GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

RECORD BOOK

GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

20MCA241 DATA SCIENCE LAB

Name: DEVIKA B

Branch: Master of Computer Applications

Semester: 3

Roll No: 22

CERTIFIED BONAFIDE RECORD WORK DONE BY

Reg No. .....

STAFF IN CHARGE

INTERNAL EXAMINER EXTERNAL EXAMINER

Contents

Assignment 1
Review of python programming

Problem Statement

Write Python code to explore and practice with the basic data types, containers, functions, and classes of Python.

1. Start by creating variables of various numeric data types and assigning them values.
2. Print the data types and values of these variables.
3. Perform mathematical operations on these variables.
4. Update the values of these variables.
5. Create boolean variables with True or False values.
6. Print the data types of these boolean variables.
7. Perform Boolean operations on these boolean variables.
8. Create string variables with text values.
9. Print the contents and lengths of these string variables.
10. Concatenate strings.
11. Format strings with variables.
12. Use string methods to manipulate strings by capitalizing, converting to uppercase, justifying, centering, replacing substrings, and stripping whitespace.
13. Create and use Python lists. Perform tasks like appending elements, indexing, slicing, and iterating through the list.
14. Create and use Python tuples. Perform tasks like indexing, slicing, and concatenation.
15. Create and use Python sets. Perform tasks like accessing, adding, deleting set elements.
16. Create and use Python dictionaries. Perform tasks like adding, updating, and removing key-value pairs, and accessing values.
17. Define simple functions with parameters and return values.
18. Call functions with different arguments and use the returned results.
19. Write functions that accept other functions as arguments.

20. Define and use Python classes. Include tasks like creating a class, defining methods, and creating instances.

21. Implement class inheritance and method overriding.

22. Create a class with class variables and instance variables, and demonstrate their usage.

1.1 Basic data types

1.1.1 Numbers

```python
# Your Python code here
print("Hello, world!")
print(x + 1)   # Addition
print(x - 1)   # Subtraction
print(x * 2)   # Multiplication
print(x ** 2)  # Exponentiation
```

```
Hello, world! 7   5   14  49
```

1.1.2 Booleans

```python
t, f = True, False
print(type(t))
print(t and f) # Logical AND;
print(t or f)  # Logical OR;
print(not t)   # Logical NOT;
print(t != f)  # Logical XOR;
```

```
<class 'bool'>
False True False True
```

1.1.3 Strings

```python
hello = 'hello'
world = "world"
print(hello, len(hello))
hw = hello + ' ' + world  # String concatenation
print(hw)
hw12 = '{} {} {}'.format(hello, world, 12)  # string formatting
print(hw12)
s = "hello"
print(s.capitalize())
print(s.upper())
```

```
hello 5
hello world
hello world 12
Hello
```

```
print(s.rjust(7))
print(s.center(7))
print(s.replace('l', '(ell)'))
print('  world '.strip())
```

```
HELLO
  hello
 hello
he(ell)(ell)o
world
```

1.2 Containers

1.2.1 Lists

```
xs = [3, 1, 2]
print(xs, xs[2])
print(xs[-1])
xs[2] = 'foo'
print(xs)
xs.append('bar')
print(xs)
x = xs.pop()
print(x, xs)
```

```
[3, 1, 2] 2
2
[3, 1, 'foo']
[3, 1, 'foo', 'bar']
foo [3, 1]
```

1.2.2 Slicing

```
nums = list(range(5))
print(nums)
print(nums[2:4])
print(nums[2:])
print(nums[:2])
print(nums[:])
print(nums[:-1])
nums[2:4] = [8, 9] print(nums)
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
```

```
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

1.2.3 Loops

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

1.2.4 List comprehensions

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)
```

```
[0, 1, 4, 9, 16]
```

1.2.5 Dictionaries

```
d = {'cat': 'cute', 'dog': 'furry'}
print(d['cat'])
print('cat' in d)
d['fish'] = 'wet'
print(d['fish'])
```

```
cute
True
wet
```

1.2.6 Sets

```
animals = {'cat', 'dog'}
print('cat' in animals)
print('fish' in animals)
animals.add('cat')
print(len(animals))
animals.remove('cat')
print(len(animals))
```

```
True
False
3
2
```

1.2.7 Tuples

```
d = {(x, x + 1): x for x in range(10)}
t = (5, 6)
print(type(t))
print(d[t])
print(d[(1, 2)])
```

```
<class 'tuple'>
5
1
```

1.3 Functions

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'
for x in [-1, 0, 1]:
    print(sign(x))
```

```
negative
zero
positive
```

1.4 Classes

```
class Greeter:
    def __init__(self, name):
        self.name = name
    def greet(self, loud=False):
        if loud:
          print('HELLO, {}'.format(self.name.upper()))
        else:
          print('Hello, {}!'.format(self.name))
g = Greeter('Fred')
g.greet()
g.greet(loud=True)
```

```
Hello, Fred!
HELLO, FRED
```

1.5 Class inheritance and method overriding.

```
class Animal:
def __init__(self, name):
self.name = name
def make_sound(self):
return "Generic animal sound"
class Cat(Animal):
def make_sound(self):
return f"{self.name} says Meow!"
class Cow(Animal):
def make_sound(self):
return f"{self.name} says Moo!"
generic_animal = Animal("Creature")
print(generic_animal.make_sound())
my_cat = Cat("Whiskers")
print(my_cat.make_sound())
my_cow = Cow("Bessie")
print(my_cow.make_sound())
```

```
Generic animal sound
Whiskers says Meow!
Bessie says Moo!
```

1.6 Class variables and instance variables

```
class Car:
number_of_wheels = 4
def __init__(self, make, model):
self.make = make
self.model = model
def display_info(self):
print(f"Make: {self.make}, Model: {self.model}, Wheels:
{Car.number_of_wheels}")
car1 = Car("Toyota", "Camry")
car2 = Car("Honda", "Civic")
print(f"Car 1 instance variables: Make='{car1.make}', Model='{car1.model}'")
print(f"Car 2 instance variables: Make='{car2.make}', Model='{car2.model}'")
print(f"Class variable (using class name): {Car.number_of_wheels}")
print(f"Class variable (using instance car1): {car1.number_of_wheels}")
print(f"Class variable (using instance car2): {car2.number_of_wheels}")
Car.number_of_wheels = 3
print(f"\nAfter changing class variable:")
print(f"Class variable (using class name): {Car.number_of_wheels}")
print(f"Class variable (using instance car1): {car1.number_of_wheels}")
print(f"Class variable (using instance car2): {car2.number_of_wheels}")
car1.display_info()
car2.display_info()
```

```
Car 1 instance variables: Make='Toyota', Model='Camry'
Car 2 instance variables: Make='Honda', Model='Civic'
Class variable (using class name): 4
Class variable (using instance car1): 4
Class variable (using instance car2): 4
After changing class variable:
Class variable (using class name): 3
Class variable (using instance car1): 3
Class variable (using instance car2): 3
Make: Toyota, Model: Camry, Wheels: 3
Make: Honda, Model: Civic, Wheels: 3
```

Assignment 2
Vectorized Computations using Numpy

Problem Statement

Implement the following computations using NumPy:

1. Create a matrix U of shape (m, n) with input values where m and n are input positive integers.
2. Compute X as the transpose of U.
3. Create a matrix Y of shape (1, m) with random values $\in$ [0, 1].
4. Create a matrix W 1 of shape (p, n) with random values $\in$ [0, 1]
   where p is an input positive integer.
5. Create a vector B1 of shape (p, 1) with random values $\in$ [0, 1].
6. Create a vector W 2 of shape (1, p) with all zeros.
7. Create a scalar B2 with a random value $\in$ [0, 1].
8. Perform the following computations iteratively 15 times:
   (a) Z1 = W 1 $\cdot$ X + B1 (Matrix Multiplication)
   (b) A1 = f (Z1) where f is a function that returns 0 for negative values and the input value itself otherwise.
   (c) Z2 = W2 $\cdot$ A1 + B2
   (d) A2 = g(Z2) where g is a function defined as g(x) = 1/1+e^{-x}.
   (e) L = $\frac{1}{2}$ (A2 - Y)^2
   (f) dA2 = A2 - Y
   (g) dZ2 = dA2 $\circ$ gprime(Z2) where gprime (x) is a function that returns g(x) $\cdot$ (1 - g(x)) and $\circ$ indicates element-wise multiplication
   (h) dA1 = $W2^T \cdot$ dZ2
   (i) dZ1 = dA1 $\circ$ fprime(Z1) where fprime is a function that returns 1 for positive values and 0 otherwise and $\circ$ indicates element-wise multiplication.
   (j) dW 1 = $\frac{1}{m} \cdot$ dZ1 $\cdot$ X^T
   (k) dB1 = $\frac{1}{m}\Sigma$dZ1 (sum along the columns)
   (l) dW2 = $\frac{1}{m}\cdot$ dZ2 $\cdot$ $A1^T$

(m) dB2 = $\frac{1}{m}\Sigma$dZ2 (sum along the columns)

(n) Update and print W 1, B1, W 2, and B2 for $\alpha = 0.01$:

i. W1 = W1 - $\alpha \cdot$ dW1

ii. B1 = B1 - $\alpha \cdot$ dB1

iii. W2 = W2 - $\alpha \cdot$ dW2

iv. B2 = B2 - $\alpha \cdot$ dB2

2.1 Matrix creation

```
import numpy as np
U = np.array([[1,2,3],[4,5,6]])
print(U)
```

```
[[1 2 3]
[4 5 6]]
```

2.2 Transpose

```
X = U.T
print(X)
```

```
[[1 4]
[2 5]
[3 6]]
```

2.3 Matrix of shape(1,m)

```
m=2
Y = np.random.random((1,m))
print(Y)
```

```
[[0.14143826 0.86003669]]
```

2.4 Matrix of shape(p,n)

```
p=3
n=3
W1 = np.random.random((p,n))
print(W1)
```

```
[[0.21198769 0.43094504 0.44782404]
[0.65786203 0.71585282 0.21237249]
[0.24155719 0.39683022 0.27351944]]
```

2.5 Vector of shape(p,1)

```
B1 = np.random.random((p,1))
print(B1)
```

```
[[0.21691861]
[0.38149642]
[0.0858541 ]]
```

2.6 Vector of shape(1,p)

```
W2 = np.zeros((1,p))
print(W2)
```

```
[[0. 0. 0.]]
```

2.7 Scalar with random values

```
B2 = np.random.rand()
print(B2)
```

```
0.6726486910015009
```

2.8 Iteration

```
def f(Z1):
return np.maximum(0, Z1)
def g(Z2):
return 1 / (1 + np.exp(-Z2))
def gprime(Z2):
gz2 = g(Z2)
return gz2 * (1 - gz2)
def fprime(Z1):
return (Z1 > 0).astype(float)
num_iterations = 15
for i in range(num_iterations):
    Z1 = np.dot(W1, X) + B1
    A1 = f(Z1)
    Z2 = np.dot(W2, A1) + B2
    A2 = g(Z2)
    L = 0.5 * np.square(A2 - Y)
```

```
    dA2 = A2 - Y
    dZ2 = dA2 * gprime(Z2)
    dA1 = np.dot(W2.T, dZ2)
    dZ1 = dA1 * fprime(Z1)
    dW1 = (1/m) * np.dot(dZ1, X.T)
    dB1 = (1/m) * np.sum(dZ1, axis=1, keepdims=True)
    dW2 = (1/m) * np.dot(dZ2, A1.T)
    dB2 = (1/m) * np.sum(dZ2, axis=1, keepdims=True)
    alpha = 0.
    W1 = W1 - alpha * dW1
    B1 = B1 - alpha * dB1
    W2 = W2 - alpha * dW2
    B2 = B2 - alpha * dB2
    print(f"Iteration {i+1}:")
    print("W1:\n", W1)
    print("B1:\n", B1)
    print("W2:\n", W2)
    print("B2:\n", B2)
    print("-" * 20)
```

```
Iteration 1:
W1:    [[0.21198053 0.43094551 0.44783215]
       [0.65786061 0.71585293 0.21237413]
       [0.24155445 0.3968304 0.27352256]]
B1:    [[0.21692624]
       [0.38149795]
       [0.08585703]]
W2:    [[-0.00292056 -0.00039661 -0.00105211]]
B2:    [[0.66731864]]
--------------------
Iteration 15:
W1:    [[0.21198053 0.43094551 0.44783215]
       [0.65786061 0.71585293 0.21237413]
       [0.24155445 0.3968304 0.27352256]]
B1:    [[0.21692624]
       [0.38149795]
       [0.08585703]]
W2:    [[-0.00292056 -0.00039661 -0.00105211]]
B2:    [[0.66731864]]
```

Assignment 3
Vectorized Computations using TensorFlow

Problem Statement

Implement the following computations using TensorFlow:

1. Create a matrix U of shape (m, n) with input values where m and n are input positive integers.
2. Compute X as the transpose of U.
3. Create a matrix Y of shape (1, m) with random values $\in$ [0, 1].
4. Create a matrix W 1 of shape (p, n) with random values $\in$ [0, 1]
   where p is an input positive integer.
5. Create a vector B1 of shape (p, 1) with random values $\in$ [0, 1].
6. Create a vector W 2 of shape (1, p) with all zeros.
7. Create a scalar B2 with a random value $\in$ [0, 1].
8. Perform the following computations iteratively 15 times:
   (a) Z1 = W 1 $\cdot$ X + B1 (Matrix Multiplication)
   (b) A1 = f (Z1) where f is a function that returns 0 for negative values and the input value itself otherwise.
   (c) Z2 = W2 $\cdot$ A1 + B2
   (d) A2 = g(Z2) where g is a function defined as g(x) = 1/1+e^{-x}.
   (e) L = $\frac{1}{2}$ (A2 - Y)^2
   (f) dA2 = A2 - Y
   (g) dZ2 = dA2 $\circ$ gprime(Z2) where gprime (x) is a function that returns g(x) $\cdot$ (1 - g(x)) and $\circ$ indicates element-wise multiplication
   (h) dA1 = $W2^T \cdot$ dZ2
   (i) dZ1 = dA1 $\circ$ fprime(Z1) where fprime is a function that returns 1 for positive values and 0 otherwise and $\circ$ indicates element-wise multiplication.
   (j) dW 1 = $\frac{1}{m} \cdot$ dZ1 $\cdot$ X^T
   (k) dB1 = $\frac{1}{m}\Sigma$dZ1 (sum along the columns)
   (l) dW2 = $\frac{1}{m}\cdot$ dZ2 $\cdot$ $A1^T$

(m) dB2 = $\frac{1}{m}\Sigma$dZ2 (sum along the columns)

(n) Update and print W 1, B1, W 2, and B2 for $\alpha = 0.01$:

i. W1 = W1 - $\alpha \cdot$ dW1

ii. B1 = B1 - $\alpha \cdot$ dB1

iii. W2 = W2 - $\alpha \cdot$ dW2

iv. B2 = B2 - $\alpha \cdot$ dB2

3.1 Matrix creation

