CROATRAD

February 21, 2025

1 Analysis of CROATRAD.TXT from Statistics and Data Analysis in Geology by John C. Davis

In the book, the dataset has been used in Page No. 30 (Fig. 2.11) to show different histograms

1.1 What does the dataset contain?

The dataset contains measurement of absorbed background radiation levels per hour (unit: nGy/hr) in the Istrian Peninsula, Croatia. The radiation levels are measured for ^{40}K, ^{224}Ra, ^{228}Ra, ^{236}U and ^{137}Cs.

1. What is a Gray (Gy)?

- Gray (Gy) is the SI unit of absorbed radiation dose.
- It represents the amount of energy absorbed per kilogram of material.
- 1 Gy = 1 joule of radiation energy absorbed per kilogram of matter.

2. What is a Nanogray (nGy)?

- 1 Nanogray (nGy) = 10 Gray (Gy) = one billionth of a Gray.
- Since natural background radiation levels are very low, we use nGy instead of Gy.

3. What Does "per Hour" (nGy/hr) Mean?

- nGy/hr (Nanogray per Hour) measures the radiation dose rate.
- It tells how much radiation energy is absorbed every hour.

The last two columns show the Natural and the Total Radiation levels in the Istrian Peninsula.

Comparison with Safety Limits for Human Exposure

Source	Radiation Level	Comparison
Typical background radiation	50-200 nGy/hr	Matches most of the data
ICRP Public Exposure Limit (International Commission on Radiological Protection)	~1,000 nGy/hr (1 μGy/hr)	Much higher than the observed values
UNSCEAR (United Nations Scientific Committee on the Effects of Atomic Radiation) natural range	400-1,500 nGy/hr	Istria's values are well below concern levels
Chernobyl-affected areas today	5,000-20,000 nGy/hr	Much higher than Istrian levels

1.2 Load Necessary Python Modules

[1]:
```
import pandas as pd      # for file handling and manipulation of dataframes
```

[2]:
```
import numpy as np       # for handling arrays and for advanced histograms in␣
↪this chapter
```

[3]:
```
import matplotlib.pyplot as plt      # for data visualization
```

To view the plots within the IPython cell in Jupyter Notebook:

[4]:
```
%matplotlib inline
```

[5]:
```
import seaborn as sns     # for stastistical visualizations not covered by␣
↪matplotlib
```

1.3 Read the data and convert it into Data Frame in Python using pandas

[6]:
```
# Step 1: Load the Data in Python
file_path = "CROATRAD.TXT"
```

[7]:
```
# Create dataframe and convert the .txt file to csv using the first row of the .
↪txt file as the headers
df = pd.read_csv(file_path, delim_whitespace=True, names=["40K", "224Ra",␣
↪"228Ra", "236U", "137Cs", "Natural", "Total"])
df.head()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_28308\2950647005.py:2: FutureWarning:
The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed
in a future version. Use ``sep='\s+'`` instead
  df = pd.read_csv(file_path, delim_whitespace=True, names=["40K", "224Ra",
"228Ra", "236U", "137Cs", "Natural", "Total"])
```

[7]:

	40K	224Ra	228Ra	236U	137Cs	Natural	Total
0	40K	224Ra	228Ra	236U	137Cs	Natural	Total
1	13.7	41.9	24.4	0.1	2.8	80.2	83
2	14	52.6	48.4	0.4	3.3	115.4	118.7
3	12.3	40.6	27.7	0.2	1.6	80.8	82.3
4	13.2	45.9	29.6	0.4	2.5	89.2	91.6

Since the above code is being deprecated, the following code can be used:

```
df = pd.read_csv(file_path, sep=r'\s+', names=["40K", "224Ra", "228Ra", "236U",
"137Cs", "Natural", "Total"])
```

[8]:
```
df = pd.read_csv(file_path, sep=r'\s+', names=["40K", "224Ra", "228Ra", "236U",␣
↪"137Cs", "Natural", "Total"])
df.head()
```

[8]:

	40K	224Ra	228Ra	236U	137Cs	Natural	Total
0	40K	224Ra	228Ra	236U	137Cs	Natural	Total
1	13.7	41.9	24.4	0.1	2.8	80.2	83
2	14	52.6	48.4	0.4	3.3	115.4	118.7
3	12.3	40.6	27.7	0.2	1.6	80.8	82.3
4	13.2	45.9	29.6	0.4	2.5	89.2	91.6

However, in the output, the first row (indexed 0)is the same as the header. Therefore, the best code would be:

```
df = pd.read_csv(file_path, sep=r'\s+', header = 0)
```

[9]:
```
df = pd.read_csv(file_path, sep=r'\s+', header = 0)
df
```

[9]:

	40K	224Ra	228Ra	236U	137Cs	Natural	Total
0	13.7	41.9	24.4	0.1	2.8	80.2	83.0
1	14.0	52.6	48.4	0.4	3.3	115.4	118.7
2	12.3	40.6	27.7	0.2	1.6	80.8	82.3
3	13.2	45.9	29.6	0.4	2.5	89.2	91.6
4	11.3	40.6	21.6	0.1	1.6	73.6	75.2
..	…	…	…	…	…	…	…
120	15.4	22.6	12.7	0.1	4.7	50.8	55.6
121	16.7	14.6	8.5	0.1	3.4	39.9	43.2
122	15.6	14.6	8.0	0.2	1.2	38.4	39.6
123	18.7	47.3	35.7	0.2	1.8	101.9	103.7
124	13.3	40.6	31.0	0.2	0.8	85.2	85.9

[125 rows x 7 columns]

1.4 Measurement of Central Tendencies using Python

1.4.1 Means

[10]:
```
print(df["40K"].mean())
```

15.0224

[11]:
```
print(df["224Ra"].mean())
```

36.14160000000001

[12]:
```
print(df["228Ra"].mean())
```

36.8016

[13]:
```
print(df["236U"].mean())
```

0.28480000000000005

[14]:
```
print(f"{df["236U"].mean():.4f}")
```

```
0.2848
```

[15]:
```
print(df["137Cs"].mean())
```

```
6.177599999999999
```

[16]:
```
print(f"{df["137Cs"].mean():.4f}")
```

```
6.1776
```

[17]:
```
print(df["Natural"].mean())
```

```
88.25199999999998
```

[18]:
```
print(f"{df["Natural"].mean():.4f}")
```

```
88.2520
```

[19]:
```
print(df["Total"].mean())
```

```
94.4184
```

1.4.2 Medians

[20]:
```
print(df["40K"].median())
```

```
15.0
```

[21]:
```
print(df["224Ra"].median())
```

```
39.3
```

[22]:
```
print(df["228Ra"].median())
```

```
31.5
```

[23]:
```
print(df["236U"].median())
```

```
0.2
```

[24]:
```
print(df["137Cs"].median())
```

```
4.4
```

[25]:
```
print(df["Natural"].median())
```

```
90.8
```

[26]:
```
print(df["Total"].median())
```

```
96.0
```

1.4.3 Simpler way of Getting Descriptive Statistics

It's rather tedious to understand the descriptive statistics of the data in the way given above. Pandas provides a better solution. We can use:

`DataFrame.mean()` to find the mean of each column

`DataFrame.median()` to find the median of each column

...

We can also type:

`DataFrame.describe()` to get the descriptive statistics in one go.

