How To Use: Downloading Data

Tools, and resources used: Python Crash Course, Third Edition By Eric Matthes, Chat GPT 4.0

How it works: In order to read data properly you need to ensure you have the right library installed into your python directory. For these examples we used CSV and json format.

After you ensure you have CSV and Json in your python directory, you need to download data either from a local or online source.

Once you have the data you want to work with there are a couple of options you can use to switch the data from computer language into a format a human would be able to easily read and process.

One method is parsing or extracting CSV header positions which are allocated using numbers to specific arrays.

For example one array position could hold information we could potentially want to target so we need to ensure we target the right CSV header and then we could use a for loop to extract lots of information.

Once we have the appropriate data we need we can use matplotlib.pyplot to visualize the data.

Another method is using Json files instead of CSV files. It is very similar to CSV files in the sense that information is stored in arrays, but for json files we need to use the json.loads(contents) method to target specific arrays.

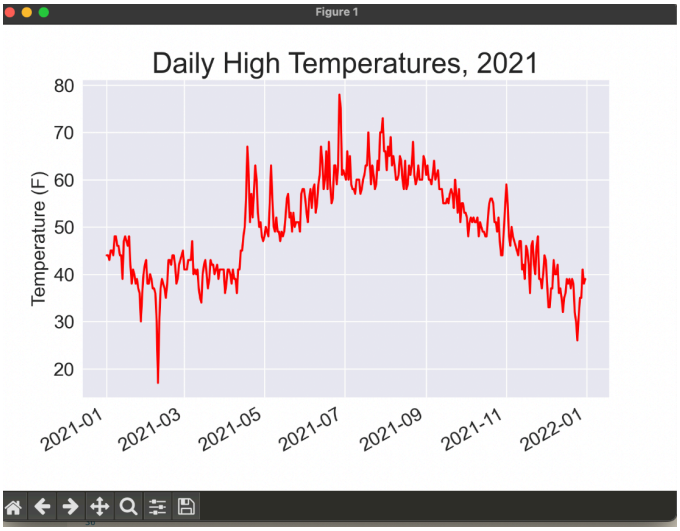
Here are some programs that utilize the the downloading data instructions and elements we discussed:

Sitka_highs.py:

```
# Import necessary libraries
from pathlib import Path  # For working with file paths
import csv  # For reading CSV files
from datetime import datetime  # For working with date and time data
import matplotlib.pyplot as plt  # For creating visualizations

# Define the path to the CSV file
# This specifies the location of the weather data file using an absolute path.
path = Path('/Users/joseochoa/Desktop/Python Coding Class/Projects/Downloading
Data/weather_data/sitka_weather_2021_simple.csv')

# Read the contents of the file and split it into lines
```

```
# Each line represents a row in the CSV file.
lines = path.read_text().splitlines()

# Create a CSV reader object to parse the lines
# This allows us to process the file as a CSV.
reader = csv.reader(lines)

# Extract the header row from the CSV file
# The header row contains the column names, which describe the data in each column.
header_row = next(reader)

# Extract dates and high temperatures from the CSV file
# Loop through the remaining rows in the CSV file.
dates, highs = [], []
for row in reader:
   current_date = datetime.strptime(row[2], '%Y-%m-%d')  # Parse the date string into
a datetime object.
   high = int(row[4])  # Convert the high temperature value to an integer.
   dates.append(current_date)  # Append the date to the `dates` list.
   highs.append(high)  # Append the high temperature to the `highs` list.

# Plot the high temperatures
# Use the seaborn style for better aesthetics.
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()

# Plot the high temperatures with dates on the x-axis and temperatures on the y-axis
ax.plot(dates, highs, color='red')

# Format the plot
# Add a title, x-axis label, and y-axis label for better readability.
ax.set_title("Daily High Temperatures, 2021", fontsize=24)  # Set the title of the
plot.
ax.set_xlabel('', fontsize=16)  # Leave the x-axis label blank.
fig.autofmt_xdate()  # Automatically format the x-axis dates for better readability.
ax.set_ylabel("Temperature (F)", fontsize=16)  # Label the y-axis with "Temperature
(F)".
ax.tick_params(labelsize=16)  # Customize the tick parameters for better readability.

# Display the plot
plt.show()
```

Output:



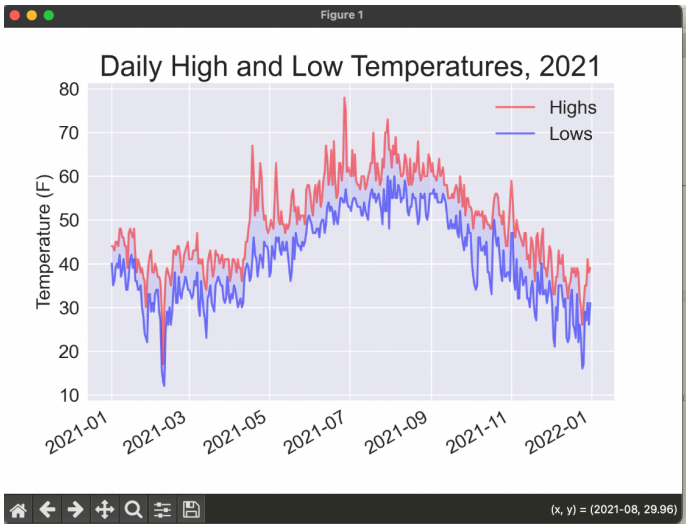
Sitka_highs_lows.py:

```
# Import necessary libraries
from pathlib import Path  # For working with file paths
import csv  # For reading CSV files
from datetime import datetime  # For working with date and time data
import matplotlib.pyplot as plt  # For creating visualizations

# Define the path to the CSV file
# This specifies the location of the weather data file using a relative path.
path = Path('/Users/joseochoa/Desktop/Python Coding Class/Projects/Downloading
Data/weather_data/sitka_weather_2021_simple.csv')

# Read the contents of the file and split it into lines
# Each line represents a row in the CSV file.
lines = path.read_text().splitlines()

# Create a CSV reader object to parse the lines
# This allows us to process the file as a CSV.
reader = csv.reader(lines)

# Extract the header row from the CSV file
# The header row contains the column names, which describe the data in each column.
header_row = next(reader)

# Extract dates, high temperatures, and low temperatures from the CSV file
```

```
# Loop through the remaining rows in the CSV file.
# Convert the date in the 3rd column (index 2) to a datetime object.
# Convert the high temperature in the 5th column (index 4) and low temperature in the
6th column (index 5) to integers.
# Append the extracted data to the `dates`, `highs`, and `lows` lists.
dates, highs, lows = [], [], []
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')  # Parse the date string into
a datetime object.
    high = int(row[4])  # Convert the high temperature value to an integer.
    low = int(row[5])  # Convert the low temperature value to an integer.
    dates.append(current_date)  # Append the date to the `dates` list.
    highs.append(high)  # Append the high temperature to the `highs` list.
    lows.append(low)  # Append the low temperature to the `lows` list.

# Plot the high and low temperatures
# Use the seaborn style for better aesthetics.
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()

# Plot the high temperatures in red and low temperatures in blue
# Use alpha for transparency to make the lines easier to read.
ax.plot(dates, highs, color='red', alpha=0.5, label='Highs')  # Plot high temperatures
with a red line.
ax.plot(dates, lows, color='blue', alpha=0.5, label='Lows')  # Plot low temperatures
with a blue line.

# Fill the area between the high and low temperatures
# Use a light blue color with transparency for better visualization.
ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

# Format the plot
# Add a title, x-axis label, and y-axis label for better readability.
ax.set_title("Daily High and Low Temperatures, 2021", fontsize=24)  # Set the title of
the plot.
ax.set_xlabel('', fontsize=16)  # Leave the x-axis label blank.
fig.autofmt_xdate()  # Automatically format the x-axis dates for better readability.
ax.set_ylabel("Temperature (F)", fontsize=16)  # Label the y-axis with "Temperature
(F)".
ax.tick_params(labelsize=16)  # Customize the tick parameters for better readability.

# Add a legend to differentiate between high and low temperatures
```

```
ax.legend(fontsize=16)

# Display the plot
plt.show()
```

Output:



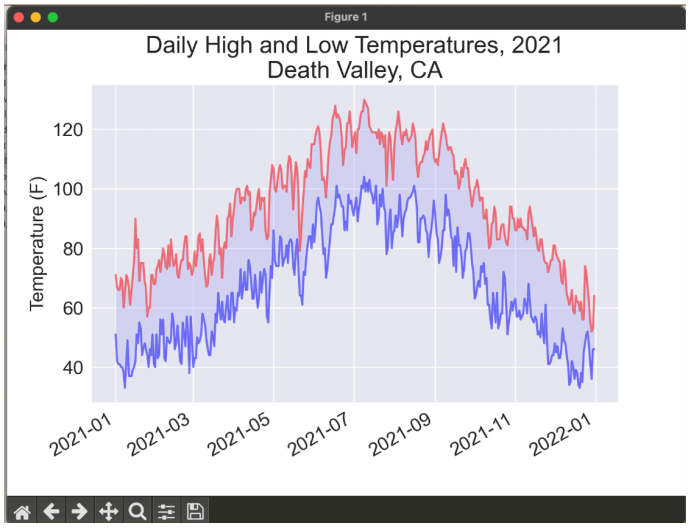
Death_valley_highs_lows.py:

```
# Import necessary libraries
from pathlib import Path  # For working with file paths
import csv  # For reading CSV files
from datetime import datetime  # For working with date and time data
import matplotlib.pyplot as plt  # For creating visualizations

# Define the path to the CSV file
# This specifies the location of the weather data file using an absolute path.
path = Path('/Users/joseochoa/Desktop/Python Coding Class/Projects/Downloading
Data/weather_data/death_valley_2021_simple.csv')

# Read the contents of the file and split it into lines
# Each line represents a row in the CSV file.
lines = path.read_text().splitlines()

# Create a CSV reader object to parse the lines
# This allows us to process the file as a CSV.
reader = csv.reader(lines)
```

```
# Extract the header row from the CSV file
# The header row contains the column names, which describe the data in each column.
header_row = next(reader)

# Extract dates, high temperatures, and low temperatures from the CSV file
# Loop through the remaining rows in the CSV file.
# Convert the date in the 3rd column (index 2) to a datetime object.
# Convert the high temperature in the 4th column (index 3) and low temperature in the
5th column (index 4) to integers.
# Handle missing data using a try-except block.
dates, highs, lows = [], [], []
for row in reader:
   current_date = datetime.strptime(row[2], '%Y-%m-%d')  # Parse the date string into
a datetime object.
   try:
       high = int(row[3])  # Convert the high temperature value to an integer.
       low = int(row[4])  # Convert the low temperature value to an integer.
   except ValueError:
       # Handle missing data by printing a message and skipping the row.
       print(f"Missing data for {current_date}")
   else:
       # Append valid data to the respective lists.
       dates.append(current_date)
       highs.append(high)
       lows.append(low)

# Plot the high and low temperatures
# Use the seaborn style for better aesthetics.
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()

# Plot the high temperatures in red and low temperatures in blue
# Use alpha for transparency to make the lines easier to read.
ax.plot(dates, highs, color='red', alpha=0.5, label='Highs')  # Plot high temperatures
with a red line.
ax.plot(dates, lows, color='blue', alpha=0.5, label='Lows')  # Plot low temperatures
with a blue line.

# Fill the area between the high and low temperatures
# Use a light blue color with transparency for better visualization.
ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)
```

```
# Format the plot
# Add a title, x-axis label, and y-axis label for better readability.
title = "Daily High and Low Temperatures, 2021\nDeath Valley, CA"  # Set the title of
the plot.
ax.set_title(title, fontsize=20)
fig.autofmt_xdate()  # Automatically format the x-axis dates for better readability.
ax.set_ylabel("Temperature (F)", fontsize=16)  # Label the y-axis with "Temperature
(F)".
ax.tick_params(labelsize=16)  # Customize the tick parameters for better readability.

# Display the plot
plt.show()
```

Output:

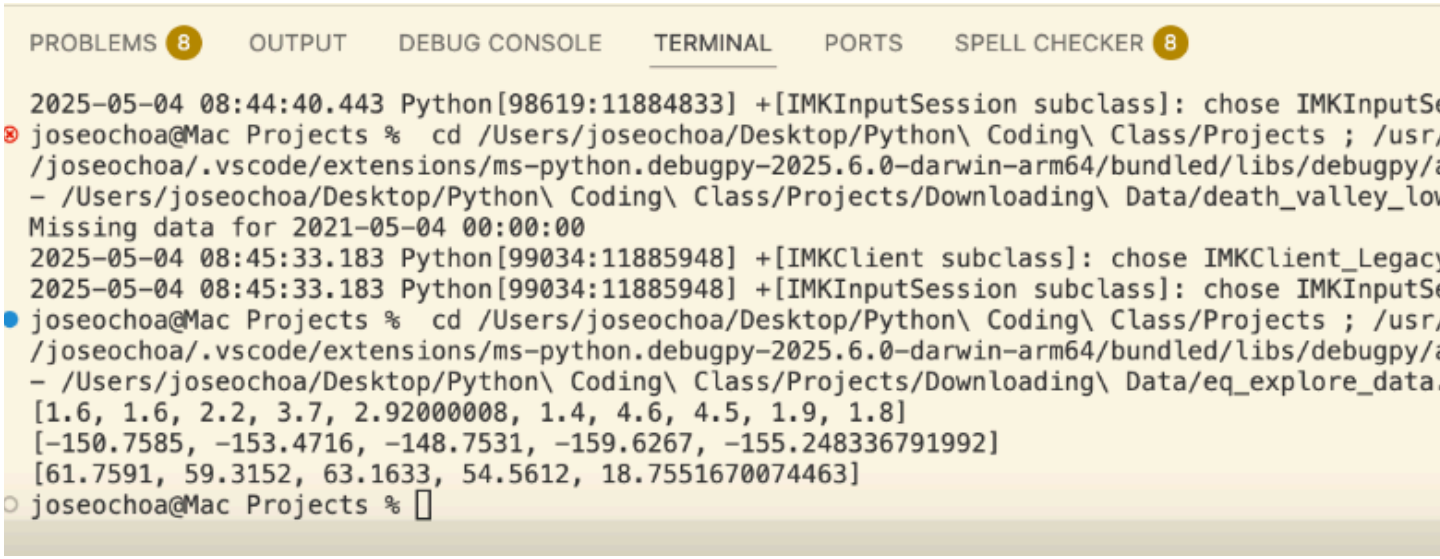


World_map_data.py:

```
from pathlib import Path
import json


# Read data as a string and convert to a Python object.
path = Path('/Users/joseochoa/Desktop/Python Coding Class/Projects/Downloading
Data/eq_data/eq_data_1_day_m1.geojson')
contents = path.read_text(encoding='utf-8')
all_eq_data = json.loads(contents)
```

```python
# Create more readable version of the data file.
path = Path('/Users/joseochoa/Desktop/Python Coding Class/Projects/Downloading
Data/eq_data/readable_eq_data.geojson')
readable_contents = json.dumps(all_eq_data, indent = 4)
path.write_text(readable_contents)

# Examine all earthquakes in the dataset.
all_eq_dicts = all_eq_data['features']

mags, lons, lats = [], [], []
for eq_dict in all_eq_dicts:
    mag = eq_dict['properties']['mag']
    lon = eq_dict['geometry']['coordinates'][0]
    lat = eq_dict['geometry']['coordinates'][1]
    mags.append(mag)
    lons.append(lon)
    lats.append(lat)

print(mags[:10])
print(lons[:5])
print(lats[:5])
```

Output



Eq_world_map.py:

```
# Import necessary libraries
from pathlib import Path  # For working with file paths
import json  # For working with JSON data
import plotly.express as px  # For creating visualizations

# Define the path to the earthquake data file
# This specifies the location of the GeoJSON file using an absolute path.
path = Path('/Users/joseochoa/Desktop/Python Coding Class/Projects/Downloading
Data/eq_data/eq_data_30_day_m1.geojson')

# Read the contents of the GeoJSON file
# The file is read as a string and then converted to a Python dictionary using
`json.loads`.
contents = path.read_text(encoding='utf-8')
all_eq_data = json.loads(contents)

# Extract earthquake data from the GeoJSON file
# The 'features' key contains a list of dictionaries, each representing an earthquake.
all_eq_dicts = all_eq_data['features']

# Initialize lists to store earthquake data
# These lists will hold magnitudes, longitudes, latitudes, and titles for each
earthquake.
mags, lons, lats, eq_titles = [], [], [], []

# Loop through each earthquake dictionary in the dataset
# Extract the magnitude, longitude, latitude, and title for each earthquake.
for eq_dict in all_eq_dicts:
    mag = eq_dict['properties']['mag']  # Extract the magnitude of the earthquake.
    lon = eq_dict['geometry']['coordinates'][0]  # Extract the longitude.
    lat = eq_dict['geometry']['coordinates'][1]  # Extract the latitude.
    eq_title = eq_dict['properties']['title']  # Extract the title of the earthquake.
    mags.append(mag)  # Append the magnitude to the `mags` list.
    lons.append(lon)  # Append the longitude to the `lons` list.
    lats.append(lat)  # Append the latitude to the `lats` list.
    eq_titles.append(eq_title)  # Append the title to the `eq_titles` list.

# Create a scatter plot of the earthquake data on a world map
# Use Plotly Express to create a scatter_geo plot with the extracted data.
```

```
title = 'Global Earthquakes'  # Set the title of the plot.
fig = px.scatter_geo(
    lat=lats,  # Latitude values for the earthquakes.
    lon=lons,  # Longitude values for the earthquakes.
    size=mags,  # Magnitudes determine the size of the markers.
    title=title,  # Title of the plot.
    color=mags,  # Magnitudes determine the color of the markers.
    color_continuous_scale='Viridis',  # Use the 'Viridis' color scale.
    labels={'color': 'Magnitude'},  # Label for the color scale.
    projection='natural earth',  # Use the 'natural earth' projection for the map.
    hover_name=eq_titles,  # Display the earthquake title when hovering over a marker.
)

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

Output: