In [1]:
```python
import pandas as pd
import numpy as np

# Import the data
data = "C:/Users/Arun/Downloads/Game.csv"
df = pd.read_csv(data)

df
df.isnull().sum()
df=df.dropna(axis=1)
df
```

Out[1]:

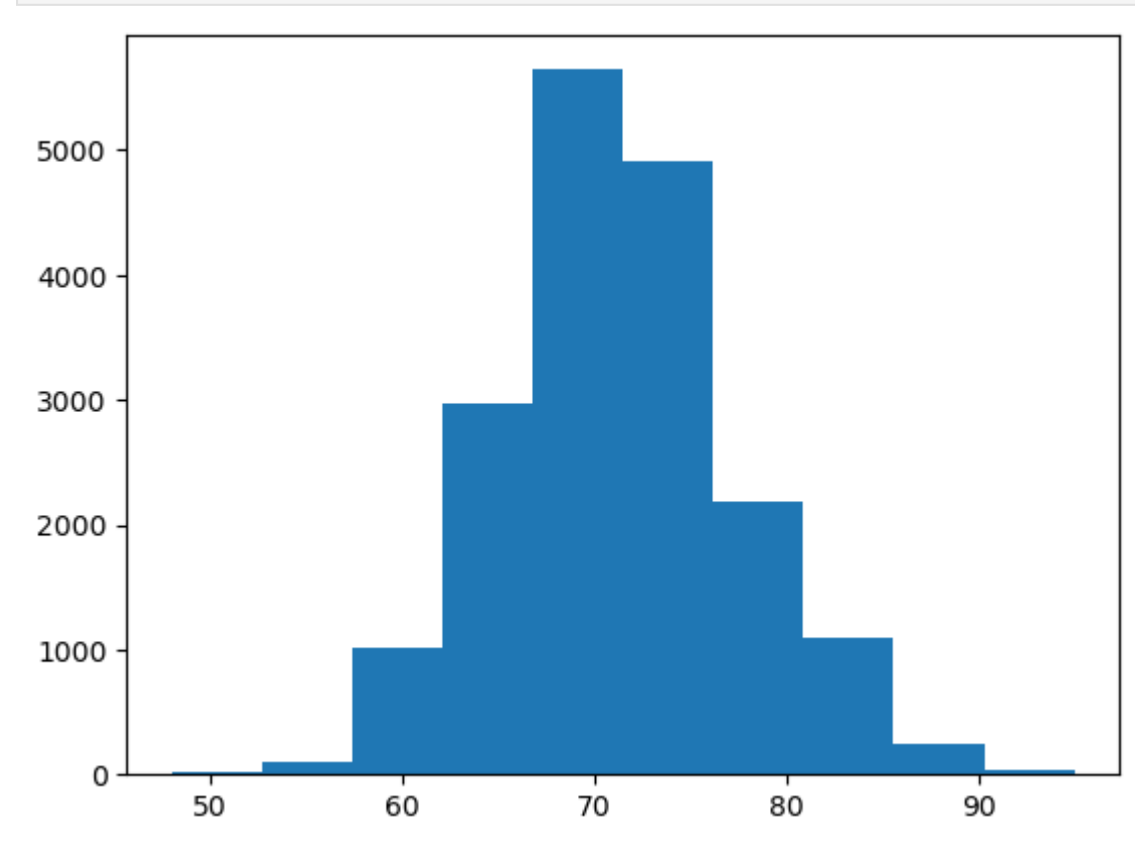
	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club Logo	Value	Wage	Special
0	16	Luis García	37	https://cdn.sofifa.org/players/4/19/16.png	Spain	https://cdn.sofifa.org/flags/45.png	71	71	https://cdn.sofifa.org/teams/2/light/2013.png	€750K	€6K	1906
1	41	Iniesta	34	https://cdn.sofifa.org/players/4/19/41.png	Spain	https://cdn.sofifa.org/flags/45.png	86	86	https://cdn.sofifa.org/teams/2/light/101146.png	€21.5M	€21K	2058
2	80	E. Belözoğlu	37	https://cdn.sofifa.org/players/4/19/80.png	Turkey	https://cdn.sofifa.org/flags/48.png	79	79	https://cdn.sofifa.org/teams/2/light/101014.png	€4M	€23K	2047
3	164	G. Pinzi	37	https://cdn.sofifa.org/players/4/19/164.png	Italy	https://cdn.sofifa.org/flags/27.png	70	70	https://cdn.sofifa.org/teams/2/light/110912.png	€240K	€2K	1882
4	657	D. Vaughan	35	https://cdn.sofifa.org/players/4/19/657.png	Wales	https://cdn.sofifa.org/flags/50.png	66	66	https://cdn.sofifa.org/teams/2/light/1937.png	€150K	€4K	1781
...	...	...	...	...	...	...	...	...	...	...	...	...
