1. What is the difference b/w Static and dynamic variable in python?

Static variables are defined inside a class but outside any methods. They remain the same across all instances of the class. Dynamic variables, on the other hand, are defined inside methods and their values can vary from one instance of the class to another.

Example: python class Example:

static_var = "I am static"

def __init__(self, dynamic_var):

self.dynamic_var = dynamic_var

e1 = Example("I am dynamic 1")

e2 = Example("I am dynamic 2")

print(e1.static_var) # Output: I am static

print(e1.dynamic_var) # Output: I am dynamic 1

print(e2.dynamic_var) # Output: I am dynamic

2. Purpose of "pop", "popitem", "clear()" in a dictionary with examples:

- *pop(key):* Removes the specified key and returns the corresponding value.

Example:

Python

```
dict1 = {'a': 1, 'b': 2, 'c': 3}
print(dict1.pop('b')) # Output: 2
print(dict1) # Output: {'a': 1, 'c': 3}
```

- *popitem():* Removes and returns the last (key, value) pair as a tuple.

Example:

Python

```
dict1 = {'a': 1, 'b': 2, 'c': 3}
print(dict1.popitem()) # Output: ('c', 3)
print(dict1) # Output: {'a': 1, 'b': 2}
```

- *clear():* Removes all items from the dictionary.

Example:

Python

Dict1 = {‘a’ :1, ‘b’ : 2, ‘c’ : 3}

Dict1.clear()

Print(Dict1) # output : {}

3 .FrozenSet :-

A frozenset is an immutable version of a set. once created, elements cannot be added and removed .

Ex:

Python

Fs = frozenset([1,2,3,4])

Print(Fs) # output frozenset ([1,2,3,4])

Fs.add (7) that would raise Attribute error.

4.Mutable vs Immutable Data Types in Python:-

Mutable : - can be changed after the creation , for example : list , dict

Immutable:- cannot be changed after the creation , for example : tuple , frozenset

Example :

Python

Mutable

list = [1,2,3]

list.append(4)

print(list) # output : [1,2,3,4]

Immutable

Tup = (1,2,3)

#tup [0] = 4 ; that would arise an Type error

5. __init__:-

The _init_ is constructor in python . it is called when an object is created.

Example :

```
Python
 Class Example :
   def_init_ (self, value):
        self.value = value
    e = Example(12)
         print(e.value) # Output: 12
```

6.Docstring:-

A docstring is a string literal used to document a module, class, function, or method.

Example :-

```
Python
def add (a,b):
“””this function adds 2 numbers “””
return a +b
print(add._doc_) #output : This function adds 2 numbers
```

7. Unit Tests in Python:

Unit test are a way to test individual units of source code to ensure they work as expected.

8. Break / continue / pass in python :-

*break : - Exits the current loop prematurely

*continue :- Skips the rest of the current loop iteration and moves to the next iteration.

*pass :- Does nothing; used as a placeholder.

9. Use of self in Python:

self refers to the instance of the class. Self is used to access methods and variables associated with the instance.

Example:

Python

```
 class Example:
def __init__(self, value):
 self.value = value
def show(self):
 print(self.value)
 e = Example(20)
e.show()  ## output  20
```

10. Global, Protected, and Private Attributes in Python:

- *Global:* Accessible everywhere.

- *Protected:* Indicated by a single underscore (_), should not be accessed outside its module.

 - *Private:* Indicated by a double underscore (__), cannot be accessed outside its class.

 Example:

python

```
class Example:
def __init__(self):
  self.public = "I am public"
    self._protected = "I am protected"
       self.__private = "I am private"
 e = Example()
print(e.public) # Output: I am public
print(e._protected) # Output: I am protected
 print(e.__private) would raise an AttributeError
```

11. Modules in Packages in Python:

Modules :- A single Python file containing functions and definitions.

Package :- A collection of modules in directories that give a package hierarchy.

