Python Assignment Solutions

Question 1: Write a program that takes a string as input, and counts the frequency of each word in the string, there might be repeated characters in the string. Your task is to find the highest frequency and returns the length of the highest-frequency word.

Note - You have to write at least 2 additional test cases in which your program will run successfully and provide an explanation for the same. Example input - string = “write write write all the number from from from 1 to 100” Example output - 5 Explanation - From the given string we can note that the most frequent words are “write” and “from” and the maximum value of both the values is “write” and its corresponding length is 5

In [3]:

```
def find_highest_frequency_word_length(string):
    words = string.split()

    word_freq = {}

    for word in words:
        word_freq[word] = word_freq.get(word, 0) + 1

    max_freq = 0
    max_length = 0
    for word, freq in word_freq.items():
        if freq > max_freq or (freq == max_freq and len(word) > max_length):
            max_freq = freq
            max_length = len(word)

    return max_length

#Example test case

string = "write write write all the number from from from 1 to 100"
result = find_highest_frequency_word_length(string)
print(result)

```

```
5
```

In [4]:

```
#Example test case
string = "hello hello hello hello hello"
result = find_highest_frequency_word_length(string)
print(result)
```

```
5
```

Explaination: In the given example, the word "write" appears three times, making it the most frequent word. Its length is 5 characters, so the program returns 5.

Explaination: In this case, the word "hello" appears five times, which is the highest frequency. Its length is 5 characters, so the program returns 5.

Question 2: Consider a string to be valid if all characters of the string appear the same number of times. It is also valid ifhe can remove just one character at the index in the string, and the remaining characters will occur the same number of times. Given a string, determine if it is valid. If so, return YES , otherwise return NO .

Note - You have to write at least 2 additional test cases in which your program will run successfully and provide an explanation for the same.

Example input 1 - s = “abc”. This is a valid string because frequencies are { “a”: 1, “b”: 1, “c”: 1 }

Example output 1- YES

Example input 2 - s “abcc”. This string is not valid as we can remove only 1 occurrence of “c”. That leaves

character frequencies of { “a”: 1, “b”: 1 , “c”: 2 }

Example output 2 - NO

In [5]:

```
from collections import Counter

def is_valid_string(s):
    # Count the frequency of each character
    char_frequency = Counter(s)

    # Find the frequencies of all characters
    frequencies = list(char_frequency.values())

    # Check if all frequencies are the same
    if len(set(frequencies)) == 1:
        return "YES"

    # Check if removing one character can make all remaining characters have the same frequency
    for char in char_frequency:
        # Try removing the current character
        updated_frequencies = frequencies.copy()
        updated_frequencies.remove(char_frequency[char])
        if len(set(updated_frequencies)) == 1:
            return "NO"

    return "NO"
```

In [6]:

```
s = "abc"
result = is_valid_string(s)
print(result)
```

```
YES
```

In [7]:

```
s = "abcc"
result = is_valid_string(s)
print(result)
```

```
NO
```

Explanation: In this case, the frequencies of all characters are { "a": 1, "b": 1, "c": 1 }. Since all characters have the same frequency, the string is considered valid. Therefore, the program correctly returns "YES" as the output

In this case, if we remove only one occurrence of "c", the frequencies of the remaining characters would be { "a": 1, "b": 1, "c": 2 }. Since the frequencies are not all the same, the string is not valid. Therefore, the program correctly returns "NO" as the output.

The program works by counting the frequency of each character using the Counter class from the collections module. It then checks if all frequencies are the same or if removing one character can make all remaining characters have the same frequency. Based on the conditions, it returns "YES" if the string is valid and "NO" if it is not.

Question 3: Write a program, which would download the data from the provided link, and then read the data and convert that into properly structured data and return it in Excel format.

`Note` - Write comments wherever necessary explaining the code written.

Link - https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json (https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json)

Data Attributes - id: Identification Number - int num: Number of the

● Pokémon in the official Pokédex - int name: Pokémon name -

● string img: URL to an image of this Pokémon - string type:

● Pokémon type -string height: Pokémon height - float

● weight: Pokémon weight - float candy: type of candy used to evolve Pokémon or given

● when transferred - string candy_count: the amount of candies required to evolve

- int

● egg: Number of kilometers to travel to hatch the egg - float spawn_chance:

● Percentage of spawn chance (NEW) - float avg_spawns: Number of this pokemon on 10.000 spawns (NEW) - int

● spawn_time: Spawns most active at the time on this field. Spawn times are the same for all time zones and are expressed in local time. (NEW) - “minutes: seconds” multipliers: Multiplier of Combat Power (CP) for calculating the CP after evolution See below - list of int weakness: Types of

● Pokémon this Pokémon is weak to - list of strings next_evolution: Number and Name of successive evolutions of Pokémon - list of dict prev_evolution: Number and Name of previous evolutions of Pokémon - - list of dict

In [19]:

```
import requests
import pandas as pd
```

In [20]:

```
def json_to_csv(link):
    response = requests.get(link)
    data = response.json()["pokemon"]

    # Convert JSON data to DataFrame
    df = pd.DataFrame(data)

    df.to_csv("Output.csv", index=False)
```

In [21]:

```
link = "https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json"
json_to_csv(link)
```

In [22]:

```
df1 = pd.read_csv("Output.csv")
df1.to_excel("Output.xlsx")
```

In [23]:

```
df1
```

Out[23]:

	id	num	name	img	type	height	weight	candy	candy_count	egg	spawn_chance	a
0	1	1	Bulbasaur	http://www.serebii.net/pokemongo/pokemon/001.png	['Grass', 'Poison']	0.71 m	6.9 kg	Bulbasaur Candy	25.0	2 km	0.6900	
1	2	2	Ivysaur	http://www.serebii.net/pokemongo/pokemon/002.png	['Grass', 'Poison']	0.99 m	13.0 kg	Bulbasaur Candy	100.0	Not in Eggs	0.0420	
2	3	3	Venusaur	http://www.serebii.net/pokemongo/pokemon/003.png	['Grass', 'Poison']	2.01 m	100.0 kg	Bulbasaur Candy	NaN	Not in Eggs	0.0170	
3	4	4	Charmander	http://www.serebii.net/pokemongo/pokemon/004.png	['Fire']	0.61 m	8.5 kg	Charmander Candy	25.0	2 km	0.2530	
4	5	5	Charmeleon	http://www.serebii.net/pokemongo/pokemon/005.png	['Fire']	1.09 m	19.0 kg	Charmander Candy	100.0	Not in Eggs	0.0120	
...	...	...	...	...	...	...	...	...	...	...	...	
146	147	147	Dratini	http://www.serebii.net/pokemongo/pokemon/147.png	['Dragon']	1.80 m	3.3 kg	Dratini Candy	25.0	10 km	0.3000	
147	148	148	Dragonair	http://www.serebii.net/pokemongo/pokemon/148.png	['Dragon']	3.99 m	16.5 kg	Dratini Candy	100.0	Not in Eggs	0.0200	
148	149	149	Dragonite	http://www.serebii.net/pokemongo/pokemon/149.png	['Dragon', 'Flying']	2.21 m	210.0 kg	Dratini Candy	NaN	Not in Eggs	0.0011	
149	150	150	Mewtwo	http://www.serebii.net/pokemongo/pokemon/150.png	['Psychic']	2.01 m	122.0 kg	None	NaN	Not in Eggs	0.0000	
150	151	151	Mew	http://www.serebii.net/pokemongo/pokemon/151.png	['Psychic']	0.41 m	4.0 kg	None	NaN	Not in Eggs	0.0000	

151 rows × 17 columns

In [24]:

```
1 df1.dtypes
```

Out[24]:

```
id                 int64
num                int64
name              object
img               object
type              object
height            object
weight            object
candy             object
candy_count      float64
egg               object
spawn_chance     float64
avg_spawns       float64
spawn_time        object
multipliers       object
weaknesses        object
next_evolution    object
prev_evolution    object
dtype: object
```

Question 4 : Write a program to download the data from the link given below and then read the data and convert the into the proper structure and return it as a CSV file.

Link - https://data.nasa.gov/resource/y77d-th95.json (https://data.nasa.gov/resource/y77d-th95.json)

Note - Write code comments wherever needed for code understanding. Sample Data -

Excepted Output Data Attributes

- Name of Earth Meteorite - string id - ID of Earth
- Meteorite - int nametype - string recclass - string
- mass - Mass of Earth Meteorite - float year - Year at which Earth
- Meteorite was hit - datetime format reclat - float recclong - float
- point coordinates - list of int

In [2]:

```
1 import json
```

In [3]:

```
import requests
import csv

def download_data(url):
    response = requests.get(url)
    data = response.json()
    return data

def convert_to_csv(data):
    attributes = ['name', 'id', 'nametype', 'recclass', 'mass', 'year', 'reclat', 'reclong', 'geolocation.coordinates']

    with open('meteorite1_data.csv', 'w', newline='', encoding='utf-8') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(attributes)

        for meteorite in data:
            row = [
                meteorite.get('name', ''),
                meteorite.get('id', ''),
                meteorite.get('nametype', ''),
                meteorite.get('recclass', ''),
                meteorite.get('mass', ''),
                meteorite.get('year', ''),
                meteorite.get('reclat', ''),
                meteorite.get('reclong', ''),
                meteorite.get('geolocation', {}).get('coordinates', [])
            ]
            writer.writerow(row)

    print("CSV file generated successfully!")

# Provide the URL to download the data from
url = "https://data.nasa.gov/resource/y77d-th95.json"

# Download the data
data = download_data(url)

# Convert and save the data as a CSV file
convert_to_csv(data)
```

```
CSV file generated successfully!
```

Question 5 : Write a program to download the data from the given API link and then extract the following data with proper formatting

Link - http://api.tvmaze.com/singlesearch/shows?q=westworld&embed=episodes (http://api.tvmaze.com/singlesearch/shows?q=westworld&embed=episodes) Note - Write proper code comments wherever needed for the code understanding Excepted Output Data Attributes -

- id - int url - string
- name - string season
- - int number - int
- type - string airdate -
- date format airtime -
- 12-hour time format
- runtime - float
- average rating - float
- summary - string
- without html tags
- medium image link - string
- Original image link - string

In [41]:

```
import requests
import json
from bs4 import BeautifulSoup
```

In [42]:

```
# API URL
url = "http://api.tvmaze.com/singlesearch/shows?q=westworld&embed=episodes"

# Send a GET request to the API URL and retrieve the data
response = requests.get(url)
data = json.loads(response.text)

# Iterate over each episode and extract the required data
for episode in data['_embedded']['episodes']:
    # Extract attributes
    episode_id = episode['id']
    episode_url = episode['url']
    episode_name = episode['name']
    season = episode['season']
    episode_number = episode['number']
    episode_type = episode['type']
    airdate = episode['airdate']
    airtime = episode['airtime']
    runtime = episode['runtime']
    average_rating = episode['rating']['average']

    # Clean up the summary by removing HTML tags using BeautifulSoup
    summary = BeautifulSoup(episode['summary'], 'html.parser').get_text(strip=True)

    # Extract image links
    image_medium = episode['image']['medium']
    image_original = episode['image']['original']

    # Print the extracted data with proper formatting
    print("Episode ID:", episode_id)
    print("URL:", episode_url)
    print("Name:", episode_name)
    print("Season:", season)
    print("Number:", episode_number)
    print("Type:", episode_type)
    print("Airdate:", airdate)
    print("Airtime:", airtime)
    print("Runtime:", runtime)
    print("Average Rating:", average_rating)
    print("Summary:", summary)
    print("Medium Image Link:", image_medium)
    print("Original Image Link:", image_original)
    print()
```

```
Number: 1
Type: regular
Airdate: 2016-10-02
Airtime: 21:00
Runtime: 68
Average Rating: 8
Summary: A woman named Dolores is a free spirit in the Old West... and unaware that she's actually an android, programm
ed to entertain rich guests seeking to act out their fantasies in an idealized vision of the 1880s. However, the people
in charge soon realize that their androids are acting in ways that they didn't anticipate.
Medium Image Link: https://static.tvmaze.com/uploads/images/medium_landscape/78/195475.jpg (https://static.tvmaze.com/u
ploads/images/medium_landscape/78/195475.jpg)
Original Image Link: https://static.tvmaze.com/uploads/images/original_untouched/78/195475.jpg (https://static.tvmaze.c
om/uploads/images/original_untouched/78/195475.jpg)

Episode ID: 911201
URL: https://www.tvmaze.com/episodes/911201/westworld-1x02-chestnut (https://www.tvmaze.com/episodes/911201/westworld-1
x02-chestnut)
Name: Chestnut
Season: 1
Number: 2
```

Question 6 :Using the data from Question 3, write code to analyze the data and answer the following questions Note

1.Draw plots to demonstrate the analysis for the following questions for better visualizations.

2. Write code comments wherever required for code understanding

Insights to be drawn - ● Get all Pokemons whose spawn rate is less than 5%

● Get all Pokemons that have less than 4 weaknesses

● Get all Pokemons that have no multipliers at all

● Get all Pokemons that do not have more than 2 evolutions

● Get all Pokemons whose spawn time is less than 300 seconds.

Note - spawn time format is "05:32", so assume “minute: second” format and perform the analysis.

● Get all Pokemon who have more than two types of capabilities

In [1]:

```
import requests
import pandas as pd
import numpy as np
```

In [2]:

```
def json_to_csv(link):
    response = requests.get(link)
    data = response.json()["pokemon"]

    # Convert JSON data to DataFrame
    df = pd.DataFrame(data)

    df.to_csv("Output.csv", index=False)
```

In [3]:

```
link = "https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json"
json_to_csv(link)
```

In [4]:

```
df1 = pd.read_csv("Output.csv")
```

In [5]:

```
df1
```

Out[5]:

	id	num	name	img	type	height	weight	candy	candy_count	egg	spawn_chance	a
0	1	1	Bulbasaur	http://www.serebii.net/pokemongo/pokemon/001.png	['Grass', 'Poison']	0.71 m	6.9 kg	Bulbasaur Candy	25.0	2 km	0.6900	
1	2	2	Ivysaur	http://www.serebii.net/pokemongo/pokemon/002.png	['Grass', 'Poison']	0.99 m	13.0 kg	Bulbasaur Candy	100.0	Not in Eggs	0.0420	
2	3	3	Venusaur	http://www.serebii.net/pokemongo/pokemon/003.png	['Grass', 'Poison']	2.01 m	100.0 kg	Bulbasaur Candy	NaN	Not in Eggs	0.0170	
3	4	4	Charmander	http://www.serebii.net/pokemongo/pokemon/004.png	['Fire']	0.61 m	8.5 kg	Charmander Candy	25.0	2 km	0.2530	
4	5	5	Charmeleon	http://www.serebii.net/pokemongo/pokemon/005.png	['Fire']	1.09 m	19.0 kg	Charmander Candy	100.0	Not in Eggs	0.0120	
...	...	...	...	...	...	...	...	...	...	...	...	...
146	147	147	Dratini	http://www.serebii.net/pokemongo/pokemon/147.png	['Dragon']	1.80 m	3.3 kg	Dratini Candy	25.0	10 km	0.3000	
147	148	148	Dragonair	http://www.serebii.net/pokemongo/pokemon/148.png	['Dragon']	3.99 m	16.5 kg	Dratini Candy	100.0	Not in Eggs	0.0200	
148	149	149	Dragonite	http://www.serebii.net/pokemongo/pokemon/149.png	['Dragon', 'Flying']	2.21 m	210.0 kg	Dratini Candy	NaN	Not in Eggs	0.0011	
149	150	150	Mewtwo	http://www.serebii.net/pokemongo/pokemon/150.png	['Psychic']	2.01 m	122.0 kg	None	NaN	Not in Eggs	0.0000	
150	151	151	Mew	http://www.serebii.net/pokemongo/pokemon/151.png	['Psychic']	0.41 m	4.0 kg	None	NaN	Not in Eggs	0.0000	

151 rows × 17 columns

In [6]:

```
# Get all Pokemons whose spawn rate is less than 5%
five_perc = df1["spawn_chance"].quantile(0.05)
df1[df1["spawn_chance"] < five_perc]["name"]
```

Out[6]:

```
131        Ditto
143     Articuno
144       Zapdos
145      Moltres
148    Dragonite
149       Mewtwo
150          Mew
Name: name, dtype: object
```

In [7]:

```
import ast
```

In [8]:

```
# Get all Pokemons that have less than 4 weaknesses

df1["weaknesses"] = df1["weaknesses"].apply(lambda x : ast.literal_eval(x))
df1[df1["weaknesses"].apply(lambda x : len(x)) < 4]["name"]
```

Out[8]:

```
3      Charmander
4      Charmeleon
5       Charizard
6        Squirtle
7       Wartortle
          ...
145       Moltres
146       Dratini
147     Dragonair
149        Mewtwo
150           Mew
Name: name, Length: 102, dtype: object
```

In [9]:

```
# Get all Pokemons that have no multipliers at all
df1[df1["multipliers"].isna()]["name"]
```

Out[9]:

```
2        Venusaur
5       Charizard
8       Blastoise
11     Butterfree
14       Beedrill
          ...
144       Zapdos
145       Moltres
148     Dragonite
149        Mewtwo
150           Mew
Name: name, Length: 81, dtype: object
```

In [10]:

```
df1
```

Out[10]:

	id	num	name	img	type	height	weight	candy	candy_count	egg	spawn_chance	av
0	1	1	Bulbasaur	http://www.serebii.net/pokemongo/pokemon/001.png	['Grass', 'Poison']	0.71 m	6.9 kg	Bulbasaur Candy	25.0	2 km	0.6900	
1	2	2	Ivysaur	http://www.serebii.net/pokemongo/pokemon/002.png	['Grass', 'Poison']	0.99 m	13.0 kg	Bulbasaur Candy	100.0	Not in Eggs	0.0420	
2	3	3	Venusaur	http://www.serebii.net/pokemongo/pokemon/003.png	['Grass', 'Poison']	2.01 m	100.0 kg	Bulbasaur Candy	NaN	Not in Eggs	0.0170	
3	4	4	Charmander	http://www.serebii.net/pokemongo/pokemon/004.png	['Fire']	0.61 m	8.5 kg	Charmander Candy	25.0	2 km	0.2530	
4	5	5	Charmeleon	http://www.serebii.net/pokemongo/pokemon/005.png	['Fire']	1.09 m	19.0 kg	Charmander Candy	100.0	Not in Eggs	0.0120	
...	...	...	...	...	...	...	...	...	...	...	...	
146	147	147	Dratini	http://www.serebii.net/pokemongo/pokemon/147.png	['Dragon']	1.80 m	3.3 kg	Dratini Candy	25.0	10 km	0.3000	
147	148	148	Dragonair	http://www.serebii.net/pokemongo/pokemon/148.png	['Dragon']	3.99 m	16.5 kg	Dratini Candy	100.0	Not in Eggs	0.0200	
148	149	149	Dragonite	http://www.serebii.net/pokemongo/pokemon/149.png	['Dragon', 'Flying']	2.21 m	210.0 kg	Dratini Candy	NaN	Not in Eggs	0.0011	
149	150	150	Mewtwo	http://www.serebii.net/pokemongo/pokemon/150.png	['Psychic']	2.01 m	122.0 kg	None	NaN	Not in Eggs	0.0000	
150	151	151	Mew	http://www.serebii.net/pokemongo/pokemon/151.png	['Psychic']	0.41 m	4.0 kg	None	NaN	Not in Eggs	0.0000	

151 rows × 17 columns

In [11]:

```
# Get all Pokemons that do not have more than 2 evolutions

df = df1.copy()
```

In [12]:

```
df["next_evolution"].dropna(inplace=True)
```

In [13]:

```
df["Evloutions_2"] = df["next_evolution"].dropna().apply(lambda x : ast.literal_eval(x)).apply(lambda x : len(x)) < 2
```

In [14]:

```
df.loc[df["Evloutions_2"] == True]["name"]
```

Out[14]:

```
1         Ivysaur
4      Charmeleon
7       Wartortle
10        Metapod
13         Kakuna
16      Pidgeotto
18        Rattata
20        Spearow
22          Ekans
24        Pikachu
26      Sandshrew
29       Nidorina
32       Nidorino
34       Clefairy
36         Vulpix
38     Jigglypuff
40          Zubat
43          Gloom
45          Paras
47        Venonat
49        Diglett
51         Meowth
53        Psyduck
55         Mankey
57      Growlithe
60      Poliwhirl
63        Kadabra
66        Machoke
69     Weepinbell
71      Tentacool
74       Graveler
76         Ponyta
78       Slowpoke
80      Magnemite
83          Doduo
85           Seel
87         Grimer
89       Shellder
92        Haunter
95        Drowzee
97         Krabby
99        Voltorb
101    Exeggcute
103       Cubone
108       Koffing
110       Rhyhorn
115        Horsea
117       Goldeen
119        Staryu
128      Magikarp
137       Omanyte
139        Kabuto
147     Dragonair
Name: name, dtype: object
```

In [15]:

```
df1
```

Out[15]:

	id	num	name	img	type	height	weight	candy	candy_count	egg	spawn_chance	av
0	1	1	Bulbasaur	http://www.serebii.net/pokemongo/pokemon/001.png	['Grass', 'Poison']	0.71 m	6.9 kg	Bulbasaur Candy	25.0	2 km	0.6900	
1	2	2	Ivysaur	http://www.serebii.net/pokemongo/pokemon/002.png	['Grass', 'Poison']	0.99 m	13.0 kg	Bulbasaur Candy	100.0	Not in Eggs	0.0420	
2	3	3	Venusaur	http://www.serebii.net/pokemongo/pokemon/003.png	['Grass', 'Poison']	2.01 m	100.0 kg	Bulbasaur Candy	NaN	Not in Eggs	0.0170	
3	4	4	Charmander	http://www.serebii.net/pokemongo/pokemon/004.png	['Fire']	0.61 m	8.5 kg	Charmander Candy	25.0	2 km	0.2530	
4	5	5	Charmeleon	http://www.serebii.net/pokemongo/pokemon/005.png	['Fire']	1.09 m	19.0 kg	Charmander Candy	100.0	Not in Eggs	0.0120	
...	...	...	...	...	...	...	...	...	...	...	...	
146	147	147	Dratini	http://www.serebii.net/pokemongo/pokemon/147.png	['Dragon']	1.80 m	3.3 kg	Dratini Candy	25.0	10 km	0.3000	
147	148	148	Dragonair	http://www.serebii.net/pokemongo/pokemon/148.png	['Dragon']	3.99 m	16.5 kg	Dratini Candy	100.0	Not in Eggs	0.0200	
148	149	149	Dragonite	http://www.serebii.net/pokemongo/pokemon/149.png	['Dragon', 'Flying']	2.21 m	210.0 kg	Dratini Candy	NaN	Not in Eggs	0.0011	
149	150	150	Mewtwo	http://www.serebii.net/pokemongo/pokemon/150.png	['Psychic']	2.01 m	122.0 kg	None	NaN	Not in Eggs	0.0000	
150	151	151	Mew	http://www.serebii.net/pokemongo/pokemon/151.png	['Psychic']	0.41 m	4.0 kg	None	NaN	Not in Eggs	0.0000	

151 rows × 17 columns

In [16]:

```
# Get all Pokemons whose spawn time is less than 300 seconds
df = df1.copy()
```

In [17]:

```
df["spawn_time_less_then_300"] = df["spawn_time"].dropna().apply(lambda x: (int(x.split(":")[0]) * 60) +(int(x.split(":")[1])) < 300
```

In [18]:

```
df.loc[df["spawn_time_less_then_300"] == True]["name"]
```

Out[18]:

```
6       Squirtle
8      Blastoise
10       Metapod
12        Weedle
13        Kakuna
          ...
127       Tauros
129     Gyarados
134      Jolteon
136      Porygon
139       Kabuto
Name: name, Length: 75, dtype: object
```

In [19]:

```
# Get all Pokemon who have more than two types of capabilities
df = df1.copy()
```

In [20]:

```
df["type"] = df["type"].apply(lambda x : ast.literal_eval(x))
```

In [21]:

```
df[df["type"].apply(lambda x : len(x) > 2)]["name"]
#There are no pokemon more than two types of capabilities
```

Out[21]:

```
Series([], Name: name, dtype: object)
```

Question 7 :Using the data from Question 4, write code to analyze the data and answer the following questions Note -

1. Draw plots to demonstrate the analysis for the following questions for better visualizations

2. Write code comments wherever required for code understanding

Insights to be drawn -

- Get all the Earth meteorites that fell before the year 2000
- Get all the earth meteorites co-ordinates who fell before the year 1970
- Assuming that the mass of the earth meteorites was in kg, get all those whose mass was more than 10000kg

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
df = pd.read_csv('meteorite1_data.csv')
```

In [3]:

```
df
```

Out[3]:

	name	id	nametype	recclass	mass	year	reclat	reclong	geolocation.coordinates
0	Aachen	1	Valid	L5	21.0	1880-01-01T00:00:00.000	50.77500	6.08333	[6.08333, 50.775]
1	Aarhus	2	Valid	H6	720.0	1951-01-01T00:00:00.000	56.18333	10.23333	[10.23333, 56.18333]
2	Abee	6	Valid	EH4	107000.0	1952-01-01T00:00:00.000	54.21667	-113.00000	[-113, 54.21667]
3	Acapulco	10	Valid	Acapulcoite	1914.0	1976-01-01T00:00:00.000	16.88333	-99.90000	[-99.9, 16.88333]
4	Achiras	370	Valid	L6	780.0	1902-01-01T00:00:00.000	-33.16667	-64.95000	[-64.95, -33.16667]
...	...	...	...	...	...	...	...	...	...
995	Tirupati	24009	Valid	H6	230.0	1934-01-01T00:00:00.000	13.63333	79.41667	[79.41667, 13.63333]
996	Tissint	54823	Valid	Martian (shergottite)	7000.0	2011-01-01T00:00:00.000	29.48195	-7.61123	[-7.61123, 29.48195]
997	Tjabe	24011	Valid	H6	20000.0	1869-01-01T00:00:00.000	-7.08333	111.53333	[111.53333, -7.08333]
998	Tjerebon	24012	Valid	L5	16500.0	1922-01-01T00:00:00.000	-6.66667	106.58333	[106.58333, -6.66667]
999	Tomakovka	24019	Valid	LL6	600.0	1905-01-01T00:00:00.000	47.85000	34.76667	[34.76667, 47.85]

1000 rows × 9 columns

In [4]:

```
df.head()
```

Out[4]:

	name	id	nametype	recclass	mass	year	reclat	reclong	geolocation.coordinates
0	Aachen	1	Valid	L5	21.0	1880-01-01T00:00:00.000	50.77500	6.08333	[6.08333, 50.775]
1	Aarhus	2	Valid	H6	720.0	1951-01-01T00:00:00.000	56.18333	10.23333	[10.23333, 56.18333]
2	Abee	6	Valid	EH4	107000.0	1952-01-01T00:00:00.000	54.21667	-113.00000	[-113, 54.21667]
3	Acapulco	10	Valid	Acapulcoite	1914.0	1976-01-01T00:00:00.000	16.88333	-99.90000	[-99.9, 16.88333]
4	Achiras	370	Valid	L6	780.0	1902-01-01T00:00:00.000	-33.16667	-64.95000	[-64.95, -33.16667]

In [5]:

```
df.shape
```

Out[5]:

```
(1000, 9)
```

In [6]:

```
# Get all the Earth meteorites that fell before the year 2000

df["only_year"] = df["year"].apply(lambda x : str(x).split("T")[0].split("-")[0])
df["before_2000"] = df["only_year"].astype(float) < 2000
df.loc[df["before_2000"] == True]["name"]
```

Out[6]:

```
0         Aachen
1         Aarhus
2           Abee
3       Acapulco
4        Achiras
         ...
994     Timochin
995     Tirupati
997        Tjabe
998     Tjerebon
999    Tomakovka
Name: name, Length: 929, dtype: object
```

In [7]:

```
# Get all the earth meteorites co-ordinates who fell before the year 1970
import ast
df["year_1970"] = df["only_year"].astype(float) < 1970
coordinates = df.loc[df["year_1970"] == True]["geolocation.coordinates"].dropna().apply(lambda x : ast.literal_eval(x))
coordinates
```

Out[7]:

```
0          [6.08333, 50.775]
1       [10.23333, 56.18333]
2           [-113, 54.21667]
4        [-64.95, -33.16667]
5              [71.8, 32.1]
              ...
994             [35.2, 54.5]
995     [79.41667, 13.63333]
997    [111.53333, -7.08333]
998    [106.58333, -6.66667]
999        [34.76667, 47.85]
Name: geolocation.coordinates, Length: 780, dtype: object
```

In [9]:

```
# Assuming that the mass of the earth meteorites was in kg, get all those whose mass was more
# than 10000kg

df["col"] = df["mass"] > 10000
df.loc[df["col"] == True]["name"]
```

Out[9]:

```
2            Abee
7            Agen
11            Aïr
16        Akyumak
27     Alfianello
          ...
991     Tieschitz
992        Tilden
994      Timochin
997         Tjabe
998      Tjerebon
Name: name, Length: 243, dtype: object
```

In [12]:

```
earth_meteorites_before_2000 = df[df['year'] < '2000-01-01']
earth_meteorites_coords_before_1970 = df[(df['year'] < '1970-01-01') & (df['reclat'].notna()) & (df['reclong'].notna())]
earth_meteorites_mass_above_10000kg = df[df['mass'] > 10000]
```

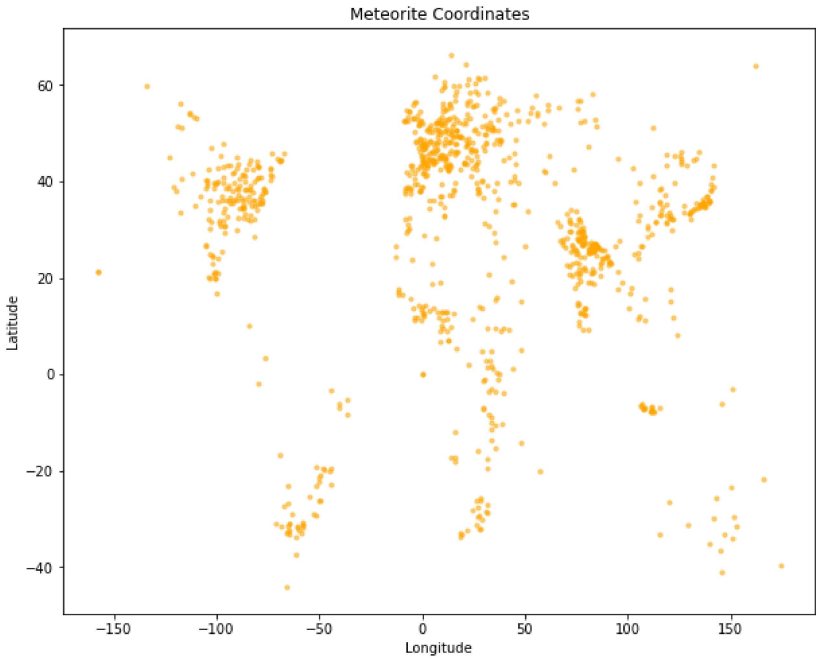
In [13]:

```
print("Insights:")
print("1. Number of Earth meteorites that fell before the year 2000:", len(earth_meteorites_before_2000))
print("2. Number of Earth meteorite coordinates that fell before the year 1970:",
      len(earth_meteorites_coords_before_1970))
print("3. Number of Earth meteorites with mass more than 10000 kg:", len(earth_meteorites_mass_above_10000kg))
```

```
Insights:
1. Number of Earth meteorites that fell before the year 2000: 929
2. Number of Earth meteorite coordinates that fell before the year 1970: 774
3. Number of Earth meteorites with mass more than 10000 kg: 243
```

In [15]:

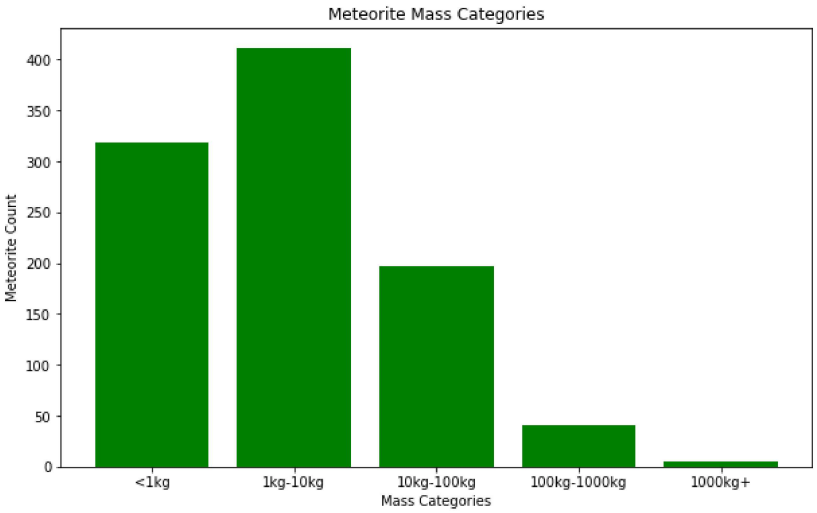
```
plt.figure(figsize=(10, 8))
plt.scatter(df['reclong'], df['reclat'], s=10, alpha=0.5, color='orange')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Meteorite Coordinates')
plt.show()
```



In [17]:

```
mass_categories = ['<1kg', '1kg-10kg', '10kg-100kg', '100kg-1000kg', '1000kg+']
mass_counts = df.groupby(pd.cut(df['mass'], bins=[0, 1000, 10000, 100000, 1000000, float('inf')])).size()
plt.figure(figsize=(10, 6))
plt.bar(mass_categories, mass_counts, color='green')
plt.xlabel('Mass Categories')
plt.ylabel('Meteorite Count')
plt.title('Meteorite Mass Categories')
plt.show()

```



In [ ]:

```
1
```