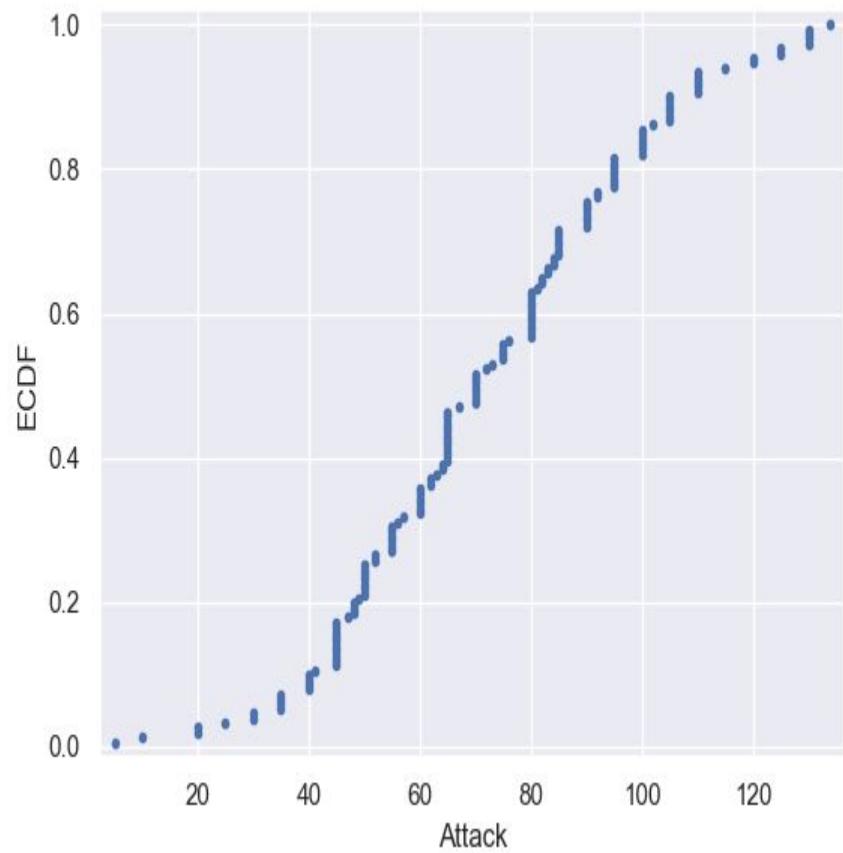
The *y-value* is the fraction of data points that have a value smaller than the corresponding x-value. For example...





Plotting an ECDF

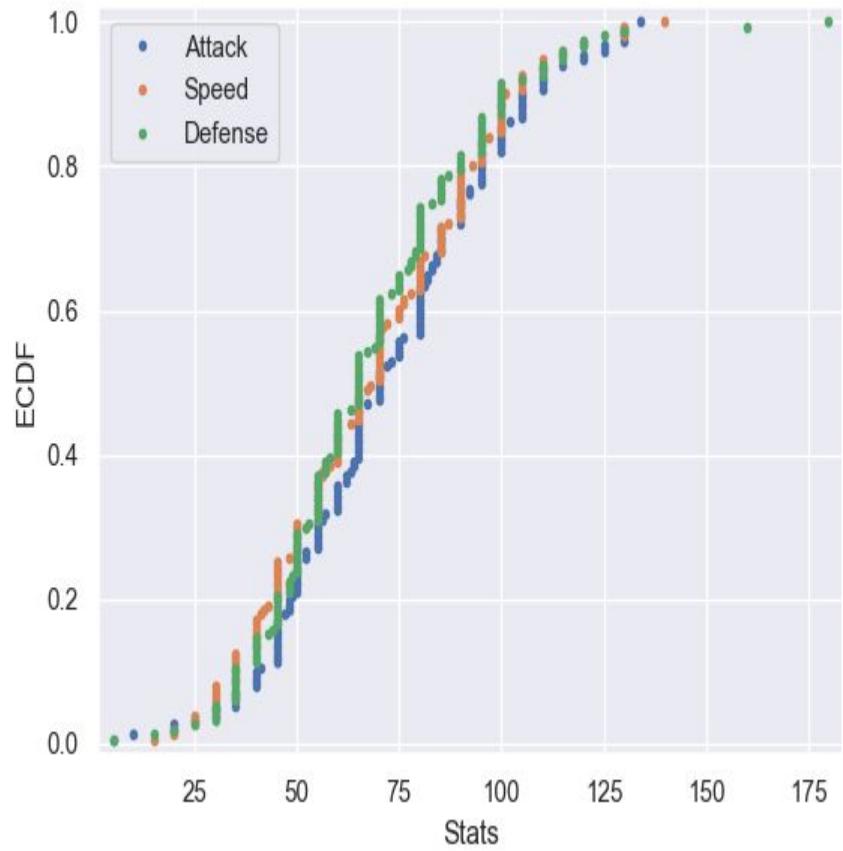
```
x = np.sort(df1['Attack'])
y=np.arange(1, len(x)+1)/len(x)
g = plt.plot(x, y, marker='.', linestyle='none')
g = plt.xlabel('Attack')
g = plt.ylabel('ECDF')
plt.margins(0.02)
plt.show()
```



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.

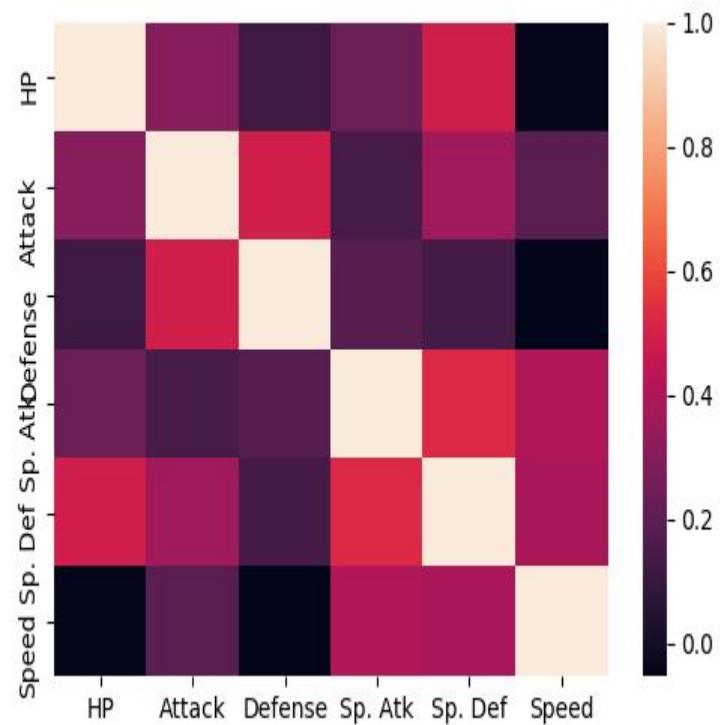
It shows all the data and gives a complete picture as to how the data are distributed.

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)
```

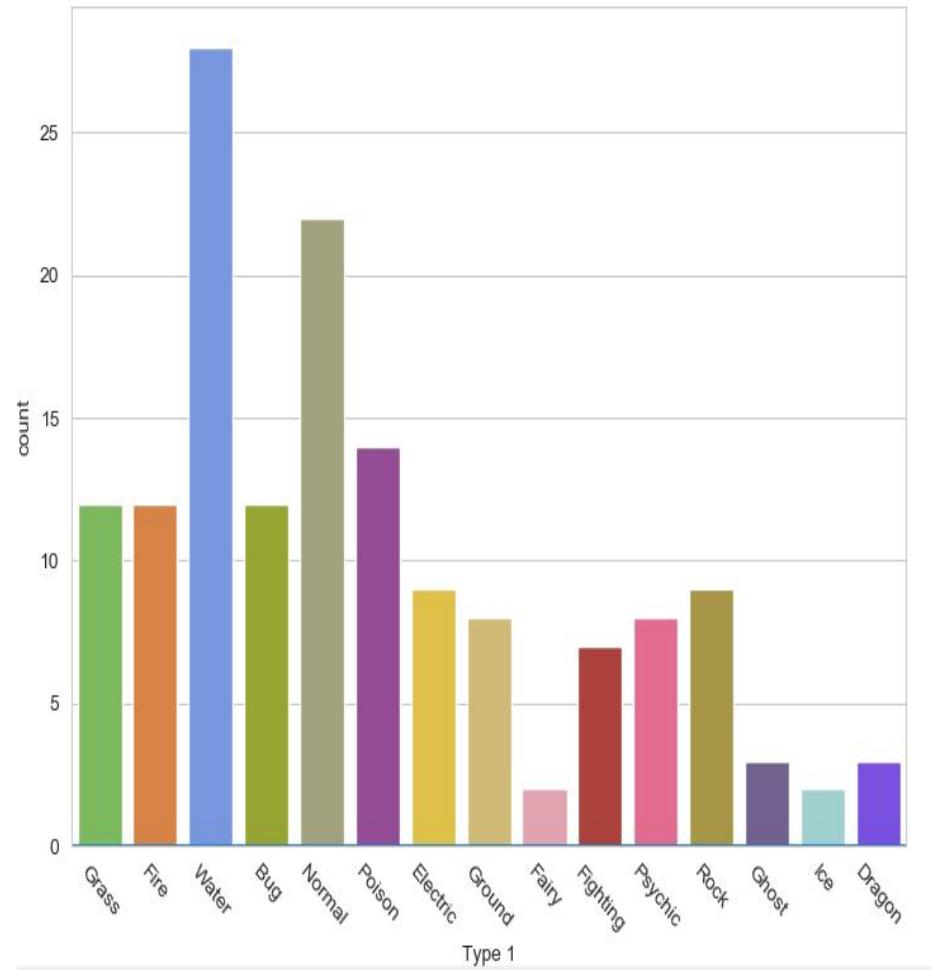


Bar plot

Visualises the distributions of categorical variables.

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

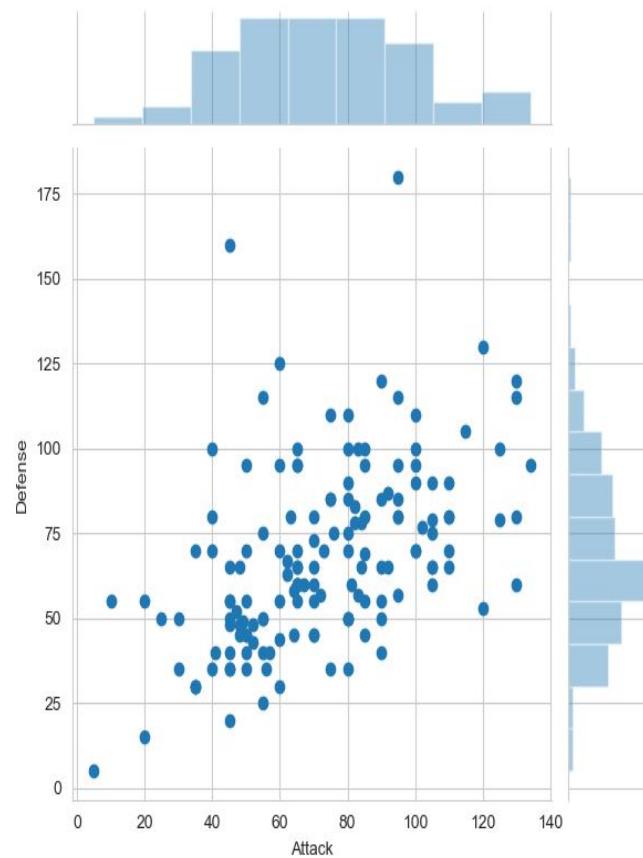
Rotates the
x-ticks 45
degrees



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)
```



Introduction of Python Framework

Flask - Flask is a **microframework** for Python. The main purpose is to develop a strong web application base. As compared to Django, Flask is best suited for **small and easy projects**.

Django - Django is a **high-level Python** Web application development framework that encourages us to develop things rapidly. It's free and open source.

Web2Py - It is a free open source full-stack development framework in python which allows the user to develop things quickly. It is a **cross-platform framework** that supports all popular operating systems.



Commonly used Flash Framework

Key features of Django



Authentication Security Object-relational mapping Templates Forms

django

Comparison between Django & Flask

 django	 Flask
Type of Framework	Full Stack Web Framework
Flexibility	Feature-packed
ORM Usage	Built-in ORM
Design	Batteries-included
Working Style	Monolithic
	WSGI framework
	Full flexibility
	SQLAlchemy is used
	Minimalistic design
	Diversified

What is Dash / Python Flask Framework?

- **Dash** is **Python framework** for building web applications. It is open source, and its app run on the web browser
- It built on top of **Flask**, Plotly.js, React and React Js.
- It enables you to build dashboards using pure **Python**.

What is Dash / Python Flask Framework?

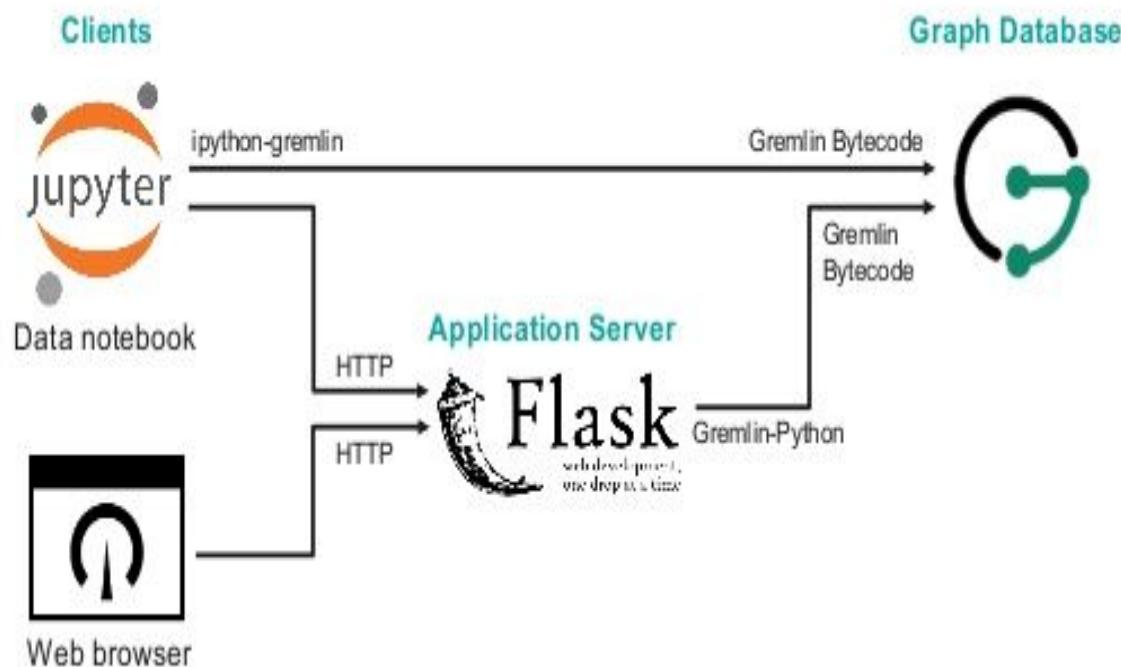
Every Dash App is composed of two main parts:

Layout = is used to define how all the content should be laid out on the application.

Callbacks = are used to make each of the desired components of the Dashboard interactive.

Python Flask Framework Architecture

Example Python Application Architecture





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

THANK YOU

