Machine Learning Module 1

February 13, 2023

1 Numpy, Pandas and Matplotlib Libraries

2 Machine Learning Module 1

3 Numpy

Numpy is a powerful library for mathematical computing and data analysis in Python. Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

3.1 Installation

Numpy can be installed by running the following command in your terminal:

[ ]:
```
pip install numpy
```

3.2 Importing Numpy

To use Numpy in your Python scripts, you will need to import it using the following statement:

[1]:
```
import numpy as np
```

3.3 Numpy Arrays

Numpy arrays are similar to Python lists, but are much faster and more convenient for mathematical operations. To create a Numpy array, you can use the `np.array` function.

[ ]:
```
# Creating a Numpy array
a = np.array([1, 2, 3, 4, 5])
print(a)
```

You can also create arrays with different data types, such as `float` or `complex`.

[3]:
```
# Creating a Numpy array with different data type
b = np.array([1.0, 2.0, 3.0, 4.0, 5.0], dtype=float)
c = np.array([1 + 2j, 2 + 3j, 3 + 4j, 4 + 5j, 5 + 6j], dtype=complex)
print(b)
print(c)
```

3.4 Array Shape and Size

You can obtain the shape and size of a Numpy array using the `shape` and `size` attributes, respectively.

[ ]:
```
# Shape and size of a Numpy array
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a.shape) # (2, 3)
print(a.size) # 6
```

3.5 Array Reshaping

You can change the shape of a Numpy array using the `reshape` method.

[ ]:
```
# Reshaping a Numpy array
a = np.array([1, 2, 3, 4, 5, 6])
b = a.reshape(2, 3)
print(b)
```

3.6 Array Slicing

You can slice Numpy arrays in the same way as Python lists.

[ ]:
```
# Slicing a Numpy array
a = np.array([1, 2, 3, 4, 5, 6])
print(a[1:4])
```

3.7 Array Indexing

You can access the elements of a Numpy array by indexing.

[ ]:
```
# Indexing a Numpy array
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a[0, 1]) # 2
```

3.8 Boolean Indexing

You can also use Boolean arrays to index Numpy arrays. A Boolean array is a Numpy array containing values of `True` and `False`.

[ ]:
```
# Boolean Indexing in Numpy
a = np.array([1, 2, 3, 4, 5])
b = a > 2
print(b) # [False, False, True, True, True]
print(a[b]) # [3, 4, 5]
```

3.9 Array Math Operations and Statistics

Numpy provides a number of functions for mathematical operations on arrays. Numpy provides a number of functions for calculating statistical measures such as mean, median, and standard

deviation.

[ ]:
```
# Array mathematical operations
a = np.array([1, 2, 3, 4, 5])
print(np.sum(a)) # 15
print(np.mean(a)) # 3.0
print(np.min(a)) # 1
print(np.max(a)) # 5
print(np.median(a)) # 3.0
print(np.std(a)) # 1.5811388300841898
```

3.10 Broadcasting

Numpy provides the ability to broadcast arrays, which allows you to perform element-wise operations between arrays of different shapes.

[ ]:
```
# Broadcasting in Numpy
a = np.array([1, 2, 3, 4, 5])
b = 10
print(a + b) # [11, 12, 13, 14, 15]
```

[ ]:
```
# Broadcasting in Numpy
a = np.array([[1, 2, 3], [4, 5, 6]])
b = 10
print(a + b)
```

3.11 Array Concatenation

You can concatenate Numpy arrays using the `np.concatenate` function.

[ ]:
```
# Concatenating Numpy arrays
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.concatenate((a, b))
print(c) # [1, 2, 3, 4, 5, 6]
```

3.12 Array Splitting

You can split Numpy arrays using the `np.split` function.

[7]:
```
# Splitting Numpy arrays
a = np.array([1, 2, 3, 4, 5, 6])
b = np.split(a, 3)
print(b) # [array([1, 2]), array([3, 4]), array([5, 6])]
```

[ ]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
b, c = np.hsplit(a, 2)
print(b)
print(c)
```

```
[ ]: a = np.array([[1, 2], [3, 4], [5, 6]])
     b, c = np.vsplit(a, 2)
     print(b)
     print(c)
```

3.13 Array Transposition

You can transpose Numpy arrays using the `T` attribute.

```
[ ]: # Transposing Numpy arrays
     a = np.array([[1, 2, 3], [4, 5, 6]])
     b = a.T
     print(b)
```

3.14 Array Element-wise Operations

You can perform element-wise operations between Numpy arrays.

```
[ ]: # Element-wise operations in Numpy
     a = np.array([1, 2, 3, 4, 5])
     b = np.array([10, 20, 30, 40, 50])
     print(a + b) # [11, 22, 33, 44, 55]
     print(a - b) # [-9, -18, -27, -36, -45]
     print(a * b) # [10, 40, 90, 160, 250]
     print(a / b) # [0.1, 0.1, 0.1, 0.1, 0.1]
```

3.15 Array Matrix Operations

You can perform matrix operations in Numpy using the `np.dot` function.

```
[ ]: # Matrix operations in Numpy
     a = np.array([[1, 2], [3, 4]])
     b = np.array([[10, 20], [30, 40]])
     print(np.dot(a, b))
```

3.16 Stacking Numpy arrays

```
[ ]: a = np.array([1, 2, 3])
     b = np.array([4, 5, 6])
     c = np.vstack((a, b))
     print(c)
```

```
[ ]: a = np.array([[1, 2], [3, 4]])
     b = np.array([[5, 6], [7, 8]])
     c = np.hstack((a, b))
     print(c)
```

3.17 Conclusion

In this tutorial, we have covered the basics of Numpy and its key features. Numpy is an essential tool for machine learning and data science, as it provides efficient support for arrays and matrices. By mastering Numpy, you will be able to perform mathematical operations and manipulate arrays with ease.

4 Pandas

Pandas is a powerful library for data analysis and manipulation. It is widely used for data preparation, feature engineering, and data exploration in machine learning and data science.

To start with Pandas, you need to import the library and create a `DataFrame`, which is the core data structure of Pandas.

[ ]:
```python
# Importing Pandas
import pandas as pd

# Creating a Pandas DataFrame
data = {"Name": ["John", "Jane", "Jim", "Joan"],
        "Age": [32, 28, 45, 50],
        "Occupation": ["Engineer", "Doctor", "Teacher", "Lawyer"]}
df = pd.DataFrame(data)
print(df)
```

4.1 DataFrame Operations

There are several operations that you can perform on a Pandas DataFrame. Some of the common operations are:

4.2 Selecting Rows and Columns

You can select rows and columns in a DataFrame using the `iloc` and `loc` attributes.

[ ]:
```python
# Selecting rows and columns using iloc
print(df.iloc[1]) # select second row
print(df.iloc[:, 1]) # select second column

# Selecting rows and columns using loc
print(df.loc[0]) # select first row
print(df.loc[:, "Age"]) # select "Age" column
```

4.3 Slicing

You can slice a DataFrame to select a subset of rows and columns.

[ ]:
```python
# Slicing a DataFrame
print(df[1:3]) # select second and third rows
print(df.loc[:, ["Name", "Age"]]) # select "Name" and "Age" columns
```

4.4 Filtering

You can filter a DataFrame based on a specific condition.

[ ]:
```
# Filtering a DataFrame
print(df[df["Age"] > 40]) # select rows where "Age" is greater than 40
```

4.5 Grouping

You can group a DataFrame based on a specific column and perform aggregate operations.

[ ]:
```
# Grouping a DataFrame
grouped = df.groupby("Occupation").mean()
print(grouped)
```

4.6 Merging

You can merge two DataFrames on a specific column.

[ ]:
```
# Merging two DataFrames
data1 = {"Name": ["John", "Jane", "Jim", "Joan"],
         "Age": [32, 28, 45, 50]}
df1 = pd.DataFrame(data1)

data2 = {"Name": ["John", "Jane", "Jim", "Joan"],
         "Occupation": ["Engineer", "Doctor", "Teacher", "Lawyer"]}
df2 = pd.DataFrame(data2)

merged = pd.merge(df1, df2, on="Name")
print(merged)
```

With this, you have a good understanding of intermediate level Pandas features and how to use them for machine learning and data science.

5 Matplotlib

Matplotlib is a powerful library for creating visualizations and plots. It is widely used for data exploration and visualizing the results of machine learning algorithms.

To start with Matplotlib, you need to import the library and use the `pyplot` module to create plots.

[12]:
```
# Importing Matplotlib
import matplotlib.pyplot as plt

# Creating a plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.show()
```

5.1 Line Plots

A line plot is a way to display the relationship between two variables.

[ ]:
```
# Line plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot")
plt.show()
```

5.2 Scatter Plots

A scatter plot is a way to display the relationship between two variables. It shows the individual data points rather than connected lines.

[ ]:
```
# Scatter plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.scatter(x, y)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Scatter Plot")
plt.show()
```

5.3 Bar Plots

A bar plot is a way to display categorical data.

[ ]:
```
# Bar plot
x = ["A", "B", "C", "D", "E"]
y = [2, 4, 6, 8, 10]
plt.bar(x, y)
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Plot")
plt.show()
```

5.4 Histograms

A histogram is a way to display the distribution of numerical data.

[ ]:
```
# Histogram
import numpy as np
data = np.random.normal(100, 20, 1000)
plt.hist(data, bins=20)
plt.xlabel("Values")
```

```
plt.ylabel("Frequency")
plt.title("Histogram")
plt.show()
```

5.5 Subplots

Subplots allow you to display multiple plots in a single figure.

[11]:
```
# Subplots
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.subplot(2, 1, 1)
plt.plot(x, y)
plt.title("Line Plot")

plt.subplot(2, 1, 2)
plt.scatter(x, y)
plt.title("Scatter Plot")

plt.tight_layout()
plt.show()
```

5.6 Customizing Plots

Matplotlib provides a wide range of customization options to decorate plots. Some of the most commonly used methods are:

5.7 Adding Legends

[ ]:
```
x = [1, 2, 3, 4, 5]
y1 = [2, 4, 6, 8, 10]
y2 = [3, 6, 9, 12, 15]

plt.plot(x, y1, label="Line 1")
plt.plot(x, y2, label="Line 2")

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot with Legends")
plt.legend()
plt.show()
```

5.8 Changing Line Style and Color

[ ]:
```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y, color="red", linestyle="dashed", linewidth=2)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot with Custom Style and Color")
plt.show()
```

5.9 Adding Markers

[ ]:
```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y, marker="o", markersize=10)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot with Markers")
plt.show()
```

5.10 Changing Axis Limits

[ ]:
```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot with Custom Axis Limits")
plt.xlim(0, 6)
plt.ylim(0, 12)
plt.show()
```

With this, you have a good understanding of Matplotlib features and how to use them for data exploration and visualizing results of machine learning algorithms.

6 Introduction to Seaborn

Seaborn is a data visualization library in Python that is built on top of Matplotlib. It provides a high-level interface for creating visualizations with attractive, informative graphics. Seaborn is particularly well-suited for plotting statistical graphics and is used extensively in data science and machine learning.

6.1 Installing Seaborn

To install Seaborn, you can use the pip package manager. Simply run the following command in your terminal:

[ ]:
```
pip install seaborn
```

6.2 Importing Seaborn

To use Seaborn in your code, you first need to import it. The standard import statement for Seaborn is as follows:

[ ]:
```
import seaborn as sns
```

6.3 Plotting with Seaborn

Seaborn provides several functions for creating different types of plots. Some of the most commonly used functions are:

- sns.distplot() for plotting histograms and kernel density plots
- sns.boxplot() for plotting box plots
- sns.violinplot() for plotting violin plots
- sns.scatterplot() for plotting scatter plots
- sns.lineplot() for plotting line plots

6.4 Scatterplot

A scatterplot is used to visualize the relationship between two variables. In Seaborn, you can create a scatterplot using the `sns.scatterplot` function. For example:

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

# Load the iris dataset
iris = sns.load_dataset("iris")

# Create a scatterplot
sns.scatterplot(x="sepal_length", y="sepal_width", data=iris)

# Add labels and title to the plot
plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Petal Length vs Petal Width for Different Species of Iris")

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

6.5 Lineplot

A lineplot is used to visualize the relationship between two variables over time. In Seaborn, you can create a lineplot using the `sns.lineplot` function. For example:

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

# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a line plot of total bill amount over time
sns.lineplot(x="total_bill", y="time", data=tips)

# Add labels and title to the plot
plt.xlabel("Total Bill Amount")
plt.ylabel("Time")
plt.title("Total Bill Amount Over Time")

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

6.6 Barplot

A barplot is a way to visualize the distribution of a categorical variable. In Seaborn, the barplot can be created using the `barplot()` function. The barplot takes in the x-axis variable, y-axis variable, and the data.

Example:

[17]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a bar plot of average total bill amount by day of the week
sns.barplot(x="day", y="total_bill", data=tips)

# Add labels and title to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Average Total Bill Amount")
plt.title("Average Total Bill Amount by Day of the Week")

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

6.7 Histogram

A histogram is a representation of the distribution of a set of continuous data. It is an estimate of the probability distribution of a continuous variable. In Seaborn, we can create a histogram using the `distplot` function.

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

# Load the example tips dataset
tips = sns.load_dataset("tips")

# Create a histogram of Total Bill amounts
sns.distplot(tips["total_bill"], kde=False)

# Add labels and title to the plot
plt.xlabel("Total Bill (USD)")
plt.ylabel("Frequency")
plt.title("Histogram of Total Bill Amounts in Tips Dataset")

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

6.8 Boxplot

A box plot, also known as a box-and-whisker plot, is a standardized way of visualizing the distribution of a dataset, based on five number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum). In Seaborn, we can create a box plot using the `boxplot` function.

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

# Load the example tips dataset
tips = sns.load_dataset("tips")

# Create a box plot of Total Bill amounts by day of the week
sns.boxplot(x="day", y="total_bill", data=tips)

# Add labels and title to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill (USD)")
plt.title("Box Plot of Total Bill Amounts by Day of the Week in Tips Dataset")

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

6.9 Violinplot

A violin plot is a combination of a box plot and a kernel density plot. It shows the distribution of the data and its probability density, similar to a histogram. In Seaborn, we can create a violin plot using the `violinplot` function.

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

# Load the example tips dataset
tips = sns.load_dataset("tips")

# Create a violin plot of Total Bill amounts by day of the week
sns.violinplot(x="day", y="total_bill", data=tips)

# Add labels and title to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Total Bill (USD)")
plt.title("Violin Plot of Total Bill Amounts by Day of the Week in Tips␣
 ↪Dataset")

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

6.10 Swarmplot

A Swarmplot is a combination of a scatter plot and a violin plot, where the scatter plot is overlaid on top of the violin plot. The swarmplot can be used to visualize the distribution of a numerical variable across different categories.

Here is an example of a Swarmplot in Seaborn:

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

# Load the iris dataset
iris = sns.load_dataset("iris")

# Create a Swarmplot of Petal Length vs Species
sns.swarmplot(x="species", y="petal_length", data=iris)

# Add labels and title to the plot
plt.xlabel("Species")
plt.ylabel("Petal Length (cm)")
plt.title("Petal Length by Species")

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

6.11 Countplot

A Countplot is a bar plot that displays the frequency of each category in a categorical variable. In Seaborn, the `countplot` function can be used to create countplots.

Here is an example of a Countplot in Seaborn:

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

# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a Countplot of Day of the Week
sns.countplot(x="day", data=tips)

# Add labels and title to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Frequency")
plt.title("Frequency of Tips by Day of the Week")

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

6.12 Factorplot

A Factorplot is a multi-plot figure that allows you to display multiple plots with the same x and y axis across different facets. In Seaborn, the `factorplot` function can be used to create factorplots.

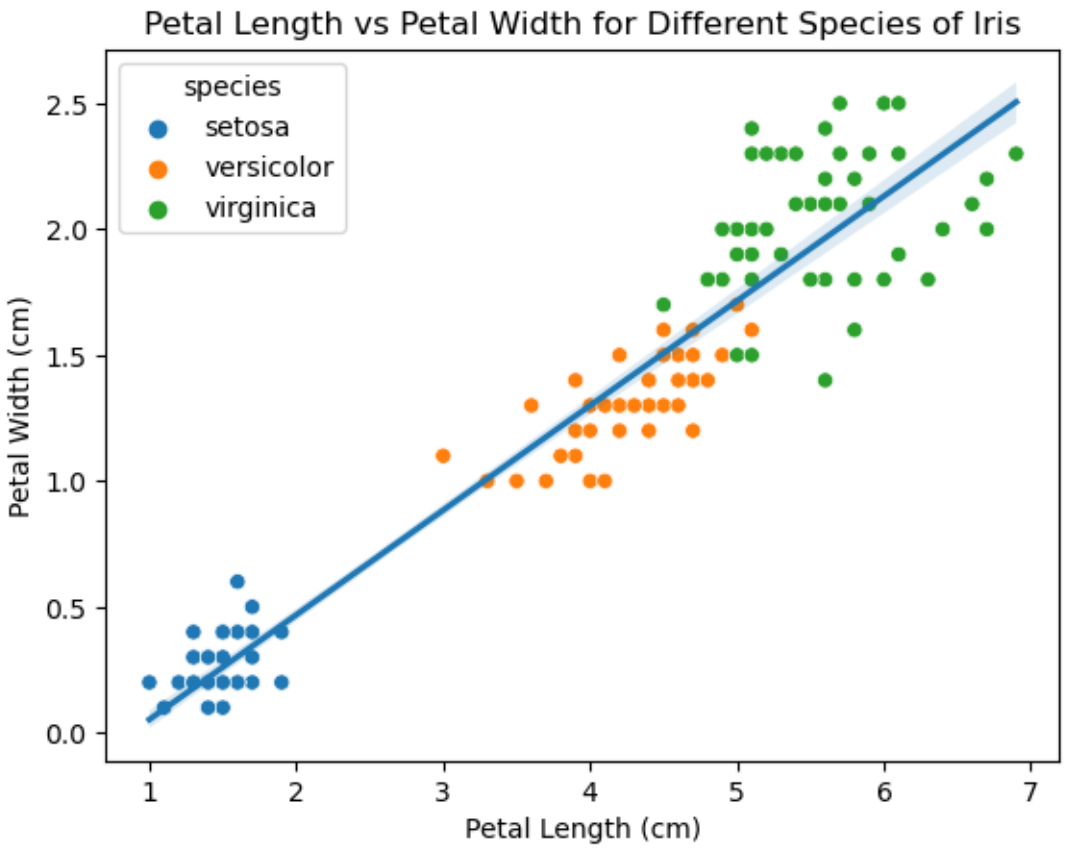
Here is an example of a Factorplot in Seaborn:

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

# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a Factorplot of the Distribution of Tips by Day of the Week and Time␣
 ↪of Day
sns.factorplot(x="day", y="tip", col="time", data=tips, kind="swarm")

# Add labels and title to the plot
plt.xlabel("Day of the Week")
plt.ylabel("Tip Amount (USD)")
plt.title("Distribution of Tips by Day of the Week and Time of Day")

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

In these examples, we used the load_dataset function from Seaborn to load publically

available datasets. You can also use your own data to create these plots. Seaborn provides a high-level interface for creating plots, which allows you to create complex visualizations with just a few lines of code.

[14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the example iris dataset
iris = sns.load_dataset("iris")

# Create a scatter plot of Petal Length vs Petal Width
sns.scatterplot(x="petal_length", y="petal_width", hue="species", data=iris)

# Add a regression line to the plot
sns.regplot(x="petal_length", y="petal_width", data=iris, scatter=False)

# Add labels and title to the plot
plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Petal Length vs Petal Width for Different Species of Iris")

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



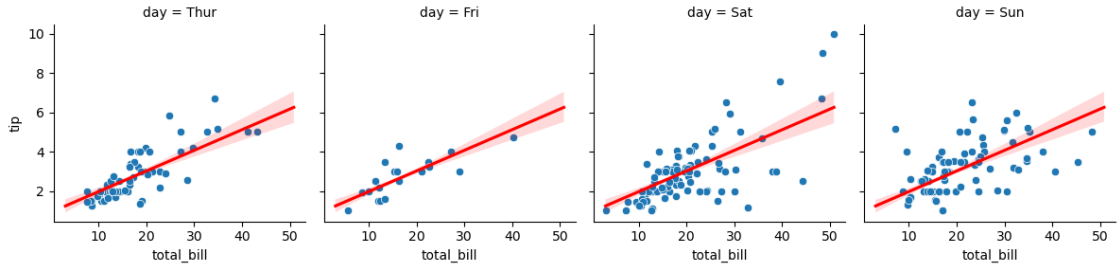
The code is a Python script that uses the `seaborn` and `matplotlib` libraries to create a scatter plot of the `petal_length` and `petal_width` features for different species of iris, with a regression line overlaid.

Here's a detailed explanation of the code:

1. Import the libraries: The first two lines of the code import the `seaborn` and `matplotlib` libraries into the script.
2. Load the iris dataset: The line `iris = sns.load_dataset("iris")` loads the iris dataset into the `iris` variable. The `iris` dataset is a commonly used dataset in machine learning and contains measurements of sepal length, sepal width, petal length, and petal width for 150 iris flowers, grouped into 3 species.
3. Create a scatter plot: The line `sns.scatterplot(x="petal_length", y="petal_width", hue="species", data=iris)` creates a scatter plot of the `petal_length` and `petal_width` features, using the `species` column to color the points based on their species. The `data` argument specifies the DataFrame containing the data to be plotted.
4. Add a regression line: The line `sns.regplot(x="petal_length", y="petal_width", data=iris, scatter=False)` adds a linear regression line to the plot. The `scatter` argument is set to `False` to prevent the creation of a scatter plot, since one already exists.
5. Add labels and title: The next three lines add labels and a title to the plot. `plt.xlabel("Petal Length (cm)")` sets the label for the x-axis, `plt.ylabel("Petal Width (cm)")` sets the label for the y-axis, and `plt.title("Petal Length vs Petal Width for Different Species of Iris")` sets the title for the plot.
6. Show the plot: The final line `plt.show()` displays the plot.

[23]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the example tips dataset from seaborn
tips = sns.load_dataset("tips")

# Create a facetgrid to plot the relationship between total bill and tip
g = sns.FacetGrid(tips, col="day")

# Add a scatter plot of total bill vs tip to each subplot
g.map(sns.scatterplot, "total_bill", "tip")

# Add a regression line to each scatter plot
for ax in g.axes.flat:
    sns.regplot(x="total_bill", y="tip", data=tips, scatter=False, color="r",␣
 ↪ax=ax)
```

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



6.13 What is regression line?

A regression line is a line of best fit that is used to represent the relationship between two continuous variables. In other words, it is a line that is used to estimate the relationship between the independent variable (x) and the dependent variable (y). The regression line is determined by using statistical methods to minimize the difference between the observed values of the dependent variable and the values predicted by the line. The regression line is typically represented by an equation that can be used to predict the value of the dependent variable for a given value of the independent variable. It is a common tool used in machine learning and data analysis to make predictions and understand the relationship between different variables in a dataset.

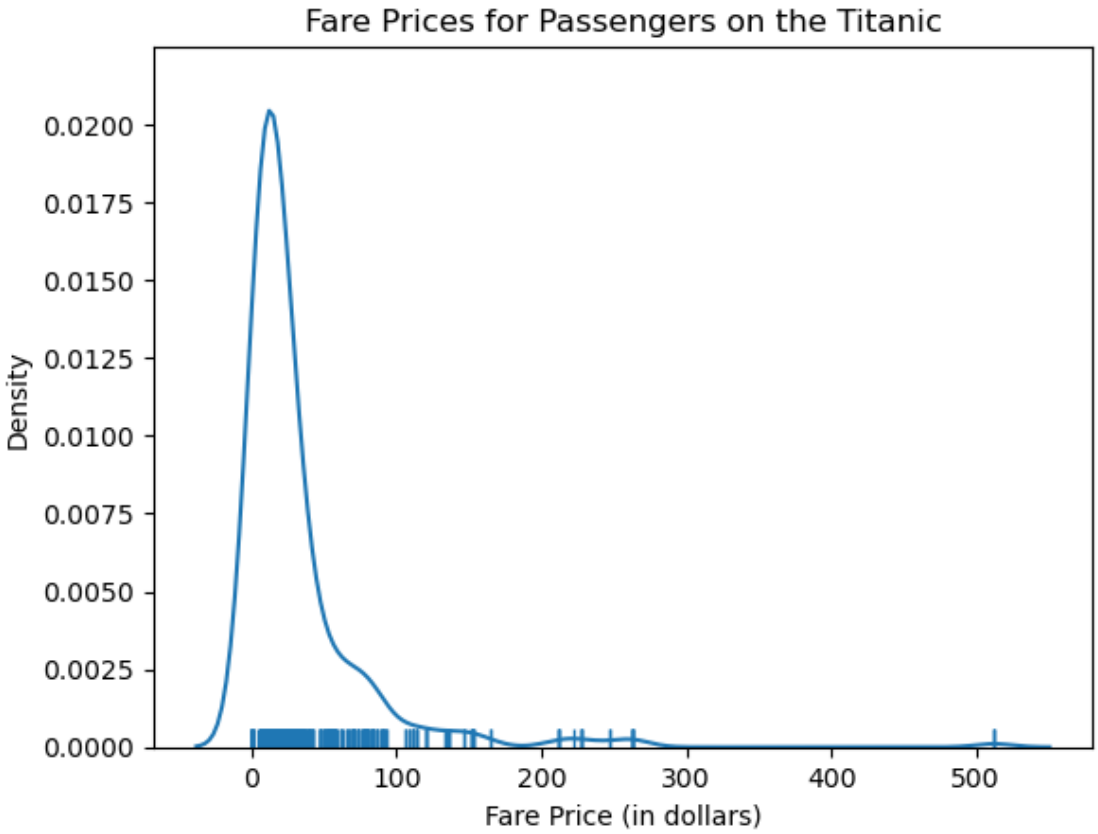
A linear regression line is represented by the equation:

`y = 0 + 1x`

where `y` is the dependent variable, `x` is the independent variable, `0` is the y-intercept, and `1` is the slope of the line.

[27]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Titanic dataset
titanic = sns.load_dataset("titanic")

# Plot a histogram of fare prices, with a rug plot and a kde plot
sns.distplot(titanic["fare"], rug=True, kde=True, hist=False)

# Add labels and title to the plot
plt.xlabel("Fare Price (in dollars)")
plt.title("Fare Prices for Passengers on the Titanic")

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

```
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `kdeplot` (an axes-level function for
kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\pc\anaconda3\lib\site-packages\seaborn\distributions.py:2103:
FutureWarning: The `axis` variable is no longer used and will be removed.
Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```



6.14 Exercises

6.14.1 Numpy

1. Create a 5x5 array with values ranging from 0 to 24.
2. Find the sum, mean, and standard deviation of a given array. arr = np.array([1,2,3,4,5])
3. Create a 2D array and extract the diagonal elements.
4. Convert a given 1D array into a 2D array using reshape. arr = np.array([1,2,3,4,5,6,7,8,9])
5. Create a 8x8 array and fill it with a chessboard pattern.
6. Stack two arrays vertically and horizontally. arr1 = np.array([1,2,3]) arr2 = np.array([4,5,6])

7. Create a 5x5 identity matrix.
8. Get the unique elements of a given array. arr = np.array([1,2,3,2,1,4,5,6,6,5,4])
9. Find the set difference between two arrays. arr1 = np.array([1,2,3,4,5,6,7,8,9]) arr2 = np.array([4,5,6,7,8,9,10,11,12])
10. Convert a given array into a one-hot encoding representation.

6.14.2 Pandas

1. Create a pandas DataFrame with the following data: `data = {'Name': ['John', 'Jane', 'Jim', 'Jill', 'Jack'],'Age': [30, 32, 28, 35, 27],'City': ['New York', 'San Francisco', 'Los Angeles', 'Boston', 'Chicago']}`
   - a. Print the DataFrame.
   - b. Sort the DataFrame by age.
   - c. Select only the rows where the age is greater than 30.
   - d. Select only the rows where the city is 'New York' or 'Boston'.
   - e. Group the data by city and find the mean age for each city.
2. Load the Titanic dataset (available in seaborn library) into a pandas DataFrame. Analyze the dataset and answer the following questions:
   - a. How many passengers were onboard?
   - b. How many columns are there in the dataset?
   - c. What are the names of the columns?
   - d. What is the average age of the passengers?
   - e. How many passengers survived?
   - f. What is the survival rate of the passengers?
3. Load a large dataset of your choice into a pandas DataFrame. Analyze the dataset and answer the following questions:
   - a. How many rows and columns are there in the dataset?
   - b. What are the names of the columns?
   - c. What is the mean, median, and standard deviation of each column?
   - d. How many missing values are there in the dataset?
   - e. Remove any columns with more than 50% missing values.
   - f. Create a histogram of each column.

6.14.3 Matplotlib

1. Create a basic line plot of the Sine function.
2. Plot the result of the equation y = x^2 for the range of x from 0 to 10.
3. Create a scatter plot of randomly generated data.

4. Plot a histogram of a normally distributed random dataset with mean 0 and standard deviation of 1.
5. Create a bar plot to show the count of unique elements in a given list.

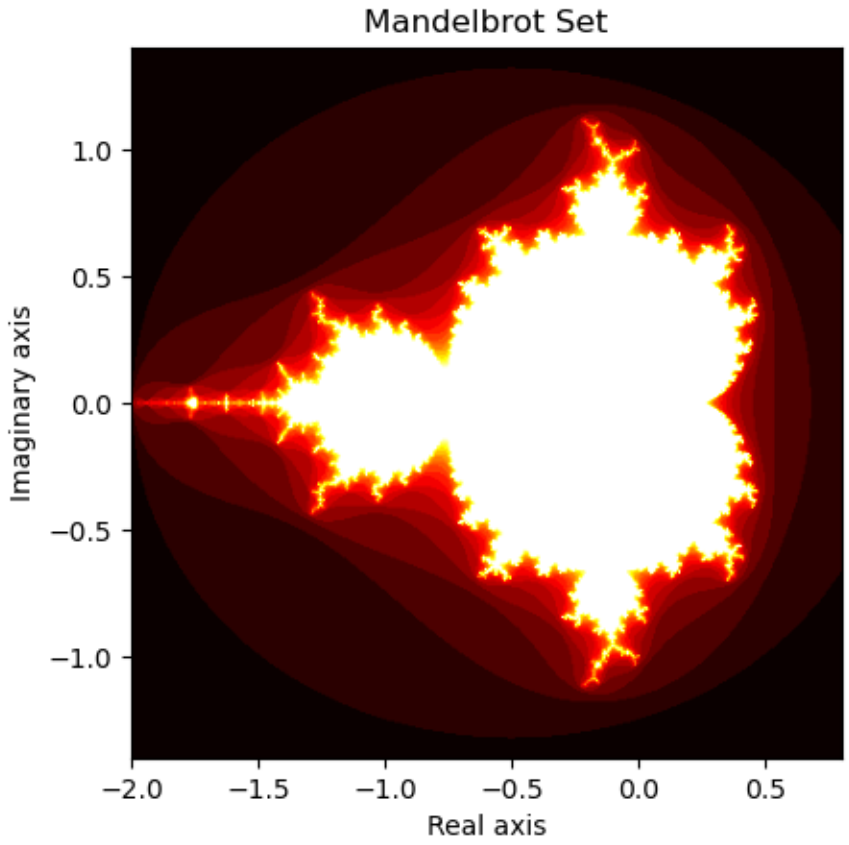
6.14.4 Seaborn

Here are a few exercises you can try with Seaborn:

1. Load the built-in "titanic" dataset and create a barplot showing the number of passengers who survived and those who did not.
2. Load the built-in "tips" dataset and create a scatterplot showing the relationship between total bill amount and tip amount. Add a regression line to the plot.
3. Load the built-in "flights" dataset and create a lineplot showing the average number of passengers for each month.
4. Load the built-in "iris" dataset and create a boxplot showing the distribution of petal lengths for each species of iris.
5. Load the built-in "exercise" dataset and create a violinplot showing the distribution of heart rate for each type of exercise.
6. Load the built-in "titanic" dataset and create a countplot showing the number of passengers who boarded from each class (1st, 2nd, 3rd).
7. Load the built-in "titanic" dataset and create a factorplot showing the relationship between passenger class and the fare they paid.
8. Load the built-in "titanic" dataset and create a histogram showing the distribution of passenger ages.

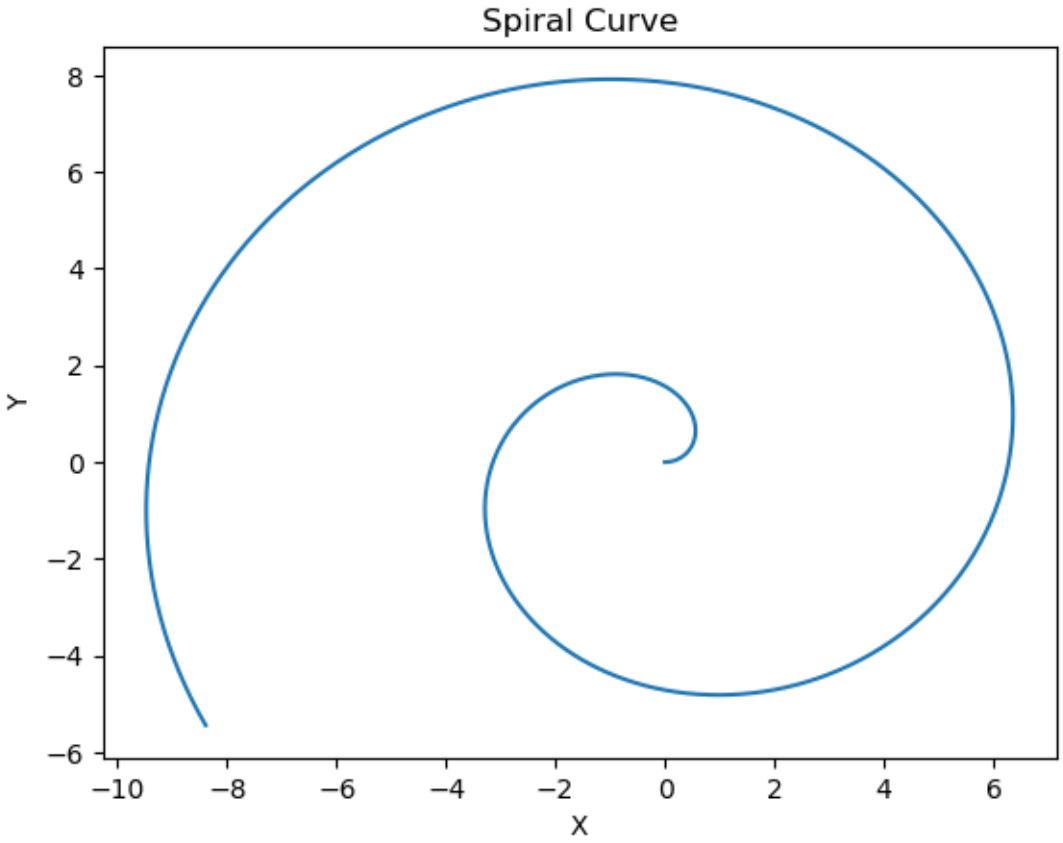
[29]:
```
import numpy as np
import matplotlib.pyplot as plt

def mandelbrot(h, w, maxit=20):
    """Returns an image of the Mandelbrot fractal of size (h,w)"""
    y, x = np.ogrid[-1.4:1.4:h*1j, -2:0.8:w*1j]
    c = x + y * 1j
    z = c
    divtime = maxit + np.zeros(z.shape, dtype=int)

    for i in range(maxit):
        z = z**2 + c
        diverge = z * np.conj(z) > 2**2          # who is diverging
        div_now = diverge & (divtime==maxit)  # who is diverging now
        divtime[div_now] = i                  # note when
        z[diverge] = 2                        # avoid diverging too much

    return divtime

def plot_mandelbrot(h, w, maxit=20):
```

```
    """Plots the Mandelbrot fractal of size (h,w)"""
    d = mandelbrot(h, w, maxit)
    plt.imshow(d, cmap='hot', extent=[-2, 0.8, -1.4, 1.4])
    plt.xlabel("Real axis")
    plt.ylabel("Imaginary axis")
    plt.title("Mandelbrot Set")
    plt.show()

plot_mandelbrot(400, 400)
```

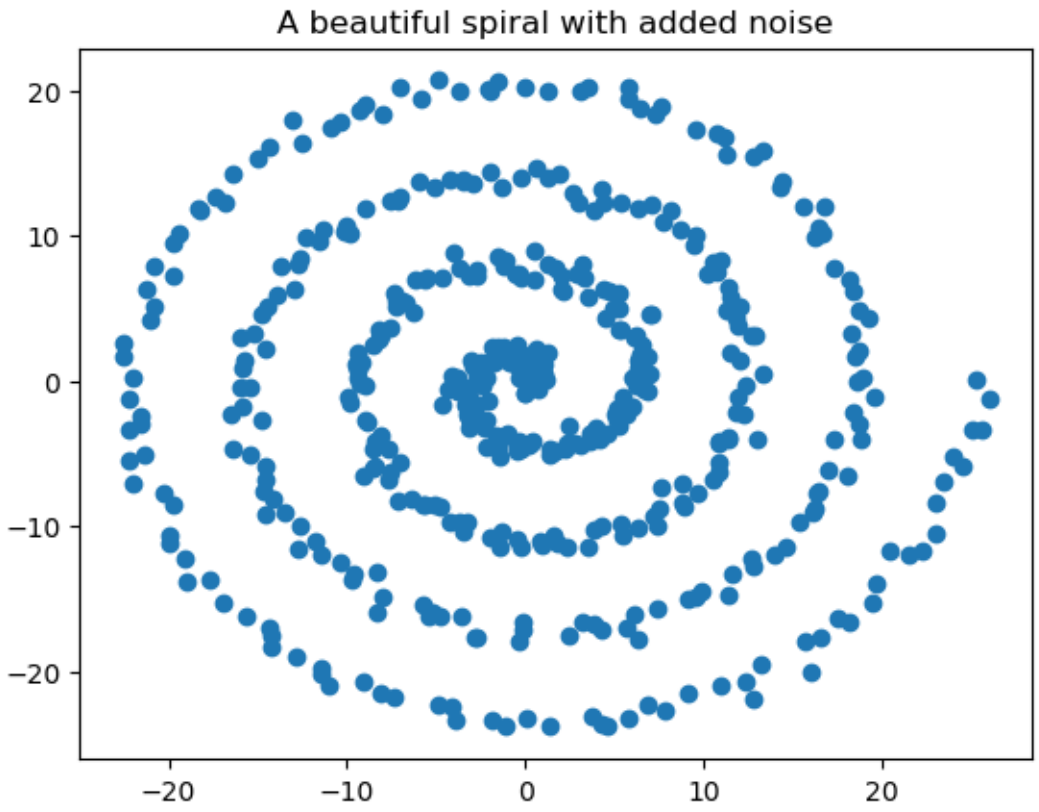


[30]:
```
import numpy as np
import matplotlib.pyplot as plt

def plot_spiral(n_points=1000, start=0, end=10):
    theta = np.linspace(start, end, n_points)
    x = theta * np.cos(theta)
    y = theta * np.sin(theta)
    fig, ax = plt.subplots()
    ax.plot(x, y)
    ax.set_title("Spiral Curve")
```

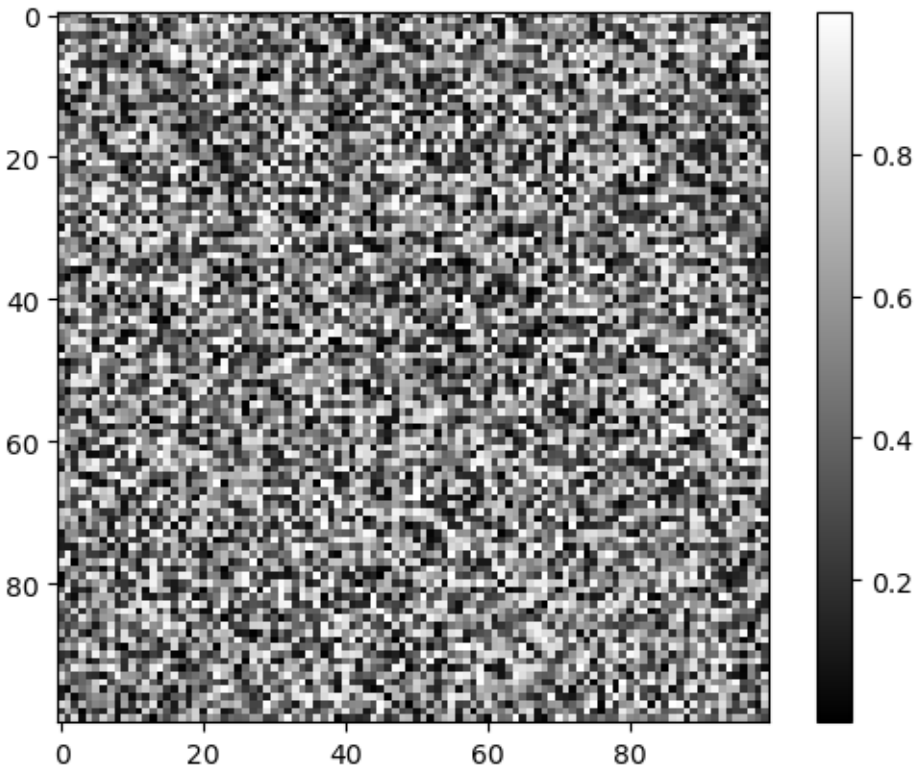
```
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    plt.show()

plot_spiral()
```



[31]:
```
import numpy as np
import matplotlib.pyplot as plt

def spiral(n_points, noise=0.5):
    theta = np.linspace(0, 8 * np.pi, n_points)
    x = theta * np.cos(theta) + np.random.normal(0, noise, n_points)
    y = theta * np.sin(theta) + np.random.normal(0, noise, n_points)

    return x, y

x, y = spiral(500)
plt.scatter(x, y)
plt.title("A beautiful spiral with added noise")
plt.show()
```

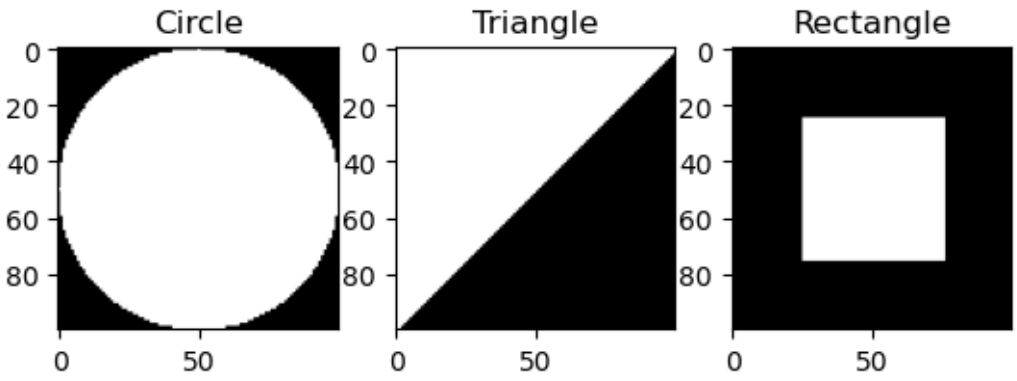


[32]:
```
import numpy as np
import matplotlib.pyplot as plt

# Creating a 2D array of shape (100, 100) with random values
array = np.random.rand(100, 100)

# Plotting the array as an image using imshow()
plt.imshow(array, cmap='gray')

# Adding color bar to the plot
plt.colorbar()

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

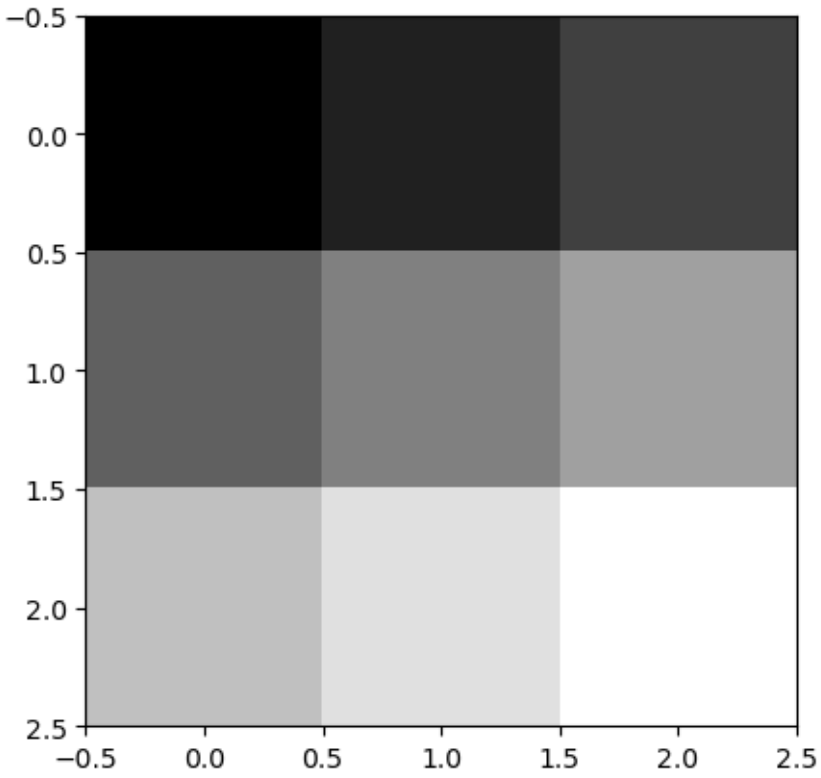


[33]:
```
import numpy as np
import matplotlib.pyplot as plt

# Generate arrays to display different shapes
circle = np.zeros((100,100))
for i in range(100):
    for j in range(100):
        if (i-50)**2 + (j-50)**2 <= 50**2:
            circle[i][j] = 1

triangle = np.zeros((100,100))
for i in range(100):
    for j in range(100):
        if i+j <= 100:
            triangle[i][j] = 1

rectangle = np.zeros((100,100))
for i in range(100):
    for j in range(100):
        if 25 <= i <= 75 and 25 <= j <= 75:
            rectangle[i][j] = 1
```

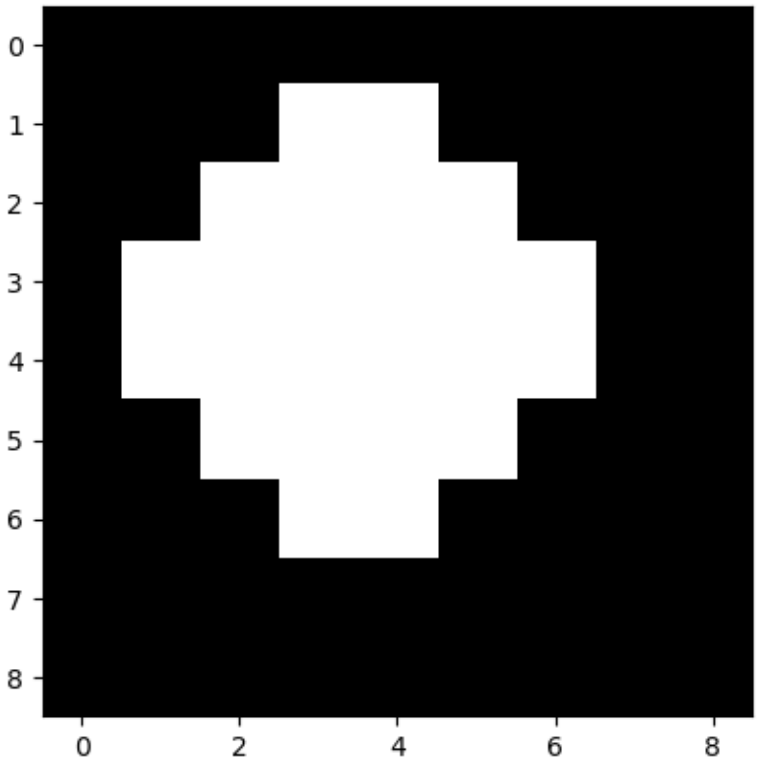
```
# Plot the arrays as images
plt.subplot(1, 3, 1)
plt.imshow(circle, cmap='gray')
plt.title('Circle')

plt.subplot(1, 3, 2)
plt.imshow(triangle, cmap='gray')
plt.title('Triangle')

plt.subplot(1, 3, 3)
plt.imshow(rectangle, cmap='gray')
plt.title('Rectangle')

plt.show()
```



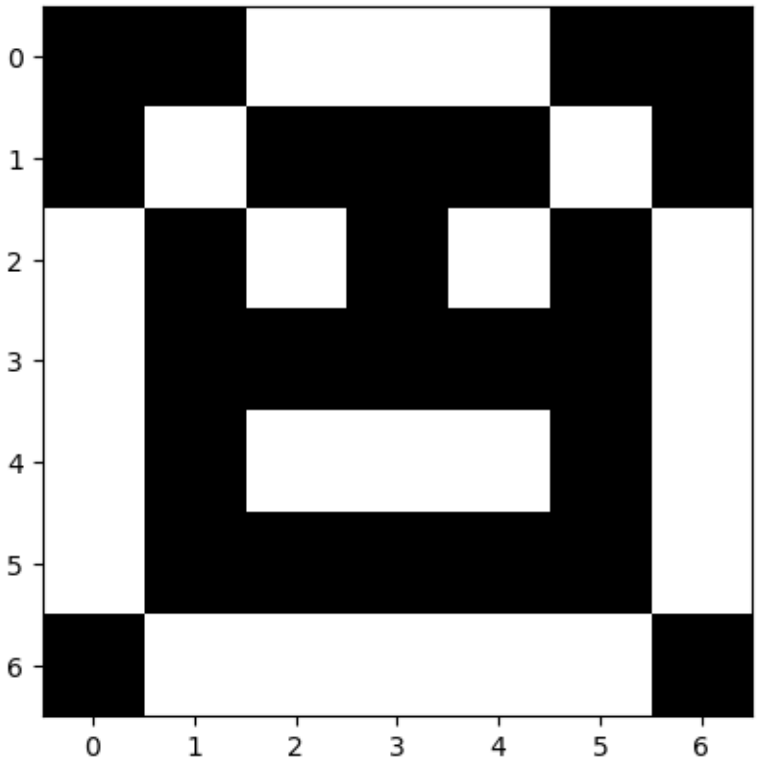
[34]:
```
import numpy as np
import matplotlib.pyplot as plt

# Create a numpy array without using any generation function
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Convert the array to image
plt.imshow(array, cmap='gray')
plt.show()
```



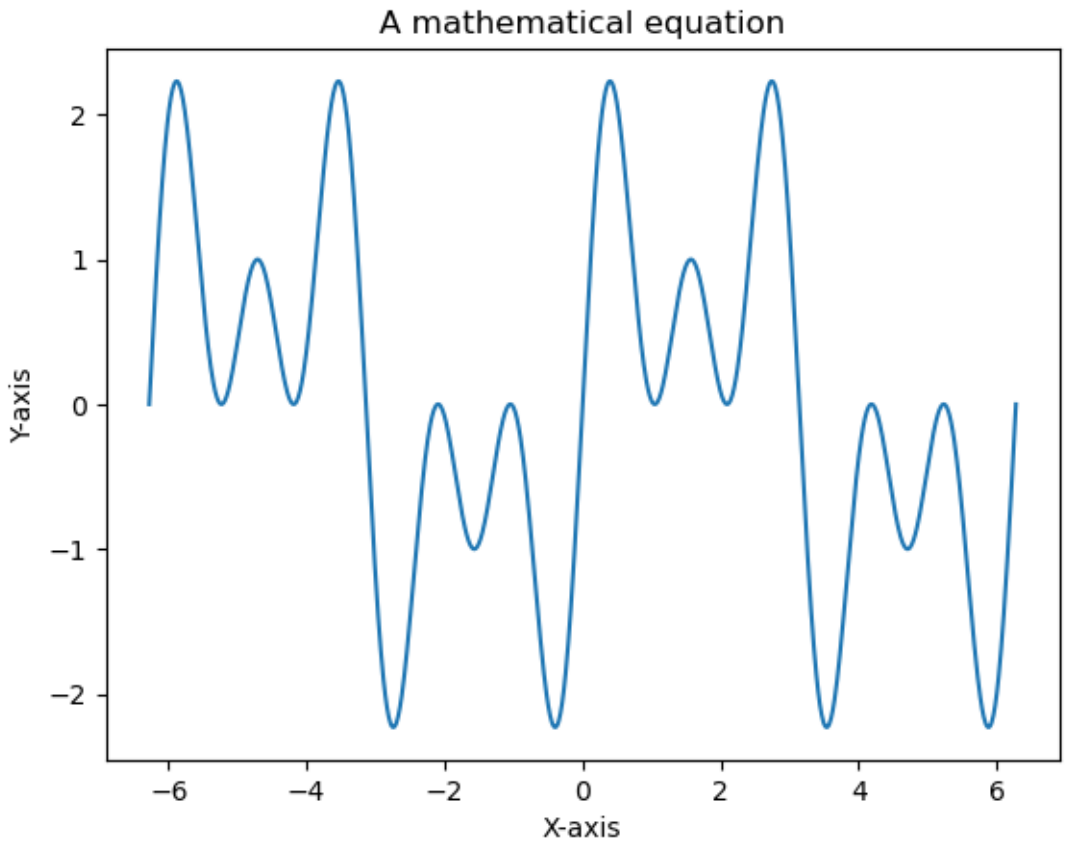
[35]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Create a numpy array without using any generation functions
array = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 1, 0, 0, 0, 0],
                  [0, 0, 1, 1, 1, 1, 0, 0, 0],
                  [0, 1, 1, 1, 1, 1, 1, 0, 0],
                  [0, 1, 1, 1, 1, 1, 1, 0, 0],
                  [0, 0, 1, 1, 1, 1, 0, 0, 0],
                  [0, 0, 0, 1, 1, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0]])