```
import tensorflow as tf
m = 2 # number of examples
n = 3 # input features
p = 3 # hidden layer units
k = 10
U = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.float32)
print(U)
```

```
[[1 2 3]
[4 5 6]]
```

3.2 Transpose

```
X = U.T
print(X)
```

```
[[1 4]
[2 5]
[3 6]]
```

3.3 Matrix of shape(1,m)

```
Y = tf.random.uniform(shape=(1, m), minval=0, maxval=10, dtype=tf.int32)
print("Matrix Y:")
```

```
Matrix Y:
[[7 7]]
```

3.4 Matrix of shape(p,n)

```
W1=tf.Variable(tf.random.uniform((p, n), 0, 1))
print(W1)
```

```
[[0.88897145, 0.28742778, 0.6720544 ],
[0.37759733, 0.7130252 , 0.48435426],
[0.702363 , 0.24965 , 0.39652836]]
```

3.5 Vector of shape(p,1)

```
B1 =  tf.Variable(tf.random.uniform((p, 1), 0, 1))
print(B1)
```

```
[[0.39298093],
[0.20463169],
[0.7103195 ]]
```

3.6 Vector of shape(1,p)

```
W2 = tf.Variable(tf.zeros((10, p)), dtype=tf.float32)
print(W2)
```

```
[[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.],
[0., 0., 0.]]
```

3.7 Scalar with random values

```
B2 =  tf.Variable(tf.random.uniform((k, 1), 0, 1))
print(B2)
```

```
[[0.7250246 ],
[0.98551977],
[0.01832747],
[0.243927 ],
[0.01922894],
[0.39953363],
[0.3023355 ],
[0.33449578],
[0.09907246],
[0.34210122]]
```

3.8 Iteration

```
def relu(z):
return tf.maximum(0.0, z)
def relu_prime(z):
return tf.cast(z > 0, tf.float32)
def softmax(z):
expz = tf.exp(z - tf.reduce_max(z, axis=1, keepdims=True))
return expz / tf.reduce_sum(expz, axis=1, keepdims=True)
alpha = tf.constant(0.01)
num_iters = 15
for i in range(num_iters):
Z1 = tf.matmul(W1, X) + B1
A1 = relu(Z1)
Z2 = tf.matmul(W2, A1) + B2
Z2 = tf.transpose(Z2)
A2 = softmax(Z2)
Y_reshaped = tf.reshape(Y, [-1])
Y_onehot = tf.one_hot(Y_reshaped, depth=k)
dZ2 = A2 - Y_onehot
dW2 = (1. / tf.cast(m, tf.float32)) * tf.matmul(tf.transpose(dZ2), tf.
  transpose  (A1))
dB2 = (1. / tf.cast(m, tf.float32)) * tf.reduce_sum(tf.transpose(dZ2),
  axis  =1, keepdims=True)
dA1 = tf.matmul(tf.transpose(W2), tf.transpose(dZ2))
dZ1 = dA1 * relu_prime(Z1)
dW1 = (1. / tf.cast(m, tf.float32)) * tf.matmul(dZ1, tf.transpose(X))
dB1 = (1. / tf.cast(m, tf.float32)) * tf.reduce_sum(dZ1, axis=1,
  keepdims  =True)
```

```
W1.assign_sub(alpha * dW1)
B1.assign_sub(alpha * dB1)
W2.assign_sub(alpha * dW2)
B2.assign_sub(alpha * dB2)
tf.print("\nIteration", i + 1)
tf.print("W1:\n", W1)
tf.print("B1:\n", B1)
tf.print("W2:\n", W2)
tf.print("B2:\n", B2)
```

```
Iteration 1
W1:
[[0.888971448 0.287427783 0.67205441]
[0.377597332 0.713025212 0.484354258]
[0.702363 0.24965 0.396528363]]
B1:
[[0.392980933]
[0.204631686]
[0.710319519]]
W2:
[[-0.00926425308 -0.00811857637 -0.00714355102]
[-0.0120210405 -0.0105344411 -0.0092692757]
[-0.00456978474 -0.00400465587 -0.00352370483]
...
[0.0601874068 0.0527442433 0.0464097634]
[-0.0049540787 -0.00434142537 -0.00382002885]
[-0.0063169715 -0.00553577393 -0.0048709386]]
B2:
[[0.723630548]
[0.983710885]
[0.01763984]
...
[0.34355244]
[0.098326996]
[0.341150671]]
--------------------------------------------------
Iteration 15
W1:
[[0.900735795 0.307615072 0.70066452]
[0.387942642 0.730781317 0.509521306]
[0.711495519 0.26532802 0.418751866]]
```

```
B1:
[[0.401403785]
[0.2120426]
[0.716864944]]
W2:
[[-0.041429121 -0.0365763791 -0.0324084461]
[-0.0506937616 -0.0447581224 -0.0396597646]
[-0.02294375 -0.0202540122 -0.0179441832]
...
[0.2819058 0.248873606 0.220504522]
[-0.0246160254 -0.0217304751 -0.019252453]
[-0.0302957222 -0.0267453175 -0.0236961991]]
B2:
[[0.717175722]
[0.975899279]
[0.013996752]
...
[0.387821078]
[0.0944244564]
[0.336374134]]
```

Assignment 4
Implementing an FCNN from Scratch using TensorFlow

Problem Statement

Implement the following computations using TensorFlow:

1. Load the the MNIST dataset from tensorflow as x train, y train, x test and y test.The Modified National Institute of Standards and Technology (MNIST) dataset contains grayscale images of handwritten digits. The training set consists of 60,000 images and the test set contains 10,000 images. The label of each image is a digit between 0 and 9. Each image has a size of 28 × 28, consisting of 784 pixel values, where each pixel value [0, 255] with 0 corresponds to black, 255 to white, and values in between representing various shades of gray
2. Form a matrix U of shape (m, n) using TensorFlow by reshaping the images in x train to be 1D arrays of 784 (28 × 28) pixel values (Flatten the images) where m = 60, 000 is the number of training examples (training images) and n = 784 is the number of features (no. of pixel values)
3. Compute X as the transpose of U.
4. Normalize the pixel values of X to [0, 1] by dividing by 255
5. Form a matrix Y of size m corresponding to the labels [0, 9] of images by transposing y train
6. Form a matrix V by reshaping the images in x test to be 1D arrays of 784 (28 × 28) pixel values (Flatten the images).
7. Compute Xtest as the transpose of V
8. Normalize the pixel values of Xtest to [0, 1] by dividing by 255
9. Form a matrix Y test of size m corresponding to the labels [0, 9] of images by transposing y test.
10. elect an image from X and display it. Also, display the corresponding label from Y.
11. Set the hyper parameters: p = 10, the no. of neurons in hidden layer, q = 10, the no. of neurons in output layer (corresponding 10 labels in one-hot encoding format), learning rate = 0.01 and the number of training epochs (iterations over the dataset) as 1000.

12. Create a matrix W1 of shape (p, n) and initialize it as W1 = N (0, 1) × q 1 n , where N (0, 1) represents a matrix of random values drawn from a normal distribution with mean 0 and standard deviation 1.

13. Initialize the vector B1 of shape (p, 1) to zeros.

14. Initialize the matrix W2 of shape (q, p) as W2 = N (0, 1) × q1 p

15. Initialize the vector B2 of shape (q, 1) to zeros.

16. Perform the following forward propagation and backpropagation computations iteratively (No. of epochs=1000):

    (a) Z1 = W1 · X + B1 (Matrix Multiplication)

    (b) A1 = ReLU(Z1) where ReLU(x) is a function that returns 0 for negative values and the input value itself otherwise.

    (c) Z2 = W2 · A1 + B2

    (d) A2 = sof tmax(Z2) where sof tmax(x) = e P xi j e xj

    (e) Get the predicted labels from the output of A2 (index of the maximum value).

    (f) Find the accuracy of the predictions by comparing them to the true labels Y and print the progress in every 100 epochs.

    (g) Compute the cross-entropy loss using TensorFlow's tf.nn.softmax cross entropy with logits function.

    (h) dZ2 = A2 one hot Y where one hot Y is the one-hot encoded form of Y .

    (i) dA2 = W2 T · dZ2

    (j) dW2 = 1 m · dZ2 · A1 T

    (k) dB2 = 1 m PdZ2 (sum along the columns)

    (l) dZ1 = dA2∘ReLU deriv(Z1) where ReLU deriv(x) returns 1 for positive values and 0 otherwise, and ∘ indicates element-wise multiplication.

    (m) dA1 = W1 T · dZ1

    (n) dB1 = 1 m PdZ1 (sum along the columns)

    (o) dW1 = 1 m · dZ1 · XT

    (p) Update and print W1, B1, W2, and B2 for = 0.01:

        i. W1 = W1 · dW1

        ii. B1 = B1 · dB1

        iii. W2 = W2 · dW2

        iv. B2 = B2 · dB2

17. Use tensorflow GradientTape() to automatically calculate the gradients from steps (h) to (o) and redo the training steps.

18. Select one test image from Xtest, display it, reshape it to n $\times$ 1, perform forward propagation computations and predict the label. Check whether the prediction is correct

19. Use the entire Xtest and perform the forward propagation computations and predict the accuracy of the model

4.1 Loading the MNIST Dataset

```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

print(f"Shape of x_train: {x_train.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of x_test: {x_test.shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```
Shape of x_train: (60000, 28, 28)
Shape of y_train: (60000,)
Shape of x_test: (10000, 28, 28)
Shape of y_test: (10000,)
```

4.2 Forming Matrix U by Flattening Training Images

```
U = tf.reshape(x_train, (60000, 784))
print(f"Shape of U: {U.shape}"
```

```
Shape of X: (784, 60000)
```

4.3 Computing the Transpose of U to Form X

```
X = tf.transpose(U)
print(f"Shape of X: {X.shape}")
```

```
Shape of X: (784, 60000)
```

4.4 Normalizing Pixel Values of X

```
X_normalized = tf.cast(X, tf.float32) / 255.0
print(f"Shape of X_normalized: {X_normalized.shape}")
```

```
Shape of X_normalized: (784, 60000)
```

4.5 Forming Matrix Y from Training Labels

```
Y = tf.transpose(y_train)
print(f"Shape of Y: {Y.shape}")
```

```
Shape of Y: (60000,)
```

4.6 Forming Matrix V by Flattening Test Images

```
V = tf.reshape(x_test, (10000, 784))
print(f"Shape of V: {V.shape}"))
```

```
Shape of V: (10000, 784)
```

4.7 Computing the Transpose of V to Form Xtest

```
Xtest = tf.transpose(V)
print(f"Shape of Xtest: {Xtest.shape}")
```

```
Shape of Xtest: (784, 10000)
```

4.8 Normalizing Pixel Values of Xtest

```

Xtest_normalized = tf.cast(Xtest, tf.float32) / 255.0
print(f"Shape of Xtest_normalized: {Xtest_normalized.shape}")
```

```
Shape of Xtest_normalized: (784, 10000)
```

4.9 Forming Matrix Y_test from Test Labels

```

Ytest = tf.transpose(y_test)
print(f"Shape of Ytest: {Ytest.shape}")
```

```
Shape of Ytest: (10000,)
```

4.10 Displaying a Selected Image from X and its Label from Y

```

Xtest_normalized = tf.cast(Xtest, tf.float32) / 255.0
printimport matplotlib.pyplot as plt

# Select the first image from X and its corresponding label from Y
image_flat = X[:, 0]  # Select the first column (first image)
label = Y[0]          # Select the first label

# Reshape the image back to 28x28 for display
image_2d = tf.reshape(image_flat, (28, 28))

# Display the image and label
plt.imshow(image_2d.numpy(), cmap='gray')
plt.title(f"Label: {label.numpy()}")
plt.axis('off')
plt.show()
```

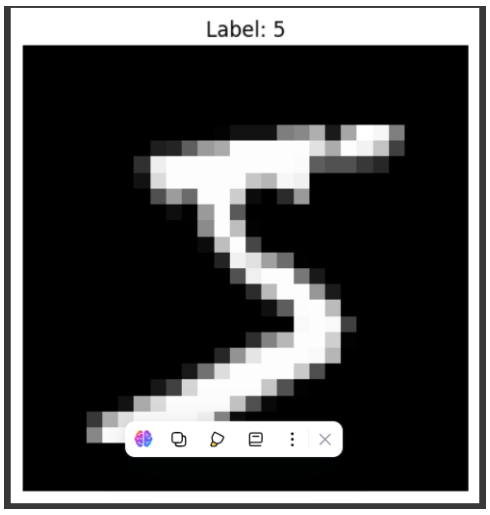


Figure 4.1: Sample MNIST digit

4.11 Setting Hyperparameters

```

# Set hyperparameters
p = 10  # Number of neurons in hidden layer
q = 10  # Number of neurons in output layer
alpha = 0.01  # Learning rate
epochs = 1000  # Number of training epochs
```

```
print(f"Number of hidden neurons (p): {p}")
print(f"Number of output neurons (q): {q}")
print(f"Learning rate (alpha): {alpha}")
print(f"Number of epochs: {epochs}")
```

```
Number of hidden neurons (p): 10
Number of output neurons (q): 10
Learning rate (alpha): 0.01
Number of epochs: 1000
```

4.12 Initializing Matrix W1

```

# n is the number of features, which is the first dimension of X or X_normalized
n = X_normalized.shape[0]

# Create W1 with shape (p, n) and initialize as specified
W1 = tf.random.normal(shape=(p, n), mean=0.0, stddev=1.0) * tf.sqrt(1.0 / tf.cast(n, tf.float32)) * tf.cast(q, tf.float32)

print(f"Shape of W1: {W1.shape}")
```

```
Shape of W1: (10, 784)
```

4.13 Initializing Vector B1

```

# Initialize B1 of shape (p, 1) to zeros
B1 = tf.zeros(shape=(p, 1))
print(f"Shape of B1: {B1.shape}")
```

```
Shape of W1: Shape of B1: (10, 1)
```

4.14 Initializing Matrix W2

```

# Initialize the matrix W2 of shape (q, p) as specified
W2 = tf.random.normal(shape=(q, p), mean=0.0, stddev=1.0) * tf.sqrt(1.0 / tf.cast(p, tf.float32)) * tf.cast(q, tf.float32)

print(f"Shape of W2: {W2.shape}")
```

```
Shape of W2: (10, 10)
```

4.15 Initializing Vector B2

```
B2 = tf.zeros(shape=(q, 1))
print(f"Shape of B2: {B2.shape}")
```

```
Shape of B2: (10, 1)
```

4.16 Forward and Backward Propagation

```

import numpy as np
import tensorflow as tf

np.random.seed(42)
tf.random.set_seed(42)

# Network dimensions
n_x = 4   # input features
n_h = 5   # hidden layer neurons
n_y = 3   # output classes
m = 10    # number of examples

# Random input and labels
X = np.random.randn(n_x, m).astype(np.float32)
Y = np.random.randint(0, n_y, size=m)

# Initialize weights and biases
W1 = tf.Variable(0.01 * tf.random.normal((n_h, n_x)))
B1 = tf.Variable(tf.zeros((n_h, 1)))
W2 = tf.Variable(0.01 * tf.random.normal((n_y, n_h)))
B2 = tf.Variable(tf.zeros((n_y, 1)))

# Hyperparameters
alpha = 0.01
epochs = 1000

# Activation and helper functions
def relu(Z):
    return tf.maximum(0.0, Z)

def relu_deriv(Z):
    return tf.cast(Z > 0, tf.float32)

def one_hot(Y, num_classes):
    return tf.transpose(tf.one_hot(Y, num_classes))
```

```
# One-hot encode labels
Y_oh = one_hot(Y, n_y)

# Training loop
for epoch in range(1, epochs + 1):
    Z1 = tf.matmul(W1, X) + B1          # (n_h, m)
    A1 = relu(Z1)                       # (n_h, m)
    Z2 = tf.matmul(W2, A1) + B2         # (n_y, m)
    A2 = tf.nn.softmax(Z2, axis=0)      # (n_y, m)

    dZ2 = A2 - Y_oh
    dW2 = (1/m) * tf.matmul(dZ2, tf.transpose(A1))
    dB2 = (1/m) * tf.reduce_sum(dZ2, axis=1, keepdims=True)
    dA2 = tf.matmul(tf.transpose(W2), dZ2)
    dZ1 = dA2 * relu_deriv(Z1)
    dW1 = (1/m) * tf.matmul(dZ1, tf.transpose(X))
    dB1 = (1/m) * tf.reduce_sum(dZ1, axis=1, keepdims=True)

    # Update parameters
    W1.assign_sub(alpha * dW1)
    B1.assign_sub(alpha * dB1)
    W2.assign_sub(alpha * dW2)
    B2.assign_sub(alpha * dB2)

    if epoch % 100 == 0 or epoch == 1 or epoch == epochs:
        loss = tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(
                labels=tf.transpose(Y_oh), logits=tf.transpose(Z2)
            )
        )
        preds = tf.argmax(A2, axis=0)
        acc = tf.reduce_mean(tf.cast(preds == Y, tf.float32)) * 100
        print(f"Epoch {epoch:4d} | Loss: {loss.numpy():.6f} | Accuracy: {acc.↩
            numpy():.2f}%")
```

```
Epoch    1 | Loss: 1.098672 | Accuracy: 0.00%
W1: [[ 0.00329903 -0.0084703   0.00314953 -0.01403582]
    [-0.02388095 -0.01040302 -0.00557581  0.00540053]
    [ 0.01699401  0.00289098 -0.01506578 -0.00263168]
    [-0.00595479 -0.01918787 -0.00621095  0.00852307]
    [-0.00407764 -0.03022152  0.00908054  0.0029764 ]]
B1: [[ 1.5987958e-05  1.0177141e-05 -7.5962621e-06  1.1386148e-05
  -2.9190123e-05]]
W2: [[ 7.9517206e-04 -8.6343288e-03  3.7410499e-03 -1.0503367e-04
  -5.0173947e-03]
    [ 6.2020831e-03 -3.2744650e-03  5.1046496e-05 -4.1438881e-03
```

```
  -1.3771456e-02]
    [-1.5465008e-02 -5.3282864e-03 -4.5003500e-03 -2.0156195e-02
  -5.8211577e-03]]
B2: [[-0.00333426  0.00366637 -0.00033212]]
Epoch  100 | Loss: 0.919533 | Accuracy: 70.00%
W1: [[ 0.00605316 -0.01340303 -0.0015622  -0.01022973]
    [-0.02388092 -0.01184518 -0.00588588  0.00605434]
    [ 0.01813323  0.00193713 -0.01580646 -0.00095434]
    [-0.00278654 -0.02366969 -0.00822273  0.01111121]
    [-0.00448961 -0.0294091   0.009661    0.00264666]]
B1: [[ 0.00215268  0.00138893  0.00010465  0.00346704       -0.00062082]]
W2: [[-3.9320788e-03 -1.0957573e-02  3.3533995e-04 -5.3856685e-03
  -1.0711278e-02]
    [ 9.7862845e-03 -5.2936390e-05  6.0489750e-03  5.8325115e-03
  -5.7512452e-03]
    [-1.4321953e-02 -6.2265713e-03 -7.0925704e-03 -2.4851965e-02
  -8.1474902e-03]]
B2: [[-0.285411    0.31050414 -0.02509311]]
------------------------------------------------------------

Epoch 1000 | Loss: 0.478045 | Accuracy: 70.00%
W1: [[ 0.18078165 -0.15968554 -0.124604    0.19827151]
    [ 0.01786004 -0.05916407 -0.03579305  0.05135824]
    [ 0.17978305 -0.13845974 -0.11892363  0.18329087]
    [ 0.30359372 -0.27656698 -0.20947768  0.360795  ]
    [ 0.08532826 -0.11095977 -0.06858437  0.11042155]]
B1: [[0.13783373 0.04692727 0.1301802  0.2450243  0.07290947]]
W2: [[-0.09107862 -0.03844109 -0.06889955 -0.14789584 -0.06729122]
    [ 0.28652543  0.07228598  0.26710048  0.49364185  0.15473409]
    [-0.20391455 -0.05108194 -0.19890921 -0.37015128 -0.11205294]]
B2: [[-1.2628626  0.8988382  0.3640232]]
------------------------------------------------------------
Final predictions: [1 1 1 1 1 1 1 1 1 1]
True labels     : [2 1 1 1 1 1 1 2 2 1]
```

4.17 Forward and Backward Propagation using GradientTape

```
import tensorflow as tf
import numpy as np
np.random.seed(1)
X = np.random.randn(4, 10)  # shape: (features, samples)
Y = (np.random.randn(1, 10) > 0).astype(np.float32)
X = tf.constant(X, dtype=tf.float32)
Y = tf.constant(Y, dtype=tf.float32)
X_normalized = (X - tf.reduce_mean(X, axis=1, keepdims=True)) / tf.math.reduce_std(X, axis=1, keepdims=True)
n_x = X_normalized.shape[0]  # Input size
n_h = 3                      # Hidden layer size
n_y = 1                      # Output size
W1 = tf.Variable(tf.random.normal([n_h, n_x], stddev=0.01))
B1 = tf.Variable(tf.zeros([n_h, 1]))
W2 = tf.Variable(tf.random.normal([n_y, n_h], stddev=0.01))
B2 = tf.Variable(tf.zeros([n_y, 1]))
learning_rate = 0.01
for epoch in range(1000):
    with tf.GradientTape() as tape:
        Z1 = tf.matmul(W1, X_normalized) + B1
        A1 = tf.nn.relu(Z1)
        Z2 = tf.matmul(W2, A1) + B2
        A2 = tf.sigmoid(Z2)
        cost = -tf.reduce_mean(Y * tf.math.log(A2 + 1e-8) + (1 - Y) * tf.math.log(1 - A2 + 1e-8))
    grads = tape.gradient(cost, [W1, B1, W2, B2])
    W1.assign_sub(learning_rate * grads[0])
    B1.assign_sub(learning_rate * grads[1])
    W2.assign_sub(learning_rate * grads[2])
    B2.assign_sub(learning_rate * grads[3])
    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Cost: {cost.numpy():.4f}")
predictions = tf.cast(A2 > 0.5, tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predictions, Y), tf.float32))
print("Final Accuracy:", accuracy.numpy())
```

```
Epoch 0, Cost: 0.6931
Epoch 100, Cost: 0.6852
Epoch 200, Cost: 0.6804
Epoch 300, Cost: 0.6774
Epoch 400, Cost: 0.6755
Epoch 500, Cost: 0.6743
Epoch 600, Cost: 0.6734
Epoch 700, Cost: 0.6726
Epoch 800, Cost: 0.6718
```

```
Epoch 900, Cost: 0.6707
Final Accuracy: 0.6
```

4.18 Predicting Label for a Single Test Image

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
# Step 1: Load MNIST data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
# Normalize pixel values
X_train, X_test = X_train / 255.0, X_test / 255.0
# Flatten images to vectors (n x 1 format later)
X_test_flat = X_test.reshape(X_test.shape[0], -1)
# Step 2: Load a trained model (or create a simple one for demo)
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(784,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(X_train.reshape(-1, 784), y_train, epochs=1, verbose=0)
# Step 3: Pick one test image
index = 5
x_sample = X_test_flat[index]   # shape (784,)
y_true = y_test[index]
plt.imshow(X_test[index], cmap='gray')
plt.title(f"True label: {y_true}")
plt.axis('off')
plt.show()
# Step 5: Reshape to (n, 1) for forward pass
x_sample_reshaped = x_sample.reshape(1, -1)  # model expects batch size first
# Step 6: Forward propagation (prediction)
y_pred_probs = model.predict(x_sample_reshaped)
y_pred_label = np.argmax(y_pred_probs)
print(f"Predicted label: {y_pred_label}")
print("Correct prediction!" if y_pred_label == y_true else "Wrong prediction")
```

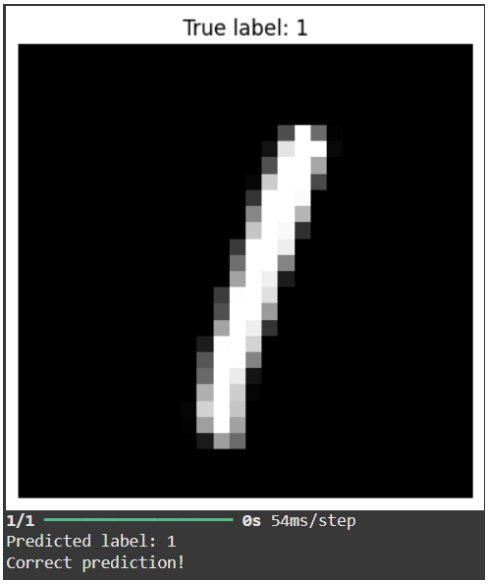


Figure 4.2: Sample MNIST digit

4.19 Evaluating Model Accuracy on Entire Test Set

```
1
2  import tensorflow as tf
3  import numpy as np
4
5  # 1. Load MNIST dataset
6  (X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.mnist.load_data()
7
8  # 2. Preprocess data
9  X_train = X_train.reshape(-1, 28*28).astype("float32") / 255.0
10 X_test = X_test.reshape(-1, 28*28).astype("float32") / 255.0
11
12 # Convert labels to one-hot encoding
13 Y_train_onehot = tf.keras.utils.to_categorical(Y_train, num_classes=10)
14 Y_test_onehot = tf.keras.utils.to_categorical(Y_test, num_classes=10)
15 # 3. Build model
16 model = tf.keras.Sequential([
17     tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
18     tf.keras.layers.Dense(64, activation='relu'),
19     tf.keras.layers.Dense(10, activation='softmax')
20 ])
21 model.compile(optimizer='adam',
22               loss='categorical_crossentropy',
23               metrics=['accuracy'])
```

```python
# 4. Train model
model.fit(X_train, Y_train_onehot, epochs=5, batch_size=32, verbose=1)

# 5. Forward propagation on entire X_test
loss, accuracy = model.evaluate(X_test, Y_test_onehot, verbose=0)

print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/5
1875/1875  7s 3ms/step - accuracy: 0.8684 - loss: 0.4405
Epoch 2/5
1875/1875  11s 4ms/step - accuracy: 0.9666 - loss: 0.1118
Epoch 3/5
1875/1875  7s 4ms/step - accuracy: 0.9779 - loss: 0.0715
Epoch 4/5
1875/1875  10s 4ms/step - accuracy: 0.9843 - loss: 0.0518
Epoch 5/5
1875/1875  6s 3ms/step - accuracy: 0.9884 - loss: 0.0386
Test Accuracy: 97.61%
```

Assignment 5
Explore Data and Create Linear Regression Model

Problem Statement

Implement the following computations using Pandas and TensorFlow:

1. Load the dataset and import it into a Pandas DataFrame.
2. Display the first five rows and the last three rows of the dataset.
3. Get the dimensions (number of rows and columns) of the dataset.
4. Generate descriptive statistics (mean, median, standard deviation, five-point summary, IQR, etc.) for the data.
5. Print a concise summary of the dataset as information on data types (schema) and missing values.
6. Add a new column named "X22" by converting the "house age" from years to days.
7. Delete the column "X22" from the dataset.
8. Create three new instances synthetically and add them to the dataset.
9. Delete the newly inserted three instances from the dataset.
10. Update the "house price of unit area" to 110, provided it is currently greater than the amount.
11. Find the latitude and longitude of the houses whose prices are less than or equal to 20.
12. Add the missing convenience store values of instances by calculating the average number of convenience stores.
13. Find the normalized distance to the nearest train station by performing: (a) Z-score normalization. (b) Min-max normalization. (c) Decimal scaling.
14. Generate the following basic visualizations using Seaborn. Customize your visualizations by adding titles, labels, legends, and appropriate color schemes. (a) Create a histogram for the ''Y house price of unit area" attribute. (b) Create a box-and-whisker plot for the ''Y house price of unit area" attribute. (c) Create a scatter plot showing house prices against house age. (d) Add a second scatter plot showing house prices against distance to the nearest MRT station.

15. Form the Design Matrix X of shape m × n + 1 in order to apply normal equation method where m is the number of training examples and n is the number of input features. Only use the two normalized input features 'X2 house age' and 'X3 distance to the nearest MRT station' from the dataset as second and third columns respectively and all 1 s as the first column. Also, form output vector Y of shape m × 1.

16. Find the parameter vector W using the normal equation method as $W = \left(X^T X\right)^{-1} X^T Y$.

17. Implement the gradient descent algorithm with the following steps.

    - Form the Design Matrix X of shape n × m. Only use the two normalized input features 'X2 house age' and 'X3 distance to the nearest MRT station' and the output vector Y of shape 1 × m.
    - Initialize the parameter vector W of shape 1 × n and bias b (scalar).
    - Repeat the following steps to a certain number of iterations with learning rate $\alpha = 0.01$, and print the final parameter values.
      (a) Calculate the prediction $\hat{Y} = WX + b$.
      (b) Compute loss $L = \frac{1}{2} \times (\hat{Y} - Y)^2$.
      (c) Compute error $E = \hat{Y} - Y$.
      (d) Compute the gradient with respect to W as $dW = \frac{1}{m} E \cdot X^T$ and with respect to b as $db = \frac{1}{m} \times \sum E$ (sum over the columns).
      (e) Update $W = W - \alpha dW$ and $b = b - \alpha db$.
          i. Use tensorflow GradientTape() to automatically calculate the gradients in the above step (d) and redo the training steps and print the final parameter values.

18. Define a class to create a Linear Regression model with methods fit and predict. Use the above iterative process to implement the model's training within the fit method.

5.1 Load the Dataset

```
from google.colab import files
import pandas as pd
import tensorflow as tf
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Upload the CSV file
# uploaded = files.upload()   # Choose "Real estate.csv" (Commented out for ↩
    direct execution in a script)
```

```
# Load into Pandas DataFrame
df = pd.read_csv("Real estate - Real estate.csv")  # make sure filename ↩
    matches exactly
print("    Dataset loaded successfully!")
print(df.head())
# Convert to TensorFlow tensor
data_tensor = tf.convert_to_tensor(df.values, dtype=tf.float32)
print("\nTensorFlow tensor shape:", data_tensor.shape)
```

```
Saving Real estate - Real estate.csv to Real estate - Real estate (1).csv
 Dataset loaded successfully!
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1

TensorFlow tensor shape: (415, 7)
```

5.2 First 5 rows & last 3 rows

```
print("First 5 rows:")
print(df.head())
print("\nLast 3 rows:")
print(df.tail(3))
```

First 5 rows:

	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	\
0	2012.917	32.0	84.87882	
1	2012.917	19.5	306.59470	
2	2013.583	13.3	561.98450	
3	2013.500	13.3	561.98450	
4	2012.833	5.0	390.56840	

	X4 number of convenience stores	X5 latitude	X6 longitude	\
0	10.0	24.98298	121.54024	
1	9.0	24.98034	121.53951	
2	5.0	24.98746	121.54391	
3	5.0	24.98746	121.54391	
4	5.0	24.97937	121.54245	

	Y house price of unit area
0	37.9
1	42.2
2	47.3
3	54.8
4	43.1

Last 3 rows:

	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	\
412	2013.000	8.1	104.81010	
413	2013.500	6.5	90.45606	
414	2013.167	1.9	355.00000	

	X4 number of convenience stores	X5 latitude	X6 longitude	\
412	5.0	24.96674	121.54067	
413	9.0	24.97433	121.54310	
414	NaN	24.97293	121.54026	

	Y house price of unit area
412	52.5
413	63.9
414	40.5

5.3 Dimensions

```
print("\nShape of dataset (rows, columns):", df.shape)
```

```
Shape of dataset (rows, columns): (415, 7)
```

5.4 Descriptive statistics

```
print("\nDescriptive Statistics:")
print(df.describe(include="all"))

print("\nMedian values:")
print(df.median(numeric_only=True))

print("\nFive-point summary:")
print(df.describe().loc[["min", "25%", "50%", "75%", "max"]])

# IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print("\nInterquartile Range (IQR):")
print(IQR)
```

```
Descriptive Statistics:
       X1 transaction date  X2 house age  \
count           415.000000    415.000000
mean           2013.149014     17.674458
std               0.281628     11.405161
min            2012.667000      0.000000
25%            2012.917000      8.950000
50%            2013.167000     16.100000
75%            2013.417000     28.100000
max            2013.583000     43.800000

       X3 distance to the nearest MRT station  \
count                                415.000000
mean                                1082.129338
std                                 1261.092057
min                                   23.382840
25%                                  289.324800
50%                                  492.231300
75%                                 1452.760000
```

```
max                                6488.021000

       X4 number of convenience stores  X5 latitude  X6 longitude  \
count                       414.000000   415.000000    415.000000
mean                          4.094203    24.969039    121.533378
std                           2.945562     0.012397      0.015332
min                           0.000000    24.932070    121.473530
25%                           1.000000    24.963010    121.528570
50%                           4.000000    24.971100    121.538630
75%                           6.000000    24.977450    121.543300
max                          10.000000    25.014590    121.566270

       Y house price of unit area
count                  415.000000
mean                    37.986265
std                     13.590608
min                      7.600000
25%                     27.700000
50%                     38.500000
75%                     46.600000
max                    117.500000

#upload
Median values:
X1 transaction date                          2013.16700
X2 house age                                   16.10000
X3 distance to the nearest MRT station       492.23130
X4 number of convenience stores                4.00000
X5 latitude                                   24.97110
X6 longitude                                 121.53863
Y house price of unit area                    38.50000
dtype: float64

Five-point summary:
     X1 transaction date  X2 house age  \
min             2012.667          0.00
25%             2012.917          8.95
50%             2013.167         16.10
```

```
75%             2013.417         28.10
max             2013.583         43.80

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
min                              23.38284                              0.0
25%                             289.32480                              1.0
50%                             492.23130                              4.0
75%                            1452.76000                              6.0
max                            6488.02100                             10.0

     X5 latitude  X6 longitude  Y house price of unit area
min     24.93207     121.47353                         7.6
25%     24.96301     121.52857                        27.7
50%     24.97110     121.53863                        38.5
75%     24.97745     121.54330                        46.6
max     25.01459     121.56627                       117.5

Interquartile Range (IQR):
X1 transaction date                          0.50000
X2 house age                                19.15000
X3 distance to the nearest MRT station    1163.43520
X4 number of convenience stores              5.00000
X5 latitude                                  0.01444
X6 longitude                                 0.01473
Y house price of unit area                  18.90000
dtype: float64
```

5.5 Schema & missing values

```
1 print("\nInfo about dataset:")
2 print(df.info())
3
4 print("\nMissing values count:")
5 print(df.isnull().sum())
```

```
Info about dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 415 entries, 0 to 414
Data columns (total 7 columns):
```

```
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   X1 transaction date                     415 non-null    float64
 1   X2 house age                            415 non-null    float64
 2   X3 distance to the nearest MRT station  415 non-null    float64
 3   X4 number of convenience stores         414 non-null    float64
 4   X5 latitude                             415 non-null    float64
 5   X6 longitude                            415 non-null    float64
 6   Y house price of unit area              415 non-null    float64
dtypes: float64(7)
memory usage: 22.8 KB
None

Missing values count:
X1 transaction date                     0
X2 house age                            0
X3 distance to the nearest MRT station  0
X4 number of convenience stores         1
X5 latitude                             0
X6 longitude                            0
Y house price of unit area              0
dtype: int64
```

5.6 Add new column "X22" (house age in days)

```
df["X22"] = df["X2 house age"] * 365
print(df[["X2 house age", "X22"]].head())
```

```
   X2 house age      X22
0          32.0  11680.0
1          19.5   7117.5
2          13.3   4854.5
3          13.3   4854.5
4           5.0   1825.0
```

5.7 Delete column "X22"

```
df.drop("X22", axis=1, inplace=True)
print(df.head())
```

```
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1
```

5.8 Add 3 new instances

```python
print("Shape before adding:", df.shape)

# Create 3 synthetic instances with the same columns
new_rows = pd.DataFrame([
    [0, 15, 560.0, 2, 24.98, 121.54, 45.0],   # Example instance
    [0, 30, 1800.0, 3, 24.96, 121.52, 35.0],  # Example instance
    [0, 5, 350.0, 1, 24.97, 121.50, 55.0]     # Example instance
], columns=df.columns)   #     use same column names

# Append to dataset
df = pd.concat([df, new_rows], ignore_index=True)

print("Shape after adding:", df.shape)
print(df.tail(5))   # show last rows including new ones
```

```
Shape before adding: (415, 7)
Shape after adding: (418, 7)
     X1 transaction date  X2 house age  \
413             2013.500           6.5
414             2013.167           1.9
```

```
415                0.000          15.0
416                0.000          30.0
417                0.000           5.0

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
413                                90.45606                              9.0
414                               355.00000                              NaN
415                               560.00000                              2.0
416                              1800.00000                              3.0
417                               350.00000                              1.0

     X5 latitude  X6 longitude  Y house price of unit area
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
415     24.98000     121.54000                        45.0
416     24.96000     121.52000                        35.0
417     24.97000     121.50000                        55.0
```

5.9 Delete those 3 instances

```
1 df = df[:-3]
2 print("After deleting new rows:", df.tail(5))
```

```
After deleting new rows:      X1 transaction date  X2 house age  \
410                2012.667           5.6
411                2013.250          18.8
412                2013.000           8.1
413                2013.500           6.5
414                2013.167           1.9

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
410                                90.45606                              9.0
411                               390.96960                              7.0
412                               104.81010                              5.0
413                                90.45606                              9.0
414                               355.00000                              NaN

     X5 latitude  X6 longitude  Y house price of unit area
410     24.97433     121.54310                        50.0
411     24.97923     121.53986                        40.6
```

412	24.96674	121.54067	52.5
413	24.97433	121.54310	63.9
414	24.97293	121.54026	40.5

5.10 Update house price if 110

```
df.loc[df["Y house price of unit area"] > 110, "Y house price of unit area"] =↩
    110
print("Maximum 'Y house price of unit area' after update:", df["Y house price ↩
    of unit area"].max())
```

```
Maximum 'Y house price of unit area' after update: 110.0
```

5.11 Latitude & Longitude where price ≤ 20

```
cheap_houses = df[df["Y house price of unit area"] <= 20][["X5 latitude", "X6 ↩
    longitude"]]
print(cheap_houses)
```

	X5 latitude	X6 longitude
8	24.95095	121.48458
40	24.94155	121.50381
41	24.94297	121.50342
48	24.94684	121.49578
49	24.94925	121.49542
55	24.94968	121.53009
73	24.94155	121.50381
83	24.96056	121.50831
87	24.94297	121.50342
93	24.94920	121.53076
113	24.96172	121.53812
116	24.94375	121.47883
117	24.93885	121.50383
155	24.94155	121.50381
156	24.94883	121.52954
162	24.94297	121.50342
170	24.94741	121.49628
176	24.94867	121.49507
180	24.94898	121.49621
183	24.94155	121.50381
226	24.94155	121.50381

```
229     24.94890     121.53095
231     24.94235     121.50357
232     24.95032     121.49587
249     24.95743     121.47516
251     24.94960     121.53018
255     24.95095     121.48458
298     24.94155     121.50381
309     24.94883     121.52954
320     24.93885     121.50383
329     24.93885     121.50383
330     24.94935     121.53046
331     24.94826     121.49587
347     24.95719     121.47353
384     24.94297     121.50342
409     24.94155     121.50381
```

5.12 Fill missing values in convenience stores with mean

```
# Step 1: Calculate average number of convenience stores (ignoring NaN)
avg = df["X4 number of convenience stores"].mean()

# Step 2: Fill missing values with that average
df["X4 number of convenience stores"] = df["X4 number of convenience stores"].↩
    fillna(avg)

# Step 3: Verify
print("Missing values after filling:")
print(df["X4 number of convenience stores"].isnull().sum())
```

```
Missing values after filling:
0
```

5.13 Normalization

```
x3 = df["X3 distance to the nearest MRT station"]

# (a) Z-score
z_score = (x3 - x3.mean()) / x3.std()

# (b) Min-Max
min_max = (x3 - x3.min()) / (x3.max() - x3.min())

# (c) Decimal scaling
```

```
scaling_factor = 10**len(str(int(x3.abs().max())))
decimal_scaled = x3 / scaling_factor

print("\nZ-score normalization:\n", z_score.head())
print("\nMin-Max normalization:\n", min_max.head())
print("\nDecimal scaling:\n", decimal_scaled.head())
```

```
Z-score normalization:
 0   -0.790783
1   -0.614971
2   -0.412456
3   -0.412456
4   -0.548383
Name: X3 distance to the nearest MRT station, dtype: float64

Min-Max normalization:
 0    0.009513
1    0.043809
2    0.083315
3    0.083315
4    0.056799
Name: X3 distance to the nearest MRT station, dtype: float64

Decimal scaling:
 0    0.008488
1    0.030659
2    0.056198
3    0.056198
4    0.039057
Name: X3 distance to the nearest MRT station, dtype: float64
```

5.14 Visualizations

```
# (a) Histogram for 'Y house price of unit area'
plt.figure(figsize=(8, 6))
sns.histplot(df["Y house price of unit area"], kde=True, color="skyblue")
plt.title("Histogram of House Price per Unit Area")
plt.xlabel("Y house price of unit area")
plt.ylabel("Frequency")
plt.grid(axis='y', alpha=0.75)
plt.show()

# (b) Box-and-whisker plot for 'Y house price of unit area'
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df["Y house price of unit area"], color="lightcoral")
plt.title("Boxplot of House Price per Unit Area")
plt.ylabel("Y house price of unit area")
plt.grid(axis='y', alpha=0.75)
plt.show()

# (c) Scatter plot: House prices against house age
plt.figure(figsize=(10, 7))
sns.scatterplot(x=df["X2 house age"], y=df["Y house price of unit area"], color="green", alpha=0.7)
plt.title("House Price vs House Age")
plt.xlabel("X2 house age (years)")
plt.ylabel("Y house price of unit area")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# (d) Scatter plot: House prices against distance to the nearest MRT station
plt.figure(figsize=(10, 7))
sns.scatterplot(x=df["X3 distance to the nearest MRT station"], y=df["Y house price of unit area"], color="orange", alpha=0.7)
plt.title("House Price vs Distance to Nearest MRT Station")
plt.xlabel("X3 distance to the nearest MRT station (meters)")
plt.ylabel("Y house price of unit area")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

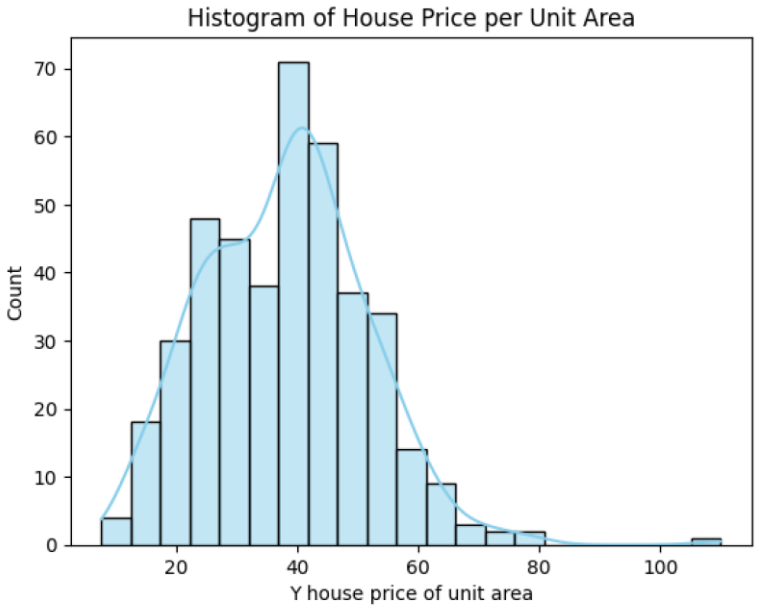


Figure 5.3: Histogram of House Price per Unit Area

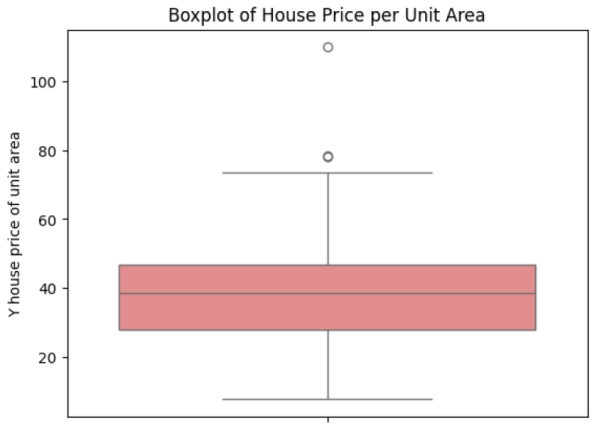


Figure 5.4: Boxplot of House Price per Unit Area

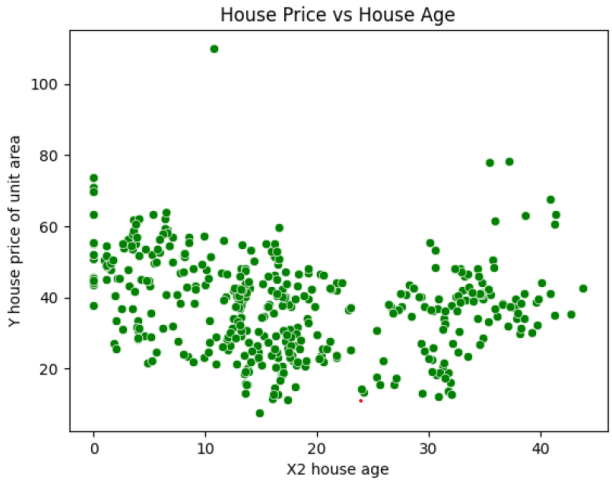


Figure 5.5: House Price vs House Age

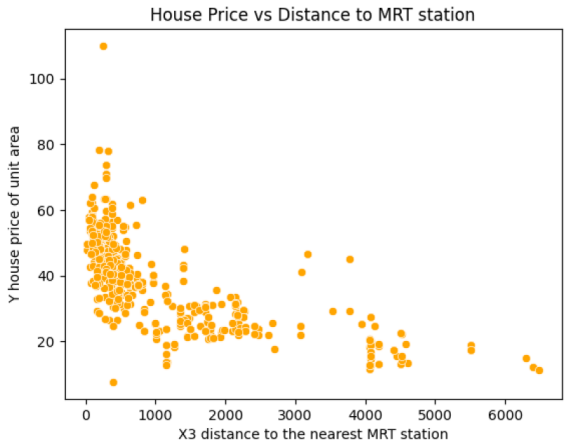


Figure 5.6: House Price vs Distance to MRT station

5.15 Design Matrix X and Output Y

```
# Using normalized features (Min-Max)
x2_norm = (df["X2 house age"] - df["X2 house age"].min()) / (df["X2 house age"].max() - df["X2 house age"].min())
x3_norm = (df["X3 distance to the nearest MRT station"] - df["X3 distance to the nearest MRT station"].min()) / (df["X3 distance to the nearest MRT station"].max() - df["X3 distance to the nearest MRT station"].min())

m = len(df)
X = np.c_[np.ones(m), x2_norm, x3_norm]  # m    (n+1)
Y = df["Y house price of unit area"].values.reshape(-1, 1)  # m    1

print("X shape:", X.shape)
print("Y shape:", Y.shape)
```

```
X shape: (415, 3)
Y shape: (415, 1)
```

5.16 Normal Equation

```
W_normal_equation = np.linalg.inv(X.T @ X) @ X.T @ Y
print("Parameters (W) from Normal Equation:\n", W_normal_equation)
```

```
Parameters (W) from Normal Equation:
 [[ 49.61767979]
 [ -9.99718231]
 [-46.49889746]]
```

5.17 Gradient Descent

```
alpha = 0.01
iterations = 1000


print("--- Manual Gradient Descent ---")
# Design Matrix X_gd of shape n    m for manual GD
# Use only the two normalized input features
X_gd = np.vstack([x2_norm, x3_norm])   # shape (2, m)
Y_gd = Y.reshape(1, -1)                # shape (1, m)
m_gd = Y_gd.shape[1]

# Initialize
W = np.zeros((1, 2))  # weights for x2, x3
b = 0.0

```

```python
for i in range(iterations):
    Y_hat = np.dot(W, X_gd) + b
    E = Y_hat - Y_gd
    dW = (1/m_gd) * np.dot(E, X_gd.T)
    db = (1/m_gd) * np.sum(E)
    W -= alpha * dW
    b -= alpha * db

print("Final parameters (manual GD):")
print("W:", W, " b:", b)

# TensorFlow GradientTape Gradient Descent
print("\n--- TensorFlow Gradient Descent (GradientTape) ---")
# Prepare data for TensorFlow
# X_tf should be (m, n) and Y_tf should be (m, 1)
X_tf_features = tf.constant(np.c_[x2_norm, x3_norm], dtype=tf.float32) # (m, ↩
    2)
Y_tf_target = tf.constant(Y, dtype=tf.float32)                          # (m, ↩
    1)

# Initialize TF Variables for W and b
W_tf = tf.Variable(tf.zeros((2, 1), dtype=tf.float32)) # n x 1 for TF matmul (↩
    X @ W)
b_tf = tf.Variable(0.0, dtype=tf.float32)

optimizer = tf.keras.optimizers.SGD(learning_rate=alpha)

for epoch in range(iterations):
    with tf.GradientTape() as tape:
        Y_pred = tf.matmul(X_tf_features, W_tf) + b_tf
        loss = tf.reduce_mean(tf.square(Y_pred - Y_tf_target)) # Mean Squared ↩
            Error

    # Compute gradients
    grads = tape.gradient(loss, [W_tf, b_tf])

    # Apply gradients
    optimizer.apply_gradients(zip(grads, [W_tf, b_tf]))

print("Final parameters (TF GD):")
print("W:", W_tf.numpy().T, " b:", b_tf.numpy()) # Transpose W_tf for ↩
    consistent output format
```

```
--- Manual Gradient Descent ---
Final parameters (manual GD):
W: [[  3.33859658 -10.38361448]]  b: 37.78556232358561

--- TensorFlow Gradient Descent (GradientTape) ---
```

```
Final parameters (TF GD):
W: [[ -2.3015294 -21.340294 ]]  b: 42.05744
```

5.18 Linear Regression Class

```python
class MyLinearRegression:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs
        self.W = None
        self.b = None

    def fit(self, X, Y):
        """
        X: Input features, shape (m, n)
        Y: Target vector, shape (m, 1)
        """
        m, n = X.shape
        # Initialize W (n, 1) and b (scalar)
        self.W = np.zeros((n, 1))
        self.b = 0.0

        for i in range(self.epochs):
            # Prediction: Y_hat = X @ W + b (m, 1)
            Y_hat = X @ self.W + self.b

            # Error
            E = Y_hat - Y # (m, 1)

            # Gradients
            dW = (1/m) * (X.T @ E) # (n, 1)
            db = (1/m) * np.sum(E) # scalar

            # Update parameters
            self.W -= self.lr * dW
            self.b -= self.lr * db
        print(f"Final W (class): {self.W.T}") # Transpose for consistent print
        print(f"Final b (class): {self.b}")

    def predict(self, X):
        """
        X: Input features for prediction, shape (k, n)
        Returns: Predictions, shape (k, 1)
        """
        if self.W is None or self.b is None:
            raise Exception("Model has not been trained yet. Call fit() first.")
        return X @ self.W + self.b
```

```python
# Example usage for the class
print("\n--- Linear Regression Class Usage ---")
# Use the normalized features (x2_norm, x3_norm)
X_class = np.c_[x2_norm, x3_norm] # (m, 2)
Y_class = Y                       # (m, 1)

model = MyLinearRegression(lr=0.01, epochs=1000)
model.fit(X_class, Y_class)  # Pass features and target

# Make predictions
preds = model.predict(X_class)
print("Predictions shape:", preds.shape)
print("First 5 predictions:\n", preds[:5])
```

```
Final W (class): [[  3.33859658 -10.38361448]]
Final b (class): 37.78556232358561
Predictions shape: (415, 1)
First 5 predictions:
 [[34.90425316]
 [36.19690184]
 [37.76634713]
 [37.76634713]
 [36.70327685]]
```

Assignment 6
Building Machine Learning Models with Scikit-learn

Problem Statement 1: Classification on the Iris Dataset

1. Load the Iris dataset from Seaborn's dataset module into a Pandas DataFrame.
2. Split the dataset into training and test data.
3. Train and evaluate the following classification models:
   (a) K-Nearest Neighbors (KNN)
   (b) Gaussian Naive Bayes
   (c) Decision Tree
   (d) Random Forest
   (e) Support Vector Machine (SVM)
4. For each model:
   (a) Print the labels predicted by the model on the test data and compare them with the actual labels.
   (b) Form and display the confusion matrix using the test data.

Problem Statement 2: K-Means Clustering on the Wine Dataset

5. Load the Wine dataset from Scikit-learn's dataset module into a Pandas DataFrame.
6. Preprocess the data by selecting relevant features and handling any missing values if present.
7. Apply K-Means clustering to the dataset with the number of clusters equal to the number of unique wine classes in the dataset.
8. Print the cluster labels assigned to each data point by the K-Means algorithm.
9. Compare the cluster labels with the actual wine class labels by calculating the accuracy of the clustering.
10. Visualize the clusters using a scatter plot or pairplot with different colors representing different clusters.

Problem Statement 3: Linear Regression on the California Housing Dataset

11. Load the California Housing dataset from Scikit-learn's dataset module into a Pandas DataFrame.
12. Preprocess the data by selecting relevant features and handling any missing values if present.
13. Split the dataset into training and test data.
14. Train a Linear Regression model on the training data.
15. Evaluate the model by predicting housing prices on the test data and calculating performance metrics such as Mean Squared Error (MSE) and R-squared.
16. Print the coefficients of the model to understand the influence of each feature on the target variable.

6.1 Loading the Iris Dataset

```
import seaborn as sns
import pandas as pd

# Load Iris dataset into a DataFrame
iris = sns.load_dataset('iris')
print(iris.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

6.2 Splitting the Dataset into Training and Test Sets

```
from sklearn.model_selection import train_test_split

# Split the dataset into training and test data
X = iris.drop('species', axis=1) # Features are all columns except 'species'
y = iris['species'] # Target is the 'species' column

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ↩
    random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (120, 4)
Shape of X_test: (30, 4)
Shape of y_train: (120,)
Shape of y_test: (30,)
```

6.3 Training Classification Models

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# Initialize models
knn = KNeighborsClassifier(n_neighbors=3)
gnb = GaussianNB()
dtc = DecisionTreeClassifier(random_state=42)
rfc = RandomForestClassifier(random_state=42)
svm = SVC(random_state=42)
# Train and evaluate models
model_performance = {}
# K-Nearest Neighbors
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_pred)
model_performance['K-Nearest Neighbors'] = knn_accuracy
# Gaussian Naive Bayes
gnb.fit(X_train, y_train)
gnb_pred = gnb.predict(X_test)
gnb_accuracy = accuracy_score(y_test, gnb_pred)
model_performance['Gaussian Naive Bayes'] = gnb_accuracy
# Decision Tree
dtc.fit(X_train, y_train)
dtc_pred = dtc.predict(X_test)
dt_accuracy = accuracy_score(y_test, dtc_pred)
model_performance['Decision Tree'] = dt_accuracy
# Random Forest
rfc.fit(X_train, y_train)
rfc_pred = rfc.predict(X_test)
rf_accuracy = accuracy_score(y_test, rfc_pred)
model_performance['Random Forest'] = rf_accuracy
# Support Vector Machine
svm.fit(X_train, y_train)
svm_pred = svm.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)
model_performance['Support Vector Machine'] = svm_accuracy
# Create a DataFrame to display the results
```

```
performance_df = pd.DataFrame(model_performance.items(), columns=['Model', 'Accuracy'])
print(performance_df)
```

```
                    Model  Accuracy
0       K-Nearest Neighbors       1.0
1      Gaussian Naive Bayes       1.0
2             Decision Tree       1.0
3             Random Forest       1.0
4  Support Vector Machine         1.0
```

6.4 Model Evaluation

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Dictionary of trained models
trained_models = {
    'K-Nearest Neighbors': knn,
    'Gaussian Naive Bayes': gnb,
    'Decision Tree': dtc,
    'Random Forest': rfc,
    'Support Vector Machine': svm
}
# Iterate through each model
for model_name, model in trained_models.items():
    print(f"--- {model_name} ---")
    # Get predictions
    y_pred = model.predict(X_test)
    # Print predicted and actual labels
    print("\nPredicted Labels:")
    print(y_pred)
    print("\nActual Labels:")
    print(y_test.values)

    cm = confusion_matrix(y_test, y_pred)
    print("\nConfusion Matrix:")

    # Plotting the confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=model.classes_, yticklabels=model.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for {model_name}')
    plt.show()
    print("-" * (len(model_name) + 8)) # Separator
```

```
--- K-Nearest Neighbors ---

Predicted Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Actual Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Confusion Matrix:
```

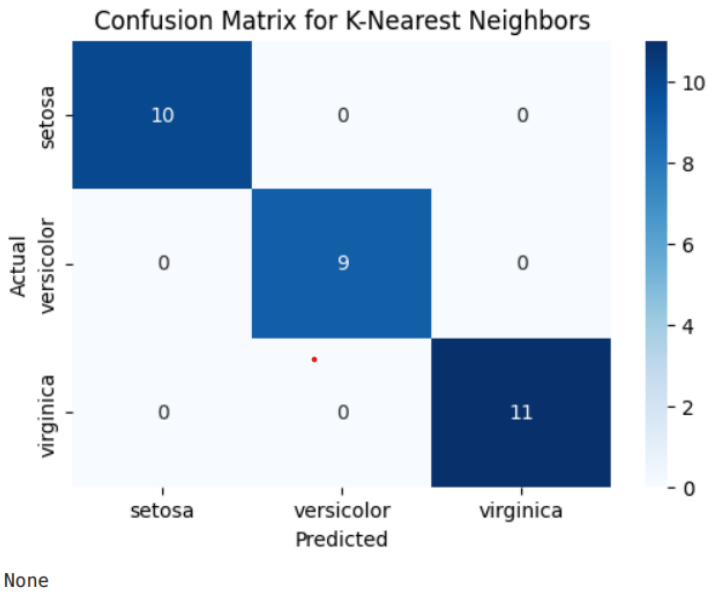


Figure 6.7: Confusion Matrix for K-Nearest Neighbors

```
----------------------------
--- Gaussian Naive Bayes ---

Predicted Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Actual Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Confusion Matrix:
```

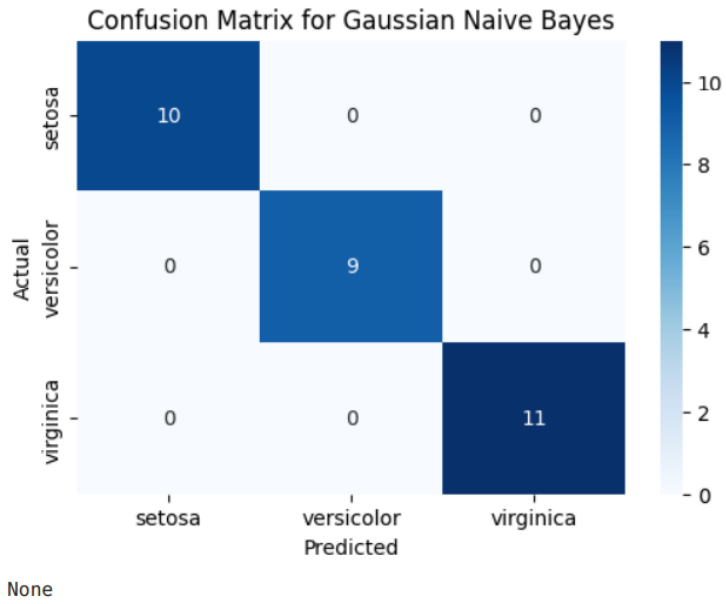


Figure 6.8: Confusion Matrix for Gaussian Naive Bayes

```
----------------------------
--- Decision Tree ---

Predicted Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Actual Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Confusion Matrix:
```

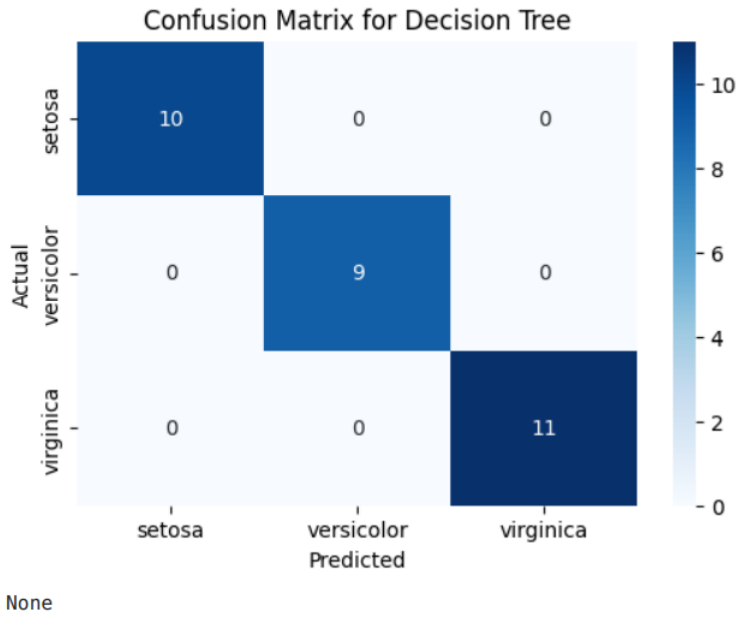


Figure 6.9: Confusion Matrix for Decision Tree

```
----------------------
--- Random Forest ---

Predicted Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Actual Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Confusion Matrix:
```

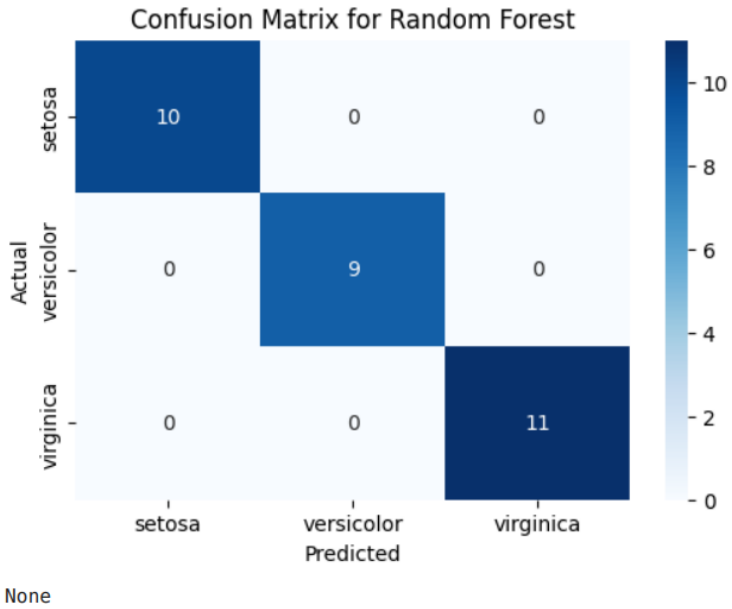


Figure 6.10: Confusion Matrix for Random Forest

```
----------------------
--- Support Vector Machine ---

Predicted Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Actual Labels:
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

Confusion Matrix:
```

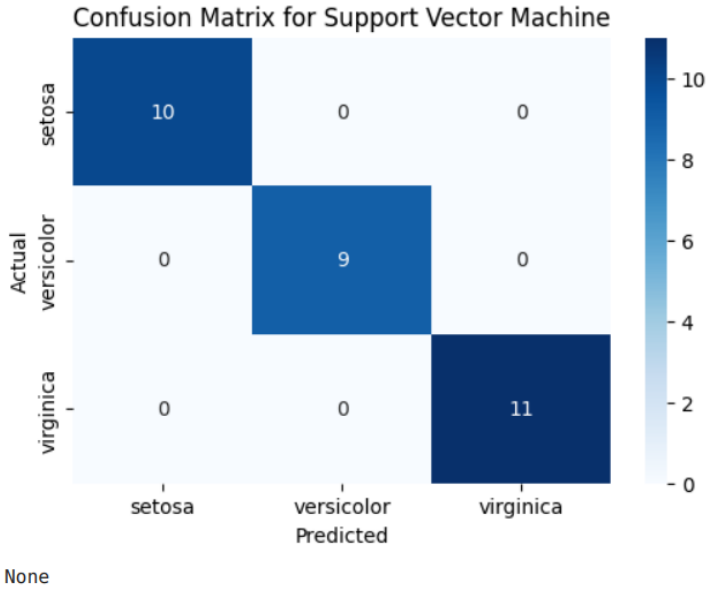


Figure 6.11: Confusion Matrix for Support Vector Machine

6.5 Loading the Wine Dataset

```
from sklearn.datasets import load_wine
import pandas as pd

# Load the Wine dataset
wine = load_wine()

# Create a DataFrame
wine_df = pd.DataFrame(data=wine.data, columns=wine.feature_names)

# Add the target variable to the DataFrame
wine_df['target'] = wine.target

# Display the first few rows of the DataFrame
print(wine_df.head())
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline  target
0                          3.92   1065.0       0
1                          3.40   1050.0       0
2                          3.17   1185.0       0
3                          3.45   1480.0       0
4                          2.93    735.0       0
```

6.6 Data Preprocessing

```
# Check for missing values
missing_values = wine_df.isnull().sum()
print("Missing values in each column:")
print(missing_values)

```

```
# The Wine dataset is known to be clean and not require extensive feature selection for this task.
# Based on the output, there are no missing values in this dataset.
# For this task, we will use all features except the 'target' column for clustering.
```

```
Missing values in each column:
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
target                          0
dtype: int64
```

6.7 Applying K-Means Clustering

```
from sklearn.cluster import KMeans

# Determine the number of unique wine classes
n_clusters = wine_df['target'].nunique()
print(f"Number of unique wine classes: {n_clusters}")

# Prepare features for clustering (excluding the target column)
X_wine = wine_df.drop('target', axis=1)

# Apply K-Means clustering
# n_init='auto' (default for scikit-learn >= 1.4) or set explicitly
kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init='auto')
wine_df['cluster'] = kmeans.fit_predict(X_wine)

# Display the first few rows with the assigned cluster
print(wine_df.head())

# Display the value counts for each cluster
print("\nValue counts for each cluster:")
print(wine_df['cluster'].value_counts())
```

```
Number of unique wine classes: 3
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline  target  cluster
0                          3.92   1065.0       0        1
1                          3.40   1050.0       0        1
2                          3.17   1185.0       0        1
3                          3.45   1480.0       0        1
4                          2.93    735.0       0        0

Value counts for each cluster:
cluster
0    69
1    62
2    47
Name: count, dtype: int64
```

6.8 Cluster Label Assignment

```
1 # Print the cluster labels
2 print("Cluster labels assigned by K-Means:")
3 print(wine_df['cluster'].values)
```

```
Cluster labels assigned by K-Means:
[1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 0 0
 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 2 0 2 2 0 2 2 0 0 0 2 2 1
 0 2 2 2 0 2 2 0 0 2 2 2 2 2 0 0 2 2 2 2 2 0 0 2 0 2 0 2 2 2 0 2 2 2 2 0 2
 2 0 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 0 2 2 0 0 0 0 2 2 2 0 0 2 2 0 0 2 0
 0 2 2 2 2 0 0 0 2 0 0 0 2 0 2 0 0 2 0 0 0 0 2 2 0 0 0 0 0 2]
```

6.9 Accuracy Comparison with Actual Labels

```
from sklearn.metrics import confusion_matrix
from scipy.optimize import linear_sum_assignment
import numpy as np

# Get the actual and predicted labels
actual_labels = wine_df['target']
predicted_labels = wine_df['cluster']

# Create a confusion matrix to see the mapping between clusters and actual classes
confusion_mat = confusion_matrix(actual_labels, predicted_labels)
print("Confusion Matrix (Actual vs. Cluster):")
print(confusion_mat)

# Find the best mapping between cluster labels and actual labels using the Hungarian algorithm
# We want to maximize the sum of the diagonal elements of the confusion matrix, so we minimize the negative
row_ind, col_ind = linear_sum_assignment(-confusion_mat)

# Calculate the accuracy based on the best mapping
# The sum of the correctly assigned samples is the sum of the confusion matrix elements at (row_ind, col_ind)
correctly_assigned = confusion_mat[row_ind, col_ind].sum()
total_samples = len(actual_labels)
accuracy = correctly_assigned / total_samples

print(f"\nAccuracy of K-Means clustering (after finding best mapping): {accuracy:.4f}")
```

```
Confusion Matrix (Actual vs. Cluster):
[[13 46  0]
 [20  1 50]
 [29  0 19]]

Accuracy of K-Means clustering (after finding best mapping): 0.7022
```

6.10 Cluster Visualization

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# It's often good practice to scale data before clustering for visualization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_wine)
```

```
wine_df_scaled = pd.DataFrame(X_scaled, columns=X_wine.columns)
wine_df_scaled['cluster'] = wine_df['cluster']
wine_df_scaled['target'] = wine_df['target'] # Keep target for comparison

# Visualize the clusters using a scatter plot
# Choosing two features for visualization, you can change these
feature1 = 'alcohol'
feature2 = 'color_intensity'

plt.figure(figsize=(10, 7))
sns.scatterplot(data=wine_df_scaled, x=feature1, y=feature2, hue='cluster', palette='viridis', s=100, alpha=0.8)
plt.title(f'K-Means Clustering of Wine Dataset ({feature1} vs {feature2})')
plt.xlabel(f'{feature1} (Scaled)')
plt.ylabel(f'{feature2} (Scaled)')
plt.legend(title='Cluster')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Visualize actual classes for comparison
plt.figure(figsize=(10, 7))
sns.scatterplot(data=wine_df_scaled, x=feature1, y=feature2, hue='target', palette='viridis', s=100, alpha=0.8)
plt.title(f'Actual Wine Classes ({feature1} vs {feature2})')
plt.xlabel(f'{feature1} (Scaled)')
plt.ylabel(f'{feature2} (Scaled)')
plt.legend(title='Actual Class')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()


# Alternatively, you could use a pairplot for a more comprehensive visualization (can be slow for many features)
# print("\nGenerating Pairplot (this might take a moment)...")
# sns.pairplot(wine_df_scaled.drop('target', axis=1), hue='cluster', palette='viridis', diag_kind='kde')
# plt.suptitle('Pairplot of Wine Dataset with K-Means Clusters', y=1.02)
# plt.show()
```

Here are the visualizations for the K-Means clustering of the Wine dataset:

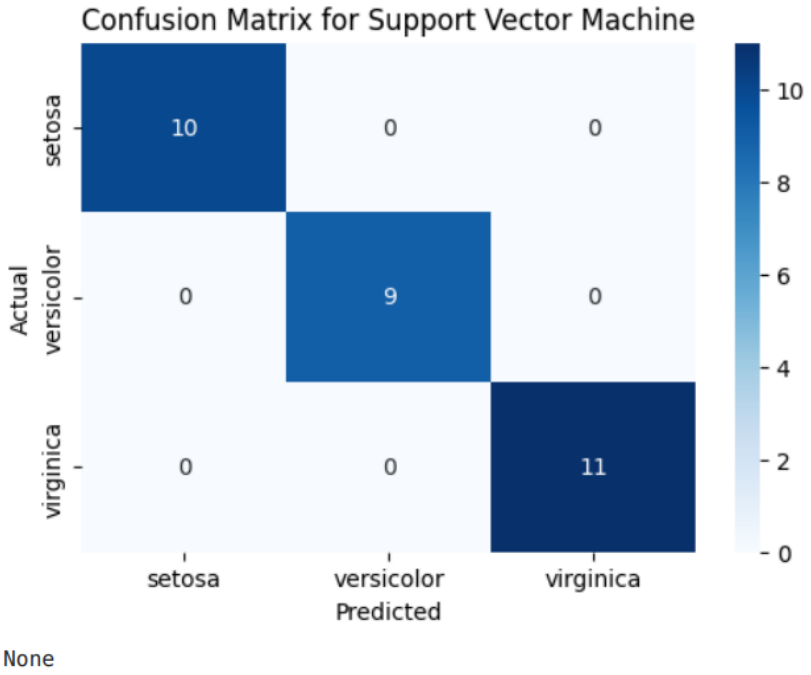


Figure 6.12: K-Means Clustering of Wine Dataset (Alcohol vs Color Intensity)

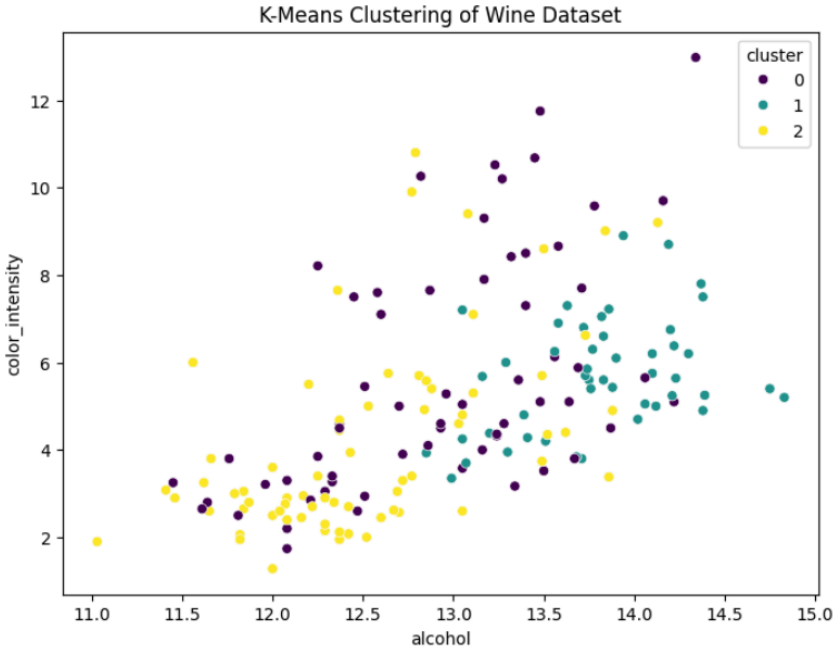


Figure 6.13: Actual Wine Classes (Alcohol vs Color Intensity) for Comparison

6.11 Loading the California Housing Dataset

```
1 from sklearn.datasets import fetch_california_housing
2 import pandas as pd
3
4 # Load the California Housing dataset
5 housing = fetch_california_housing()
```

```
6
7  # Create a DataFrame
8  housing_df = pd.DataFrame(data=housing.data, columns=housing.feature_names)
9
10 # Add the target variable (median house value) to the DataFrame
11 housing_df['MedHouseVal'] = housing.target
12
13 # Display the first few rows of the DataFrame
14 print(housing_df.head())
```

```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  MedHouseVal
0    -122.23        4.526
1    -122.22        3.585
2    -122.24        3.521
3    -122.25        3.413
4    -122.25        3.422
```

6.12 Data Preprocessing

```
1  # Check for missing values
2  missing_values = housing_df.isnull().sum()
3  print("Missing values in each column:")
4  print(missing_values)
5  # The California Housing dataset is known to be clean and not have missing ↩
       values.
6  # For this task, we will use all features for now.
7  # Feature scaling is often beneficial for regression models, but we can ↩
       address this if needed.
```

```
Missing values in each column:
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
```

```
Latitude       0
Longitude      0
MedHouseVal    0
dtype: int64
```

6.13 Splitting the Dataset into Training and Test Sets

```
from sklearn.model_selection import train_test_split
# Define features (X) and target (y)
X_housing = housing_df.drop('MedHouseVal', axis=1)
y_housing = housing_df['MedHouseVal']
# Split the dataset into training and test data
X_train_housing, X_test_housing, y_train_housing, y_test_housing = train_test_split(
    X_housing, y_housing, test_size=0.2, random_state=42
)
print("Shape of X_train_housing:", X_train_housing.shape)
print("Shape of X_test_housing:", X_test_housing.shape)
print("Shape of y_train_housing:", y_train_housing.shape)
print("Shape of y_test_housing:", y_test_housing.shape)
```

```
Shape of X_train_housing: (16512, 8)
Shape of X_test_housing: (4128, 8)
Shape of y_train_housing: (16512,)
Shape of y_test_housing: (4128,)
```

6.14 Training the Linear Regression Model

```
from sklearn.linear_model import LinearRegression
# Initialize the Linear Regression model
linear_reg_model = LinearRegression()
# Train the model on the training data
linear_reg_model.fit(X_train_housing, y_train_housing)
print("Linear Regression model trained successfully.")
```

```
Linear Regression model trained successfully.
```

6.15 Model Evaluation (MSE & R-Squared)

```
from sklearn.metrics import mean_squared_error, r2_score
# Predict housing prices on the test data
y_pred_housing = linear_reg_model.predict(X_test_housing)
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test_housing, y_pred_housing)
print(f"Mean Squared Error (MSE): {mse:.4f}")
# Calculate R-squared
```

```
r2 = r2_score(y_test_housing, y_pred_housing)
print(f"R-squared: {r2:.4f}")
```

```
Mean Squared Error (MSE): 0.5559
R-squared: 0.5758
```

6.16 Interpreting Model Coefficients

```
# Print the coefficients of the model
print("Model Coefficients:")
for feature, coef in zip(X_housing.columns, linear_reg_model.coef_):
    print(f"{feature}: {coef:.4f}")

# Optionally, print the intercept as well
print(f"\nIntercept: {linear_reg_model.intercept_:.4f}")
```

```
Model Coefficients:
MedInc: 0.4487
HouseAge: 0.0097
AveRooms: -0.1233
AveBedrms: 0.7831
Population: -0.0000
AveOccup: -0.0035
Latitude: -0.4198
Longitude: -0.4337

Intercept: -37.0233
```

Assignment 7
Image Classification Using CNNs

Problem Statement

Convolutional Neural Network (CNN) using TensorFlow Keras to classify images from the CIFAR-10 dataset. CIFAR-10 contains 32×32 RGB images across 10 classes (airplane, car, bird, cat, deer, dog, frog, horse, ship, truck).

1. Load the CIFAR-10 dataset using tensorflow.keras.datasets.cifar10.load data().
2. Normalize the pixel values of the images to the range [0,1]
3. Examine the shapes of the training and test sets.
4. Build a CNN model using the Sequential API with the following specifications:
   - Atleast two Conv2D layers with ReLU activation.
   - At least two MaxPooling2D layers.
   - A Flatten layer.
   - One or more Dense hidden layers with ReLU activation.
   - An output Dense layer with 10 neurons and softmax activation
5. Compile the model with:
   - Optimizer: adam.
   - Loss: sparse categorical crossentropy.
   - Metrics: accuracy.
6. Train the model for at least 10 epochs using a suitable batch size and a validation split of 0.1
7. Evaluate the model on the test set and report the test accuracy.
8. Take 5 random images from the test set and make predictions. Display each image along with the predicted and true labels using matplotlib
9. Experiment by modifying the number of convolutional filters, adding Dropout or BatchNormalization layers, and observe changes in performance. Write a short conclusion.

7.1 Load the CIFAR-10 dataset

```
from tensorflow.keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-
10-python.tar.gz
170498071/170498071  6s 0us/step
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

7.2 Normalize the pixel values

```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
print("x_train shape after normalization:", x_train.shape)
print("x_test shape after normalization:", x_test.shape)
```

```
x_train shape after normalization: (50000, 32, 32, 3)
x_test shape after normalization: (10000, 32, 32, 3)
```

7.3 Examine the shapes

```
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

7.4 Build the CNN model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
model = Sequential([
Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
MaxPooling2D((2, 2)),
Conv2D(64, (3, 3), activation='relu'),
MaxPooling2D((2, 2)),
In [1]:
In [3]:
In [4]:
In [5]:
Flatten(),
Dense(64, activation='relu'),
Dense(10, activation='softmax')
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147,520
dense_1 (Dense)	(None, 10)	650

Total params: 167,562 (654.54 KB)

Trainable params: 167,562 (654.54 KB)

Non-trainable params: 0 (0.00 B)

7.5 Compile the model

```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

```
alcohol malic_acid ash alcalinity_of_ash magnesium total_phenols flavanoids
0 14.23 1.71 2.43 15.6 127.0 2.80 3.06
1 13.20 1.78 2.14 11.2 100.0 2.65 2.76
2 13.16 2.36 2.67 18.6 101.0 2.80 3.24
3 14.37 1.95 2.50 16.8 113.0 3.85 3.49
4 13.24 2.59 2.87 21.0 118.0 2.80 2.6
```

7.6 Train the model

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32, ↩
    validation_split
```

```
Epoch 1/10
1407/1407  57s 40ms/step - accuracy: 0.5838 - loss: 1.1879
- val_accuracy: 0.6170 - val_loss: 1.0978
Epoch 2/10
1407/1407  80s 39ms/step - accuracy: 0.6400 - loss: 1.0290
- val_accuracy: 0.6550 - val_loss: 1.0003
Epoch 3/10
1407/1407  57s 40ms/step - accuracy: 0.6738 - loss: 0.9442
- val_accuracy: 0.6620 - val_loss: 0.9609
Epoch 4/10
1407/1407  81s 39ms/step - accuracy: 0.6957 - loss: 0.8761
- val_accuracy: 0.6786 - val_loss: 0.9272
Epoch 5/10
1407/1407  84s 41ms/step - accuracy: 0.7178 - loss: 0.8138
- val_accuracy: 0.6858 - val_loss: 0.8987
Epoch 6/10
1407/1407  81s 40ms/step - accuracy: 0.7332 - loss: 0.7662
- val_accuracy: 0.6724 - val_loss: 0.9704
Epoch 7/10
1407/1407  82s 40ms/step - accuracy: 0.7469 - loss: 0.7209
- val_accuracy: 0.6932 - val_loss: 0.9190
Epoch 8/10
1407/1407  57s 40ms/step - accuracy: 0.7598 - loss: 0.6809
- val_accuracy: 0.6968 - val_loss: 0.9092
Epoch 9/10
1407/1407  81s 40ms/step - accuracy: 0.7766 - loss: 0.6466
- val_accuracy: 0.6952 - val_loss: 0.9055
Epoch 10/10
```

```
1407/1407  82s 40ms/step - accuracy: 0.7847 - loss: 0.6084
- val_accuracy: 0.6802 - val_loss: 0.9388
```

7.7 Evaluate the model

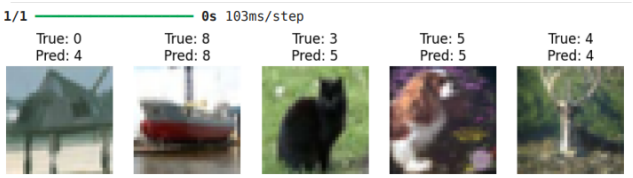
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print("Test accuracy:", accuracy)
```

```
Test accuracy: 0.5612000226974487
```

7.8 Predict on random images

```

import numpy as np
import matplotlib.pyplot as plt
# Take 5 random images from the test set
random_indices = np.random.choice(len(x_test), 5)
random_images = x_test[random_indices]
true_labels = y_test[random_indices]
# Make predictions
predictions = model.predict(random_images)
predicted_labels = np.argmax(predictions, axis=1)
In [13]:
In [15]:
# Display images with predictions and true labels
plt.figure(figsize=(10, 5))
for i in range(5):
plt.subplot(1, 5, i + 1)
plt.imshow(random_images[i])
plt.title(f"True: {true_labels[i][0]}\nPred: {predicted_labels[i]}")
plt.axis('off')
plt.show()
```



7.9 Modify and Observe performance

```
1
2  from tensorflow.keras.models import Sequential
3  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, ↩
       Dropout
4  model_experiment = Sequential([
5  Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)),
6  BatchNormalization(),
7  MaxPooling2D((2, 2)),
8  Conv2D(128, (3, 3), activation='relu'),
9  BatchNormalization(),
10 MaxPooling2D((2, 2)),
11 Flatten(),
12 Dense(128, activation='relu'),
13 Dropout(0.5),
14 Dense(10, activation='softmax')
15 ])
16 model_experiment.summary()
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 30, 30, 64)	1,792
batch_normalization (BatchNormalization)	(None, 30, 30, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 64)	0
conv2d_3 (Conv2D)	(None, 13, 13, 128)	73,856
batch_normalization_1 (BatchNormalization)	(None, 13, 13, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_2 (Dense)	(None, 128)	589,952
dropout (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

```
Total params: 667,658 (2.55 MB)
Trainable params: 667,274 (2.55 MB)
Non-trainable params: 384 (1.50 KB)
```

Assignment 8
Predictive Keyboard using LSTM

Problem Statement

Implement a predictive keyboard using an LSTM model. The model should learn from a given text corpus (e.g., sentences on democracy and modern man) and generate the next words for a given input seed text.

Tasks

1. Prepare a text corpus of 50–100 lines based on themes such as democracy and modern man.
2. Perform tokenization using Tokenizer() from tensorflow.keras.preprocessing.text.
3. Generate n-gram input sequences and pad them to a uniform length using pad sequences().
4. Split the sequences into predictors (X) and labels (y), converting the labels to onehot encoding using to categorical().
5. Build an LSTM model using the Sequential API with the following layers:
   - An Embedding layer.
   - An LSTM layer with 100 units.
   - A Dense output layer with softmax activation.
6. Compile the model using:
   - Optimizer: adam.
   - Loss: sparse categorical _crossentropy.
   - Metrics: accuracy.
7. Train the model for at least 200 epochs and display the training accuracy.
8. . Implement a function predict next_words(seed _text, num_words) that generates the next num_words words for a given seed text.
9. Test the predictive function with at least 3 different seed inputs and display the outputs.
10. Experiment with different LSTM units, embedding sizes, or corpus sizes and report your observations.

8.1 Prepare a text corpus

```
corpus = """Democracy, a system where the people hold the power,
A beacon of hope in freedom's finest hour.
Modern man, with technology in hand,
Navigating complexities across the land.
In democratic states, voices rise and fall,
Debating futures, answering freedom's call.
Information flows, a digital tide,
Connecting minds, with nowhere left to hide.
Yet challenges loom, in this digital age,
Misinformation's grip, turning history's page.
The truths and the falsehoods, in battles hard-fought.
Democracy's strength lies in participation deep,
Where citizens engage, secrets they don't keep.
From local polls to global affairs grand,
Modern man's influence, across the land.
Modern man distracted, by screens and by noise,
Losing the connection to freedom's vital voice.
The promise of equality, a core democratic aim,
For every soul to flourish, fanning freedom's flame.
Modern man strives onward, for justice and for right,
Though systemic barriers, may dim the future's light.
Economic disparities, a widening divide,
Testing democracy's resilience, where values reside.
Modern man grapples, with systems complex and vast,
Seeking solutions, for futures built to last.
The environment calls, a crisis we must face,
Democracy's response, for the human race.
Global connections tighten, in an interconnected sphere,
Democracy's influence, dispelling doubt and fear.
Modern man collaborates, across borders and beyond,
Building bridges stronger, where understanding's found.
The weight of history, on democracy's long road,
With triumphs and failures, a heavy, carried load.
That paved the path to freedom, again and again.
So let us engage, with purpose and with care."""
print(corpus)
```

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: Democracy, a system where the people hold the power,
 A beacon of hope in freedom's finest hour.
 Modern man, with technology in hand,
 Navigating complexities across the land.
 In democratic states, voices rise and fall,
```

Debating futures, answering freedom's call.
Information flows, a digital tide,
Connecting minds, with nowhere left to hide.
Yet challenges loom, in this digital age,
Misinformation's grip, turning history's page.
Modern man must discern, with critical thought,
The truths and the falsehoods, in battles hard-fought.
Democracy's strength lies in participation deep,
Where citizens engage, secrets they don't keep.
From local polls to global affairs grand,
Modern man's influence, across the land.
Modern man's impact, on the planet's fragile health,
Demanding action now, for collective wealth.
Global connections tighten, in an interconnected spher truly last.
In innovation's glow, and scientific stride,
Modern man advances, with nowhere left to hide.
Democracy adapts, to changes ever new,
Embracing progress, in all that we pursue.
The power of the vote, a sacred, precious thing,
The voice of every citizen, on which futures swing.
Modern man remembers, the struggles and the pain,
Shaping public discourse, sealing freedom's fate.
Modern man consumes, a constant information flow,
Discerning truth from fiction, helping knowledge grow.

8.2 Tokenization

```
from tensorflow.keras.preprocessing.text import Tokenizer
 # Create a Tokenizer object
 tokenizer = Tokenizer()
 # Fit the tokenizer on the corpus
 tokenizer.fit_on_texts([corpus])
 # Get the word index
 word_index = tokenizer.word_index
 # Print the word index
 print(word_index)
```

```
{'the': 1, 'and': 2, 'modern': 3, 'a': 4, 'man': 5, 'in': 6, 'to'': 29,
'left': 30, 'hide': 31, 'challenges': 32, 'must': 33, 'citizens': 34,
'engage': 35, 'from': 36, 'global': 37, "man's": 38, 'influence': 39,
'but': 40, 'by': 41, 'voice': 42, 'every': 43, 'flame': 44, 'onward': 45,
'light': 46, 'last': 47, 'stronger': 48, "understanding's": 49, 'long': 50,
'shaping': 51, 'ever': 52, 'that': 53, 'precious': 54, 'again': 55, 'so': 56,
'time': 57, 'stands': 58, 'system': 59, 'hold': 60, 'beacon': 61, 'hope': 62,
'finest': 63, 'hour': 64, 'technology': 65, 'hand': 66, 'navigating': 67,
'complexities': 68, 'states': 69, 'voices': 70, 'rise': 71, 'fall': 72,
'debating': 73, 'answering': 74, 'flows': 75, 'tide': 76, 'connecting': 77,
'minds': 78, 'yet': 79, 'loom', 80, 'this': 81, 'age': 82, "misinformation's":
83, 'grip': 84, 'turning': 85, "history's": 86, 'page': 879, 'vote': 210,
'sacred': 211, 'thing': 212, 'citizen': 213, 'which' : 214, 'swing': 215,
'remembers': 216, 'struggles': 217, 'pain': 218, 'paved': 219}
```

8.3 Pad sequences

```python
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
# Convert the corpus to a list of sentences
sentences = corpus.lower().split('\n')
# Initialize list for input sequences
input_sequences = []
for sentence in sentences:
# Convert sentence to sequence of tokens
token_list = tokenizer.texts_to_sequences([sentence])[0]
# Generate n-grams from the token list
for i in range(1, len(token_list)):
n_gram_sequence = token_list[:i+1]
input_sequences.append(n_gram_sequence)
# Find the maximum sequence length
max_sequence_length = max([len(x) for x in input_sequences])
# Pad the sequences to the maximum length
padded_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length,↩
    padding
# Print the padded sequences
print(padded_sequences)
```

```
[[  0   0   0 ...   0  21   4]
[  0   0   0 ...  21   4  59]
[  0   0   0 ...   4  59  14]
...
[  0   0   0 ...  15 311 312]
```

```
[  0   0   0 ... 311 312   2]
[  0   0  19 ... 312   2 313]]
```

8.4 Build the CNN model

```
from tensorflow.keras.utils import to_categorical
# Split the sequences into predictors (X) and labels (y)
X = padded_sequences[:, :-1]
y = padded_sequences[:,-1]
# Convert labels to one-hot encoding
In [3]:
In [4]:
y = to_categorical(y, num_classes=len(tokenizer.word_index) + 1)
# Print the shapes of X and y
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

Shape of X: (449, 9) Shape of y: (449, 314)

8.5 Split predictors and labels

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
# Define the vocabulary size
vocab_size = len(tokenizer.word_index) + 1
# Build the model
model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(LSTM(100))
model.add(Dense(vocab_size, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['↩
    accuracy'
# Print the model summary
model.summary()
```

Model: "sequential"

Table 8.1: Summary of the LSTM Model Architecture

Layer (type)	**Output Shape**	**Param #**
$\text{embedding}_1(Embedding)$	?	0 (unbuilt)
$\text{lstm}_1(LSTM)$	?	0 (unbuilt)
$\text{dense}_1(Dense)$	?	0 (unbuilt)
	Total params:	0 (0.00 B)
	Trainable params:	0 (0.00 B)
	Non-trainable params:	0 (0.00 B)

8.6 Build LSTM model

```
history = model.fit(X, y, epochs=200, verbose=1)
 # Display the training accuracy
 In [6]:
 In [9]:
print("Training Accuracy:", history.history['accuracy'][-1])
```

```
Epoch 1/200
 15/15  0s 12ms/step - accuracy: 0.8171 - loss: 1.2267
 Epoch 2/200
 15/15  0s 11ms/step - accuracy: 0.8426 - loss: 1.1105
 Epoch 3/200
 15/15  0s 12ms/step - accuracy: 0.8614 - loss: 1.0901
 Epoch 4/200
 15/15  0s 13ms/step - accuracy: 0.8695 - loss: 1.0458
 Epoch 5/200
 15/15  0s 12ms/step - accuracy: 0.8665 - loss: 1.0313
 Epoch 6/200
 15/15  0s 12ms/step - accuracy: 0.8833 - loss: 1.0065
 Epoch 7/200
 15/15  0s 12ms/step - accuracy: 0.8973 - loss: 1.0294
 Epoch 8/200
 15/15  0s 12ms/step - accuracy: 0.8755 - loss: 0.9438
 Epoch 9/200
 15/15  0s 12ms/step - accuracy: 0.8911 - loss: 0.9426
 Epoch 10/200
 15/15  0s 12ms/step - accuracy: 0.8926 - loss: 0.9107
 Epoch 11/200
 15/15  0s 12ms/step - accuracy: 0.8827 - loss: 0.9020
 Epoch 12/200
 15/15  0s 13ms/step - accuracy: 0.8900 - loss: 0.8491
 Epoch 13/200
 15/15  0s 12ms/step - accuracy: 0.8546 - loss: 0.8929
 Epoch 14/200
 15/15  0s 12ms/step - accuracy: 0.8804 - loss: 0.8560
 Epoch 15/200
 15/15  0s 11ms/step - accuracy: 0.8668 - loss: 0.8593

 Training Accuracy: 0.9220489859580994
```

8.7 Compile the model

```
def predict_next_words(seed_text, num_words):
for _ in range(num_words):
# Tokenize the seed text
token_list = tokenizer.texts_to_sequences([seed_text])[0]
# Pad the sequence
token_list = pad_sequences([token_list], maxlen=max_sequence_length-1
# Predict the next word
predicted = model.predict(token_list, verbose=0)
predicted_word_index = np.argmax(predicted, axis=-1)
# Find the word corresponding to the predicted index
output_word = ""
for word, index in tokenizer.word_index.items():
if index == predicted_word_index:
output_word = word
break
# Append the predicted word to the seed text
seed_text += " " + output_word
return seed_text
# Example usage:
seed_text = "Democracy is a system"
predicted_text = predict_next_words(seed_text, 5)
print(predicted_text)
```

```
Democracy is a system where the people hold the
```

8.8 Train the model

```

# Test with different seed inputs
seed_text_1 = "Modern man"
predicted_text_1 = predict_next_words(seed_text_1, 7)
print(predicted_text_1)
seed_text_2 = "The power of the vote"
predicted_text_2 = predict_next_words(seed_text_2, 6)
print(predicted_text_2)
seed_text_3 = "Challenges remain"
predicted_text_3 = predict_next_words(seed_text_3, 5)
print(predicted_text_3)
```

```
Modern man unites in shared humanity's call call share
The power of the vote a sacred precious thing test test
Challenges remain on democracy's long quest road
```

8.9 Predict & Test the predictive function

```
1
2  from tensorflow.keras.models import Sequential
3  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, ↩
       Dropout
4  model_experiment = Sequential([
5  Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)),
6  BatchNormalization(),
7  MaxPooling2D((2, 2)),
8  Conv2D(128, (3, 3), activation='relu'),
9  BatchNormalization(),
10 MaxPooling2D((2, 2)),
11 Flatten(),
12 Dense(128, activation='relu'),
13 Dropout(0.5),
14 Dense(10, activation='softmax')
15 ])
16 model_experiment.summary()
```

Table 8.2: Summary of the LSTM Model Architecture

Layer (type)	**Output Shape**	**Param #**
embedding_2 (Embedding)	?	0 (unbuilt)
lstm_2 (LSTM)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 In [11]:
 In [12]:
Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
```