18202	246609	J. Requena	19	https://cdn.sofifa.org/players/4/19/246609.png	Argentina	https://cdn.sofifa.org/flags/52.png	57	72	https://cdn.sofifa.org/teams/2/light/110396.png	€220K	€1K	1555
18203	246613	J. Zwarts	19	https://cdn.sofifa.org/players/4/19/246613.png	Netherlands	https://cdn.sofifa.org/flags/34.png	62	77	https://cdn.sofifa.org/teams/2/light/246.png	€650K	€1K	1502
18204	246616	José Uche	18	https://cdn.sofifa.org/players/4/19/246616.png	Spain	https://cdn.sofifa.org/flags/45.png	58	69	https://cdn.sofifa.org/teams/2/light/110839.png	€180K	€1K	1381
18205	246617	Javi Mier	19	https://cdn.sofifa.org/players/4/19/246617.png	Spain	https://cdn.sofifa.org/flags/45.png	62	76	https://cdn.sofifa.org/teams/2/light/110827.png	€650K	€1K	1581
18206	246620	E. McCue	17	https://cdn.sofifa.org/players/4/19/246620.png	Sweden	https://cdn.sofifa.org/flags/46.png	51	74	https://cdn.sofifa.org/teams/2/light/698.png	€70K	€1K	1203