Example

Python

```
# module: mymodule.py

def greet(name):

  return f"Hello, {name}"


# package: mypackage

# __init__.py

# module1.py

 # module2.py
```

12. Lists and Tuples:

List :- they are mutable , ordered and can contain duplicate elements

Tuple :- They are immutable , ordered can contain duplicate elements.

Difference :

List is mutable whereas tuple is not.

Example :

Python

```
lst = [1, 2, 3]

tup = (1, 2, 3)

lst[0] = 0 # Works

 tup[0] = 0 would raise a TypeError.
```

13. Interpreted Language & Dynamically Typed Language:

Interpreted Language:* Executes code line by line. Example: Python.

Dynamically Typed Language:* Variable types are determined at runtime. Example: Python

Differences:

- Dynamically typed languages do not require variable declarations.
- Interpreted languages often provide immediate feedback.
- Debugging is often easier in interpreted languages.
- Dynamically typed languages determine variable types at runtime.
- Interpreted languages execute code directly.

14. Dict and List Comprehensions:

- List Comprehension: Creates a new list by applying an expression to each element in an existing list.
- Dict Comprehension: Creates a new dictionary by applying an expression to each element in an existing dictionary.

  Example
  Python

```
lst = [1, 2, 3, 4]
new_lst = [x * 2 for x in lst

dict1 = {'a': 1, 'b': 2, 'c': 3}
new_dict = {k: v * 2 for k, v in dict1.items()}
```

15. Decorators in Python:

A decoration is a function that can modifies the behavior of another function.

Example

Python

```
def wrapper():
    print("Something is happening before the function is called.")
   func()
```

```
print("Something is happening after the function is called.")
        return wrapper
@decorator_func
def say_hello():
    print("Hello!")
    say_hello()
```

16. Memory Management in Python:

Memory in Python is managed by the Python Memory Manager. Python uses a private heap to store objects and data structures, with automatic garbage collection to reclaim memory.

17. Lambda in Python:

A lambda function is an anonymous function defined with the lambda keyword. It can have any number of arguments but only one expression.

Example:

```
python
add = lambda x, y: x + y
 print(add(2, 3)) # Output: 5
```

18. split() and join() functions in Python:

Splits – splits a string into a list

Join – joins a list into strings

Example

```
Python
S = “hello  world”
lst = s.split()
```

```
print(lst) # output [“hello” , “world”]


s = "-".join(lst)

print(s) # Output: hello-world
```

Qu.1.2

(b) 1st_Room (starts with digit)

(c)Hundred$ (contains Doller sigh)

(f) Total Marks (contains space)

(g) True (keyword)

Qu.1.4 find the output of the following

```
animal = ['Human', 'cat', 'mat', 'cat', 'rat', 'Human', 'lion']


print(animal.count('Human'))  # output 2
print(animal.index('rat'))  # output 4
print(len(animal))    #output 7
```

qu. 1.5. tuple1 = (10,20,"Apple", 3.4, "a", ["master", "Ji"], ("sita", "geeta", 22), [{"roll_no": 1},{"name":"Navneet"}])

a) print(len(tuple1)) # Output 8

b) print(tuple1[-1][-1]["name"]) # Output: Navneet

c) print(tuple1[-1][0]) / print(tuple1[-1][0]["roll_no"]) # Output: {'roll_no': 1} / 1

d) print(tuple1[-3][1]) # Output: Ji

e) print(tuple1[-2][-1]) # Output: 22

qu.1.6 . program to display the appropriate message as per the color of signal

```
def display_signal_message(color):
    if color.lower() == "red":
        print("Red light - Stop")
    elif color.lower() == "yellow":
        print("Yellow light - Stay")
    elif color.lower() == "green":
        print("Green light - Go")
    else:
        print("Invalid color")
```

Qu.1.7 Write a program to create a simple calculator performing only four basic operations(+,-,/,*) .

```
def add(x, y):
    return x + y


def subtract(x, y):
    return x - y


def multiply(x, y):
    return x * y


def divide(x, y):
    if y != 0:
        return x / y
    else:
        return "Error! Division by zero."
```

```
def calculator():
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    while True:
        choice = input("Enter choice (1/2/3/4): ")

        if choice in ('1', '2', '3', '4'):
            try:
                num1 = float(input("Enter first number: "))
                num2 = float(input("Enter second number: "))
            except ValueError:
                print("Invalid input. Please enter a number.")
                continue

            if choice == '1':
                print(f"{num1} + {num2} = {add(num1, num2)}")
            elif choice == '2':
                print(f"{num1} - {num2} = {subtract(num1, num2)}")
            elif choice == '3':
                print(f"{num1} * {num2} = {multiply(num1, num2)}")
            elif choice == '4':
                print(f"{num1} / {num2} = {divide(num1, num2)}")

            next_calculation = input("Do you want to perform another calculation? (yes/no): ")
            if next_calculation.lower() != 'yes':
```

```
            break
        else:
            print("Invalid input. Please enter a valid choice.")

calculator()
```

1.8. Write a program to find the larger of the three pre- specified numbers using temary operators

```
# Pre-specified numbers
num1 = 10
num2 = 20
num3 = 15

# Using ternary operator to find the largest number
largest = num1 if (num1 > num2 and num1 > num3) else (num2 if num2 > num3 else num3)

print(f"The largest number among {num1}, {num2}, and {num3} is {largest}.")
```

1.9. Write a program to find the factors Of a whole number using a while loop.

```
def find_factors(number):
    i = 1
    print(f"The factors of {number} are:")
    while i <= number:
        if number % i == 0:
            print(i)
```

```
        i += 1

# Example usage
num = int(input("Enter a whole number: "))
find_factors(num)
```

1.10. Write a program to find the sum of all the positive numbers entered by the user. As soon as the user enters a negative number, stop taking in any further input from the user and display the sum .

```
def sum_positive_numbers():
    total_sum = 0
    while True:
        try:
            number = float(input("Enter a number (negative number to stop): "))
            if number < 0:
                break
            total_sum += number
        except ValueError:
            print("Invalid input. Please enter a valid number.")
    print(f"The sum of all positive numbers entered is: {total_sum}")

# Run the function
sum_positive_numbers()
```

1.11. Write a program to find prime numbers between 2 to 100 using nested for loops.

```
# Function to check if a number is prime
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

# Finding prime numbers between 2 and 100
print("Prime numbers between 2 and 100 are:")
for num in range(2, 101):
    if is_prime(num):
        print(num, end=" ")
```

1.12. Write the programs for the following:

Accept and Display Marks of Five Major Subjects

```
# Accept marks for five subjects
marks = []
for i in range(1, 6):
    mark = float(input(f"Enter marks for subject {i}: "))
    marks.append(mark)

# Display the marks
print("Marks for the five subjects are:")
for i, mark in enumerate(marks, start=1):
    print(f"Subject {i}: {mark}")
```

Calculate Sum, Percentage, and Display Grade

```
# Calculate the sum of the marks
```

```
total_marks = sum(marks)

# Calculate the percentage
percentage = total_marks / 5

# Display the total marks and percentage
print(f"Total Marks: {total_marks}")
print(f"Percentage: {percentage}%")

# Determine the grade using match-case
def determine_grade(percentage):
    match percentage:
        case p if p > 85:
            return "A"
        case p if 75 <= p <= 85:
            return "B"
        case p if 50 <= p < 75:
            return "C"
        case p if 30 <= p < 50:
            return "D"
        case _:
            return "Reappear"

grade = determine_grade(percentage)
print(f"Grade: {grade}")
```

VIBGYOR Spectrum Based on Wavelength

```
def vibgyor_spectrum(wavelength):
    match wavelength:
```

```
        case w if 380 <= w < 450:
            return "Violet"
        case w if 450 <= w < 495:
            return "Blue"
        case w if 495 <= w < 570:
            return "Green"
        case w if 570 <= w < 590:
            return "Yellow"
        case w if 590 <= w < 620:
            return "Orange"
        case w if 620 <= w < 750:
            return "Red"
        case _:
            return "Wavelength out of VIBGYOR range"

# Example usage
wavelength = float(input("Enter the wavelength (in nm): "))
color = vibgyor_spectrum(wavelength)
print(f"The color corresponding to the wavelength {wavelength} nm is {color}.")
```

1.14.Consider the gravitational interactions between the Earth, Moon, and Sun in Our solar system.

Given:

mass _earth = 5.972e24 # Mass of Earth in kilograms mass _moon = 7.34767309e22 # Mass of Moon in kilograms mass_sun 1.989e30 # Mass of Sun in kilograms

distance_earth _sun 1.496e11 # Average distance between Earth and Sun in meters distance_ moon_earth 3.844e8 # Average distance between Moon and Earth in meters

1. Calculate the Gravitational Force between the Earth and the Sun:

The formula for gravitational force is: [F = \frac{G \cdot M_1 \cdot M_2}{d^2}]

For Earth and Sun: [F_{\text{earth-sun}} = \frac{6.67430 \times 10^{-11} \cdot 5.972 \times 10^{24} \cdot 1.989 \times 10^{30}}{(1.496 \times 10^{11})2}]

[F_{\text{earth-sun}} \approx 3.54 \times 10^{22} , \text{N}]

2. Calculate the Gravitational Force between the Moon and the Earth:

For Moon and Earth: [F_{\text{moon-earth}} = \frac{6.67430 \times 10^{-11} \cdot 5.972 \times 10^{24} \cdot 7.34767309 \times 10^{22}}{(3.844 \times 10^{8})2}]

[F_{\text{moon-earth}} \approx 1.98 \times 10^{20} , \text{N}]

3. Compare the Calculated Forces:

- Gravitational force between Earth and Sun: (3.54 \times 10^{22}) N
- Gravitational force between Moon and Earth: (1.98 \times 10^{20}) N

The gravitational force between the Earth and the Sun is significantly stronger than the gravitational force between the Moon and the Earth.

Based on the comparison, the Earth is more strongly attracted to the Sun than the Moon is to the Earth. This is due to the much larger mass of the Sun compared to the Earth, despite the greater distance between the Earth and the Sun.

```
class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    def get_roll_number(self):
        return self.__roll_number

    # Setter methods
    def set_name(self, name):
        self.__name = name

    def set_age(self, age):
        self.__age = age

    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
```

```
    # Method to display student information
    def display_info(self):
        print(f"Name: {self.__name}")
        print(f"Age: {self.__age}")
        print(f"Roll Number: {self.__roll_number}")

    # Method to update student details
    def update_details(self, name=None, age=None, roll_number=None):
        if name:
            self.__name = name
        if age:
            self.__age = age
        if roll_number:
            self.__roll_number = roll_number

# Creating instances of the Student class and testing the functionality
student1 = Student("Alice", 20, "S12345")
student2 = Student("Bob", 22, "S67890")

# Displaying student information
student1.display_info()
student2.display_info()

# Updating student details
```

```
student1.update_details(name="Alicia", age=21)

student2.update_details(roll_number="S67891")


# Displaying updated student information

print("\nUpdated Student Information:")

student1.display_info()

student2.display_info()
```

39. Basic Flask Route to Display “Hello, World!”

```
from flask import Flask


app = Flask(__name__)


@app.route('/')

def hello_world():

    return 'Hello, World!'


if __name__ == '__main__':

    app.run(debug=True)
```

40. Handling Form Submissions Using POST Requests

```
<!DOCTYPE html>

<html>

<body>

    <form action="/submit" method="post">

        Name: <input type="text" name="name"><br>

        Age: <input type="text" name="age"><br>

        <input type="submit" value="Submit">
```

```
    </form>
</body>
</html>
```

41. **Flask Route with URL Parameter**

```
from flask import Flask

app = Flask(__name__)

@app.route('/user/<username>')
def show_user_profile(username):
    return f'User: {username}'

if __name__ == '__main__':
    app.run(debug=True)
```

42. **Implementing User Authentication in Flask**

To implement user authentication in a Flask application, you can use the Flask-Login extension. Here's a basic example:

1. **Install Flask-Login**:
2. pip install flask-login
3. **Set up the Flask application**:

```
from flask import Flask, render_template, redirect, url_for, request
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user

app = Flask(__name__)
```

```
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
db = SQLAlchemy(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'

class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(150), nullable=False)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if user and user.password == password:
            login_user(user)
            return redirect(url_for('dashboard'))
    return render_template('login.html')

@app.route('/dashboard')
@login_required
def dashboard():
```

```
    return f'Hello, {current_user.username}!'

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
```

42. Connecting Flask to SQLite Database Using SQLAlchemy

***Install Flask-SQLAlchemy:**

```
pip install flask-sqlalchemy
```

***Set up the Flask application:**

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)

if __name__ == '__main__':
    app.run(debug=True)
```

* **Create the database**:

```
from your_application import db

db.create_all()
```

44. Creating a RESTful API Endpoint in Flask that Returns JSON Data

```
from flask import Flask, jsonify


app = Flask(__name__)


@app.route('/api/data', methods=['GET'])

def get_data():

    data = {

        'name': 'John Doe',

        'age': 30,

        'city': 'New York'

    }

    return jsonify(data)


if __name__ == '__main__':

    app.run(debug=True)
```

45. Using Flask-WTF to Create and Validate Forms

1. **Install Flask-WTF: pip install flask-wtf**
2. **Set up the Flask application :**

```
from flask import Flask, render_template, request
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'

class MyForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    submit = SubmitField('Submit')
```

```
@app.route('/form', methods=['GET', 'POST'])
def form():
    form = MyForm()
    if form.validate_on_submit():
        return f'Hello, {form.name.data}!'
    return render_template('form.html', form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

3. **Create the HTML template:**

```
<!DOCTYPE html>
<html>
<body>
  <form method="POST">
    {{ form.hidden_tag() }}
    {{ form.name.label }} {{ form.name() }}<br>
    {{ form.submit() }}
  </form>
</body>
</html>
```

46. Implementing File Uploads in Flask :

Set up the Flask application:-

```
from flask import Flask, request, redirect, url_for
from werkzeug.utils import secure_filename
import os

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = '/path/to/upload'
app.config['ALLOWED_EXTENSIONS'] = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}

def allowed_file(filename):
```

```
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']


@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return 'File uploaded successfully'
    return '''
    <!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''


if __name__ == '__main__':
    app.run(debug=True)
```

47. Creating a Flask Blueprint

```
from flask import Blueprint
```

```
bp = Blueprint('bp', __name__)


@bp.route('/hello')
def hello():
    return 'Hello, Blueprint!'
```

48. Deploying a Flask Application with Gunicorn and Nginx

To deploy a Flask application using Gunicorn and Nginx:

1. **Install Gunicorn**:

```
pip install gunicorn
```

2. **Run your application with Gunicorn:**

```
gunicorn -w 4 yourapp:app
```

3. **Set up Nginx**:

```
server {
    listen 80;
    server_name your_domain;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }}
```

49. Creating a Fully Functional Web Application with Flask and MongoDB

1. **Install Flask-PyMongo**:

```
pip install flask-pymongo
```

2.Set up the Flask application:

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_pymongo import PyMongo
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.config['MONGO_URI'] = 'mongodb://localhost:27017/myDatabase'
```

```
app.config['SECRET_KEY'] = 'your_secret_key'
mongo = PyMongo(app)

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        password = generate_password_hash(request.form
```

ML- 50

Difference Between Series and DataFrames

Series:

- **A one-dimensional array-like object that can hold data of any type (integer, float, string, etc.).**
- **Each element in a Series has a unique identifier called an index.**

DataFrame:

- **A two-dimensional table-like structure consisting of multiple Series objects.**
- **It can hold data of different types (integer, float, string, etc.) across multiple columns.**

Creating and Reading a MySQL Table with Pandas

Create the Database and Table in MySQL

```
CREATE DATABASE Travel_planner;

USE Travel_planner;


CREATE TABLE bookings (
    user_id INT,
    flight_id INT,
    activity_id DATE
);


INSERT INTO bookings (user_id, flight_id, activity_id) VALUES
(1, 101, '2024-07-31'),
(2, 102, '2024-08-01'),
```

(3, 103, '2024-08-02');

Read the Table Content Using Pandas

```
import pandas as pd
import mysql.connector

# Connect to the MySQL database
conn = mysql.connector.connect(
    host="localhost",
    user="your_username",
    password="your_password",
    database="Travel_planner"
)

# Read the table into a pandas DataFrame
query = "SELECT * FROM bookings"
df = pd.read_sql(query, conn)

# Display the DataFrame
print(df)

# Close the connection
conn.close()
```

Output:

```
   user_id  flight_id activity_id
0        1        101  2024-07-31
1        2        102  2024-08-01
2        3        103  2024-08-02
```

Difference Between LOC and iloc :

- loc: Label-based indexing. You use labels to select rows and columns.

Python

```
df.loc[0:2, 'column_name']
```

- iloc: Integer-based indexing. You use integer positions to select rows and columns.

Python

```
df.iloc[0:2, 0:1]
```

example:

```
import pandas as pd


data = {'A': [1, 2, 3], 'B': [4, 5, 6]}

df = pd.DataFrame(data)


# Using loc

print(df.loc[0:1, 'A'])


# Using iloc

print(df.iloc[0:1, 0:1])
```

Difference Between Supervised and Unsupervised Learning

Supervised Learning:

- Uses labeled data to train the model.
- The goal is to predict the output from the input data.
- Examples: Regression, Classification.

Unsupervised Learning:

- Uses unlabeled data to find hidden patterns.
- The goal is to explore the structure of the data.
- Examples: Clustering, Association.

Bias-Variance Tradeoff

The **bias-variance tradeoff** is a fundamental concept in machine learning that describes the tradeoff between two sources of error that affect the performance of predictive models:

- **Bias**: Error due to overly simplistic models that fail to capture the underlying patterns in the data (underfitting).
- **Variance**: Error due to overly complex models that capture noise in the training data (overfitting).

Precision and Recall vs. Accuracy

- **Precision**: The ratio of true positive predictions to the total predicted positives. It measures the accuracy of the positive predictions.

Precision=True Positives+False PositivesTrue Positives

- **Recall**: The ratio of true positive predictions to the total actual positives. It measures the ability to find all relevant instances.

Recall=True Positives+False NegativesTrue Positives

- **Accuracy**: The ratio of correct predictions (both true positives and true negatives) to the total number of predictions.

Accuracy=Total PredictionsTrue Positives+True Negatives

Overfitting and How to Prevent It

Overfitting occurs when a model learns the noise in the training data rather than the actual patterns, leading to poor performance on new data.

Prevention Techniques:

- **Cross-validation**: Use techniques like k-fold cross-validation to ensure the model generalizes well.
- **Regularization**: Apply L1 or L2 regularization to penalize large coefficients.
- **Pruning**: In decision trees, prune the tree to remove unnecessary branches.
- **Early Stopping**: Stop training when performance on a validation set starts to degrade.

Cross-Validation

Cross-validation is a technique used to evaluate the performance of a model by dividing the data into multiple subsets (folds). The model is trained on some folds and tested on the remaining fold. This process is repeated multiple times, and the results are averaged to provide a robust estimate of model performance

Classification vs. Regression

- **Classification**: Predicts discrete labels or categories. Example: Classifying emails as spam or not spam.
- **Regression**: Predicts continuous values.

Ensemble Learning

- **Ensemble learning** involves combining multiple models to improve overall performance. Techniques include:
- **Bagging**: Training multiple models on different subsets of the data and averaging their predictions (e.g., Random Forest).
- **Boosting**: Sequentially training models to correct the errors of previous models (e.g., AdaBoost, Gradient Boosting).

Gradient Descent

Gradient Descent is an optimization algorithm used to minimize the loss function in machine learning models. It works by iteratively adjusting the model parameters in the direction of the negative gradient of the loss function.

Batch Gradient Descent vs. Stochastic Gradient Descent

- **Batch Gradient Descent**: Uses the entire dataset to compute the gradient at each step. It is stable but can be slow for large datasets.
- **Stochastic Gradient Descent (SGD)**: Uses a single data point to compute the gradient at each step. It is faster but can be noisy and less stable.

Curse of Dimensionality

The **Curse of Dimensionality** refers to the challenges that arise when working with high-dimensional data. As the number of features increases, the volume of the feature space grows exponentially, making it difficult for models to generalize well.

Difference Between L1 and L2 Regularization

- **L1 Regularization (Lasso)**: Adds the absolute value of the coefficients as a penalty term to the loss function. It can lead to sparse models where some coefficients are exactly zero, making it useful for feature selection.
- **L2 Regularization (Ridge)**: Adds the squared value of the coefficients as a penalty term to the loss function.

Confusion Matrix

A confusion matrix

is a table used to evaluate the performance of a classification model. It summarizes the results of predictions by comparing them to the actual outcomes. The matrix includes:

- **True Positives (TP)**: Correctly predicted positive cases.
- **True Negatives (TN)**: Correctly predicted negative cases.
- **False Positives (FP)**: Incorrectly predicted positive cases.
- **False Negatives (FN)**: Incorrectly predicted negative cases.

AUC-ROC Curve

The **AUC-ROC curve** (Area Under the Receiver Operating Characteristic curve) is a performance measurement for classification models. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at various threshold settings.

K-Nearest Neighbors Algorithm

The **k-nearest neighbors (KNN)** algorithm is a non-parametric, supervised learning method used for classification and regression. It classifies a data point based on the majority class among its k-nearest neighbors in the feature space.

Basic Concept of Support Vector Machine (SVM)

A **Support Vector Machine (SVM)** is a supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that maximizes the margin between different classes in the feature space.

Kernel Trick in SVM

The **kernel trick** allows SVMs to perform classification in higher-dimensional spaces without explicitly computing the coordinates of the data in that space.

Types of Kernels in SVM

- **Linear Kernel**: Used when the data is linearly separable.
- **Polynomial Kernel**: Suitable for non-linear data with polynomial relationships.
- **Radial Basis Function (RBF) Kernel**: Effective for non-linear data with complex relationships.
- **Sigmoid kernels** : used in neural networks and for certain type of non- linear data.

Hyperplane in SVM

The **hyperplane** is a decision boundary that separates different classes in the feature space. It is determined by maximizing the margin, which is the distance between the hyperplane and the nearest data points from each class .

Pros and Cons of Using SVM

Pros:

- Effective in high-dimensional spaces.
- Works well with clear margin of separation.
- Robust to overfitting, especially in high-dimensional space.

Cons:

- Not suitable for large datasets due to high computational cost.
- Less effective when the data has a lot of noise and overlaps.

Hard Margin vs. Soft Margin SVM

- **Hard Margin SVM**: Assumes that the data is linearly separable and finds a hyperplane that perfectly separates the classes. It does not allow any misclassifications.
- **Soft Margin SVM**: Allows some misclassifications to handle non-linearly separable data.

Constructing a Decision Tree

The process of constructing a decision tree involves:

1. **Selecting the Best Attribute**: Use criteria like information gain or Gini impurity to choose the attribute that best splits the data.
2. **Splitting the Data**: Divide the dataset into subsets based on the selected attribute.

Working Principle of a Decision Tree

A decision tree works by recursively splitting the data into subsets based on the value of the best attribute at each node. The goal is to create branches that lead to the most homogeneous subsets (i.e., subsets where instances belong to the same class)

Information Gain in Decision Trees

Information gain measures the reduction in entropy or impurity when a dataset is split based on an attribute. It is used to select the attribute that best separates the data into classes.

Gini Impurity in Decision Trees

Gini impurity is a measure of the likelihood of incorrect classification of a randomly chosen element if it was randomly labeled according to the distribution of labels in the subset. It is used to evaluate the quality of a split in decision trees.

Advantages and Disadvantages of Decision Trees

Advantages:

- **Easy to Understand and Interpret**: Decision trees are intuitive and their visual representation is easy to understand.
- **Handles Both Numerical and Categorical Data**: They can handle various types of data without much preprocessing.
- **Non-Parametric**: They do not assume any underlying distribution of the data.

Disadvantages:

- **Prone to Overfitting**: Decision trees can easily overfit the training data, especially if they are deep.
- **Unstable**: Small changes in the data can result in a completely different tree.
- **Biased with Imbalanced Data**: They can be biased if the classes are imbalanced.

How Random Forests Improve Upon Decision Trees

Random forests improve upon decision trees by:

- **Reducing Overfitting**: By averaging multiple decision trees, random forests reduce the risk of overfitting.
- **Increasing Accuracy**: They generally provide better accuracy by combining the predictions of multiple trees.

How a Random Forest Algorithm Works

A random forest algorithm works by:

1. **Creating Multiple Decision Trees**: It builds multiple decision trees using different subsets of the data.
2. **Bootstrapping**: Each tree is trained on a bootstrapped sample of the data.
3. **Random Feature Selection**: At each split in the tree, a random subset of features is considered.

Bootstrapping in Random Forests

Bootstrapping is a resampling technique used in random forests where multiple subsets of the data are created by sampling with replacement.

Feature Importance in Random Forests

Feature importance in random forests is determined by evaluating how much each feature contributes to reducing the impurity (e.g., Gini impurity or entropy) in the trees.

Key Hyperparameters of a Random Forest

- **Number of Trees (**n_estimators**)**: More trees generally improve performance but increase computational cost.
- **Maximum Depth (**max_depth**)**: Limits the depth of each tree to prevent overfitting.
- **Minimum Samples Split (**min_samples_split**)**: The minimum number of samples required to split an internal node.
- **Minimum Samples Leaf (**min_samples_leaf**)**: The minimum number of samples required to be at a leaf node.

Logistic Regression Model and Its Assumptions

Logistic regression is a statistical model used for binary classification. It assumes:

- **Linearity**: The log-odds of the outcome are a linear combination of the predictor variables.
- **Independence**: Observations are independent of each other.

Handling Binary Classification with Logistic Regression

Logistic regression handles binary classification by modeling the probability that a given input belongs to a particular class.

Sigmoid Function in Logistic Regression

The **sigmoid function** is defined as:

$$\sigma(x)=1+e-x1$$

Cost Function in Logistic Regression

The **cost function** in logistic regression is the log-loss function, which measures the performance of a classification model whose output is a probability value between 0 and 1. It is defined as:

$$J(\theta)= -m1i=1\sum m[yilog(h\theta(xi))+(1-yi)log(1-h\theta(xi))]$$

Extending Logistic Regression to Multiclass Classification

Logistic regression can be extended to multiclass classification using techniques like:

- **One-vs-Rest (OvR)**: Train a separate binary classifier for each class.

Difference Between L1 and L2 Regularization in Logistic Regression

- **L1 Regularization (Lasso)**: Adds the absolute value of the , as a penalty term to the loss function, leading to sparse models.
- **L2 Regularization** : adds the squared value of the coefficients as per a penalty team , leading to smaller but non zero coefficients.

Concept of Boosting in Ensemble Learning

Boosting is an ensemble technique that combines multiple weak learners to create a strong learner. its works by sequentially training model . each one correcting the error of its predecessor.

Handling Missing Values in XGBoost

XGBoost handles missing values by automatically learning the best direction to go when it encounters a missing value during training. This allows it to handle the data set with the missing value without requiring imputation.