Infosys Internship Learning Summary

Day 1 (10/11/25)

- **Introduction:** Overview of internship workflow, expectations, and guidelines.
- **Project Allocation:** Assigned the Smart Stock Inventory Optimisation project and responsibilities.

Day 2 (11/11/25)

- **Tech Stack Selection:** Decided on tools like Python, Fast API/Flask, Django, ML. model, and databases.
- **API Basics:** Understood how APIs allow communication between client and server.
- **Postman:** Learned to test and debug APIs using requests and responses.
- **WebSocket:** Understood real-time data exchange through persistent server connection.
- **NumPy:** NumPy is a Python library used for fast numerical operations on large arrays and matrices.
- **Pandas:** Pandas is a Python library that helps you handle, and analyse data using DataFrames.

Day 3 (12/11/25)

- **Fast API:** High-performance Python framework for building fast, modern API.
- **Flask API:** Lightweight Python framework used for simple and flexible API development
- **Pydantic**: Library for validating and converting data using Python models.

Day 4 (13/11/25)

GitHub: Platform for code storage, version control, and team collaboration.

- **git init** ○ Creates a new Git repository in your project folder.

- **git clone <url>** ○ Copies an existing remote repository to your system.
- **git status** ○ Shows changes in your working directory.
- **git add.**
  - Stages all changed files for the next commit.

- **git add <filename>** ○ Stages a specific file.
- **git commit -m "message"** ○ Saves your staged changes with a message.
- **git push origin main** ○ Uploads your commits to the main branch on GitHub.
- **git pull** ○ Downloads and merges the latest changes from the remote repository.
- **git branch** ○ Shows all branches in your repository.
- **git branch <name>** ○ Creates a new branch.
- **git checkout <branch>** ○ Switches to another branch.
- **git merge <branch>** ○ Combines another branch into the current branch.
- **git log** ○ Shows commit history.
- **git remote -v** ○ Shows linked remote repositories.
- **Virtual Environment:** Isolated Python environment to manage project-specific packages.
- **requirements.txt:** File listing all dependencies needed to run the project.

Day 5 (14/11/25)

- **Payload:** The actual data sent in an API request or response.
- **Endpoint:** A specific API URL that performs a defined action.
- **Al Models**: Algorithms that learn patterns from data to make predictions or decisions.
- **Types of Machine Learning:** Supervised, unsupervised and reinforcement learning approaches.
- **Llama Model:** A large language model useful for NLP and text-generation tasks

Day 6 (16/11/25)

- **Database Concepts**: Learned tables, keys, relationships, and data storage principles.
- **Normalisation:** Process of organising data to reduce redundancy and improve consistency.
- **1NF-First Normal Form**: Data is organised so each column has atomic values and no repeating groups.

- **2NF-Second Normal Form:** The table is in 1NF and every non-key column depends on the primary key.
- **3NF Third Normal Form**: Table is in 2NF and has no transitive dependencies (non-key fields don't depend on other non-key fields).
- **BCNF - Boyce Codd Normal Form**: Stronger version of 3NF where every determinant must be a candidate key.
- **4NF-Fourth Normal Form:** Table must not contain a multivalued dependency.
- **5NF-Fifth Normal Form**: Data is broken into tables to avoid join dependency issues.
- **PostgreSQL vs MySQL:** PostgreSQL is feature-rich and strict, and MySQL is simpler and faster for basic use.
- **Vector Databases:** Databases that store embeddings for AI search, similarity, and recommendations.

Day 7 (17/11/25)

- **GitHub Repository:** A centralised place to store and manage version-controlled project code.
- **Project Review:** Evaluated current project progress and clarified next development steps.

Day 8 (18/11/25)

- Documents were reviewed and corrected.

Day 9(19/11/25)

- PPT presentation.
- Code was reviewed.

Day 10(20/11/2025)

- PPT presentation.
- Code was reviewed.

Day 11(21/11/25)

- Discussed about project.
- Reviewed the milestone 1.

Day 12(24/11/25)

Machine Learning Workflow:

- Collect Data: Gather all required raw data from various sources.
- Prepare and Engineer Features: Clean the data, handle noise, and create meaningful features for the model.
- Train a Model: Use machine learning algorithms to learn patterns from the prepared data.
- Evaluate on Holdout Data Using Metrics: Test the model on unseen data and measure performance using evaluation metrics.
- Deploy to Production and Monitor: Put the trained model into real use and continuously track its performance.

Applications Discussed:

- Demand Forecasting: Predict future product demand using machine learning methods.
- Dynamic Pricing: Adjust prices automatically based on demand, supply, and competition.
- Customer Analytics: Analyze customer behavior, segmentation, and patterns for business insights.

Day 13(25/11/25)

NLP points:

- **Sentiment Analysis**
  Understanding positive, negative, or neutral emotions from text.
  Used for reviews, feedback, and opinion mining.
- **Tokenization**
  Breaking text into smaller units like words or sub words.
  First step in most NLP pipelines.
- **Stemming**
  Reducing words to their root form.
  Example: *playing* → *play*
- **Lemmatization**
  Converting words to their dictionary/base form.
  Examples: *better* → *good*, *running* → *run*
- **Stop-Word Removal**
  Removing common words (the, is, in) that add little meaning.
  Helps reduce noise.
- **Text Preprocessing**
  Cleaning text before model building.
  Includes lowercasing, removing symbols, punctuation, numbers, etc.
- **Part-of-Speech (POS) Tagging**
  Identifying grammar roles such as noun, verb, adjective, etc.
  Helps understand sentence structure.

- **Named Entity Recognition**
  Identifying key entities in text.
  Types include PERSON, ORGANIZATION, GPE, DATE, PRODUCT
  Examples: dmart, India
- **Building small NLP models**
  General steps: preprocessing → feature engineering → training → evaluation

LSTM (Long Short-Term Memory):

A type of RNN used for sequential and time-dependent data.
Remembers long-term patterns and avoids forgetting.
LSTM Gates: Forget Gate, Input Gate, Output Gate.

Day 14(26/11/25)

1. **Margin:** Margin is the distance between the decision boundary and the nearest data points. A larger margin generally means better separation between classes.
2. **Hard margin vs Soft margin:**
   Hard margin allows *no misclassification* & fits data with perfect separating boundary. Data clear clean. There should not be noise.
   Soft margin allows some errors to avoid overfitting and handle noisy or overlapping data.
3. **TF-IDF:**
   a. TF (Term frequency) measures how often a word appears in a document. Higher TF means the word is more common within that specific document.
   b. IDF (Inverse Document Frequency) measures how rare a word is across all documents in a collection. Words that appear in fewer documents get higher IDF because they are more unique.
   c. Ex: "Git" word appears 30 times in the document and the total word in the document is 1000. "The" word appears 900 times in the document and the total word in the document is 1000.
   d. 30/1000 => High IDF and 900/1000 => low IDF
4. **Tools:**
   a. Calculator: used for computing TF, IDF values, ratios, margins, and numerical calculations.
   b. Unit Converter: used for converting values during preprocessing or normalization steps.
5. **SVM:** Support vector machine
   a. SVM is a machine learning algorithm that finds the best boundary that separates classes.
   b. Works best for small or medium-sized datasets.
   c. Not suitable for highly noisy features because it tries to create a clear margin.
   d. Performs high-dimensional separation, can handle large feature spaces.

    e. Can solve non-linear problems using kernel tricks (RBF, polynomial, etc.).
    f. Gives strong accuracy when data is clean and well-separated.
    g. Focuses on maximizing margin, which makes the model more robust.

6. **Embedding:** Embedding converts raw text into numerical vectors so that a machine learning model can understand it.
7. **Reinforcement learning:** Reinforcement learning uses a try-and-error method where an agent learns from mistakes. The agent receives rewards or penalties and gradually learns the best actions.

Day 15(27/11/25)

- **Git vs GitHub**: Git is a local version control tool used to track code changes. GitHub is an online platform to store Git repositories and collaborate with others.
- **Agentic AI workflow:** Agentic AI follows a loop of planning, acting, observing, and improving. It continues autonomously until the goal is completed.
- **Types of agents:**

  • Reactive Agent

  A reactive agent responds instantly to the environment without memory. It makes decisions based only on current inputs, not past experiences.

  • Goal-Driven Agent

  A goal-driven agent selects actions based on achieving specific goals. It evaluates different actions and chooses the one that moves it closer to its goal.

  • Multi-Agent System

  A multi-agent system contains multiple agents that interact and collaborate. They work together to solve complex tasks that a single agent cannot handle efficiently. Manager agent will be there to look after the remaining agents works.

- **Autogen framework:** Autogen is a framework for building multi-agent AI systems. It lets different AI agents communicate, collaborate, and complete tasks automatically.
- **User proxy agent:** Mediator b/w user and assistant agent. A user proxy agent represents the user inside the agent system. It gives instructions, checks outputs, and ensures results match the user's needs.
- **Assistant agent:** An assistant agent is the main problem-solving agent. It performs tasks, generates outputs, and follows the instructions or plans provided.

Day 16 (28/11/2025)

1.OpenAI

- pip install openai
- resp = client. chat. completions. create
  (

```
model= "gpt-4o-mini" #can replace with an available model like "gpt-4o" or
"gpt-5" etc.
messages = [
        {“role": "system", "content": "you are a helpful assistant”}
        {“role": "user", "content": "write a two-line poem about chair”}
        ]
max_tokens = 120
)
```

- Response contains: model, messages, max_tokens

2. **Code Structure (General Flow)**

```
from openai import OpenAI
client = OpenAI()
assistant_text = resp.choices[0].message["content"]
```

3. **Google Generative AI (Gemini)**

- Installation: pip install google-generativeai
- Import: import google.generativeai as genai
- Model: model = genai.GenerativeModel("gemini-1.5-flash")
- User sends a message: response = model.generate_content("Write a short poem about coffee.")
- Chat Completions: model returns the next message.

4. **Function Calling**

```
functions = {
"name": "get_temperature"
"description": "returns temperature of a city"
"parameters": {
        "type": "object"
        "properties": {"city": {“type": "string"}}
                }}
```

Using function:

- model = genai.GenerativeModel("gemini-1.5-flash", tools=[functions])
- stream = True

5. **Flow**

- Text → Tokens

  Input text is broken into tokens.
- Tokens → Embeddings

  Tokens are converted into dense vectors.
- Embeddings → Transformer Layers

  Transformer processes embeddings.
- Attention Mechanism

Model identifies which words are important.

- Output → Text (decoded)
  Final generated/decoded text is returned.

6. Temperature

- Controls creativity of output.
- Range: **0.0 to 1.0**

7. Rate Limits & Quotas

- Each API has usage limits depending on model and plan.
- Limits apply to:
  - Requests per minute
  - Tokens per minute
  - Daily quotas

Day 17 (01/12 /2025)

1. Logistic Regression

- Logistic Regression is used for classification problems.
- It predicts outcomes like yes/no, spam/not spam, 0/1.
- The model calculates:
  $z = w1x1 + w2x2 + \ldots + b$
- This value is converted into a probability using the sigmoid function:
  $p = 1 / (1 + e^{-z})$
- If $p > 0.5$, the output is classified as Yes (1).
- Features = x, Bias = b, Weights = w.

Implementation:
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

2. Decision Tree

- A Decision Tree splits data into smaller groups based on questions (conditions).
- Example: *"Is it sunny?"* → Yes/No → next split.
- It uses two main concepts:
  - Entropy: Measures impurity, disorder or uncertainty in the data.
  - Information Gain: Tells how much entropy is reduced after splitting the data.
- Simple to visualize and interpret.

Implementation:
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()

3. Random Forest

- Random Forest is a collection of many decision trees.
- Works for classification and regression.
- Each tree gives an output; the final result is:
    - Majority vote (classification)
    - Average (regression)
- Reduces bias and overfitting.

Implementation:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
```

4. KNN (K-Nearest Neighbors)

- Uses distance to find the nearest data points.
- Distance formula used:
  Euclidean distance = $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$
- The algorithm chooses the closest k points and selects the majority class.
- Works for both classification and regression.

Implementation:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
```

Main Steps:

1. Choose k
2. Find nearest neighbors
3. Assign class based on neighbors
4. Repeat for all points

5. K-Means Clustering

- An unsupervised algorithm (no labels).
- Used to form k clusters.
- You choose k, and the algorithm finds centroids.
- Each data point is assigned to the nearest centroid.

Implementation:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
```

6. Linear Regression

- Used for predicting continuous values (e.g., price, temperature).
- Formula: $y = mx + c$
- Creates a straight line through data points.
- Simple, easy to understand, good for baseline models.

Implementation:
```
from sklearn.linear_model import LinearRegression
```

7. XGBoost

- Boosting algorithm that trains models sequentially.
- Each model fixes the mistakes of the previous model.
- Uses gradient boosting:
  - Start with an initial prediction
  - Compute gradients (errors)
  - Improve prediction step-by-step
- Very powerful for both classification and regression.

Day 18 (02/12 /2025)

1. CREATE

Creates a new table.
```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype);
```

2.INSERT

Adds new records into a table.
```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

3.SELECT

Retrieves data.
```
SELECT * FROM table_name;
SELECT column1, column2 FROM table_name;
```

4.UPDATE

Modifies existing data.
```
UPDATE table_name
SET column = value
WHERE condition;
```

5.DELETE

Deletes rows from a table.
```
DELETE FROM table_name
WHERE condition;
```

6.ALTER

Changes table structure.
```
ALTER TABLE table_name
ADD column_name datatype;
```

7.SELECT DISTINCT

Returns unique values.

```
SELECT DISTINCT column_name FROM table_name;
```

8.Aggregate Functions:

COUNT(), MAX(), MIN(), SUM(), AVG()

Example:

```
SELECT MAX(salary) FROM employees;
```

9. WHERE CLAUSE

Used to filter rows.

```
SELECT * FROM table_name
WHERE column_name = value;
```

10. LIKE OPERATOR (Pattern Matching)

- a% → starts with a
- %a → ends with a
- %or% → contains "or"
- _p% → second character is p
- a__% → starts with a and has at least three characters
- a%0 → starts with a and ends with 0

Example:

```
SELECT * FROM employees
WHERE name LIKE 'a__%';
```

11. IN AND NOT IN

```
SELECT * FROM employees
WHERE department IN ('HR', 'Finance');
SELECT * FROM employees
WHERE department NOT IN ('HR', 'Finance');
```

12. BETWEEN

Used for ranges.

```
SELECT * FROM products
WHERE price BETWEEN 100 AND 500;
```

13. ORDER BY

Used for sorting.

```
SELECT * FROM employees
ORDER BY salary ASC;
SELECT * FROM employees
ORDER BY salary DESC;
```

14. UNION AND UNION ALL

- UNION

Combines results and removes duplicates.

SELECT name FROM table1
UNION
SELECT name FROM table2;

- UNION ALL

Combines results and preserves duplicates.
SELECT name FROM table1
UNION ALL
SELECT name FROM table2;
Rules:

- Same number of columns
- Same data type

15. SQL OPERATORS

Comparison Operators
=
!= or <>
<
=
<=
Logical Operators
AND
OR
NOT
Arithmetic Operators
/
%

16. CONDITIONAL SQL

```
SELECT name, salary,
CASE
    WHEN salary > 50000 THEN 'High'
    WHEN salary BETWEEN 30000 AND 50000 THEN 'Medium'
    ELSE 'Low'
END AS salary_level
FROM employees;
```

17. SQL JOINS

- INNER JOIN
  Returns only matching rows.
- LEFT JOIN
  Returns all rows from the left table and matching rows from the right.
- RIGHT JOIN
  Returns all rows from the right table and matching rows from the left.

- CROSS JOIN
  Returns the Cartesian product of both tables.

18. GROUP BY

Used to group rows based on one or more columns.

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department;
```

19. HAVING

Used to filter groups after GROUP BY.

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

Day 19 (03/12 /2025)

YOLO (You Only Look Once)

YOLO is a **real-time object detection algorithm** that can detect multiple objects in an image with **high speed and good accuracy**. It is widely used in computer vision tasks like surveillance, self-driving cars, traffic monitoring, face detection, etc.

1. What YOLO Does

YOLO performs two tasks at the same time:

- **Object Localization** – finds where the object is (bounding box)
- **Object Classification** – identifies what the object is (label)

YOLO predicts:

- Bounding box coordinates
- Object class
- Confidence score (probability)

2. Why It Is Called "You Only Look Once"

Traditional models scan the image many times.
YOLO scans the entire image **only once** and predicts all objects in a **single pass** through the network.
This makes YOLO extremely **fast**.

3. How YOLO Works

- The image is divided into a **grid** (S × S).
- Each grid cell predicts:
  1. Bounding box (x, y, w, h)

2. Confidence score
3. Class probability

- YOLO combines these predictions and selects the best boxes using **Non-Max Suppression (NMS)**.

4. Advantages of YOLO

- Very **fast** (real-time detection)
- Detects **multiple objects** at once
- Works well for video streams
- Single-stage network (no region proposal needed)

5. Where YOLO Is Used

- CCTV surveillance
- Traffic detection (cars, signals)
- Face and mask detection
- Retail store counting
- Medical imaging
- Agriculture (detecting crops, pests)
- Robotics

6. Simple YOLO Pipeline

- Input image
- Divide into grid
- Predict bounding boxes and classes
- Apply confidence threshold
- Apply Non-Max Suppression (NMS)
- Output final objects detected

Day 20 (04/12 /2025)

- PPT presentation.
- Code was reviewed.
- Checked the documentation.
- And checked the chatbot execution.

Day 21 (05/12 /2025)

- Discussed about project.
- Reviewed the milestone 2.

Day 22 (06/12/2025)

- Informed that the project should be shown on next class.
- Backend and frontend need to be integrated.
- Project must be partially ready for the review.

Day 23 (08/12 /2025)

- Checked and reviewed the completed work by all the teams.
- Gave suggestions and shared some tips.

Day 24 (09/12 /2025)

String Methods

1. **split():** Splits a string into a list by a delimiter.
   Ex: "a,b,c".split(",") # ['a', 'b', 'c']
2. **join():** Joins elements of a list into a string with a separator.
   Ex: ",".join(['a', 'b', 'c']) # 'a,b,c'
3. **replace():** Replaces occurrences of a substring with another string.
   Ex: "hello".replace("l", "p") # 'heppo'
4. **upper():** Converts a string to uppercase.
   Ex: "abc".upper() # 'ABC'
5. **lower():** Converts a string to lowercase.
   Ex: "ABC".lower() # 'abc'
6. **startswith():** Checks if string starts with a given substring.
   Ex: "hello".startswith("he") # True
7. **endswith():** Checks if string ends with a given substring.
   Ex: "hello".endswith("lo") # True
8. **isdigit():** Checks if all characters in string are digits.
   Ex: "123".isdigit() # True
9. **isalpha():** Checks if all characters in string are alphabetic.
   Ex: "abc".isalpha() # True
10. **find():** Returns the index of first occurrence of substring or -1 if not found.
    Ex: "hello".find("l") # 2

List Methods

1. **append():** Adds an element to the end of the list.
   Ex: lst = [1, 2]; lst.append(3); print(lst) # [1, 2, 3]
2. **extend():** Adds all elements of an iterable to the end of the list.
   Ex: lst = [1]; lst.extend([2, 3]); print(lst) # [1, 2, 3]

3. **insert():** Inserts an element at a specified index.
   Ex: lst = [1, 3]; lst.insert(1, 2); print(lst) # [1, 2, 3]
4. **remove():** Removes first occurrence of a value.
   Ex: lst = [1, 2, 3]; lst.remove(2); print(lst) # [1, 3]
5. **pop():** Removes and returns element at index (default last).
   Ex: lst = [1, 2, 3]; print(lst.pop()) # 3
   Ex: print(lst) # [1, 2]
6. **index():** Returns index of first occurrence of a value.
   Ex: [1, 2, 3].index(2) # 1
7. **count():** Counts occurrences of a value.
   Ex: [1, 2, 2, 3].count(2) # 2
8. **sort():** Sorts the list in ascending order.
   Ex: lst = [3, 1, 2]; lst.sort(); print(lst) # [1, 2, 3]
9. **reverse():** Reverses the list in place.
   Ex: lst = [1, 2, 3]; lst.reverse(); print(lst) # [3, 2, 1]

Dictionary Methods

1. **get():** Returns value for key or default if key not found.
   Ex: d = {'a':1}; d.get('a', 0) # 1
   d.get('b', 0) # 0
2. **values():** Returns view of dictionary values.
   Ex: d = {'a': 1, 'b': 2}; list(d.values()) # [1, 2]
3. **keys():** Returns view of dictionary keys.
   Ex: d = {'a': 1, 'b': 2}; list(d.keys()) # ['a', 'b']
4. **items():** Returns view of (key, value) pairs.
   Ex: d = {'a': 1}; list(d.items()) # [('a', 1)]
5. **update():** Updates dictionary with key-value pairs from another dictionary.
   Ex: d = {'a': 1}; d.update({'b': 2}); print(d) # {'a': 1, 'b': 2}
6. **pop():** Removes key and returns its value.
   Ex: d = {'a': 1}; d.pop('a') # 1
7. **popitem():** Removes and returns a (key, value) pair.
   Ex: d = {'a': 1, 'b': 2}; d.popitem() # ('b', 2)

Set Methods

1. **add():** Adds an element to the set.
   Ex: s = {1, 2}; s.add(3); print(s) # {1, 2, 3}
2. **remove():** Removes element; error if not present.
   Ex: s = {1, 2}; s.remove(2); print(s) # {1}
3. **discard():** Removes element if present; no error if absent.
   Ex: s = {1}; s.discard(2); print(s) # {1}

4. **pop():** Removes and returns an arbitrary element.
   Ex: s = {1, 2}; print(s.pop()) # 1 or 2
5. **clear():** Removes all elements.
   Ex: s = {1, 2}; s.clear(); print(s) # set()
6. **union():** Returns a set with all elements from both sets.
   Ex: {1, 2}.union({2, 3}) # {1, 2, 3}
7. **intersection():** Returns a set with common elements.
   Ex: {1, 2}.intersection({2, 3}) # {2}
8. **difference():** Elements in first but not second.

File I/O Methods

- **open():** Opens a file.
  Ex: f = open('file.txt', 'w')
- **read():** Reads entire content.
  Ex: f = open('file.txt', 'r'); data = f.read(); f.close()
- **readline():** Reads one line.
  Ex: f = open('file.txt'); line = f.readline(); f.close()
- **readlines():** Reads all lines as a list.
  Ex: f = open('file.txt'); lines = f.readlines(); f.close()
- **write():** Writes a string.
  Ex: f = open('file.txt', 'w'); f.write("hello"); f.close()
- **writelines():** Writes list of strings.
  Ex: f = open('file.txt', 'w'); f.writelines(["line1\n", "line2\n"]); f.close()
- **close():** Closes file.
  Ex: f = open('file.txt'); f.close()

General Purpose Functions

- **len():** Returns length.
  Ex: Len([1, 2, 3]) # 3
- **range():** Generates sequence.
  Ex :list(range(3)) # [0, 1, 2]
- **print():** Prints.
  Ex: print("hello") # hello
- **type():** Returns type.
  type(3) # <class 'int'>
- **id():** Returns memory id.
  Ex :id("hello") # e.g. 140366921034000
- **sorted():** Returns sorted list.
  Ex :sorted([3, 1, 2]) # [1, 2, 3]
- **enumerate():** Returns index–value pairs.

Ex: list(enumerate(['a', 'b'])) # [(0, 'a'), (1, 'b')]

- **zip():** Combines iterables.
  Ex: list(zip([1,2], ['a','b'])) # [(1, 'a'), (2, 'b')]

Day 25 (10/12 /2025)

Class and Object Related Functions

getattr(obj, name[, default]):

Returns the value of the named attribute of an object. If not found and default is provided, returns default. Otherwise, raises AttributeError.

```
class Person:
    def __init__(self, name):
        self.name = name
p = Person("Alice")
print(getattr(p, 'name'))  # Output: Alice
print(getattr(p, 'age', 25))  # Output: 25 (default)
```

setattr(obj, name, value):

Sets the named attribute of an object to the given value.

```
setattr(p, 'age', 30)
print(p.age)  # Output: 30
```

hasattr(obj, name):

Returns True if the object has the named attribute, otherwise False.

```
print(hasattr(p, 'age'))  # Output: True
print(hasattr(p, 'height'))  # Output: False
```

delattr(obj, name):

Deletes the named attribute from the object.

```
delattr(p, 'age')
print(hasattr(p, 'age'))  # Output: False
```

isinstance(obj, classinfo):

Returns True if the object is an instance of the specified class or any of its subclasses.

```
print(isinstance(p, Person))  # Output: True
```

issubclass(class, classinfo):

Returns True if the first class is a subclass of the second class.

```
class Student(Person): pass
 print(issubclass(Student, Person))  # Output: True
```

Miscellaneous Functions

- **globals():** Returns a dictionary representing the current global symbol table.
- **locals():** Returns a dictionary representing the current local symbol table.
- **callable(obj):** Returns True if the object appears callable.
- **exec(code):** Executes the given code (string or code object).
- **eval(expression):** Evaluates the given expression and returns the result.

Exception Handling

try, except, finally: Used for handling exceptions.

- Try is the block of code that may cause error
- Except is the block of code that handles the error
- Final that runs always with or without error

```
try:
   x = 1 / 0
except ZeroDivisionError:
   print("Cannot divide by zero")
finally:
   print("This always executes")
```

Memory and Object Management

- **del obj:** Removes the reference to the object. When all references are gone, the object becomes eligible for garbage collection.
  ```
  del p  # Removes reference to p
  ```
- **gc.collect():** Forces garbage collection to free up memory by cleaning up unreferenced objects.
  ```
  import gc
  gc.collect()  # Forces garbage collection
  ```

Working with Iterables

- **next(iterator):** Returns the next item from the iterator. Raises StopIteration if exhausted.
- **iter(obj):** Returns an iterator object for the given object.

Decorators and Metaprogramming

- **staticmethod():** Defines a static method in a class.
- **classmethod():** Defines a class method in a class.

Day 26 (11/12 /2025)

A. Python Language Fundamentals

1. What are Python’s key features?

- Interpreted language
- High-level language
- Dynamically typed
- Object-oriented
- Cross-platform
- Supports multiple programming paradigms (procedural, OOP,functional)
- Rich standard library and third-party packages

2. What are Python’s data types?

- Numeric: int, float, complex
- Sequence: list, tuple, str
- Set: set
- Mapping: dict
- Boolean: bool
- Special: NoneType

3. What is PEP 8 and why is it important?

PEP 8 is Python’s official style guide. It ensures:

- Consistent coding style
- Better readability
- Clean and maintainable code
- Industry-standard formatting

4. Explain mutable vs immutable objects

- Mutable: Can be modified after creation (list, dict, set)
- Immutable: Cannot be changed after creation (int, float, str, tuple)

5. What are Python’s built-in data structures?

- List
- Tuple
- Set
- Dictionary

6. What are Python’s memory management features?

- Private heap space
- Reference counting
- Automatic garbage collection

7. Explain indentation in Python

- Defines code blocks instead of {}
- Mandatory in Python
- Improves readability
- Incorrect indentation raises IndentationError

8. How is Python interpreted?

- Source code is compiled into bytecode
- Bytecode is executed by Python Virtual Machine (PVM)

9. Explain namespaces

A namespace is a mapping of variable names to objects.
Types: Local, Global, Built-in

10. Difference between List and Tuple

Feature	List	Tuple
Mutability	Mutable	Immutable
Performance	Slower	Faster
Syntax	[ ]	( )
Use case	Dynamic data	Fixed data

11. How are sets used?

- Store unique elements
- Remove duplicates
- Perform set operations

12. What are dictionaries?

- Key-value pair data structure
- Keys must be unique and immutable
- Fast lookup operations

13. How to merge dictionaries?

```
dict1.update(dict2)
merged = {**dict1, **dict2}
```

14. Shallow copy vs Deep copy

- Shallow copy: Copies references (nested objects shared)
- Deep copy: Copies all objects recursively

15. Syntax error vs Runtime error

• Syntax error: Invalid code structure (before execution)
• Runtime error: Occurs during execution

16. Difference between is and ==

• == checks value equality
• is checks object identity

B. Control Flow & Advanced Concepts

17. Exception handling

```
 try:
        # risky code
except Exception:
        # handle error
finally:
        # always executes
```

18. Function vs Method

• Function: Defined outside a class
• Method: Defined inside a class

19. What are *args and kwargs?

• *args: Variable positional arguments
• **kwargs: Variable keyword arguments

20. What are decorators?

• Modify another function's behavior
• Implemented using @decorator_name

21. What are closures?

A closure remembers variables from its enclosing scope.

22. Lambda function

• Anonymous, single-expression function
lambda x: x * 2

23. Generators

• Use yield
• Generate values lazily

24. Inheritance types
- Multiple inheritance
- Multilevel inheritance

25. Polymorphism
- Same function name, different behavior
- Achieved via overriding or overloading

26. Abstraction
- Hides implementation details
- Uses abstract classes (abc module)

27. Monkey patching
- Dynamic modification of classes or modules at runtime

C. Built-in Functions & Tools

28. Functional programming tools
- map(), filter(), reduce()

29. Common built-in functions
- len(), range(), print()
- type(), id()
- sorted(), enumerate(), zip()

30. Input and output
- input(), print()

31. File I/O methods
- open(), read(), readline(), readlines()
- write(), writelines(), close()

32. Difference between read, readline, readlines
- read(): Entire file
- readline(): One line
- readlines(): All lines as list

33. Check if file exists

```
import os
os.path.exists("file.txt")
```

34. Delete a file

```
import os
os.remove("file.txt")
```

D. Data Structure Operations

35. Set methods

- add(), remove(), discard(), pop(), clear()
- union(), intersection(), difference()

36. Working with iterables

- iter(), next()

37. Stack and Queue implementation

- Stack: LIFO
- Queue: FIFO

(Using list or collections.deque)

E. Object & Memory Management

38. Memory management in Python

- Private heap space
- Reference counting
- Garbage collector

39. Memory-related functions

- del, gc.collect(), import gc

40. Object inspection functions

- getattr(), setattr(), hasattr(), delattr()
- isinstance(), issubclass()

F. pip & Environment

41. What is pip?

- Python package manager
- Installs, updates, removes packages

G. Java Concepts (Basic)

42. JDK, JRE, JVM

- JVM: Executes bytecode
- JRE: JVM + libraries
- JDK: JRE + development tools

H. NumPy Concepts

43. NumPy vs Python Lists

- Faster operations
- Less memory usage
- Supports vectorized operations

44. What is broadcasting in NumPy?

Automatic alignment of arrays with different shapes during arithmetic operations.

Day 27 (12/12 /2025)

Software Project Roles and Responsibilities:

Functional Consultant / Business Analyst

Responsible for understanding business requirements from stakeholders. Translates business needs into functional requirements and designs the Functional Specification Document (FSD). Acts as a bridge between business users and the technical team.

Developer / Programmer

Responsible for writing code based on the Functional Specification Document. Converts functional requirements into technical solutions and develops application features.

Technical Document

Prepared by developers or architects. Describes system architecture, data flow, technologies used, APIs, and technical implementation details.

Testing / QA (Quality Assurance)

Ensures the application meets the functional and technical requirements. Prepares and executes test cases and test scripts based on the Functional Specification Document to identify bugs and defects.

Test Script

A detailed step-by-step set of instructions used by QA to validate application functionality. It includes input data, expected results, and actual results.

Administration

Manages system configuration, user access, system monitoring, backups, and maintenance.

Network Administration
Handles network infrastructure including servers, routers, firewalls, connectivity, and performance monitoring.

Database Administration
Manages databases, including data storage, performance tuning, backups, recovery, and security.

Application Administration
Ensures applications are running smoothly. Manages deployments, patches, version upgrades, and application-level access.

Security
Responsible for protecting systems and data. Implements security policies, access control, encryption, vulnerability management, and compliance.

Product / Project Management
Plans, schedules, and tracks project progress. Manages scope, timelines, resources, and communication between teams and stakeholders.

Sales and Marketing
Promotes the product or service, manages client relationships, market analysis, and revenue generation.

Human Resource
Handles recruitment, employee onboarding, training, payroll, performance management, and employee relations.

Documentation
Creates and maintains all project-related documents such as user manuals, technical documents, SOPs, and training materials.
Support
Provides post-deployment support. Handles user issues, bug reports, incident management, and ensures system availability.

Day 28 (13/12 /2025)

- Informed that the project should be shown on next class.
- Backend and frontend need to be integrated properly with correct database.
- Project must be partially ready for the review.

- The project should have modifications and some additional features as mam suggested in previous milestone.
- And about seminars.

Day 29 (15/12 /2025)

Software Engineering

Definition:

Software Engineering is the systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software systems.

Goals of Software Engineering:

• Produce high-quality software
• Meet user requirements
• Ensure reliability and maintainability
• Deliver software on time and within budget

Software Engineering Framework

A software engineering framework provides a structured approach to software development by defining activities, practices, and workflows.

Core Framework Activities:

1. **Communication**
   • Understanding customer requirements
   • Requirement gathering and analysis
2. **Planning**
   • Estimating time, cost, and resources
   • Project scheduling and risk planning
3. **Modeling**
   • System design and architecture
   • Data and interface design
4. **Construction**
   • Coding and implementation
   • Unit and integration testing
5. **Deployment**
   • Software delivery
   • Feedback collection and maintenance

Types of Software Engineering Practices

Software engineering practices are techniques and methods used to carry out framework activities effectively.

1. Requirement Engineering Practices

• Requirement elicitation

- Requirement analysis
- Requirement specification
- Requirement validation

2. Design Practices

- Architectural design
- UML diagrams
- Database design

3. Coding Practices

- Coding standards
- Code reviews
- Version control (Git)

4. Testing Practices

- Unit testing
- Integration testing
- System testing
- Acceptance testing

5. Maintenance Practices

- Bug fixing
- Performance optimization
- Software updates

Software Process Models (Framework Types)

1. Waterfall Model

- Linear and sequential approach
- Each phase completed before next begins

2. Incremental Model

- Software developed in increments
- Each increment adds functionality

3. Spiral Model

- Risk-driven development
- Combines waterfall and prototyping

4. Agile Model

- Iterative and incremental
- Focus on customer collaboration

5. V-Model

- Verification and validation model
- Each development phase has a testing phase

Types of Workflows (Flow Types)

A workflow defines how activities are organized and executed during software development.

1. Linear Workflow
- Activities follow a fixed sequence
- Used in Waterfall model

2. Iterative Workflow
- Repeated cycles of development
- Used in Agile and Incremental models

3. Parallel Workflow
- Multiple activities occur simultaneously
- Reduces development time

4. Evolutionary Workflow
- Software evolves through continuous feedback
- Used in Spiral and Agile models

Software Development Life Cycle (SDLC) Flow

1. Requirement Analysis
2. System Design
3. Implementation
4. Testing
5. Deployment
6. Maintenance

Day 30 (16/12 /2025)

- Discussed about project.
- Worked on project.

Day 31 (17/12 /2025)

A* Search Algorithm

Definition:
A* (A-star) is a graph search algorithm used to find the shortest path from a start node to a goal node using both path cost and heuristic information.

How it works:
A* uses an evaluation function:
$f(n) = g(n) + h(n)$
where:
- $g(n)$: Actual cost from the start node to the current node
- $h(n)$: Heuristic estimate from the current node to the goal

The algorithm always expands the node with the lowest $f(n)$ value.

Example:
Start Node: A
Goal Node: F

Edges with weights:
(A, B, 1), (A, C, 4)
(B, D, 3), (B, E, 5)
(C, F, 2)
(D, F, 1), (D, E, 1)
(E, F, 2)
Heuristic values:
h(A)=4, h(B)=3, h(C)=2, h(D)=1, h(E)=2, h(F)=0
Step 1: Start at A
g(A)=0, f(A)=4
Step 2: Expand A
B → g=1, f=4
C → g=4, f=6
Choose B
Step 3: Expand B
D → g=4, f=5
E → g=6, f=8
Choose D
Step 4: Expand D
F → g=5, f=5
Goal reached
Final Path: A → B → D → F
Total Cost: 5

Sorting

Definition:
Sorting is the process of arranging elements in a specific order.
Types of order:
• Ascending: Smallest to largest
• Descending: Largest to smallest

Bubble Sort

Definition:
Bubble sort repeatedly compares adjacent elements and swaps them if they are in the wrong order.
How it works:
The largest element bubbles to the end after each pass.
Example:
Input: 5613

Pass 1: 5613 → 5163 → 5136
Pass 2: 5136 → 1536 → 1356
Pass 3: No swaps required
Sorted Output: 1356

Selection Sort

Definition:
Selection sort selects the smallest element from the unsorted portion and places it at the beginning.
How it works:
Find the minimum element and swap it with the first unsorted position.
Example:
Input: 5613
Step 1: Smallest is 1 → 1563
Step 2: Smallest among remaining is 3 → 1356
Sorted Output: 1356

Insertion Sort

Definition:
Insertion sort builds the sorted list one element at a time.
How it works:
Each element is inserted into its correct position.
Example:
Input: 5613
Insert 5 → 5
Insert 6 → 56
Insert 1 → 156
Insert 3 → 1356
Sorted Output: 1356

Merge Sort

Definition:
Merge sort is a divide-and-conquer algorithm that divides, sorts, and merges lists.
How it works:
The list is split until single elements remain, then merged in sorted order.
Example:
Input: 5613
Split → 56 and 13
Split → 5,6 and 1,3

Merge → 56 and 13
Final Merge → 1356
Sorted Output: 1356

Quick Sort

Definition:
Quick sort is a divide-and-conquer algorithm that partitions the array using a pivot.
How it works:
Elements smaller than the pivot go left, larger go right.
Example:
Input: 5613
Pivot = 5
Left → 1, 3
Right → 6
Combine → 1 3 5 6
Sorted Output: 1356

Different Frameworks under Agentic AI

Agentic AI refers to AI systems that can perceive, reason, plan, act, and learn autonomously to achieve goals. These agents often operate in dynamic environments with feedback loops.

Below are the major frameworks / approaches used in Agentic AI,
1. Reactive Agent Framework
• Agents act only on current perceptions
• No memory of past states
Example: Rule-based systems, reflex agents Use case: Simple games, obstacle avoidance

2. Deliberative Agent Framework
• Agents plan before acting
• Maintain an internal model of the environment Techniques: • Search algorithms • Planning (STRIPS, A*)
Use case: Robotics, route planning

3. Hybrid Agent Framework
• Combines reactive + deliberative agents
• Fast reactions + intelligent planning
Use case: Autonomous vehicles, smart robots

4. Reinforcement Learning–Based Agents

• Agents learn from reward and punishment
• Improve decisions over time
Algorithms: • Q-Learning , DQN ,Policy Gradient
Use case: Games, robotics, recommendation systems

5. Multi-Agent Systems (MAS)
• Multiple agents interact or cooperate
• Agents may be cooperative or competitive
Examples: • Swarm intelligence • Distributed AI
Use case: Traffic control, stock trading bots

Day 32 (18/12 /2025)

- Discussed about project.
- Reviewed the milestone 3.
- Checked the documentation.

Day 33 (19/12 /2025)

- Checked the documentation.

Day 34 (20/12 /2025)

- Informed that the project should be shown on next class.
- Said to work on project.
- Project must be ready for the review.

Day 35 (22/12 /2025)

- Checked and reviewed the completed work by all the teams.
- Gave suggestions and shared some tips.

Day 36 (23/12 /2025)

Q-Learning in Reinforcement Learning

Q-Learning is a popular model-free reinforcement learning algorithm that helps an agent learn how to make the best decisions by interacting with its environment. Instead of needing a model of the environment the agent learns purely from experience by trying different actions and seeing their results.
Imagine a system that sees an apple but incorrectly says, “It’s a mango.” The system is told, “Wrong! It’s an apple.” It learns from this mistake. Next time, when shown the apple, it correctly says “It’s an apple.” This trial-and-error process, guided by feedback is like how Q Learning works.

Q-Learning
The core idea is that the agent builds a Q-table which stores Q-values. Each Q-value estimates how good it is to take a specific action in a given state in terms of the expected future rewards. Over time the agent updates this table using the feedback it receives

Key Components

1. Q-Values or Action-Values
Q-values represent the expected rewards for taking an action in a specific state. These values are updated over time using the Temporal Difference (TD) update rule.

2. Rewards and Episodes The agent moves through different states by taking actions and receiving rewards. The process continues until the agent reaches a terminal state which ends the episode.

3. Temporal Difference or TD-Update
The agent updates Q-values using the formula:
$Q(S,A) \leftarrow Q(S,A)+\alpha(R+\gamma Q(S',A')-Q(S,A))Q(S,A) \leftarrow Q(S,A)+\alpha(R+\gamma Q(S',A')-Q(S,A))$
Where,

- S is the current state.
- A is the action taken by the agent.
- S' is the next state the agent moves to.
- A' is the best next action in state S'.
- R is the reward received for taking action A in state S.
- γ (Gamma) is the discount factor which balances immediate rewards with future rewards. • α (Alpha) is the learning rate determining how much new information affects the old Q-values.

4. ϵ-greedy Policy (Exploration vs. Exploitation)
The ϵ-greedy policy helps the agent decide which action to take based on the current Q value estimates:

- Exploitation: The agent picks the action with the highest Q-value with probability $1-\epsilon 1-\epsilon$. This means the agent uses its current knowledge to maximize rewards.
- Exploration: With probability $\epsilon\epsilon$, the agent picks a random action, exploring new possibilities to learn if there are better ways to get rewards. This allows the agent to discover new strategies and improve its decision-making over time.

How does Q-Learning Works?

Q-learning models follow an iterative process where different components work together to train the agent.

Here's how it works step-by-step:

Q learning algorithm

1. Start at a State (S)

The environment provides the agent with a starting state which describes the current situation or condition.

2. Agent Selects an Action (A)

Based on the current state and the agent chooses an action using its policy. This decision is guided by a Q-table which estimates the potential rewards for different state-action pairs. The agent typically uses an ε-greedy strategy:

- It sometimes explores new actions (random choice).
- It mostly exploits known good actions (based on current Q-values).

3. Action is Executed and Environment Responds

The agent performs the selected action.

The environment then provides:

- A new state (S′) — the result of the action.
- A reward (R) — feedback on the action's effectiveness.

4. Learning Algorithm Updates the Q-Table

The agent updates the Q-table using the new experience:

- It adjusts the value for the state-action pair based on the received reward and the new state.
- This helps the agent better estimate which actions are more beneficial over time.

5. Policy is Refined and the Cycle Repeats

With updated Q-values the agent:

- Improves its policy to make better future decisions.
- Continues this loop — observing states, taking actions, receiving rewards and updating Q-values across many episodes.

Over time the agent learns the optimal policy that consistently yields the highest possible reward in the environment.

Methods for Determining Q-values

1. Temporal Difference (TD):

Temporal Difference is calculated by comparing the current state and action values with the previous ones. It provides a way to learn directly from experience, without needing a model of the environment.

2. Bellman's Equation:

Bellman's Equation is a recursive formula used to calculate the value of a given state and determine the optimal action. It is fundamental in the context of Q-learning and is expressed as:

$Q(s,a)=R(s,a)+\gamma \max aQ(s',a)Q(s,a)=R(s,a)+\gamma maxaQ(s',a)$

Where:

- $Q(s, a)$ is the Q-value for a given state-action pair.
- $R(s, a)$ is the immediate reward for taking action a in state s.
- γ is the discount factor, representing the importance of future rewards.
- $maxaQ(s',a)maxaQ(s',a)$ is the maximum Q-value for the next state s' and all possible actions.

What is a Q-table?

The Q-table is essentially a memory structure where the agent stores information about which actions yield the best rewards in each state. It is a table of Q-values representing the agent's understanding of the environment. As the agent explores and learns from its interactions with the environment, it updates the Q-table. The Q-table helps the agent make informed decisions by showing which actions are likely to lead to better rewards.

Structure of a Q-table:

- Rows represent the states.
- Columns represent the possible actions.
- Each entry in the table corresponds to the Q-value for a state-action pair.

Over time, as the agent learns and refines its Q-values through exploration and exploitation, the Q-table evolves to reflect the best actions for each state, leading to optimal decision making.

Implementation

Here, we implement basic Q-learning algorithm where agent learns the optimal action selection strategy to reach a goal state in a grid-like environment.

Step 1: Define the Environment

We first define the environment including the number of states, actions, goal state and the Q-table. Each state represents a position in a 4×4 grid and actions represent movements in four directions.

```
import numpy as np
import matplotlib.pyplot as plt
n_states = 16
n_actions = 4
goal_state = 15
Q_table = np.zeros((n_states, n_actions))
```

Step 2: Set Hyperparameters

These parameters control the learning process:

- learning_rate (α): How much new info overrides old info.
- discount_factor (γ): How much future rewards are valued.
- exploration_prob (ε): Probability of taking a random action.
- epochs: Number of training episodes.

learning_rate = 0.8
discount_factor = 0.95
exploration_prob = 0.2
epochs = 1000

Step 3: Define the State Transition Function
This function calculates the next state based on the current state and chosen action.
We assume:

- 0 → Left
- 1 → Right
- 2 → Up
- 3 → Down

Day 37 (24/12 /2025)

What is a Depth-First Search in AI?

DFS is a traversing algorithm used in tree and graph-like data structures. It generally starts by exploring the deepest node in the frontier. Starting at the root node, the algorithm proceeds to search to the deepest level of the search tree until nodes with no successors are reached.

DFS Working

Depth-first search (DFS) explores a graph by selecting a path and traversing it as deeply as possible before backtracking.

Search operation of the depth-first search

- Originally it starts at the root node, then it expands its one branch until it reaches a dead end. Then it backtracks to the most recent unexplored node, repeating until all nodes are visited or a specific condition is met. (As shown in the above image, starting from node A, DFS explores its successor B, then proceeds to its descendants until reaching a dead end at node D. It then backtracks to node B and explores its remaining successors i.e E.)
- This systematic exploration continues until all nodes are visited or the search terminates. (In our case after exploring all the nodes of B. DFS explores the right side node i.e C then F and and then G. After exploring the node G. All the nodes are visited. It will terminate.

Key characteristics of DFS

In simple terms, the DFS algorithms in AI holds the power of extending the current path as deeply as possible before considering the other options.

• **Not Cost-Optimal**:

DFS is not cost-optimal because it does not guarantee finding the shortest path to the goal. It may find a solution quickly, but that path might not be the shortest or the cheapest one.

• **Stack-Based Search:**

DFS uses a stack to remember which nodes to explore next. When DFS visits a new node, it adds it to the stack. It keeps going deeper by exploring neighbors. If it reaches a dead end (a node with no more children), it backtracks by removing nodes from the stack and exploring other possible paths.

• **Backtracking Search**:

A variant of DFS is called backtracking search, which uses less memory. Instead of creating all possible paths at once, it generates one child node at a time. It can also modify the current state directly without making a new copy. This saves memory by only keeping one state and the path taken so far. Depth Limited Search for AI

Depth Limited Search (DLS)

Depth Limited Search is a modified version of DFS that imposes a limit on the depth of the search.

How Depth Limited Search Works

1. Initialization: Begin at the root node with a specified depth limit.
2. Exploration: Traverse the tree or graph, exploring each node's children.
3. Depth Check: If the current depth exceeds the set limit, stop exploring that path and backtrack.
4. Goal Check: If the goal node is found within the depth limit, the search is successful.
5. Backtracking: If the search reaches the depth limit or a leaf node without finding the goal, backtrack and explore other branches.

Uniform Cost Search (UCS) in AI

Uniform Cost Search (UCS) is a search algorithm used in artificial intelligence (AI) for finding the least cost path in a graph. It is a variant of Dijkstra's algorithm and is particularly useful when all edges of the graph have different weights, and the goal is to find the path with the minimum total cost from a start node to a goal node.

Key Concepts of Uniform Cost Search

1. Priority Queue:

UCS uses a priority queue to store nodes. The node with the lowest cumulative cost is expanded first. This ensures that the search explores the most promising paths first.

2. Path Cost:

The cost associated with reaching a particular node from the start node. UCS calculates the cumulative cost from the start node to the current node and prioritizes nodes with lower costs.

3. Exploration:

UCS explores nodes by expanding the least costly node first, continuing this process until the goal node is reached. The path to the goal node is guaranteed to be the least costly one.

4. Termination:

The algorithm terminates when the goal node is expanded, ensuring that the first time the goal node is reached, the path is the optimal one.

Bidirectional Search

Bidirectional search is a graph search algorithm which find smallest path from source to goal vertex. It runs two simultaneous search –

1. Forward search from source/initial vertex toward goal vertex
2. Backward search from goal/target vertex toward source vertex.

Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs – one starting from initial vertex and other starting from goal vertex. The search terminates when two graphs intersect.

Just like A* algorithm, bidirectional search can be guided by a heuristic estimate of remaining distance from source to goal and vice versa for finding shortest path possible.

When to use bidirectional approach?

We can consider bidirectional approach when-

1. Both initial and goal states are unique and completely defined.
2. The branching factor is exactly the same in both directions.

Performance measures

- Completeness: Bidirectional search is complete if BFS is used in both searches.
- Optimality: It is optimal if BFS is used for search and paths have uniform cost.
- Time and Space Complexity: Time and space complexity is O(bd/2).

Beam Search Algorithm

Beam Search is a heuristic search algorithm that navigates a graph by systematically expanding the most promising nodes within a constrained set. This approach combines elements of breadth-first search to construct its search tree by generating all successors at each level. However, it only evaluates and expands a set number, W, of the best nodes at each level, based on their heuristic values. This selection process is repeated at each level of the tree.

Characteristics of Beam Search

• **Width of the Beam (W):** This parameter defines the number of nodes considered at each level. The beam width W directly influences the number of nodes evaluated and hence the breadth of the search.

• **Branching Factor (B):** If B is the branching factor, the algorithm evaluates W×BW×B nodes at every depth but selects only W for further expansion.

• **Completeness and Optimality:** The restrictive nature of beam search, due to a limited beam width, can compromise its ability to find the best solution as it may prune potentially optimal paths.

• **Memory Efficiency:** The beam width bounds the memory required for the search, making beam search suitable for resource-constrained environments.

How Beam Search Works?

The process of Beam Search can be broken down into several steps:

1. Initialization: Start with the root node and generate its successors.

2. Node Expansion: From the current nodes, generate successors and apply the heuristic function to evaluate them.

3. Selection: Select the top W nodes according to the heuristic values. These selected nodes form the next level to explore.

4. Iteration: Repeat the process of expansion and selection for the new level of nodes until the goal is reached or a certain condition is met (like a maximum number of levels).

5. Termination: The search stops when the goal is found or when no more nodes are available to expand.

Day 38 (26/12 /2025)

Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a probabilistic model that assumes data points are generated from a mixture of several Gaussian (normal) distributions with unknown parameters.

Working of Gaussian Mixture Model

A Gaussian Mixture Model assumes that the data is generated from a mixture of K Gaussian distributions, each representing a cluster.

Every Gaussian has its own mean μkμk, covariance ΣkΣk and mixing weight πkπk.

1. Posterior Probability (Cluster Responsibility)
2. Likelihood of a Data Point

3. Expectation-Maximization (EM) Algorithm
4. Log-Likelihood of the Mixture Model

Backpropagation in Neural Network

Backpropagation stands for Backward Propagation of Errors.
It is an algorithm used to train neural networks by reducing the error between the predicted output and the actual output.
In this method, the error is passed backward through the network, and weights and biases are updated using the chain rule of calculus. This helps the network learn from its mistakes and improve its performance over time.

Why is Backpropagation Important?

Backpropagation helps neural networks learn efficiently.
• **Efficient Weight Update:** It calculates the gradient of the loss function with respect to each weight, allowing accurate and efficient weight updates.
• **Scalability:** The algorithm works well with multiple hidden layers, making deep neural networks possible.
• **Automated Learning:** The network automatically adjusts its weights to minimize error without manual intervention.

Working of Backpropagation Algorithm

Backpropagation consists of two main steps:

1. Forward Pass

In the forward pass, the network generates a prediction.

- Input data is given to the input layer
- Inputs are multiplied by their respective weights
- A bias is added
- An activation function (such as ReLU) is applied
- The output of one layer becomes the input of the next layer
- The final layer produces the predicted output

This step is used only to compute the output.

2. Backward Pass

In the backward pass, the network learns from the error.

- The difference between predicted output and actual output is calculated
- This difference is called error or loss
- The error is propagated backward through the network
- Gradients are calculated using the chain rule
- Weights and biases are updated to reduce the error

Day 39 (27/12 /2025)

PCA:

PCA (Principal Component Analysis) model is a statistical technique for dimensionality reduction, transforming large, complex datasets into a smaller set of new, uncorrelated variables (principal_components) that capture most of the original information, primarily by identifying directions of maximum data variance, making data easier to visualize, analyze, and use in machine learning by reducing redundancy and preventing overfitting. It works by finding orthogonal axes (eigenvectors) with the highest variance (eigenvalues) and projecting the data onto these new axes, effectively simplifying high-dimensional data while preserving essential patterns.

How a PCA Model Works

1. **Standardize Data**: Scale features to have a mean of 0 and variance of 1, ensuring all features contribute equally.
2. **Compute Covariance Matrix**: Calculate how variables change together.
3. **Calculate Eigenvalues & Eigenvectors**: Find the directions (eigenvectors) of greatest variance and their importance (eigenvalues) from the covariance matrix.
4. **Select Principal Components**: Sort components by eigenvalues (descending) and pick the top ones that explain a desired percentage of total variance (e.g., 95%).
5. **Transform Data**: Project the original data onto the selected principal components to get the reduced dataset.

Day 40 (29/12 /2025)

- Discussed about project.
- Worked on project.
- Discussed about the next milestone.

Day 41 (30/12 /2025)

- Discussed about project.
- Reviewed the milestone 4.
- Checked the documentation.
- PPT presentation.

Day 42 (31/12 /2025)

- Reviewed the milestone 4.

Day 43 (02/01 /2026)