## Setting up the Environment

In [2]:
```
!pip install matplotlib seaborn pandas
```

## Importing the Libraries

In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

## Loading the Dataset

In [4]:
```
# Loading built-in dataset from seaborn
tips = sns.load_dataset('tips')
```

## Basic Statistics and Data Exploration

In [6]:
```
tips.head()
```

Out[6]:

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

In [7]:
```
tips.describe()
```

Out[7]:

| | total_bill | tip | size |
|---|---|---|---|
| **count** | 244.000000 | 244.000000 | 244.000000 |
| **mean** | 19.785943 | 2.998279 | 2.569672 |
| **std** | 8.902412 | 1.383638 | 0.951100 |
| **min** | 3.070000 | 1.000000 | 1.000000 |
| **25%** | 13.347500 | 2.000000 | 2.000000 |
| **50%** | 17.795000 | 2.900000 | 2.000000 |
| **75%** | 24.127500 | 3.562500 | 3.000000 |
| **max** | 50.810000 | 10.000000 | 6.000000 |

## Questions to Ask Yourself Before Visualizing Your Data

Before diving into plotting your data, it's indeed important to ask some key questions that can guide your visualization process. Here are some fundamental ones:

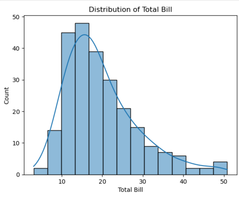
- What is the purpose of the visualization? Are you trying to explore your data, present findings, or persuade an audience?
- Who is the audience? Is it a technical audience, general public, or decision-makers in a business setting? This will help guide the level of complexity and detail you need to include in your plots.
- What kind of plot would best explain the data? Different types of plots are suited to different types of data and analysis. Histograms for distributions, scatter plots for relationships, bar plots for comparisons, etc.
- Do you want to represent relationships between variables? If so, how many variables do you want to include in a single plot? This can influence the choice of the plot (e.g., scatter plots for two continuous variables, bubble plots for three variables, etc.)
- How can you ensure clarity and simplicity? You should aim for a plot that communicates the necessary information as simply and clearly as possible. Avoid unnecessary decorations, complex layouts, or confusing color schemes.
- What insights do you want to highlight? Your plot should ideally make these insights easily noticeable at first glance.
- How can you ensure that the plot is accessible? This includes considerations like color blindness-friendly palettes, clear font sizes, and appropriate contrast.
- Is interactivity necessary? Interactive plots can be beneficial when dealing with high-dimensional data or when you want the audience to explore the data on their own.

## Plotting the Data

## Histogram:

We use a histogram when we want to see the distribution of a single variable. It gives us an idea about the spread and central tendency of the data. In this plot, the X-axis represents the total bill amount, and the Y-axis shows how often that amount appears (count). The 'bell curve' (Kernel Density Estimation line) provides a smoothed estimate of the distribution. Ideally, the height of each bar should reflect the frequency of occurrence for each bin of values, with the pattern showing data distribution.

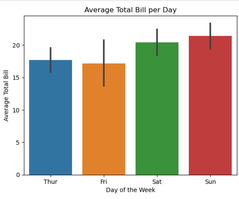
In [8]:
```
# Histogram: Shows the distribution of a single variable
sns.histplot(data=tips, x="total_bill", kde=True)
plt.title('Distribution of Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Count')
plt.show()
```




## Bar Plot:

We use a bar plot to compare a categorical variable (in this case, day of the week) with a continuous variable (total bill). Each bar represents a category (day), and its height signifies its mean value (average total bill). The tiny line on each bar represents the confidence interval around that mean. This plot lets us easily compare the average total bill for each day of the week. Ideally, each bar's height should accurately represent the average value, and categories should be clearly separated.

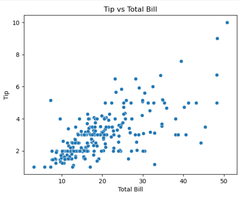
In [10]:
```
# Bar Plot: Compares a categorical variable with a continuous variable
sns.barplot(x="day", y="total_bill", data=tips)
plt.title('Average Total Bill per Day')
plt.xlabel('Day of the Week')
plt.ylabel('Average Total Bill')
plt.show()
```




## Scatter Plot:

A scatter plot is ideal when we want to observe the relationship between two numerical variables. Each dot represents an observation - its X-Y position corresponds to its values for the two variables. This plot illustrates the relationship between total bill and tip amount. A trend line (not present here) would further help visualize the correlation. Ideally, a scatter plot should show a clear trend or pattern, or lack thereof, between the variables.

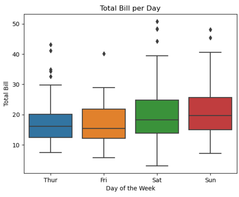
In [10]:
```
# Scatter Plot: Shows the relationship between two numeric variables
sns.scatterplot(x="total_bill", y="tip", data=tips)
plt.title('Tip vs Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```




## Box Plot:

A box plot (or box-and-whisker plot) shows the distribution of numerical data by depicting the data quartiles, and potentially, outliers. The 'box' represents the interquartile range (25th to 75th percentile), the line inside the box is the median, and the 'whiskers' represent the range of the data. This plot gives us a good idea of how the total bill varies each day, showing median, spread, and potential outliers. Ideally, a box plot should clearly indicate the median, quartiles, and any potential outliers (data points that fall beyond the ends of the whiskers).

In [12]:
```
# Box Plot: Depicts groups of numerical data through their quartiles
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title('Total Bill per Day')
plt.xlabel('Day of the Week')
plt.ylabel('Total Bill')
plt.show()
```

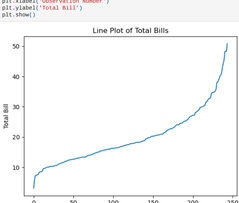



## Line Plot

Line plots are used to observe trends and patterns over a time or ordered categories. However, the 'tips' dataset doesn't contain any time series or ordinal categories, so for the sake of an example, we'll create a line plot of total bills sorted by size. We're using a line plot here to show the trend of total bills when sorted from smallest to largest. Each point on the line represents an individual observation, and the line helps visualize the trend. The X-axis represents the ordered observation number, and the Y-axis represents the total bill amount. In an ideal line plot, clear trends or patterns should emerge.

In [13]:
```
# Sorting values by total_bill
tips_sorted = tips.sort_values('total_bill')

sns.lineplot(x=range(len(tips_sorted)), y="total_bill", data=tips_sorted)
plt.title('Line Plot of Total Bills')
plt.xlabel('Observation Number')
plt.ylabel('Total Bill')
plt.show()
```

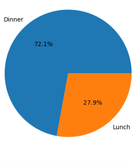



## Pie Chart

Pie charts are best used when you want to compare parts of a whole. They do not work well with very large categories or small proportional differences. Let's create a pie chart to show the distribution of meals in our dataset. First, we need to get the count of meals (lunch and dinner) from our 'tips' dataset. We use a pie chart to show the proportional distribution of categorical data - in this case, the distribution of lunch and dinner in our dataset. Each slice of the pie represents a category, and its size shows its proportion to the whole. Ideally, a pie chart should clearly show relative proportions, and labels or annotations should provide exact values or percentages.

In [14]:
```
meal_counts = tips['time'].value_counts()
print(meal_counts)
plt.pie(meal_counts, labels = meal_counts.index, autopct='%1.1f%%')
plt.title('Meal Type Distribution')
plt.show()
```

```
Dinner    176
Lunch      68
Name: time, dtype: int64
```




In [ ]:
```

```