**PW Assignment**

**1. What is the difference between static and dynamic variables in Python?**

### Solution.

### Static Variables

Static variables are typically used in the context of classes. These are variables that are shared among all instances of a class. They are defined within a class but outside any of its methods. Static variables are also known as class variables.

### Dynamic Variables

Dynamic variables, also known as instance variables, are specific to each instance of a class. They are typically defined within methods and are unique to each object created from the class.

**2. Explain the purpose of "pop","popitem","clear(" in a dictionary with suitable examples.**

**Solution.**

**1. pop**

The pop method removes the specified key and returns the corresponding value. If the key is not found, it raises a KeyError unless a default value is provided.

**my\_dict = {'a': 1, 'b': 2, 'c': 3}**

**# Removing a key and getting its value**

**value = my\_dict.pop('b')**

**print(value) # Output: 2**

**print(my\_dict) # Output: {'a': 1, 'c': 3}**

**# Removing a non-existing key with a default value**

**value = my\_dict.pop('d', 'Not Found')**

**print(value) # Output: Not Found**

**2. popitem**

The popitem method removes and returns an arbitrary (key, value) pair from the dictionary. If the dictionary is empty, it raises a KeyError. This method is useful for implementing algorithms that need to destructively iterate over a dictionary**.**

**my\_dict = {'a': 1, 'b': 2, 'c': 3}**

**# Removing an arbitrary item**

**key, value = my\_dict.popitem()**

**print(key, value) # Output: (e.g., 'c', 3)**

**print(my\_dict) # Output: {'a': 1, 'b': 2}**

**3. clear**

The clear method removes all items from the dictionary, leaving it empty.

**my\_dict = {'a': 1, 'b': 2, 'c': 3}**

**# Clearing all items from the dictionary**

**my\_dict.clear()**

**print(my\_dict) # Output: {}**

**3.** **What do you mean by FrozenSet? Explain it with suitable examples.**

**Solution.** A frozenset in Python is an immutable version of a set. Unlike sets, which are mutable and can be modified after their creation (e.g., elements can be added or removed), frozensets cannot be changed once they are created. This immutability makes frozensets hashable, which means they can be used as keys in dictionaries or stored in other sets.

Example.

**# Creating a frozenset from a list**

my\_list = [1, 2, 3, 4, 5]

my\_frozenset = frozenset(my\_list)

print(my\_frozenset) # Output: frozenset({1, 2, 3, 4, 5})

**# Creating a frozenset from a set**

my\_set = {1, 2, 3, 4, 5}

my\_frozenset = frozenset(my\_set)

print(my\_frozenset) # Output: frozenset({1, 2, 3, 4, 5})

**# Creating a frozenset from a string**

my\_string = "hello"

my\_frozenset = frozenset(my\_string)

print(my\_frozenset) # Output: frozenset({'e', 'h', 'l', 'o'})

**4. Differentiate between mutable and immutable data types in Python and give examples of mutable andimmutable data types.**

**Solution.** Mutable data types are those whose values can be changed after they are created. You can modify, add, or remove elements from these objects without creating a new object.

**Examples of Mutable Data Types:**

List:

**my\_list = [1, 2, 3]**

**my\_list.append(4) # Modifying the list**

**print(my\_list) # Output: [1, 2, 3, 4]**

Dictionary:

**my\_dict = {'a': 1, 'b': 2}**

**my\_dict['c'] = 3 # Adding a new key-value pair**

**print(my\_dict) # Output: {'a': 1, 'b': 2, 'c': 3}**

Set:

**my\_set = {1, 2, 3}**

**my\_set.add(4) # Adding an element**

**print(my\_set) # Output: {1, 2, 3, 4}**

### Immutable Data Types

Immutable data types are those whose values cannot be changed after they are created. If you try to change the value, a new object is created instead.

**Examples of Immutable Data Types:**

String:

**my\_string = "hello"**

**new\_string = my\_string.upper() # Creating a new string**

**print(new\_string) # Output: "HELLO"**

**print(my\_string) # Output: "hello"**

**Tuple:**

my\_tuple = (1, 2, 3)

new\_tuple = my\_tuple + (4,) # Creating a new tuple

print(new\_tuple) # Output: (1, 2, 3, 4)

print(my\_tuple) # Output: (1, 2, 3)

**5. What is \_\_init\_\_?Explain with an example.**

**Solution.** The \_\_init\_\_ method in Python is a special method called a constructor. It is automatically called when a new instance of a class is created. The primary purpose of \_\_init\_\_ is to initialize the attributes of the class with specific values.

**Example**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name # Initialize the 'name' attribute

self.age = age # Initialize the 'age' attribute

def display\_info(self):

print(f"Name: {self.name}, Age: {self.age}")

# Creating an instance of the Person class

person1 = Person("Alice", 30)

# Accessing attributes

print(person1.name) # Output: Alice

print(person1.age) # Output: 30

# Calling a method

person1.display\_info() # Output: Name: Alice, Age: 30

**6.** **What is docstring in Python?Explain with an example.**

**Solution.** A docstring in Python is a special string literal that is used to document a module, class, method, or function. It provides a convenient way of associating documentation with Python code. Docstrings are written using triple quotes (''' ''' or """ """) and can span multiple lines. They are the preferred way to document code because they are accessible via the \_\_doc\_\_ attribute and various documentation generation tools.

Example

def add\_numbers(a, b):

"""

Add two numbers and return the result.

Parameters:

a (int or float): The first number.

b (int or float): The second number.

Returns:

int or float: The sum of the two numbers.

"""

return a + b

class Person:

"""

A class to represent a person.

Attributes:

name (str): The name of the person.

age (int): The age of the person.

Methods:

display\_info(): Prints the person's information.

"""

def \_\_init\_\_(self, name, age):

"""

Initialize the person's name and age.

Parameters:

name (str): The name of the person.

age (int): The age of the person.

"""

self.name = name

self.age = age

def display\_info(self):

"""

Print the person's name and age.

"""

print(f"Name: {self.name}, Age: {self.age}")

# Accessing the docstrings

print(add\_numbers.\_\_doc\_\_)

print(Person.\_\_doc\_\_)

print(Person.\_\_init\_\_.\_\_doc\_\_)

print(Person.display\_info.\_\_doc\_\_)

**7. What are unit tests in Python.**

**Solution.** Unit tests in Python are a type of software testing where individual units or components of a program are tested in isolation. The primary goal of unit tests is to validate that each unit of the software performs as expected. In Python, unit tests are typically written using the unittest module, which is part of the standard library.

import unittest

def add(a, b):

"""Function to add two numbers."""

return a + b

def subtract(a, b):

"""Function to subtract one number from another."""

return a - b

class TestMathOperations(unittest.TestCase):

def test\_add(self):

"""Test the add function."""

self.assertEqual(add(2, 3), 5)

self.assertEqual(add(-1, 1), 0)

self.assertEqual(add(-1, -1), -2)

def test\_subtract(self):

"""Test the subtract function."""

self.assertEqual(subtract(3, 2), 1)

self.assertEqual(subtract(2, 3), -1)

self.assertEqual(subtract(-1, -1), 0)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**8. What is break, continue and pass in Python.**

**Solution.**

In Python, break, continue, and pass are control flow statements that are used to control the execution of loops and other code blocks. Each of these statements serves a different purpose:

**The break statement** is used to terminate the loop prematurely. When break is executed, the loop stops immediately, and the program control moves to the statement following the loop.

**The continue statement** is used to skip the rest of the code inside the current iteration of the loop and move to the next iteration. It does not terminate the loop but simply skips the remaining code in the current iteration.

**The pass statement** is a null operation; it is used as a placeholder in code where a statement is syntactically required but no action is needed. It is often used in empty functions, classes, or loops during the development process.

**9. What are global, protected and private attributes in Python.**

**Solution**

**Global Attributes**

* **Scope**: Global attributes are defined outside of any class or function and are accessible throughout the entire module.
* **Access**: They can be accessed and modified from any function or class within the module.
* **Use**: Typically used for constants or configuration settings that need to be shared across different parts of the program.

**Protected Attributes**

* **Scope**: Protected attributes are intended to be accessible within the class and its subclasses.
* **Access**: In Python, protected attributes are indicated by a single underscore prefix (\_). This is a convention rather than enforced by the language, meaning they can still be accessed from outside the class, but it is generally discouraged.
* **Use**: Used when you want to allow subclasses to access or modify attributes, but indicate that they should not be accessed directly from outside the class hierarchy.

**Private Attributes**

* **Scope**: Private attributes are intended to be accessible only within the class where they are defined.
* **Access**: In Python, private attributes are indicated by a double underscore prefix (\_\_). This triggers name mangling, where the interpreter changes the name of the attribute in a way that makes it harder (but not impossible) to access from outside the class.
* **Use**: Used to prevent accidental access and modification of attributes from outside the class, providing a stronger form of encapsulation.

**10. What are modules and packages in Python?**

**Solution**

### Modules

A module is a single file (with a .py extension) that contains Python code. This code can include functions, classes, variables, and runnable code. Modules allow you to logically organize your Python code, making it more readable and easier to manage.

### Packages

A package is a collection of related modules organized in a directory hierarchy. Packages allow you to structure your project into submodules and subpackages, making large projects more manageable.

**11. What is an Interpreted language & dynamically typed language?Write 5 differences between them?**

**Solution**

### Interpreted Language

An interpreted language is a type of programming language in which most of its implementations execute instructions directly, without the need for prior compilation into machine-language instructions. Instead, an interpreter reads and executes the source code directly.

### Dynamically Typed Language

A dynamically typed language is a programming language in which variable types are determined at runtime rather than at compile-time. This means that you can change the type of a variable by reassigning it to a different type of value during execution.

**12.** **What are decorators in Python? Explain it with an example.Write down its use cases?**

**Solution**

Decorators are a powerful and expressive feature in Python that allow you to modify or enhance the behavior of functions or methods without changing their actual code. Decorators are essentially functions that wrap another function to extend its behavior.

**Example**

**def my\_decorator(func):**

**def wrapper():**

**print("Something is happening before the function is called.")**

**func()**

**print("Something is happening after the function is called.")**

**return wrapper**

**@my\_decorator**

**def say\_hello():**

**print("Hello!")**

**say\_hello()**

**13.** **What is lambda in Python? Why is it used?**

**Solution** A lambda function in Python, also known as an anonymous function, is a small and simple function defined without a name using the lambda keyword. These functions can have any number of arguments but only one expression. The expression is evaluated and returned.

Example

add = lambda x, y: x + y

print(add(5, 3)) # Output: 8

**15. What are iterators , iterable & generators in Python?**

**Solution**

**Iterables**

An iterable is any Python object capable of returning its members one at a time. It can be iterated over using a loop (like for), or converted to an iterator using the iter() function. Common examples include lists, tuples, strings, and dictionaries.

**Iterators**

Iterator ek object hota hai jo data stream ko represent karta hai. Yeh iterator protocol ko implement karta hai, jo methods \_\_iter\_\_() aur \_\_next\_\_() se banta hai. Iterators ko collection ke saare items ek ek karke traverse karne ke liye use kiya jata hai.

**Generators**

Generators ek special type ke iterator hote hain jo function aur yield statement ka use karke banaye jaate hain. Yeh allow karte hain aapko ek aisa function declare karna jo iterator ki tarah behave kare. Generators less boilerplate aur memory overhead ke sath iterators banate hain.

**Question 1. 2. Which of the following identifier names are invalid and why?**

**a) Serial\_no.**

**b) 1st\_Room**

**c) Hundred$**

**d) Total\_Marks**

**e) total-Marks**

**f) Total Marks**

**g) True**

**h) \_Percentag**

**Solution**

 **Serial\_no.** - Invalid (ends with a period)

 **1st\_Room** - Invalid (starts with a digit)

 **Hundred$** - Invalid (contains a dollar sign)

 **Total\_Marks** - Valid

 **total-Marks** - Invalid (contains a hyphen)

 **Total Marks** - Invalid (contains a space)

 **True** - Invalid (reserved keyword)

 **\_Percentag** – Valid

**Question 1.3**

**name = ["Mohan", "dash", "karam", "chandra","gandhi","Bapu"]**

**do the following operations in this list;**

**a) add an element "freedom\_fighter" in this list at the 0th index.**

name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"]

name.insert(0, "freedom\_fighter")

print(name)

**c) add two more elements in the name ["NetaJi","Bose"] at the end of the list.**

name = ["Mohan", "dash", "karam", "chandra", "gandhi", "Bapu"]

name.extend(["NetaJi", "Bose"])

print(name)

**Question 1.5. tuple1=(10,20,"Apple",3.4,'a',["master","ji"],("sita","geeta",22),[{"roll\_no"N1},**

{"name"N"Navneet"}])

a)print(len(tuTle1)@

b)print(tuTle1[-1][-1]["name"]@

c)fetch the value of roll\_no from this tuple.

d)print(tuTle1[-3][1]@

e)fetch the element "22" from this tuple.

**8**

**Navneet**

**N1**

**geeta**

**22**

**Question 1.6**  Write a program to display the appropriate message as per the color of signal(RED-Stop/Yellow-Stay/ Green-Go) at the road crossing.

**Solution**

**def traffic\_signal(color):**

**if color.lower() == "red":**

**print("Stop! The signal is RED.")**

**elif color.lower() == "yellow":**

**print("Stay! The signal is YELLOW.")**

**elif color.lower() == "green":**

**print("Go! The signal is GREEN.")**

**else:**

**print("Invalid signal color. Please enter either Red, Yellow, or Green.")**

**output**

Stop! The signal is RED.

Stay! The signal is YELLOW.

Go! The signal is GREEN.

Invalid signal color. Please enter either Red, Yellow, or Green.

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

Solution

**# Pre-specified numbers**

num1 = 30

num2 = 50

num3 = 40

**# Using conditional expressions to find the largest number**

largest = num1 if (num1 >= num2 and num1 >= num3) else (num2 if (num2 >= num1 and num2 >= num3) else num3)

**# Print the largest number**

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

**output**

The largest number among 30, 50, and 40 is: 50

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

**Solution**

# Initialize an empty list to store prime numbers

prime\_numbers = []

# Iterate through numbers from 2 to 100

for num in range(2, 101):

is\_prime = True

# Check if num is divisible by any number other than 1 and itself

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

is\_prime = False

break

# If num is prime, append it to the prime\_numbers list

if is\_prime:

prime\_numbers.append(num)

# Print the list of prime numbers

print("Prime numbers between 2 and 100 are:")

print(prime\_numbers)

Machine Learning

**1. What is the difference beween Series & Dataframes.**

**Solution**

 **Series**: A Series is a one-dimensional array-like object containing a sequence of values of any data type (integers, strings, floats, etc.). It is similar to a column in a table or an Excel sheet.

 **DataFrame**: A DataFrame is a two-dimensional table-like data structure with rows and columns. Each column in a DataFrame is a Series. DataFrames can hold data of multiple types..

**2. Difference beween loc and iloc.**

**Solution**

loc: Accesses a group of rows and columns by labels or a boolean array.

Example: df.loc[0, 'user\_id'] retrieves the value in the first row of the 'user\_id' column.

iloc: Accesses a group of rows and columns by integer positions (like numpy arrays).

Example: df.iloc[0, 0] retrieves the value at the first row and first column (0-based index).

**3. Explain he bias-variance tradeoff.**

**Solution**

 **Bias**: Error due to overly simplistic assumptions in the learning algorithm. High bias can cause the model to underfit.

 **Variance**: Error due to too much complexity in the learning algorithm. High variance can cause the model to overfit.

 **Tradeoff**: Balancing bias and variance is crucial. A model with too much bias will not capture the underlying trend, while a model with too much variance will capture noise as if it were the underlying trend.

**4. What are precision and recall? How are hey differen from accuracy ?**

**Solution.**

Precision: The ratio of correctly predicted positive observations to the total predicted positives.

Formula: Precision = TP / (TP + FP) (True Positives / (True Positives + False Positives))

Recall: The ratio of correctly predicted positive observations to all the observations in the actual class.

Formula: Recall = TP / (TP + FN) (True Positives / (True Positives + False Negatives))

Accuracy: The ratio of correctly predicted observations to the total observations.

Formula: Accuracy = (TP + TN) / (TP + TN + FP + FN)

**5. What is overfitting and how can it be prevented?**

**Solution**

 **Overfitting**: A model is overfitting when it performs well on the training data but poorly on the test data because it has learned the noise and details in the training data.

 **Prevention Methods**:

* **Cross-Validation**: Using techniques like k-fold cross-validation.
* **Regularization**: Techniques like L1 and L2 regularization.
* **Simplify the Model**: Reduce the complexity of the model.
* **Pruning**: For decision trees, pruning helps in reducing the size of the tree.
* **Use More Data**: More data can help the model generalize better.

**6.** **Explain the conept of cross-validation?**

**Solution**

**Cross-validation** is a statistical technique used to assess the performance of a machine learning model and ensure its generalizability to an independent dataset. The primary goal of cross-validation is to mitigate overfitting and provide a more reliable estimate of model performance by using multiple subsets of the data for training and validation.

**7. What is the differene between a classifiation and a regression problem?**

**Solution**

 **Classification**: The task is to predict a discrete label or category. Examples include predicting whether an email is spam or not, or classifying an image as a cat or dog.

* **Output**: Categorical (e.g., class labels).
* **Algorithms**: Logistic Regression, Decision Trees, SVM, Random Forest, etc.

 **Regression**: The task is to predict a continuous value. Examples include predicting the price of a house or the temperature for the next day.

* **Output**: Continuous numerical values.
* **Algorithms**: Linear Regression, Ridge Regression, Lasso Regression, etc

**8. Explain the concept of ensemble learning?**

**Solution**

**Ensemble Learning**: Combines multiple individual models (called base learners or weak learners) to create a stronger model. The main goal is to improve the performance and robustness over a single model.

* **Types**:
  + **Bagging**: Training multiple models in parallel on different subsets of the data. Example: Random Forest.
  + **Boosting**: Training models sequentially, each new model correcting the errors of the previous ones. Example: AdaBoost, Gradient Boosting.
  + **Stacking**: Combining the outputs of multiple models using another model (meta-learner).

**9.** W**hat is gradient desent and how does it work?**

**Solution**

**Gradient Descent**: An optimization algorithm used to minimize the cost function in machine learning models by iteratively moving towards the minimum value.

* **Steps**:
  1. **Initialize** the model parameters (weights).
  2. **Compute the gradient** of the cost function with respect to each parameter.
  3. **Update the parameters** in the direction opposite to the gradient to reduce the cost function.
  4. **Repeat** until convergence (when changes are minimal).

**10.** W**hat is the curse of dimensionality in mahine learning?**

**Solution**

**Curse of Dimensionality**: As the number of features (dimensions) increases, the volume of the space increases exponentially, making the data sparse. This sparsity makes it difficult for the model to learn and generalize.

* **Challenges**:
  + Requires more data to achieve the same performance.
  + Increases computational cost.
  + Harder to visualize and interpret.
* **Mitigation**: Dimensionality reduction techniques like PCA (Principal Component Analysis) or feature selection methods.

**11. Explain the differene between L1 and L2 regularization**

**Solution**

 **L1 Regularization (Lasso)**:

* **Description**: Adds the absolute value of the magnitude of coefficients as a penalty term to the loss function.
* **Effect**: Can lead to sparse models where some coefficients are zero (feature selection).
* **Formula**: Loss=Loss+λ∑∣wi∣\text{Loss} = \text{Loss} + \lambda \sum |w\_i|Loss=Loss+λ∑∣wi​∣

 **L2 Regularization (Ridge)**:

* **Description**: Adds the squared magnitude of coefficients as a penalty term to the loss function.
* **Effect**: Tends to shrink coefficients but does not set them to zero.
* **Formula**: Loss=Loss+λ∑wi2\text{Loss} = \text{Loss} + \lambda \sum w\_i^2Loss=Loss+λ∑wi2​

**12.** W**hat is a confusion matrix and how is it used?**

**Solution**

**confusion Matrix**: A table used to evaluate the performance of a classification model. It compares the actual target values with those predicted by the model.

* **Components**:
  + **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.
* **Usage**: Helps in calculating various performance metrics like accuracy, precision, recall, and F1-score.

**13. Define AUC-ROC curve**

**Solution**

**AUC-ROC Curve**: A graphical representation of a classifier’s performance.

* **ROC (Receiver Operating Characteristic) Curve**: Plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.
* **AUC (Area Under the ROC Curve)**: Measures the entire two-dimensional area underneath the ROC curve. It provides a single scalar value to summarize the performance.
* **Interpretation**: Higher AUC indicates better model performance. AUC value ranges from 0 to 1, with 1 indicating perfect classification and 0.5 indicating random guessing.

**14. Explain the basic concept of a Support Vector Mahine (SVM)**

**Solution**

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks.

Basic Concept: SVM finds the hyperplane that best separates the data points of different classes with the maximum margin.

* Hyperplane: A decision boundary that separates the data points of different classes.
* Support Vectors: Data points that are closest to the hyperplane and influence its position and orientation.

**15 .How does the kernel trick work in SVM**

**Solution**

The **Kernel Trick** allows SVM to create a non-linear decision boundary by transforming the input data into a higher-dimensional space without explicitly computing the transformation.

* **Kernel Function**: Computes the inner product of two data points in the transformed feature space.
* **Common Kernels**:
  + **Linear Kernel**: Suitable for linearly separable data.
  + **Polynomial Kernel**: Suitable for polynomially separable data.
  + **RBF (Radial Basis Function) Kernel**: Suitable for non-linearly separable data.

**16. What is the hyperplane in SVM and how is it determined**

**Solution**

**Hyperplane in SVM and How It Is Determined**

Hyperplane: A flat affine subspace (e.g., a line in 2D, a plane in 3D) that separates different classes.

Determination: SVM maximizes the margin, the distance between the hyperplane and the nearest data points of each class (support vectors).

**17. Desribe the working priniple of a deision tree**

**Solution** A Decision Tree splits the data at each node based on the value of an attribute, creating branches for each possible value. This process continues until it reaches a leaf node, which represents a class label.

**18. How do random forests improve upon deision trees**

**Solution**

Random Forests are an ensemble learning method that improves the performance and robustness of decision trees by combining multiple trees into a "forest." This aggregation leads to better generalization and predictive performance. Here are the key ways in which random forests improve upon decision trees:

1. Reduction in Overfitting

* Decision Trees: Prone to overfitting, especially when the tree is allowed to grow deep, capturing noise and small fluctuations in the training data.
* Random Forests: Reduce overfitting by averaging the predictions of multiple trees. The randomization in the training process helps to produce trees that are less correlated, thus reducing the variance.

2. Ensemble Approach

* Bootstrap Aggregating (Bagging): Random forests use bagging, where each tree is trained on a random sample of the training data with replacement. This means that each tree is trained on a slightly different dataset, increasing the diversity among the trees.
* Averaging Predictions: The final prediction is obtained by averaging the predictions of individual trees (for regression) or by majority voting (for classification). This ensemble method helps in smoothing out the predictions and reducing variance.

3. Random Feature Selection

* Feature Randomness: When splitting nodes, random forests select a random subset of features instead of considering all features. This further decorrelates the trees, as each tree is likely to make different splits based on different features.
* Improvement in Variance Reduction: By decorrelating trees through feature randomness, random forests can reduce the overall variance of the model, leading to better generalization.

4. Robustness and Stability

* Decision Trees: Sensitive to small changes in the data. A small change can result in a completely different tree structure.
* Random Forests: More stable as they aggregate the results of multiple trees. Small changes in the data are less likely to affect the overall model performance significantly.

5. Handling Missing Values and Outliers

* Decision Trees: Can handle missing values but can be sensitive to outliers.
* Random Forests: More robust to outliers and missing values. The averaging process helps mitigate the impact of anomalies.

6. Feature Importance

* Decision Trees: Can provide feature importance based on the splits, but this may be biased towards features with more levels.
* Random Forests: Provide more reliable estimates of feature importance by averaging the importance measures across many trees.

7. Performance and Accuracy

* Decision Trees: Can achieve high accuracy on training data but may perform poorly on unseen data due to overfitting.
* Random Forests: Tend to have better performance on unseen data due to the ensemble approach, which improves the model's ability to generalize.

**19.Describe the logistic regression model and its assumptions**

**Solution**

Logistic Regression is a statistical method used for binary classification problems, where the goal is to predict the probability that an instance belongs to one of two classes. Despite its name, it is a classification algorithm, not a regression algorithm.

**20. What is the differene between L1 and L2 re0ularization in logistic regression?**

**Solution**

**L1 Regularization (Lasso Regression):**

* **Penalty Term**: Adds the sum of the absolute values of the coefficients to the loss function.
* **Effect on Weights**: Encourages sparsity, meaning it can drive some coefficients to exactly zero, effectively performing feature selection.
* **Formula**: The regularization term is λ∑j=1n∣wj∣\lambda \sum\_{j=1}^n |w\_j|λ∑j=1n​∣wj​∣, where λ\lambdaλ is the regularization parameter and wjw\_jwj​ are the model coefficients.
* **Use Case**: Useful when you suspect that only a few features are important and want to perform feature selection.

**L2 Regularization (Ridge Regression):**

* **Penalty Term**: Adds the sum of the squares of the coefficients to the loss function.
* **Effect on Weights**: Encourages smaller weights but does not drive them to zero. It tends to distribute the error among all the coefficients.
* **Formula**: The regularization term is λ∑j=1nwj2\lambda \sum\_{j=1}^n w\_j^2λ∑j=1n​wj2​, where λ\lambdaλ is the regularization parameter and wjw\_jwj​ are the model coefficients.
* **Use Case**: Useful when you have many features that all contribute to the output and want to keep all features but reduce their impact.