# Plot the array as an image using matplotlib
plt.imshow(array, cmap="gray")
plt.show()
```



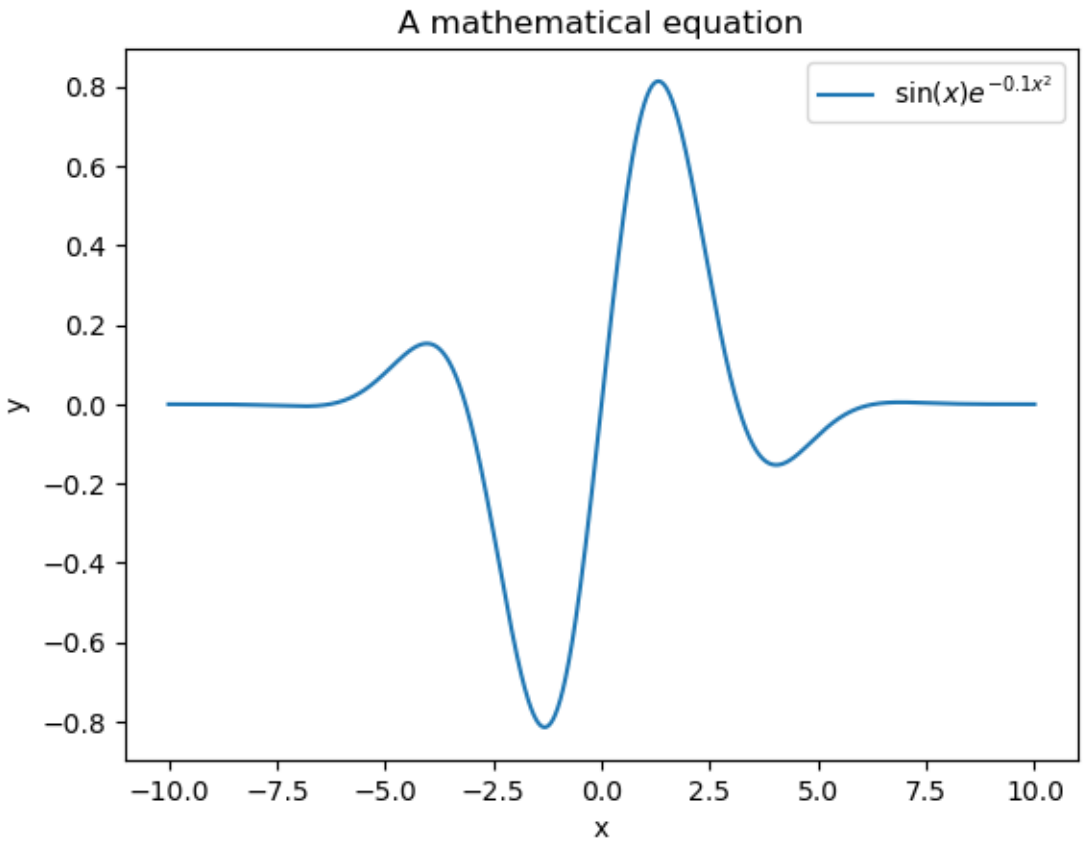
[36]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the array that represents the smiley face
smiley = np.array([[0, 0, 1, 1, 1, 0, 0],
                   [0, 1, 0, 0, 0, 1, 0],
                   [1, 0, 1, 0, 1, 0, 1],
                   [1, 0, 0, 0, 0, 0, 1],
                   [1, 0, 1, 1, 1, 0, 1],
                   [1, 0, 0, 0, 0, 0, 1],
                   [0, 1, 1, 1, 1, 1, 0]])

# Plot the smiley face using imshow
plt.imshow(smiley, cmap='gray')
plt.show()
```



[39]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the domain of the equation
x = np.linspace(-2*np.pi, 2*np.pi, 1000)

# Define the equation
y = np.sin(x) + np.sin(3*x) + np.sin(5*x)

# Plot the equation
plt.plot(x, y)

# Add labels and title to the plot
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("A mathematical equation")

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

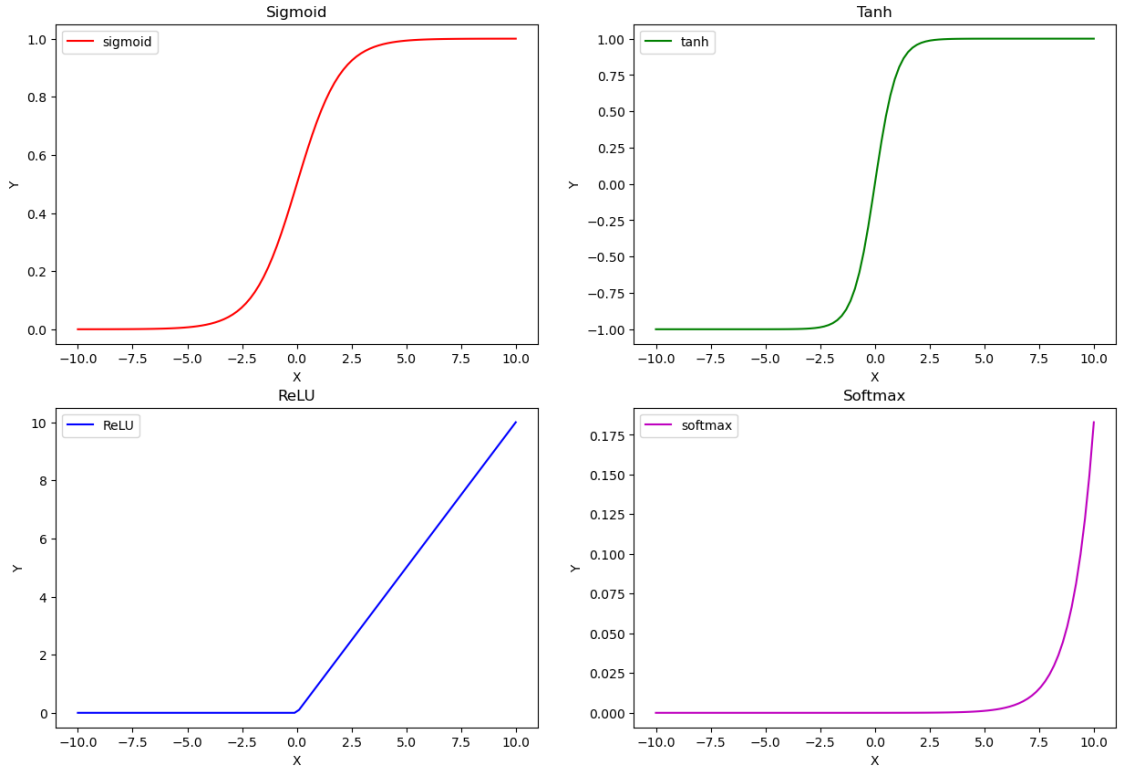


[41]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the equation using numpy operations
x = np.linspace(-10, 10, num=1000)
y = np.sin(x) * np.exp(-0.1 * x**2)

# Plot the equation
plt.plot(x, y, label='$\sin(x)e^{-0.1x^2}$')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('A mathematical equation')
plt.show()
```



[44]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Define the activation functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def tanh(x):
    return np.tanh(x)

def relu(x):
    return np.maximum(0, x)

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis=0)

# Define the input values
x = np.linspace(-10, 10, 100)

# Compute the output of the activation functions
```

```
y_sigmoid = sigmoid(x)
y_tanh = tanh(x)
y_relu = relu(x)
y_softmax = softmax(x)

# Plot the activation functions in subplots
fig, axs = plt.subplots(2, 2, figsize=(15,10))

axs[0, 0].plot(x, y_sigmoid, 'r', label='sigmoid')
axs[0, 0].set_title('Sigmoid')
axs[0, 0].set_xlabel('X')
axs[0, 0].set_ylabel('Y')
axs[0, 0].legend()

axs[0, 1].plot(x, y_tanh, 'g', label='tanh')
axs[0, 1].set_title('Tanh')
axs[0, 1].set_xlabel('X')
axs[0, 1].set_ylabel('Y')
axs[0, 1].legend()

axs[1, 0].plot(x, y_relu, 'b', label='ReLU')
axs[1, 0].set_title('ReLU')
axs[1, 0].set_xlabel('X')
axs[1, 0].set_ylabel('Y')
axs[1, 0].legend()

axs[1, 1].plot(x, y_softmax, 'm', label='softmax')
axs[1, 1].set_title('Softmax')
axs[1, 1].set_xlabel('X')
axs[1, 1].set_ylabel('Y')
axs[1, 1].legend()

plt.show()
```



Activation functions are used in artificial neural networks to introduce non-linearity into the output of each neuron. The activation functions determine the output of each neuron based on the inputs it receives. There are several activation functions used in deep learning, including:

1. Sigmoid: The sigmoid activation function maps any input value to the range of 0 to 1. It is widely used in binary classification problems.
2. ReLU (Rectified Linear Unit): The ReLU activation function outputs 0 for negative input values and the positive input value for positive input values. It is computationally efficient and widely used in deep learning.
3. Tanh (Hyperbolic Tangent): The tanh activation function maps the input values to the range of -1 to 1. It is similar to the sigmoid activation function, but better suited for multi-class classification problems.
4. Softmax: The softmax activation function is used in multiclass classification problems to convert a set of values into a probability distribution.
5. Leaky ReLU: The leaky ReLU activation function is similar to the ReLU activation function, but allows a small gradient for negative input values to improve the training process.

In terms of implementation, activation functions are typically added to a neural network using the activation argument in a dense layer. For example, in Keras, one could add a sigmoid activation function to a dense layer as follows:

```
model.add(Dense(32, activation='sigmoid'))
```

[ ]: