COURSE WORK 1: PRINCIPLES OF DATA SCIENCE

MSc Data Science, Coventry University, UK

B M J N Balasuriya
Index No: COMScDS242P-009
2024 Batch

Date: 28th Feb 2025

Part 1: Mathematics

Calculus : Question 1

1. The sigmoid activation function is widely used in neural networks and is defined as:

$$sigma(z) = \frac{1}{1 + e^{-z}}$$

Derivation the formula for using differentiation.

Steps:

To simplify the calculation define x

$$x = 1 + e^{-z}$$

Then find the derivative of x with respect to the z change

$$\frac{dx}{dz} = 0 + (-1)e^{-z}$$

When derivate 1 becomes 0 (because 1 doesn't change) and derivate of e^{-z} is $-e^{-z}$

$$\frac{dx}{dz} = -e^{-z}$$

The derivative of x respect to the z is $-e^{-z}$

Since x = $1 + -e^{-z}$ we can rewrite original the Sigmoid function using x

$$\sigma(z) = \frac{1}{x}$$

$$\sigma'(z) = x^{-1}$$

Then we can find the derivative of σ with respect to x (Finding how σ change when x changes)

$$\frac{d\sigma}{dx} = -x^{-2}$$

Since $x = 1 + e^{-z}$, we can substitute it back to the derivative.

$$\frac{d\sigma}{dx} = -(1 + e^{-z})^{-2}$$

Using the chain rule we can re write the derivative of σ with respect to the z. Here we are combining the derivative of σ with respect to x and x respect to z. The chain rule is math rule that help find how one thing change when another thing changes.

$$\frac{d\sigma}{dz} = \frac{d\sigma}{dx} \times \frac{dx}{dz}$$

Then we subside the findings from the above calculations.

$$\frac{d\sigma}{dz} = -(1 + e^{-z})^{-2} \times -e^{-z}$$

$$\frac{d\sigma}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

This is the final answer for the derivation of the sigmoid function changes when the input z changes.

The sigmoid function is a magic math function that takes any number(big, small, positive, negative) and squishes it into value between 0 and 1. This is why it's called and **activation** function. Because it help computers to convert any input into a "yes" and "no".

The graph of the sigmoid function smooth "S" shaped curve.

Neural Network: Neural network is a super smart computer brain that learn from examples. It learns looking at lot of inputs and it adjust itself to get better at telling the differences.

Backpropergation: Backpropergation is is how neural networks learn from it's mistakes. When it output some wrong value and it figures out how wrong it was and adjust it self to make better decisions next time. The process of going back and fixing mistakes called backpropergation.

Sigmoid Function: The sigmoid function is a special math tool that helps the neural network make decisions. It takes any number and squishes into 0 and 1.

Sigmoid Derivative: The sigmoid derivative is a measure how much sigmoid function changes when the input changes. For a example if the input changes a lot or just a little, the derivative helps to figure that out.

Why is sigmoid derivative important for backpropergation?

The neural network needs to find out how to adjust to matter suggestions. To do this it needs to know:

a) How wrong its guess was (**error**)
b) How much each part of the neural network contributed to the **error**.

The sigmoid derivative comes into play in the second part. It tells neural network how much the output of the sigmoid function when the input changes. This is important because;

- If the output changes a lot for small changes in the input, the network needs to adjust a lot
- If the output changes only a little, the neural network needs to adjust only a little

By using the sigmoid derivative, the neural network can figure out exactly how to tweak itself reduce the error and improve.

2. In a neural network, the Mean Squared Error (MSE) cost function is given by:

$$E = \frac{1}{2n}\sum_{j=1}^{n}(t_j - y_j)^2$$

Differentiate E with respect to y_j and explain the role of this gradient in adjusting the weights of the network.

Where:

E = The Error (MSE)
t_j = The actual (true) value for the j
y_j = The predicted value for the j
n = Number of samples

Steps:
Differentiate E with respect to the y_j

$$\frac{dE}{dy_i} = \frac{d}{dy_i}\left(\frac{1}{2n}(t_i - y_i)^2\right)$$

Using the chain rule,

$$\frac{dE}{dy_i} = \frac{1}{2n} \times 2(t_i - y_i) \times (-1)$$

Simplify,

$$\frac{dE}{dy_i} = -\frac{1}{n}(t_i - y_i)$$

Example:

When $t_i = 5, \quad y_i = 3, \quad x_i = 2, \quad \eta = 0.1$

Calculate the gradient $\frac{dE}{dy_i}$

$$\frac{dE}{dy_i} = -\frac{1}{1}(5-3) = -2$$

Calculate the gradient $\frac{dE}{dw}$

$$\frac{dE}{dw} = \frac{dE}{dy_i} \cdot x_i = -2 \cdot 2 = -4$$

$$w_{\text{new}} = w_{\text{old}} - 0.1 \cdot (-4) = w_{\text{old}} + 0.4$$

Using the MSE formula the weight is now adjusted to reduce the error when making prediction y_i closer to the actual value x_i.

The mean squad error (MSE) is a way to measure how well a neural network is performing. It calculate average squad difference between the actual values and the predicted values.

Weights are the parameters that the network adjust to make better predictions in a neural networks. Each weight connected one neuron to another, and the network learns by tweaking these weights.

The gradient $\frac{dE}{dy_i}$ tells how much the error E changes when the predicted value y_i changes, but y_i depends on the weights of the network.

The gradient $\frac{dE}{dw}$ tells how much to adjust the weight w to reduce the error E.

The gradient is crucial because it tells the network how to adjust its weights to make better predictions. By minimising the error E, the network learns to predict values closer to actual values.

The size of the gradient is crucial in the learning process of the neural network and can consider 3 different sizes.

Large Gradient: A large gradient indicates that the error is changing rapidly with the respect to the weights. The weights will be updated by larger amount. This can lead to faster learning.

Small Gradient: A small gradient indicates that the error is changing slowly with the respect to the weights. The weights will be updated by a smaller amount. This can lead to slower learning.

Zero Gradient: A zero gradient indicates that the error is not changing with the respect to the weights. The weight will not be updated at all. This means the network has reached a critical point. If the critical point is a global minimum, the network has found the best solution. If it's a local minimum the network might get stuck and stop learning.

The size of the gradient plays a crucial role in the process of a neural network.

3. The ReLU activation function is defined as:

$$f(z) = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$$

Derivative of ReLU tells that how the function changes when the input z changes.

$$f'(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$$

In simple terms if the z is positive, ReLU returns z. If the input z is a negative or zero, ReLU returns 0.

ReLU stands for Rectified Linear Unit. It's a mathematical function used in neural networks to help the network learn and make decisions.

ReLU is one of the most popular activation function in neural networks because;

- It's easy to compute and doesn't involve complex math.
- ReLU allows the network to learn faster when compared to other activation functions.
- Unlike other functions, ReLU doesn't squash large inputs, which helps neural networks learn better.

The ReLU and sigmoid activation functions are two of the most commonly used activation functions in neural networks. Each has its own advantages and limitations.

The ReLU activation function is used to train deep neural networks and sigmoid activation function is used for binary classification problems.

Computation is very simple easy in ReLU since it's only involve in a max(0, z) operation. Sigmoid involves exponential calculations which are more computation expensive.

The ReLU can face dying ReLU problem due to too many neutrons output 0 (fir negative inputs), the network can get stuck and stop learning. Sigmoid has no dying neuron problem.

When using sigmoid it's smooth and differentiable activation function where ReLU is not smooth z = 0. The derivative is discontinuous at this point.

ReLU is generally preferred for deep learning due to simplicity, speed and ability to avoid the vanishing gradient problem.

Sigmoid is worth to solve problems such as binary classification or when need a smooth and interpretable outputs.

4. The weighted input to a neuron is modeled as:

$$Z = \int_0^x (2t + 1)\, dt$$

Solve for z in terms of x.

First, split the integral into two parts:

$$Z = \int_0^x 2t\, dt + \int_0^x 1\, dt$$

Then, integrate each part separately:

$$Z = \int 2t\, dt = t^2 + C \; + \int 1\, dt = t + C$$

$$Z = t^2 + t + C$$

Now, evaluate the definite integral from 0 to x.

$$Z = \left[t^2 + t\right]_0^x$$

$$Z = (x^2 + x) - (0^2 + 0)$$

$$Z = x^2 + x$$

If x = 3, calculate Z:

$$Z = x^2 + x$$

$$Z = 3^2 + 3$$

$$Z = 9 + 3$$

$$Z = 12$$

So when x = 3, Z = 12

Integration is a fundamental concept in calculus, and it has several applications in time-series data and neural networks.

Integration can be used to calculate the total change or accumulation of quantity over time. For a example when integrating daily sales data over a week gives the total sales for the week,

Integration can be used to smooth noisy time-series data. Smoothing helps reduce noise and highlights the underlying trend in the data. Example: Temperatures are generally increasing over time.

In neural networks, integration can be used to create new features from time-series data. For a example the integral of velocity over time gives displacement.

5. A simple loss function for a neural network is defined as:

$$E(w) = w^2 - 4w + 4$$

Compute the first deviation.

$$E'(w) = \frac{d}{dw}(w^2 - 4w + 4)$$

$$= 2w - 4 + 0$$

$$= 2w - 4$$

So the first derivative is:

$$E'(w) = 2w - 4$$

If the initial weight is w=3 and the learning rate η=0.1, calculate the new weight after one iteration of gradient descent.

The formula for updating the weight using gradient decent is:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot E'(w_{\text{old}})$$

First substitute the w=3 into $E'(w)$:

$$E'(3) = 2(3) - 4 = 6 - 4 = 2$$

And then substitute into the gradient decent formula:

$$w_{\text{new}} = 3 - 0.1 \cdot 2 = 3 - 0.2 = 2.8$$

After one iteration of gradient decent, the new weight is:

$$w_{\text{new}} = 2.8$$

Discuss how adjusting the learning rate affects the convergence of the gradient:

The learning rate is one of the most important hyper parameter in training neural networks using gradient descent. It controls how much it adjust the weights of the network in response to the gradient of the loss function.

The learning rate is a small positive number that scales the gradient during the weight updates.

The weight update rule in the gradient decent is:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot E'(w_{\text{old}})$$

Where:
- $E'(w)$: The gradient of the loss function with respect to the weight w.
- η: The learning rate.

When learning rate is too small the weights are updated by very small amount in each iteration.

For a example:

If $\eta = 0.001$ and $E'(w) = 2$, the weight update is:

$$w_{\text{new}} = w_{\text{old}} - 0.001 \cdot 2 = w_{\text{old}} - 0.002$$

This tiny update means the network learns slowly.

When learning rate is too large the weights are updated by large amounts in each iteration.

For a example:

If $\eta = 1$ and $E'(w) = 2$, the weight update is:

$$w_{\text{new}} = w_{\text{old}} - 1 \cdot 2 = w_{\text{old}} - 2$$

This large update means network learns faster, but this might cause the network to jump over the optimal weights.

When learning rate is is just right the weights are updated by moderate amounts in each iteration.

For a example:

If $\eta = 0.1$ and $E'(w) = 2$, the weight update is:

$$w_{\text{new}} = w_{\text{old}} - 0.1 \cdot 2 = w_{\text{old}} - 0.2$$

The balance update allows network to learn efficiently without overshooting.

Matrix : Question 2

1. In a neural network, the output of a layer is computed as:

$$Z = W \cdot X + b$$

Explain the role of each term (W, X, b) in this equation.

W (Weight): The weights are the parameters of the neural network that are learned during the training. Weight determine the strength of the connection between the input and the output.

X (Input): The input represents the data feed into the layer. It could be the raw input data for the first layer the output from the previous layer.

b **(Bias)**: The bias is a additional parameter that allow the network to shift the output (Z) by a constant value.

Z (Output): The output of the result.

$$W = \begin{bmatrix} 2 & -3 & 4 \\ 1 & 0 & -1 \\ -2 & 5 & 3 \end{bmatrix}, X = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

$$Z = W \cdot X + b$$

Step 1:

$$W \cdot X = \begin{bmatrix} 2 & -3 & 4 \\ 1 & 0 & -1 \\ -2 & 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$$

First row:
$(2 \cdot 1) + (-3 \cdot 2) + (4 \cdot -1) = 2 - 6 - 4 = -8$

Second row:
$(1 \cdot 1) + (0 \cdot 2) + (-1 \cdot -1) = 1 + 0 + 1 = 2$

Third row:
$(-2 \cdot 1) + (5 \cdot 2) + (3 \cdot -1) = -2 + 10 - 3 = 5$

$$W \cdot X = \begin{bmatrix} -8 \\ 2 \\ 5 \end{bmatrix}$$

Step 2:

$$Z = W \cdot X + b = \begin{bmatrix} -8 \\ 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

First element:
$-8 + 1 = -7$

Second element:
$2 + (-1) = 1$

Third element:
$5 + 0 = 5$

$$Z = \begin{bmatrix} -7 \\ 1 \\ 5 \end{bmatrix}$$

2. During backpropagation, the gradient of the loss with respect to the weights W in a layer is computed as:

$$\nabla W = \delta \cdot X^T$$

Explain what δ and Xt represent in this equation.

δ (Delta): The error signal or gradient of the loss with respect to the output of the current layer. It tells how much the loss changes when the output of the layer changes.

X^T (Transpose of the Input): The transpose of the input to the layer. It represents the input data that was fed into the current layer during the forward pass.

∇W (Gradient of the Loss with Respect to the Weights): The gradient of the loss with respect to the weights. It tells how much the loss changes when the weights change.

Given $\delta = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}, \quad X = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$

Calculate:

$$\nabla W = \delta \cdot X^T$$

$$X^T = [1 \quad 3]$$

$$\nabla W = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix} \cdot [1 \quad 3]$$

First row:

$$0.2 \cdot 1 = 0.2, \quad 0.2 \cdot 3 = 0.6$$

Second row:

$$0.4 \cdot 1 = 0.4, \quad 0.4 \cdot 3 = 1.2$$

Result:

$$\nabla W = \begin{bmatrix} 0.2 & 0.6 \\ 0.4 & 1.2 \end{bmatrix}$$

3. In a neural network with m inputs and n outputs, the weight matrix W has dimensions n×m. Explain why these dimensions are necessary.

In a neural network,

- m: number of inputs (features) to the layer
- n: number if outputs (neurons) in the layer

The weight matrix W connect the m inputs to the n outputs. Each weight W_{ij} represents the strength of the connection between the j-th input and the i-th neuron.

Dimensions of $W : n \times m$:
n - number of rows (one row per neuron)
m - number of columns (one column per input feature)

This ensures that the matrix multiplication W.X is defined and produces an output vector size n x 1.

If the weight matrix W for a layer is:

$$W = \begin{bmatrix} 1 & -1 & 2 \\ 0 & 3 & 2 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Calculate W.X

$$W \cdot X = \begin{bmatrix} 1 & -1 & 2 \\ 0 & 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

First row: $(1 \cdot 1) + (-1 \cdot 2) + (2 \cdot 3) = 1 - 2 + 6 = 5$

Second row: $(0 \cdot 1) + (3 \cdot 2) + (2 \cdot 3) = 0 + 6 + 6 = 12$

Result: $W \cdot X = \begin{bmatrix} 5 \\ 12 \end{bmatrix}$

4. In a neural network, weights are updated as:

$$W_{\text{new}} = W - \eta \cdot \nabla W$$

Explain the role of each term in this equation.

This equation is used to updates the weight of a neural network during training.

W (Current Weights): The current weights of a neural network.

η **(Learning rate)**: A small positive number that control how much the weights are updated in each iteration. It determines step size of the update.

∇W: The gradient of the loss function with respect to the weights.

W_{new} (Updated Weights): The new weights after applying the update rule.

Given: $W = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}, \quad \eta = 0.1, \quad \nabla W = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$

Compute: $W_{\text{new}} = W - \eta \cdot \nabla W$

Frist compute

$$\eta \cdot \nabla W = 0.1 \cdot \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & 0.04 \end{bmatrix}$$

Then

$$W_{\text{new}} = W - \eta \cdot \nabla W = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} - \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & 0.04 \end{bmatrix}$$

$$W_{\text{new}} = \begin{bmatrix} 2 - 0.01 & 3 - 0.02 \\ 4 - 0.03 & 5 - 0.04 \end{bmatrix} = \begin{bmatrix} 1.99 & 2.98 \\ 3.97 & 4.96 \end{bmatrix}$$

$$W_{\text{new}} = \begin{bmatrix} 1.99 & 2.98 \\ 3.97 & 4.96 \end{bmatrix}$$

5. The transpose of a matrix A is denoted A^T. Define the transpose operation and explain its importance in neural network computations.

Transpose of a matrix: The transpose of a matrix A, denoted A^T, is taken by flipping the matrix over its diagonal. Once flipped:

- The rows of A becomes columns of A^T
- The columns A becomes rows of A^T

Example:

If A is a 2 x 3 matrix,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Then Then, its transpose $A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

The transpose operation is important in neural network computations for several reasons.

- Matrix Multiplication:
- Backpropagation:
- Weight Updates
- Data Representation

Given, $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

Verify if $(A \cdot B)^T = B^T \cdot A^T$

Step 1: $A \cdot B$

$$A \cdot B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

First row:
$(1 \cdot 5) + (2 \cdot 7) = 5 + 14 = 19$
$(1 \cdot 6) + (2 \cdot 8) = 6 + 16 = 22$

Second row:
$(3 \cdot 5) + (4 \cdot 7) = 15 + 28 = 43$
$(3 \cdot 6) + (4 \cdot 8) = 18 + 32 = 50$

$$A \cdot B = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Step 2: $(A \cdot B)^T$

$$(A \cdot B)^T = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}^T = \begin{bmatrix} 19 & 43 \\ 22 & 50 \end{bmatrix}$$

Step 3: $B^T \cdot A^T$

$$A^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}^T = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

Now multiply, B^T and A^T

$$B^T \cdot A^T = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

First row:
$(5 \cdot 1) + (7 \cdot 2) = 5 + 14 = 19$
$(5 \cdot 3) + (7 \cdot 4) = 15 + 28 = 43$

Second row:
$(6 \cdot 1) + (8 \cdot 2) = 6 + 16 = 22$
$(6 \cdot 3) + (8 \cdot 4) = 18 + 32 = 50$

$$B^T \cdot A^T = \begin{bmatrix} 19 & 43 \\ 22 & 50 \end{bmatrix}$$

Now from From Steps 2 and 3, we have:

$$(A \cdot B)^T = \begin{bmatrix} 19 & 43 \\ 22 & 50 \end{bmatrix} \quad B^T \cdot A^T = \begin{bmatrix} 19 & 43 \\ 22 & 50 \end{bmatrix}$$

Since both matrices are equal $(A \cdot B)^T = B^T \cdot A^T$

Part 2 : Statistics

1. The Federal Reserve provides daily interest rate information for a variety of core interest rates, such as T bills and the federal funds rate. For some rates, this dataset extends back to 1954.

Use a web scraping method to download the data for the latest 10-year period.

FRED, Federal Reserve Economic Data (https://fred.stlouisfed.org/) provides public trusted economic data. Using FRED we can download the data via different methods.

CODE REFERENCE: Notebook file name: FEDFUNDS.ipynb

federal_funds: https://fred.stlouisfed.org/data/FEDFUNDS

1_year_treasury_bill: https://fred.stlouisfed.org/data/GS1.txt

10_year_treasury_constant_maturity: https://fred.stlouisfed.org/data/GS10.txt

Above links gives required data set

Step 1: import necessary libraries

```
# Import necessary libraries
import requests  # For downloading data from the internet
from bs4 import BeautifulSoup  # For parsing HTML content
import pandas as pd  # For data manipulation and analysis
import matplotlib.pyplot as plt  # For creating charts and visualizations
from datetime import datetime  # For working with dates and times
```

Step 2: Download web page content using web scraping method

```
# Function 1: Download Web Page Content
def download_webpage(url):
    """Download the webpage content from the provided URL."""
    response = requests.get(url)
    response.raise_for_status()
    return response.content
```

This function handles the download given web page url page content. This function is designed to parse any url and it will return the downloaded web page content to caller.

Step 3: Parse the content through BeautifulSoup Parser and make a tree based text for iterating.

```
# Function 2: Parse HTML Content
def parse_html(content):
    """Parse HTML content and extract plain text using BeautifulSoup."""
    soup = BeautifulSoup(content, 'html.parser')
    return soup.get_text()
```

Step 4: Parse the iterative text content and search the required text lines

```python
# Function 3: Extract Data Lines from Text
def extract_data_lines(text_content):
    """Extract and return data lines from plain text, ignoring headers and empty lines."""
    lines = text_content.splitlines()
    data_lines = []
    start_collecting = False

    for line in lines:
        if line.startswith("DATE"):
            start_collecting = True
            continue
        if start_collecting and line.strip():
            data_lines.append(line)
    return data_lines
```

Step 5: Parse extracted data lines to covert into structured data

```python
# Function 4: Parse Lines into Structured Data
def parse_lines_to_columns(data_lines):
    """Parse each line into columns (values and dates)."""
    data = [line.split() for line in data_lines]
    flat_data = [item[0] for item in data]
    values = flat_data[0::2]  # Interest rates
    dates = flat_data[1::2]   # Dates
    return values, dates
```

Step 6: Create data frame using Pandas from values and dates and make it ready for analysis

```python
# Function 5: Create DataFrame from Values and Dates
def create_dataframe(values, dates):
    """Create a Pandas DataFrame from values and dates."""
    df = pd.DataFrame({
        'Date': pd.to_datetime(dates, errors='coerce'),
        'Value': pd.to_numeric(values, errors='coerce')
    })
    return df
```

Step 7: Main function put everything together.

```python
# Main Function: Download and Parse Interest Rate Data
def download_and_parse_interest_rates(url):
    """Download, parse, and return interest rate data as a Pandas DataFrame."""
    content = download_webpage(url)
    text_content = parse_html(content)
    data_lines = extract_data_lines(text_content)
    values, dates = parse_lines_to_columns(data_lines)
    df = create_dataframe(values, dates)
    return df
```

Federal Funds Rate Summery

```
# Download Federal Funds, convert to a data frame and print descriptive statistics:
url = 'https://fred.stlouisfed.org/data/FEDFUNDS.txt'
df = download_and_parse_interest_rates(url)

# Filter the DataFrame for the last 10 years
filtered_df = df[df['Date'] >= ten_years_ago]
print(filtered_df.describe())
```

Output:

```
                                Date       Value
count                            119  119.000000
mean   2020-01-31 02:25:12.605041920    1.796639
min              2015-03-01 00:00:00    0.050000
25%              2017-08-16 12:00:00    0.130000
50%              2020-02-01 00:00:00    1.160000
75%              2022-07-16 12:00:00    2.405000
max              2025-01-01 00:00:00    5.330000
std                              NaN    1.874870
```

Federal funds effective rates is the interest rate that banks charge each other for borrowing money daily basis in the US financial system.

Summery Statistics:
Count: There are 119 data points in this dataset for last 10 years period.
Mean: The average federal funds rate over the last 10 years is 1.80%
Minimum: The lowest rate in the dataset 0.05% for the last 10 years
Maximum: The maximum rate in the dataset 5.33% for the last 10 years
25th Percentile: 25% of the time, the rate was 0.13% or lower
50th Percentile: 50% of the time, rate was 1.16%
75th Percentile: 75% of the time, rate was 2.41% or over.
Standard Deviation: The rate varied by about 1.87% on average from the mean. This means overtime federal funds rate can change significantly.

1 Year Treasury Bill

```
# Download 1_year_treasury_bill, convert to a data frame and print descriptive statistics:
url = 'https://fred.stlouisfed.org/data/GS1.txt'
df = download_and_parse_interest_rates(url)

# Filter the DataFrame for the last 10 years
filtered_df = df[df['Date'] >= ten_years_ago]
print(filtered_df.describe())
```

Output:

	Date	Value
count	119	119.000000
mean	2020-01-31 02:25:12.605041920	1.971261
min	2015-03-01 00:00:00	0.050000
25%	2017-08-16 12:00:00	0.375000
50%	2020-02-01 00:00:00	1.530000
75%	2022-07-16 12:00:00	2.860000
max	2025-01-01 00:00:00	5.440000
std	NaN	1.789965

This represents the interests rates that investors get for lending money to the U.S government for one year.

Summery Statistics:
Count: There are 119 data points for the last 10 years period.
Mean: The average 1-year Treasury yield over this period is 1.97%.
Minimum: The lowest rate in this dataset is 0.05%
Maximum: The maximum rate in the dataset is 5.44%
25th Percentile: 25% of the time the rate was 0.38% or lower
50th Percentile: 50% of the time 1.53%
75th Percentile: 75% of the time the rate was 2.86%
Standard Deviation: The rate varied about 1.79% on average from the mean.

10 Year Treasury Constant Maturity

```
# Download 10_year_treasury_constant_maturity, convert to a data frame and print descriptive statistics:
url = 'https://fred.stlouisfed.org/data/GS10.txt'
df = download_and_parse_interest_rates(url)

# Filter the DataFrame for the last 10 years
filtered_df = df[df['Date'] >= ten_years_ago]
print(filtered_df.describe())
```

Output:

	Date	Value
count	119	119.000000
mean	2020-01-31 02:25:12.605041920	2.486639
min	2015-03-01 00:00:00	0.620000
25%	2017-08-16 12:00:00	1.735000
50%	2020-02-01 00:00:00	2.320000
75%	2022-07-16 12:00:00	3.130000
max	2025-01-01 00:00:00	4.800000
std	NaN	1.054729

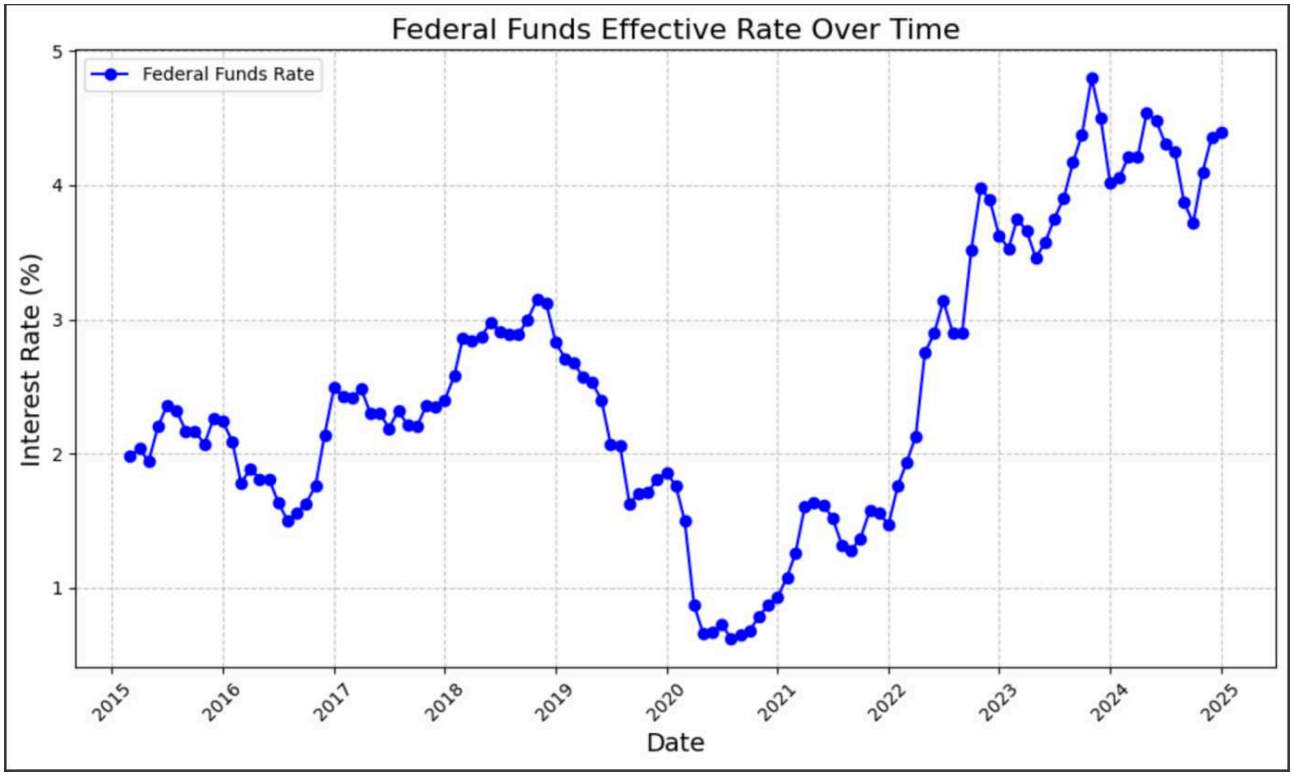
This represents the interest rate that investors earn on U.S government treasury with 10 year maturity.

Summery Statistics:
Count: There are 119 data points in the dataset
Mean: The average 10-year Treasury rate over this period is 2.49%
Minimum: The lowest rate in this dataset is 0.62%
Maximum: The highest yield in this dataset is 4.80%
25th Percentile: 25% of the time, the rate was 1.74% or lower
50th Percentile: The middle value is 2.32%
75th Percentile: 75% of the time, the rate was 3.13% or lower
Standard Deviation: The rate varied by about 1.05% on average from the mean

Data Visualisation

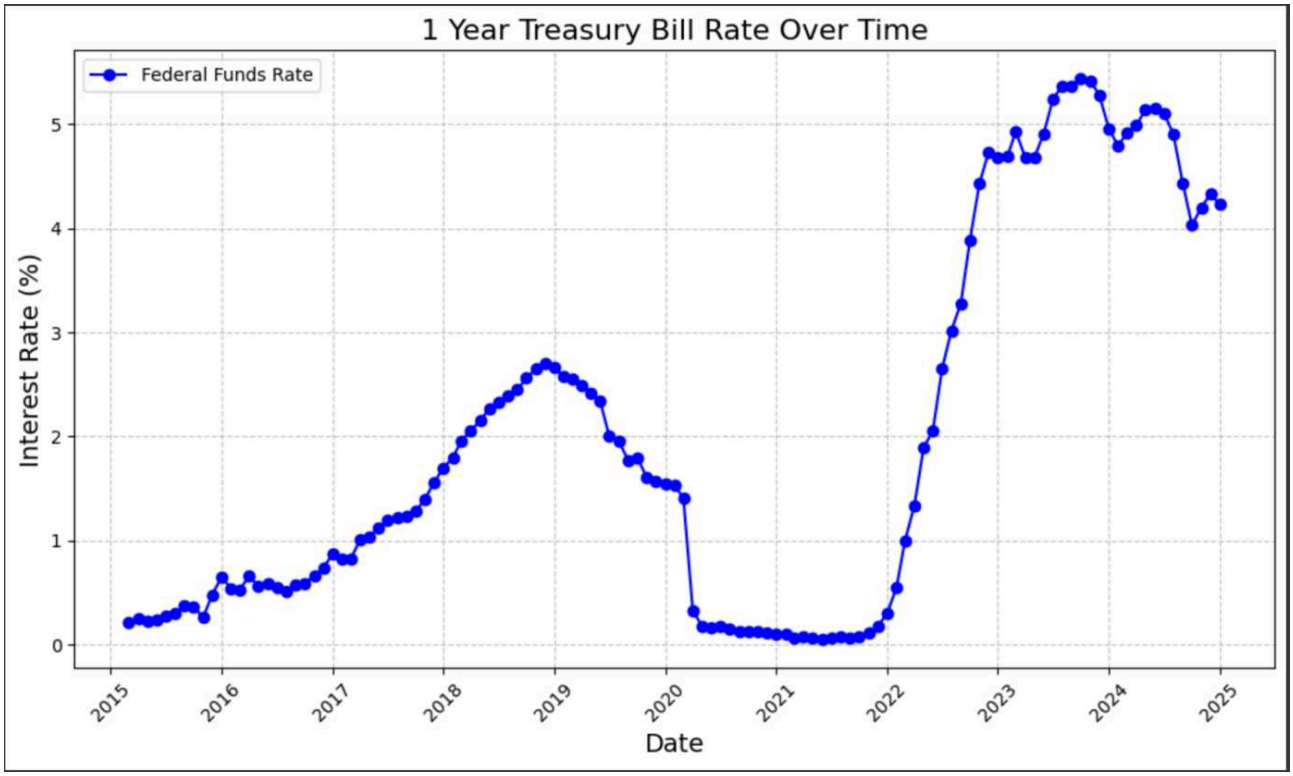
Line plot is the most appropriate choice. A line plot effectively shows how the interest rate changes over time, making it easy to observe trends, fluctuations, and key events.

```
# Create the plot
plt.figure(figsize=(10, 6))  # Set the figure size
plt.plot(filtered_df['Date'], filtered_df['Value'], marker='o', linestyle='-',
         color='b', label='Federal Funds Rate')

# Add labels and title
plt.title('Federal Funds Effective Rate Over Time', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Interest Rate (%)', fontsize=14)

# Add grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Add a legend
plt.legend()

# Show the plot
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```



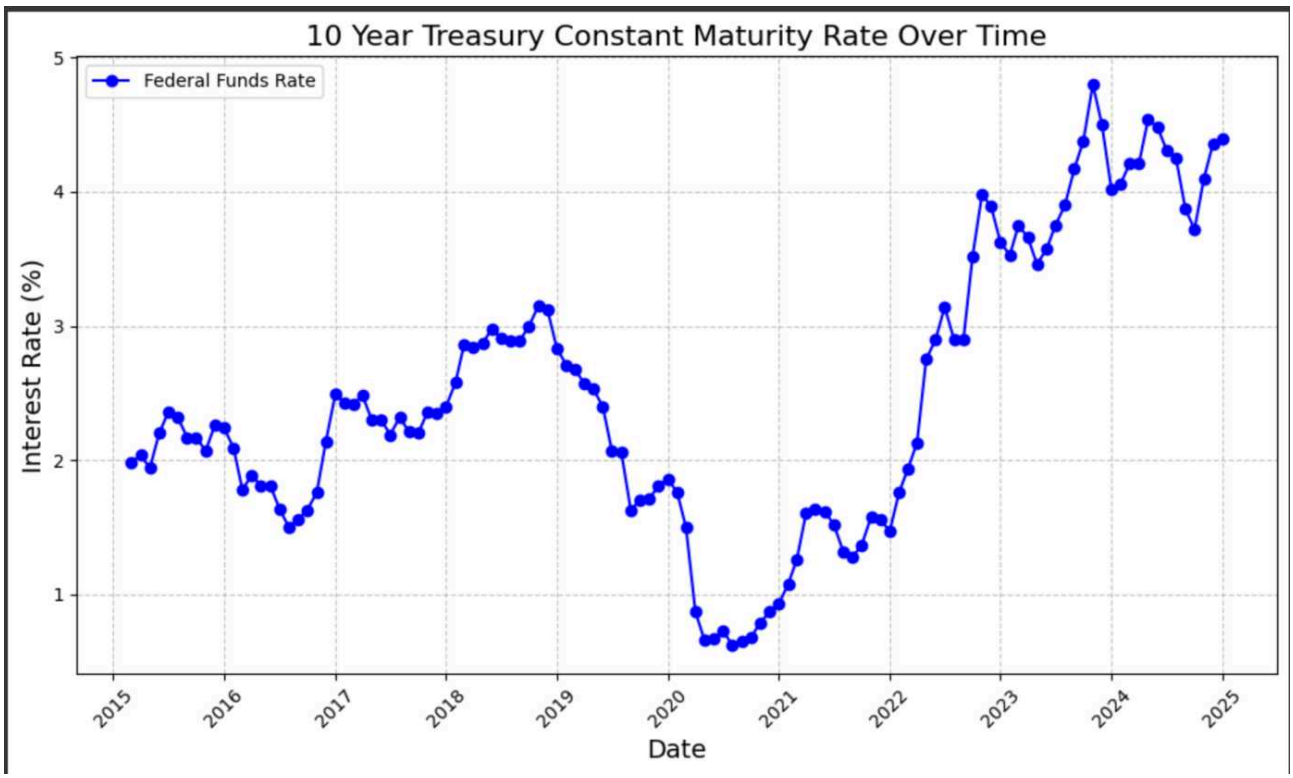
Findings:

- The line plot shows how the federal funds rate has changed over time, with markers indicating specific data points.
- Between 2019-2022 the federal funds rate was very low, this could be due to financial crisis with the pandemic.
- The graph shows a sharp rise in the federal funds rate specially after 2022.
- The graph shows some ups and downs, this could be due to support the economy stability in the country.



Findings:

- The graph likely shows a period where the 1-Year Treasury Bill Rate was very low (close to 0%) during the financial crisis between 2020-2022.
- After 2022 the economy starts recovering, the Federal Reserve may gradually raise rates.



Findings:

- The 10-Year Treasury Rate is a key indicator of long-term interest rates in the economy. It affects mortgage rates, corporate borrowing costs, and other long-term loans.
- The graph shows how the Federal Reserve has responded to different economic conditions over time, with periods of low rates and after 2022 gradually increased due to the inflation.

Barriers found in the data set

i) **Missing Values**

```
# Identify Missing Data
missing_values_count = df.isnull().sum()

# Display the count of missing values
print("Missing Values Count:")
print(missing_values_count)
```

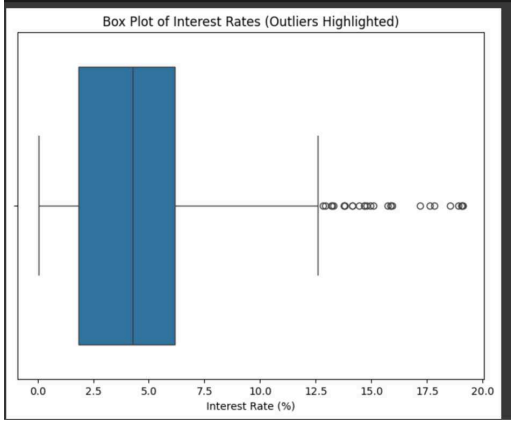
```
Missing Values Count:
Date     9
Value    9
dtype: int64
```

Missing data refers to the absence of values in certain rows or columns of the dataset. For example, some dates might not have corresponding interest rate values.

The total number of observations (count) will be lower, which can misrepresent the dataset's completeness.

ii). Outliers

```
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Value'])
plt.title('Box Plot of Interest Rates (Outliers Highlighted)')
plt.xlabel('Interest Rate (%)')
plt.show()
```



Outliers are data points that are significantly higher or lower than the rest of the dataset.

Outliers can pull the mean upward or downward, making it unrepresentative of the dataset.

iii). Negative Values

Negative values in interest rate datasets are unusual unless the dataset includes periods of negative interest rate policies

Negative values can pull the mean and median downward, distorting the central tendency.

The presence of negative values can distort the shape of the data distribution

2. Consider "heart.xls" dataset to answer the following questions. This database contains 14 attributes. The "target" field refers to the presence of heart disease in the patient. It is integer-valued 0 = no/less chance of heart attack and 1 = more chance of heart attack.

a) Statistical test can be done by using chi-square test of independence to test the relationship between the two categorical variables: **sex** (male/female) and **target** (no/less chance of heart attack vs. more chance of heart attack)

Load and prepare data set to understand the data structure can be done by running below steps.

CODE REFERENCE: Notebook file name: heart_xlsx.ipynb

Step 1: Import required libraries

```
import pandas as pd
import os
import scipy.stats as stats
from scipy.stats import chi2_contingency
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2. Read excel file and print head to visualise top 5 data set

```
df = pd.read_excel('heart.xlsx')
print(df.head())
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
```

Step 3. Check for null values before starting the analysis to make sure we have the accurate results.

```
print(df.isnull().sum())
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Step 4. Check data types of the target variables **sex** and **target**

```
df.dtypes
```

	0
age	int64
sex	int64
cp	int64
trestbps	int64
chol	int64
fbs	int64
restecg	int64
thalach	int64
exang	int64
oldpeak	float64
slope	int64
ca	int64
thal	int64
target	int64

dtype: object

Step 5. Create a contingency table with identified columns **sex** and **target**

```
# Create a contingency table
contingency_table = pd.crosstab(df['sex'], df['target'])

# Display the contingency table
print("Contingency Table:")
print(contingency_table)
```

```
Contingency Table:
target    0   1
sex
0        24  72
1       114  93
```

Key Observations from the Contingency Table

Variables:

- sex: Represents the gender of the patients.
  - 0: Female
  - 1: Male
- target: Represents the presence of heart disease.
  - 0: No/less chance of heart attack
  - 1: More chance of heart attack

- The rows represent the sex of the patients (female and male).
- The columns represent the target (presence of heart disease).
- Each cell in the table shows the count of patients falling into that category.

Female Patients (sex = 0):

- 24: Females with no/less chance of heart attack.
- 72: Females with more chance of heart attack.

Male Patients (sex = 1):

- 114: Males with no/less chance of heart attack.
- 93: Males with more chance of heart attack.

Total Patients:

Total females: 24 + 72 = 96
Total males: 114 + 93 = 207
Total patients: 96 + 207 = 303

Distribution of Heart Disease:

Females:

- 72 out of 96 (75%) have a more chance of heart attack.
- 24 out of 96 (25%) have a no/less chance of heart attack.

Males:

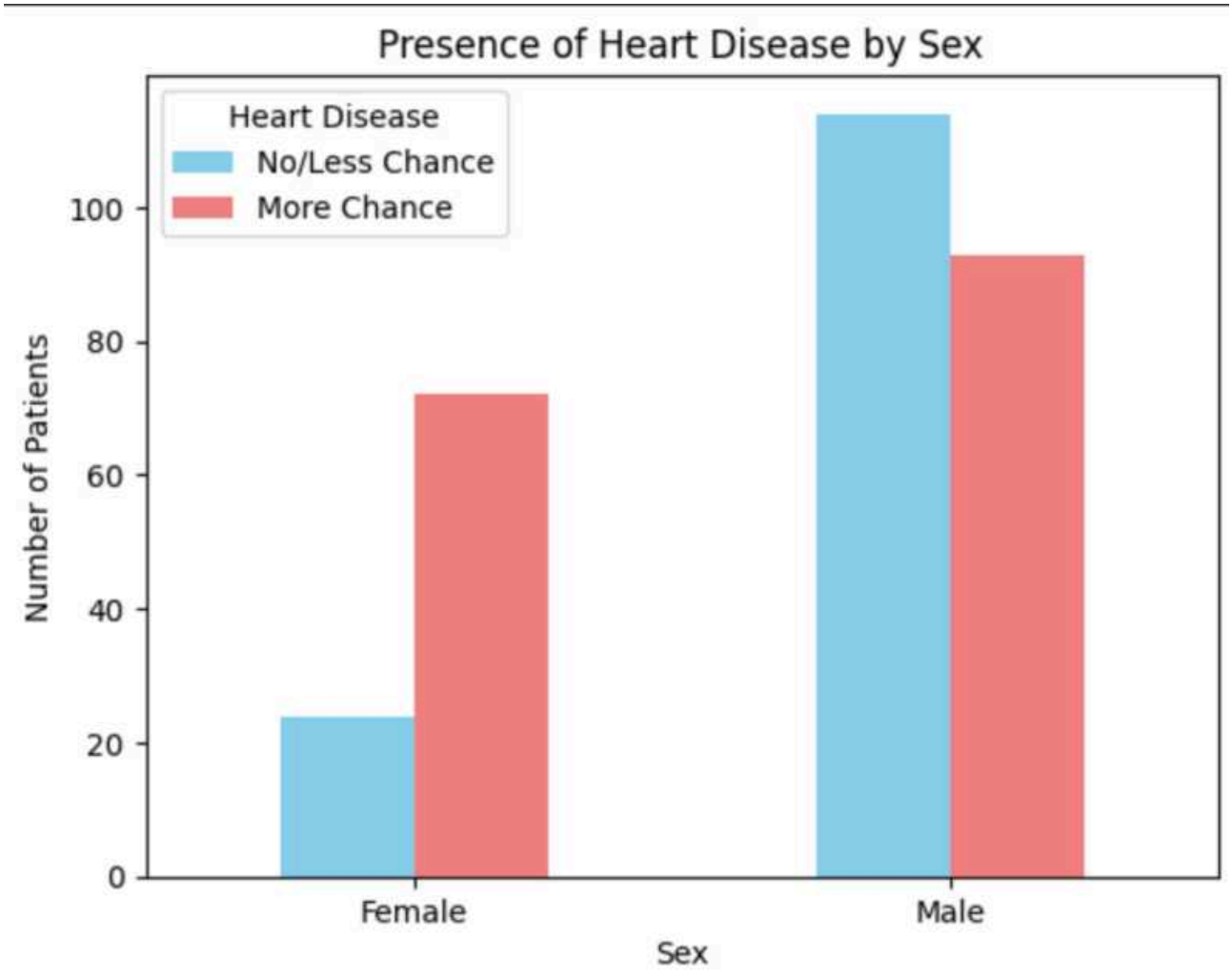
- 93 out of 207 (45%) have a more chance of heart attack.
- 114 out of 207 (55%) have a no/less chance of heart attack.

Comparison Between Genders:

- Females have a higher proportion of heart disease (75%) compared to males (45%).
- Males have a higher proportion of no/less chance of heart attack (55%) compared to females (25%).

Step 6: Visualize the relationship between sex and target

```
contingency_table.columns = ['No/Less Chance', 'More Chance']
contingency_table.index = ['Female', 'Male']

#Display the contingency table
print("Contingency Table:")
print(contingency_table)

# Plot the data
plt.figure(figsize=(8, 6))
contingency_table.plot(kind='bar', stacked=False, color=['skyblue', 'lightcoral'])
plt.title('Presence of Heart Disease by Sex')
plt.xlabel('Sex')
plt.ylabel('Number of Patients')
plt.xticks(rotation=0)
plt.legend(title='Heart Disease')
plt.show()
```



Step 7: chi-square test of independence

```python
# Create a contingency table of 'sex' vs. 'target'
contingency_table = pd.crosstab(df["sex"], df["target"])

# Perform the Chi-square test
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)

# Display results
print("Chi-square statistic:", chi2)
print("p-value:", p)
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies Table:")
print(expected)

# Interpretation
if p < 0.05:
    print("There is a significant difference between male and female patients regarding heart disease presence.")
else:
    print("There is no significant difference between male and female patients regarding heart disease presence.")
```

```
Chi-square statistic: 22.717227046576355
p-value: 1.8767776216941503e-06
Degrees of Freedom: 1
Expected Frequencies Table:
[[ 43.72277228  52.27722772]
 [ 94.27722772 112.72277228]]
There is a significant difference between male and female patients regarding heart disease presence.
```

The Results

Chi-Square Statistic (22.72)

- In this case, 22.72 is a pretty big number, so the detective is suspicious that something is going on.

p-value (0.00000188):
The p-value is extremely small (much less than 0.05), which means the detective is very confident that the differences are real and not just random.

Degrees of Freedom (1):
This is like the number of ways the data can vary. In this case, it's 1 because we're comparing two groups (males and females) and two outcomes (heart disease or no heart disease).

Expected Frequencies Table:
This is what the data would look like if there were no difference between males and females.

- Females with no heart disease: Expected 43.72, but we observed 24.
- Females with heart disease: Expected 52.28, but we observed 72.
- Males with no heart disease: Expected 94.28, but we observed 114.
- Males with heart disease: Expected 112.72, but we observed 93.

Conclusion:
The detective (chi-square test) concludes that there is a significant difference between males and females in terms of heart disease presence. In other words:

Females in this dataset are more likely to have heart disease compared to males.
This difference is not due to random chance—it's a real pattern in the data.

b) Do you think normalization or standardization techniques would help to draw meaningful insights from the dataset? If so, what are those?

Yes. Normalization and standardization are data preprocessing techniques that can help improve the performance of machine learning models and make it easier to draw meaningful insights from the dataset.

Normalization:
Scales all numeric features to a range of [0, 1] or [-1, 1].

The heart disease dataset contains both categorical (e.g., sex, cp, fbs) and numeric features (age, trestbps, chol, thalach, oldpeak).

Features like age, trestbps, chol, thalach, and oldpeak have different scales:
- age might range from 20 to 80.
- chol might range from 100 to 600.

Normalize using Min-Max Normalization

Formula:

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

Rane: [0, 1]

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

# Min-Max Normalization
min_max_scaler = MinMaxScaler()
df_min_max = pd.DataFrame(min_max_scaler.fit_transform(df), columns=df.columns)
```

```
Min-Max Normalized Data:
          age  sex        cp  trestbps      chol  fbs  restecg   thalach  \
0    0.708333  1.0  1.000000  0.481132  0.244292  1.0      0.0  0.603053
1    0.166667  1.0  0.666667  0.339623  0.283105  0.0      0.5  0.885496
2    0.250000  0.0  0.333333  0.339623  0.178082  0.0      0.0  0.770992
3    0.562500  1.0  0.333333  0.245283  0.251142  0.0      0.5  0.816794
4    0.583333  0.0  0.000000  0.245283  0.520548  0.0      0.5  0.702290
..        ...  ...       ...       ...       ...  ...      ...       ...
```

Standardization:
Transforms the data to have a mean of 0 and a standard deviation of 1.

```
# Z-Score Standardization
standard_scaler = StandardScaler()
df_standardized = pd.DataFrame(standard_scaler.fit_transform(df), columns=df.columns)
print("\nZ-Score Standardized Data:")
print(df_standardized)
```

```
Z-Score Standardized Data:
          age       sex        cp  trestbps      chol       fbs   restecg  \
0    0.952197  0.681005  1.973123  0.763956 -0.256334  2.394438 -1.005832
1   -1.915313  0.681005  1.002577 -0.092738  0.072199 -0.417635  0.898962
2   -1.474158 -1.468418  0.032031 -0.092738 -0.816773 -0.417635 -1.005832
3    0.180175  0.681005  0.032031 -0.663867 -0.198357 -0.417635  0.898962
4    0.290464 -1.468418 -0.938515 -0.663867  2.082050 -0.417635  0.898962
..        ...       ...       ...       ...       ...       ...       ...
```

Normalization or standardization can help draw meaningful insights from the dataset, especially when using machine learning algorithms sensitive to feature scales. For the heart disease dataset, standardization (Z-score) is often a good choice because it handles outliers better and is suitable for many algorithms.

c) Perform suitable regression analysis and develop a predictive model. Clearly mention the model, feature selection method and variables in the model. Further, interpret the model parameters.

To perform regression analysis and develop a predictive model for the heart disease dataset, we can use logistic regression. Logistic regression is suitable because the target variable (target) is binary (0 = no/less chance of heart attack, 1 = more chance of heart attack).

Steps for Regression Analysis.

Import Required Libraries

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler  # For scaling
```

CODE REFERENCE: Notebook file name: techgear_sales.xlsx.ipynb

Step 1: Data Preparation

```
# Step 1: Data Preparation
# Separate features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']
```

Step 2: Split the data into training and testing sets

```
# Split 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_state=42)
```

Step 3: Feature scaling

```
# Step 2: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 4: Feature selection

```
# Step 3: Feature Selection (Recursive Feature Elimination)
model = LogisticRegression(max_iter=1000)  # Increase max_iter
rfe = RFE(model, n_features_to_select=5)  # Select top 5 features
X_train_rfe = rfe.fit_transform(X_train_scaled, y_train)
X_test_rfe = rfe.transform(X_test_scaled)
```

Step 5: Visualise selected features

```
# Selected features
selected_features = X.columns[rfe.support_]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['sex', 'cp', 'exang', 'oldpeak', 'ca'], dtype='object')
```

Step 6: Model development

```
# Step 4: Model Development
model.fit(X_train_rfe, y_train)
```

```
LogisticRegression(max_iter=1000)
```

Step 7: Model selection

```
# Step 5: Model Interpretation
print("\nModel Coefficients:")
for feature, coef in zip(selected_features, model.coef_[0]):
    print(f"{feature}: {coef:.4f}")
```

```
Model Coefficients:
sex: -0.6552
cp: 0.8426
exang: -0.6848
oldpeak: -0.9797
ca: -0.7242
```

Step 8: Model evaluation

```
# Step 6: Model Evaluation
y_pred = model.predict(X_test_rfe)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8688524590163934

Confusion Matrix:
[[26  3]
 [ 5 27]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87        29
           1       0.90      0.84      0.87        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

Findings

- The model correctly predicted 86.89% of the cases in the test dataset.
- The model correctly predicted 26 cases where there was no/less chance of heart attack (class 0).
- The model incorrectly predicted 3 cases as having a more chance of heart attack (class 1) when they actually had no/less chance.
- The model incorrectly predicted 5 cases as having no/less chance of heart attack (class 0) when they actually had a more chance.
- The model correctly predicted 27 cases where there was a more chance of heart attack (class 1).

3. Analysing techgear_sales

a) Create a histogram of purchase amounts and comment on the shape of the distribution.

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_excel('techgear_sales.xlsx')
print(df.head())
```

```
   purchase_amount  customer_age  promo_period purchase_category
0        16.257116            46             1       Accessories
1        16.864348            40             1       Accessories
2        17.356663            36             1       Accessories
3        17.598137            43             1       Accessories
4        17.677614            40             1       Accessories
```

```
# Set the style for the plot
sns.set(style="whitegrid")

# Create a histogram of purchase_amount
plt.figure(figsize=(10, 6))
plt.hist(df['purchase_amount'], bins=20, color='skyblue', edgecolor='black')

# Add labels and title
plt.title('Distribution of Purchase Amounts', fontsize=16)
plt.xlabel('Purchase Amount (USD)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)

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

Findings:

- The histogram appears to be right-skewed (positively skewed).
- Most of the purchases are concentrated on the left side of the histogram (lower purchase amounts).
- There is a long tail on the right side, indicating a few high-value purchases.
- The distribution of purchase amounts is right-skewed, indicating that most customers spend smaller amounts, while a few customers make high-value purchases. This is common in retail datasets, where a small number of high-spending customers contribute significantly to total revenue.

b) What is the probability that a randomly selected purchase was made during a promotional period?

To calculate the probability that a randomly selected purchase was made during a promotional period, need to:

- Count the number of purchases made during a promotional period (promo_period = 1).
- Divide this count by the total number of purchases in the dataset.

```
# Count the number of purchases made during a promotional period
promo_purchases = (df['promo_period'] == 1).sum()

# Count the total number of purchases
total_purchases = len(df)

# Calculate the probability
probability_promo = promo_purchases / total_purchases

# Display the result
print(f"Probability that a randomly selected purchase was made during a promotional period:\
 {probability_promo:.4f} (or {probability_promo * 100:.2f}%)")
```

```
Probability that a randomly selected purchase was made during a promotional period: 0.3130 (or 31.30%)
```

Explanation of the Code

- (df[‘promo_period'] == 1).sum():
  This counts the number of rows where promo_period is equal to 1 (i.e., purchases made during a promotional period).
- len(df):
  This gives the total number of rows in the dataset, which represents the total number of purchases.
- probability_promo:
  This is the probability that a randomly selected purchase was made during a promotional period.

Conclusion:
This means that 31.30% of all purchases in the dataset were made during a promotional period. In other words, if you randomly select a purchase from the dataset, there’s a 31.30% chance that it was made during a promotional period.

c) Calculate the conditional probability that a purchase amount exceeds $1000 given that it was made during a promotional period.

To calculate the conditional probability that a purchase amount exceeds $1000 given that it was made during a promotional period, we need to:

1. Count the number of purchases made during a promotional period where the purchase amount exceeds $1000.
2. Divide this count by the total number of purchases made during a promotional period.

```
# Filter purchases made during a promotional period
promo_purchases = df[df['promo_period'] == 1]

# Count purchases exceeding $1000 during promotional periods
high_value_promo_purchases = promo_purchases[promo_purchases['purchase_amount'] > 1000]

# Calculate the conditional probability
conditional_probability = len(high_value_promo_purchases) / len(promo_purchases)

# Display the result
print(f"Conditional probability that a purchase amount exceeds $1000 given that it \
was made during a promotional period: {conditional_probability:.4f} (or \
{conditional_probability * 100:.2f}%)")
```

```
Conditional probability that a purchase amount exceeds $1000 given that it was made during a promotional period: 0.1981
```

df[df['promo_period'] == 1]:

- Filters the dataset to include only purchases made during a promotional period.
- promo_purchases[promo_purchases['purchase_amount'] > 1000]:
- Filters the promotional purchases to include only those where the purchase amount exceeds $1000.

len(high_value_promo_purchases):

- Counts the number of purchases exceeding $1000 during promotional periods.

len(promo_purchases):

- Counts the total number of purchases made during promotional periods.

conditional_probability:

- This is the conditional probability that a purchase amount exceeds $1000 given that it was made during a promotional period.

The conditional probability that a purchase amount exceeds $1000 given that it was made during a promotional period is 0.1981 (or 19.81%). If a purchase was made during a promotional period, there's a 19.81% chance that the purchase amount is greater than $1000.

d) Test whether purchase amounts follow a normal distribution using an appropriate statistical test. State your null and alternative hypotheses and interpret the results at a 5% significance level.

Step 1: State of the Hypotheses:
Null Hypothesis (H0): The purchase amounts follow a normal distribution.
Alternative Hypothesis (H1): The purchase amounts do not follow a normal distribution.

Step 2: Perform the Shapiro-Wilk Test

```python
from scipy.stats import shapiro

# Perform the Shapiro-Wilk test
stat, p_value = shapiro(df['purchase_amount'])

# Display the results
print(f"Shapiro-Wilk Test Statistic: {stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpret the results at a 5% significance level
alpha = 0.05
if p_value > alpha:
    print("Fail to reject the null hypothesis: The data follows a normal distribution.")
else:
    print("Reject the null hypothesis: The data does not follow a normal distribution.")
```

```
Shapiro-Wilk Test Statistic: 0.9378
P-value: 0.0000
Reject the null hypothesis: The data does not follow a normal distribution.
```

Based on the Shapiro-Wilk test results:

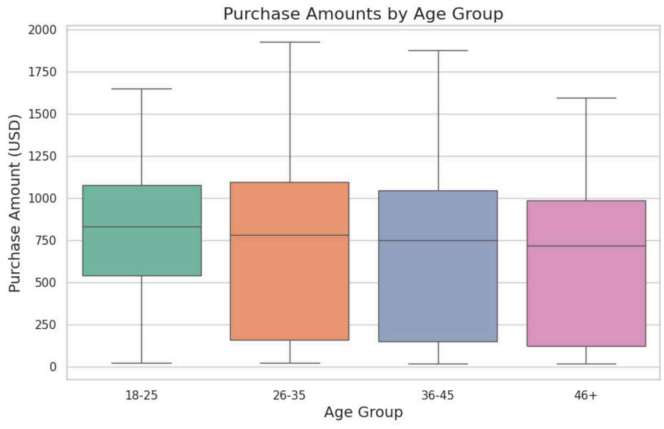
- Test Statistic (W): 0.9378
  - The test statistic ranges from 0 to 1, where values closer to 1 indicate that the data is more likely to follow a normal distribution.
  - A value of 0.9378 suggests that the data is not very close to a normal distribution.
- P-value: 0.0000
  - The p-value is 0.0000, which is much smaller than the significance level of 0.05.
  - This means we reject the null hypothesis and conclude that the purchase amounts do not follow a normal distribution.

Conclusion*:*
The purchase amounts in the dataset do not follow a normal distribution.
This is consistent with the right-skewed distribution observed in the histogram earlier, where most purchases are of lower value, with a few high-value purchases creating a long tail on the right.

e) Create age groups for customers (18-25, 26-35, 36-45, 46+) and calculate the mean purchase amount for each group.

```python
import pandas as pd
import numpy as np
from scipy.stats import f_oneway

# Create age groups
bins = [18, 25, 35, 45, np.inf]
labels = ['18-25', '26-35', '36-45', '46+']
df['age_group'] = pd.cut(df['customer_age'], bins=bins, labels=labels)

# Calculate mean purchase amount for each age group
mean_purchase_by_age = df.groupby('age_group')['purchase_amount'].mean()
print("Mean Purchase Amount by Age Group:")
print(mean_purchase_by_age)

# Conduct a one-way ANOVA test
age_groups = [df[df['age_group'] == group]['purchase_amount'] for group in labels]
f_statistic, p_value = f_oneway(*age_groups)

# Display the results
print(f"\nOne-Way ANOVA Results:")
print(f"F-statistic: {f_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpret the results at a 5% significance level
alpha = 0.05
if p_value > alpha:
    print("\nConclusion: There are no significant differences in purchase amounts across age groups.")
else:
    print("\nConclusion: There are significant differences in purchase amounts across age groups.")
```

```
Mean Purchase Amount by Age Group:
age_group
18-25    771.363320
26-35    726.627558
36-45    703.935997
46+      618.887435
Name: purchase_amount, dtype: float64

One-Way ANOVA Results:
F-statistic: 2.5785
P-value: 0.0524
```



Conclusion: There are no significant differences in purchase amounts across age groups.