# Module 6 – Python APIs

**Day 1, Activity 2 🡪 Space Launch (printing specific launch)**

# Dependencies

import requests

import json

# URL for GET requests to retrieve vehicle data

url = "https://api.spacexdata.com/v4/launchpads"

# Pretty print JSON for all launchpads

response = requests.get(url).json()

print(json.dumps(response, indent=4, sort\_keys=True))

# Pretty print JSON for a specific launchpad

response = requests.get(url + "/5e9e4502f509094188566f88").json()

print(json.dumps(response, indent=4, sort\_keys=True))

**Day 1, Activity 3 🡪 Instructor Manipulating Responses**

# Dependencies

import requests

import json

# Performing a GET Request and saving the

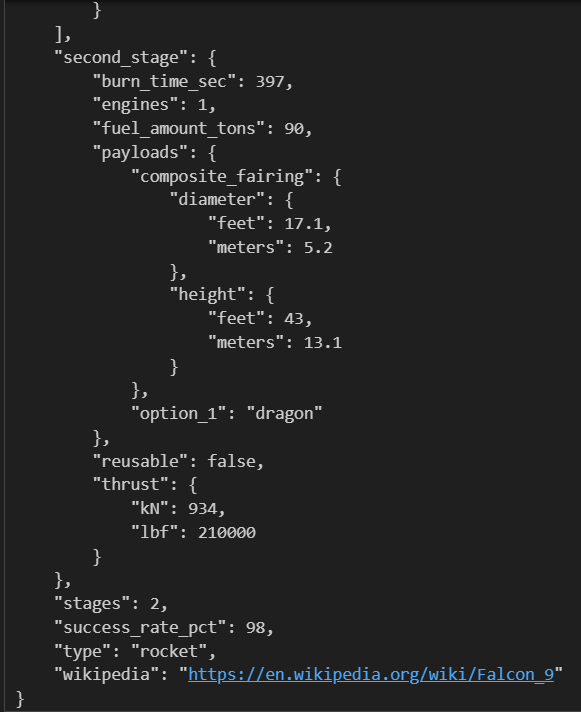
# API's response within a variable

url = "https://api.spacexdata.com/v4/rockets/5e9d0d95eda69973a809d1ec"

response = requests.get(url)

response\_json = response.json()

print(json.dumps(response\_json, indent=4, sort\_keys=True))

****

# It is possible to grab a specific value

# from within the JSON object

print(response\_json["cost\_per\_launch"])

****

# It is also possible to perform some

# analyses on values stored within the JSON object

number\_payloads = len(response\_json["payload\_weights"])

print(f"There are {number\_payloads} payloads.")

****

# Finally, it is possible to reference the

# values stored within sub-dictionaries and sub-lists

payload\_weight = response\_json["payload\_weights"][0]["kg"]

print(f"The first payload weighed {payload\_weight} Kilograms")

****

**Day 1, Activity 4 🡪 Student FarFarAway API Data**

# Dependencies

import requests

import json

# URL for GET requests to retrieve Star Wars character data

base\_url = "https://swapi.dev/api/people/"

# Create a url with a specific character id

character\_id = '4'

url = base\_url + character\_id

print(url)

<https://swapi.dev/api/people/4>

# Perform a get request for this character

response = requests.get(url)

print(response.url)

<https://swapi.dev/api/people/4>

# Storing the JSON response within a variable

data = response.json()

# Use json.dumps to print the json

print(json.dumps(data, indent=4, sort\_keys=True))

A computer screen shot of a program

Description automatically generated <https://swapi.dev/api/films/1/>",

<https://swapi.dev/api/films/2/>",

<https://swapi.dev/api/films/3/>",

<https://swapi.dev/api/films/6/>"

# Print the name of the character retrieved

character\_name = data["name"]

print(character\_name)

Darth Vader

# Print the number of films that they were in (hint: use len())

film\_number = len(data["films"])

print(film\_number)

4

# Request the starships URI found in the starships property of the

# previously retreived json, then use the response to figure out what this

# character's first starship was

first\_ship\_url = data["starships"][0]

ship\_response = requests.get(first\_ship\_url).json()

ship\_response

A computer screen shot of a program

Description automatically generated

# Print the name of the character's first starship

first\_ship = ship\_response["name"]

print(f"Their first ship: {first\_ship}")

Their first ship: TIE Advanced x1

# BONUS

films = []

for film in data['films']:

    cur\_film = requests.get(film).json()

    film\_title = cur\_film["title"]

    films.append(film\_title)

print(f"{character\_name} was in:")

print(films)

Darth Vader was in:

['A New Hope', 'The Empire Strikes Back', 'Return of the Jedi', 'Revenge of the Sith']

**Day 1, Activity 5 🡪 Par\_Number Facts API Application**

# Dependencies

import requests

import json

# Base URL for GET requests to retrieve number/date facts

url = "http://numbersapi.com/"

# Ask the user what kind of data they would like to search for

question = ("What type of data would you like to search for? "

            "[Trivia, Math, Date, or Year] ")

kind\_of\_search = input(question)

What type of data would you like to search for? [Trivia, Math, Date, or Year] Math

# If the kind of search is "date" take in two numbers

if(kind\_of\_search.lower() == "date"):

  # Collect the month to search for

  month = input("What month would you like to search for? ")

  # Collect the day to search for

  day = input("What day would you like to search for? ")

  # Make an API call to the "date" API and convert response object to JSON

  response = requests.get(f"{url}{month}/{day}/{kind\_of\_search.lower()}?json").json()

  # Print the fact stored within the response

  print(response["text"])

# If the kind of search is anything but "date" then take one number

else:

  # Collect the number to search for

  number = input("What number would you like to search for? ")

  # Make an API call to the API and convert response object to JSON

  response = requests.get(url + number + "/" +  kind\_of\_search.lower()+ "?json").json()

  # Print the fact stored within the response

  print(response["text"])

What number would you like to search for? 50

50 is the smallest number that can be written as the sum of of 2 squares in 2 ways.

**Day 1, Activity 6 🡪 Instructor OMDb Requests**

import requests

import json

from config import api\_key

# New Dependency! Use this to pretty print the JSON

# https://docs.python.org/3/library/pprint.html

from pprint import pprint

# Note that the ?t= is a query param for the t-itle of the

# movie we want to search for.

url = "http://www.omdbapi.com/?t="

api\_key = "&apikey=" + api\_key

# Performing a GET request similar to the one we executed

# earlier

response = requests.get(url + "Aliens" + api\_key)

# Converting the response to JSON, and printing the result.

data = response.json()

pprint(data)

A computer screen with white text

Description automatically generated

# Print a few keys from the response JSON.

print(f"Movie was directed by {data['Director']}.")

print(f"Movie was released in {data['Country']}.")

Movie was directed by James Cameron.

Movie was released in United Kingdom, United States.

**Day 1, Activity 7 🡪 Student Explore OMDb API**

**# OMDb API**

In this activity, you’ll review the OMDb API documentation, and you’ll practise using the API.

**## Instructions**

Read the OMDb [documentation](http://www.omdbapi.com/), and make a few API calls to get some information about your favourite movie.

- - -

© 2022 edX Boot Camps LLC. Confidential and Proprietary. All Rights Reserved.

**Need to git pull again / listen to video**

**Day 1, Activity 8 🡪 Student Movie Questions**

# Dependencies

import requests

from config import api\_key

url = f"http://www.omdbapi.com/?apikey={api\_key}&t="

# Who was the director of the movie Aliens?

movie = requests.get(url + "Aliens").json()

print(f'The director of Aliens was {movie["Director"]}.')

The director of Aliens was James Cameron.

# What was the movie Gladiator rated?

movie = requests.get(url + "Gladiator").json()

print(f'The rating of Gladiator was {movie["Rated"]}.')

The rating of Gladiator was R.

# What year was 50 First Dates released?

movie = requests.get(url + "50 First Dates").json()

print(f'The movie 50 First Dates was released in {movie["Year"]}.')

The movie 50 First Dates was released in 2004.

# Who wrote Moana?

movie = requests.get(url + "Moana").json()

print(f'Moana was written by {movie["Writer"]}.')

Moana was written by Jared Bush, Ron Clements, John Musker.

# What was the plot of the movie Sing?

movie = requests.get(url + "Sing").json()

print(f'The plot of Sing was: {movie["Plot"]}')

The plot of Sing was: In a city of humanoid animals, a hustling theater impresario's attempt to save his theater with a singing competition becomes grander than he anticipates even as its finalists find that their lives will never be the same.

**Day 1, Activity 9 🡪 Instructor Iterative Requests**

# Dependencies

import random

import json

import requests

# Let's get the JSON for 100 posts sequentially.

url = "http://jsonplaceholder.typicode.com/posts/"

# Create an empty list to store the responses

response\_json = []

# Create random indices representing

# a user's choice of posts

indices = random.sample(list(range(1, 100)), 10)

indices

[1, 95, 6, 27, 85, 11, 46, 72, 26, 4]

# Make a request for each of the indices

for x in range(len(indices)):

    print(f"Making request number: {x} for ID: {indices[x]}")

    # Get one of the posts

    post\_response = requests.get(url + str(indices[x]))

    # Save post's JSON

    response\_json.append(post\_response.json())

Making request number: 0 for ID: 1

Making request number: 1 for ID: 95

Making request number: 2 for ID: 6

Making request number: 3 for ID: 27

Making request number: 4 for ID: 85

Making request number: 5 for ID: 11

Making request number: 6 for ID: 46

Making request number: 7 for ID: 72

Making request number: 8 for ID: 26

Making request number: 9 for ID: 4

# Now we have 10 post objects,

# which we got by making 100 requests to the API.

print(f"We have {len(response\_json)} posts!"

We have 10 posts!

# Print JSON response

response\_json

**A computer screen with white text

Description automatically generated**

**Day 1, Activity 10 🡪 Student Movie Loop**

# Dependencies

import requests

from config import api\_key

url = "http://www.omdbapi.com/?apikey=" + api\_key + "&t="

# Make a list of movies

movies = ["Aliens", "Sing", "Moana"]

responses = []

# Make a request to the OMDb API for each movie in the list.

for movie in movies:

    movie\_data = requests.get(url + movie).json()

    # Print the director of each movie

    print(f'The director of {movie} is {movie\_data["Director"]}')

    # Save the responses in another list

    responses.append(movie\_data)

The director of Aliens is James Cameron

The director of Sing is Garth Jennings, Christophe Lourdelet

**A computer screen shot of a program

Description automatically generated**The director of Moana is Ron Clements, John Musker, Don Hall

**SJ GETS ERROR**

# Print the responses

responses

A screenshot of a computer

Description automatically generated

**Day 1, Activity 11 🡪 Instructor NYT API**

# Dependencies

import requests

from pprint import pprint

from config import api\_key

url = "https://api.nytimes.com/svc/search/v2/articlesearch.json?"

# Search for articles that mention granola

query = "granola"

# Build query URL

query\_url = url + "api-key=" + api\_key + "&q=" + query

# Request articles

articles = requests.get(query\_url).json()

# The "response" property in articles contains the actual articles

# list comprehension.

articles\_list = articles["response"]["docs"]

pprint(articles\_list)

**A computer screen shot of a program

Description automatically generated**

# Print the web\_url of each stored article

print("Your Reading List")

for article in articles\_list:

    print(article["web\_url"])

Your Reading List

<https://cooking.nytimes.com/recipes/1020980-strawberry-granola>

<https://www.nytimes.com/2012/03/25/magazine/who-made-that-granola.html>

<https://cooking.nytimes.com/recipes/1014414-granola>

<https://www.nytimes.com/2016/05/11/dining/granola-recipe-clusters-video.html>

<https://cooking.nytimes.com/recipes/1014040-granola>

<https://www.nytimes.com/2000/11/22/living/granola.html>

<https://www.nytimes.com/video/dining/1194817105861/making-granola.html>

<https://cooking.nytimes.com/recipes/1822-granola-muffins>

<https://dinersjournal.blogs.nytimes.com/2013/02/22/the-last-word-on-granola/>

<https://bitten.blogs.nytimes.com/2009/05/04/blogs/04bitten-homemade2111.html>

**Day 1, Activity 12 🡪 Student Retrieve Articles**

# Dependencies

import requests

from config import api\_key

import time

url = "https://api.nytimes.com/svc/search/v2/articlesearch.json?"

# Store a search term

query = "obama"

# Search for articles published between a begin and end date

begin\_date = "20160101"

end\_date = "20160130"

# Build URL

query\_url = f"{url}api-key={api\_key}&q={query}&begin\_date={begin\_date}&end\_date={end\_date}"

# Retrieve articles

articles = requests.get(query\_url).json()

articles\_list = articles["response"]["docs"]

for article in articles\_list:

    print(f'A snippet from the article: {article["snippet"]}')

    print('---------------------------')

**A black and white screen shot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**SJ GETS ERROR**

# BONUS: How would we get 30 results?

# HINT: Look up the page query param

# Empty list for articles

articles\_list = []

# loop through pages 0-2

for page in range(0, 3):

    query\_url = f"{url}api-key={api\_key}&q={query}&begin\_date={begin\_date}&end\_date={end\_date}"

    # create query with page number

    query\_url = f"{query\_url}&page={str(page)}"

    articles = requests.get(query\_url).json()

    # Add a one second interval between queries to stay within API query limits

    time.sleep(1)

    # loop through the response and append each article to the list

    for article in articles["response"]["docs"]:

        articles\_list.append(article)

    for article in articles\_list:

    print(article['snippet'])

    print('---------------------------')

**A screenshot of a computer screen

Description automatically generated**

**Day 2, Activity 1**

# Isolate "data items" for easy reading

data = video\_json["data"]

data\_items = data["items"]

# Retrieve the video's title

title = data\_items[0]["title"]

print("Title:", title)

Title: Introduction to Python Programming

# Retrieve the video's rating

rating = data\_items[0]["rating"]

print("Rating:", rating)

Rating: 2.35

# Retrieve the link to the video's first tag

tag\_one = data\_items[0]["tags"][0]

print("Tags:", tag\_one)

Tags: Python

# Retrieve the number of views this video has

view\_count = data\_items[0]["viewCount"]

print(f"View count: {view\_count}")

View count: 1350

**Day 2, Activity 2 – Requests Review**

# Dependencies

import json

import requests

from pprint import pprint

# Specify the URL

url = "https://static.bc-edx.com/data/dla-1-2/m6/lessons/2/request\_review.json"

# Make request and store response

response = requests.get(url)

# Verify status code

response.status\_code

200

# JSON-ify response

response\_json = response.json()

# Print first and last articles

pprint(f"The first response is {response\_json[0]}.")

pprint(f"The last response is {response\_json[-1]}.")

("The first response is {'id': 1, 'common\_name': 'Tiger cat', "

"'scientific\_name': 'Dasyurus maculatus', 'info': 'Curabitur at ipsum ac "

'tellus semper interdum. Mauris ullamcorper purus sit amet nulla. Quisque '

"arcu libero, rutrum ac, lobortis vel, dapibus at, diam.', 'location': "

"'Russia'}.")

("The last response is {'id': 1000, 'common\_name': 'Sloth, two-toed tree', "

"'scientific\_name': 'Choloepus hoffmani', 'info': 'Curabitur in libero ut "

'massa volutpat convallis. Morbi odio odio, elementum eu, interdum eu, '

'tincidunt in, leo. Maecenas pulvinar lobortis est.\\n\\nPhasellus sit amet '

"erat. Nulla tempus. Vivamus in felis eu sapien cursus vestibulum.', "

"'location': 'United States'}.")

# Print the number of responses

print(f"We received {len(response\_json)} responses.")

We received 1000 responses

**Day 2, Activity 3 – Open Weather Request**

# Dependencies

import json

import requests

from config import api\_key

# Save config information

url = "http://api.openweathermap.org/data/2.5/weather?"

city = "London"

# Build query URL

query\_url = url + "appid=" + api\_key + "&q=" + city

# Get weather data

weather\_response = requests.get(query\_url)

weather\_json = weather\_response.json()

# Get the temperature from the response

print(f"The weather API responded with: {weather\_json}.")

The weather API responded with: {'coord': {'lon': -0.1257, 'lat': 51.5085}, 'weather': [{'id': 804, 'main': 'Clouds', 'description': 'overcast clouds', 'icon': '04n'}], 'base': 'stations', 'main': {'temp': 279.59, 'feels\_like': 276.23, 'temp\_min': 278.1, 'temp\_max': 280.51, 'pressure': 1014, 'humidity': 81}, 'visibility': 10000, 'wind': {'speed': 5.14, 'deg': 240}, 'clouds': {'all': 100}, 'dt': 1673644432, 'sys': {'type': 2, 'id': 2075535, 'country': 'GB', 'sunrise': 1673596874, 'sunset': 1673626584}, 'timezone': 0, 'id': 2643743, 'name': 'London', 'cod': 200}.

**Day 2, Activity 4 – Burundi**

# Dependencies

import requests

from config import api\_key

# Save config information.

url = "http://api.openweathermap.org/data/2.5/weather?"

city = "Bujumbura"

units = "metric"

# Build query URL and request your results in Celsius

query\_url = f"{url}appid={api\_key}&q={city}&units={units}"

# Get weather data

weather\_response = requests.get(query\_url)

weather\_json = weather\_response.json()

# Get temperature from JSON response

temperature = weather\_json["main"]["temp"]

# Report temperature

print(f"The temperature in Bujumbura is {temperature} C.")

The temperature in Bujumbura is 21.08 C.

# BONUS

# use list of units

units = ["metric", "imperial"]

# set up list to hold two different temperatures

temperatures = []

# loop through the list of units and append them to temperatures list

for unit in units:

    # Build query URL based on current element in units

    query\_url = url + "appid=" + api\_key + "&q=" + city + "&units=" + unit

    # Get weather data

    weather\_response = requests.get(query\_url)

    weather\_json = weather\_response.json()

    # Get temperature from JSON response

    temperature = weather\_json["main"]["temp"]

    temperatures.append(temperature)

# Report temperatures by accessing each element in the list

print(

    f"The temperature in Bujumbura is {temperatures[0]}C or {temperatures[1]}F.")

The temperature in Bujumbura is 21.08C or 69.94F.

**Day 2, Activity 5 – Open Weather Data**

# Dependencies

import matplotlib.pyplot as plt

import requests

import pandas as pd

from config import api\_key

# Save config information.

url = "http://api.openweathermap.org/data/2.5/weather?"

units = "metric"

# Build partial query URL

query\_url = f"{url}appid={api\_key}&units={units}&q="

# List of cities

cities = ["Paris", "London", "Oslo", "Beijing"]

# set up lists to hold reponse info

lat = []

temp = []

# Loop through the list of cities and perform a request for data on each

for city in cities:

    response = requests.get(query\_url + city).json()

    lat.append(response['coord']['lat'])

    temp.append(response['main']['temp'])

print(f"The latitude information received is: {lat}")

print(f"The temperature information received is: {temp}")

The latitude information received is: [48.8534, 51.5085, 59.9127, 39.9075]

The temperature information received is: [7.52, 6.44, 2.08, -5.06]

# create a DataFrame from cities, lat, and temp

weather\_dict = {

    "city": cities,

    "lat": lat,

    "temp": temp

}

weather\_data = pd.DataFrame(weather\_dict)

weather\_data.head()

A screenshot of a black screen

Description automatically generated

# Build a scatter plot for each data type

plt.scatter(weather\_data["lat"], weather\_data["temp"], marker="o")

# Incorporate the other graph properties

plt.title("Temperature in World Cities")

plt.ylabel("Temperature (Celsius)")

plt.xlabel("Latitude")

plt.grid(True)

# Save the figure

plt.savefig("TemperatureInWorldCities.png")

# Show plot

plt.show()

**A graph with blue dots

Description automatically generated**

**Day 3, Activity 6 - TV Ratings**

#Dependencies

import requests

import json

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# list of TV show titles to query

tv\_shows = ["Wolf Like Me", "The Tourist", "Troppo", "Wentworth", "Farscape"]

# TV maze show search base URL

base\_url = "http://api.tvmaze.com/search/shows?q="

# set up lists to hold response data for name and rating

titles = []

ratings = []

# loop through TV show titles, make requests and parse

for show in tv\_shows:

    target\_url = base\_url + show

    response = requests.get(target\_url).json()

    titles.append(response[0]['show']['name'])

    ratings.append(response[0]['show']['rating']['average'])

**A screenshot of a black screen

Description automatically generated**

# create DataFrame

shows\_df = pd.DataFrame({

    "title": titles,

    "rating": ratings

})

shows\_df

# Use Pandas to create a bar chart from the DataFrame

shows\_df.plot('title', 'rating', kind='bar', figsize=(10,5), rot=45)

<AxesSubplot:xlabel='title'>

**A graph of a number of blue bars

Description automatically generated with medium confidence**

**Day 2, Activity 7 – Weather Stats**

# Dependencies

import matplotlib.pyplot as plt

import requests

from scipy import stats

import pandas as pd

from config import api\_key

# Save config information.

url = "http://api.openweathermap.org/data/2.5/weather?"

units = "metric"

# Build partial query URL

query\_url = f"{url}appid={api\_key}&units={units}&q="

# List of cities

cities = ["Paris", "London", "Oslo", "Beijing", "Mumbai", "Manila", "New York", "Seattle", "Dallas", "Taipei"]

# set up lists to hold reponse info

lat = []

temp = []

# Loop through the list of cities and perform a request for data on each

for city in cities:

    response = requests.get(query\_url + city).json()

    lat.append(response['coord']['lat'])

    temp.append(response['main']['temp'])

print(f"The latitude information received is: {lat}")

print(f"The temperature information received is: {temp}")

The latitude information received is: [48.8534, 51.5085, 59.9127, 39.9075, 19.0144, 14.6042, 40.7143, 47.6062, 32.7668, 25.0478]

The temperature information received is: [7.53, 6.44, 1.89, -5.06, 19.99, 23.89, 6.42, 10.89, 14.17, 20.09]

# create a data frame from cities, lat, and temp

weather\_dict = {

**A screenshot of a black screen

Description automatically generated**    "city": cities,

    "lat": lat,

    "temp": temp

}

weather\_data = pd.DataFrame(weather\_dict)

weather\_data

# Create a Scatter Plot for temperature vs latitude

**A graph with blue dots

Description automatically generated**x\_values = weather\_data['lat']

y\_values = weather\_data['temp']

plt.scatter(x\_values,y\_values)

plt.xlabel('Latitude')

plt.ylabel('Temperature')

plt.show()

# Perform a linear regression on temperature vs. latitude

(slope, intercept, rvalue, pvalue, stderr) = stats.linregress(x\_values, y\_values)

# Get regression values

regress\_values = x\_values \* slope + intercept

print(regress\_values)

0 5.434164

1 4.165065

2 0.147979

3 9.710174

4 19.696772

5 21.804784

6 9.324535

7 6.030307

8 13.123325

9 16.812895

Name: lat, dtype: float64

# Create line equation string

line\_eq = "y = " + str(round(slope,2)) + "x +" + str(round(intercept,2))

print(line\_eq)

y = -0.48x +28.79

# Create Plot

plt.scatter(x\_values,y\_values)

plt.plot(x\_values,regress\_values,"r-")

# Label plot and annotate the line equation

plt.xlabel('Latitude')

plt.ylabel('Temperature')

plt.annotate(line\_eq,(20,5),fontsize=15,color="red")

# Print r value

print(f"The r-value is: {rvalue\*\*2}")

# Show plot

plt.show()

**A red line with blue dots and numbers

Description automatically generated**The r-value is: 0.6192503127504022

# Calculate the temperature for Florence at 43.77 degrees

florence\_lat = 43.77

florence\_predicted\_temp = round(slope \* florence\_lat + intercept,2)

print(f"The Predicted temperature for Florence will be {florence\_predicted\_temp}.")

The Predicted temperature for Florence will be 7.86.

# Use API to determine actual temperature

response = requests.get(query\_url + "Florence").json()

florence\_actual\_temp = response['main']['temp']

print(f"The actual temperature of Florence is {florence\_actual\_temp}")

The actual temperature of Florence is 8.97

**Day 2, Activity 8 – Exception Handling**

students = {

    # Name  : Age

    "James": 27,

    "Sarah": 19,

    "Jocelyn": 28

}

print(students["Mary"])

print("This line will never print.")

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

/var/folders/1n/50wf\_pbs5bsb8\_\_bwdkxwvq86mm3js/T/ipykernel\_77676/95543560.py in <module>

**6** }

**7**

----> 8 print(students["Mary"])

**9**

**10** print("This line will never print.")

KeyError: 'Mary'

**Day 2, Activity 9 – Making Exceptions**

# Your assignment is to get the last line to print without changing any

# of the code below. Instead, wrap each line that throws an error in a

# try/except block.

try:

    print("Infinity looks like + " + str(10 / 0) + ".")

except ZeroDivisionError:

    print("Whoops. Can't do that.")

try:

    print("I think her name was + " + name + "?")

except NameError:

    print("Oh, I forgot to define 'name'. D'oh.")

try:

    print("Your name is a nonsense number. Look: " + int("Gabriel"))

except ValueError:

    print("Drat. 'Gabriel' isn't a number?")

print("I made it through the gauntlet. The message survived!")

Whoops. Can't do that.

Oh, I forgot to define 'name'. D'oh.

Drat. 'Gabriel' isn't a number?

I made it through the gauntlet. The message survived!

**Day 2, Activity 10 – api Exceptions**

import json

import requests

import pandas as pd

# List of characters

search\_characters = ['R2-D2', 'Darth Vader', 'Godzilla', 'Luke Skywalker', 'Frodo', \

              'Boba Fett', 'Iron Man', 'Jon Snow', 'Han Solo']

# Set url for API

url = 'https://swapi.dev/api/people/?search='

# Set empty lists to hold characters height and mass

height = []

mass = []

starwars\_characters = []

# Loop through each character

for character in search\_characters:

    # Create search query, make request and store in json

    query = url + character

    response = requests.get(query)

    response\_json = response.json()

    # Try to grab the height and mass of characters if they are available in the Star Wars API

    try:

        height.append(response\_json['results'][0]['height'])

        mass.append(response\_json['results'][0]['mass'])

        starwars\_characters.append(character)

        print(f"{character} found! Appending stats")

    # Handle exceptions for a character that is not available in the Star Wars API

    except:

        # Append null values

        print("Character not found")

        pass

R2-D2 found! Appending stats

Darth Vader found! Appending stats

Character not found

Luke Skywalker found! Appending stats

Character not found

Boba Fett found! Appending stats

Character not found

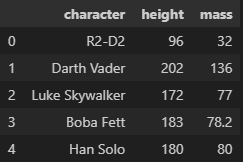
Character not found

Han Solo found! Appending stats

# Create DataFrame

character\_height = pd.DataFrame({

    'character': starwars\_characters,

****    'height': height,

    'mass': mass

})

character\_height

**Day 2, Activity 11 – World Bank API**

# Dependencies

import requests

url = "http://api.worldbank.org/v2/"

api\_format = "json"

# Get country information in JSON format

countries\_response = requests.get(f"{url}countries?format={api\_format}").json()

# First element is general information, second is countries themselves

countries = countries\_response[1]

# Report the names

for country in countries:

    print(country["name"])

Aruba

Africa Eastern and Southern

Afghanistan

Africa

Africa Western and Central

Angola

Albania

Andorra

Arab World

United Arab Emirates

Argentina

Armenia

American Samoa

Antigua and Barbuda

Australia

Austria

Azerbaijan

Burundi

East Asia & Pacific (IBRD-only countries)

Europe & Central Asia (IBRD-only countries)

Belgium

Benin

Burkina Faso

Bangladesh

Bulgaria

IBRD countries classified as high income

Bahrain

Bahamas, The

Bosnia and Herzegovina

Latin America & the Caribbean (IBRD-only countries)

Belarus

Belize

Middle East & North Africa (IBRD-only countries)

Bermuda

Bolivia

Brazil

Barbados

Brunei Darussalam

Sub-Saharan Africa (IBRD-only countries)

Bhutan

Botswana

Sub-Saharan Africa (IFC classification)

Central African Republic

Canada

East Asia and the Pacific (IFC classification)

Central Europe and the Baltics

Europe and Central Asia (IFC classification)

Switzerland

Channel Islands

Chile

**Day 2, Activity 12 – World Bank API – countries and Lending Type**

# Dependencies

import requests

url = "http://api.worldbank.org/v2/"

# Get the list of lending types the world bank has

lending\_response = requests.get(f"{url}lendingTypes?format=json").json()

lending\_types = [lending\_type["id"] for lending\_type in lending\_response[1]]

# Next, determine how many countries fall into each lending type.

# Hint: Look at the first element of the response array.

country\_count\_by\_type = {}

for lending\_type in lending\_types:

    query = f"{url}countries?lendingType={lending\_type}&format=json"

    response = requests.get(query).json()

    country\_count\_by\_type[lending\_type] = response[0]["total"]

# Print the number of countries of each lending type

for key, value in country\_count\_by\_type.items():

    print(f"The number of countries with lending type {key} is {value}.")

The number of countries with lending type IBD is 140.

The number of countries with lending type IDB is 30.

The number of countries with lending type IDX is 118.

The number of countries with lending type LNX is 74.

**Day 3, Activity 1 – Instructor Geoapify Geocode**

# Dependencies

import requests

import json

# Import the API key

from config import geoapify\_key

# Target city

target\_city = "Sydney, Australia"

# Build the endpoint URL

target\_url = f"https://api.geoapify.com/v1/geocode/search?text={target\_city}&format=json&apiKey={geoapify\_key}"

# Run a request to endpoint and convert result to json

geo\_data = requests.get(target\_url).json()

# Print the json

print(geo\_data)

{'results': [{'datasource': {'sourcename': 'openstreetmap', 'attribution': '© OpenStreetMap contributors', 'license': 'Open Database License', 'url': 'https://www.openstreetmap.org/copyright'}, 'country': 'Australia', 'country\_code': 'au', 'state': 'New South Wales', 'city': 'Sydney', 'municipality': 'Council of the City of Sydney', 'lon': 151.2082848, 'lat': -33.8698439, 'state\_code': 'NSW', 'formatted': 'Sydney, NSW, Australia', 'address\_line1': 'Sydney', 'address\_line2': 'NSW, Australia', 'category': 'administrative', 'timezone': {'name': 'Australia/Sydney', 'offset\_STD': '+10:00', 'offset\_STD\_seconds': 36000, 'offset\_DST': '+11:00', 'offset\_DST\_seconds': 39600, 'abbreviation\_STD': 'AEST', 'abbreviation\_DST': 'AEDT'}, 'result\_type': 'city', 'rank': {'importance': 0.8345908962989684, 'popularity': 7.332617625087506, 'confidence': 1, 'confidence\_city\_level': 1, 'match\_type': 'full\_match'}, 'place\_id': '512088e244aae662405903907f0b57ef40c0f00101f901f5bc570000000000c00208', 'bbox': {'lon1': 150.260825, 'lat1': -34.1732416, 'lon2': 151.343898, 'lat2': -33.3641864}}, {'datasource': {'sourcename': 'openstreetmap', 'attribution': '© OpenStreetMap contributors', 'license': 'Open Database License', 'url': 'https://www.openstreetmap.org/copyright'}, 'country': 'Australia', 'country\_code': 'au', 'state': 'New South Wales', 'city': 'Sydney', 'municipality': 'Council of the City of Sydney', 'postcode': '2000', 'suburb': 'Sydney', 'lon': 151.21013810170962, 'lat': -33.867926, 'state\_code': 'NSW', 'formatted': 'Sydney, NSW, Australia', 'address\_line1': 'Sydney', 'address\_line2': 'NSW, Australia', 'category': 'administrative', 'timezone': {'name': 'Australia/Sydney', 'offset\_STD': '+10:00', 'offset\_STD\_seconds': 36000, 'offset\_DST': '+11:00', 'offset\_DST\_seconds': 39600, 'abbreviation\_STD': 'AEST', 'abbreviation\_DST': 'AEDT'}, 'result\_type': 'suburb', 'rank': {'importance': 0.5748447123160637, 'popularity': 7.332617625087506, 'confidence': 1, 'confidence\_city\_level': 1, 'match\_type': 'full\_match'}, 'place\_id': '51904f8a73b9e66240598eacfc3218ef40c0f00101f901fe6c570000000000c00205', 'bbox': {'lon1': 151.1970047, 'lat1': -33.8797755, 'lon2': 151.223011, 'lat2': -33.8561096}}, {'datasource': {'sourcename': 'openstreetmap', 'attribution': '© OpenStreetMap contributors', 'license': 'Open Database License', 'url': 'https://www.openstreetmap.org/copyright'}, 'name': 'Sydney (Kingsford Smith) Airport', 'country': 'Australia', 'country\_code': 'au', 'state': 'New South Wales', 'county': 'Bayside Council', 'city': 'Sydney', 'postcode': '2019', 'district': 'Eastern Suburbs', 'suburb': 'Botany', 'street': 'McFall Street', 'lon': 151.18196819346016, 'lat': -33.9498935, 'state\_code': 'NSW', 'formatted': 'Sydney (Kingsford Smith) Airport, McFall Street, Botany NSW 2019, Australia', 'address\_line1': 'Sydney (Kingsford Smith) Airport', 'address\_line2': 'McFall Street, Botany NSW 2019, Australia', 'timezone': {'name': 'Australia/Sydney', 'offset\_STD': '+10:00', 'offset\_STD\_seconds': 36000, 'offset\_DST': '+11:00', 'offset\_DST\_seconds': 39600, 'abbreviation\_STD': 'AEST', 'abbreviation\_DST': 'AEDT'}, 'result\_type': 'amenity', 'rank': {'importance': 0.6078707058500873, 'popularity': 4.635071362519116, 'confidence': 1, 'confidence\_city\_level': 1, 'match\_type': 'inner\_part'}, 'place\_id': '515bfaf5aed2e56240596c97361c96f940c0f00102f9018617a00c00000000c002019203205379646e657920284b696e6773666f726420536d6974682920416972706f7274', 'bbox': {'lon1': 151.1614895, 'lat1': -33.9740292, 'lon2': 151.19555, 'lat2': -33.9259414}}, {'datasource': {'sourcename': 'openstreetmap', 'attribution': '© OpenStreetMap contributors', 'license': 'Open Database License', 'url': 'https://www.openstreetmap.org/copyright'}, 'name': 'Chinatown', 'country': 'Australia', 'country\_code': 'au', 'state': 'New South Wales', 'city': 'Sydney', 'municipality': 'Council of the City of Sydney', 'postcode': '2000', 'suburb': 'Haymarket', 'street': 'Goulburn Street', 'lon': 151.2040733935, 'lat': -33.87849425, 'state\_code': 'NSW', 'formatted': 'Chinatown, Goulburn Street, Haymarket NSW 2000, Australia', 'address\_line1': 'Chinatown', 'address\_line2': 'Goulburn Street, Haymarket NSW 2000, Australia', 'category': 'tourism.attraction', 'timezone': {'name': 'Australia/Sydney', 'offset\_STD': '+10:00', 'offset\_STD\_seconds': 36000, 'offset\_DST': '+11:00', 'offset\_DST\_seconds': 39600, 'abbreviation\_STD': 'AEST', 'abbreviation\_DST': 'AEDT'}, 'result\_type': 'amenity', 'rank': {'importance': 0.49265030002841015, 'popularity': 7.050568320159216, 'confidence': 1, 'confidence\_city\_level': 1, 'match\_type': 'inner\_part'}, 'place\_id': '511fe2ecc487e6624059aebce47f72f040c0f00102f9012309e31900000000c002019203094368696e61746f776e', 'bbox': {'lon1': 151.20339, 'lat1': -33.8794356, 'lon2': 151.2048011, 'lat2': -33.8775558}}, {'datasource': {'sourcename': 'openstreetmap', 'attribution': '© OpenStreetMap contributors', 'license': 'Open Database License', 'url': 'https://www.openstreetmap.org/copyright'}, 'name': "St Mary's Cathedral College, Sydney", 'country': 'Australia', 'country\_code': 'au', 'state': 'New South Wales', 'city': 'Sydney', 'municipality': 'Council of the City of Sydney', 'postcode': '2000', 'district': 'Koreatown', 'suburb': 'Sydney', 'street': 'Saint Marys Road', 'housenumber': '2', 'lon': 151.21438772912785, 'lat': -33.871456699999996, 'state\_code': 'NSW', 'formatted': "St Mary's Cathedral College, Sydney, 2 Saint Marys Road, Sydney NSW 2000, Australia", 'address\_line1': "St Mary's Cathedral College, Sydney", 'address\_line2': '2 Saint Marys Road, Sydney NSW 2000, Australia', 'category': 'education.school', 'timezone': {'name': 'Australia/Sydney', 'offset\_STD': '+10:00', 'offset\_STD\_seconds': 36000, 'offset\_DST': '+11:00', 'offset\_DST\_seconds': 39600, 'abbreviation\_STD': 'AEST', 'abbreviation\_DST': 'AEDT'}, 'result\_type': 'amenity', 'rank': {'importance': 0.4075805605530676, 'popularity': 6.845648842729769, 'confidence': 1, 'confidence\_city\_level': 1, 'match\_type': 'inner\_part'}, 'place\_id': '5192a8a743dce6624059a630a5e48bef40c0f00102f9018fce3f2c00000000c002019203235374204d61727927732043617468656472616c20436f6c6c6567652c205379646e6579', 'bbox': {'lon1': 151.2135968, 'lat1': -33.8719009, 'lon2': 151.2145886, 'lat2': -33.8710113}}], 'query': {'text': 'Sydney, Australia', 'parsed': {'city': 'sydney', 'country': 'australia', 'expected\_type': 'city'}}}

# Print the json (pretty printed)

**A screenshot of a computer screen

Description automatically generated**print(json.dumps(geo\_data, indent=4, sort\_keys=True))

"url": "<https://www.openstreetmap.org/copyright>"

# Extract latitude and longitude

lat = geo\_data["results"][0]["lat"]

lon = geo\_data["results"][0]["lon"]

# Print the latitude and longitude

print('''

    City: {0}

    Latitude: {1}

    Longitude: {2}

    '''.format(target\_city, lat, lon))

City: Sydney, Australia

Latitude: -33.8698439

Longitude: 151.2082848

**{'statusCode': 401, 'error': 'Unauthorized', 'message': 'Invalid apiKey'}**

**{**

**"error": "Unauthorized",**

**"message": "Invalid apiKey",**

**"statusCode": 401**

**FOR SJ – NEED to resolve**

**Day 3, Activity 2 – Instructor Geoapify Places**

# Dependencies

import requests

import json

# Import the API key

from config import geoapify\_key

# Set the geographical coordinates for Sydney, Australia

latitude = -33.8698439

longitude = 151.2082848

# Set the parameters for the type of place

categories = "catering.restaurant"

conditions = "vegetarian"

radius = 8000

# Set the parameters for the type of search

filters = f"circle:{longitude},{latitude},{radius}"

bias = f"proximity:{longitude},{latitude}"

limit = 20

# set up a parameters dictionary

params = {

    "categories":categories,

    "conditions":conditions,

    "limit":limit,

    "filter":filters,

    "bias":bias,

    "apiKey":geoapify\_key

}

# Set base URL

base\_url = "https://api.geoapify.com/v2/places"

# run a request using our params dictionary

response = requests.get(base\_url, params=params)

# print the response url, avoid doing for public github repos in order to avoid exposing key

print(response.url)

<https://api.geoapify.com/v2/places?categories=catering.restaurant&conditions=vegetarian&limit=20&filter=circle%3A151.2082848%2C-33.8698439%2C8000&bias=proximity%3A151.2082848%2C-33.8698439&apiKey=YOUR+KEY+HERE>

# convert response to json

places\_data = response.json()

# Print the json (pretty printed)

print(json.dumps(places\_data, indent=4, sort\_keys=True))

A computer screen shot of white text

Description automatically generated

**truncated**

# Print the name and address of the first restaurant that appears

print(places\_data["features"][0]["properties"]["name"])

print(places\_data["features"][0]["properties"]["address\_line2"])

Macchiato

338 Pitt Street, Sydney NSW 2000, Australia

**Day 3, Activity 3 – Student Geoapify Drills**

**# Geoapify Drills**

Create code to answer each of the following questions.

> **\*\*Hint:\*\*** You will need multiple target URLs and multiple API requests.

# Dependencies

import requests

import json

# Import the API key

from config import geoapify\_key

**## 1. What are the geocoordinates (latitude and longitude) of Brisbane, Australia?**

# Set the search parameters

target\_city = "Brisbane, Australia"

params = {

    "text": target\_city,

    "apiKey": geoapify\_key

}

# Build URL using the geocode endpoint

base\_url = "https://api.geoapify.com/v1/geocode/search"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

A screenshot of a computer program

Description automatically generated

# Extract lat/lon

lat = response["features"][0]["properties"]["lat"]

lon = response["features"][0]["properties"]["lon"]

# Print results

print(f"{target\_city}: {lat}, {lon}")

Brisbane, Australia: -27.4689682, 153.0234991

**## 2. What are the geocoordinates (latitude and longitude) of The Sydney Opera House?**

# update params dict

target\_city = "Sydney Opera House"

params = {

    "text": target\_city,

    "apiKey": geoapify\_key

}

# Build URL using the geocode endpoint

base\_url = "https://api.geoapify.com/v1/geocode/search"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

**A screen shot of a computer

Description automatically generated**

# Extract lat/lon

lat = response["features"][0]["properties"]["lat"]

lon = response["features"][0]["properties"]["lon"]

# Print results

print(f"{target\_city}: {lat}, {lon}")

Sydney Opera House: -33.85719805, 151.21512338473752

**## 3. Find the name and address of a pharmacy in Brisbane, Australia.**

> **\*\*Hint:\*\*** Review <https://apidocs.geoapify.com/docs/places/#categories>.

# Set the geographical coordinates for Brisbane, Australia

latitude = -27.4705

longitude = 153.0260

# Set the parameters for the type of place

categories = "healthcare.pharmacy"

# Set the parameters for the type of search

bias = f"proximity:{longitude},{latitude}"

limit = 20

# set up a parameters dictionary

params = {

    "categories":categories,

    "limit":limit,

    "bias":bias,

    "apiKey":geoapify\_key

}

# Set base URL

base\_url = "https://api.geoapify.com/v2/places"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

A screenshot of a computer screen

Description automatically generated

# Print the results

print(response["features"][0]["properties"]["name"])

print(response["features"][0]["properties"]["address\_line2"])

Priceline Pharmacy

120 Queen Street, Brisbane City QLD 4001, Australia

**## 4. Find a pet shop within a 5km radius from the Sydney Opera House.**

# Set the geographical coordinates for the Sydney Opera House

latitude = -33.85729805

longitude = 151.21512338473752

# Set the parameters for the type of place

categories = "commercial.pet"

radius = 5000

# Set the parameters for the type of search

filters = f"circle:{longitude},{latitude},{radius}"

bias = f"proximity:{longitude},{latitude}"

limit = 20

# set up a parameters dictionary

params = {

    "categories":categories,

    "limit":limit,

    "filter":filters,

    "bias":bias,

    "apiKey":geoapify\_key

}

# Set base URL

base\_url = "https://api.geoapify.com/v2/places"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

A screenshot of a computer screen

Description automatically generated

# Print the results

print(response["features"][0]["properties"]["name"])

print(response["features"][0]["properties"]["address\_line2"])

Petbarn

Kiaora Road, Double Bay NSW 2028, Australia

**## 5. Find the nearest dentist to your house.**

> **\*\*Hint:\*\*** Use Geoapify Geocode to find your latitude and Geoapify Places to find the dentist.

# Geoapify Geocode to find latitude and longitude

my\_address = "1309 Hay St, West Perth WA 6005, Australia"

params = {

    "text": my\_address,

    "apiKey": geoapify\_key

}

# Build URL using the geocode endpoint

base\_url = "https://api.geoapify.com/v1/geocode/search"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

A screenshot of a computer program

Description automatically generated

# Extract lat/lon

lat = response["features"][0]["properties"]["lat"]

lon = response["features"][0]["properties"]["lon"]

# Print results

print(f"{my\_address}: {lat}, {lon}")

1309 Hay St, West Perth WA 6005, Australia: -31.9477365, 115.8387213036199

# Set the geographical coordinates for my home

latitude = -31.9477365

longitude = 115.8387213036199

# Set the parameters for the type of place

categories = "healthcare.dentist"

# Set the parameters for the type of search

bias = f"proximity:{longitude},{latitude}"

limit = 20

# set up a parameters dictionary

params = {

    "categories":categories,

    "limit":limit,

    "bias":bias,

    "apiKey":geoapify\_key

}

# Set base URL

base\_url = "https://api.geoapify.com/v2/places"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

A screenshot of a computer program

Description automatically generated

# Print the results

print(response["features"][0]["properties"]["name"])

print(response["features"][0]["properties"]["address\_line2"])

West Perth Dental Centre

39 Colin Street, West Perth WA 6005, Australia

**## 6. Bonus: Find the names and addresses of the five nearest restaurants with internet access in your home city area.**

> **\*\*Hint:\*\*** Read about the `proximity` bias to order the results from the nearest to the farest place. Also, read about the `conditions` to identify places that provide internet access.

# Set the geographical coordinates for my home

latitude = -31.9477365

longitude = 115.8387213036199

# Set the parameters for the type of place

categories = "catering.restaurant"

conditions = "internet\_access"

# Set the parameters for the type of search

bias = f"proximity:{longitude},{latitude}"

limit = 20

# set up a parameters dictionary

params = {

    "categories":categories,

    "conditions":conditions,

    "limit":limit,

    "bias":bias,

    "apiKey":geoapify\_key

}

# Set base URL

base\_url = "https://api.geoapify.com/v2/places"

# Run request

response = requests.get(base\_url, params=params).json()

# Print the json (pretty printed)

print(json.dumps(response, indent=4, sort\_keys=True))

A screenshot of a computer screen

Description automatically generated

# Print the results

counter = 0

for place in response["features"]:

    print(f"Restaurant {counter+1}:")

    print(place["properties"]["name"])

    print(place["properties"]["address\_line2"])

    print("\*"\*20)

    counter = counter + 1

    if counter == 5:

        break

Restaurant 1:

Cespresso CBD West

996 Hay Street, Perth WA 6000, Australia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Restaurant 2:

El Peruvian Food Co.

Outridge Crescent, Subiaco WA 6008, Australia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Restaurant 3:

PappaRich

101 James Street, Northbridge WA 6000, Australia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Restaurant 4:

Le Vietnam

80 Barrack Street, Perth WA 6000, Australia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Restaurant 5:

Burgermeister

170B Hampden Road, Nedlands WA 6009, Australia

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Day 3, Activity 4 – Instructor Nearest Restaurants**

# Dependencies

import pandas as pd

import numpy as np

import requests

import json

from pathlib import Path

# Import the API key

from config import geoapify\_key

# Load the restaurant ethnicities into a DataFrame

csv\_file = Path("../Resources/ethnicities.csv")

types\_df = pd.read\_csv(csv\_file)

# Display sample data

**A screenshot of a phone

Description automatically generated**types\_df.head()

# Set up additional columns to hold information

types\_df["name"] = ""

types\_df["address"] = ""

types\_df["distance"] = ""

# Display sample data

types\_df.head()

**A screenshot of a computer

Description automatically generated**

Find the closest restaurant of each type to the Plaza Mayor in Madrid, Spain

\* Latitude: 40.415392

\* Longitude: -3.7073743182788528

# Set the geographical coordinates

latitude = 40.415392

longitude = -3.7073743182788528

# Set the parameters for the type of search

limit = 20

radius = 1000

filters = f"circle:{longitude},{latitude},{radius}"

bias = f"proximity:{longitude},{latitude}"

# set up a parameters dictionary

params = {

    "filter":filters,

    "limit":limit,

    "bias":bias,

    "apiKey":geoapify\_key

}

# Set base URL

base\_url = "https://api.geoapify.com/v2/places"

# Print a message to follow up the restaurant search

print("Starting restaurant search")

# Iterate through the types\_df DataFrame

for index, row in types\_df.iterrows():

    # Get the ethnicity type from the current DataFrame's row

    ethnicity = types\_df.loc[index, "ethnicity"]

    # Add the current ethnicity type to the parameters

    params["categories"] = f"catering.restaurant.{ethnicity}"

    # Make an API request using the params dictionary

    restaurant = requests.get(base\_url, params=params)

    # Convert the API response to JSON format

    restaurant = restaurant.json()

    # Grab the first restaurant from the results and store the details in the DataFrame

    try:

        types\_df.loc[index, "name"] = restaurant["features"][0]["properties"]["name"]

        types\_df.loc[index, "address"] = restaurant["features"][0]["properties"]["address\_line2"]

        types\_df.loc[index, "distance"] = int(restaurant["features"][0]["properties"]["distance"])

    except (KeyError, IndexError):

        # If no restaurant is found, set the restaurant name as "No restaurant found".

        types\_df.loc[index, "name"] = "No restaurant found"

        # Set the distance column value to np.nan to allow sorting values

        types\_df.loc[index, "distance"] = np.nan

    # Log the search results

    print(f"nearest {types\_df.loc[index, 'ethnicity']} restaurant: {types\_df.loc[index, 'name']}")

**A screenshot of a computer

Description automatically generated**# Display sample data

types\_df

**not working for SJ**

**A black screen with white text

Description automatically generated**

# Sort restuls by distance

types\_df = types\_df.sort\_values(by=["distance"])

# Display the top 10 nearest restaurants

types\_df.head(10)

**A black screen with white text

Description automatically generated**

**Day 3, Activity 4 – Evr Exploring Airports**

**# Exploring Airports in Australia**

# Dependencies

import pandas as pd

import numpy as np

import requests

import json

from pathlib import Path

# Import API key

from config import geoapify\_key

**## DataFrame Preparation**

# Import cities file as DataFrame

csv\_file = Path("../Resources/cities.csv")

cities\_pd = pd.read\_csv(csv\_file)

cities\_pd.head()

**A screenshot of a black screen

Description automatically generated**

# Add columns the airports data we will fetch using the Geoapify API

# Note that we used "" to specify initial entry.

cities\_pd["Lat"] = ""

cities\_pd["Lon"] = ""

cities\_pd["Airport Name"] = ""

cities\_pd["IATA Name"] = ""

cities\_pd["Airport Address"] = ""

cities\_pd["Distance"] = ""

cities\_pd["Website"] = ""

cities\_pd.head()

**A screen shot of a computer

Description automatically generated**

**## Look for Geographical Coordinates for Each City**

# Define the API parameters

params = {

    "apiKey":geoapify\_key,

    "format":"json"

}

# Set the base URL

base\_url = "https://api.geoapify.com/v1/geocode/search"

# Print a message to follow up the airport search

print("Starting airport search")

# Loop through the cities\_pd DataFrame and search coordinates for each city

for index, row in cities\_pd.iterrows():

    # Get the city's name & add ", Australia" to the string so geoapify finds the correct city

    city = row["City"] + ", Australia"

    # Add the current city to the parameters

    params["text"] = f"{city}"

    # Make the API request

    response = requests.get(base\_url, params=params)

    # Convert reponse to JSON

    response = response.json()

    # Extract latitude and longitude

    cities\_pd.loc[index, "Lat"] = response["results"][0]["lat"]

    cities\_pd.loc[index, "Lon"] = response["results"][0]["lon"]

    # Log the search results

    print(f"Coordinates for {city} fetched...")

# Display sample data to confirm that the coordinates appear

cities\_pd.head()

**A screenshot of a computer

Description automatically generated**

**## Retrieve Airports' Information**

# Set parameters to search for airport's details

radius = 50000

params = {

    "categories":"airport",

    "apiKey": geoapify\_key,

    "limit":20

}

# Print a message to follow up the airport search

print("Starting airport details search")

# Iterate through the types\_df DataFrame

for index, row in cities\_pd.iterrows():

    # Get the city's name

    city = row["City"]

    # Get latitude, longitude from the DataFrame

    latitude = row["Lat"]

    longitude = row["Lon"]

    # Add the current city's latitude and longitude to the params dictionary

    params["filter"] = f"circle:{longitude},{latitude},{radius}"

    params["bias"] = f"proximity:{longitude},{latitude}"

    # Set base URL

    base\_url = "https://api.geoapify.com/v2/places"

    # Make an API request using the params dictionary

    response = requests.get(base\_url, params=params)

    # Convert the API response to JSON format

    response = response.json()

    # Grab the first airport from the results and store the details in the DataFrame

    try:

        cities\_pd.loc[index, "Airport Name"] = response["features"][0]["properties"]["name"]

        cities\_pd.loc[index, "IATA Name"] = response["features"][0]["properties"]["datasource"]["raw"]["iata"]

        cities\_pd.loc[index, "Airport Address"] = response["features"][0]["properties"]["address\_line2"]

        cities\_pd.loc[index, "Distance"] = response["features"][0]["properties"]["distance"]

        cities\_pd.loc[index, "Website"] = response["features"][0]["properties"]["datasource"]["raw"]["website"]

    except KeyError as e:

        # If no airport is found, log the error.

        print(f"{e.args[0]} not found for {cities\_pd.loc[index, 'Airport Name']}")

    # Log the search results

    print(f"nearest airport from {city}: {cities\_pd.loc[index, 'Airport Name']}")

# Display sample data

cities\_pd

**A screenshot of a computer

Description automatically generated**

# Save Data to csv

cities\_pd.to\_csv("Airport\_Output.csv", index=False)

**Day 3, Activity 6 – Evr Geoviews\_Maps**

**# GeoViews Maps Demo**

**## Import Libraries**

# Import the required libraries

import hvplot.pandas

import pandas as pd

# Turn off warning messages

import warnings

warnings.filterwarnings("ignore")

**## Define Sample Data**

# Create a dictionary containing coordinates

coordinates = {

    "latitude": [-31.9523, -33.8688, -27.4705],

    "longitude": [115.8613, 151.2093, 153.0260],

    "City": ["Perth", "Sydney", "Brisbane"],

    "Population": [2099530, 4637436, 2495825]

}

# Create a Pandas DataFrame containing coordinates

coordinates\_df = pd.DataFrame(coordinates)

# Display sample data

coordinates\_df

**A screenshot of a black screen

Description automatically generated**

**## Simple Map**

# Configure the map plot\_1

map\_plot\_1 = coordinates\_df.hvplot.points(

    "longitude",

    "latitude",

    geo = True,

    tiles = "OSM"

)

# Display the map plot\_1

map\_plot\_1

**A map of the united states

Description automatically generated**

**## Map with Fixed Size**

# Configure the map plot\_2

map\_plot\_2 = coordinates\_df.hvplot.points(

    "longitude",

    "latitude",

    geo = True,

    tiles = "OSM",

    frame\_width = 700,

    frame\_height = 500

)

# Display the map plot

map\_plot\_2

**A map of the country

Description automatically generated**

**## Map with Custom Point Colours and Size**

# Configure the map plot\_3

map\_plot\_3 = coordinates\_df.hvplot.points(

    "longitude",

    "latitude",

    geo = True,

    tiles = "OSM",

    frame\_width = 700,

    frame\_height = 500,

    size = "Population",

    scale = 0.01,

    color = "City"

)

# Display the map plot

map\_plot\_3

**A map of australia with orange dots

Description automatically generated**

**## Map with Custom Style and Point Colours and Size**

# Configure the map plot\_4

map\_plot\_4 = coordinates\_df.hvplot.points(

    "longitude",

    "latitude",

    geo = True,

    tiles = "EsriNatGeo",

    frame\_width = 700,

    frame\_height = 500,

    size = "Population",

    scale = 0.01,

    color = "City"

)

# Display the map plot

map\_plot\_4

**A map of australia with orange dots

Description automatically generated**

**Day 3, Activity 7 – Student Airport Map**

**# Australian Airports Map**

# Import the required libraries

import hvplot.pandas

import pandas as pd

from pathlib import Path

# Turn off warning messages

import warnings

warnings.filterwarnings("ignore")

**## 1. Load airports' data into a Pandas DataFrame**

# Create airport dataframe

csv\_file = Path("../Resources/Airport\_Output.csv")

airports\_df = pd.read\_csv(csv\_file)

# Display sample data

A screenshot of a computer

Description automatically generatedairports\_df.head()

**## 2. Create a simple map using GeoViews by adding a point per airport and setting a fixed size at your convenience**

# Configure the map

map\_plot\_1 = airports\_df.hvplot.points(

    "Lon",

    "Lat",

    geo = True,

    tiles = "OSM",

**A map of australia with a country

Description automatically generated**    frame\_width = 800,

    frame\_height = 600

)

# Display the map plot

map\_plot\_1

**## 3. Use GeoViews to create a custom map by setting values for colour, size, and a title different than OSM.**

  > **\*\*Hint:\*\*** From the column values, you should decide what columns can be used to set a different colour and size for each city.

# Configure the map

map\_plot\_2 = airports\_df.hvplot.points(

    "Lon",

    "Lat",

    geo = True,

    tiles = "EsriImagery",

    frame\_width = 800,

    frame\_height = 600,

    size = "Distance",

    scale = 0.5,

    color = "City"

)

# Display the map plot

map\_plot\_2

**A map of australia with colored circles

Description automatically generated**

**Day 3, Activity 8 – Instructor Census**

**# U.S. Census Demo**

# Dependencies

import requests

import numpy as np

import pandas as pd

from census import Census

import matplotlib.pyplot as plt

# Import U.S. Census API Key

from config import api\_key

# Create an instance of the Census library

c = Census(

    api\_key,

    year = 2013

)

**## Retrieve data from the U.S. Census using the Census library**

References:

\* Review the following page to review the Python library documentation: <https://github.com/CommerceDataService/census-wrapper>

\* Review the following page to learn more about the data labels: <https://gist.github.com/afhaque/60558290d6efd892351c4b64e5c01e9b>

# Run Census Search to retrieve data on all zip codes (2013 ACS5 Census)

census\_data = c.acs5.get(

    (

        "NAME",

        "B19013\_001E",

        "B01003\_001E",

        "B01002\_001E",

        "B19301\_001E",

        "B17001\_002E"

    ),

    {'for': 'zip code tabulation area:\*'}

)

# Convert to DataFrame

census\_pd = pd.DataFrame(census\_data)

# Column renaming

census\_pd = census\_pd.rename(

    columns = {

        "B01003\_001E": "Population",

        "B01002\_001E": "Median Age",

        "B19013\_001E": "Household Income",

        "B19301\_001E": "Per Capita Income",

        "B17001\_002E": "Poverty Count",

        "NAME": "Name",

        "zip code tabulation area": "Zipcode"

    }

)

# Add a Poverty Rate column (Poverty Count / Population)

census\_pd["Poverty Rate"] = 100 \* census\_pd["Poverty Count"].astype(int) / census\_pd["Population"].astype(int)

# Configure the final DataFrame

census\_pd = census\_pd[

    [

        "Zipcode",

        "Population",

        "Median Age",

        "Household Income",

        "Per Capita Income",

        "Poverty Count",

        "Poverty Rate"

    ]

]

# Display DataFrame length and sample data

print(f"Number of rows in the DataFrame: {len(census\_pd)}")

census\_pd.head()

**A screenshot of a computer screen

Description automatically generated**

# Save the DataFrame as a CSV

# Note: To avoid any issues later, use encoding="utf-8"

census\_pd.to\_csv("census\_data.csv", encoding="utf-8", index=False)

**Day 3, Activity 9 – Banking Deserts**

**# Banking Deserts**

# Dependencies

from census import Census

from config import census\_key

import hvplot.pandas

import pandas as pd

import requests

import time

from scipy.stats import linregress

from matplotlib import pyplot as plt

from pathlib import Path

# Turn off warning messages

import warnings

warnings.filterwarnings("ignore")

# Create an instance of the Census library

c = Census(census\_key, year=2019)

**## 1. Retrieve data from the U.S. Census using the Census Python library and the preconfigured labels**

References:

\* Review the following page to review the Python library documentation: <https://github.com/CommerceDataService/census-wrapper>

\* Review the following page to learn more about the data labels: <https://gist.github.com/afhaque/60558290d6efd892351c4b64e5c01e9b>

# Run Census Search to retrieve data on all zip codes (2013 ACS5 Census)

census\_data = c.acs5.get(

    (

        "B01003\_001E",

        "B17001\_002E"

    ),

    {'for': 'zip code tabulation area:\*'}

)

# Convert to DataFrame

census\_pd = pd.DataFrame(census\_data)

# Column renaming

census\_pd = census\_pd.rename(

    columns = {

        "B01003\_001E": "Population",

        "B17001\_002E": "Poverty Count",

        "zip code tabulation area": "Zipcode"

    }

)

# Add a Poverty Rate column (Poverty Count / Population)

census\_pd["Poverty Rate"] = 100 \* census\_pd["Poverty Count"].astype(int) / census\_pd["Population"].astype(int)

# Configure the final DataFrame

census\_pd = census\_pd[

    [

        "Zipcode",

        "Population",

        "Poverty Rate"

    ]

]

# Display DataFrame length and sample data

print(f"Number of rows in the DataFrame: {len(census\_pd)}")

census\_pd.head()

**A screenshot of a black screen

Description automatically generated**

**## 2. Load the `zip\_bank\_data.csv` file into a DataFrame. Next, use Pandas to merge this data set with the Census data that you retrieved along the zip code.**

# Import the zip bank data. Use dtype="object" to ensure all data is read in accurately.

csv\_file = Path("../Resources/zip\_bank\_data.csv")

zip\_code\_pd = pd.read\_csv(

    csv\_file,

    dtype = "object",

    encoding = "utf-8"

)

# Display sample data

zip\_code\_pd.head()

A screenshot of a computer screen

Description automatically generated

# Merge the two data sets along zip code

census\_data\_complete = pd.merge(

    zip\_code\_pd,

    census\_pd,

    how = "left",

    on = ["Zipcode", "Zipcode"]

)

# Remove rows with missing data

census\_data\_complete = census\_data\_complete.dropna()

# Transform latitude and longitude to float

census\_data\_complete["Lat"] = census\_data\_complete["Lat"].astype(float)

census\_data\_complete["Lng"] = census\_data\_complete["Lng"].astype(float)

# Display sample data

**A screenshot of a black screen

Description automatically generated**census\_data\_complete.head()

**## 3. Use GeoViews to create a poverty rate map.**

\* Use the "Poverty Rate" column to set the point's size. Recall to use the `scale` parameter to modify the size appearance.

\* Use the "Zipcode" column to set the point's colour.

\* Read the HoloViews documentation and learng how you can use the `hover\_cols` parameter to add additional information to the tooltip of a point. Add the "Address" and the "Bank Count" columns.

# Configure the map

poverty\_rate\_map = census\_data\_complete.hvplot.points(

    "Lng",

    "Lat",

    geo = True,

    tiles = "OSM",

    frame\_width = 800,

    frame\_height = 600,

    size = "Poverty Rate",

    scale = 2,

    color = "Zipcode",

    hover\_cols = ["Address", "Bank Count"]

)

# Display the map plot

poverty\_rate\_map

**A map of the united states

Description automatically generated**

**## 4. Compute and print the summary statistics for "Poverty Rate", "Bank Count", and "Population"**

# Mean, median, mode for Poverty Rate

poverty\_mean = round(census\_data\_complete["Poverty Rate"].astype("float").mean(), 2)

poverty\_median = round(census\_data\_complete["Poverty Rate"].astype("float").median(), 2)

poverty\_mode = round(census\_data\_complete["Poverty Rate"].astype("float").mode(), 2)

print(f"Poverty Rate Mean: {poverty\_mean}")

print(f"Poverty Rate Median {poverty\_median}")

print(f"Poverty Rate mode {poverty\_mode}")

Poverty Rate Mean: 13.66

Poverty Rate Median 11.05

Poverty Rate mode 0 0.0

Name: Poverty Rate, dtype: float64

# Mean, median, mode for Bank Count

bank\_mean = round(census\_data\_complete["Bank Count"].astype("float").mean(), 2)

bank\_median = round(census\_data\_complete["Bank Count"].astype("float").median(), 2)

bank\_mode = round(census\_data\_complete["Bank Count"].astype("float").mode(), 2)

print(f"Bank Count Mean: {bank\_mean}")

print(f"Bank Count Median {bank\_median}")

print(f"Bank Count mode {bank\_mode}")

Bank Count Mean: 41.01

Bank Count Median 36.0

Bank Count mode 0 4.0

Name: Bank Count, dtype: float64

# Mean, median, mode for Population

population\_mean = round(census\_data\_complete["Population"].astype("float").mean(), 2)

population\_median = round(census\_data\_complete["Population"].astype("float").median(), 2)

population\_mode = round(census\_data\_complete["Population"].astype("float").mode(), 2)

print(f"Population Mean: {population\_mean}")

print(f"Population Median {population\_median}")

print(f"Population mode {population\_mode}")

Population Mean: 9886.26

Population Median 2655.0

Population mode 0 167.0

Name: Population, dtype: float64

**## 5. Create a scatter plot with linear regression for \*\*bank count\*\* vs. \*\*poverty rate\*\*.**

\* Be sure `NaN` values are dropped from the DataFrame.

\* Plot the data points.

\* Plot the linear regression line.

\* Print the R<sup>2</sup> value.

## Convert to floats and store Poverty Rate and Bank Count as x and y values

x\_values = census\_data\_complete["Poverty Rate"].astype("float")

y\_values = census\_data\_complete["Bank Count"].astype("float")

# Run linear regression

(slope, intercept, rvalue, pvalue, stderr) = linregress(x\_values, y\_values)

regress\_values = x\_values \* slope + intercept

line\_eq = "y = " + str(round(slope,2)) + "x + " + str(round(intercept,2))

# Plot scatter plot

plt.scatter(x\_values,y\_values)

# Plot regression line

plt.plot(x\_values,regress\_values,"r-")

plt.annotate(line\_eq,(6,10),fontsize=15,color="red")

# Label plot

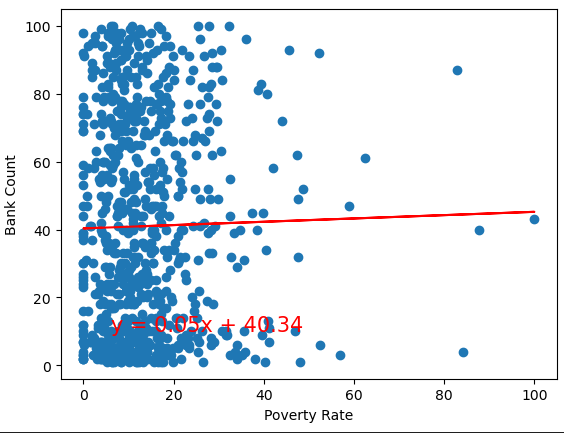
plt.xlabel('Poverty Rate')

plt.ylabel('Bank Count')

# Print r square value

print(f"R squared: {rvalue\*\*2}")

# Show plot

plt.show()

R squared: 0.0003075681459118965

**## 6. Analysis.**

Write a sentence describing your findings. Were they what you expected? What other factors could be at play?

\* There is a very weak correlation between poverty rates and bank counts. Keep in mind that linear regression will not consider other factors such as population or size of the city.