18207 rows × 12 columns

In [20]:
```python
#task1

import re
# Convert the 'Wage' column to a numerical data type (e.g., float)
df['Wage'] = pd.to_numeric(df['Wage'], errors='coerce')

# Drop any rows where the 'Wage' column couldn't be converted to a numeric value
df.dropna(subset=['Wage'], inplace=True)

if df.empty:
    print("DataFrame is empty after dropping NaN rows. Please check your data.")
else:
    Q1 = df['Wage'].quantile(0.25)
    Q3 = df['Wage'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df['Wage'] < lower_bound) | (df['Wage'] > upper_bound)]

    if outliers.empty:
        print("No outliers found in the 'Wage' column.")
    else:
        print("Outliers in the 'Wage' column:")
        print(outliers)
```

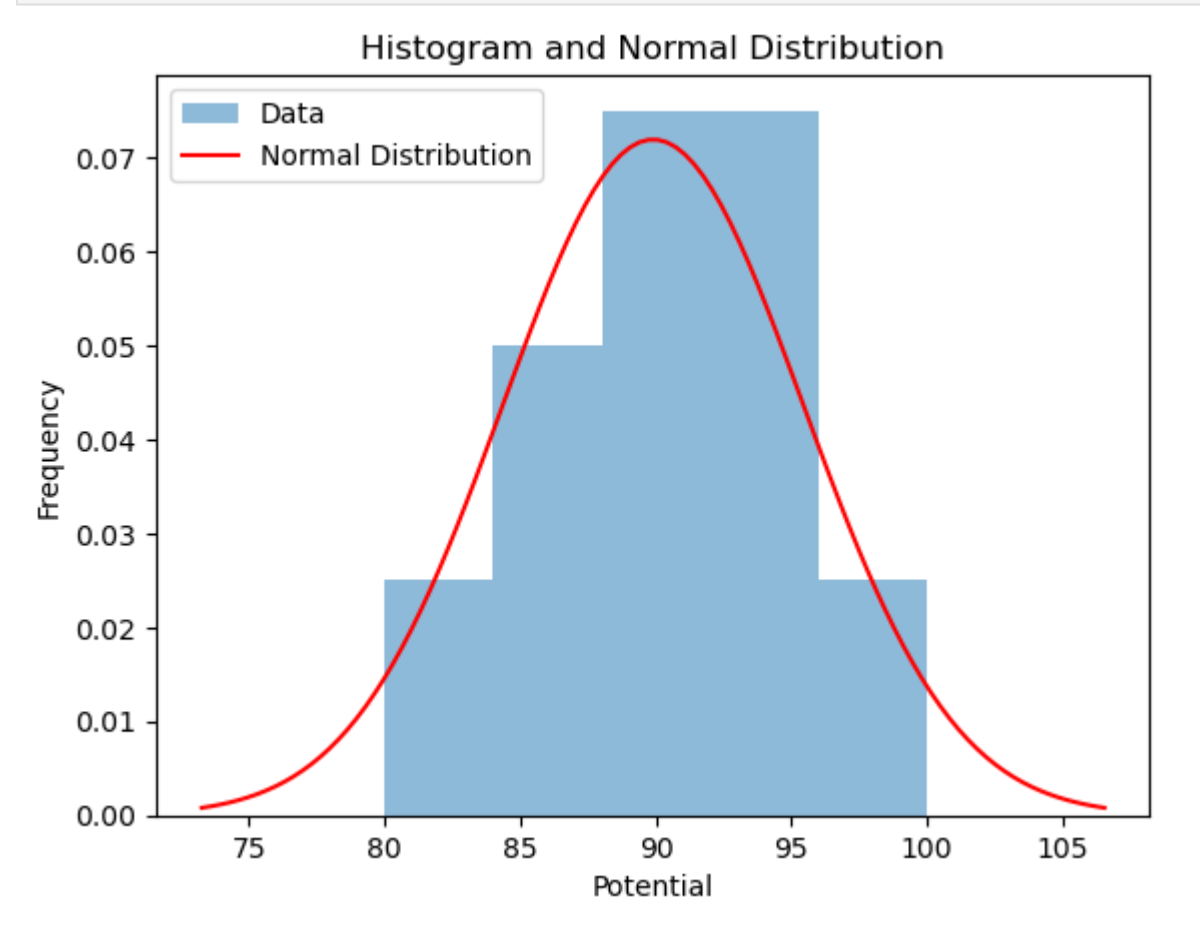
```
No outliers found in the 'Wage' column.
```

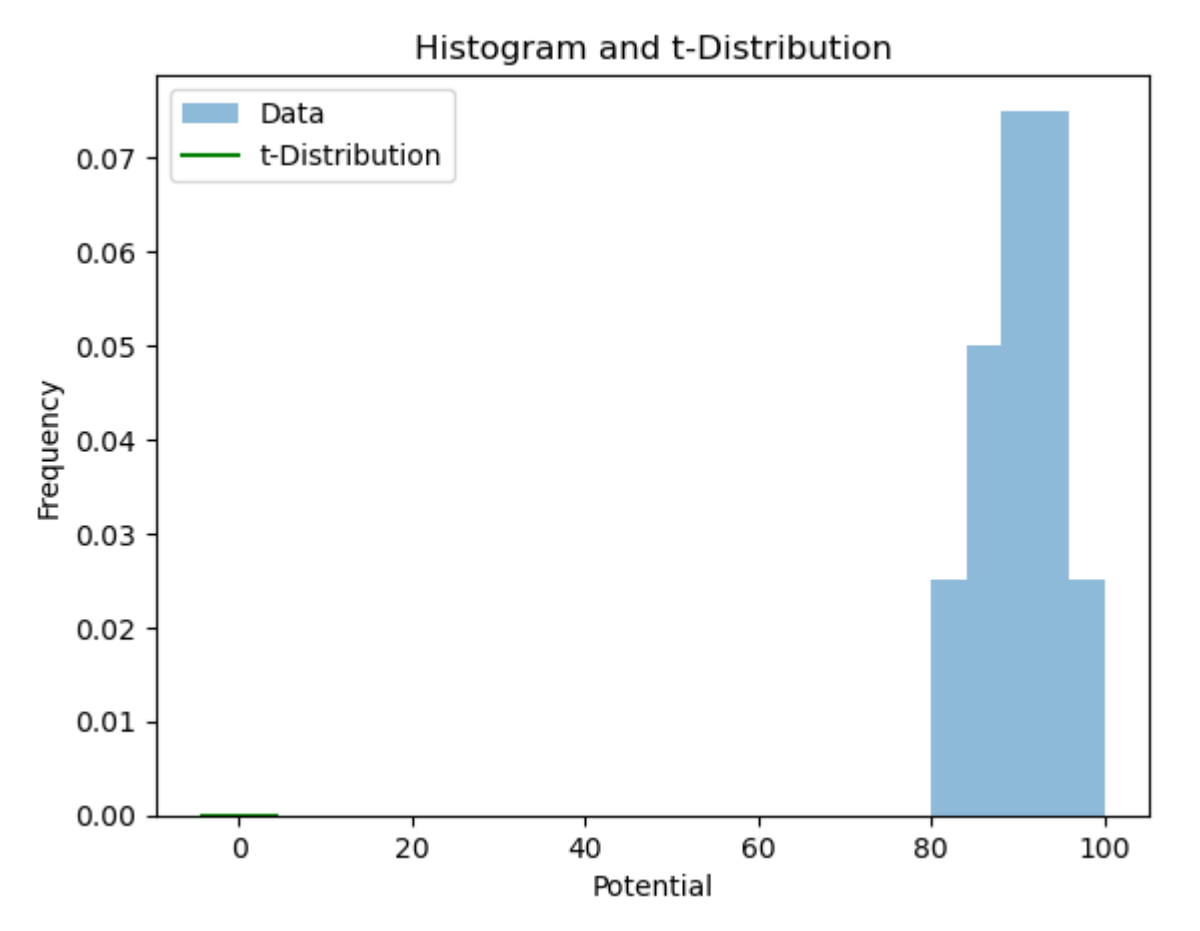
In [4]:
```python
#task2
import matplotlib.pyplot as plt



# Check for any non-numeric values in the 'potential' column. If there are any, replace them with NaN.
df.loc[df['Potential'] == "string", 'Potential'] = np.NAN

# Convert the 'potential' column to a numeric dtype.
df['Potential'] = df['Potential'].astype('float')

# Check if the 'potential' column is a listlike object.
if isinstance(df['Potential'], list):
    # Convert the 'potential' column to a Series object.
    df['Potential'] = df['Potential'].squeeze()

# Plot the distribution of the 'potential' column using a histogram.
plt.hist(df['Potential'])
plt.show()

# Calculate the mean, median, and standard deviation of the 'potential' column.
mean = df['Potential'].mean()
median = df['Potential'].median()
std = df['Potential'].std()

# Analyze the distribution of the 'potential' column and identify any outliers.
print("The mean of the 'Potential' column is:", mean)
print("The median of the 'Potential' column is:", median)
print("The standard deviation of the 'Potential' column is:", std)
```



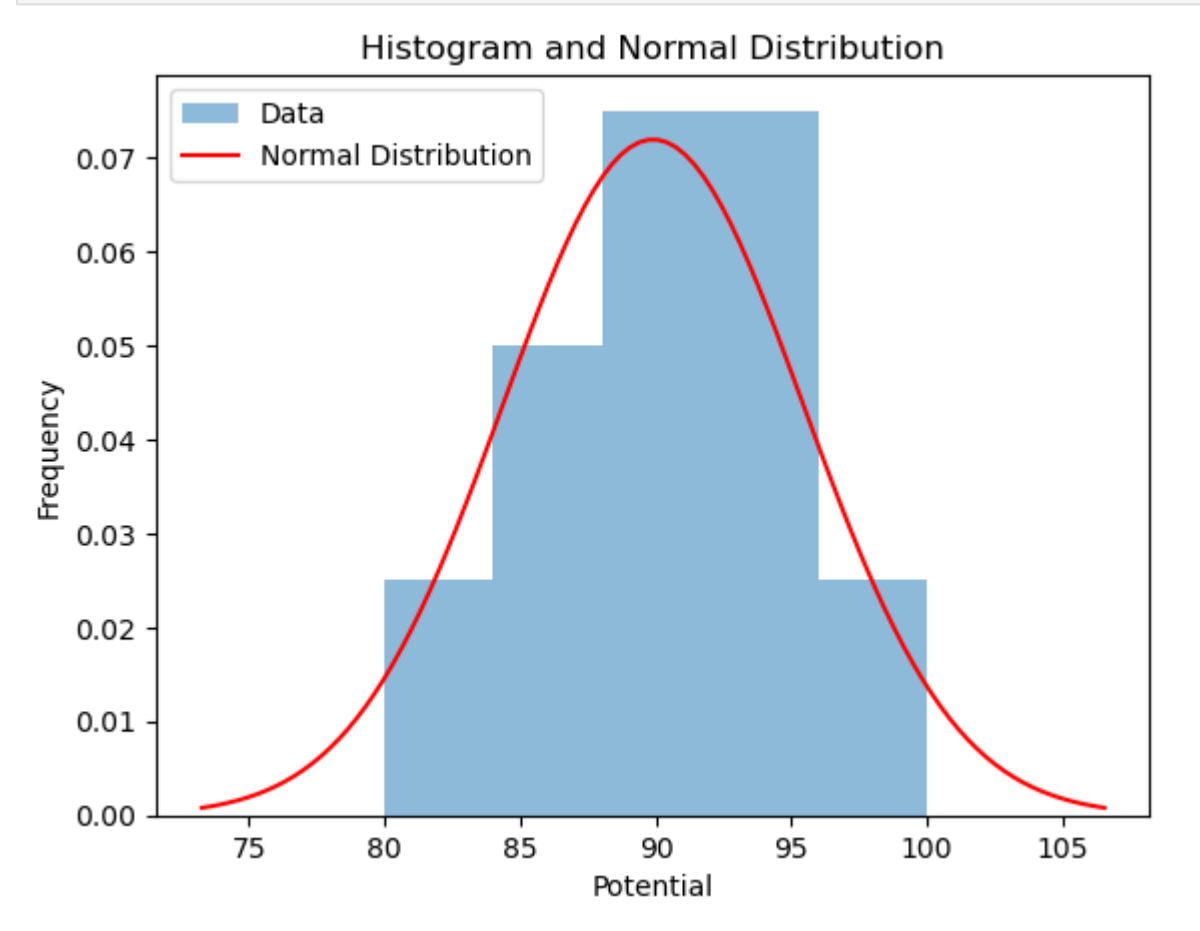
```
The mean of the 'Potential' column is: 71.30729939034437
The median of the 'Potential' column is: 71.0
The standard deviation of the 'Potential' column is: 6.136495583499051
```

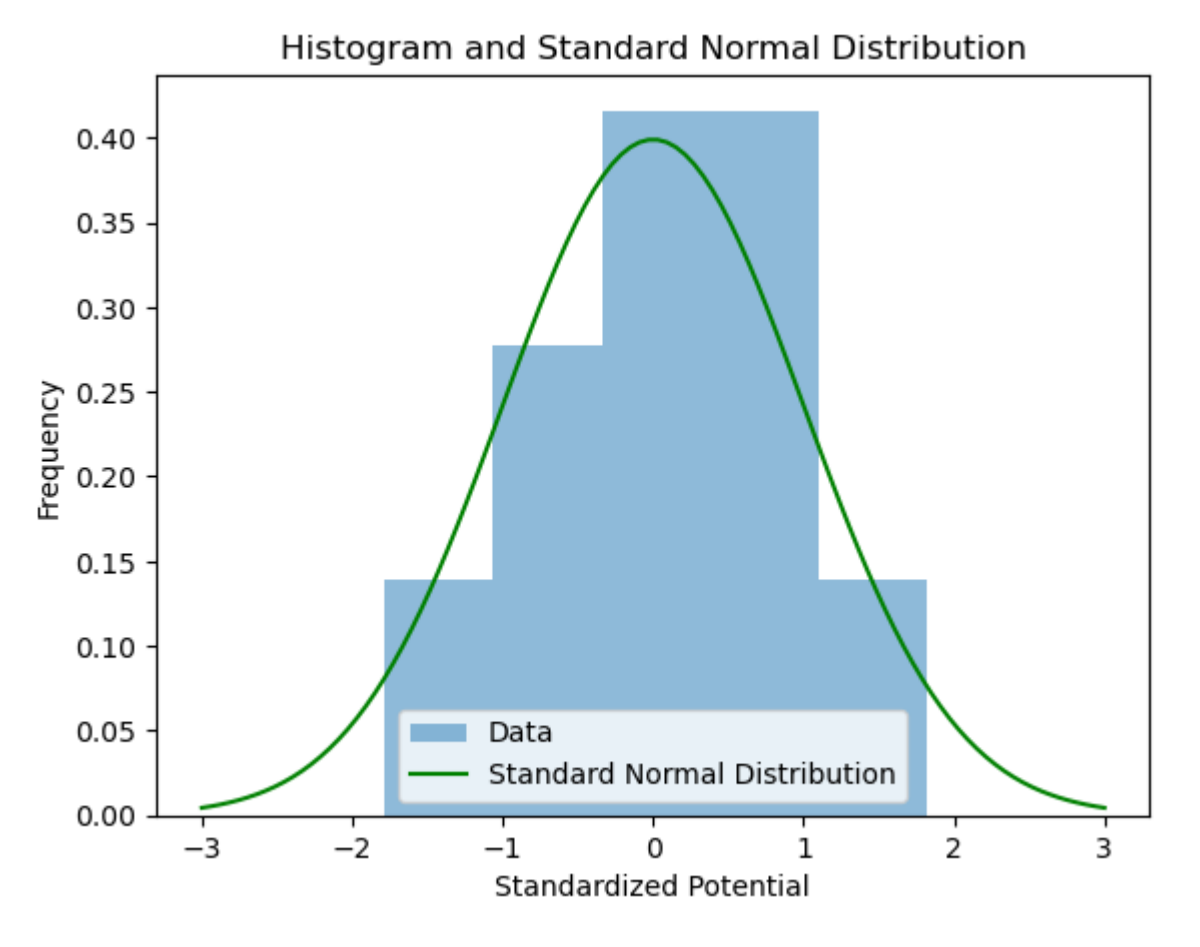
In [6]:
```python
#task3

from scipy.stats import norm, t
#Normal Distribution (Gaussian Distribution):
     #The normal distribution is a symmetric bell-shaped probability distribution. It is fully defined by its mean (μ) and
     #standard deviation (σ). The shape of the normal distribution is determined solely by these two parameters.
#Student's t-Distribution:
    #The t-distribution is also bell-shaped, but it has thicker tails compared to the normal distribution.
    #It is used when the sample size is small, and the population standard deviation is unknown.


# Convert the 'Potential' column to a numerical data type (e.g., float)
df['Potential'] = pd.to_numeric(df['Potential'], errors='coerce')

# Drop any rows where the 'Potential' column couldn't be converted to a numeric value
df.dropna(subset=['Potential'], inplace=True)

# Calculate mean and standard deviation
mean = df['Potential'].mean()
std = df['Potential'].std()

# Generate points for the x-axis to plot the theoretical normal distribution curve
x_norm = np.linspace(mean - 3 * std, mean + 3 * std, 100)

# Generate points for the x-axis to plot the theoretical t-distribution curve
df_t = len(df) - 1  # Degrees of freedom for t-distribution
x_t = np.linspace(t.ppf(0.001, df_t), t.ppf(0.999, df_t), 100)

# Plot the histogram and theoretical normal distribution curve
plt.hist(df['Potential'], bins=5, density=True, alpha=0.5, label='Data')
plt.plot(x_norm, norm.pdf(x_norm, mean, std), 'r', label='Normal Distribution')
plt.xlabel('Potential')
plt.ylabel('Frequency')
plt.title('Histogram and Normal Distribution')
plt.legend()
plt.show()

# Plot the histogram and theoretical t-distribution curve
plt.hist(df['Potential'], bins=5, density=True, alpha=0.5, label='Data')
plt.plot(x_t, t.pdf(x_t, df_t, mean, std), 'g', label='t-Distribution')
plt.xlabel('Potential')
plt.ylabel('Frequency')
plt.title('Histogram and t-Distribution')
plt.legend()
plt.show()
```





In [8]:
```python
#task4

#Normal Distribution:  The normal distribution represents a bell-shaped probability distribution that can take any real value
                       #for the mean (μ) and standard deviation (σ).
#Standard Normal Distribution:The standard normal distribution is a special case of the normal distribution with a mean
                       #of 0 and a standard deviation of 1.

# Convert the 'Potential' column to a numerical data type (e.g., float)
df['Potential'] = pd.to_numeric(df['Potential'], errors='coerce')

# Drop any rows where the 'Potential' column couldn't be converted to a numeric value
df.dropna(subset=['Potential'], inplace=True)

# Calculate mean and standard deviation
mean = df['Potential'].mean()
std = df['Potential'].std()

# Generate points for the x-axis to plot the theoretical normal distribution curve
x_norm = np.linspace(mean - 3 * std, mean + 3 * std, 100)

# Plot the histogram and theoretical normal distribution curve
plt.hist(df['Potential'], bins=5, density=True, alpha=0.5, label='Data')
plt.plot(x_norm, norm.pdf(x_norm, mean, std), 'r', label='Normal Distribution')
plt.xlabel('Potential')
plt.ylabel('Frequency')
plt.title('Histogram and Normal Distribution')
plt.legend()
plt.show()

# Standardize the 'Potential' column to obtain the standard normal distribution
df['Standardized_Potential'] = (df['Potential'] - mean) / std

# Generate points for the x-axis to plot the theoretical standard normal distribution curve
x_standard_norm = np.linspace(-3, 3, 100)

# Plot the histogram and theoretical standard normal distribution curve
plt.hist(df['Standardized_Potential'], bins=5, density=True, alpha=0.5, label='Data')
plt.plot(x_standard_norm, norm.pdf(x_standard_norm, 0, 1), 'g', label='Standard Normal Distribution')
plt.xlabel('Standardized Potential')
plt.ylabel('Frequency')
plt.title('Histogram and Standard Normal Distribution')
plt.legend()
plt.show()
```





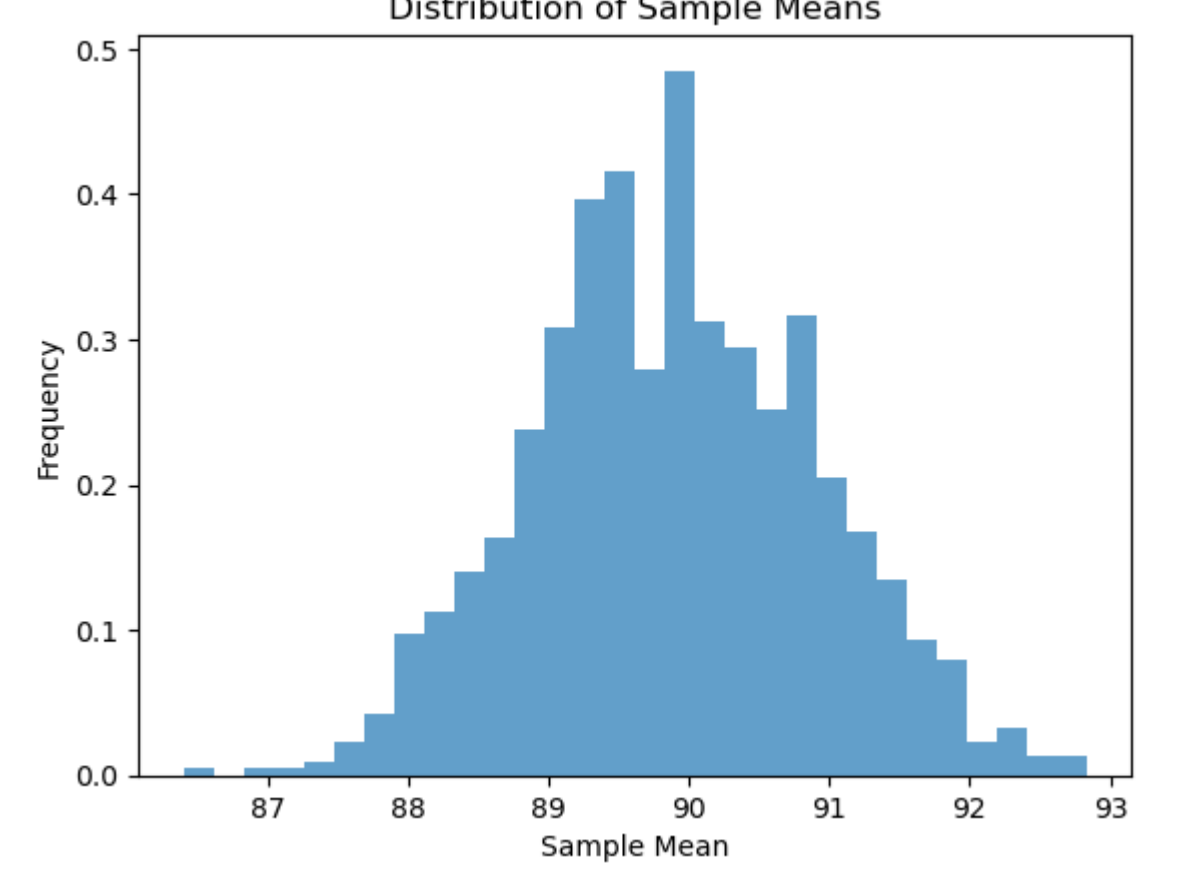
In [13]:
```python
#task5

from scipy.stats import t


# Calculate the sample size
n = len(df)

# Calculate the degrees of freedom (df) for t-distribution
df_t = n - 1

# Define the confidence levels (95%, 90%, and 99%)
confidence_levels = [0.95, 0.90, 0.99]

# Create dictionaries to store the confidence intervals for each column
confidence_intervals = {}

# Calculate the confidence intervals for each column
for col in ['Potential', 'Wage', 'Weight']:
    mean = df[col].mean()
    std_error = df[col].std() / np.sqrt(n)  # Standard error of the mean

    # Calculate the critical t-values for the corresponding confidence levels
    critical_t_values = [t.ppf(1 - (1 - conf_level) / 2, df_t) for conf_level in confidence_levels]

    # Calculate the confidence intervals
    intervals = []
    for t_val in critical_t_values:
        lower_bound = mean - t_val * std_error
        upper_bound = mean + t_val * std_error
        intervals.append((lower_bound, upper_bound))

    # Store the confidence intervals for the column in the dictionary
    confidence_intervals[col] = intervals

# Display the confidence intervals for each column
for col, intervals in confidence_intervals.items():
    print(f"Confidence intervals for '{col}':")
    for i, conf_level in enumerate(confidence_levels):
        print(f"{conf_level * 100}% Confidence Interval: {intervals[i]}")
    print()
```

```
Confidence intervals for 'Potential':
95.0% Confidence Interval: (85.9320791793384, 93.86792082066161)
90.0% Confidence Interval: (86.68464118589925, 93.11535881410076)
99.0% Confidence Interval: (84.19964887886034, 95.60035112113967)

Confidence intervals for 'Wage':
95.0% Confidence Interval: (17007.4025116438, 51912.5974883562)
90.0% Confidence Interval: (20317.489173944232, 48602.51082605577)
99.0% Confidence Interval: (9387.43979630059, 59532.56020369941)

Confidence intervals for 'Weight':
95.0% Confidence Interval: (155.32564518727537, 181.67435481272463)
90.0% Confidence Interval: (157.82431371951574, 179.17568628048426)
99.0% Confidence Interval: (149.57360211525472, 187.42639788474528)
```

In [15]:
```python
#task6

from scipy import stats
# Calculate the sample size
n = len(df)

# Define the confidence levels (95%, 90%, and 99%)
confidence_levels = [0.95, 0.90, 0.99]

# Create dictionaries to store the confidence intervals for each column
confidence_intervals = {}

# Calculate the confidence intervals for each column
for col in ['Potential', 'Wage', 'Weight']:
    mean = df[col].mean()
    std_error = df[col].sem()  # Standard error of the mean

    # Calculate the critical t-values for the corresponding confidence levels
    critical_t_values = [stats.t.ppf(1 - (1 - conf_level) / 2, n - 1) for conf_level in confidence_levels]

    # Calculate the confidence intervals
    intervals = []
    for t_val in critical_t_values:
        lower_bound = mean - t_val * std_error
        upper_bound = mean + t_val * std_error
        intervals.append((lower_bound, upper_bound))

    # Store the confidence intervals for the column in the dictionary
    confidence_intervals[col] = intervals

# Display the confidence intervals for each column
for col, intervals in confidence_intervals.items():
    print(f"Confidence intervals for '{col}':")
    for i, conf_level in enumerate(confidence_levels):
        print(f"{conf_level * 100}% Confidence Interval: {intervals[i]}")
    print()
```

```
Confidence intervals for 'Potential':
95.0% Confidence Interval: (85.9320791793384, 93.86792082066161)
90.0% Confidence Interval: (86.68464118589925, 93.11535881410076)
99.0% Confidence Interval: (84.19964887886034, 95.60035112113967)

Confidence intervals for 'Wage':
95.0% Confidence Interval: (17007.4025116438, 51912.5974883562)
90.0% Confidence Interval: (20317.489173944232, 48602.51082605577)
99.0% Confidence Interval: (9387.43979630059, 59532.56020369941)

Confidence intervals for 'Weight':
95.0% Confidence Interval: (155.32564518727537, 181.67435481272463)
90.0% Confidence Interval: (157.82431371951574, 179.17568628048426)
99.0% Confidence Interval: (149.57360211525472, 187.42639788474528)
```

In [16]:
```python
#task7

import matplotlib.pyplot as plt

# Assuming 'game_data' is your DataFrame with the 'Potential' column

# Select the 'Potential' column from the DataFrame
potential_data = df['Potential']

# Set the sample size and the number of samples for the experiment
sample_size = 30
num_samples = 1000

# Initialize an empty list to store the sample means
sample_means = []

# Perform the sampling experiment
for _ in range(num_samples):
    # Randomly select 'sample_size' elements from the 'Potential' column
    sample = np.random.choice(potential_data, size=sample_size, replace=True)
    # Calculate the sample mean and add it to the list
    sample_means.append(np.mean(sample))

# Plot the histogram of the sample means
plt.hist(sample_means, bins=30, density=True, alpha=0.7)
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.title('Distribution of Sample Means')
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: