CONTENTS

Lab 1: Python Program to Plot Line Graph

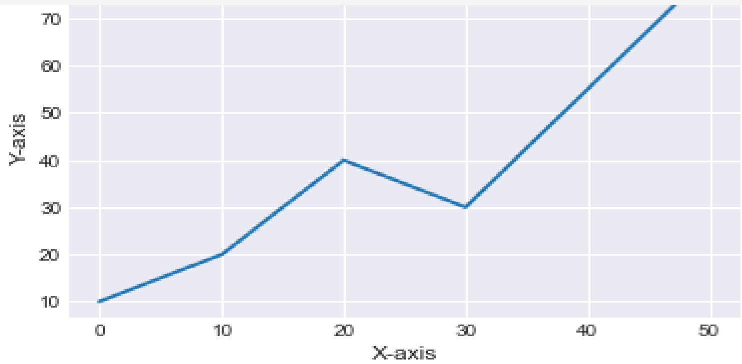
In [6]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Using Matplotlib

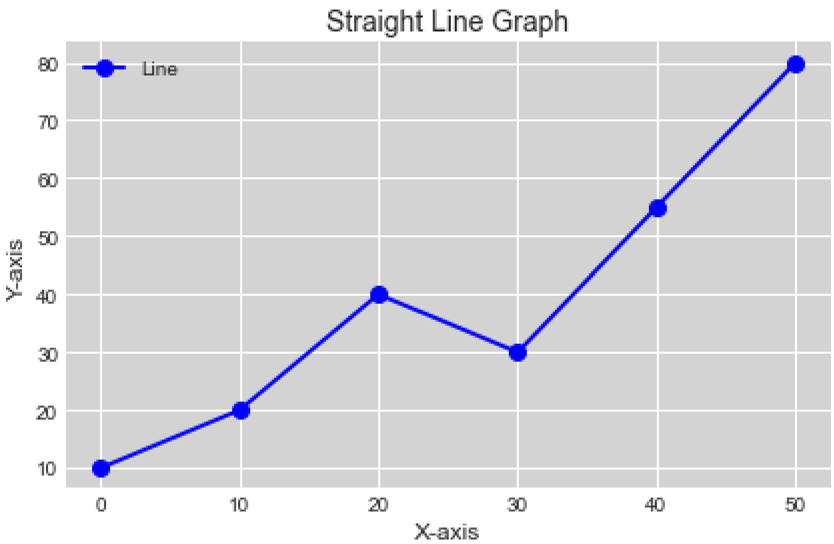
In [18]:
```
# Define X co-ordinates and Y co-ordinates
x = [0, 10, 20, 30, 40, 50]
y = [10, 20, 40, 30, 55, 80]
```

In [19]:
```
# Plot the line graph
plt.plot(x,y)

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Straight Line Graph')

# Display the graph
plt.show()
```

In [20]:
```
# Plot the line graph with customized style
plt.plot(x, y, color='blue', linestyle='-', linewidth=2, marker='o', markersize=8,

# Add labels and title
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis',  fontsize=12)
plt.title('Straight Line Graph', fontsize=14)

# Add grid
plt.grid(True)

# Customize ticks
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
```

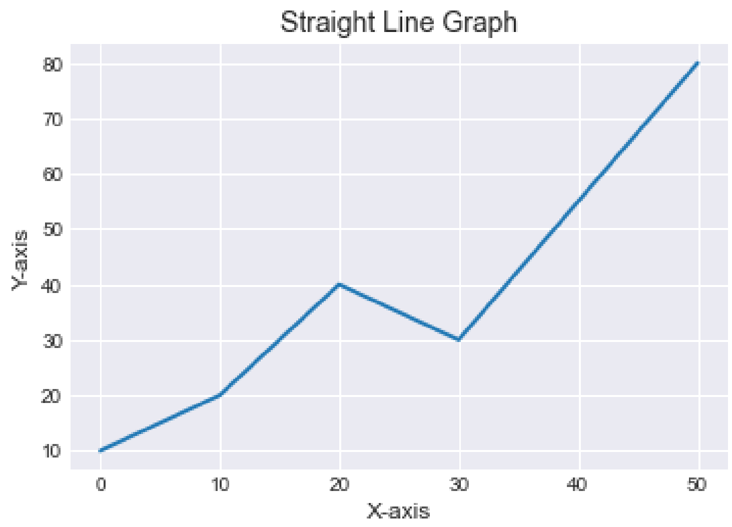


```python
# Add legend
plt.legend(loc='best',  fontsize=10)

# Add background color
plt.gca().set_facecolor('lightgrey')

# Adjust layout
plt.tight_layout()

# Display the graph
plt.show()
```



Using Seaborn

In [21]:
```
# Set seaborn style
sns.set_style('darkgrid')
# Plot the line graph
sns.lineplot(x=x, y=y)
# Add labels and title
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis',  fontsize=12)
plt.title('Straight Line Graph',  fontsize=14)
# Display the graph
plt.show()
```



Lab 2: Python Program to Plot Complete Graph where Number of Vertices are User Input

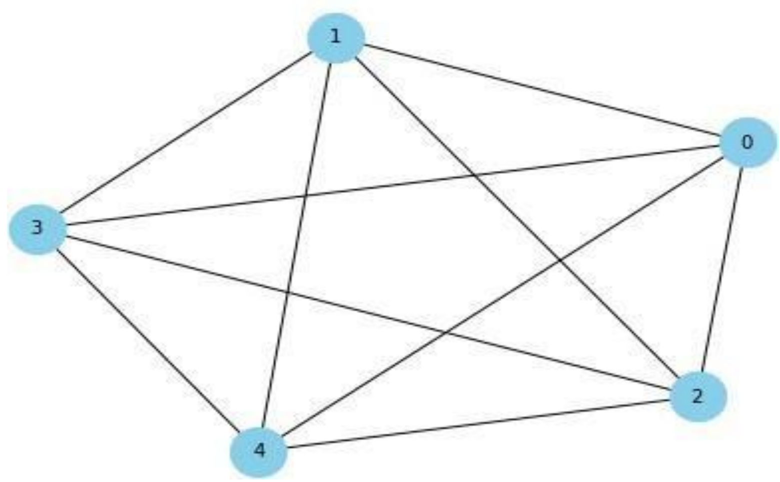
In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
```

In [3]:
```
def plot_complete_graph(num_vertices):
    # Create a complete graph
    G = nx.complete_graph(num_vertices)

    # Draw the graph
    nx.draw(G, with_labels=True, node_color='skyblue', node_size=800, font_size=10

    # Display the plot
    plt.title(f'Complete Graph with {num_vertices} Vertices')
    plt.show()

if __name___== "__main__":
    # Get user input for the number of vertices
    num_vertices = int(input("Enter the number of vertices: "))

    # Plot the complete graph
    plot_complete_graph(num_vertices)
```

Enter the number of vertices: 5

In [4]
```
def plot_complete_graph(num_vertices):
    # Calculate the positions of vertices on a circle
    positions = [(0.5 + 0.5 * np.cos(2 * np.pi * i / num_vertices),
                  0.5 + 0.5 * np.sin(2 * np.pi * i / num_vertices))
                  for i in range(num_vertices)]

    # Plot edges
```

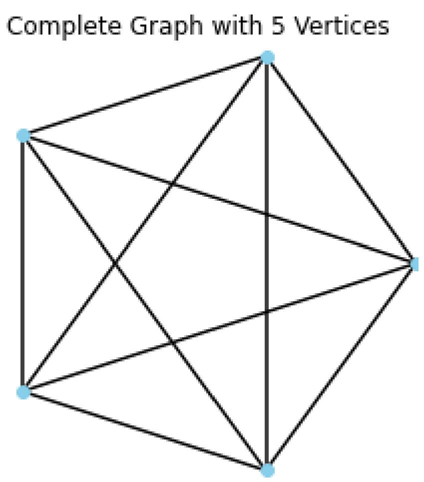
Complete Graph with 5 Vertices



```
    for i in range(num_vertices):
        for j in range(i + 1, num_vertices):
            plt.plot([positions[i][0], positions[j][0]],
                     [positions[i][1], positions[j][1]], 'k-')

    # Plot vertices
    for pos in positions:
        plt.plot(pos[0], pos[1], 'o', color='skyblue')

    # Set plot limits and aspect ratio
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.gca().set_aspect('equal', adjustable='box')

    # Remove axes
    plt.axis('off')

    # Display the plot
    plt.title(f'Complete Graph with {num_vertices} Vertices')
    plt.show()

if __name___== "__main__":
    import numpy as np

    # Get user input for the number of vertices
```

```
Enter the number of vertices: 5
```

In [ ]:



Lab 3: Python Program to demonstrate Graph Coloring where Number of Vertices are User Input

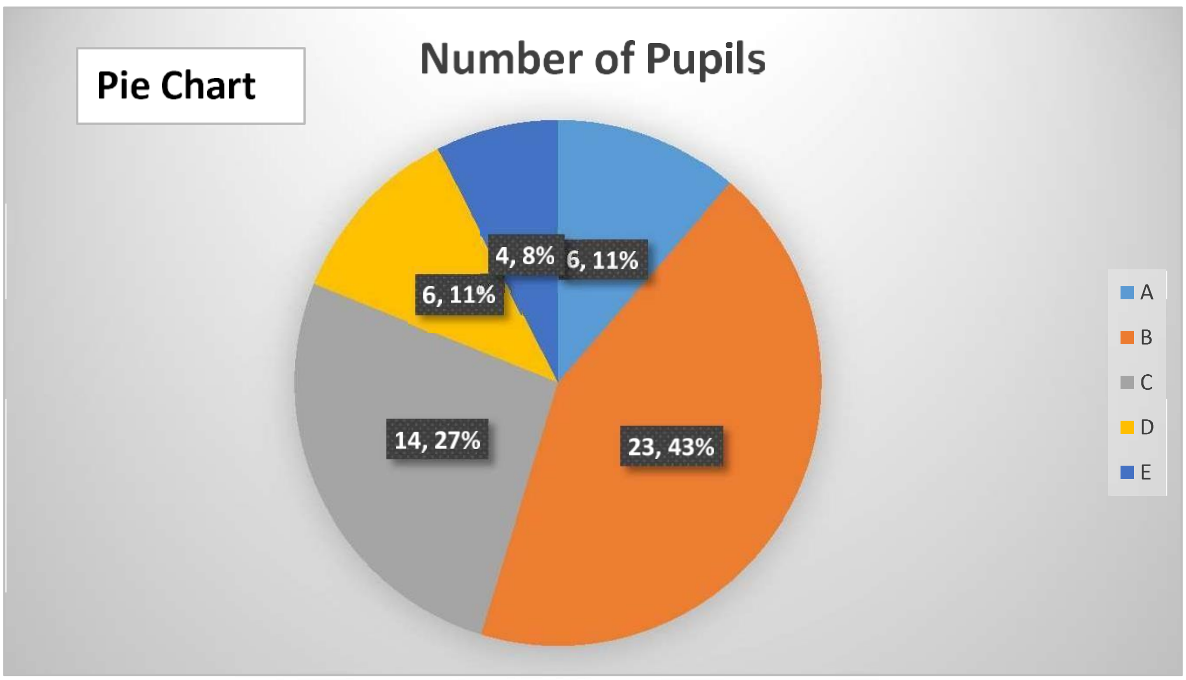
In [9]:

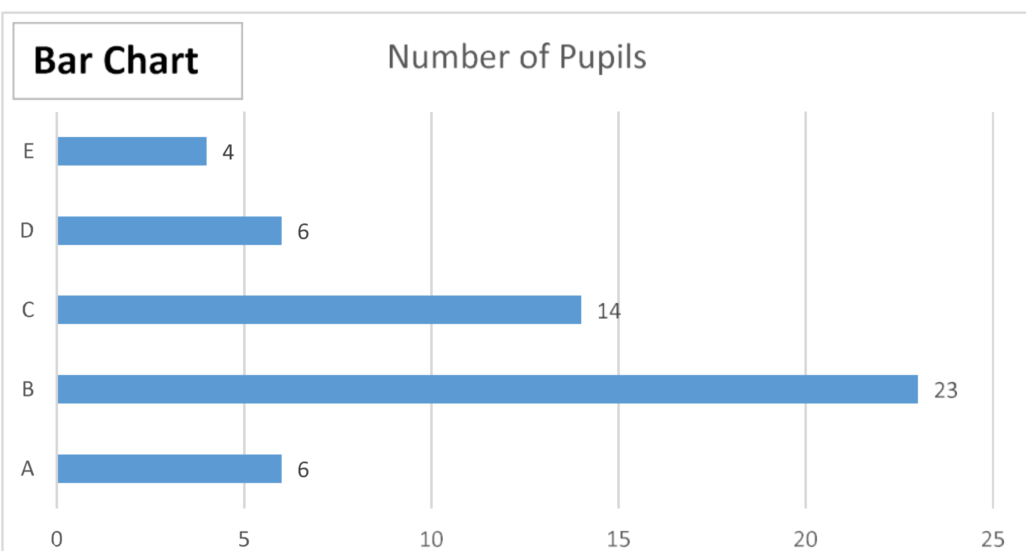
```
def is_safe(vertex, graph, color, c):
    for i in range(len(graph)):
        if graph[vertex][i] and color[i] == c:
            return False
    return True

def graph_coloring_util(graph, m, color, v):
    if v == len(graph):
        return True

    for c in range(1, m+1):
        if is_safe(v, graph, color, c):
            color[v] = c
            if graph_coloring_util(graph, m, color, v+1):
                return True
            color[v] = 0

def graph_coloring(graph, m):
    color = [0] * len(graph)
    if not graph_coloring_util(graph, m, color, 0):
        print("No solution exists")
        return False

    print("The assigned colors are:")
    for c in color:
        print(c, end=" ")
    return True

if __name___== "__main__":
    n = int(input("Enter the number of vertices: "))
    graph = []
    print("Enter the adjacency matrix (row-wise) of the graph:")
    for _ in range(n):
        row = list(map(int, input().split()))
        graph.append(row)

    m = int(input("Enter the number of colors available: "))

    graph_coloring(graph, m)
```

```
Enter the number of vertices: 4
Enter the adjacency matrix (row-wise) of the graph:
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Enter the number of colors available: 3
The assigned colors are:
1 2 3 2
```

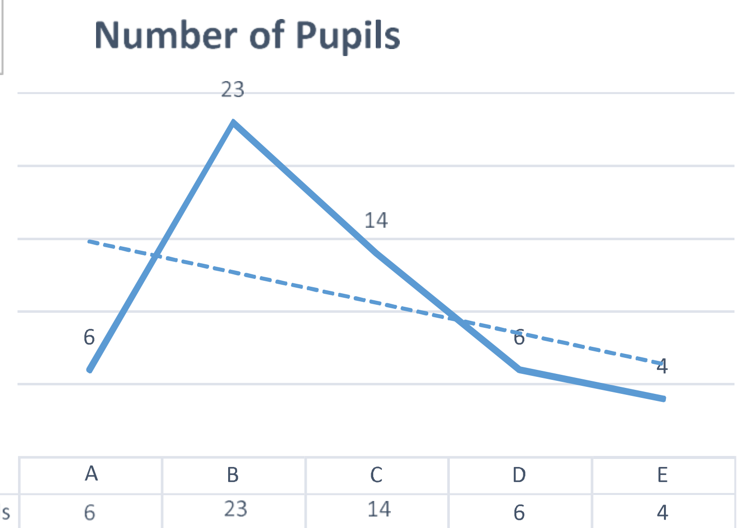
Lab 4 - Bar chart,pie chart ,line graph ,area graph, scatter plot in Excel.

Test Score	Number of Pupils
A	6
B	23
C	14
D	6
E	4

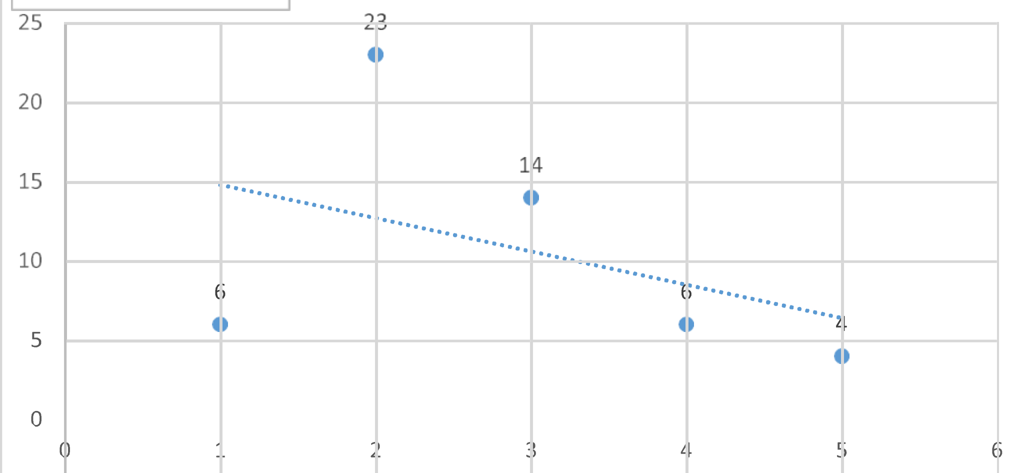




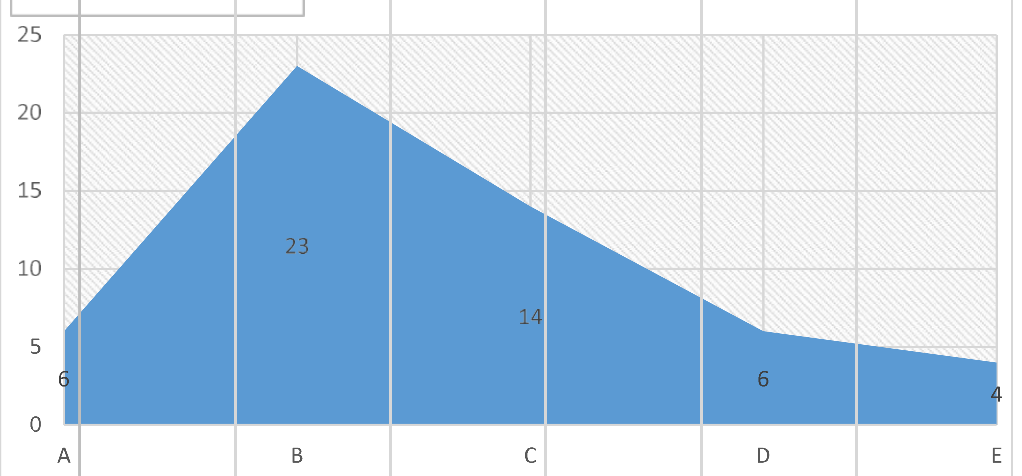
Line Graph



Scatter Plot



Area Chart



Lab - 5 : Python program to Scale 2D Data Co-ordinates

In [1]:
```
def scale_points(points, scale_factor):
    """
    Scale 2D coordinate points by a scale factor.

    Args:
    - points: A list of tuples representing 2D coordinate points,
      e.g., [(x1, y1), (x2, y2), ...]
    - scale_factor: The scale factor by which to scale the points.

    Returns:
    - scaled_points: A list of tuples representing the scaled 2D coordinate points.
    """
    scaled_points = []
    for point in points:
        scaled_x = point[0] * scale_factor
        scaled_y = point[1] * scale_factor
        scaled_points.append((scaled_x, scaled_y))
    return scaled_points

# Example usage:
points = [(1, 2), (3, 4), (5, 6)]
scale_factor = 2
scaled_points = scale_points(points, scale_factor)
print("Original Points:", points)
print("Scaled Points:", scaled_points)
```

```
Original Points: [(1, 2), (3, 4), (5, 6)]
Scaled Points: [(2, 4), (6, 8), (10, 12)]
```

In [ ]:

Lab 6: Python Program to Plot Histogram, Bar Graph, Pie Chart, Line Graph, Scatter Plot and Area Graph for a given excel (csv) dataset

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [7]:
```
# Read data from Excel file
insurance_df = pd.read_csv('insurance.csv')
```

In [8]:
```
df
```

Out[8]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

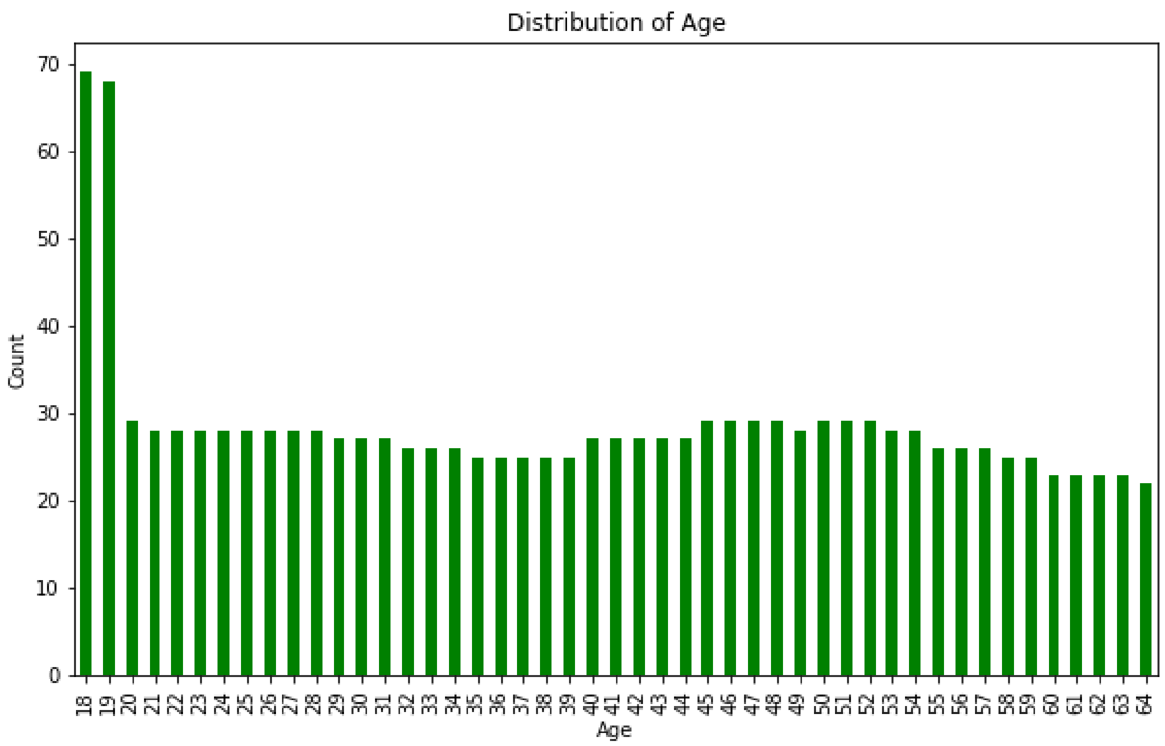
In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338  non-null  float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ K
```

Bar Chart

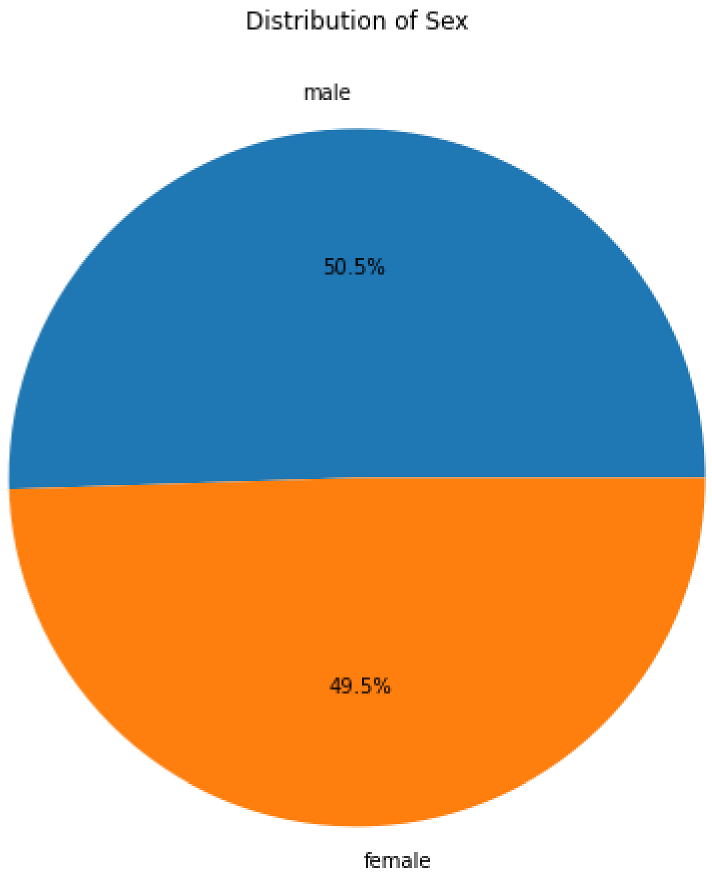
```
# Bar chart
plt.figure(figsize=(10, 6))
insurance_df['age'].value_counts().sort_index().plot(kind='bar', color='green')
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Distribution of Age')
plt.show()
```

In [15]:



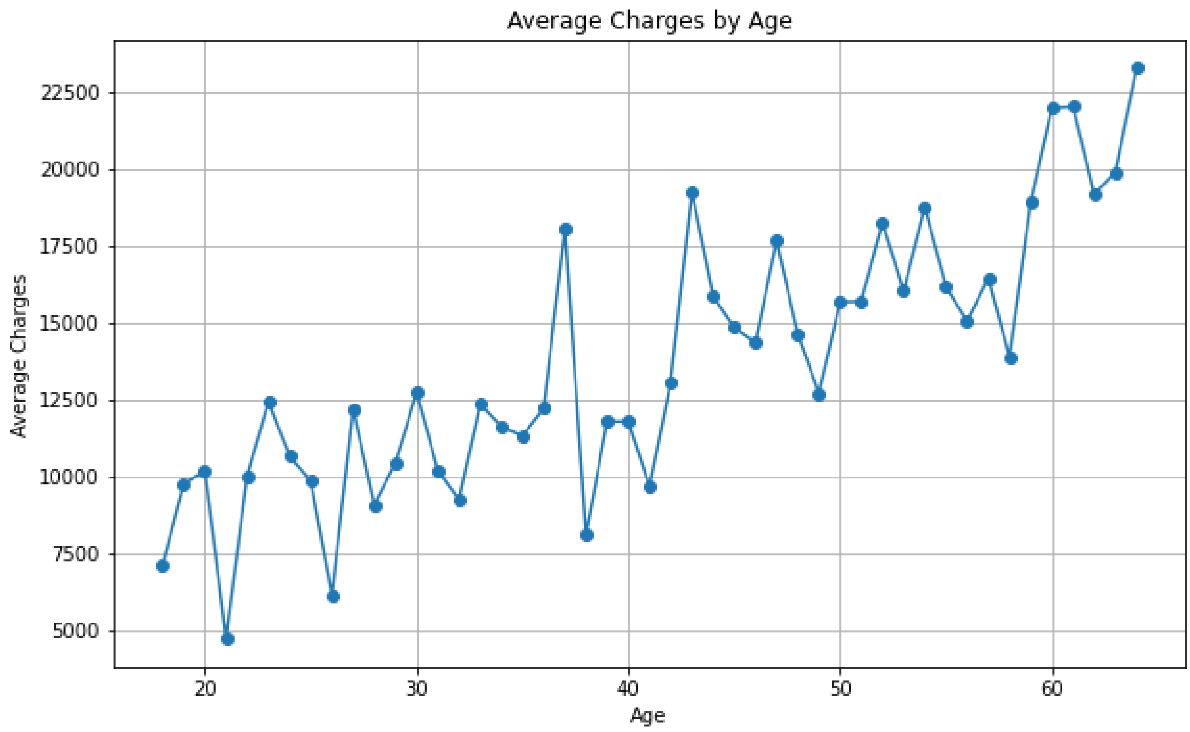
Pie Chart

In [11]:
```
# Pie chart
plt.figure(figsize=(8, 8))
insurance_df['sex'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of Sex')
plt.ylabel('')
plt.show()
```



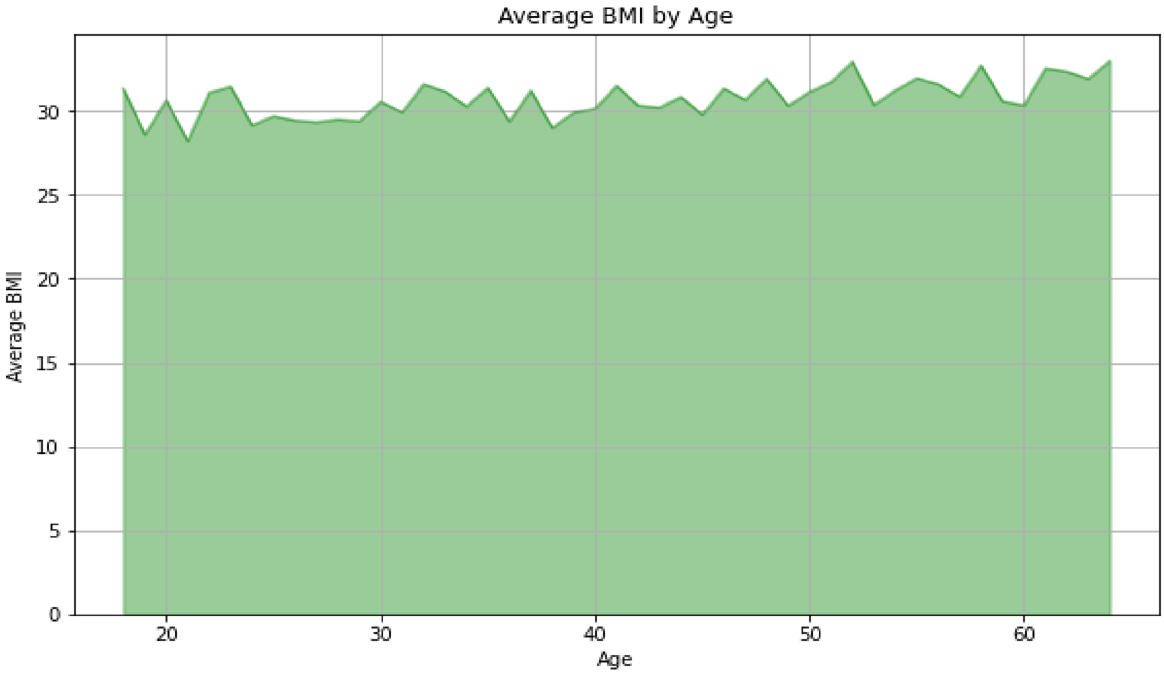
Line Graph

In [12]:
```
# Line graph
plt.figure(figsize=(10, 6))
insurance_df.groupby('age')['charges'].mean().plot(kind='line', marker='o')
plt.xlabel('Age')
plt.ylabel('Average Charges')
plt.title('Average Charges by Age')
plt.grid(True)
plt.show()
```



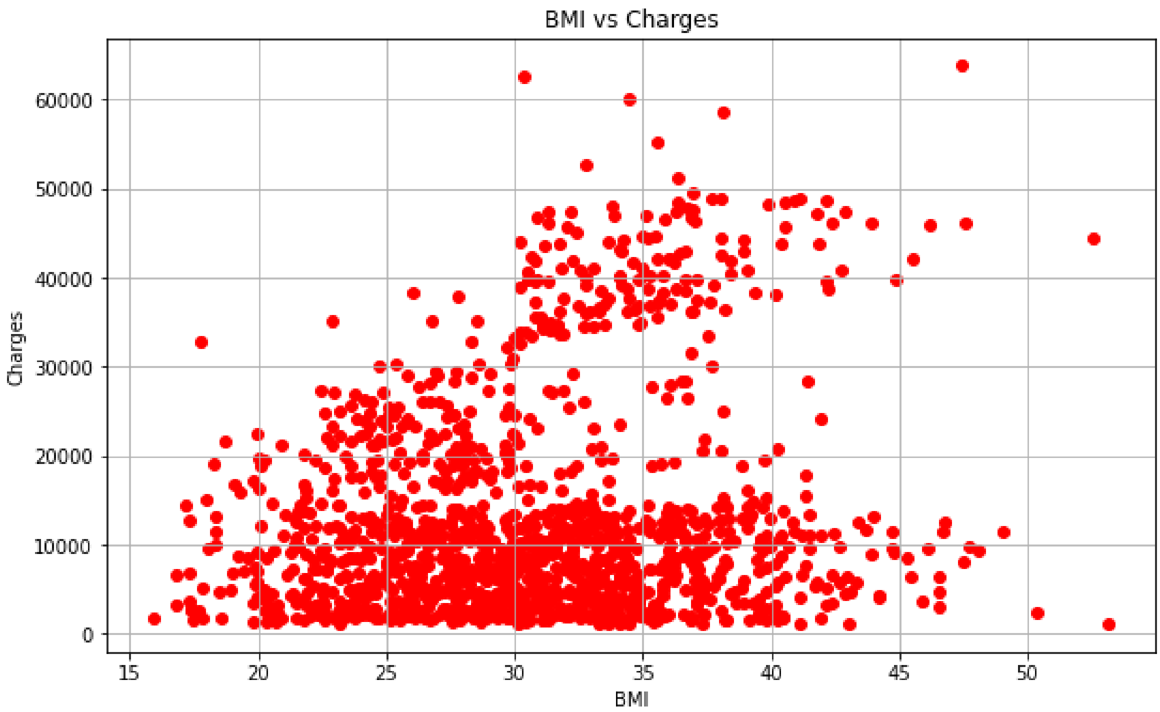
Area Graph

In [14]:
```
# Area graph
plt.figure(figsize=(10, 6))
insurance_df.groupby('age')['bmi'].mean().plot(kind='area',
                                                color='green',  alpha=0.4)
plt.xlabel('Age')
plt.ylabel('Average BMI')
plt.title('Average BMI by Age')
plt.grid(True)
plt.show()
```



Scatter Plot

In [16]:
```
# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(insurance_df['bmi'], insurance_df['charges'], color='red')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title('BMI vs Charges')
plt.grid(True)
plt.show()
```



Lab 7: Linear Regression on Boston Housing Dataset

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [4]:
```
# Load the Boston Housing Dataset
boston_dataset = load_boston()
```

In [5]:
```
# Convert the dataset into a DataFrame
boston_df = pd.DataFrame(data=boston_dataset.data, columns=boston_dataset.feature_
boston_df['MEDV'] = boston_dataset.target  # Adding target variable 'MEDV'
```

In [8]:
```
boston_df
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.1
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.9
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.6
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.0
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.6
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.4
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.8

506 rows × 14 columns

In [6]:
```
# Splitting the dataset into features and target variable
X = boston_df.drop('MEDV', axis=1)
y = boston_df['MEDV']
```

Train Test Split

In [9

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

Instantiate a Linear Regression Model

In [10]:
```
# Create a linear regression model
model = LinearRegression()
```

Fitting/Training the Model on the Train Set

In [11]:
```
# Train the model using the training sets
model.fit(X_train, y_train)
```

Out[11]:
```
LinearRegression()
```

Performance Evaluation on Test Set

In [12]:
```
# Make predictions using the testing set
y_pred = model.predict(X_test)
```

In [13]:
```
# Print the coefficients
print('Coefficients:', model.coef_)
```

```
Coefficients: [-1.13055924e-01  3.01104641e-02  4.03807204e-02  2.78443820e+00
 -1.72026334e+01 4.43883520e+00 -6.29636221e-03 -1.44786537e+00
  2.62429736e-01 -1.06467863e-02 -9.15456240e-01  1.23513347e-02
 -5.08571424e-01]
```

In [14]:
```
# Print the mean squared error
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
```

```
Mean Squared Error: 24.29111947497371
```

In [15]:
```
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred)
print('R-squared Score:', r2)
```

```
R-squared Score: 0.6687594935356294
```

In [16]:
```
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', mae)
```

```
Mean Absolute Error: 3.189091965887875
```

In [18]:
```
import numpy as np
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', rmse)
```

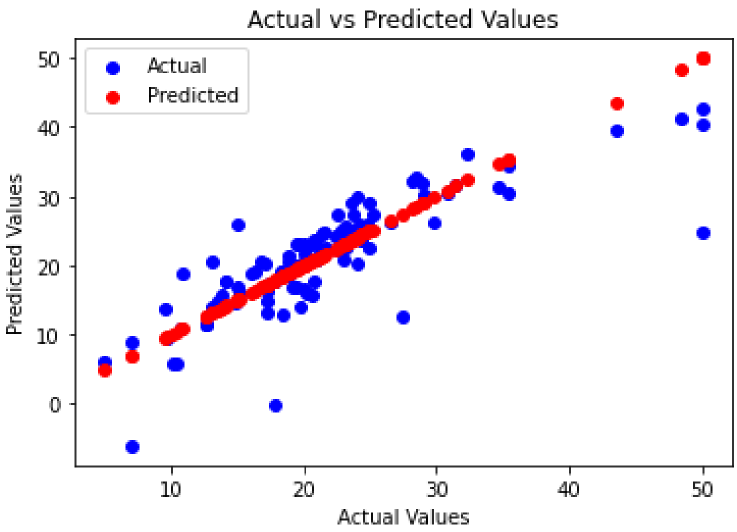
```
Root Mean Squared Error: 4.928602182665355
```

In [19]:
```
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
print('Mean Absolute Percentage Error:', mape)
```

```
Mean Absolute Percentage Error: 16.866394539378827
```

Scatter Plot for Visualizing Actual vs Predicted Values

In [22]:
```
import matplotlib.pyplot as plt

# Scatter plot of Actual vs. Predicted Values with different colors
plt.scatter(y_test, y_pred, color='blue', label='Actual')
plt.scatter(y_test, y_test, color='red', label='Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```



Lab 8: Multiple Linear Regression on Boston Housing Dataset

In [3]:
```
# Importing necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [5]:
```
# Load the Boston Housing dataset
boston = load_boston()
```

In [6]:
```
# Creating a DataFrame from the dataset
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
```

In [7]:
```
boston_df
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.1
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.9
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.6
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.0
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.6
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.4
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.8

506 rows × 13 columns

In [8]:
```
# Adding the target variable to the DataFrame
boston_df['PRICE'] = boston.target
```

In [9]:
```
# Separating features and target variable
X = boston_df.drop('PRICE', axis=1)
y = boston_df['PRICE']
```

Train Test Split

In [10]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

Fitting a Linear Regression Model on Multiple Independent Variables

In [11]:
```
# Creating and training the model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[11]:
```
LinearRegression()
```

Performance Evaluation

In [15]:
```
# Predicting on the testing set
y_pred = model.predict(X_test)
```

In [21]:
```
# Print the coefficients
print('Coefficients:', model.coef_)
```

```
Coefficients: [-1.19443447e-01  4.47799511e-02  5.48526168e-03  2.34080361e+00
 -1.61236043e+01  3.70870901e+00 -3.12108178e-03 -1.38639737e+00
  2.44178327e-01 -1.09896366e-02 -1.04592119e+00  8.11010693e-03
 -4.92792725e-01]
```

In [16]:
```
# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [18]:
```
print("Mean Squared Error:", mse)
print("R^2 Score:", r2)
print("Intercept:", model.intercept_)
```

```
Mean Squared Error: 33.448979997676524
R^2 Score: 0.589222384918251
Intercept: 38.09169492630278
```

In [22]:
```
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error:', mae)
```

```
Mean Absolute Error: 3.842909220444505
```

In [23]:
```
import numpy as np
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', rmse)
```

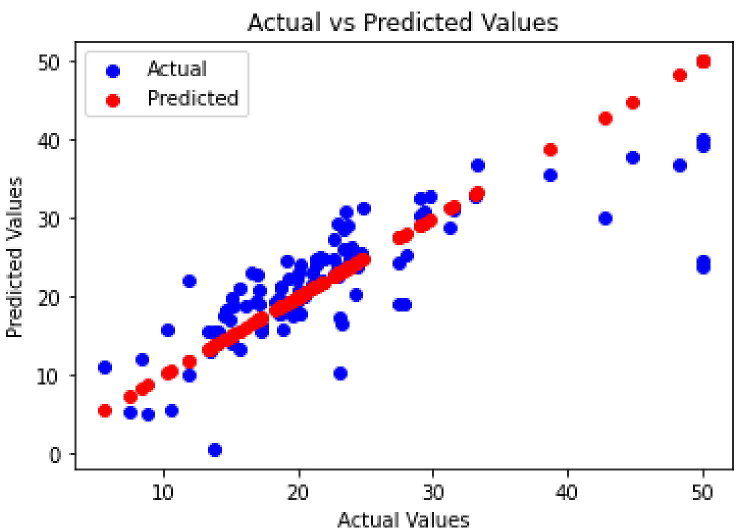
```
Root Mean Squared Error: 5.783509315085134
```

In [24]:
```
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
print('Mean Absolute Percentage Error:', mape)
```

```
Mean Absolute Percentage Error: 18.356285293906495
```

Scatter Plot for Visualizing Actual vs Predicted Values

In [20]:
```
import matplotlib.pyplot as plt

# Scatter plot of Actual vs. Predicted Values with different colors
plt.scatter(y_test, y_pred, color='blue', label='Actual')
plt.scatter(y_test, y_test, color='red', label='Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```



Lab 9: Python Program to Create and Handle Frequency Table

In [1]:

```
def create_frequency_table(data):
    frequency_table = {}
    for item in data:
        if item in frequency_table:
            frequency_table[item] += 1
        else:
            frequency_table[item] = 1
    return frequency_table

def display_frequency_table(frequency_table):
    print("Item\tFrequency")
    print("----------------- ")
    for item, frequency in frequency_table.items():
        print(f"{item}\t{frequency}")
```

In [2]:

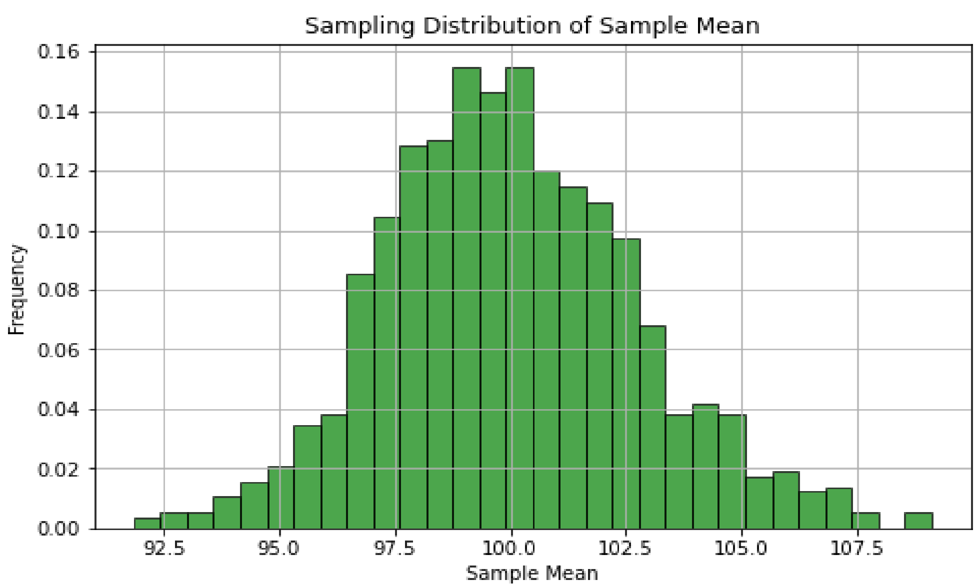
```
def main():
    # Sample data
    data = [1, 2, 3, 1, 2, 3, 4, 5, 1, 2, 3, 4, 1, 2, 1]

    # Create frequency table
    frequency_table = create_frequency_table(data)

    # Display frequency table
    display_frequency_table(frequency_table)

if _name_____== "_main _":
    main()
```

```
Item     Frequency
--------------------
1        5
2        4
3        3
4        2
5        1
```

Lab 10: Python Program to Demonstrate Sampling Distribution

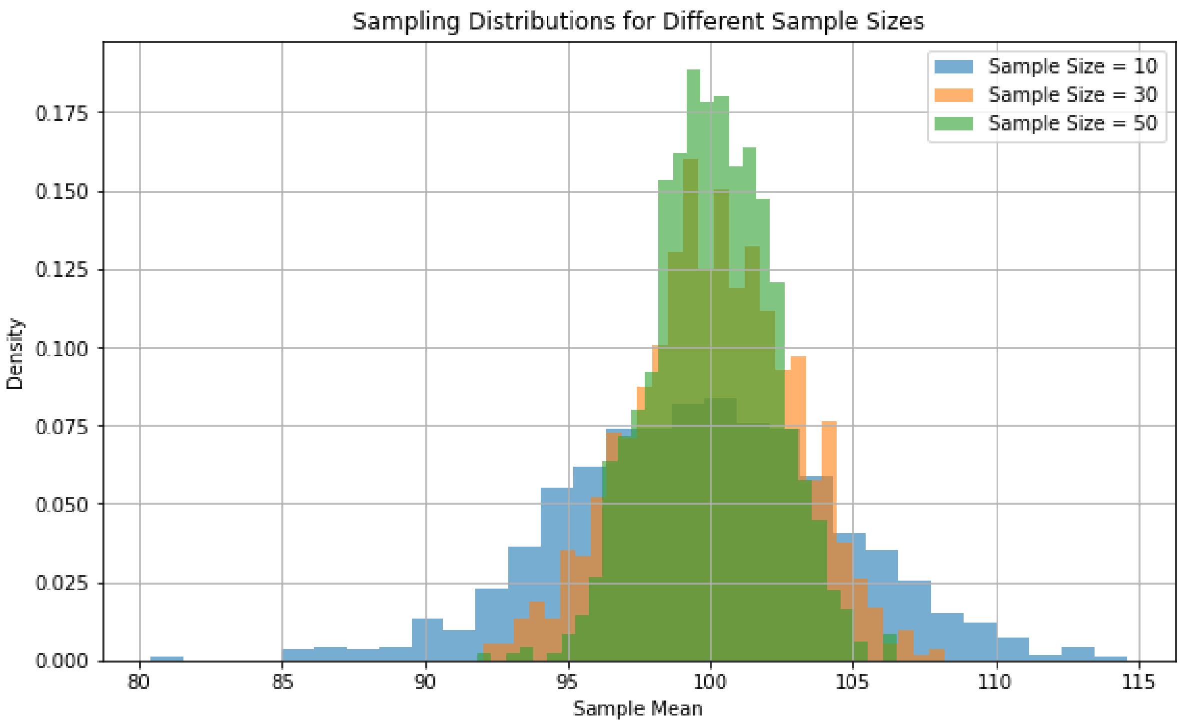
In [5]:

```
import numpy as np
import matplotlib.pyplot as plt

# Population parameters
population_mean = 100
population_std = 15
population_size = 10000

# Generate the population data
population_data = np.random.normal(population_mean, population_std, population_size

# Number of samples and sample size
num_samples = 1000
sample_size = 30

# Generate sampling distribution of the sample mean
sample_means = []
for _ in range(num_samples):
    sample = np.random.choice(population_data, size=sample_size, replace=False)
    sample_mean = np.mean(sample)
    sample_means.append(sample_mean)

# Plotting the sampling distribution
plt.figure(figsize=(8, 5))
plt.hist(sample_means, bins=30, density=True, color='green', edgecolor='black', al
plt.title('Sampling Distribution of Sample Mean')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



Analysis of how sampling distribution changes with sample size.

In [6]:
```
# Population parameters
population_mean = 100
population_std = 15
population_size = 10000

# Generate the population data
population_data = np.random.normal(population_mean, population_std, population_size

# Number of samples
num_samples = 1000

# Sample sizes to analyze
sample_sizes = [10, 30, 50]

# Plotting the sampling distributions for different sample sizes
plt.figure(figsize=(10, 6))

for sample_size in sample_sizes:
    sample_means = []
    for _ in range(num_samples):
        sample = np.random.choice(population_data, size=sample_size, replace=False
        sample_mean = np.mean(sample)
        sample_means.append(sample_mean)

    plt.hist(sample_means, bins=30, density=True, alpha=0.6, label=f'Sample Size =

plt.title('Sampling Distributions for Different Sample Sizes')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```


Sampling Distributions for Different Sample Sizes
Sample Size = 10
Sample Size = 30
Sample Size = 50
Density
Sample Mean
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
80
85
90
95
100
105
110
115