PYTHON FUNDAMENTALS: A COMPREHENSIVE TRAINING

Mastering Data Visualization with Python

Python Course -By Binatna Data-

- Sessions of December 2023
- Session of 30/12/2023

What is Data Visualization?

- The process of transforming data into visual representations to gain insights and communicate information effectively.
- Crucial for understanding complex data.

Example

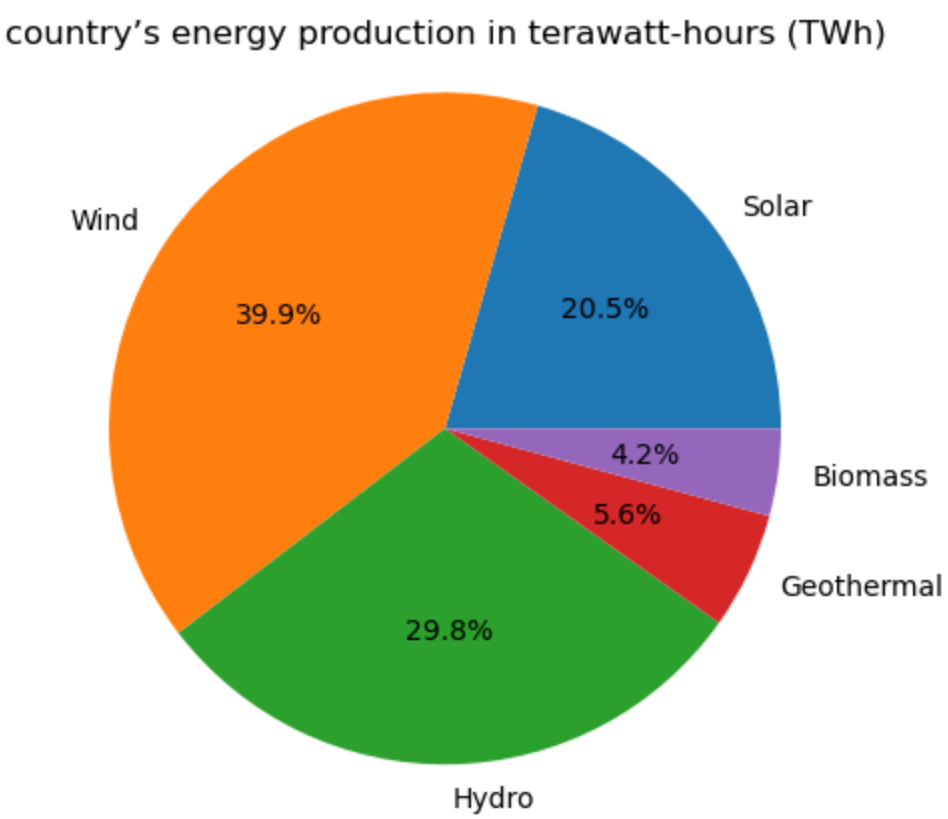
Suppose we have the following data for a `country's energy production in terawatt-hours (TWh)` from various renewable sources for the year 2023:

In [1]:
```
data = {
    'Solar': 274,
    'Wind': 532,
    'Hydro': 397,
    'Geothermal': 75,
    'Biomass': 56
}
```

This data is informative, but it's not immediately clear how the different sources compare to each other or what part each source plays in the total renewable energy production.

Now, let's transform this data into a visual representation:

In [15]:
```
import matplotlib.pyplot as plt

# Data to plot
labels = data.keys()
sizes = data.values()

# Plot
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('country's energy production in terawatt-hours (TWh)')
plt.show()
```

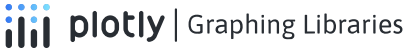


Different libraries in Python for data visualization

Python offers a variety of libraries for data visualization. Here are some of the most commonly used ones:

- `Matplotlib` : This is a 2-D plotting library that is widely used in the Python community. It can be used to create plots, bar charts, pie charts, histograms, scatterplots and more

- `Seaborn` : This is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.



- `Plotly` : This is a free open-source graphing library that can be used to form data visualizations. It provides more than 40 unique chart types like scatter plots, histograms, line charts, bar charts, pie charts..etc

---

Setting up the Environment

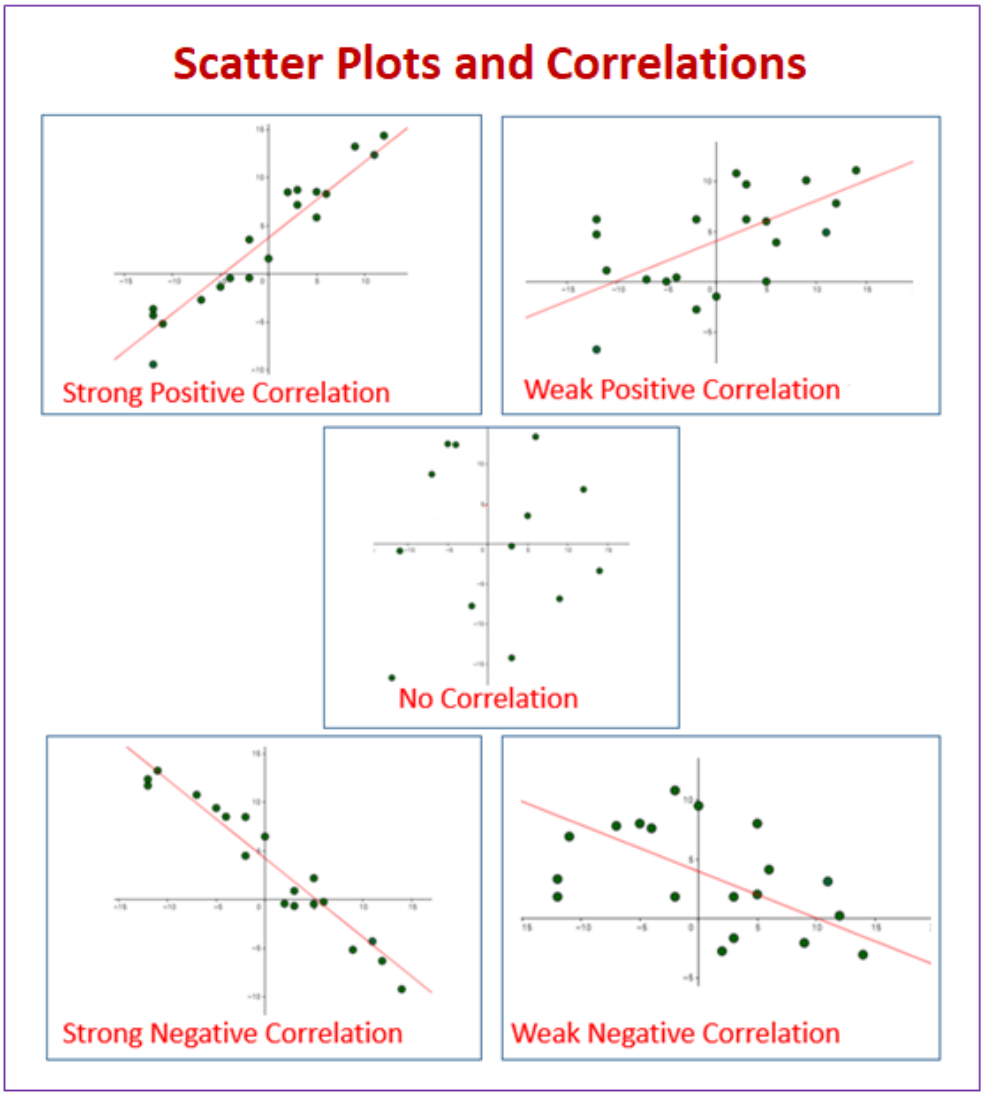
Before we can start visualizing data, `we need to install the necessary Python libraries`.

Here are the commands to install them using pip:

- `pip install matplotlib`
- `pip install seaborn`
- `pip install plotly`

Basic Charts and Their Uses

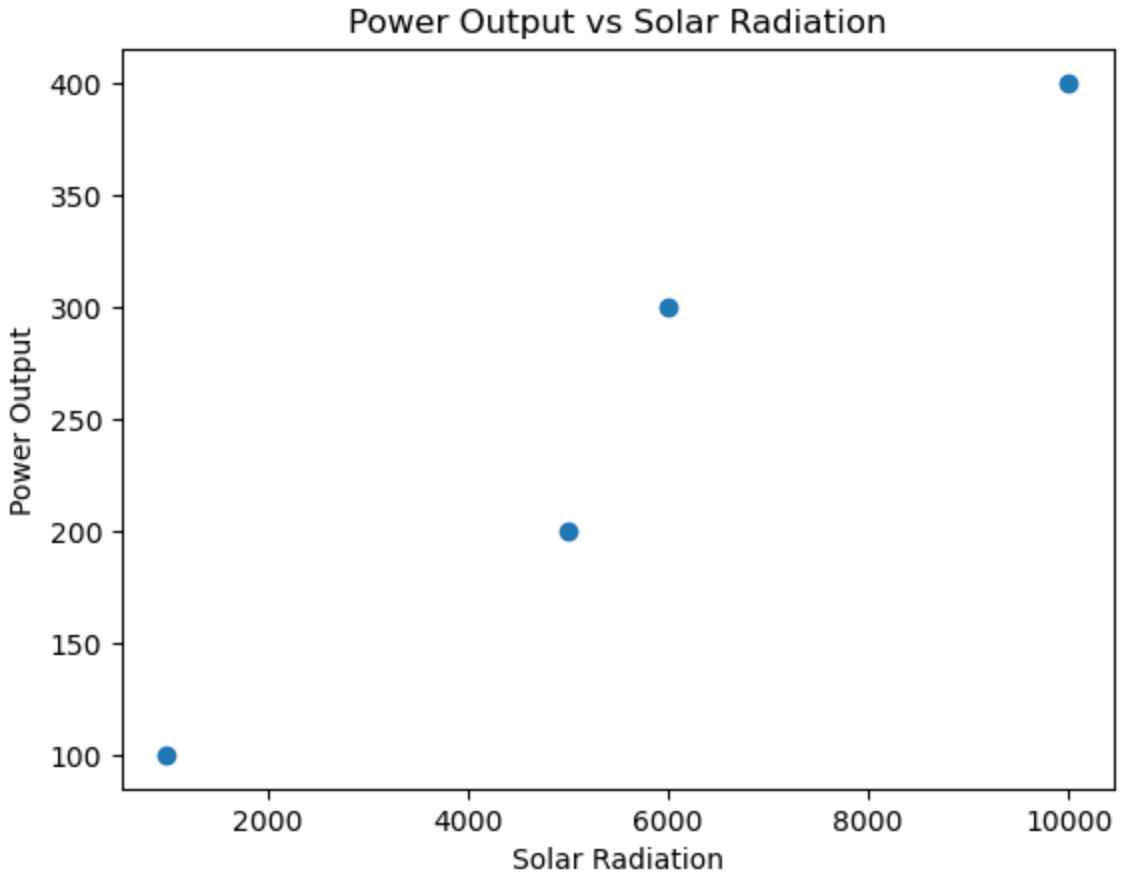
- `Scatter Plot`: A scatter plot can be used to observe `relationships between two variables`. For example, you might use a scatter plot to observe the relationship between solar radiation (x-axis) and power output (y-axis). Each point on the plot represents a single observation. If the points cluster along a line, this suggests a strong correlation between the variables.



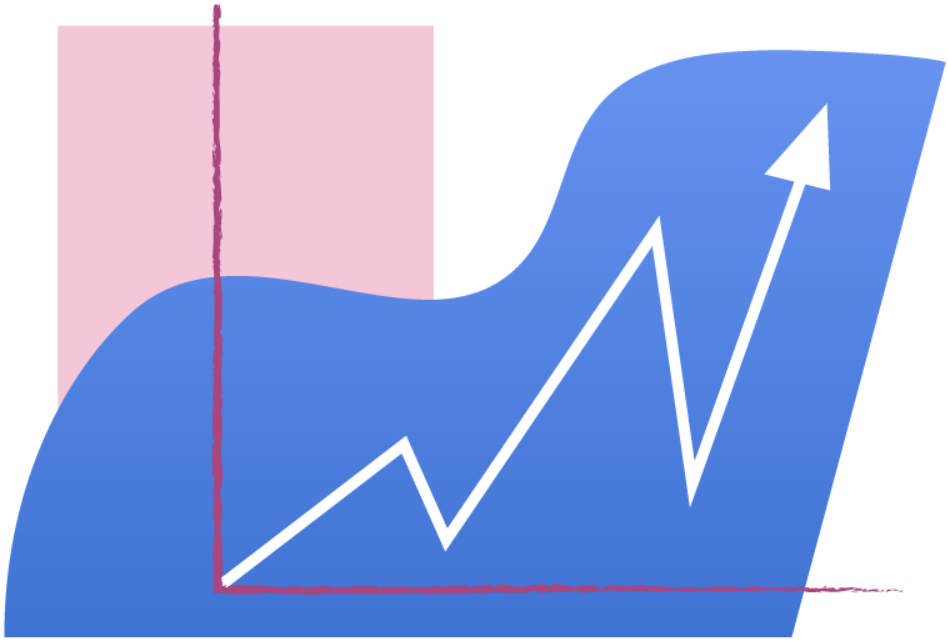
Example

Here's an example of a scatter plot using matplotlib:

In [16]:
```python
# import the package
import matplotlib.pyplot as plt

# Define or get data
solar_radiation = [1000, 5000, 6000, 10000]
power_output = [100, 200, 300, 400]

# Create your scatter plot
plt.scatter(solar_radiation, power_output)
plt.title('Power Output vs Solar Radiation')
plt.xlabel('Solar Radiation')
plt.ylabel('Power Output')
plt.show()
```

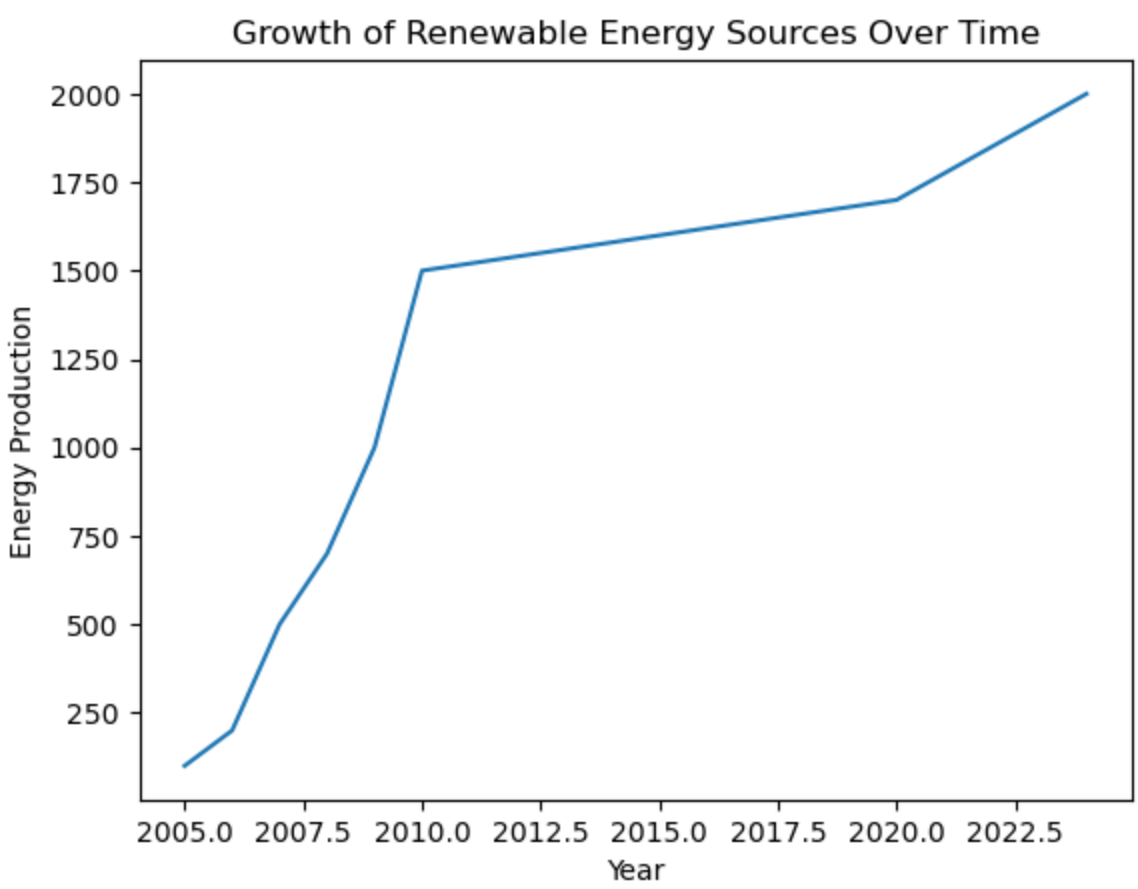


- `Line Chart` : A line chart is useful for displaying trends over time. For example, you might use a line chart to show the growth of renewable energy sources over several years. The x-axis represents time, and the y-axis represents the quantity you're interested in.

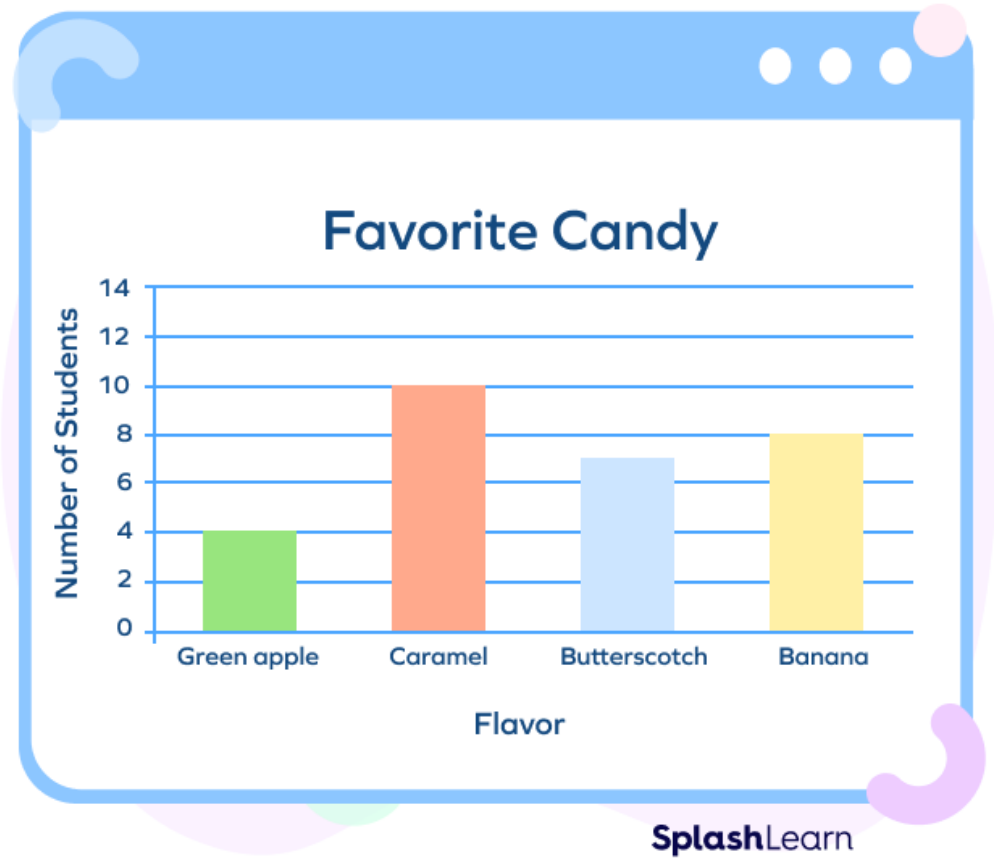


Here's an example of a line chart using matplotlib:

In [35]:
```
import matplotlib.pyplot as plt

# Get data
years = [2005,2006,2007,2008,2009,2010,2020,2024]
energy_production = [100,200,500,700,1000,1500,1700,2000]

plt.plot(years, energy_production)
plt.title('Growth of Renewable Energy Sources Over Time')
plt.xlabel('Year')
plt.ylabel('Energy Production')
plt.show()
```

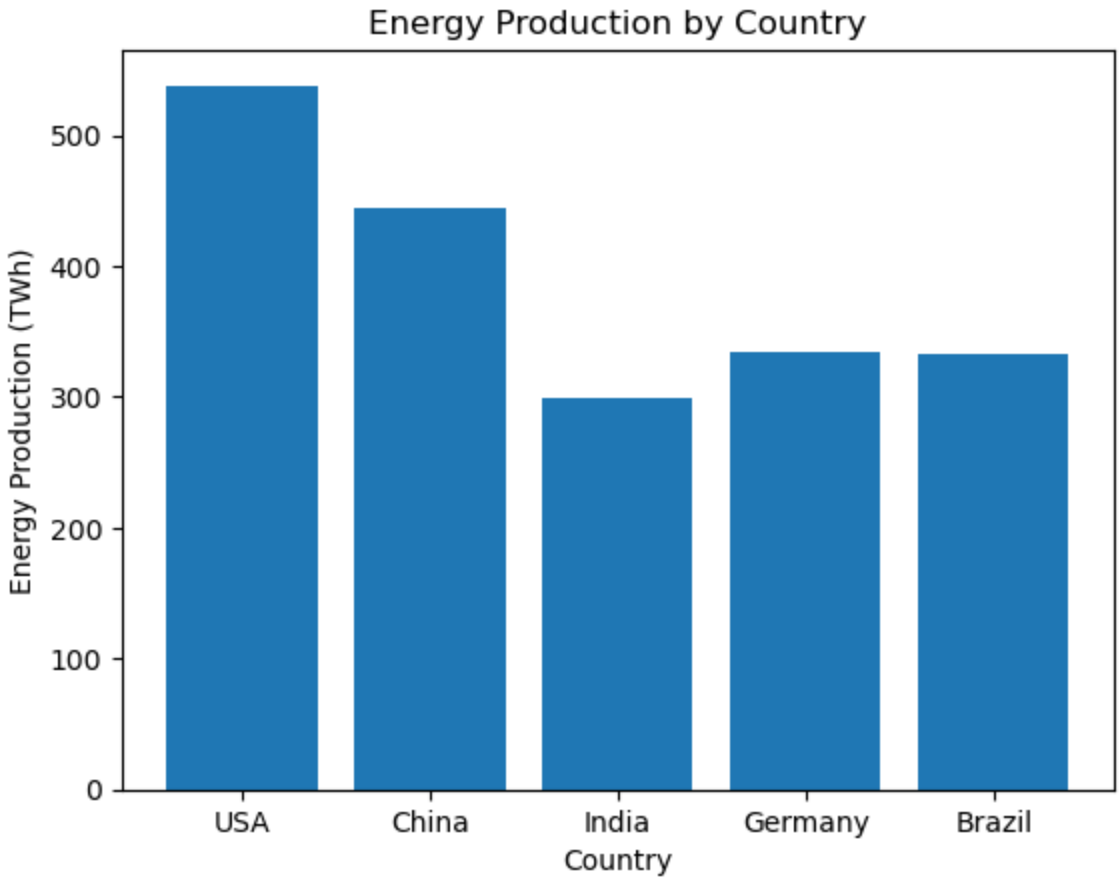


- `Bar Chart` : A bar chart is ideal for comparing quantities in different categories. For example, you might use a bar chart to compare the energy production of different countries. Each bar represents a category, and the height of the bar represents the quantity.

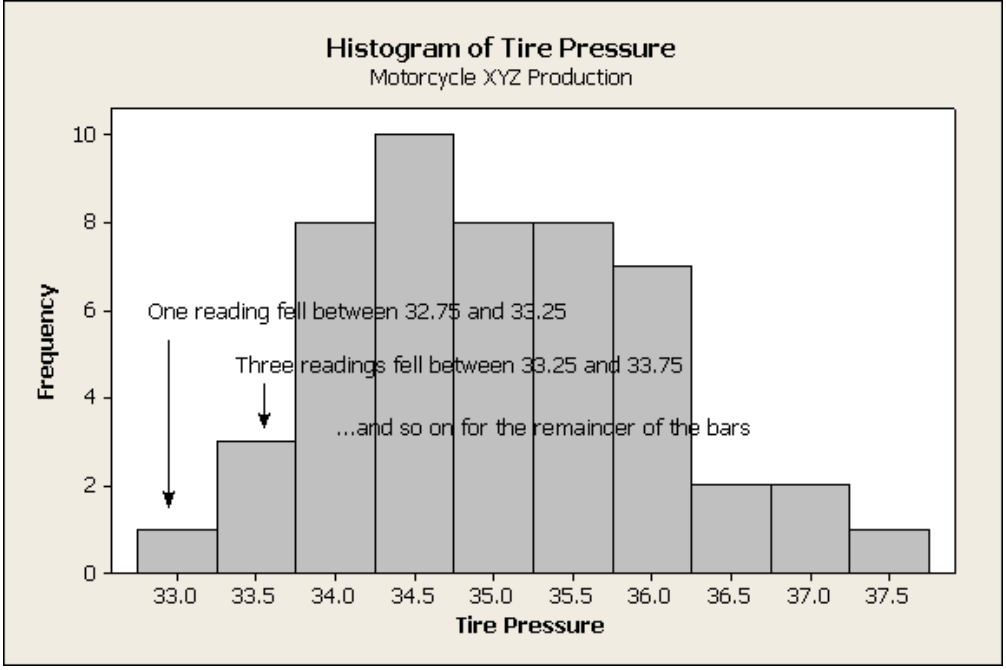


Here's an example of a line chart using matplotlib:

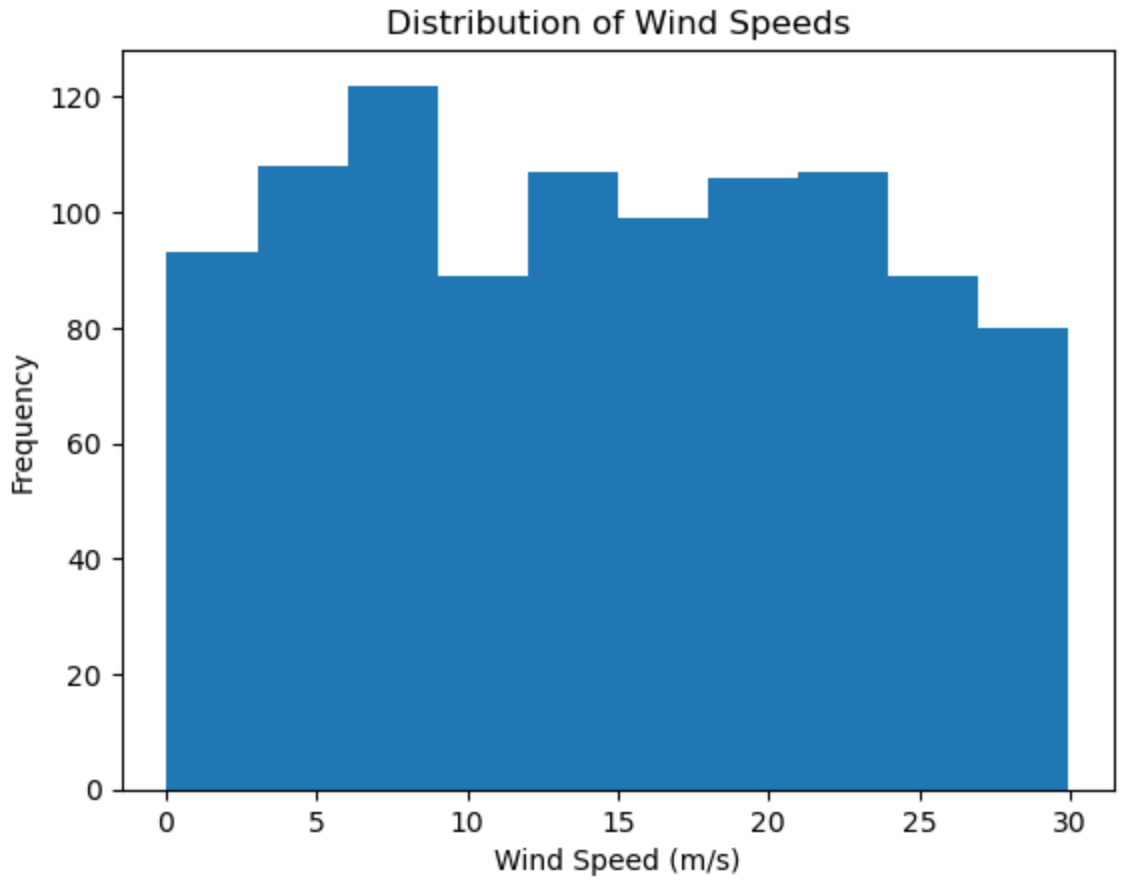
In [39]:
```
import random

# Get data
countries = ['USA', 'China', 'India', 'Germany', 'Brazil']
energy_production = [random.randint(200, 600) for _ in range(len(countries))]

# Create the bar chart
plt.bar(countries, energy_production)
plt.title('Energy Production by Country')
plt.xlabel('Country')
plt.ylabel('Energy Production (TWh)')
plt.show()
```



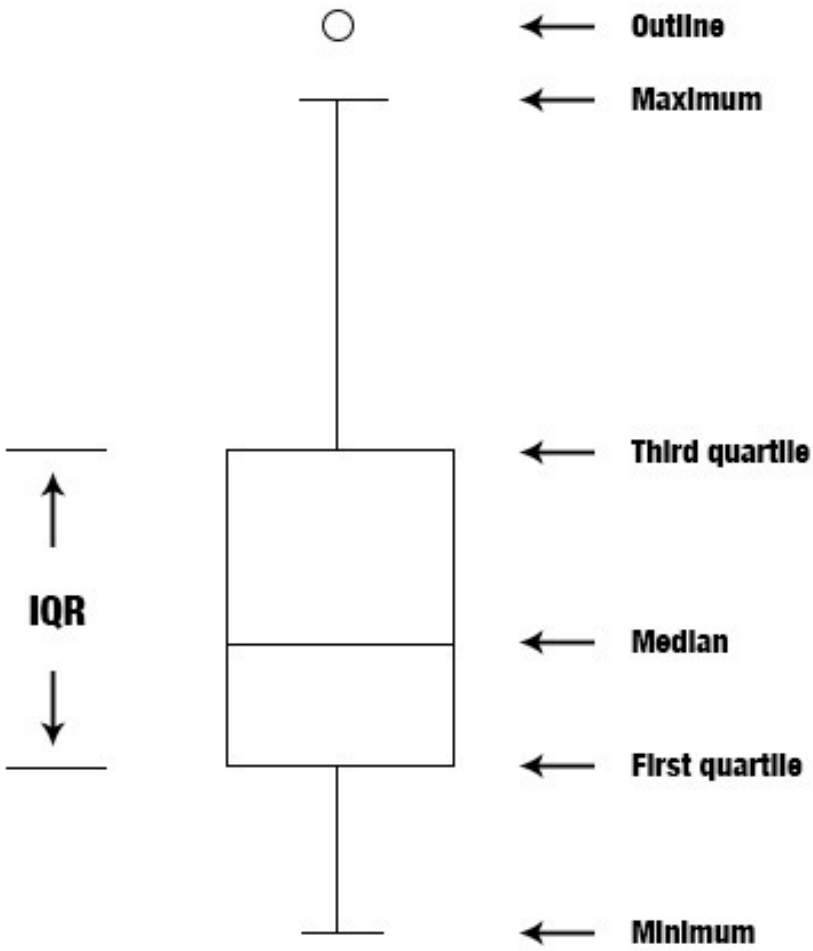
- `Histogram`: A histogram is used to represent the distribution of continuous data. For example, you might use a histogram to represent the distribution of wind speeds at a particular location. The x-axis represents the range of values, and the y-axis represents the frequency of each value.



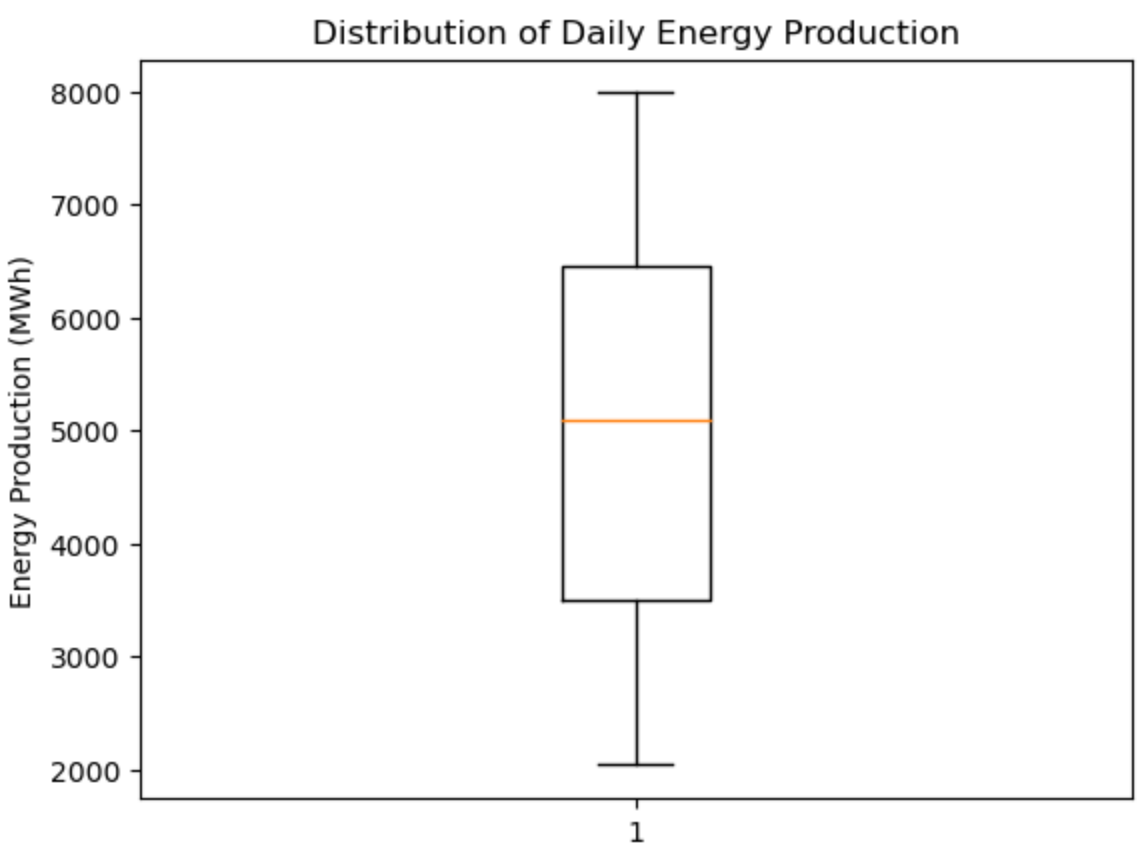
In [52]:
```
import matplotlib.pyplot as plt
import random

# Get data
wind_speeds = [random.uniform(0, 30) for _ in range(1000)]

# Create the histogram
plt.hist(wind_speeds, bins=10)
plt.title('Distribution of Wind Speeds')
plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Frequency')
plt.show()
```



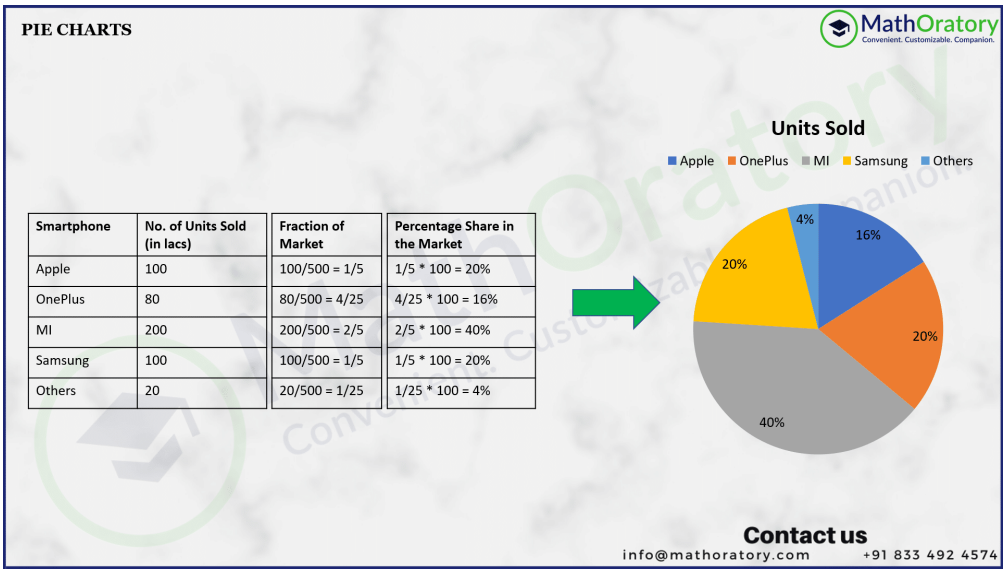
- `Box Plot` : A box plot is used to represent statistical data based on quartiles. For example, you might use a box plot to represent the distribution of daily energy production. The box represents the interquartile range (IQR), the line inside the box represents the median, and the whiskers represent the range of the data.



In [53]:
```
import matplotlib.pyplot as plt
import random

# Get data
daily_energy_production = [random.uniform(2000, 8000) for _ in range(365)]

# Create the box plot
plt.boxplot(daily_energy_production)
plt.title('Distribution of Daily Energy Production')
plt.ylabel('Energy Production (MWh)')
plt.show()
```

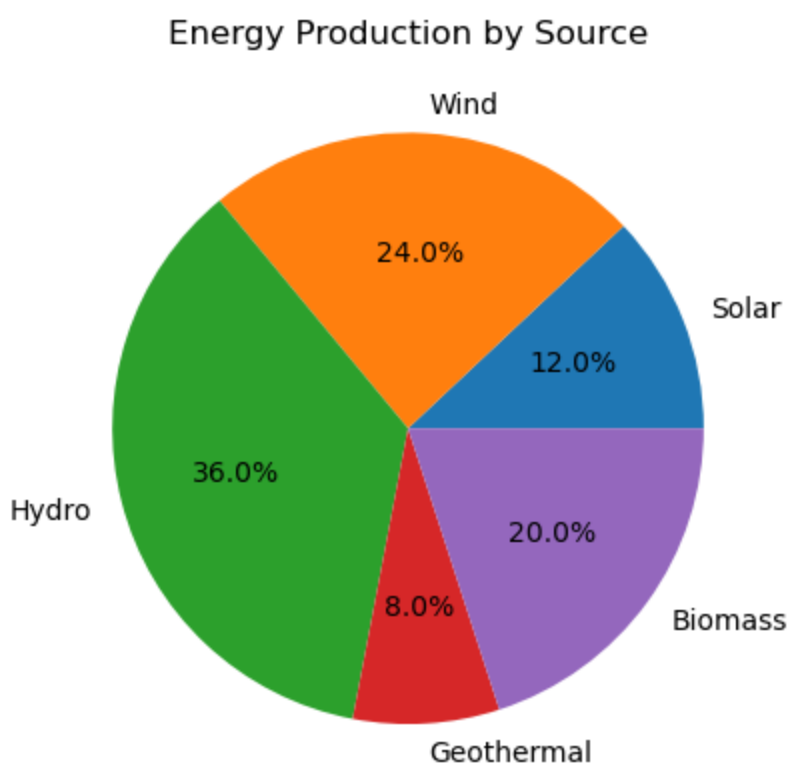


- `pie chart` : also known as a circle chart, is a circular statistical graphic which is divided into slices to illustrate numerical proportion



Smartphone	No. of Units Sold (in lacs)	Fraction of Market	Percentage Share in the Market
Apple	100	100/500 = 1/5	1/5 * 100 = 20%
OnePlus	80	80/500 = 4/25	4/25 * 100 = 16%
MI	200	200/500 = 2/5	2/5 * 100 = 40%
Samsung	100	100/500 = 1/5	1/5 * 100 = 20%
Others	20	20/500 = 1/25	1/25 * 100 = 4%

In [54]:
```
import matplotlib.pyplot as plt

# Get data
labels = ['Solar', 'Wind', 'Hydro', 'Geothermal', 'Biomass']
sizes = [15, 30, 45, 10, 25]

# Create the pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Energy Production by Source')
plt.show()
```



Basic Pattern to Create a Plot Using Matplotlib

Creating a plot in Matplotlib typically involves the following steps:

1- **Import the Matplotlib library**: This is usually done with the line import `matplotlib.pyplot as plt`.

2- **Prepare the data**: Define the data that you want to plot. This could be a list of `x-values` and a list of corresponding `y-values`, or it could be data in a more complex form like a `Pandas DataFrame`.

3- **Plot the data**: This could be a line plot, bar plot, scatter plot, etc., depending on what you want to create. You can use the `plot function for line plots`, the `scatter function for scatter plots`, etc.

4- **Customize the plot**: Add `labels`, `title`, `legend`, etc. You can use functions like `xlabel`, `ylabel`, and `title` to add labels and title to the plot.

In [ ]:
```
plt.title('Title')  # Add a title
plt.xlabel('x-label')  # Add x-label
plt.ylabel('y-label')  # Add y-label
```

5- **Set x and y axis limits**: You can use `xlim and ylim` functions to set the limits of x and y axis respectively.

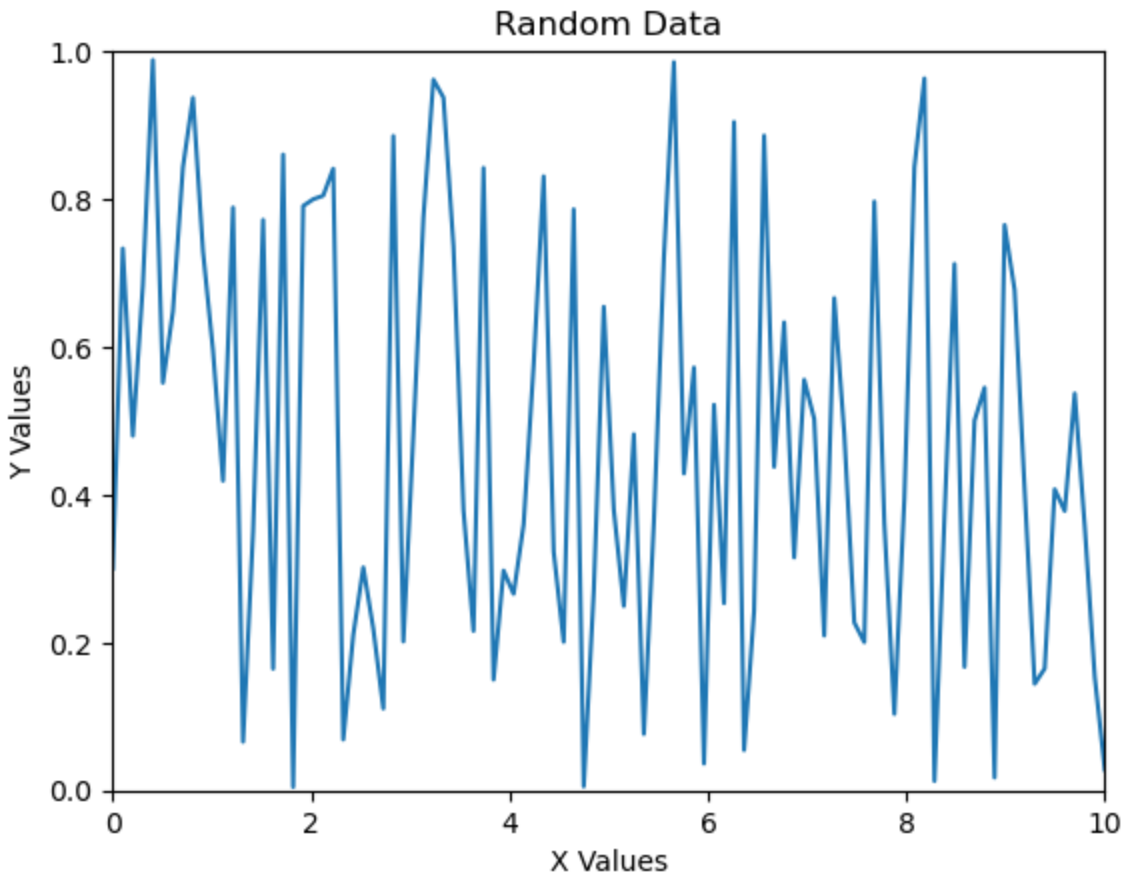
In [ ]:
```
plt.xlim(0, 10)  # Set x-axis limits
plt.ylim(0, 20)  # Set y-axis limits
```

6- **Display the plot**: Finally, you can use the `show function` to display the plot.

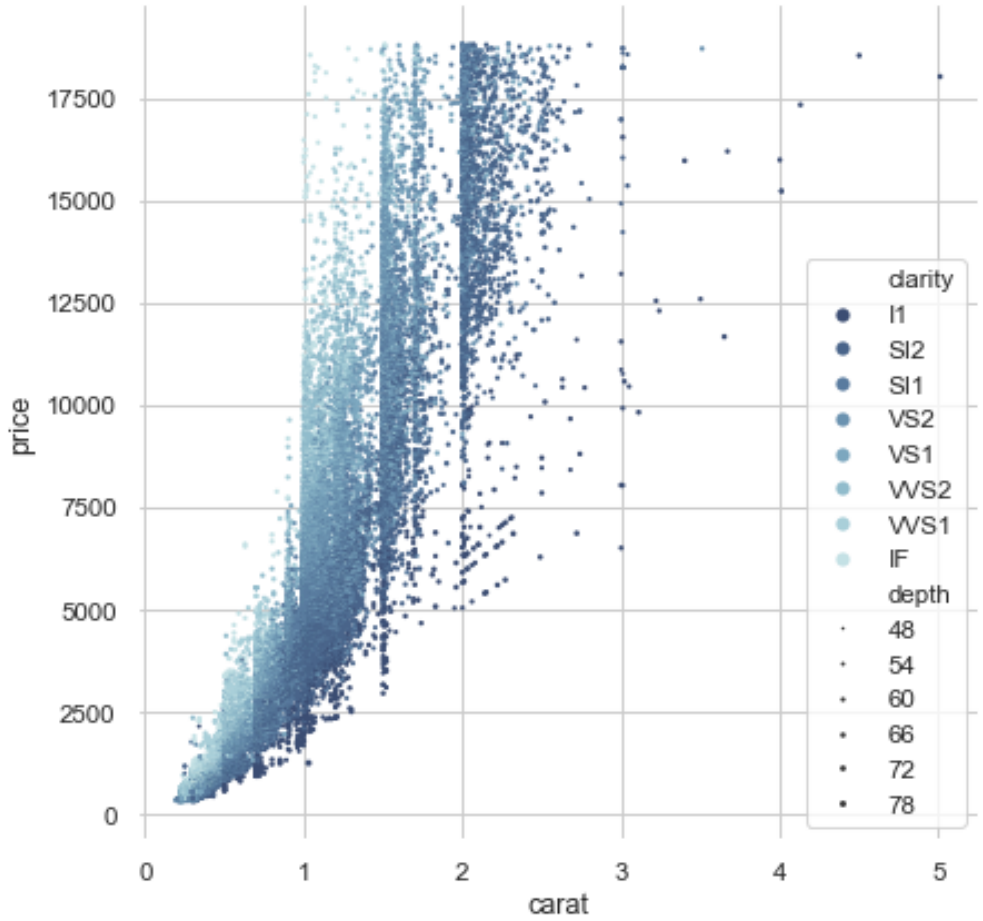
In [ ]:
```
plt.show()  # Display the plot
```

Summary

In [3]:
```
# Step 1: Import the Matplotlib library
import matplotlib.pyplot as plt
import numpy as np  # We'll use numpy to generate random data

# Step 2: Prepare the data
x = np.linspace(0, 10, 100)  # 100 evenly spaced numbers from 0 to 10
y = np.random.rand(100)  # 100 random numbers

# Step 3: Plot the data
plt.plot(x, y)  # Create a line plot of y vs x

# Step 4: Customize the plot
plt.title('Random Data')  # Add a title
plt.xlabel('X Values')  # Add label for x-axis
plt.ylabel('Y Values')  # Add label for y-axis

# Step 5: Set x and y axis limits
plt.xlim(0, 10)  # x-axis values will range from 0 to 10
plt.ylim(0, 1)  # y-axis values will range from 0 to 1

# Step 6: Display the plot
plt.show()  # Display the plot
```



Introduction to Seaborn



Why Use Seaborn Over Matplotlib ?

> While `Matplotlib is great for creating basic graphs` , Seaborn offers even `more functionality and creates more sophisticated and aesthetically pleasing visuals` . It also proves more efficient as it often requires `less code` to be written to achieve the same result. Seaborn provides more `attractive default color palettes and themes than Matplotlib` . It also provides convenient functions to plot `categorical variables and supports visualizing` regression models`.

Basic Plots Using Seaborn

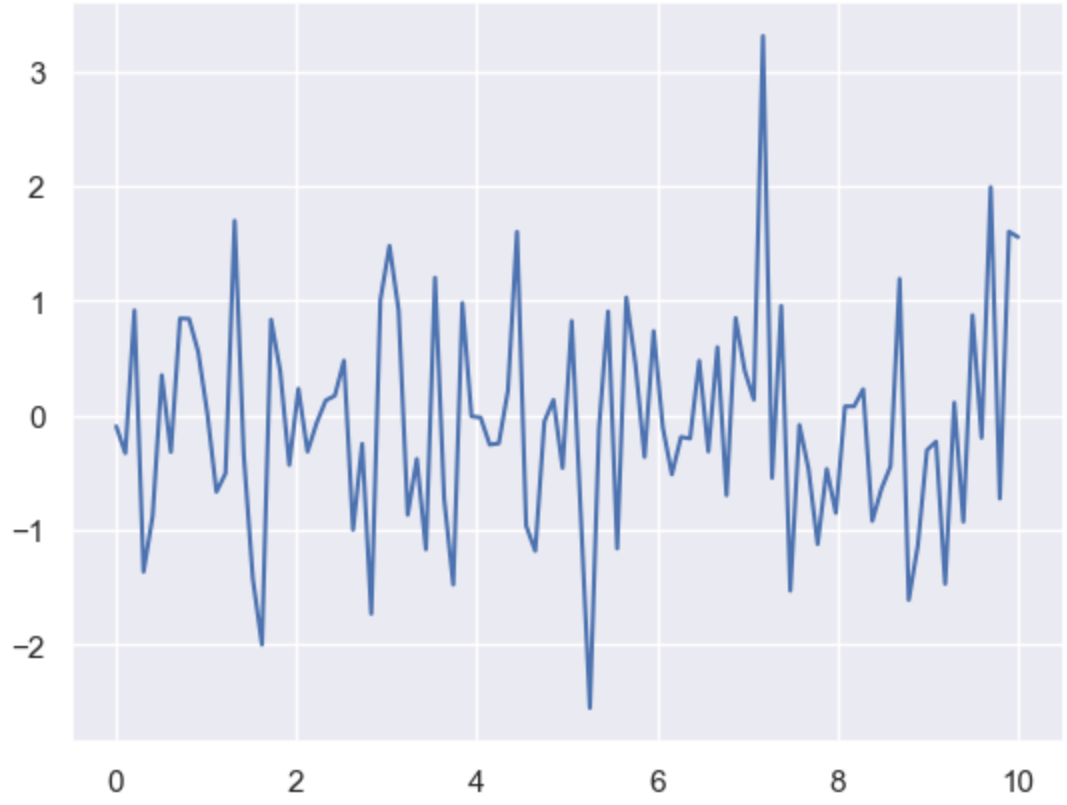
Let's start with the `sns.set()` function. This function is used to set the default parameters for the plots. `It affects all subsequent plots that you create` .

Here's an example of how to create a simple `line plot using Seaborn` :

In [5]:
```
# Import necessary libraries
import seaborn as sns
import numpy as np

# Set the default Seaborn style
sns.set()

# Generate some data
x = np.linspace(0, 10, 100)
y = np.random.normal(size=100)
```

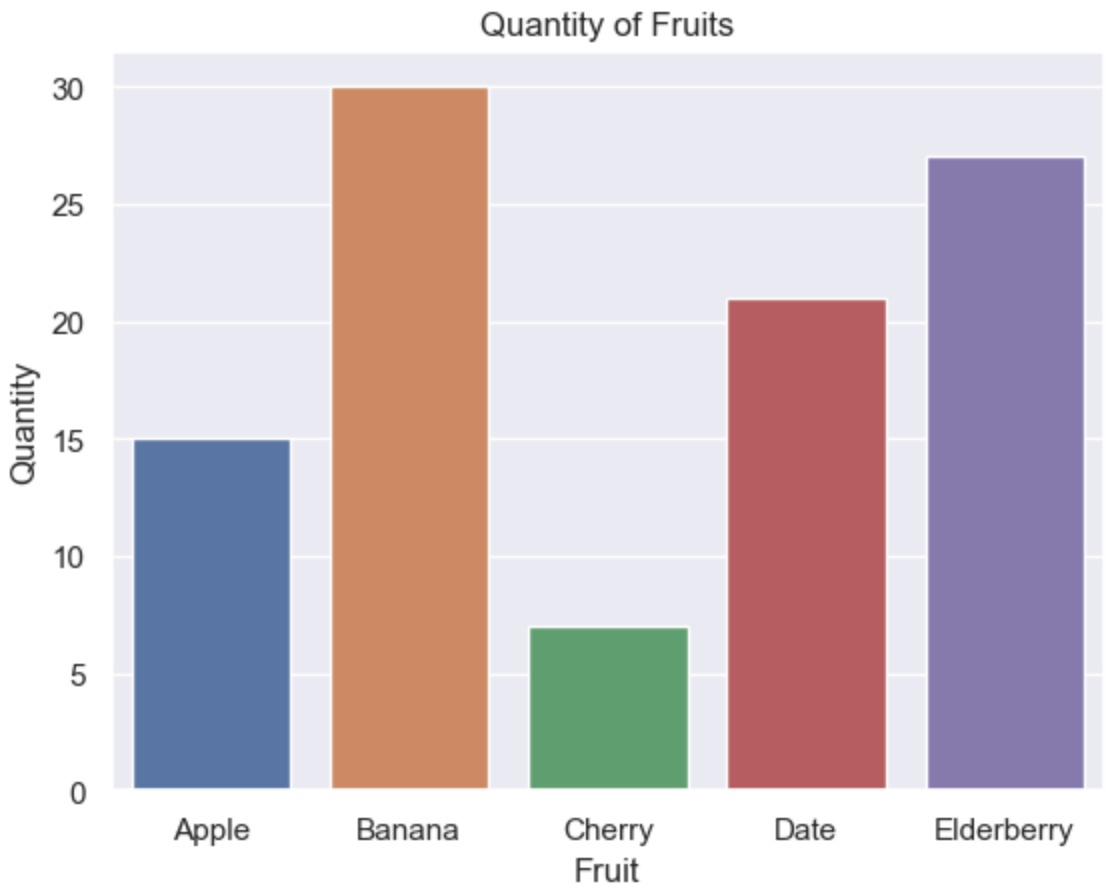
```
# Create a line plot
sns.lineplot(x=x, y=y)
```

Out[5]:
```
<Axes: >
```



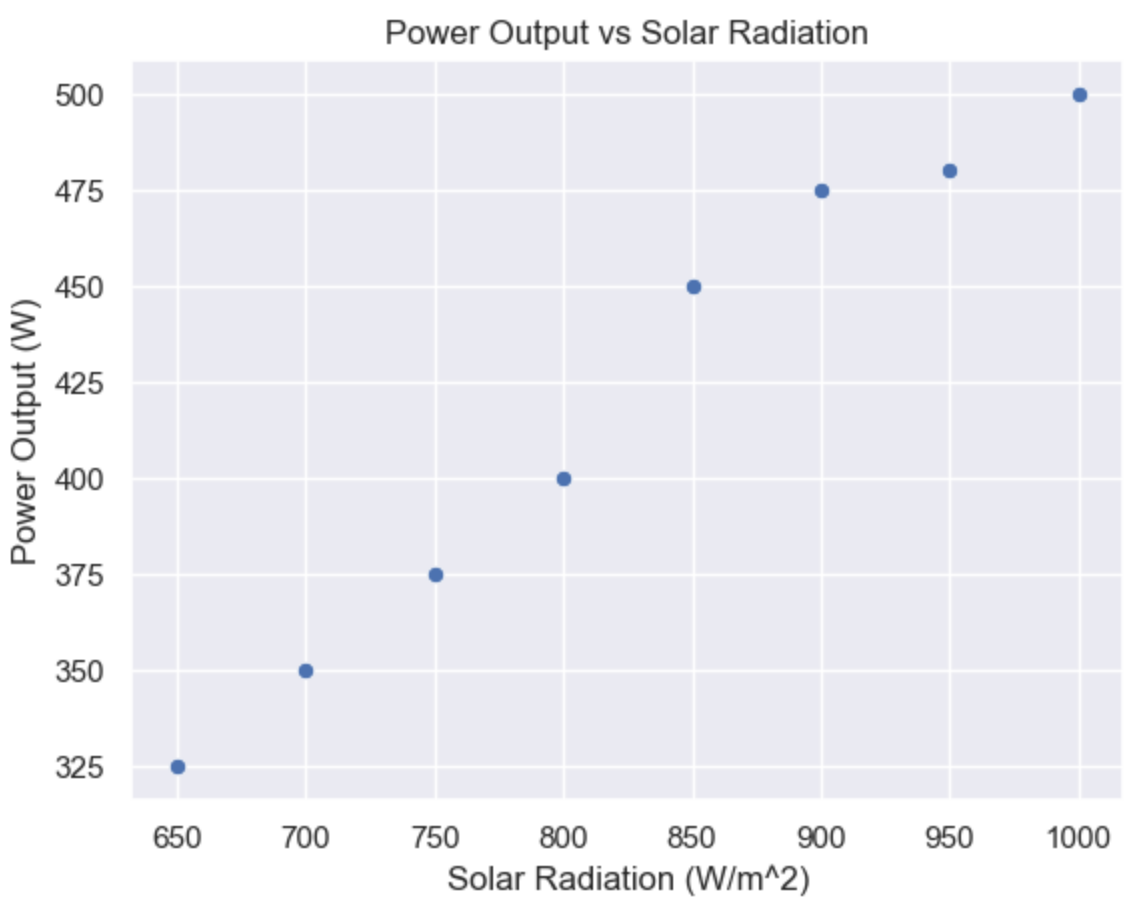
Bar Plot Using Seaborn

In [9]:
```
# Import necessary libraries
import seaborn as sns
import pandas as pd

# Set the default Seaborn style
sns.set()

# Prepare the data
data = {'Fruits': ['Apple', 'Banana', 'Cherry', 'Date', 'Elderberry'],
        'Quantity': [15, 30, 7, 21, 27]}
df = pd.DataFrame(data)

# Create a bar plot
sns.barplot(x='Fruits', y='Quantity', data=df)

# Add a title and labels
plt.title('Quantity of Fruits')
plt.xlabel('Fruit')
plt.ylabel('Quantity')

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



Scatter plot Using Seaborn

In [10]:
```
# Import necessary libraries
import seaborn as sns
import pandas as pd

# Set the default Seaborn style
sns.set()

# Prepare the data
data = {'Solar Radiation': [1000, 850, 900, 950, 800, 700, 750, 650],
        'Power Output': [500, 450, 475, 480, 400, 350, 375, 325]}
df = pd.DataFrame(data)

# Create a scatter plot
sns.scatterplot(x='Solar Radiation', y='Power Output', data=df)

# Add a title and labels
plt.title('Power Output vs Solar Radiation')
plt.xlabel('Solar Radiation (W/m^2)')
plt.ylabel('Power Output (W)')

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



Introduction to Plotly

Here are some reasons to consider using Plotly:

- `Interactive` : Plotly's graphs are `interactive` . Users can `zoom and hover over data points to see their values` , and much more.
- `Customizable` : Plotly offers users control over what is being plotted, simple syntax, and endless customization of graphs.
- `Attractive` : Plotly's graphs are visually appealing and can be accepted by a wide range of audiences.
- `Supports Multiple Languages` : Plotly is compatible with multiple languages including Python, R, MATLAB, Perl, Julia, Arduino, and REST.
- `Sharing` : Plotly allows users to share their plots online, making it easier to collaborate with others

In [ ]:
```python
import plotly.express as px

# Create a simple bar chart
fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2])

# Add title and axis labels
fig.update_layout(
    title={
        'text': "My Bar Chart",
        'y':0.95,
        'x':0.5,
        'xanchor': 'center',
```

```
        'yanchor': 'top'},
    xaxis_title="X Axis Title",
    yaxis_title="Y Axis Title",
)

# Display the plot
fig.show()
```