# Preserving Earth: Impact of Carbon Dioxide Emissions on Ice Caps

## 1. Introduction

### 1.1 Relevance

Climate change is one of the biggest challenges we have to face. This project aims to highlight the severity of the problem. This research will study the relationship between CO2 emissions and the melting of ice caps, to contribute to the understanding of our climate.

CO2 is a greenhouse gas that is released by burning of fossil fuels such as coal, oil, and natural gas. The overconsumption of these energy sources release emissions into the atmosphere which traps heat close to the Earth's surface. This causes a phenomenon called the greenhouse effect that raises the average temperature of the planet. This causes many climate issues such as the decline of biodiveristy, natural habitats, and extreme weather events. Not to mention, they add to respiratory problems due to air pollution.

High CO2 emission also causes the melting of ice caps. Glaciers survive by seasonal snowfall and cold temperatures. The greenhouse effect is warming the climate, and thus causing ice to melt. Loss of glaciers will cause rise of sea levels, loss of habitat for animals that live there, and also affect societies that depend on glaciers for water and agriculture.

### 1.2 Scope of Research

The focus is mainly on global CO2 emissions and how it affects glacier conditions over time. While there are many research done on this topic, this research will use long-term historical data to analyse trends over the past two centuries. Unlike other research, I will focus on glaciers on a global scale rather than regional glacier conditions.

Here is a more specific breakdown of the scope:

#### Inclusion Criteria

- Time Period: Data from the years 1990 to 2020 for CO2 emissions from World Bank and 1750 to 2020 from Global Carbon Project (GCP).
- Data Source: Only data from trusted sources such as the World Bank, Global Carbon Project, and the National Snow and Ice Data Center.
- Data Type: CO2 emissions data and glacier conditions data.
- Geographical Coverage: Global data for CO2 emissions and specific geographical locations for glacier conditions.

- Data Quality: Data that is complete and has been verified by the respective organizations.

### Exclusion Criteria

- Incomplete Data: Any data entries that are missing significant information or have gaps.
- Unverified Sources: Data from sources that are not recognized or verified by reputable organizations.
- Irrelevant Data: Data that does not pertain to reginonal CO2 emissions or glacier conditions.
- Outliers: Data points that are extreme outliers and do not fit within the expected range of values.
- Duplicate Data: Any duplicate entries from the same or different sources.

These criteria help ensure that the data used for analysis is reliable, relevant, and comprehensive.

## 1.3 Data Processing Pipeline

1. Acquire data:

- CO2 emission data scraped from Macrotrends.net, combined with data downloaded from Zenodo, forming a datafile with data from 1750 to 2020.
- Glacier data downloaded from NSIDC's Kaggle profile, forming a datafile with data from 1900 to 2000.

2. Clean data:

- Remove missing and irrelevant data
- Fill in missing data using various methods
- Standardise data types, units, and format
- Align data by time and merge datafiles

3. Exploratory Data Analysis:

- Plot CO2 emissions and glacier metrics over time
- Do correlation analysis
- Do statistical analysis (polynomial regression)

4. Visualisation and Conclusion

- Plot graphs to visualise
- Summarise findings

# 2. Aims and Objectives

## 2.1 Objectives

1. gather data on CO2 emissions and glacier conditions
2. observe CO2 emissions and glacier condition trend over time
3. observe relationship between CO2 emissions and glaciers

### 2.1 Evaluation of Aims

I will evaluate the success of my research by its ability to identify a trend or correlation between CO2 emissions and glacier data. Followed by whether my research is able to discover valuable insights through analysis. Lastly, if my research can match my hypothesis that CO2 emissions contribute to the melting of glaciers.

Let's look at some data I have collected.

# 3. Data

## 3.1 Data Collection and Rationale

I gathered data from two sources for CO2 emissions. The first is scraped from the website 'www.macrotrends.net', a website that displays data for global average CO2 emissions from the year 1990-2020. MacroTrends takes its data from World Bank, an organisation dedicated to environmental preservation among many others. Since MacroTrends takes data from trusted organisations like the World Bank, we can ensure accuracy of their data. With only 30 years worth of data, it is not enough to make a good analysis on CO2 emission trends, so I had to find another source.

I downloaded the second source for CO2 emissions from the website 'https://www.zenodo.org/'. It is a digital library that allows users to have access to data and resources. Zenodo takes its data from the Global Carbon Project (GCP), an organisation dedicated to reducing carbon emissions. This dataset has information from 1750 to 2020, which is much more than the first source. The historical data is what makes this source very valuable. Instead of only taking this one, I decided to combine these two data sources and use them for analysis.

Data gathered for glacier conditions was from the National Snow and Ice Data Center (NSIDC)'s Kaggle profile. NSIDC is a trusted organisation that specialises in monitoring and providing access to data on earth's cryospheric related regions. This makes the dataset and relevant to the research. The data has information about glacier conditions from 1900 to 2000. It has information regarding geographical location, glacier area, length and elevation. Their dataset has the CC0: Public Domain license which states that the data is open for public use. Which means I can use it for analysis.

## 3.2 Data Provenance

MacroTrends takes and aggregates data from the World Bank. Founded in 1944, the World Bank documents environmental data from government sources and reputable environmental organisations.

Data processing for MacroTrends dataset:

1. Web Scraping
2. Check for missing values and duplicates
3. Rename columns for clarity

Zenodo keeps an archive of research data taken from GCP. Founded in 2001, GCP collects and makes a scientific calculation to come up with a best estimate of CO2 emissions data. Here is the actual source https://zenodo.org/records/7215364

Data processing for Zenodo dataset:

1. Remove unecessary columns
2. Rename columns for clarity
3. Round values to same decimal points for consistency
4. Sort 'Year' column by ascending order for consistency
5. Check for missing values, data types, and duplicates

Merge the MacroTrends and Zenodo dataset:

1. Merge both datasets by year in ascending order
2. Aggregate values with overlapping year

Established in 1982, NSIDC uses satellite imagery, scientific models, and field observations to collect and document data. They also source data from NASA Earth-observing satellite (EOS) missions. These EOS missions collect data regarding Earth's atmosphere, oceans, ice, and biosphere. The National Aeronautics and Space Administration (NASA) is the world's top organisation in observing Earth's systems.

Data processing for NSIDC dataset:

1. Ensure correct data types for consistency
2. Drop unecessary columns
3. Merge 'Topographic Map Year' and 'Photograph Year' into 'Year Recorded'
4. Remove columns and rows with more than 50% missing values
5. Remove outliers
6. Fill columns 'Minimum Elevation', 'Maximum Elevation', 'Glacier Area', 'Maximum Length' with median value
7. Fill in columns 'mean elevation' and 'mean depth' using linear regression
8. Sort 'Year' in ascending order
9. Create 'continent_ID' column to prepare for iterative imputation
10. Fill in missing values in 'Year' column using iterative imputation

11. Fill in remaining missing values in 'Year' column using linear interpolation.
12. Remove 'continent_ID' column
13. Round 'Year Recorded' to nearest whole number, then convert to integer data type
14. Sort 'Year Recorded' in ascending order
15. Reset index
16. Round all float values to 2 decimal places

## 3.3 Format of Data

The datafile are all csv type files because I am primarily working with numerical and categorical data. I can manipulate the columns and rows for data cleaning.

## 3.4 Alternative Considerations

Another dataset that I considered for CO2 emissions was from the Emissions Database for Global Atmospheric Research (EDGAR). They have a dataset with global average CO2 emission which is what the project is focusing on. They have data from 1970 to 2023, which is good information. However, it is still shorter than what I already have from the Zenodo dataset (1750-2020). So I decided not to use this.

Another dataset that I considered for glacier data was the Randolph Glacier Inventory (RGI). It gives data for glaciers with area and ice thickness which is what my research needs. However, it does not indicate when they recorded each entry. A lot of cleaning would be needed to transform it into a more appropriate format for time analysis. This might disrupt the integrity of the data, therefore I did not choose to use this.

# 4. Ethical Considerations

## 4.1 Use/Reusage of Data

Data is open to the public so I am permitted to conduct analysis with it. This analysis can be used for further research. All data sources is properly credited under the 'References' section in the analysis.

## 4.2 Potential for Discrimination

This analysis does not contain personal information so there is no potential for discrimination against certain demographics or groups.e

## 4.3 Harmful Assumptions

This analysis focuses on the topic of CO2 emissions and its effect on glaciers. Please note to be careful when drawing conclusions when it comes to topics like climate change so as to

not spread misinformation about environmental issues. For example, results that show weak correlations should not be misinterpreted as direct causation.

## 4.4 Data Processing Pipeline

All data is publicly available, so there is no need to anonymise ita Every step of data cleaning and manipulation is clearly shown in the notebook with clear explanation

## 4.5 Potential Biases

The glacier data might have some biases where glacier data may come from certain regions more than others. This may be because certain regions has better resources to monitor glacier data (e.g. Arctic, Greenland, the Alps etc). While other regions are underrepresented and lead to missing data. This leads to overrepresentation of data for certain regions which might will effect global average if data is skewed towards one bias. CO2 emissions data spans from 1750 which might not be properly recorded since modern records, mid 20th century onwards, are more reliable. This leads to less accurate records which may affect the results of long term CO2 emissions.

In [ ]:
```
!pip install -r requirements.txt
```

# 5. Web Scraping

In [1]:
```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import WebDriverWait
import json
import os
import pandas as pd

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

url = 'https://www.macrotrends.net/global-metrics/countries/WLD/world/carbon-co2-em

# Open the page (fail if site don't exist)
try:
    driver.get(url)

    WebDriverWait(driver, 30).until(
        lambda d: d.execute_script('return window.chartData != undefined;')
    )
    # Get chart data
    chart_data = driver.execute_script('return window.chartData;')

    if chart_data:
        print("Chart data retrieved successfully!")

        if isinstance(chart_data, str):
```

```
            chart_data = json.loads(chart_data)

        # Place chart data into dataframe
        df = pd.DataFrame(chart_data)
        print(df)

        # Save data to a JSON file
        with open('archive/scraped_data.json', 'w') as json_file:
            json.dump(chart_data, json_file)

    else:
        print("Failed to retrieve data.")

except Exception as e:
    print(f"Error occurred: {e}")
    print("Site is down. Falling back to backup data...")

    # If scraping fails (e.g., site is down), load from JSON file
    if os.path.exists('archive/scraped_data.json'):
        with open('archive/scraped_data.json', 'r') as f:
            chart_data = json.load(f)
        df = pd.DataFrame(chart_data)
        print("Loaded data from backup:", df)
    else:
        print("No backup data available.")

finally:
    driver.quit()
```

```
Chart data retrieved successfully!
         date           v1      v2
0   1990-12-31  21284042.79  4.0208
1   1991-12-31  21440490.20  3.9833
2   1992-12-31  21390040.84  3.9102
3   1993-12-31  21531833.41  3.8749
4   1994-12-31  21676894.94  3.8420
5   1995-12-31  22299203.28  3.8938
6   1996-12-31  22778983.29  3.9195
7   1997-12-31  23202623.39  3.9352
8   1998-12-31  23365547.83  3.9074
9   1999-12-31  23530421.41  3.8814
10  2000-12-31  24280271.96  3.9516
11  2001-12-31  24644437.95  3.9580
12  2002-12-31  24989512.93  3.9614
13  2003-12-31  26133423.08  4.0900
14  2004-12-31  27331852.94  4.2237
15  2005-12-31  28371817.05  4.3297
16  2006-12-31  29308175.54  4.4171
17  2007-12-31  30418994.07  4.5283
18  2008-12-31  30632298.65  4.5038
19  2009-12-31  30238050.03  4.3915
20  2010-12-31  32095872.94  4.6049
21  2011-12-31  33079721.35  4.6895
22  2012-12-31  33460087.50  4.6853
23  2013-12-31  34119894.39  4.7196
24  2014-12-31  34261369.66  4.6822
25  2015-12-31  34070176.85  4.6014
26  2016-12-31  34145652.30  4.5583
27  2017-12-31  34687837.09  4.5780
28  2018-12-31  35560555.79  4.6417
29  2019-12-31  35477245.40  4.5820
30  2020-12-31  33566427.59  4.2917
```

The URL I will be scraping is macrotrends.net for CO2 emission data. Selenium WebDriver will be used instead of BeautifulSoup because the website relies on JavaScript to load content dynamically into the page. I confirmed this when I inspect the page and the table, which is called 'chartData', has JavaScript script tags. I included error handling in the code such that if the site is down, the program will fall back to my backup JSON file and retrieve data from there.

I placed previously scraped data in the JSON file. I also have error handling where I state that there is no backup data. If this error is shown, it might mean the JSON file is not in the archive directory. The scraped data has a table of 30 entries with the headings not explicitely named, but it is the date, global average CO2 emissions, and metric tons per capita. This data is only for 30 years which is a short time to analyse CO2 emissions. Nevetheless, it is still valuable data and I will clean and process it to use in my analysis.

# 6. Data Cleaning

In [3]:
```
# check for missing values and duplicates
print(df.isnull().sum())
print(df.duplicated().sum())
```

```
date    0
v1      0
v2      0
dtype: int64
0
```

This datafile has no missing values and duplicates in all columns.

In [4]:
```
chartData = [{'date': '1990-12-31', 'v1': 21284042.79, 'v2': 4.0208}, {'date': '199

df = pd.DataFrame(chartData)

# Data cleaning
df.rename(columns={
    'date': 'Date',
    'v1': 'CO2_Emissions_Kilotons',
    'v2': 'CO2_Metric_Tons_per_Capita'
}, inplace=True)

df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df.drop('Date', axis=1, inplace=True)
df = df[['Year', 'CO2_Emissions_Kilotons', 'CO2_Metric_Tons_per_Capita']]

print(df.head())
```

```
   Year  CO2_Emissions_Kilotons  CO2_Metric_Tons_per_Capita
0  1990             21284042.79                      4.0208
1  1991             21440490.20                      3.9833
2  1992             21390040.84                      3.9102
3  1993             21531833.41                      3.8749
4  1994             21676894.94                      3.8420
```

I renamed the columns for clarity, and I extracted the year from the dates. I only need the year data, because a precise measurement of the day and month is not a necessary detail, since the research focuses on trends by the year.

In [5]:
```
# Check and filter for outliers
Q1 = df['CO2_Emissions_Kilotons'].quantile(0.25)
Q3 = df['CO2_Emissions_Kilotons'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
filtered_df = df[(df['CO2_Emissions_Kilotons'] >= lower_bound) & (df['CO2_Emissions
original_size, filtered_size = len(df), len(filtered_df)

print(f"Original Data Size: {original_size}")
print(f"Filtered Data Size: {filtered_size}")
```

```
Original Data Size: 31
Filtered Data Size: 31
```

There are no outliers. The original data and filtered data size is the same at 31. This means no data has been filtered.

In [6]:
```
# Save and rename
file_name = 'archive/cleaned_co2_30_years.csv'
filtered_df.to_csv(file_name, index=False)

print(f"Data saved to {file_name}")
```

```
Data saved to archive/cleaned_co2_30_years.csv
```

In [7]:
```
# Read new dataset
pd.set_option('display.max_columns', None)
file_name = 'archive/GCB2022v27_MtCO2_flat.csv'
df = pd.read_csv(file_name)

print(df.head())
```

```
       Country ISO 3166-1 alpha-3  Year  Total  Coal  Oil  Gas  Cement  \
0  Afghanistan                AFG  1750    0.0   NaN  NaN  NaN     NaN
1  Afghanistan                AFG  1751    0.0   NaN  NaN  NaN     NaN
2  Afghanistan                AFG  1752    0.0   NaN  NaN  NaN     NaN
3  Afghanistan                AFG  1753    0.0   NaN  NaN  NaN     NaN
4  Afghanistan                AFG  1754    0.0   NaN  NaN  NaN     NaN

   Flaring  Other  Per Capita
0      NaN    NaN         NaN
1      NaN    NaN         NaN
2      NaN    NaN         NaN
3      NaN    NaN         NaN
4      NaN    NaN         NaN
```

This datafile is obtained from the Global Carbon Project that I downloaded from the site zenodo.org. It has regional data for CO2 emissions, and data for the type of fossil fuel that is consumed. However, in this research I am focusing on global average. So I will take only the global data and only the average emissions.

I select only 'Global' rows under the 'Country' column, then I drop the 'Country' column since I no longer need it after filtering. I choose to keep the columns 'Total' and 'Per Capita' , and rename them to 'CO2_Emissions_Kilotons' and 'CO2_Metric_Tons_per_Capita' respectively so I can match the first datafile's column names before merging them. I set them to the same unit and round them to the same decimal values for consistency. I sort the year by ascending order, again to match with the first datafile.

In [8]:
```
# Data cleaning
df_cleaned = df[['Country', 'Year', 'Total', 'Per Capita']]
df_cleaned = df_cleaned[df_cleaned['Country'] == 'Global']
df_cleaned = df_cleaned.drop(columns=['Country'])
df_cleaned = df_cleaned.sort_values(by='Year', ascending=True)
df_cleaned['Total'] = df_cleaned['Total'] * 1000
df_cleaned.rename(columns={'Total': 'CO2_Emissions_Kilotons'}, inplace=True)
df_cleaned = df_cleaned.rename(columns={'Per Capita': 'CO2_Metric_Tons_per_Capita'}
df_cleaned['CO2_Emissions_Kilotons'] = df_cleaned['CO2_Emissions_Kilotons'].round(2
```

```
df_cleaned['CO2_Metric_Tons_per_Capita'] = df_cleaned['CO2_Metric_Tons_per_Capita']
df_cleaned = df_cleaned.reset_index(drop=True)

print(df_cleaned.head())
```

```
   Year  CO2_Emissions_Kilotons  CO2_Metric_Tons_per_Capita
0  1750                 9350.53                      0.0115
1  1751                 9350.53                      0.0114
2  1752                 9354.19                      0.0114
3  1753                 9354.19                      0.0113
4  1754                 9357.86                      0.0113
```

In [9]:
```
year_summary = df_cleaned['Year'].describe()
print(year_summary)
```

```
count     272.000000
mean     1885.500000
std        78.663842
min      1750.000000
25%      1817.750000
50%      1885.500000
75%      1953.250000
max      2021.000000
Name: Year, dtype: float64
```

This dataset has records from 1750 to 2021.

In [10]:
```
# Check for missing values, data types, and duplicates
missing_values = df_cleaned.isnull().sum()
data_types = df_cleaned.dtypes
duplicates = df_cleaned[df_cleaned.duplicated()]

print("Missing Values:\n", missing_values)
print("\nData Types:\n", data_types)
print(f"\nNumber of exact duplicate rows: {len(duplicates)}")
```

```
Missing Values:
 Year                          0
CO2_Emissions_Kilotons         0
CO2_Metric_Tons_per_Capita     0
dtype: int64

Data Types:
 Year                            int64
CO2_Emissions_Kilotons        float64
CO2_Metric_Tons_per_Capita    float64
dtype: object

Number of exact duplicate rows: 0
```

The data has no missing values and no duplicates. Data types seem to be correct.

In [11]:
```
# Identify outliers for 'CO2_Emissions_Kilotons' and 'CO2_Metric_Tons_per_Capita'
Q1 = df_cleaned['CO2_Emissions_Kilotons'].quantile(0.25)
Q3 = df_cleaned['CO2_Emissions_Kilotons'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
```

```
upper_bound = Q3 + 1.5 * IQR
outliers_emissions = df_cleaned[(df_cleaned['CO2_Emissions_Kilotons'] < lower_bound
Q1_per_capita = df_cleaned['CO2_Metric_Tons_per_Capita'].quantile(0.25)
Q3_per_capita = df_cleaned['CO2_Metric_Tons_per_Capita'].quantile(0.75)
IQR_per_capita = Q3_per_capita - Q1_per_capita
lower_bound_per_capita = Q1_per_capita - 1.5 * IQR_per_capita
upper_bound_per_capita = Q3_per_capita + 1.5 * IQR_per_capita
outliers_per_capita = df_cleaned[(df_cleaned['CO2_Metric_Tons_per_Capita'] < lower_

print(f"Number of outliers in 'CO2_Emissions_Kilotons': {len(outliers_emissions)}")
print(f"Number of outliers in 'CO2_Metric_Tons_per_Capita': {len(outliers_per_capit
```

```
Number of outliers in 'CO2_Emissions_Kilotons': 49
Number of outliers in 'CO2_Metric_Tons_per_Capita': 0
```

I choose to keep the outliers of CO2 emission as removing them might remove valuable context (e.g. global industrial activities). However, I will handle them by excluding them from calculations (e.g. finding the mean or mode).

In [12]:
```
# Check for duplicates in 'Year'
num_rows = df_cleaned.shape[0]
unique_years = df_cleaned['Year'].nunique()
duplicates = df_cleaned[df_cleaned['Year'].duplicated()]

print(f"Number of rows: {num_rows}")
print(f"Number of duplicate rows in 'Year' column: {duplicates.shape[0]}")
print(f"Number of unique years: {unique_years}")
```

```
Number of rows: 272
Number of duplicate rows in 'Year' column: 0
Number of unique years: 272
```

Since there are no duplicates and missing values, that might mean there is one entry for each year.

In [13]:
```
file_name = 'archive/cleaned_co2_3_centuries.csv'
df_cleaned.to_csv(file_name, index=False)

print(f"Data saved to {file_name}")
```

```
Data saved to archive/cleaned_co2_3_centuries.csv
```

In [14]:
```
# Data cleaning
df_century = pd.read_csv('archive/cleaned_co2_3_centuries.csv')
df_30_years = pd.read_csv('archive/cleaned_co2_30_years.csv')

merged_df = pd.merge(df_century, df_30_years, on='Year', how='outer', suffixes=('_c
merged_df['CO2_Emissions_Kilotons'] = merged_df[['CO2_Emissions_Kilotons_century',
merged_df['CO2_Metric_Tons_per_Capita'] = merged_df[['CO2_Metric_Tons_per_Capita_ce
merged_df.drop(columns=['CO2_Emissions_Kilotons_century', 'CO2_Emissions_Kilotons_3
                        'CO2_Metric_Tons_per_Capita_century', 'CO2_Metric_Tons_per
merged_df['CO2_Emissions_Kilotons'] = merged_df['CO2_Emissions_Kilotons'].round(2)
merged_df['CO2_Metric_Tons_per_Capita'] = merged_df['CO2_Metric_Tons_per_Capita'].r

print(merged_df.head())
```

```
merged_file_name = 'archive/merged_co2_data.csv'
merged_df.to_csv(merged_file_name, index=False)
print(f"Data saved to {merged_file_name}")
```

```
   Year  CO2_Emissions_Kilotons  CO2_Metric_Tons_per_Capita
0  1750                 9350.53                      0.0115
1  1751                 9350.53                      0.0114
2  1752                 9354.19                      0.0114
3  1753                 9354.19                      0.0113
4  1754                 9357.86                      0.0113
Data saved to archive/merged_co2_data.csv
```

I merged 'cleaned_co2_3_centuries.csv' and 'cleaned_co2_30_years.csv' to form one ultimate dataset, 'merged_co2_data.csv'. Rows with overlapping years, I took the average values of CO2 emissions and metric tons per capita. The ultimate CO2 emissions dataset is saved to current directory for further processing.

In [15]:
```
# Read dataset for glaciers
glacier_file_name = 'archive/database_glacier.csv'
glacier_df = pd.read_csv(glacier_file_name, low_memory=False)
print(glacier_df.head())
```

| | Glacier ID | Political Unit | Continent | Basin Code | Location Code | Glacier Code | Glacier Name | Latitude | Longitude | Primary Class | Glacier Source | Basin Count | Glacier Form | Glacier Activity | Activity Start | Activity End | Minimum Elevation | Minimum Elevation Exposed | Mean Elevation | Mean Elevation Accumulation | Mean Elevation Ablation | Maximum Elevation | Snow Line Elevation | Snow Line Accuracy | Glacier Area | Area Accuracy | Area Exposed | Mean Width | Mean Length | Maximum Length | Maximum Length Exposed | Maximum Length Ablation | Mean Depth |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | AF5Q112B0001 | AFGHANISTAN | ASIA | Q112 | B0 | 1 | NaN | 34.672 | 68.874 | 9.0 | 0.0 | NaN | 3.0 | 0.0 | NaN | NaN | 3975.0 | NaN | 4110.0 | NaN | NaN | 4250.0 | NaN | NaN | 1.28 | 3.0 | NaN | NaN | NaN | 1.9 | NaN | NaN | NaN |
| 1 | AF5Q112B0002 | AFGHANISTAN | ASIA | Q112 | B0 | 2 | NaN | 34.676 | 68.855 | 9.0 | 0.0 | NaN | 3.0 | 0.0 | NaN | NaN | 4250.0 | NaN | 4350.0 | NaN | NaN | 4450.0 | NaN | NaN | 0.31 | 3.0 | NaN | NaN | NaN | 0.8 | NaN | NaN | NaN |
| 2 | AF5Q112B0003 | AFGHANISTAN | ASIA | Q112 | B0 | 3 | NaN | 34.689 | 68.854 | 9.0 | 0.0 | NaN | 3.0 | 0.0 | NaN | NaN | 4000.0 | NaN | 4100.0 | NaN | NaN | 4200.0 | NaN | NaN | 0.60 | 3.0 | NaN | NaN | NaN | 1.5 | NaN | NaN | NaN |
| 3 | AF5Q112B0004 | AFGHANISTAN | ASIA | Q112 | B0 | 4 | NaN | 34.707 | 68.857 | 9.0 | 0.0 | NaN | 3.0 | 0.0 | NaN | NaN | 4000.0 | NaN | 4175.0 | NaN | NaN | 4350.0 | NaN | NaN | 0.56 | 3.0 | NaN | NaN | NaN | 1.5 | NaN | NaN | NaN |
| 4 | AF5Q112B0005 | AFGHANISTAN | ASIA | Q112 | B0 | 5 | NaN | 34.719 | 68.852 | 9.0 | 0.0 | NaN | 2.0 | 0.0 | NaN | NaN | 3750.0 | NaN | 4050.0 | NaN | NaN | 4350.0 | NaN | NaN | 1.06 | 3.0 | NaN | NaN | NaN | 2.0 | NaN | NaN | NaN |

```
   Depth Accuracy Accumulation Orientation Ablation Orientation  \
0             NaN                       NE                   NE
1             NaN                       NW                   NW
2             NaN                       NW                   NW
3             NaN                       NE                   NE
4             NaN                      NaN                  NaN

   Topographic Map Year  Topographic Map Scale  Photograph Year
0                1959.0               100000.0              NaN
1                1959.0               100000.0              NaN
2                1959.0               100000.0              NaN
3                1959.0               100000.0              NaN
4                1959.0               100000.0              NaN
```

Since my research is aiming to learn the relationship between CO2 emissions and the melting of ice caps, the 'Glacier Code' will have a categorical identifier rather than a numerical value. In simple terms, it's like having a name for each glacier, even though its name is a number. Hence, I will change the entire column to be string type.

In [16]:
```
# Changing 'Glacier Code' column values to string
glacier_df['Glacier Code'] = glacier_df['Glacier Code'].astype(str)
print(glacier_df.iloc[:, 5].apply(type).value_counts())
```

```
Glacier Code
<class 'str'>    132890
Name: count, dtype: int64
```

In [17]:
```
# Print each column
for column in glacier_df.columns:
    print(column)
```

```
Glacier ID
Political Unit
Continent
Basin Code
Location Code
Glacier Code
Glacier Name
Latitude
Longitude
Primary Class
Glacier Source
Basin Count
Glacier Form
Glacier Activity
Activity Start
Activity End
Minimum Elevation
Minimum Elevation Exposed
Mean Elevation
Mean Elevation Accumulation
Mean Elevation Ablation
Maximum Elevation
Snow Line Elevation
Snow Line Accuracy
Glacier Area
Area Accuracy
Area Exposed
Mean Width
Mean Length
Maximum Length
Maximum Length Exposed
Maximum Length Ablation
Mean Depth
Depth Accuracy
Accumulation Orientation
Ablation Orientation
Topographic Map Year
Topographic Map Scale
Photograph Year
```

The dataset for glaciers has many columns. However, not every column is necessary for the analysis. Let's take a look at which columns can be excluded.

'Glacier Name' - 'Glacier ID' is sufficient name classification.

'Political Unit' - 'Region' column is sufficient geographical classification.

'Latitude', 'Longitude', 'Basin Code', 'Location Code' - specific locale not needed.

'Glacier Code' and 'Primary Class' - 'Glacier ID' is sufficient identification.

'Glacier Source' - glacier origin not relevant to melting patterns.

'Minimum Elevation Exposed', 'Mean Elevation Accumulation', 'Maximum Length Exposed' - 'Glacier Area' is a more accurate metric relating to the research topic.

'Area Exposed' and 'Basin Count' - does not contribute to research topic.

'Glacier Form' - physical form of glaciers is not the focus.

'Activity Start' and 'Activity End' - specific events of glacier activity not needed.

'Area Accuracy', 'Depth Accuracy', 'Snow Line Accuracy' - mean values are sufficient; accuracy levels will not impact research.

'Topographic Map Scale' - does not contribute to research topic.

In [18]:
```
# List of columns to drop
columns_to_drop = [
    'Latitude',
    'Longitude',
    'Political Unit',
    'Basin Code',
    'Location Code',
    'Glacier Code',
    'Primary Class',
    'Glacier Source',
    'Minimum Elevation Exposed',
    'Mean Elevation Accumulation',
    'Maximum Length Exposed',
    'Area Exposed',
    'Basin Count',
    'Glacier Form',
    'Glacier Activity',
    'Activity Start',
    'Activity End',
    'Area Accuracy',
    'Depth Accuracy',
    'Snow Line Accuracy',
    'Topographic Map Scale',
    'Glacier Name',
    'Accumulation Orientation',
    'Ablation Orientation'
]

glacier_df_cleaned = glacier_df.drop(columns=columns_to_drop)

for column in glacier_df_cleaned.columns:
    print(column)
```

```
Glacier ID
Continent
Minimum Elevation
Mean Elevation
Mean Elevation Ablation
Maximum Elevation
Snow Line Elevation
Glacier Area
Mean Width
Mean Length
Maximum Length
Maximum Length Ablation
Mean Depth
Topographic Map Year
Photograph Year
```

This dataset has two time classification columns, 'Topographic Map Year' and ' Photograph Year'. The two will be merged into a single column, 'Year Recorded'. The latest year from either column will be used to represent the most recent record available for the glacier. This will simplify the data without losing valuable information.

In [19]:
```
# Merge 'Topographic Map Year' and 'Photograph Year' into 'Year Recorded'
glacier_df_cleaned['Year Recorded'] = glacier_df_cleaned[['Topographic Map Year', '
glacier_df_cleaned = glacier_df_cleaned.drop(columns=['Topographic Map Year', 'Phot

print(glacier_df_cleaned['Year Recorded'].head())
```

```
0    1959.0
1    1959.0
2    1959.0
3    1959.0
4    1959.0
Name: Year Recorded, dtype: float64
```

In [20]:
```
# Check data types of each column
data_types = glacier_df_cleaned.dtypes
print("Data Types:\n", glacier_df_cleaned.dtypes)
```

```
Data Types:
 Glacier ID                 object
Continent                  object
Minimum Elevation         float64
Mean Elevation            float64
Mean Elevation Ablation   float64
Maximum Elevation         float64
Snow Line Elevation       float64
Glacier Area              float64
Mean Width                float64
Mean Length               float64
Maximum Length            float64
Maximum Length Ablation   float64
Mean Depth                float64
Year Recorded             float64
dtype: object
```

all data types seem to be correct

In [21]:
```
# percentage of missing data in each column
missing_percentage = glacier_df_cleaned.isnull().mean() * 100
print(missing_percentage)
```

```
Glacier ID                 0.000000
Continent                  0.000000
Minimum Elevation         11.835353
Mean Elevation            38.596584
Mean Elevation Ablation   91.682595
Maximum Elevation         13.110091
Snow Line Elevation       78.291820
Glacier Area               3.398299
Mean Width                47.350440
Mean Length               58.418241
Maximum Length            22.798555
Maximum Length Ablation   87.169087
Mean Depth                47.271427
Year Recorded             23.228986
dtype: float64
```

This dataset has alot of missing values. This is unfortunate as I would have to fill in the values to the best of my ability so I can do further analysis. It would mean losing accuracy of data. This would affect trends and compromise the reliability of the data.

In [22]:
```
# Drop columns and rows with more than 50% missing values
columns_to_drop = missing_percentage[missing_percentage > 50].index
glacier_df_cleaned = glacier_df_cleaned.drop(columns=columns_to_drop)
threshold = 0.5
missing_percentage = glacier_df_cleaned.isnull().mean(axis=1)
glacier_df_cleaned = glacier_df_cleaned[missing_percentage <= threshold]

# Removing outliers
columns_to_check = ['Minimum Elevation', 'Mean Elevation', 'Maximum Elevation', 'Gl
Q1 = glacier_df_cleaned[columns_to_check].quantile(0.25)
Q3 = glacier_df_cleaned[columns_to_check].quantile(0.75)
IQR = Q3 - Q1
non_outliers = ~((glacier_df_cleaned[columns_to_check] < (Q1 - 1.5 * IQR)) | (glaci
glacier_df_no_outliers = glacier_df_cleaned[non_outliers]

print(f"Remaining rows and columns: {glacier_df_no_outliers.shape}")
```

```
Remaining rows and columns: (103454, 10)
```

I choose to remove outliers because i will use linear regression to impute the missing values for the 'mean elevation' column using information from the column 'minimum elevation' and 'maximum elevation', as these two coulumns are the most related to finding the mean elevation of the glaciers. similarly, I filled in the missing values for 'mean depth' and 'mean width' column using data from 'maximum length' and 'glacier area' column.

FIrstly, the columns 'Minimum Elevation', 'Maximum Elevation', 'Glacier Area', 'Maximum Length' will be filled in with a median value.

In [23]:
```
# Fill columns with median
columns_to_fill = ['Minimum Elevation', 'Maximum Elevation', 'Glacier Area', 'Maxim
glacier_df_cleaned = glacier_df_cleaned.copy()
glacier_df_cleaned[columns_to_fill] = glacier_df_cleaned[columns_to_fill].apply(
    lambda col: col.fillna(col.median())
)
```

Before I use linear regression to impute values, I will verify the accuracy of the model by running a train test split. I will split data into a training set and a testing set. The training set will be used to train the model, and the accuracy will be verified using the testing set.

In [24]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_absolute_error, r2_score

# Model and evaluation
def reg_and_eval(glacier_df_cleaned, predictors, target):
    print(f"\n--- Scores for {target} ---")
    X = glacier_df_cleaned[predictors]
    y = glacier_df_cleaned[target]

    X_missing = X[y.isnull()]
    y_missing = y[y.isnull()]
    X_train = X[~y.isnull()]
    y_train = y[~y.isnull()]

    X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(
        X_train, y_train, test_size=0.2, random_state=42
    )

    model = LinearRegression()
    model.fit(X_train_split, y_train_split)
    y_val_pred = model.predict(X_val_split)
    mae = mean_absolute_error(y_val_split, y_val_pred)
    r2 = r2_score(y_val_split, y_val_pred)
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_ab

    print(f"MAE: {mae:.2f}")
    print(f"R-squared: {r2:.2f}")
    print(f"Cross-Validation MAE: {cv_scores.mean():.2f}")

    if not X_missing.empty:
        y_imputed = model.predict(X_missing)
        glacier_df_cleaned.loc[glacier_df_cleaned[target].isnull(), target] = y_imp
        print(f"Imputed missing values for {target}.")

# Impute missing values for the 3 columns
reg_and_eval(
    glacier_df_cleaned,
    predictors=['Minimum Elevation', 'Maximum Elevation'],
    target='Mean Elevation'
)

reg_and_eval(
```

```
    glacier_df_cleaned,
    predictors=['Maximum Length', 'Glacier Area'],
    target='Mean Depth'
)

reg_and_eval(
    glacier_df_cleaned,
    predictors=['Maximum Length', 'Glacier Area'],
    target='Mean Width'
)
```

```
--- Scores for Mean Elevation ---
MAE: 31.54
R-squared: 1.00
Cross-Validation MAE: -32.20
Imputed missing values for Mean Elevation.

--- Scores for Mean Depth ---
MAE: 10.70
R-squared: 0.11
Cross-Validation MAE: -10.56
Imputed missing values for Mean Depth.

--- Scores for Mean Width ---
MAE: 0.25
R-squared: 0.29
Cross-Validation MAE: -0.25
Imputed missing values for Mean Width.
```

Scoring definition:

MAE means mean absolute error. It is the average value of how far off the prediction is from the actual value. A smaller MAE is a better performance. R-squared is the percentage of how much the model can explain the results in the target variable. It ranges from 0(very weak) to 1(perfectly explains results). Cross-validation MAE is the result of splitting data into multiple pairs of training and validation sets. In this case, we split it 5 times and take the average MAE. A smaller value is a better performance.

Result conclusions:

Mean elevataion MAE is 31.54 units off from the actual value, consistent with the cross-validation result (-32.20). The R^2 value shows a perfect variance in the data. This is a very high number which may mean overfitting in small datasets. However, the glacier dataset is decently large, so I can conclude that the regression model works well with mean elevation.

Mean depth MAE is 10.70, with a consistent result from cross-validation (-10.56). However, the R^2 value is 0.11 which is very low. I can conclude that the model predicts well, but will hinder the accuracy of the dataset since it does not explain the variance in the data well.

Mean width MAE is 0.25, with a consistent result from cross-validation (-0.25). The R^2 value (0.29) suggests a better explanation in variance of data than mean depth. Therefore, this model works better for mean width than mean depth.

Next steps: The regression model works best with mean elevation, very poorly with mean depth, and moderately for mean width. However, I will still proceed to use linear regression to fill in data for all 3 columns. Despite not having an ideal result, this would be the best strategy to clean up missing values for further analysis.

In [25]:
```
missing_percentage = glacier_df_cleaned.isna().mean() * 100
print(missing_percentage)
```

```
Glacier ID            0.00000
Continent             0.00000
Minimum Elevation     0.00000
Mean Elevation        0.00000
Maximum Elevation     0.00000
Glacier Area          0.00000
Mean Width            0.00000
Maximum Length        0.00000
Mean Depth            0.00000
Year Recorded        23.41198
dtype: float64
```

There is still 23.41% of missing values for 'Year Recorded' column.

In [26]:
```
# Sort year chronologically
glacier_df_cleaned = glacier_df_cleaned.sort_values(by='Year Recorded')
print(glacier_df_cleaned[['Year Recorded']].head())
```

```
       Year Recorded
17539         1904.0
15258         1907.0
21810         1929.0
9939          1931.0
3092          1939.0
```

In [27]:
```
# Check unique glacier IDs
unique_glacier_ids = glacier_df_cleaned['Glacier ID'].nunique()
print(f"Unique Glacier IDs: {unique_glacier_ids}")
```

```
Unique Glacier IDs: 121852
```

Here I am checking the correlation between 'year recorded' column with the other columns. However, 'glacier id' and 'continent' is a categorial variable, so I have to encode it into a numerical variable first. I choose to exclude 'glacier id' column as it has too many unique values. It would not be practical to handle each one as its own category.

In [28]:
```
# Map continent to integer
continent_mapping = glacier_df_cleaned['Continent'].astype('category').cat.categori
glacier_df_cleaned['continent_ID'] = glacier_df_cleaned['Continent'].astype('catego
print(glacier_df_cleaned[['Continent', 'continent_ID']].head())

glacier_df_cleaned_no_id = glacier_df_cleaned.drop(columns=['Glacier ID', 'Continen
```

```
           Continent  continent_ID
17539  NORTH AMERICA             4
15258  NORTH AMERICA             4
21810  NORTH AMERICA             4
9939   NORTH AMERICA             4
3092   SOUTH AMERICA             5
```

'continent_ID' column values is continent encoded into a numerical value.

In [29]:
```
correlation_matrix = glacier_df_cleaned_no_id.corr()
print(correlation_matrix['Year Recorded'])
```

```
Minimum Elevation    0.346810
Mean Elevation       0.344017
Maximum Elevation    0.331474
Glacier Area         0.019035
Mean Width          -0.010267
Maximum Length      -0.045586
Mean Depth           0.005695
Year Recorded        1.000000
continent_ID        -0.338996
Name: Year Recorded, dtype: float64
```

Based on the results of the correlation matrix, 'year recorded' has the highest correlation with the 'Minimum Elevation', ' Mean Elevation' and ' Maximum Elevation' columns. The iterative imputation method will be used to fill in the missing values in 'year recorded' , taking into account the three related columns.

In [30]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Iterative imputation
features = glacier_df_cleaned[['Minimum Elevation', 'Mean Elevation', 'Maximum Elev
iterative_imputer = IterativeImputer(max_iter=10, random_state=42)
imputed_values = iterative_imputer.fit_transform(features)
imputed_df = pd.DataFrame(imputed_values, columns=features.columns)
glacier_df_cleaned['Year Recorded'] = imputed_df['Year Recorded']

missing_percentage = glacier_df_cleaned['Year Recorded'].isnull().mean() * 100
print(f"Percentage of missing values in 'Year Recorded': {missing_percentage:.2f}%"
```

```
Percentage of missing values in 'Year Recorded': 8.67%
```

I managed to bring down the percentage of missing values of 'year recorded' from 23.41% to 8.67%. the remaining 8.67% will be filled in using a linear interpolation.

In [31]:
```
# Apply linear interpolation
glacier_df_cleaned['Year Recorded'] = glacier_df_cleaned['Year Recorded'].interpola
missing_values = glacier_df_cleaned['Year Recorded'].isnull().sum()

print(f"Number of missing values for Year Recorded: {missing_values}")
```

```
Number of missing values for Year Recorded: 0
```

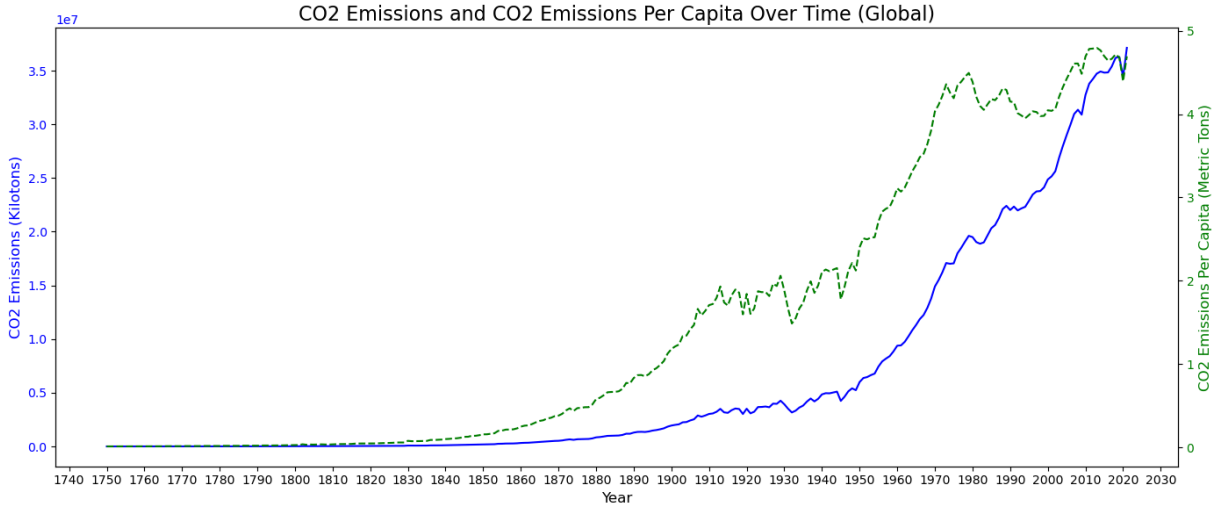
In [32]:
```
# Data cleaning
glacier_df_cleaned = glacier_df_cleaned.drop(columns=['continent_ID'])
glacier_df_cleaned['Year Recorded'] = glacier_df_cleaned['Year Recorded'].round()
glacier_df_cleaned['Year Recorded'] = glacier_df_cleaned['Year Recorded'].astype(in
glacier_df_cleaned = glacier_df_cleaned.sort_values(by='Year Recorded', ascending=T
glacier_df_cleaned = glacier_df_cleaned.reset_index(drop=True)
float_columns = glacier_df_cleaned.select_dtypes(include=['float64']).columns
glacier_df_cleaned[float_columns] = glacier_df_cleaned[float_columns].round(2)

print(glacier_df_cleaned.dtypes)
print(glacier_df_cleaned.head())

cleaned_file_name = 'archive/cleaned_glacier_data.csv'
glacier_df_cleaned.to_csv(cleaned_file_name, index=False)
print(f"Data saved to {cleaned_file_name}")
```

```
Glacier ID            object
Continent             object
Minimum Elevation    float64
Mean Elevation       float64
Maximum Elevation    float64
Glacier Area         float64
Mean Width           float64
Maximum Length       float64
Mean Depth           float64
Year Recorded          int32
dtype: object
     Glacier ID Continent  Minimum Elevation  Mean Elevation  \
0  AF5Q112B0001      ASIA             3975.0          4110.0
1  AF5Q112B0002      ASIA             4250.0          4350.0
2  AF5Q112B0003      ASIA             4000.0          4100.0
3  AF5Q112B0004      ASIA             4000.0          4175.0
4  AF5Q112B0006      ASIA             3550.0          3900.0

   Maximum Elevation  Glacier Area  Mean Width  Maximum Length  Mean Depth  \
0             4250.0          1.28        0.67             1.9       35.13
1             4450.0          0.31        0.48             0.8       23.37
2             4200.0          0.60        0.60             1.5       30.86
3             4350.0          0.56        0.60             1.5       30.86
4             4250.0          0.75        0.68             2.0       36.20

   Year Recorded
0           1904
1           1907
2           1929
3           1931
4           1939
Data saved to archive/cleaned_glacier_data.csv
```

# 7. Exploratary Data Analysis

In [33]:
```
import matplotlib.pyplot as plt
from matplotlib import ticker
```

```
merged_file_name = 'archive/merged_co2_data.csv'
df_merged = pd.read_csv(merged_file_name)

# Plot a graph of 'CO2_Emissions_Kilotons' and 'CO2_Metric_Tons_per_Capita' over ti
fig, ax1 = plt.subplots(figsize=(14, 6))
ax1.plot(df_merged['Year'], df_merged['CO2_Emissions_Kilotons'], color='blue', labe
ax1.set_xlabel('Year', fontsize=12)
ax1.set_ylabel('CO2 Emissions (Kilotons)', fontsize=12, color='blue')
ax1.tick_params(axis='y', labelcolor='blue')
ax2 = ax1.twinx()
ax2.plot(df_merged['Year'], df_merged['CO2_Metric_Tons_per_Capita'], color='green',
ax2.set_ylabel('CO2 Emissions Per Capita (Metric Tons)', fontsize=12, color='green'
ax2.tick_params(axis='y', labelcolor='green')
plt.title('CO2 Emissions and CO2 Emissions Per Capita Over Time (Global)', fontsize
plt.grid(False)
ax1.xaxis.set_major_locator(ticker.MultipleLocator(10))
ax1.xaxis.set_major_formatter(ticker.FuncFormatter(lambda x, _: f'{int(x)}'))
plt.xticks(fontsize=8)

fig.tight_layout()
plt.show()
```

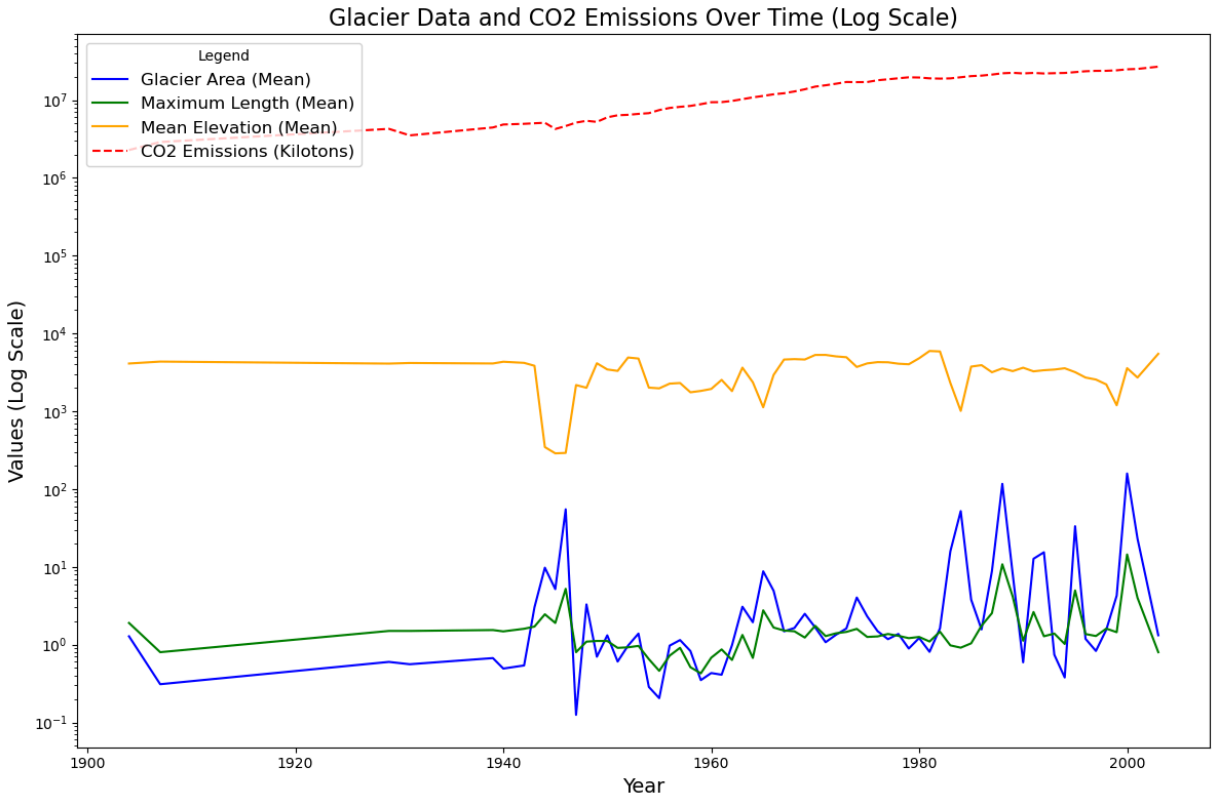



This is a graph of CO2 emissions and CO2 per capita over time. The graph shows that there is a sharp increase in CO2 emissions starting from the mid 19th century(1850-1860).

During this time, the industrial revolution had spread to Europe and the United States. The world experienced a growth in technological advancements which introduced the use of fossil fuels. These are new energy sources that when consumed, will release CO2. These advancements are what likely caused the spike in emissions.

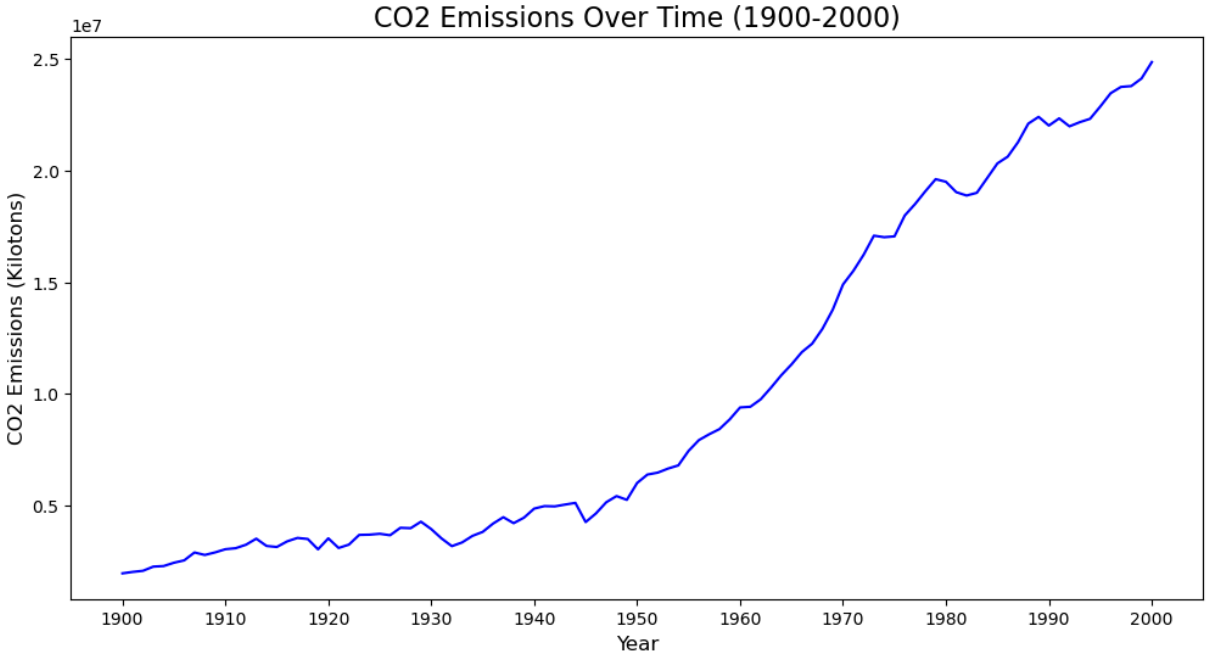
In [34]:
```
import seaborn as sns

glacier_data = pd.read_csv("archive/cleaned_glacier_data.csv")
co2_data = pd.read_csv("archive/merged_co2_data.csv")

# Aggregate and merge glacier data by year
glacier_aggregated = glacier_data.groupby("Year Recorded").agg({
    "Glacier Area": "mean",
```

```
    "Maximum Length": "mean",
    "Mean Elevation": "mean",
}).reset_index().rename(columns={"Year Recorded": "Year"})

merged_data = pd.merge(glacier_aggregated, co2_data, on="Year", how="inner")

# Plotting
plt.figure(figsize=(12, 8))
sns.lineplot(
    data=merged_data,
    x="Year",
    y="Glacier Area",
    label="Glacier Area (Mean)",
    color="blue",
)

sns.lineplot(
    data=merged_data,
    x="Year",
    y="Maximum Length",
    label="Maximum Length (Mean)",
    color="green",
)

sns.lineplot(
    data=merged_data,
    x="Year",
    y="Mean Elevation",
    label="Mean Elevation (Mean)",
    color="orange",
)

sns.lineplot(
    data=merged_data,
    x="Year",
    y="CO2_Emissions_Kilotons",
    label="CO2 Emissions (Kilotons)",
    color="red",
    linestyle="--",
)

# Log scale for y-axis
plt.yscale("log")

plt.title("Glacier Data and CO2 Emissions Over Time (Log Scale)", fontsize=16)
plt.xlabel("Year", fontsize=14)
plt.ylabel("Values (Log Scale)", fontsize=14)
plt.legend(title="Legend", fontsize=12)
plt.grid(False)

plt.tight_layout()
plt.show()
```




A logarithmic scale is used for the y-axis because a linear scale would cause large values to overshadow smaller values, preventing a clear visual on the graph. The log scale spreads the values evenly which helps us focus on how the data is changing over time. In simple terms, making the small values more obvious and the large values less extreme. From this graph, we can see that CO2 emissions is increasing at a constant rate. The glacier area, length and elevation stays stagnant until the mid 20th century. From that point onwards, the glacier data fluctuates rapidly.

In [35]:
```
co2_data = pd.read_csv('archive/merged_co2_data.csv')

co2_data_filtered = co2_data[(co2_data['Year'] >= 1900) & (co2_data['Year'] <= 2000
plt.figure(figsize=(12, 6))
plt.plot(co2_data_filtered['Year'], co2_data_filtered['CO2_Emissions_Kilotons'], li
plt.title('CO2 Emissions Over Time (1900-2000)', fontsize=16)
plt.xlabel('Year', fontsize=12)
plt.ylabel('CO2 Emissions (Kilotons)', fontsize=12)
plt.xticks(range(1900, 2001, 10))
plt.grid(False)
plt.show()
```




This is a linear graph of CO2 emissions over time from 1900 to 2000. We can observe that co2 emissions increased sharply from the mid 20th century. It could be possible that the sharp increase of co2 emissions has affected our environment such that the glaciers are experiencing rapid changes.

Let's find out if this is the case by doing a correlation analysis to determine if CO2 emissions has any relation to glacier conditions.

In [36]:
```
from scipy.stats import pearsonr

glacier_data = pd.read_csv('archive/cleaned_glacier_data.csv')
co2_data = pd.read_csv('archive/merged_co2_data.csv')

# Merge datasets
merged_data = pd.merge(glacier_data, co2_data, left_on='Year Recorded', right_on='Y
merged_data = merged_data.drop(columns=['Year Recorded'])
cols = ['Year'] + [col for col in merged_data if col != 'Year']
merged_data = merged_data[cols]
merged_data.to_csv('archive/merged_co2_glacier_data.csv', index=False)

# Calculate the mean
glacier_yearly_data = merged_data.groupby('Year')[['Glacier Area', 'Maximum Length'
co2_yearly_data = merged_data.groupby('Year')['CO2_Emissions_Kilotons'].mean()

# Correlation and p-value
def calculate_correlation_and_pvalue(x, y):
    correlation, p_value = pearsonr(x, y)
    return correlation, p_value

correlation_area, p_value_area = calculate_correlation_and_pvalue(glacier_yearly_da
correlation_length, p_value_length = calculate_correlation_and_pvalue(glacier_yearl
correlation_elevation, p_value_elevation = calculate_correlation_and_pvalue(glacier
```

```
print(f"Correlation between Glacier Area and CO2 Emissions: {correlation_area}, P-v
print(f"Correlation between Glacier Length and CO2 Emissions: {correlation_length},
print(f"Correlation between Glacier Elevation and CO2 Emissions: {correlation_eleva

# Time-Lag
co2_column = 'CO2_Emissions_Kilotons'
glacier_column = 'Glacier Area'
max_lag = 10
lags = range(-max_lag, max_lag + 1)
correlations_lag = []
p_values_lag = []

for lag in lags:
    if lag < 0:
        lagged_co2 = merged_data[co2_column].shift(-lag)
        glacier_values = merged_data[glacier_column]
    else:
        lagged_co2 = merged_data[co2_column]
        glacier_values = merged_data[glacier_column].shift(lag)

    valid_indices = ~lagged_co2.isna() & ~glacier_values.isna()
    lagged_co2 = lagged_co2[valid_indices]
    glacier_values = glacier_values[valid_indices]

    if len(lagged_co2) > 1:
        correlation, p_value = pearsonr(lagged_co2, glacier_values)
        correlations_lag.append(correlation)
        p_values_lag.append(p_value)
    else:
        correlations_lag.append(np.nan)  # Not enough data for correlation (error h
        p_values_lag.append(np.nan)

lag_results = pd.DataFrame({'Lag': lags, 'Correlation': correlations_lag, 'P-value'
optimal_lag = lag_results.loc[lag_results['Correlation'].idxmax()]

print("Optimal Lag:", optimal_lag['Lag'], "years")
print(f"Correlation between Glacier area and CO2 Emissions after lag: {optimal_lag[
print(f"P-value at optimal lag: {lag_results.loc[lag_results['Lag'] == optimal_lag[
```

```
Correlation between Glacier Area and CO2 Emissions: 0.2750948606861497, P-value: 0.0
24256939308768016
Correlation between Glacier Length and CO2 Emissions: 0.29064658976848184, P-value:
0.017031332071203262
Correlation between Glacier Elevation and CO2 Emissions: 0.17637682531179902, P-valu
e: 0.15336158125577976
Optimal Lag: 10.0 years
Correlation between Glacier area and CO2 Emissions after lag: 0.021937082257742664
P-value at optimal lag: 1.808694755463033e-14
```

Here I performed correlation and P-value analysis between CO2 emissions and glacier metrics. The correlation coefficient measures the strength of a relationship between two variables. It ranges from -1(strong negative) to 1(strong positive). 0 indicates no correlation. The P-value measures if the correlation is statistically significant. This means it measures if the relationship is likely to be real or a coincidence. $p < 0.05$ is a statistically significent correlation.

There is a weak positive correlation between glacier area and CO2 emissions (0.275). The P-value is statistically significant (0.024), showing a real but weak relationship.

There is another weak positive correlation between glacier length and CO2 emissions (0.290). The P-value is statistically significant (0.017), also showing a real but weak relationship.

There is an even weaker correlation between glacier elevation and CO2 emissions (0.176). The P-value is not statistically significant (0.153), showing no relationship.

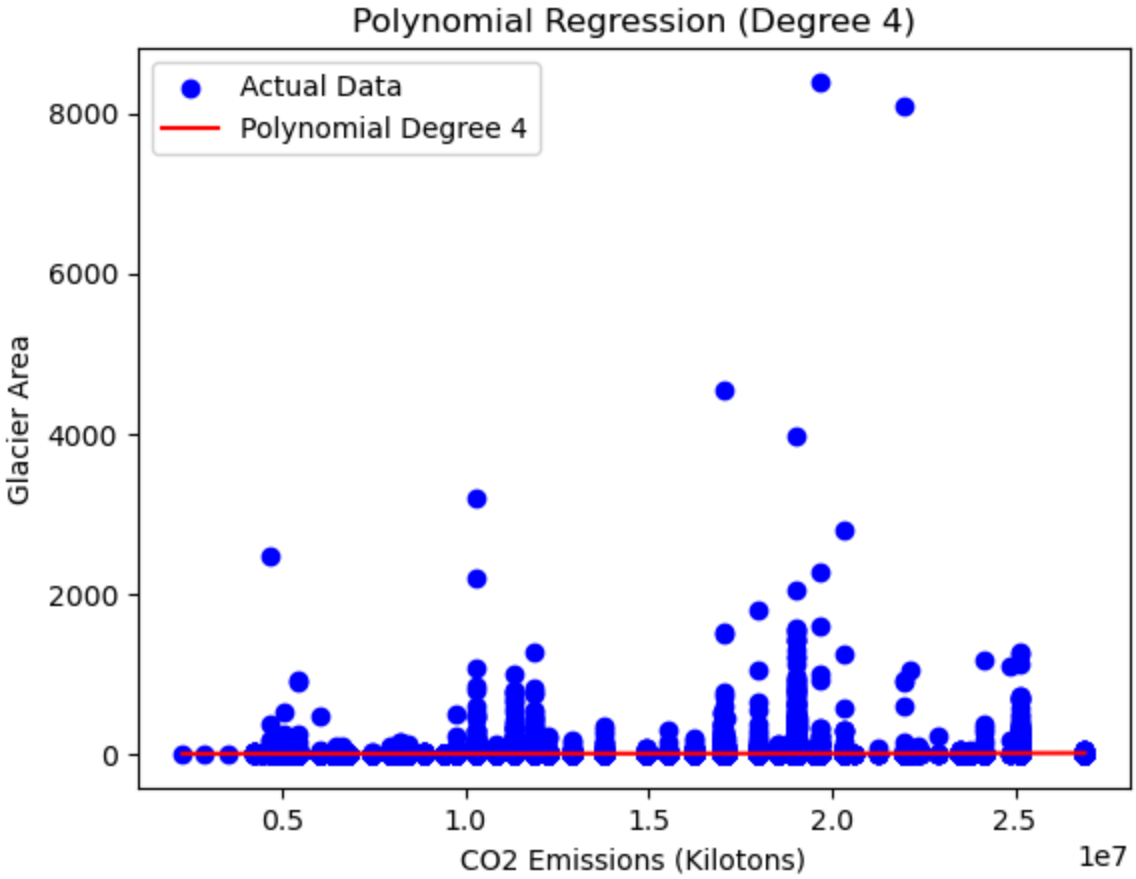
I also added a time lag factor (6 years) as glaciers might take time to adjust to the effects. I did this for glacier area and CO2 emissions. The correlation (0.022) is still low, meaning the relationship is very weak. Interestingly, the P-value (1.797e-14) is extremely small, indicating the correlation after the lag is statistically significant. The P-value is significant enough to show that the time-lagged correlation between the two variables might still be important, however small.

The provides insights into the relationship between CO2 emissions and glacier conditions. The area and length, although showing a weak relationship with CO2 emissions, are both significantly influenced by it. The elevation is not correlated or statistically influenced. The optimal lag of 6 years suggest there may be a delayed effect of CO2 emissions on glacier area, but the relationship between them is very weak.

It is also important to note that while P-value is influenced by the magnitude of correlation, it is also more likely to achieve a statisticaly significant value (e.g. 0.01) with a larger sample size. The sample size I use for this analysis is very big. Therefore, even with a statistically significant result, I will still conclude that the relationship between CO2 emissions and glaciers are complicated and likely involve other factors.

In [37]:
```
from sklearn.preprocessing import PolynomialFeatures

data = pd.read_csv('archive/merged_co2_glacier_data.csv')

# Polynomial Regression
X = data[['CO2_Emissions_Kilotons']]
y = data['Glacier Area']
degree = 4
poly = PolynomialFeatures(degree=degree)
X_poly = poly.fit_transform(X)

model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)
r_squared = model.score(X_poly, y)

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label=f'Polynomial Degree {degree}')
plt.xlabel('CO2 Emissions (Kilotons)')
plt.ylabel('Glacier Area')
plt.title(f'Polynomial Regression (Degree {degree})')
```

```
plt.legend()
plt.show()

print(f"R² Score: {r_squared}")
```



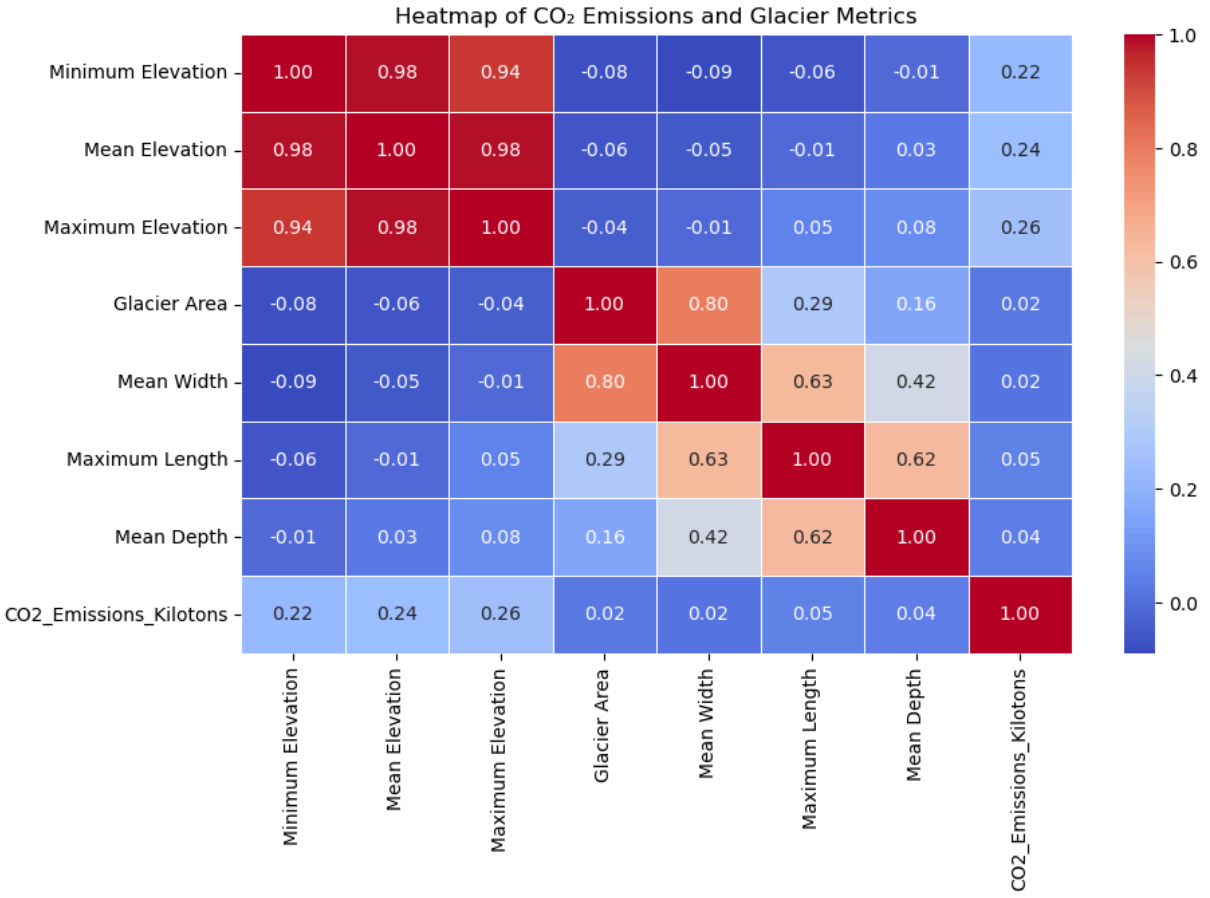

```
R² Score: 0.0006806918005253237
```

Here I performed a degree 4 polynomial regression analysis between CO2 emissions and glacier area to explore non-linear trends. This is to observe the rate of change in glacier area based on CO2 emissions. The R^2 score is very low, meaning that the CO2 emissions does not explain the changes in glacier area. This indicates that CO2 emissions alone is not a strong factor impacting glacier area.

The scatter plot does not show a clear pattern between both variables, showing the weak relationship.

In [38]:
```
data = pd.read_csv('archive/merged_co2_glacier_data.csv')

variables = ['Minimum Elevation', 'Mean Elevation', 'Maximum Elevation',
             'Glacier Area', 'Mean Width', 'Maximum Length', 'Mean Depth',
             'CO2_Emissions_Kilotons']

correlation_matrix = data[variables].corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=
```

```
plt.title('Heatmap of CO₂ Emissions and Glacier Metrics')
plt.show()
```




This is a heatmap to see how correlated each variable is from each other. The elevation variables has the highest correlation with CO2 emissions, but a weak value of about 0.22-0.26. Glacier area is strongly related to its width. Glacier depth and width are moderately related to its length.

# 8. Conclusion

In conclusion, there are many complex factors involved which prevents us from making a strong causation between CO2 emissions and glacier conditions. Although the research found a weak positive correlation, it is still too weak that I cannot give a firm answer to my hypothesis that CO2 emissions cause melting of glaciers. That is a limitation of this research that could potentially be explored further, particularly in looking at other factors that might explain the results.

# 9. References

Macrotrends.net, 2024. *World Carbon (CO2) Emissions 1990-2024*. Available at: https://www.macrotrends.net/global-metrics/countries/WLD/world/carbon-co2-emissions [Accessed 13 December 2024]

Andrew, R. M. and Peters, G. P. (2022) 'The Global Carbon Project's fossil CO2 emissions dataset'. Zenodo. doi: 10.5281/zenodo.721536 [Accessed 13 December 2024]

National Snow and Ice Data Center (NSIDC), n.d. *Glacier Inventory*. Available at: https://nsidc.org/data/glacier_inventory/ [Accessed 14 December 2024].

NSIDC, 2021. *Glacier Inventory*. Available at: https://www.kaggle.com/datasets/nsidcorg/glacier-inventory [Accessed 14 December 2024].

EDGAR (Emissions Database for Global Atmospheric Research) Community GHG Database (a collaboration between the European Commission, Joint Research Centre (JRC), the International Energy Agency (IEA), and comprising IEA-EDGAR CO2, EDGAR CH4, EDGAR N2O, EDGAR F-GASES version EDGAR_2024_GHG (2024) European Commission.

RGI Consortium. (2023). Randolph Glacier Inventory - A Dataset of Global Glacier Outlines. (NSIDC-0770, Version 7). [Data Set]. Boulder, Colorado USA. National Snow and Ice Data Center. https://doi.org/10.5067/F6JMOVY5NAVZ. Date Accessed 12-14-2024.

World Resources Institute (WRI), 2023. The History of Carbon Dioxide Emissions. Available at: https://www.wri.org/insights/history-carbon-dioxide-emissions [Accessed 14 December 2024].

Encyclopaedia Britannica, n.d. The First Industrial Revolution. Available at: https://www.britannica.com/event/Industrial-Revolution/The-first-Industrial-Revolution [Accessed 14 December 2024].

National Geographic, n.d. Greenhouse Gases. Available at: https://www.nationalgeographic.com/environment/article/greenhouse-gases [Accessed 14 December 2024].

University of Colorado Boulder, 2023. To Save the Planet's Glaciers, Human Actions Still Matter. Available at: https://www.colorado.edu/today/2023/11/20/save-planets-glaciers-human-actions-still-matter#:~:text=Losses%20of%20glacier%20ice%20around,water%20for%20agriculture%20and% [Accessed 14 December 2024].