[27]:
```
df.mean()
```

[27]:
```
40K         15.0224
224Ra       36.1416
228Ra       36.8016
236U         0.2848
137Cs        6.1776
Natural     88.2520
Total       94.4184
dtype: float64
```

[28]:
```
df.median()
```

[28]:
```
40K         15.0
224Ra       39.3
228Ra       31.5
236U         0.2
137Cs        4.4
Natural     90.8
Total       96.0
dtype: float64
```

[29]:
```
df.describe()
```

[29]:
```
              40K       224Ra       228Ra        236U       137Cs      Natural  \
count  125.000000  125.000000  125.000000  125.000000  125.000000  125.000000
mean    15.022400   36.141600   36.801600    0.284800    6.177600   88.252000
std      2.750304   11.922405   24.065301    0.180079    5.999541   31.467419
min      5.200000    8.000000    7.100000    0.000000    0.600000   31.600000
25%     13.200000   26.600000   21.600000    0.200000    2.400000   66.800000
50%     15.000000   39.300000   31.500000    0.200000    4.400000   90.800000
75%     16.700000   44.600000   47.000000    0.300000    7.500000  105.400000
max     22.400000   59.200000  146.200000    1.000000   37.100000  211.800000

             Total
count  125.000000
```

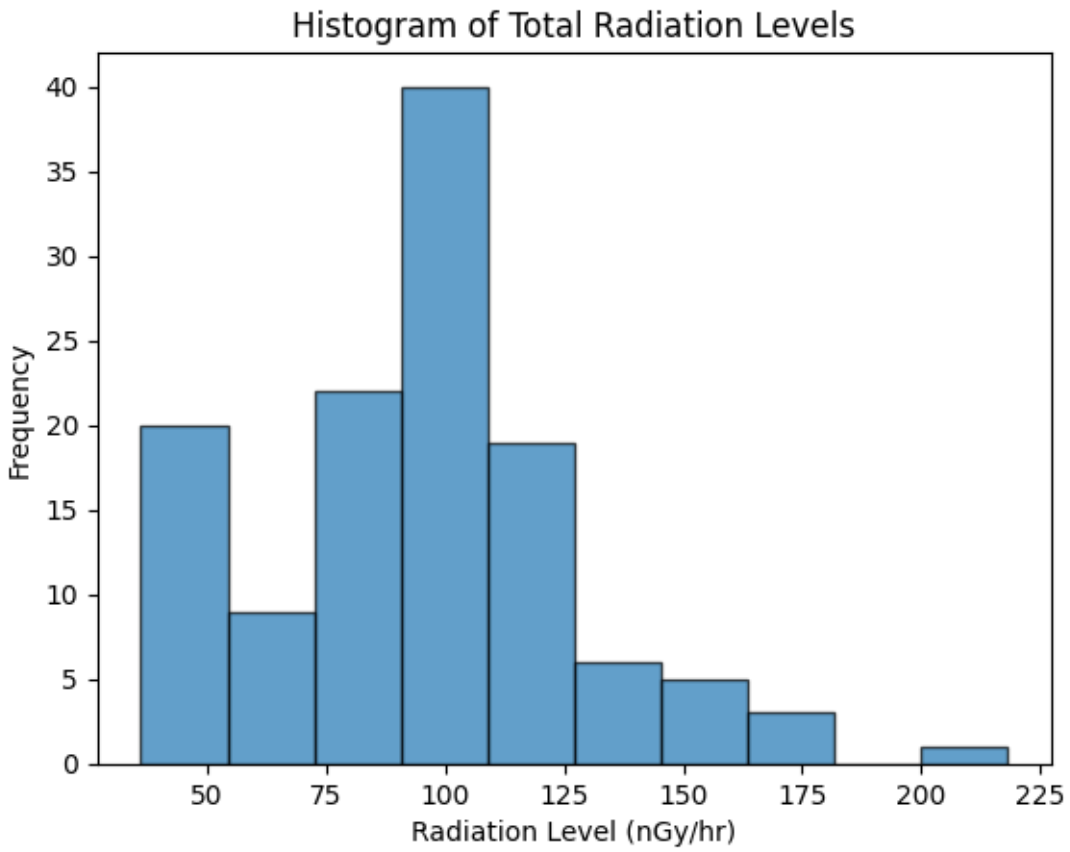
```
mean     94.418400
std      32.470407
min      36.000000
25%      77.100000
50%      96.000000
75%     109.700000
max     218.200000
```

1.5 Histogram for Total Radiation Dose Absorbed Per Hour using Python

1.5.1 Basic Histogram

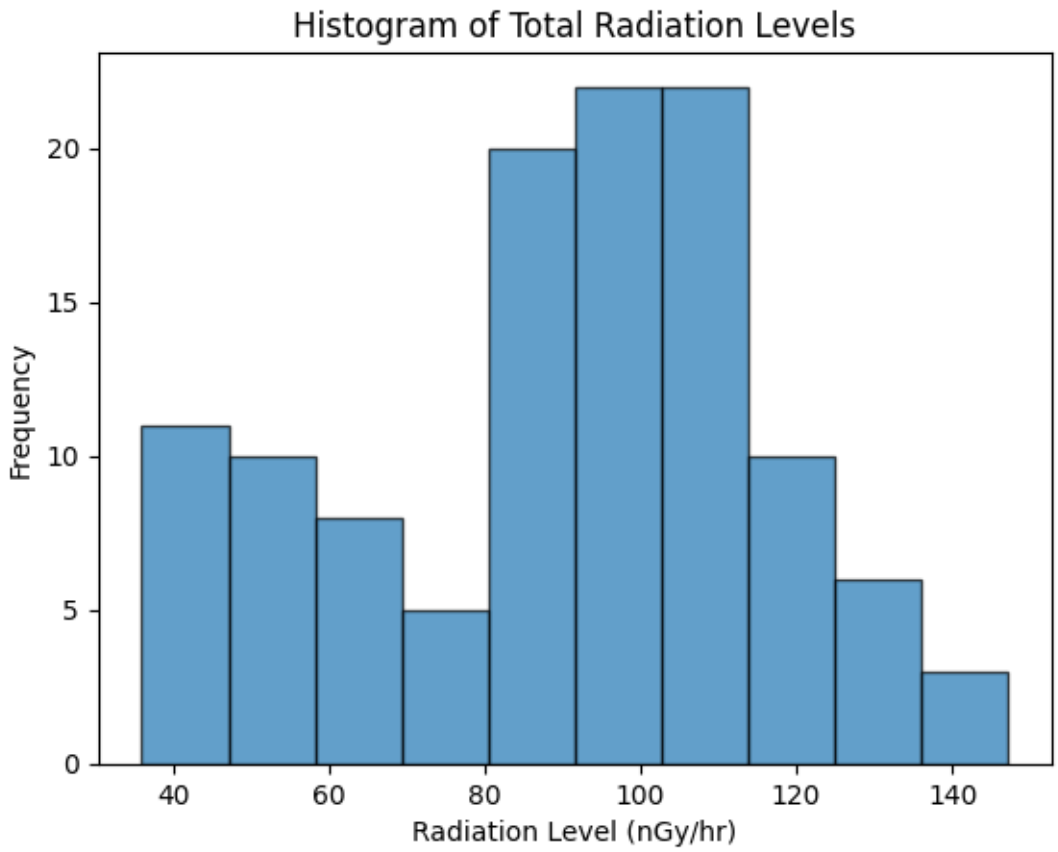
Using matplotlib to visualize the distribution of Total Radiation.

[30]:
```
# histogram
plt.hist(df["Total"], bins = 10, edgecolor = "black", alpha = 0.7)

# Labels and Title
plt.xlabel("Radiation Level (nGy/hr)")
plt.ylabel("Frequency")
plt.title("Histogram of Total Radiation Levels")

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

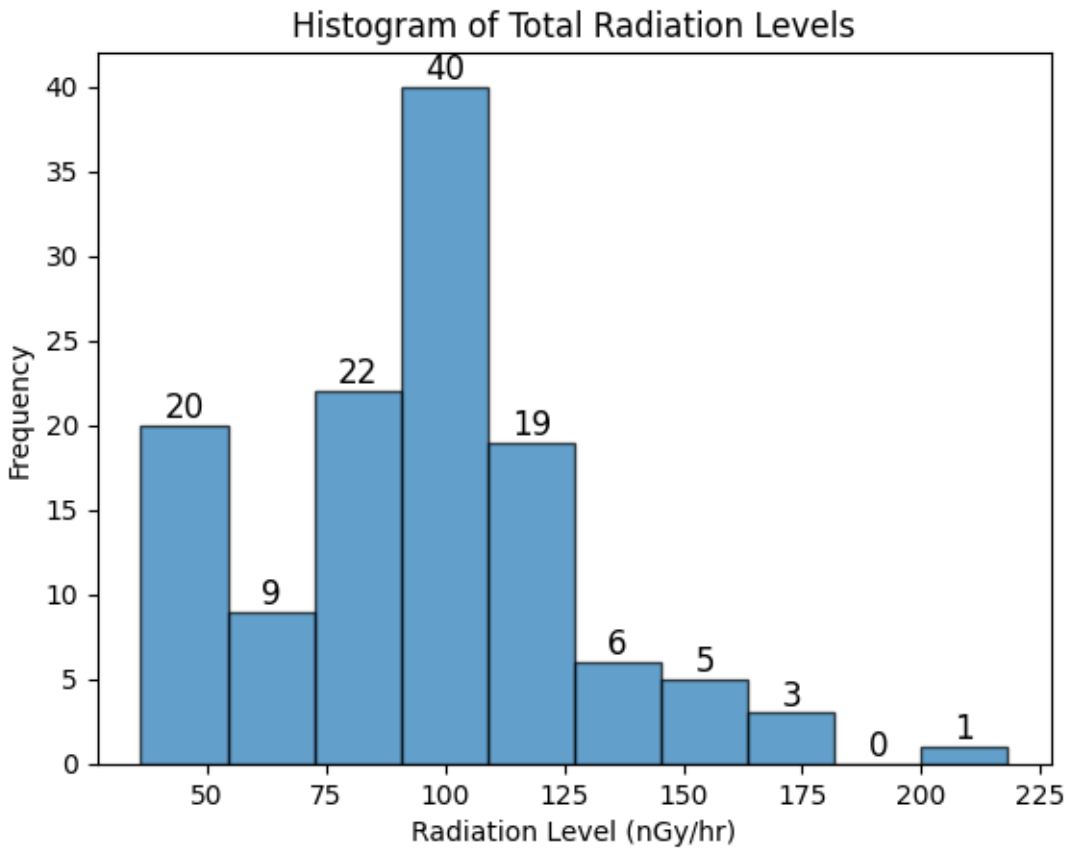


Basic Histogram taking values < 150 nGy/hr

[31]:
```
filtered_data = df[df["Total"] < 150]["Total"]
plt.hist(filtered_data, bins = 10, edgecolor = "black", alpha = 0.7)
# Labels and Title
plt.xlabel("Radiation Level (nGy/hr)")
plt.ylabel("Frequency")
plt.title("Histogram of Total Radiation Levels")

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



[32]:
```
counts, bin_edges, _ = plt.hist(df["Total"], bins=10, edgecolor="black",␣
 ↪alpha=0.7)

# Add labels on top of bars
for count, bin_edge in zip(counts, bin_edges[:-1]):
    plt.text(bin_edge + (bin_edges[1] - bin_edges[0]) / 2, count,␣
 ↪str(int(count)),
             ha='center', va='bottom', fontsize=12)
# Labels and Title
plt.xlabel("Radiation Level (nGy/hr)")
plt.ylabel("Frequency")
plt.title("Histogram of Total Radiation Levels")

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

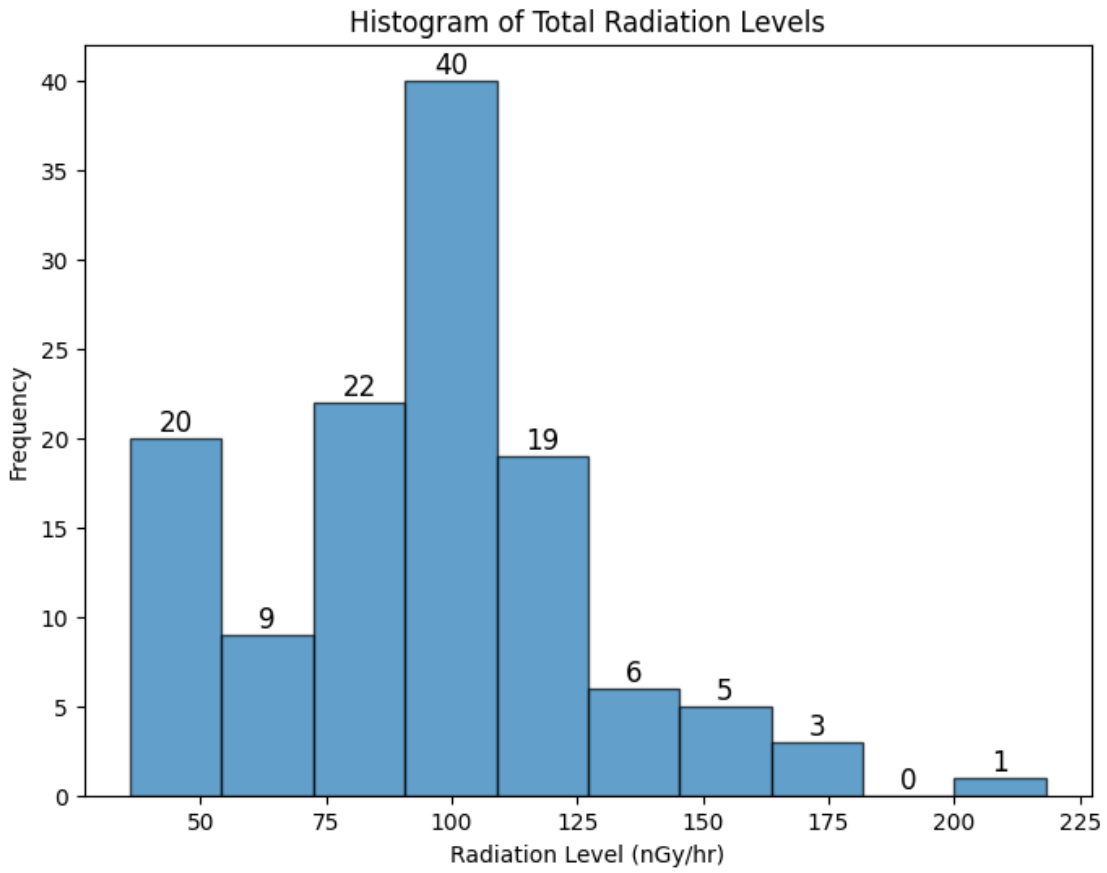


[33]:
```
bin_width = bin_edges[1] - bin_edges[0]
print(bin_width)
```

```
18.22
```

[34]:
```
# histogram
fig, ax = plt.subplots(figsize=(8, 6))
counts, bin_edges, patches = ax.hist(df["Total"], bins=10, edgecolor="black",␣
 ↪alpha=0.7)

# Add labels on top of bars
for count, bin_edge in zip(counts, bin_edges[:-1]):
    ax.text(bin_edge + (bin_edges[1] - bin_edges[0]) / 2, count,␣
 ↪str(int(count)),
            ha='center', va='bottom', fontsize=12)


# Labels and Title
ax.set_xlabel("Radiation Level (nGy/hr)")
ax.set_ylabel("Frequency")
```

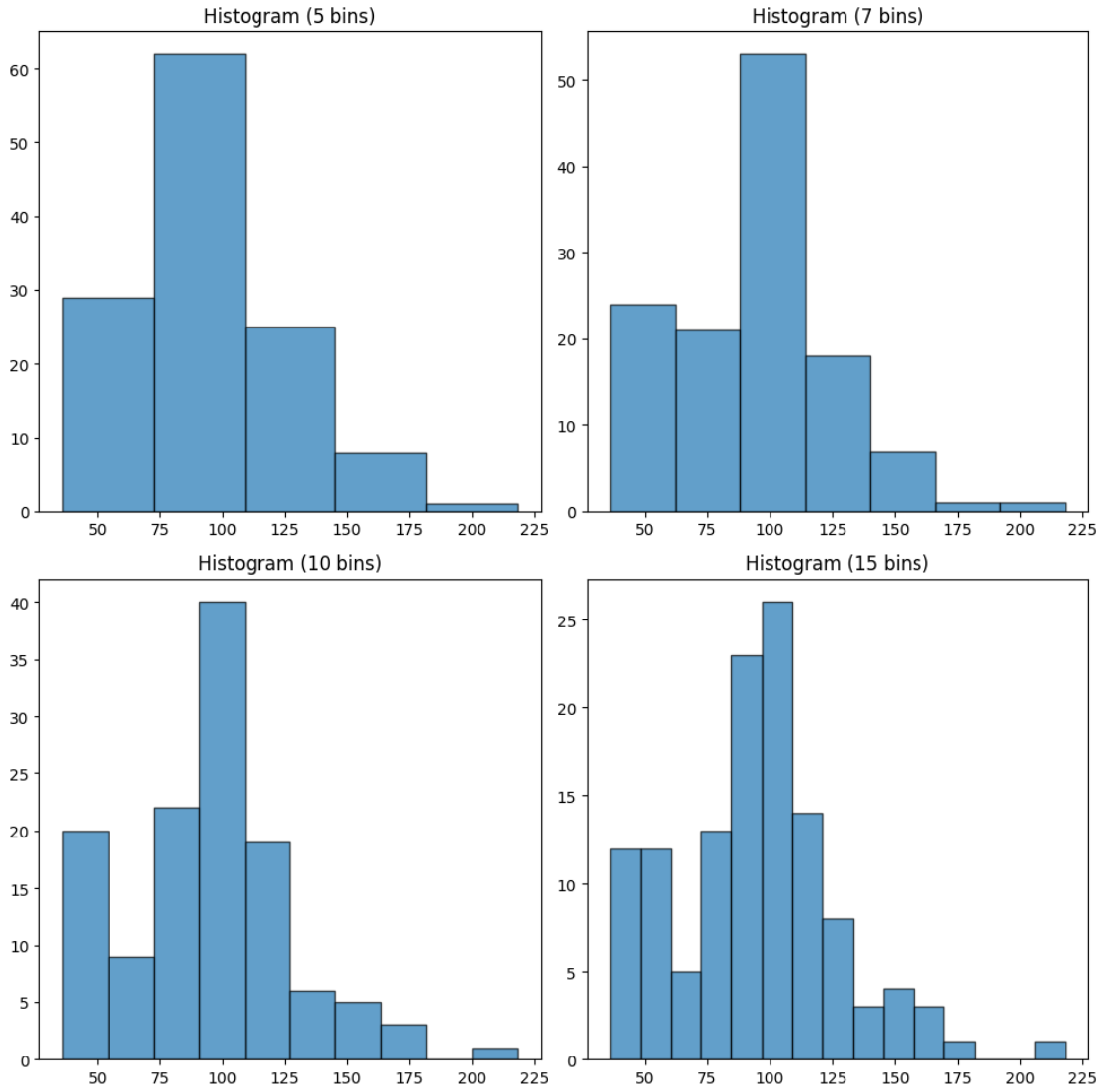
```
ax.set_title("Histogram of Total Radiation Levels")

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



1.5.2 Histograms with different Bin Sizes

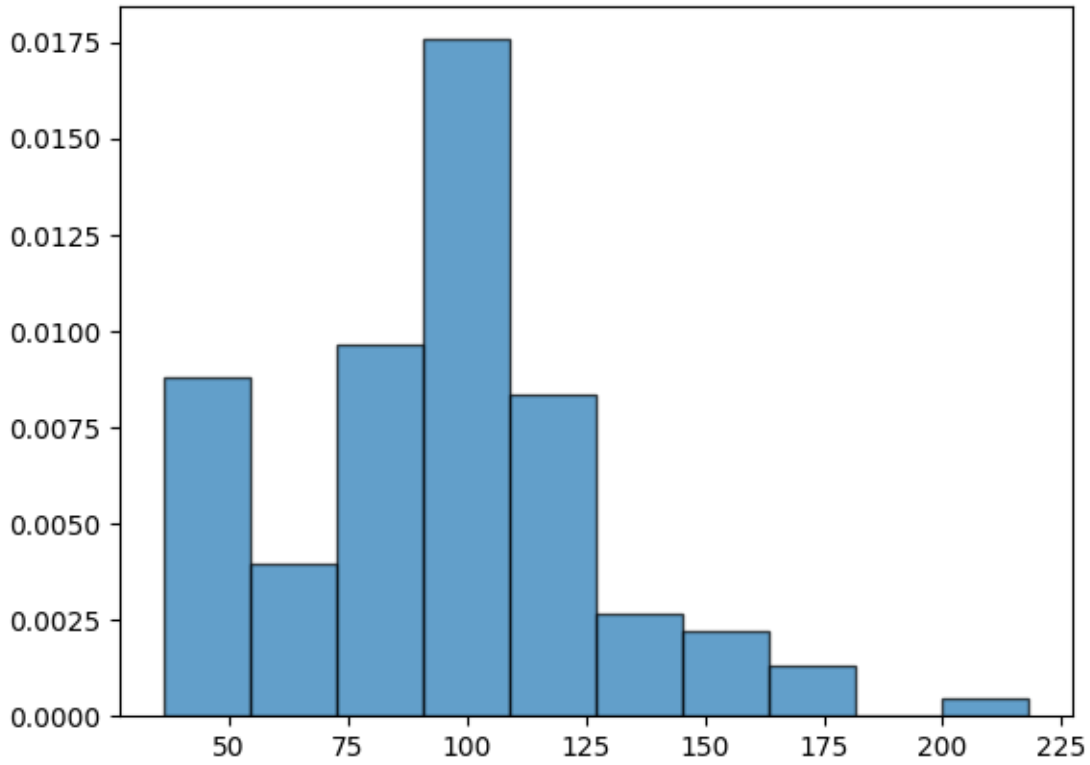
[35]:
```
fig, axes = plt.subplots(2, 2, figsize = (10, 10))

# 5 bins
axes[0, 0].hist(df["Total"], bins = 5, edgecolor = "black", alpha = 0.7)
axes[0, 0].set_title("Histogram (5 bins)")

# 7 bins
axes[0, 1].hist(df["Total"], bins = 7, edgecolor = "black", alpha = 0.7)
axes[0, 1].set_title("Histogram (7 bins)")

# 10 bins
axes[1, 0].hist(df["Total"], bins = 10, edgecolor = "black", alpha = 0.7)
```

```
axes[1, 0].set_title("Histogram (10 bins)")

# 15 bins
axes[1, 1].hist(df["Total"], bins = 15, edgecolor = "black", alpha = 0.7)
axes[1, 1].set_title("Histogram (15 bins)")

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



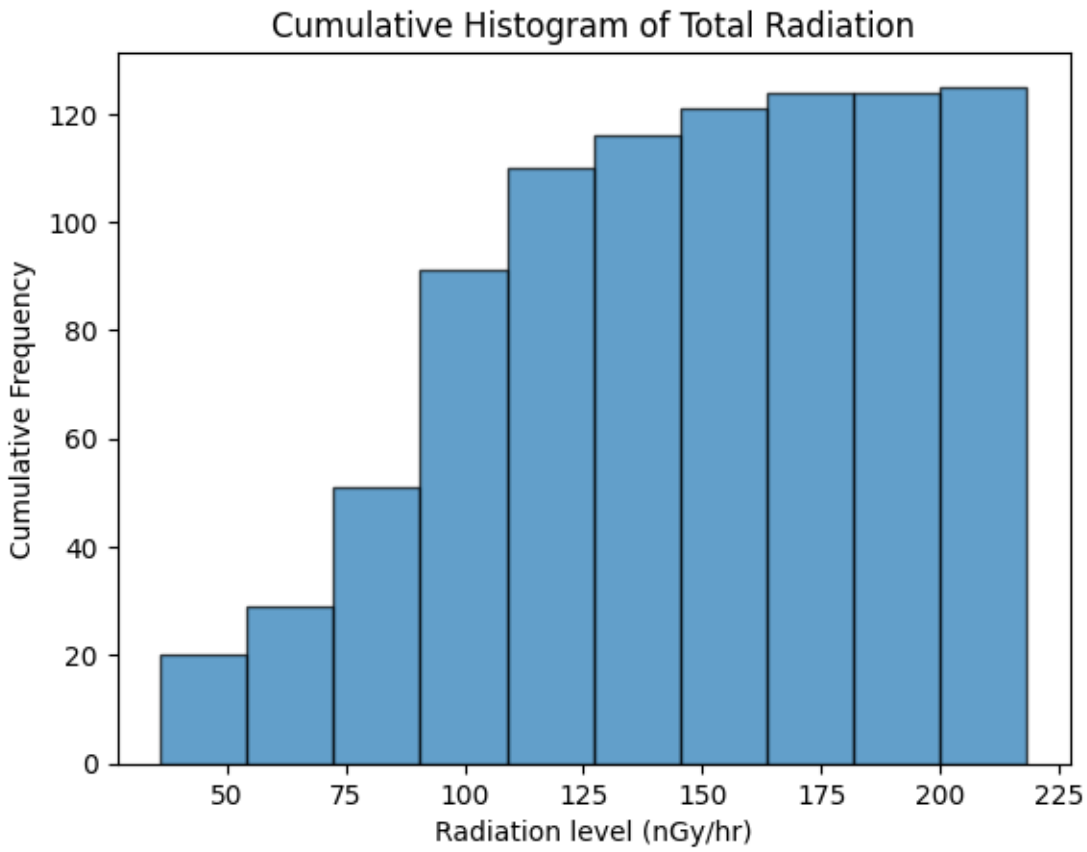
1.5.3 Normalized Histogram (Percentage)

[36]:
```
plt.hist(df["Total"], bins = 10, density = True, edgecolor = "black", alpha = 0.
↪7)

plt.show()
```



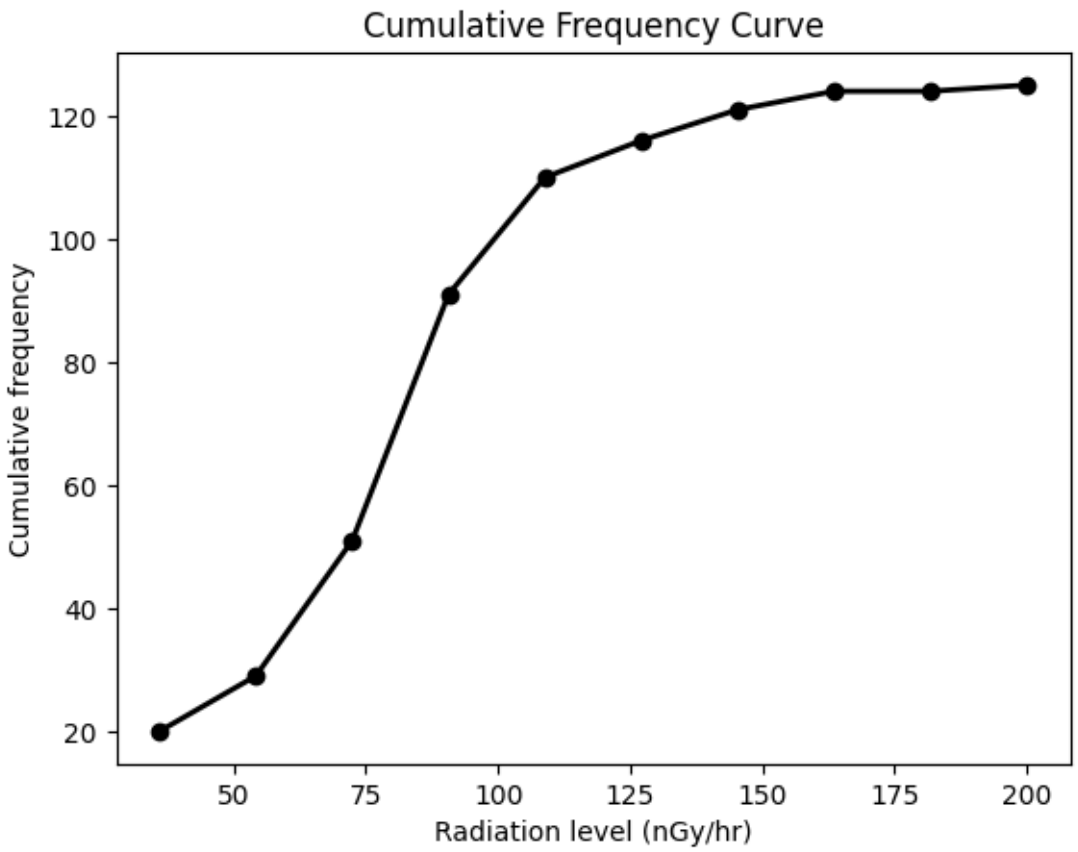
1.5.4 Cumulative Histogram

[37]:
```
plt.hist(df["Total"], bins = 10, cumulative = True, edgecolor = "black", alpha␣
↪= 0.7)

plt.xlabel("Radiation level (nGy/hr)")
plt.ylabel("Cumulative Frequency")
plt.title("Cumulative Histogram of Total Radiation")

plt.show()
```



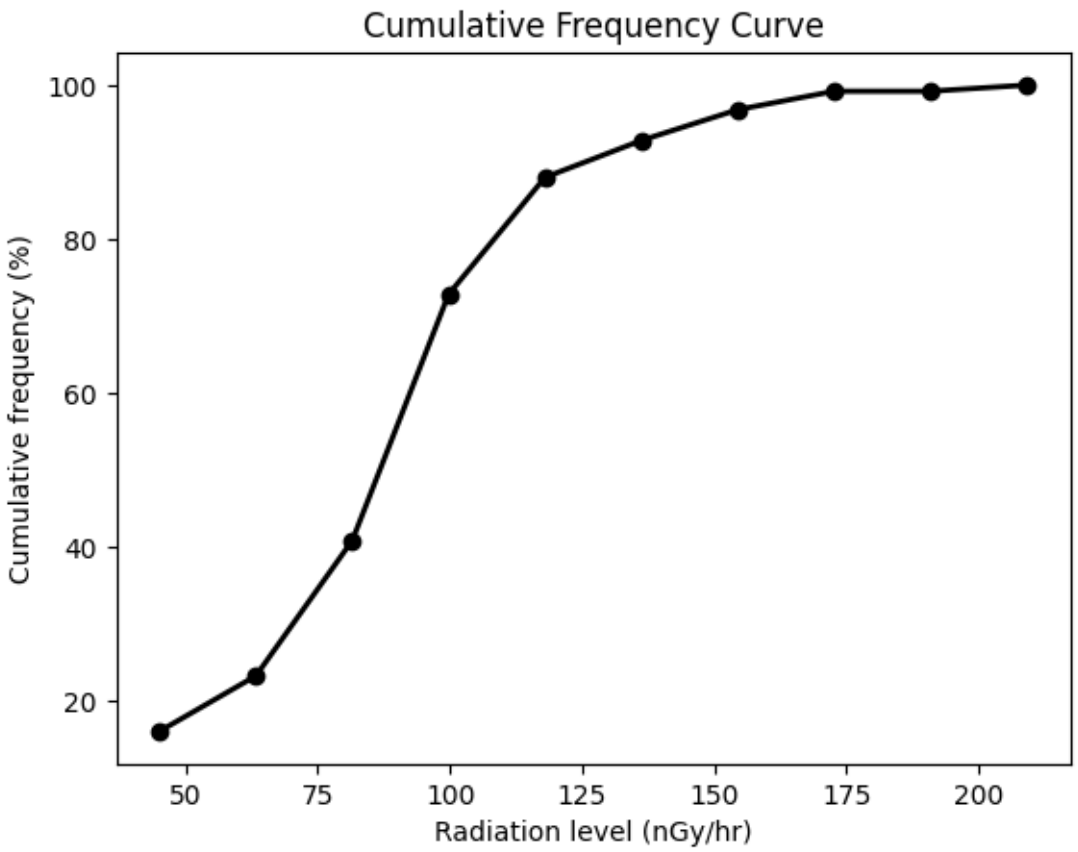
The cumulative histogram looks quite odd. It could be better represented with the cumulative frequency curve.

[38]:
```
# Calculate cumulative frequency using numpy
counts, bin_edges = (np.histogram(df["Total"], bins = 10))
cumulative_counts = np.cumsum(counts)

# Plot cumulative frequency curve using matplotlib
plt.plot(bin_edges[:-1], cumulative_counts, linewidth = 2, linestyle = '-',␣
 ↪color = "black", marker = 'o')
plt.xlabel("Radiation level (nGy/hr)")
plt.ylabel("Cumulative frequency")
plt.title("Cumulative Frequency Curve")

plt.show()
```



Cumulative frequency percentage curve

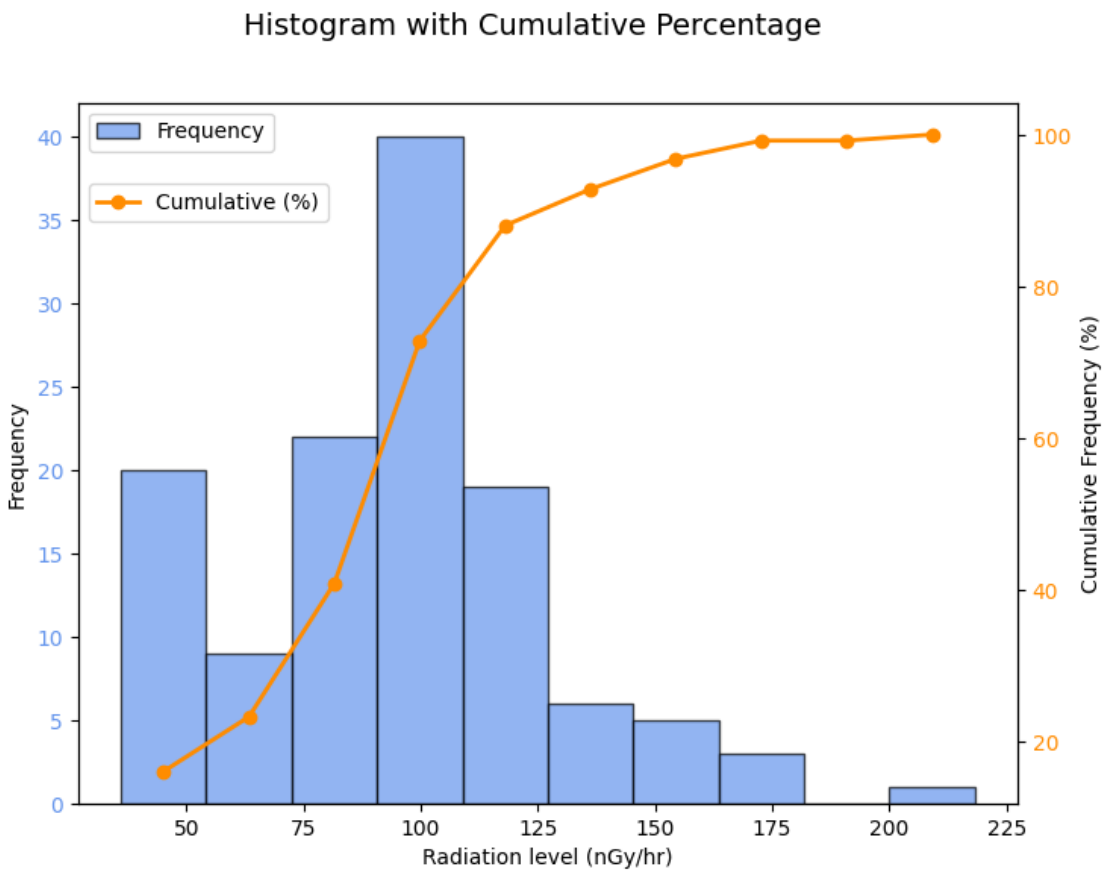
[39]:
```
# Calculate cumulative frequency using numpy
counts, bin_edges = (np.histogram(df["Total"], bins = 10))

cumulative_counts = np.cumsum(counts)
cumulative_percentage = (cumulative_counts / cumulative_counts[-1]) * 100

bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Plot cumulative frequency curve using matplotlib
plt.plot(bin_centers, cumulative_percentage, linewidth = 2, linestyle = '-',␣
 ↪color = "black", marker = 'o')
plt.xlabel("Radiation level (nGy/hr)")
plt.ylabel("Cumulative frequency (%)")
plt.title("Cumulative Frequency Curve")

plt.show()
```



Basic Histogram and Cumulative Percentage Curve Combined in the Same Plot

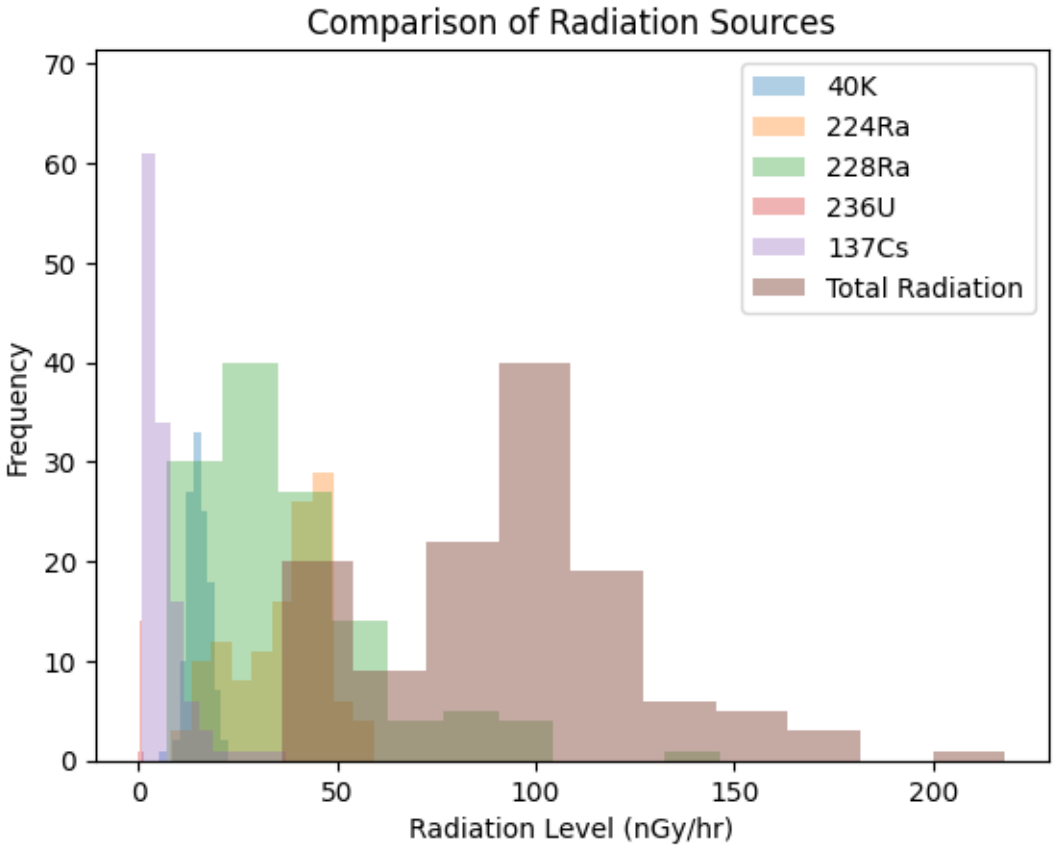
[40]:
```
# Calculate cumulative frequency using numpy
counts, bin_edges = (np.histogram(df["Total"], bins = 10))

# Calculate bin width for plotting histogram (bar plot)
bin_width = (bin_edges[1] - bin_edges[0])

# Calculate cumulative frequency and cumulative frequency percentage
cumulative_counts = np.cumsum(counts)
cumulative_percentage = (cumulative_counts / cumulative_counts[-1]) * 100

# Calculate bin centers for plotting cumulative frequency curve
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Create a dual-axis figure
fig, ax1 = plt.subplots(figsize = (8, 6))
ax2 = ax1.twinx()

# Plot histogram (frequency/count) on the Left axis
```

```
ax1.bar(bin_centers, counts, bin_width, alpha = 0.7, color = "cornflowerblue",␣
 ↪edgecolor = "black", label = "Frequency")

# Set x-label and y-label
ax1.set_xlabel("Radiation level (nGy/hr)")
ax1.set_ylabel("Frequency")
ax1.tick_params(axis = "y", labelcolor = "cornflowerblue")

# Plot cumulative frequency curve on the Right axis
ax2.plot(bin_centers, cumulative_percentage, marker = "o", linestyle = "-",␣
 ↪linewidth = 2, color = "darkorange", label = "Cumulative (%)")
ax2.set_ylabel("Cumulative Frequency (%)")
ax2.tick_params(axis = "y", labelcolor = "darkorange")

# Title and Legend
fig.suptitle("Histogram with Cumulative Percentage", fontsize=14)
ax1.legend(loc='upper left')  # Move frequency label to upper left
ax2.legend(loc='upper left', bbox_to_anchor=(0, 0.9))  # Move cumulative␣
 ↪percentage below frequency in left corner

# Show plot
plt.show()
```



1.5.5 Step 5: Multiple Histograms

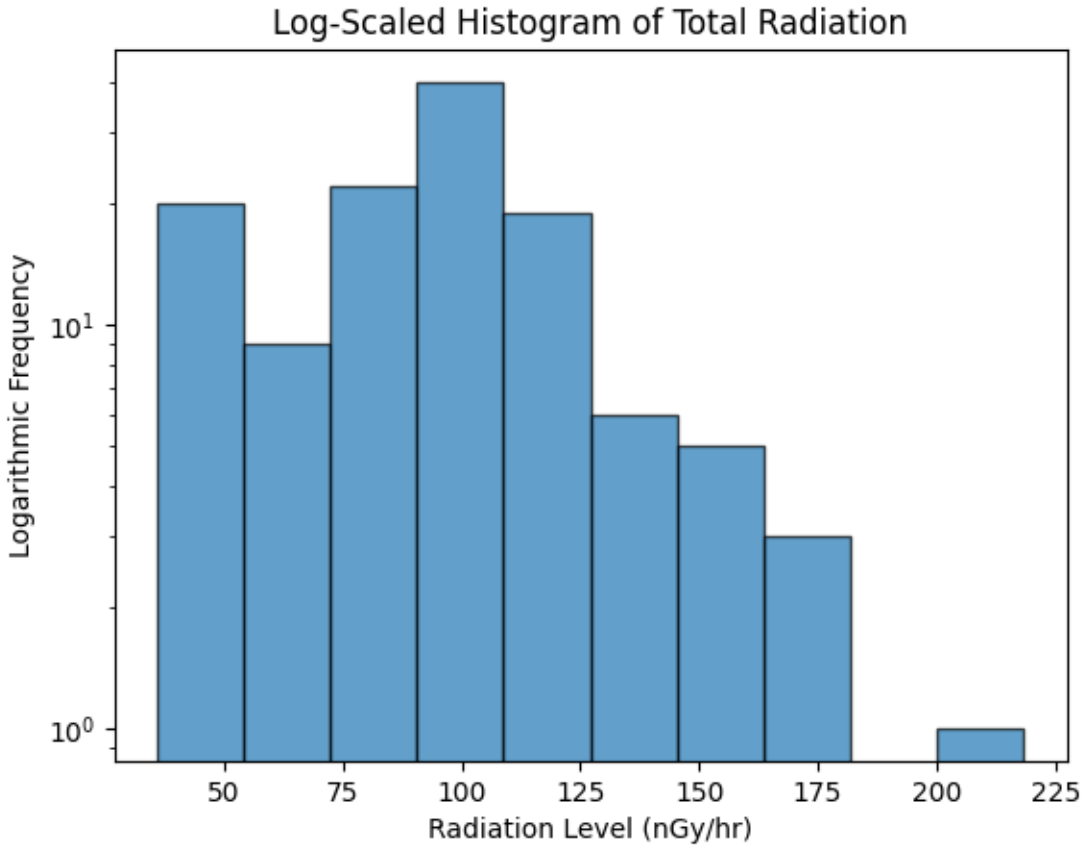
[41]:
```
plt.hist(df["40K"], bins=10, alpha=0.35, label="40K")
plt.hist(df["224Ra"], bins=10, alpha=0.35, label="224Ra")
plt.hist(df["228Ra"], bins=10, alpha=0.35, label="228Ra")
plt.hist(df["236U"], bins=10, alpha=0.35, label="236U")
plt.hist(df["137Cs"], bins=10, alpha=0.35, label="137Cs")
plt.hist(df["Total"], bins=10, alpha=0.5, label="Total Radiation")

plt.xlabel("Radiation Level (nGy/hr)")
plt.ylabel("Frequency")
plt.title("Comparison of Radiation Sources")
plt.legend()

plt.show()
```



1.5.6 Logarithmic Scale Histogram

Just a demo, not meaningful for this data

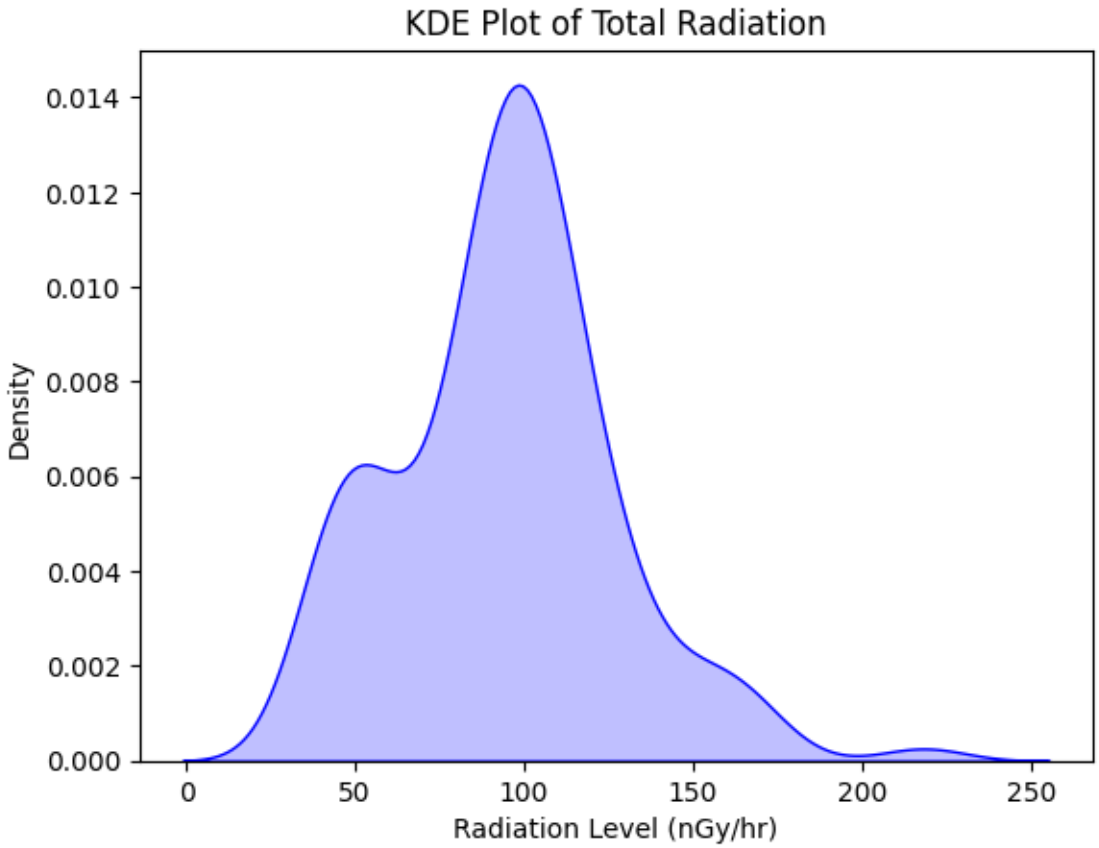
[42]:
```
plt.hist(df["Total"], bins=10, edgecolor="black", alpha=0.7)
plt.yscale("log")  # Set Y-axis to logarithmic scale

plt.xlabel("Radiation Level (nGy/hr)")
plt.ylabel("Logarithmic Frequency")
plt.title("Log-Scaled Histogram of Total Radiation")

plt.show()
```



1.5.7 KDE (Kernel Density Estimation)/Smoothed Histogram Using Seaborn

[43]:
```
sns.kdeplot(df["Total"], fill = True, color = "blue")

plt.xlabel("Radiation Level (nGy/hr)")
plt.ylabel("Density")
plt.title("KDE Plot of Total Radiation")
```

```
plt.show()
```



This is a continuous approximation of the distribution

1.5.8 Which Histogram should you Use?

1. Basic Histogram : General Overview of Distibution
2. Different bin sized : To see how bin size affects interpretation
3. Normalized Histogram : To show percentage instead of counts
4. Cumulative Histogram : Understanding threshold exceedance
5. Multiple Overlaid Histograms : Comparing multiple data
6. Logarithmic Histogram : Data spanning large magnitude differences
7. KDE (Smoothed) : Continuous approximation of distribution

1.5.9 The Importance of Histograms in Data Analysis

A **histogram** is one of the most fundamental tools in data analysis, especially for visualizing distributions. Here's why histograms are so useful:

---

1. Understanding Data Distribution

Histograms show how data values are distributed across different ranges. This helps in identifying:

- **Central Tendency**: The most frequent values in a dataset.
- **Dispersion**: How spread out the values are.
- **Skewness**: Whether the data is symmetrical or skewed to one side.

Example:
In radiation data analysis, histograms help reveal whether most locations have normal radiation levels or if some areas have **outliers** with unusually high exposure.

---

2. Identifying Patterns and Trends

Histograms make it easy to spot patterns in data:

- **Normal Distribution**: A bell-shaped histogram suggests that most values are clustered around the mean.
- **Bimodal or Multimodal Distributions**: Two or more peaks indicate multiple distinct groups within the data.
- **Uniform Distribution**: A flat histogram suggests equal frequency across the range.

Example: If radiation levels in Istria had two peaks (one near 80 nGy/hr and another near 200 nGy/hr), it could indicate two distinct geological sources of radiation.

---

3. Detecting Outliers and Anomalies

Histograms help identify extreme values that **deviate** from the normal distribution.

- Outliers may indicate **errors in data collection** or **unusual environmental conditions**.

Example: If most radiation measurements are **below 150 nGy/hr**, but a few exceed **500 nGy/hr**, those locations may require **further investigation**.

---

4. Comparing Different Data Sets

Histograms allow easy comparison of multiple datasets by overlaying different distributions.

Example: Comparing radiation levels from different regions (e.g., **Istria vs. a nuclear power plant zone**) using histograms can reveal how safe or hazardous each region is.

---

5. Choosing the Right Bin Size Matters

The level of detail in a histogram depends on **bin width**:

- **Too few bins** $\rightarrow$ Oversimplifies the data, hiding important details.
- **Too many bins** $\rightarrow$ Creates noise, making interpretation harder.

Example: In the **Istrian Peninsula histograms**, using 5 bins smoothed the distribution, while 15 bins revealed **more granular details**.

---

6. Aids in Probability and Statistical Analysis

- Used to **approximate probability distributions** (e.g., normal, Poisson, or exponential).
- Helps estimate measures like **mean, variance, and standard deviation**.

Example: In radiation safety, understanding the probability of exceeding **safe exposure levels** is crucial. Histograms help estimate these probabilities.

Conclusion: Why Use Histograms?

- **Visualizes Data Easily** → Helps see patterns, trends, and outliers.
- **Simplifies Large Datasets** → Useful when dealing with thousands of data points.
- **Assists in Decision Making** → Identifies safe vs. hazardous conditions.
- **Improves Statistical Analysis** → Supports hypothesis testing and probability estimates.

1.5.10 Disadvantages of Histograms

While histograms are useful for visualizing data distributions, they have some limitations:

1. Bin Size Sensitivity

- The choice of **bin width** significantly affects the shape of the histogram.
- **Too few bins** → Oversimplifies the distribution (hides patterns).
- **Too many bins** → Adds unnecessary noise, making it hard to interpret.

Example: A histogram with **5 bins** may look smooth, while one with **50 bins** may appear jagged.

---

2. Loss of Individual Data Points

- Histograms **group data into bins**, meaning **individual data points are not visible**.
- This makes it **impossible to see exact values** or detect small variations.

Alternative: Use a **scatter plot or KDE (Kernel Density Estimation)** to preserve individual points.

---

3. Difficult to Compare Multiple Distributions

- Comparing multiple histograms requires **overlaying them**, which can lead to **cluttered visuals**.
- **Alternative:** Use a **box plot or violin plot** for better side-by-side comparison.

---

4. Misleading Interpretations

- The histogram's appearance can be **misleading depending on the bin width or starting point**.
- Different bin choices can make the **same dataset appear skewed or normally distributed**.

Example: A **shift in bin edges** can change the **perceived peaks and valleys** in the data.

---

5. Not Ideal for Small Datasets

- Histograms are **less effective with small datasets** because binning may result in **empty or uninformative bins**.
- **Alternative:** A **dot plot or strip plot** is better for small sample sizes.

---

6. Cannot Show Exact Values or Outliers Easily

- Unlike scatter plots or box plots, histograms **do not explicitly show outliers**.
- Extreme values may be **hidden** within wide bins.

Solution: Use a **box plot** to visualize outliers more clearly.

When to Use and Avoid Histograms

Use Histogram When...	Avoid Histogram When...
You need to visualize **frequency distributions**.	You need to see **individual data points**.
Your dataset is **large enough** to justify binning.	The dataset is **too small** (bins may be empty).
You want to compare **shape, skewness, and spread**.	You need **exact values** or **outlier detection**.