Q1. Given a dictionary {'name': 'Tom', 'age': 15, 'city': 'Chicago'}, write a Python program to print all the keys and values

In [44]:
```
user_details = {'name': 'Tom', 'age': 15, 'city': 'Chicago'}

for key ,value in user_details.items():
    print(f"The key {key} has a value {value}")
```

```
The key name has a value Tom
The key age has a value 15
The key city has a value Chicago
```

Q2. Write a Python program to create a Pandas DataFrame and perform 3 data manipulation operations. (Use the dataset that you have downloaded)

In [45]:
```
import pandas as pd

df = pd.read_csv("Final_Exam/death-rate-smoking new.csv")
print("First 5 rows : \n " , df.head(5))
print("----------------- \n")
print("Data types: \n ",df.dtypes)

#Add a new column
df['Result'] = 'Injurious'

# Checking  null in Code column
null_series = pd.isnull(df["Code"])
print(df[null_series])

#Getting only 2015 data
df_2015 = df[df['Year']==2015]

print("Year 2015 data: \n", df_2015.head(10))

## Sorting the DataFrame based on Smoking mortality
df_sorted = df.sort_values(by='Smoking mortality')
print("Sorted by Smoking mortaility: \n",df_sorted)

#Group by country
country_wise = df.groupby('Entity')

for Entity,Year in country_wise:
    print(Entity)
    print('-'*10)
    print(Year)
```

```
764    Guam  GUM  1993       106.784.325  Injurious
992    Guam  GUM  1994       104.232.216  Injurious
1220   Guam  GUM  1995            95.801  Injurious
1448   Guam  GUM  1996         8.962.097  Injurious
1676   Guam  GUM  1997        92.751.205  Injurious
1904   Guam  GUM  1998         8.772.596  Injurious
2132   Guam  GUM  1999        87.986.534  Injurious
2360   Guam  GUM  2000           887.053  Injurious
2588   Guam  GUM  2001         8.891.177  Injurious
2816   Guam  GUM  2002         8.906.343  Injurious
3044   Guam  GUM  2003         8.616.087  Injurious
3272   Guam  GUM  2004        84.587.425  Injurious
3500   Guam  GUM  2005         8.177.444  Injurious
3728   Guam  GUM  2006         8.140.138  Injurious
3956   Guam  GUM  2007        80.963.745  Injurious
4184   Guam  GUM  2008         8.133.574  Injurious
4412   Guam  GUM  2009         8.233.946  Injurious
4640   Guam  GUM  2010         8.234.459  Injurious
4868   Guam  GUM  2011        80.943.504  Injurious
5096   Guam  GUM  2012         8.142.323  Injurious
```

Q3. Explain the purpose of “try”, “except” and “finally”. Write a Python program that copies the contents of one file to another while handling exceptions

Exception Handling:

In Python, There are errors that will occur during the program execution, if the intended code which we write is not handling specific scenario so, to handle different exception we will use

```
1.try
2.except
3.finally
```

Error processing can be done by using the try-except-finally block. Below is the syntax for exception handling using they try-except-finally block

```
try:
    # Code that may raise an exception

except ExceptionType:
    # Code to handle the exception

else:
    # Code to execute if no exception occurred

finally:
    # Code that is always executed, whether an exception occur
red or not
```

In Python, try, except, else and finally are used for exception handling. Here's what each of them does:

```
try:     It is used to wrap the block of code that might raise
an exception. If an exception occurs within
         this block, Python looks for an except block that mat
ches the type of the exception.

except:  This block is executed if an exception of the specifi
ed type (or any of its base types) occurs
          in the preceding try block. It allows you to handle
the exception gracefully by
          providing an alternative course of action.

else:    This block will run if there is no exception occured
in the try block.

finally: This block, if present, is executed regardless of whe
ther an exception occurred or not.
         It's typically used for cleanup operations, such as c
losing files or releasing resources,
         that must be performed under all circumstances.
```

Exception Raising: Python allows you to manually raise exceptions using the raise statement. You can also create custom exception classes to handle specific scenarios. Syntax: raise ExceptionType("Error message")

In [46]:
```
input_file = 'Final_Exam/input_file.txt'
output_file ='Final_Exam/output_file.txt'

try:
    with open (input_file, 'r') as source_file , open(output_file,'w')
        for line in source_file:
            destination_file.write(line.upper())
except FileNotFoundError:
        print("File not found.")
except Exception as e:
        print("An error occurred:", e)
finally:
        print("File copying process completed.")
```

```
File copying process completed.
```

Q4. Write a Python code to demonstrate the use of Strings and any 3 string functions.

In [48]:
```
# In Python, strings are used as basic blocks where we can manipulate
#string input to get the desired output by using the inbuilt string fu

# Simple string
input_string = "Learn, explore, share"

# Using upper() function to convert the string to uppercase
uppercased_string = input_string.upper()
print("Uppercased string:", uppercased_string)

# Using split() function to split the string into a list of words
words_list = input_string.split()
print("List of words:", words_list)

# Using join() function to join the list of words into a single string
joined_string = ' '.join(words_list)
print("Joined string:", joined_string)

# Getting the list of letter using the start,end, increement

char_by_index = input_string[1:16:2]
print("char by indexes:", char_by_index)
```

```
Uppercased string: LEARN, EXPLORE, SHARE
List of words: ['Learn,', 'explore,', 'share']
Joined string: Learn, explore, share
char by indexes: er,epoe
```

Q5. Write a Python program using NumPy to create a 3x3 matrix with values ranging from 1 to 9. Print the matrix and calculate its transpose.

In [49]:
```
import numpy as np

array_3 = np.arange(1,10)
reshaped_matrix = array_3.reshape(3,3)
print(f"Original matrix before transpose : \n {reshaped_matrix}")

transposed_matrix = np.transpose(reshaped_matrix)
print(f"After transpose: \n {transposed_matrix}")
```

```
Original matrix before transpose :
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
After transpose:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Q6 a. Discuss the importance of data visualization. Use Matplotlib to plot a simple line graph and a pie chart. (Use the dataset that you have downloaded) b. Explain the purpose of Seaborn in data visualization. Create a histogram using Seaborn library

Importance of Data visualization

Data visualisation plays an important role in data science, as this will give us a visual representation of the transformed data after applying programming logic. The end users, who don't know anything about the Data wrangling and transformations can easily visualise how the data is getting changed and how it can be used in future insights based on the old-year data analysis, which is shown in pictorial form.

Visualization provides clear and understandable insights to complex datasets. By representing data visually, patterns, trends, and relationships become more apparent, enabling easier comprehension and interpretation of data.

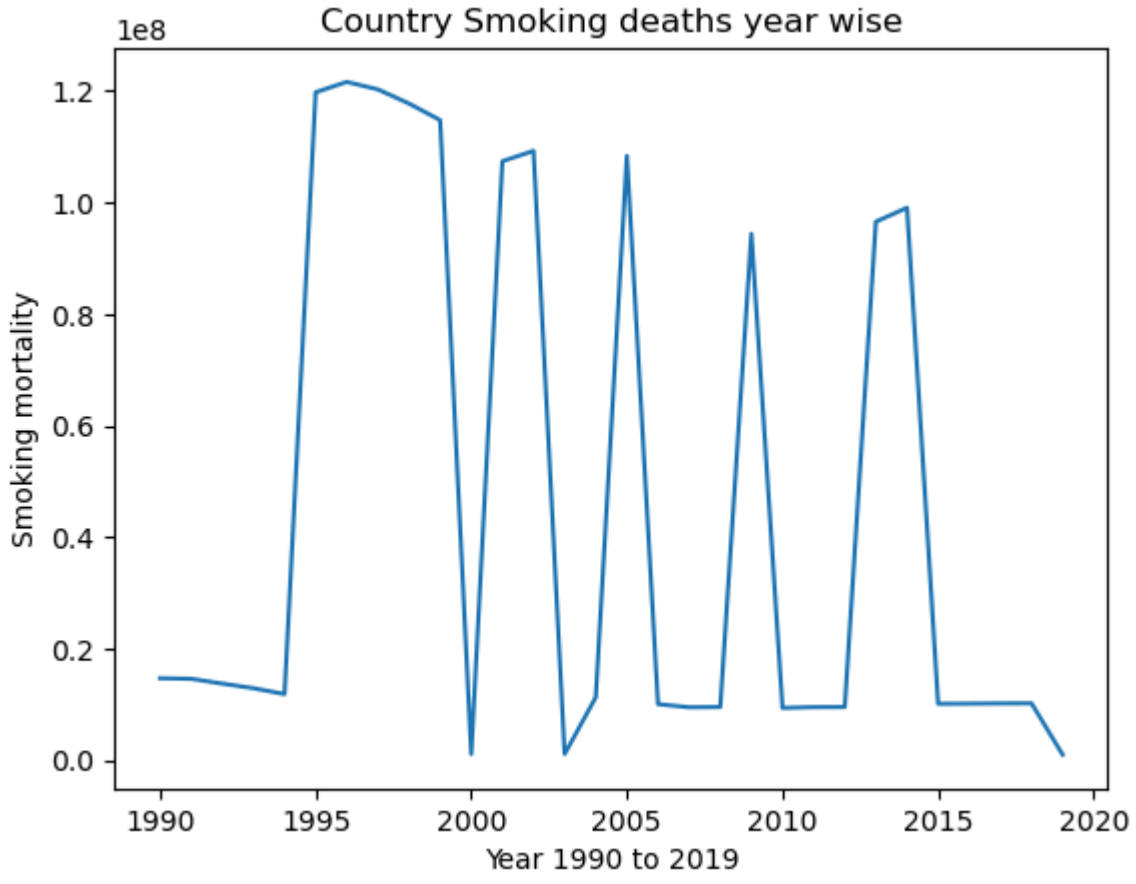
Visualizations helps us to have future ready which is not aviable on the rawdata . Through graphical representation, outliers, correlations, and other significant data points become more apparent, leading to valuable discoveries and informed decision-making.

Provide better communication their to stakeholders who may not have expertise in data analysis. Visualizations enable storytelling with data, making it easier to convey insights, trends, and recommendations to a broader audience.

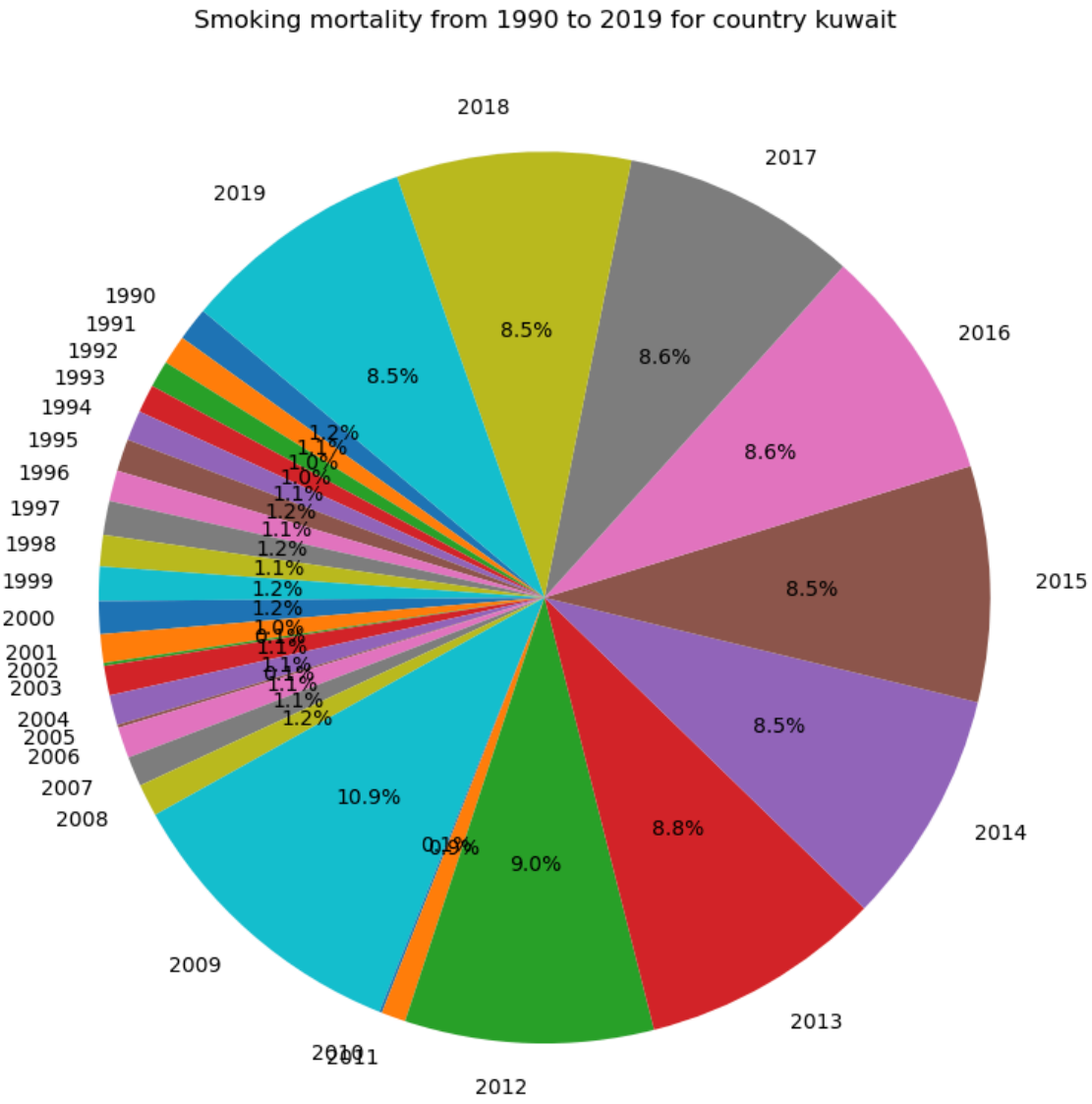
Visualization techniques such as line charts, bar graphs, and scatter plots enable the identification of trends, patterns, and anomalies within the data. These visual cues can help analysts make informed predictions and guide strategic decision-making.

Visualizations empower decision-makers to make more informed and data-driven decisions. By presenting data in a visually appealing and accessible format, decision-makers can quickly grasp the implications of different choices and take appropriate actions.

In [50]:
```
#Use Matplotlib to plot a simple line graph and a pie chart. (Use the

import matplotlib.pyplot as plt

df = pd.read_csv("Final_Exam/death-rate-smoking new.csv")

# Showing first 10 rows
df.head(10)

# Fill nan
df.fillna("Not available")

df['Smoking mortality'] = df['Smoking mortality'].str.replace('.', '')
#Plotting Line graph for country from year 1990
filtered_country_df = df[df['Entity']=='Albania']
plt.plot(filtered_country_df['Year'], filtered_country_df['Smoking mor

plt.xlabel("Year 1990 to 2019")
plt.ylabel("Smoking mortality")
plt.title("Country Smoking deaths year wise")
plt.tight_layout
plt.show()
```



In [51]:
```
# Pie chart
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("Final_Exam/death-rate-smoking new.csv")

# Showing first 10 rows
df.head(10)
df['Smoking mortality'] = df['Smoking mortality'].str.replace('.', '')
filtered_country_df = df[df['Entity']=='Kuwait']
# Plotting big chart
plt.figure(figsize=(10, 8))
plt.pie(filtered_country_df['Smoking mortality'], labels=filtered_cour
plt.title("Smoking mortality from 1990 to 2019 for country kuwait")
plt.tight_layout()

plt.show()
```



b. Explain the purpose of Seaborn in data visualization. Create a histogram using Seaborn library

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is particularly useful for creating complex visualizations with minimal code and offers a wide range of built-in themes and color palettes. Features of Seaborn in data visualization:

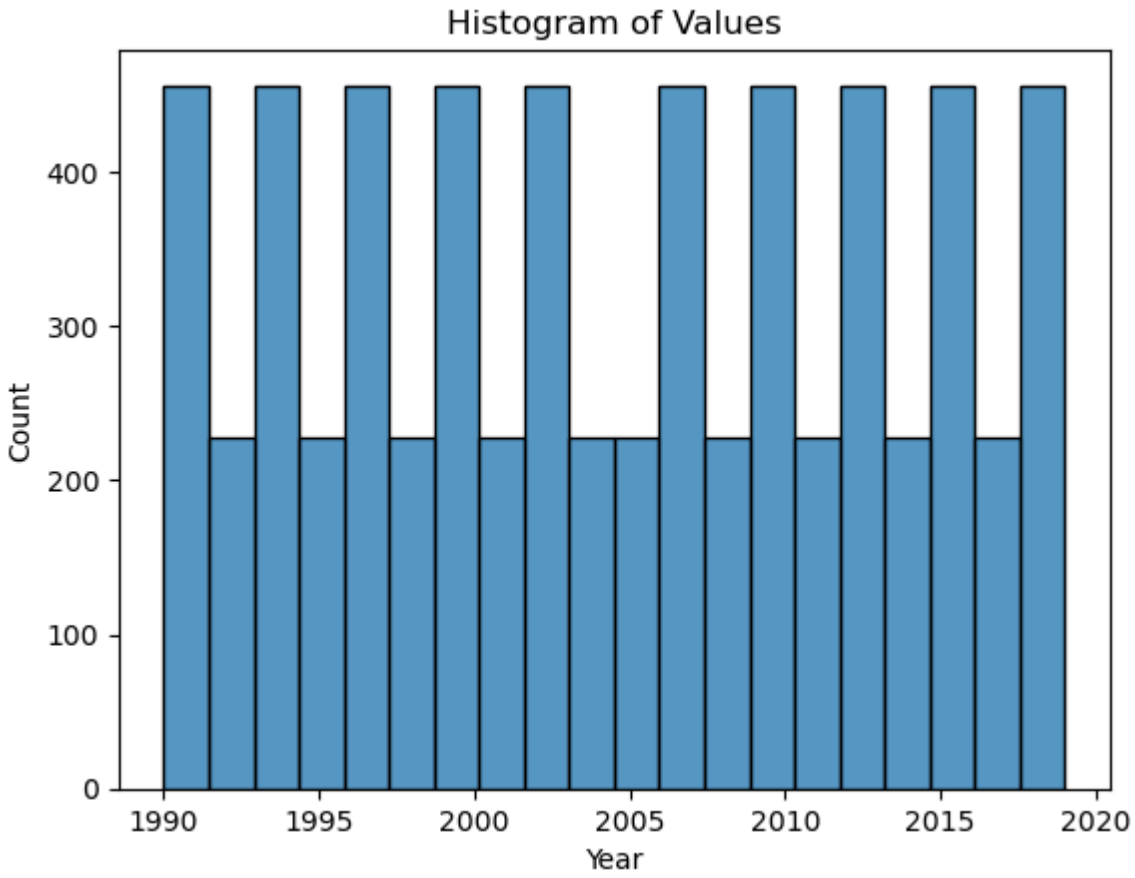
Statistical Visualization: Seaborn simplifies the process of creating statistical visualizations by providing functions specifically tailored to common statistical tasks. It offers functions for visualizing distributions, relationships between variables, categorical data, and more.

Integration with Pandas: Seaborn seamlessly integrates with pandas DataFrames, allowing users to directly pass DataFrame objects to its plotting functions. This integration simplifies the data visualization workflow, as users can easily work with their data without needing to perform extensive data manipulation.

Complex Plot Types: Seaborn supports a wide range of plot types, including but not limited to histograms, scatter plots, line plots, bar plots, violin plots, box plots, and heatmaps. These plot types enable users to explore complex relationships and patterns in their data effectively.

Ease of Use: Seaborn's high-level interface makes it easy to create complex visualizations with minimal code. Many Seaborn functions offer sensible default settings, reducing the need for manual customization. Additionally, Seaborn's functions often accept tidy data formats, aligning with best practices in data analysis.

In [52]:
```
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("Final_Exam/death-rate-smoking new.csv")
# Create a histogram using Seaborn
sns.histplot(data=df, x='Year',bins=20, edgecolor='black')

# Show the plot
plt.title('Histogram of Values')
plt.xlabel('Year')
plt.show()
```



In [ ]:

In [ ]: