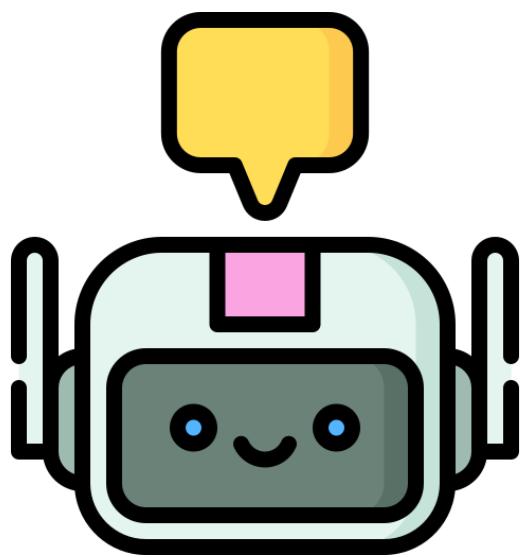


AI Agents

AI Agents Part 1 – Tutorial

Daniel



AI Agents Part - 1

AI Agents

1. What is "AI" in AI Agent?	3
2. What is "Agent" in AI Agent?	3
3. What is AI Agent?	3
4. Real-world AI agent examples	4
4.1. Self-Driving Cars (Tesla, Waymo)	4
4.2. Chatbots in Banking (HDFC's EVA, SBI's SIA)	4
4.3. Stock Trading Bots (Zerodha's Streak, Robinhood's AI Trading)	4
4.4. Healthcare AI (IBM Watson, Google DeepMind)	4
4.5. Fraud Detection in Banks (PayPal, Mastercard AI)	5
5. Key Characteristics of AI Agents	5
6. AI Agent Categories Based on Functionality	5
6.1. Reactive Agents	5
6.2. Deliberative Agents	5
6.3. Learning Agents	5
6.4. Multi-Agent Systems	5
7. The Need for AI Agents	6
7.1. Software development perspective	6
7.2. Autonomous system perspective	15
6. Core Components of AI Agents	18
6.1. Role-Playing	19
6.2. Focus	20
6.3. Tools	21
6.4. Cooperation	22
6.5. Guardrails	23
6.6. Memory	24
7. Crewai Introduction	25
8. ollama Introduction	26
9. Creating AI Agents	27
10. Environment Setup	27
11. Creating Agents in CrewAI	29
12. Single-Agent vs. Multi-Agent Workflow in CrewAI	29

AI Agents Part - 1

12.1. Single-Agent Workflow	29
12.2. Multi-Agent Workflow	29
13. AI Agents technical flow.....	30
14. Hello World Example: AI Research & Writing with CrewAI	30
14.1. AI Agents technical flow for Simple Greeting AI using CrewAI.....	30
15. Use Case 1: Automated Content Creation using CrewAI	33
15.1. AI Agents technical flow for Automated content creation using CrewAI	33
16. Use Case 2: Automated Healthcare AI Summary with CrewAI.....	41
16.1. AI Agents technical flow for healthcare content creation using CrewAI.....	41
17. Use Case 3: Multi-Agent Research Assistant with CrewAI.....	47
17.1. AI Agents technical flow for Multi-Agent Research Assistant with CrewAI.....	47

AI Agents

1. What is "AI" in AI Agent?

- ✓ AI refers to machines or software that can perform tasks that typically require human intelligence.
- ✓ These tasks include learning, reasoning, problem-solving, understanding language, and recognizing patterns.

2. What is "Agent" in AI Agent?

- ✓ An agent is something that can sense its environment, make decisions, and take actions to achieve a goal.
- ✓ An AI agent is a system that perceives the world (through sensors, data, or inputs), processes that information, and takes actions accordingly.

3. What is AI Agent?

- ✓ An AI agent is a smart system that interacts with its surroundings, learns from experience, and makes decisions to complete tasks efficiently.
- ✓ They can be software-based (like virtual assistants) or embodied in hardware (like robots).

AI Agent = AI + Agent

4. Real-world AI agent examples

- ✓ AI agents are widely used across industries to improve efficiency and automate tasks. Some key examples included here.

4.1. Self-Driving Cars (Tesla, Waymo)

- ✓ **AI Role:** Detects roads, traffic signals, pedestrians, and other vehicles using sensors and cameras.
- ✓ **Agent Behavior:** Makes driving decisions like stopping, turning, or changing lanes safely.

4.2. Chatbots in Banking (HDFC's EVA, SBI's SIA)

- ✓ **AI Role:** Understands customer queries about balances, transactions, and loans.
- ✓ **Agent Behavior:** Responds with relevant answers or directs users to human support.

4.3. Stock Trading Bots (Zerodha's Streak, Robinhood's AI Trading)

- ✓ **AI Role:** Analyses stock market trends, news, and data.
- ✓ **Agent Behavior:** Buys or sells stocks based on predicted profitability.

4.4. Healthcare AI (IBM Watson, Google DeepMind)

- ✓ **AI Role:** Reads medical reports, diagnoses diseases, and suggests treatments.
- ✓ **Agent Behavior:** Assists doctors in decision-making for better patient care.

4.5. Fraud Detection in Banks (PayPal, Mastercard AI)

- ✓ **AI Role:** Scans millions of transactions in real-time to find fraud patterns.
- ✓ **Agent Behavior:** Flags suspicious activity and prevents fraudulent transactions.

5. Key Characteristics of AI Agents

- ✓ **Autonomy** : Operate independently.
- ✓ **Perception** : Collect data from inputs.
- ✓ **Decision-making** : Analyze and act.
- ✓ **Action** : Execute tasks like responding or controlling systems.
- ✓ **Adaptability** : Improve over time.

6. AI Agent Categories Based on Functionality

6.1. Reactive Agents

- ✓ Respond to inputs without memory (e.g., rule-based chatbots).

6.2. Deliberative Agents

- ✓ Use reasoning and planning to make decisions (e.g., self-driving car AI).

6.3. Learning Agents

- ✓ Improve through experience (e.g., recommendation systems).

6.4. Multi-Agent Systems

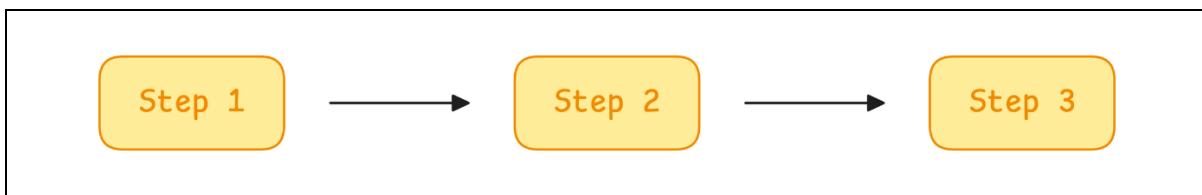
- ✓ Multiple AI agents working together (e.g., swarm robotics).

7. The Need for AI Agents

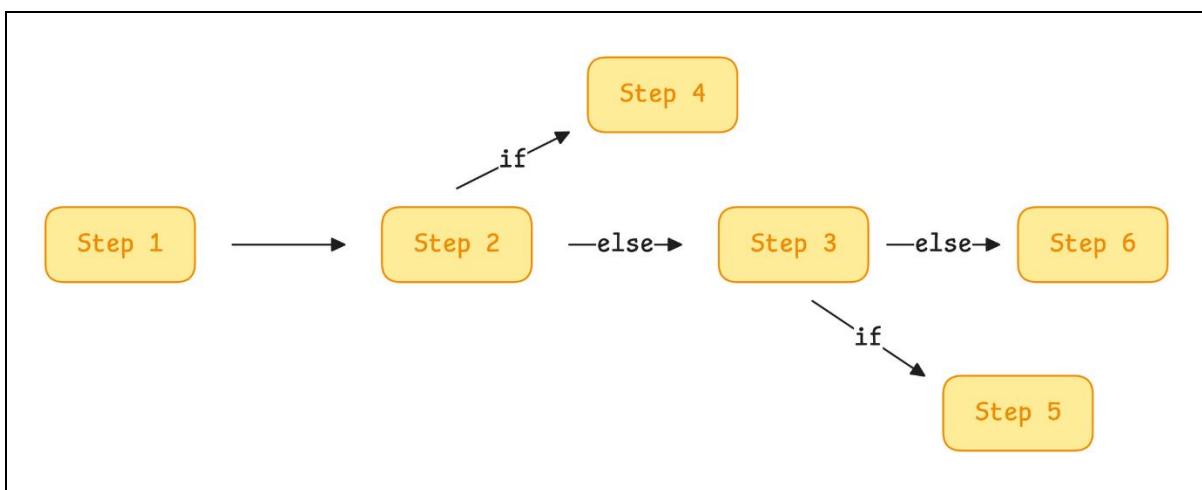
- ✓ There are several key perspectives to understanding the motivation behind AI agents:
 1. Software Development Perspective
 2. Autonomous Perspective

7.1. Software development perspective

- ✓ Think about traditional software applications, they operate based on strict, predefined rules.
 - If a program needs to complete a task, every step must be explicitly defined.



- As new scenarios arise, developers constantly need to add more conditions and logic.



- Over time, the system becomes more difficult to scale and maintain.

Traditional automation

- ✓ In other words, traditional automation follows strict predefined logic:
 - If condition A is met, do X.
 - If condition B is met, do Y.
 - Otherwise, do Z.
- ✓ Inputs have a predefined data type (text, number, etc.).
- ✓ Transformations on the input are mostly fixed.
- ✓ Output types are also fixed.



When to Stick with Traditional Automation?

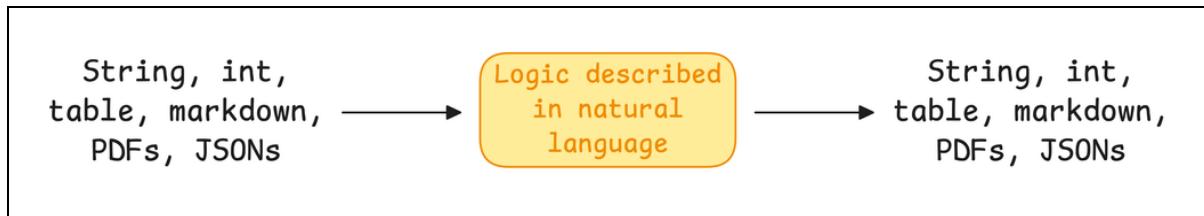
- ✓ This isn't a criticism of traditional automation, and we're not discouraging it.
- ✓ If it works for your needs, stick with it.
- ✓ There's no need to build AI agents unless truly necessary.

Why AI Agents Are Different?

- ✓ No need for explicit instructions.
- ✓ Gather information dynamically.
- ✓ Use reasoning for ambiguous problems.
- ✓ Collaborate with other agents to complete tasks.
- ✓ Leverage external tools for real-time decisions.

How AI Agents Offer More Flexibility?

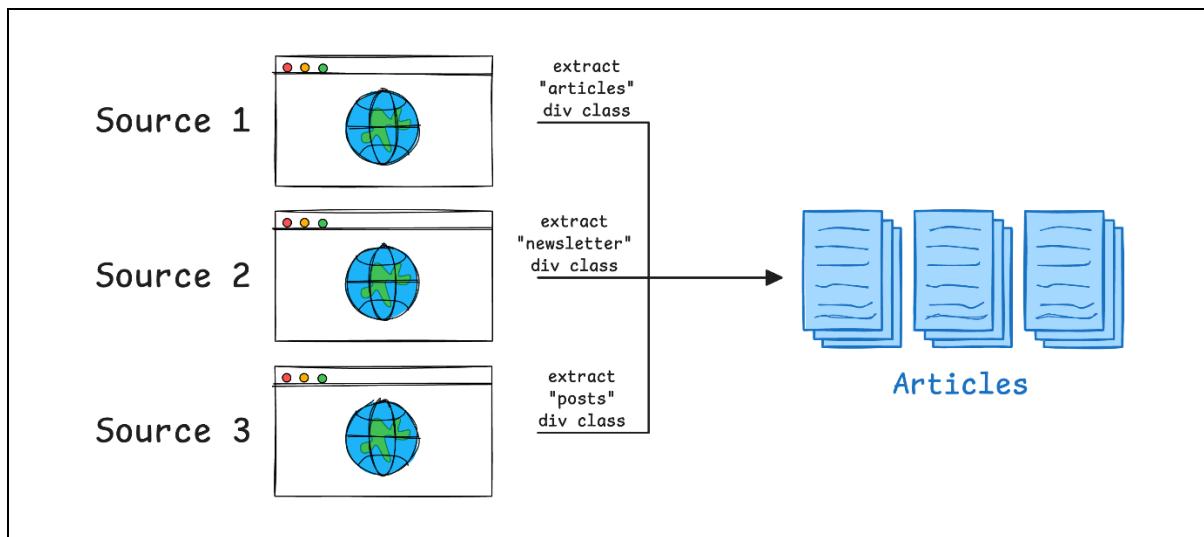
- ✓ Inputs are not limited to a specific data type.
- ✓ They can be text, numbers, PDFs, tables, markdown, JSON, and more.



- ✓ Transformations are flexible and depend on what you ask the LLM to do.
- ✓ Outputs can vary, tokens, lists, structured data, JSON, code, and more.

Traditional Approach: Manual Effort Required

- ✓ Imagine managing a news aggregation platform that collects articles from various sources.
- ✓ Traditionally, we need to manually extract and process data from different sources



- ✓ Write **scripts** to scrape multiple websites.
- ✓ Filter and categorize articles using hardcoded rules.
- ✓ Manually verify whether an article is relevant.

Limitation

- ✓ If a website changes its layout, your scraper breaks.
- ✓ If new topics arise, you must update the filtering logic manually.

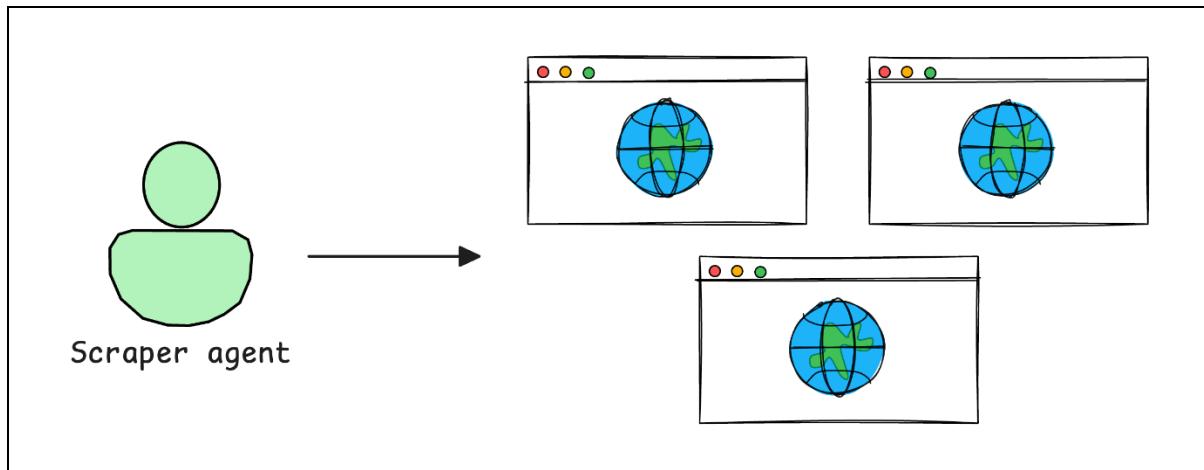
AI Agent Approach: Automation & Intelligence

How AI Agents Work Together?

- ✓ AI agents collaborate to automate the entire news processing workflow from data collection to verification and summarization.
- ✓ Each agent plays a specific role in ensuring accuracy, relevance, and efficiency:
 1. A Web Scraper Agent
 2. A Topic Analysis Agent
 3. A Fact-Checking Agent
 4. A Summarization Agent

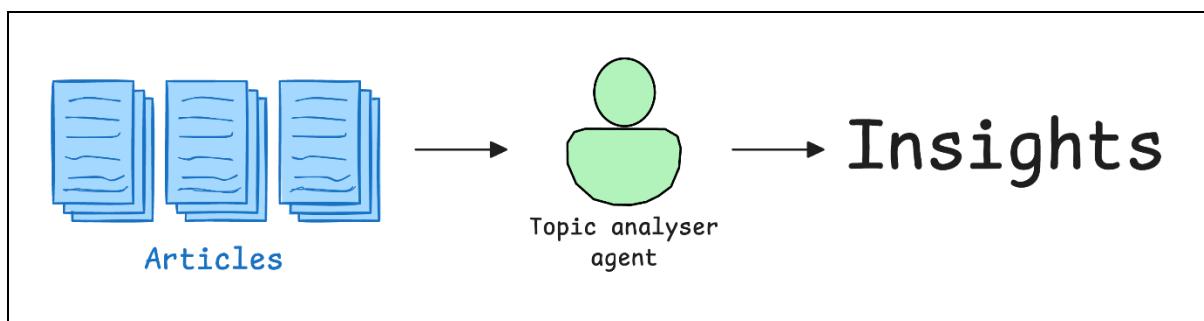
1. A Web Scraper Agent

- ✓ Automatically discovers new sources.
- ✓ Adapts to changes in web page structures without manual intervention.



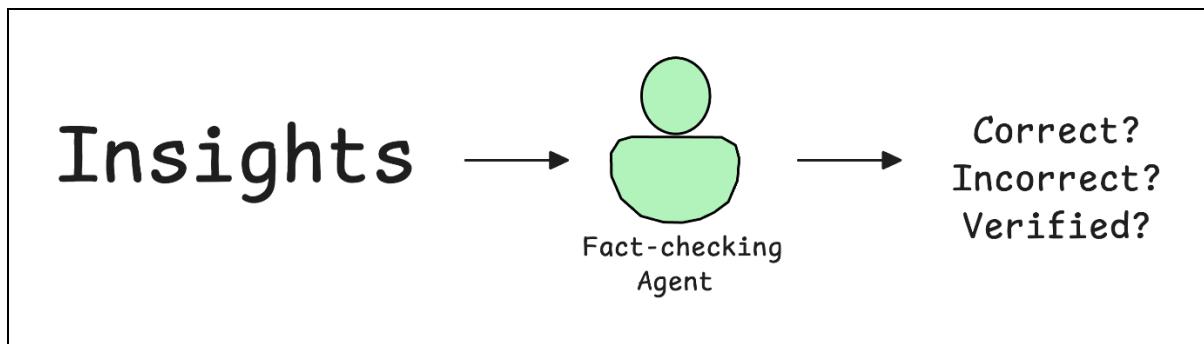
2. A Topic Analysis Agent

- ✓ Analyzes articles in real time.
- ✓ Detects emerging trends.
- ✓ Classifies content efficiently.



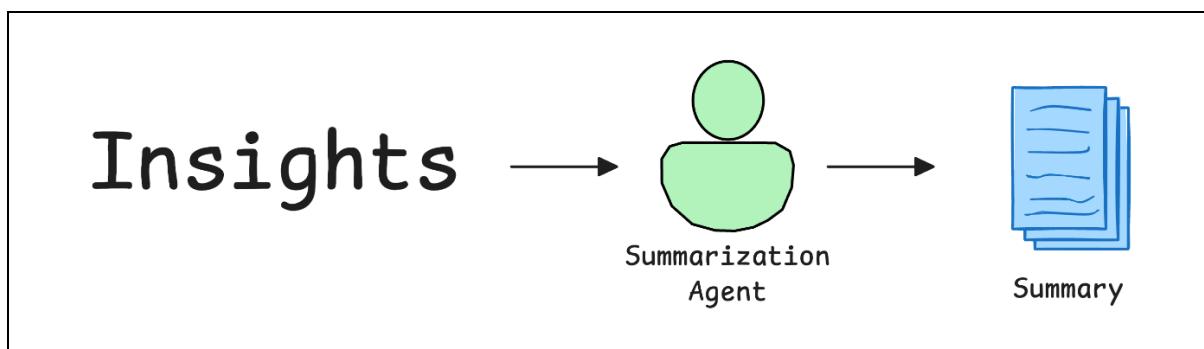
3. A Fact-Checking Agent

- ✓ It ensures article credibility by:
 - Cross-referencing external data sources.
 - Verifying claims for accuracy.
 - Detecting misinformation before publication.



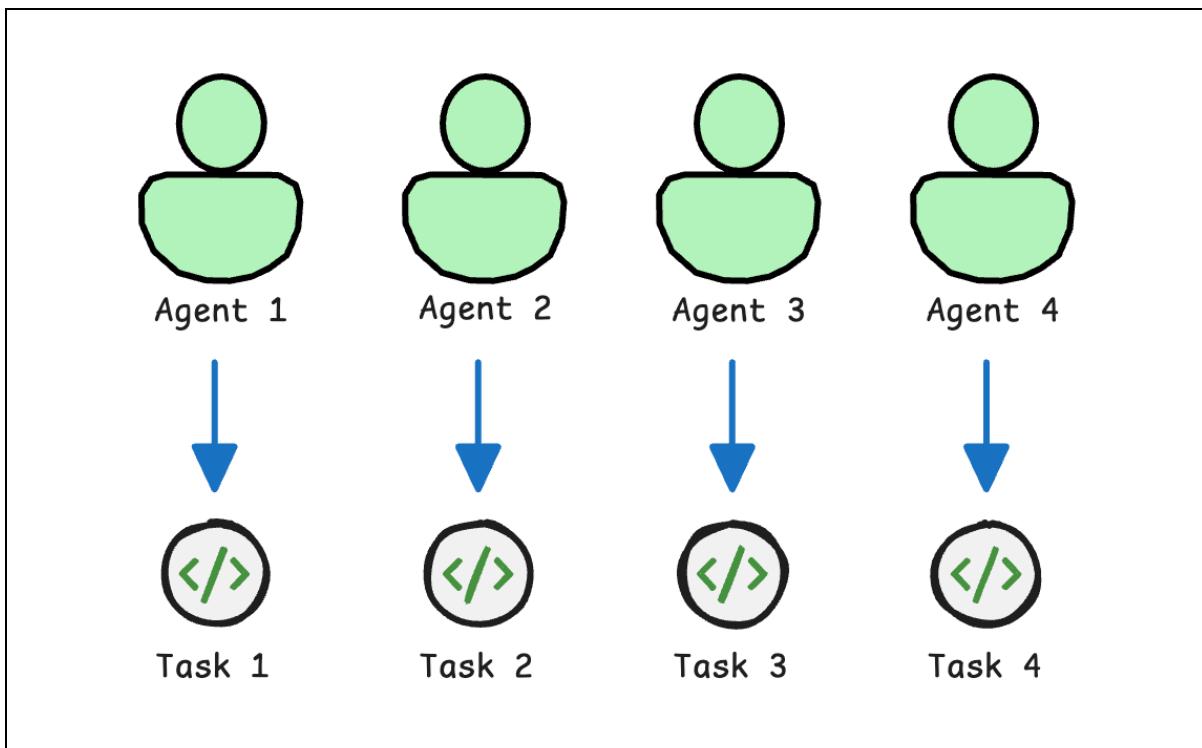
4. A Summarization Agent

- ✓ Extracts key points from articles.
- ✓ Generates concise, easy-to-read summaries.



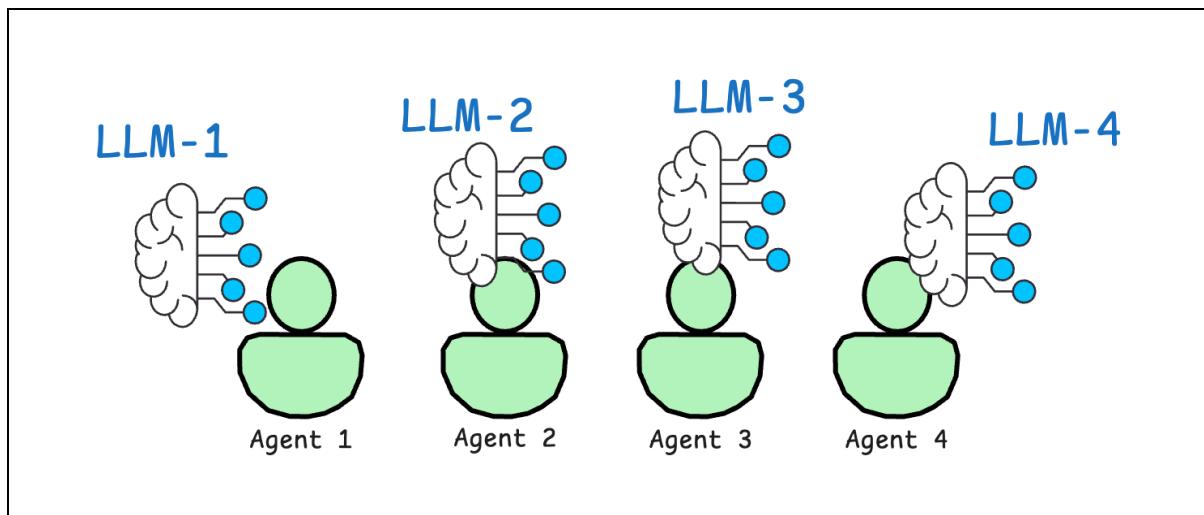
Multi-agent framework

- ✓ Multi-agent means, it's a system where multiple AI agents work together.
- ✓ Each agent has a specific role but collaborates to achieve a common goal.



Optimizing Performance with Multiple LLMs

- ✓ Each agent uses a specialized LLM for its task.
- ✓ Selecting the best LLM enhances efficiency and accuracy.



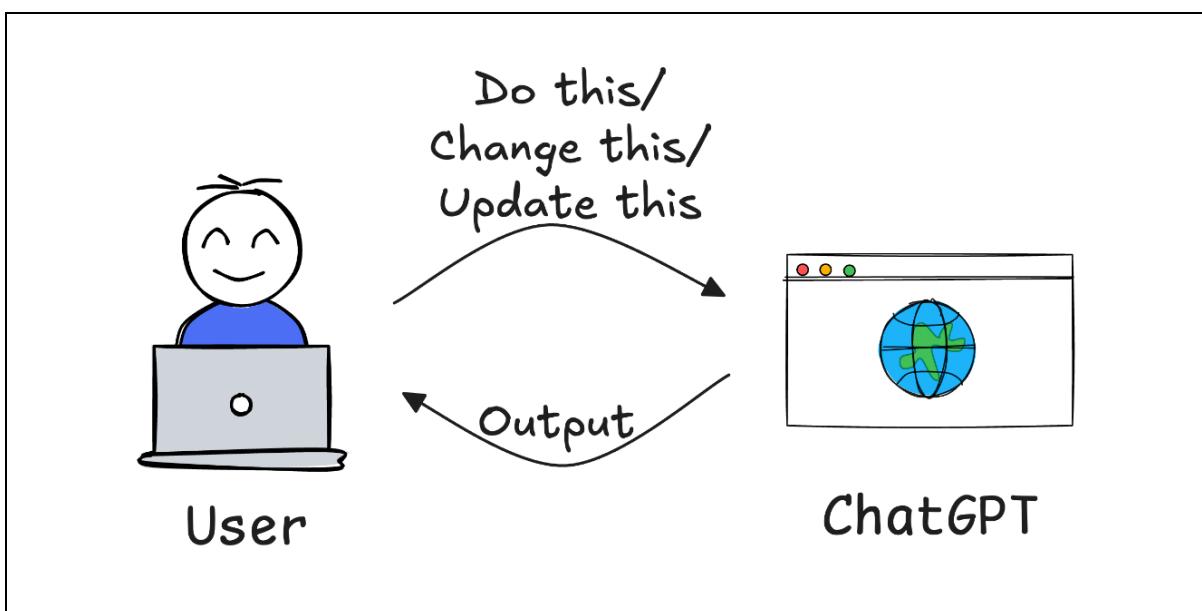
AI Agents: Learning and Adapting

- ✓ AI agents continuously learn, adapt, and optimize.
- ✓ They update automatically without manual intervention.
- ✓ Beyond fixed rules, they collaborate and solve problems autonomously.

7.2. Autonomous system perspective

Limitations of Traditional LLM Interactions

- ✓ Users must refine outputs manually.
- ✓ The process is iterative and time-consuming:
 - Ask a question.
 - Review the response.
 - Adjust the prompt.
 - Repeat until satisfied



The Challenge of Using a Standard LLM for Research

- ✓ Imagine you need a report on the latest AI research trends. With a standard LLM, you would:
 - Ask for a summary of recent AI papers.
 - Review the response and realize you need sources.
 - Request citations and get a list of papers.
 - Notice outdated sources and refine your query.
 - Repeat the process until you get a useful report.
- ✓ This iterative process takes time and effort, requiring you to make decisions at every step.

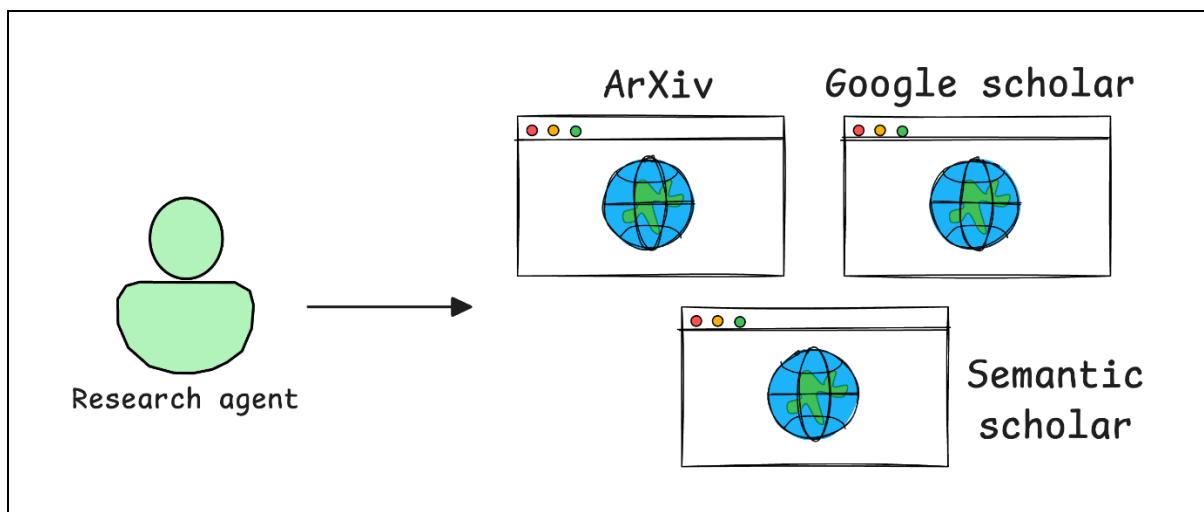
AI Agents Part - 1

AI Agents in Research

- ✓ Automate data gathering, analysis, and verification.
- ✓ Reduce the need for constant user input.
- ✓ Enhance efficiency and accuracy in research.

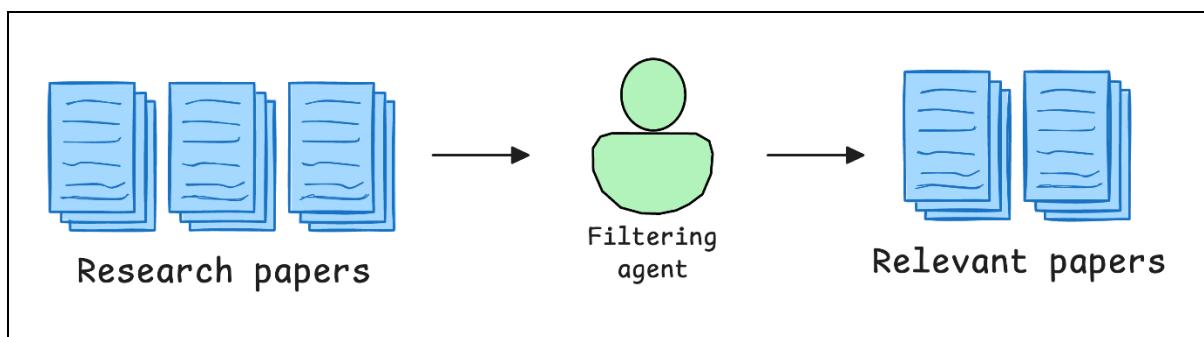
1. A Research Agent

- ✓ Searches for AI research papers automatically.
- ✓ Retrieves data from sources like arXiv, Semantic Scholar, or Google Scholar.
- ✓ Ensures access to the latest credible information.



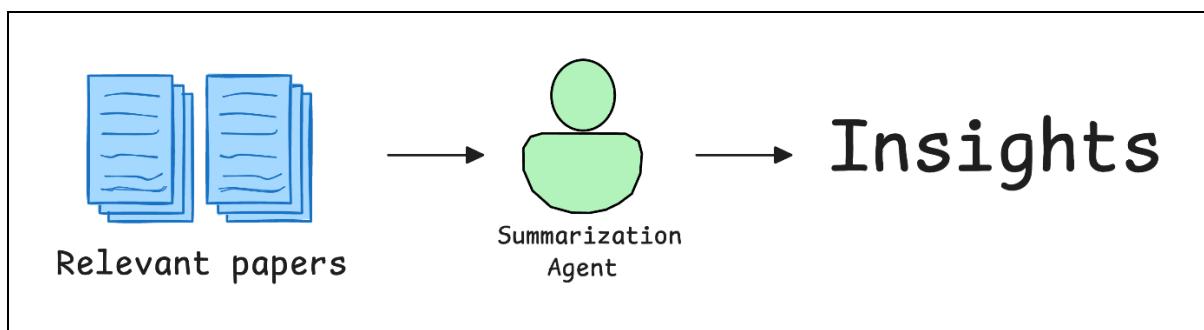
2. A Filtering Agent

- ✓ Analyzes retrieved papers for relevance.
- ✓ Filters based on citation count, publication date, and keywords.
- ✓ Ensures only high-quality research is considered.



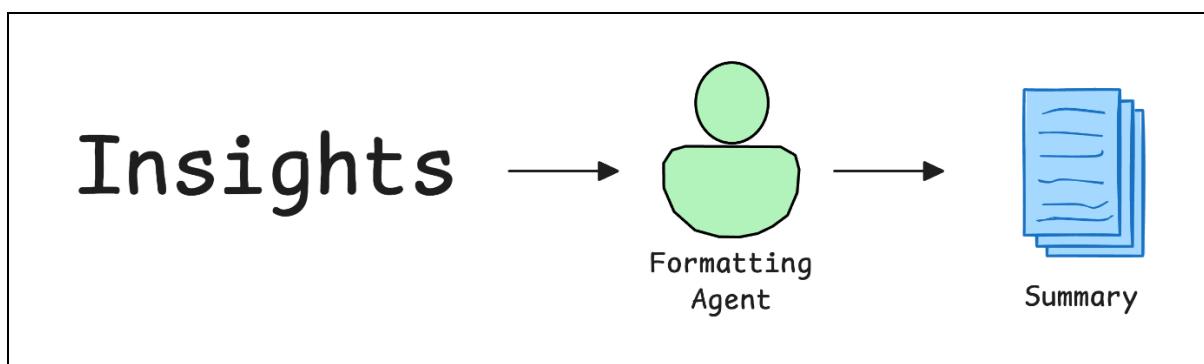
3. A Summarization Agent

- ✓ Extracts key insights from research papers.
- ✓ Compiles information into a concise, easy-to-read report.
- ✓ Saves time by presenting only the most relevant details.



4. A Formatting Agent

- ✓ Organizes the final report for clarity.
- ✓ Ensures a professional and structured layout.
- ✓ Enhances readability and presentation.



6. Core Components of AI Agents

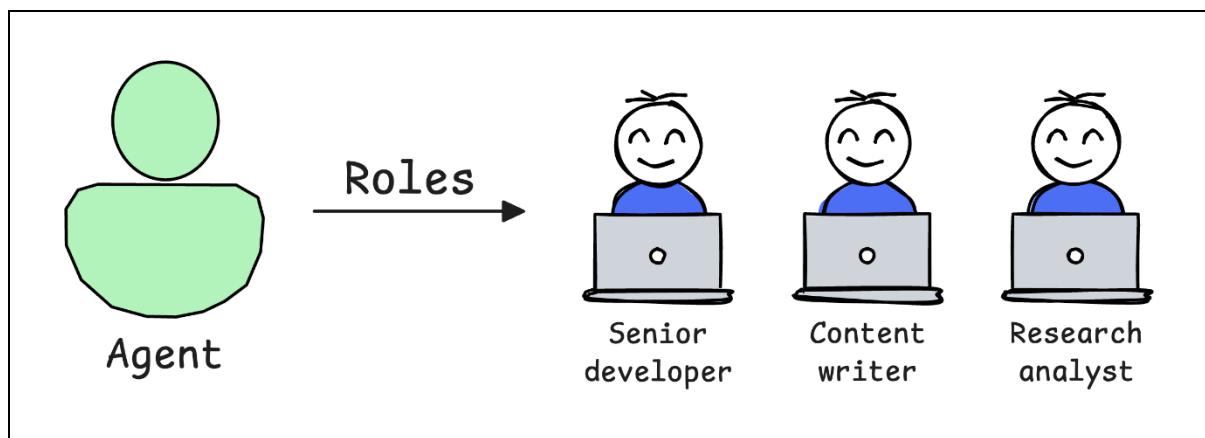
- ✓ AI agents' reason, plan, and act autonomously.
- ✓ Six key building blocks ensure reliability and intelligence.
- ✓ Essential for real-world effectiveness.

Key Elements of AI Agents

1. Role-playing
2. Focus
3. Tools
4. Cooperation
5. Guardrails
6. Memory

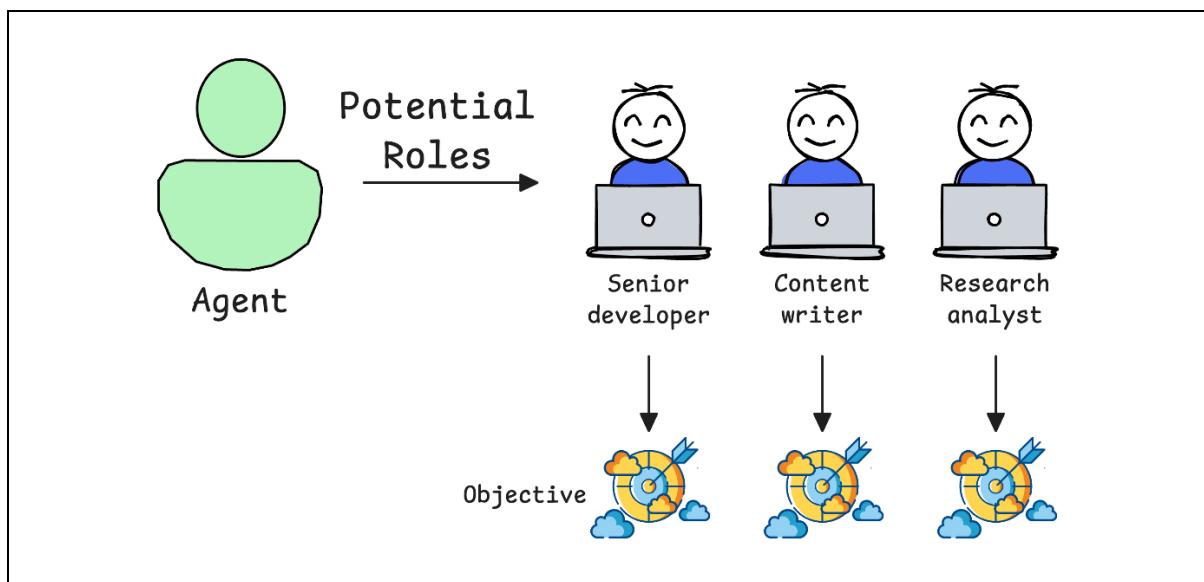
6.1. Role-Playing

- ✓ Specific roles enhance task specialization.
- ✓ Defined roles improve structured and relevant responses.
- ✓ Clear identity and objectives ensure better-aligned outputs.



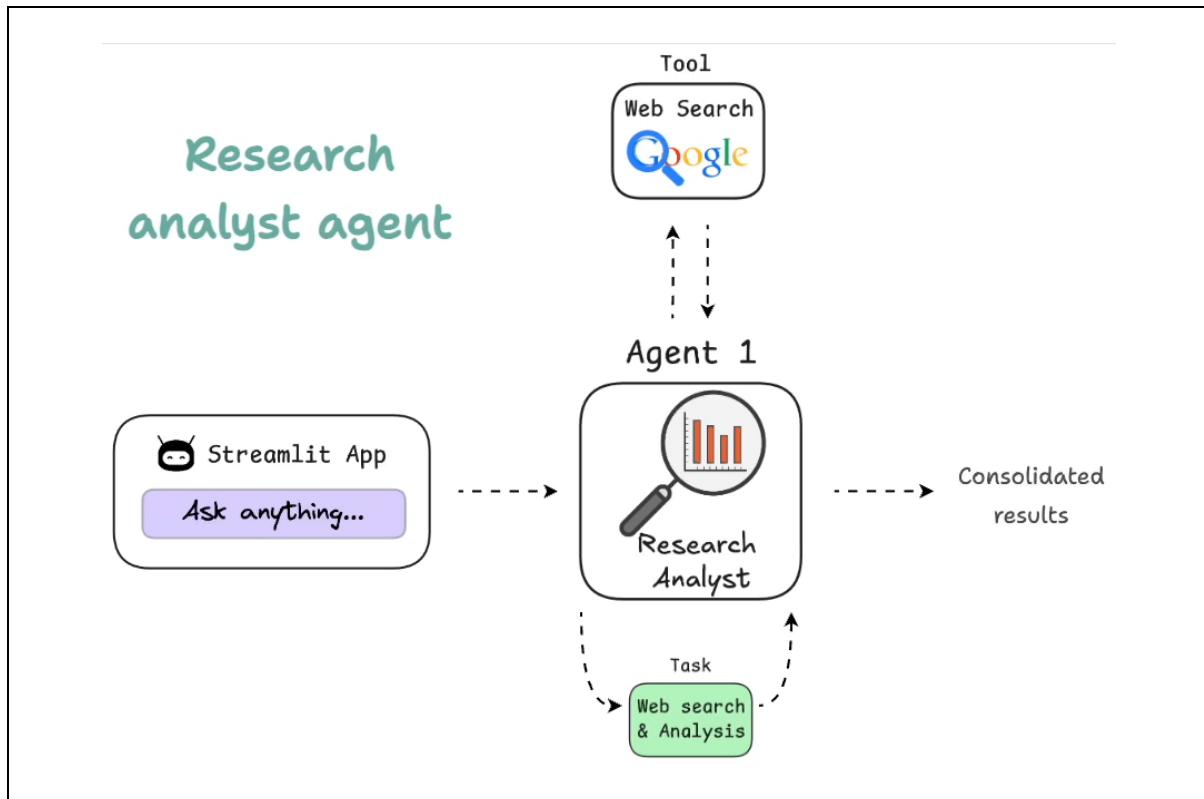
6.2. Focus

- ✓ Ensures agents stay aligned with their objectives.
- ✓ Minimizes errors and improves accuracy.
- ✓ More data isn't always better—structured input leads to better results.



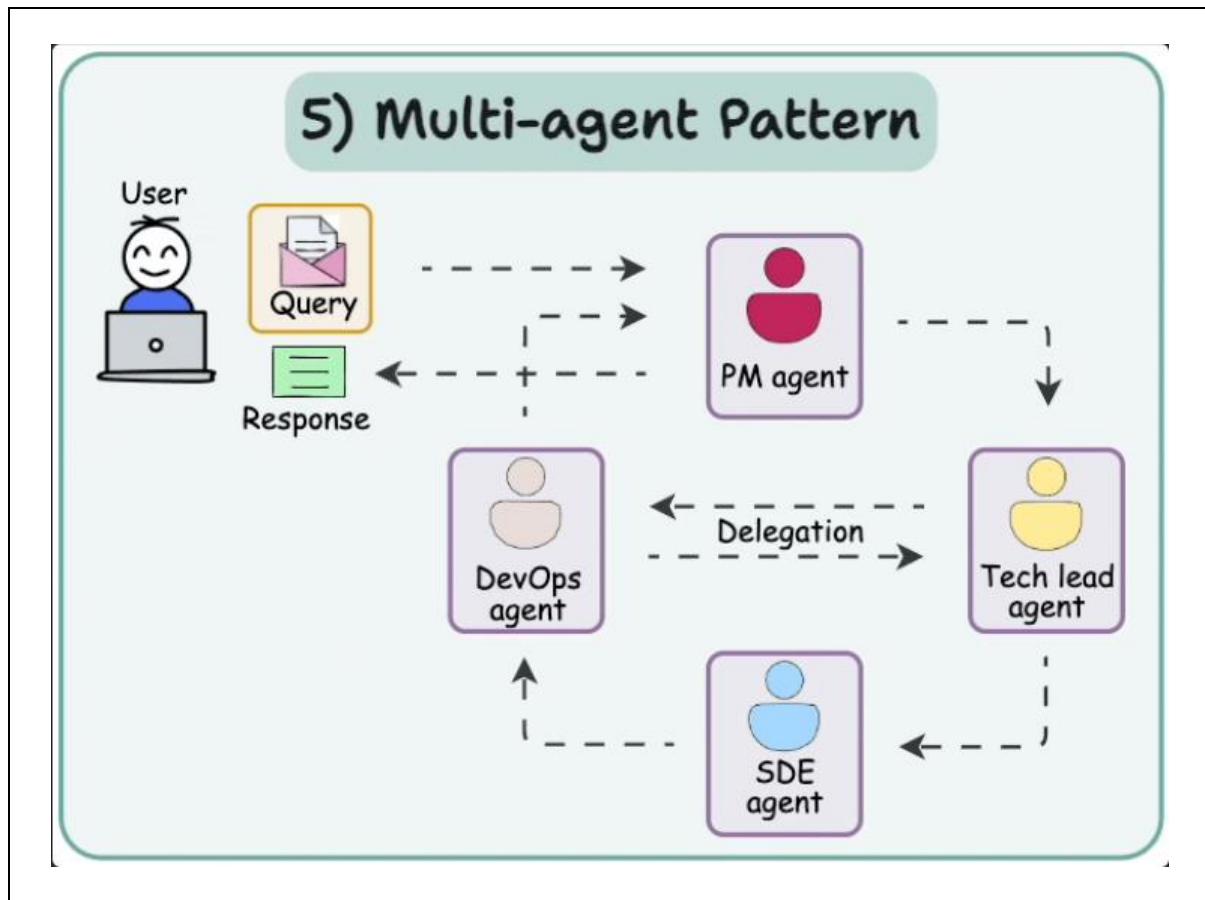
6.3. Tools

- ✓ Enhance AI agents beyond text-based reasoning.
- ✓ Enable real-time data retrieval and system interactions.



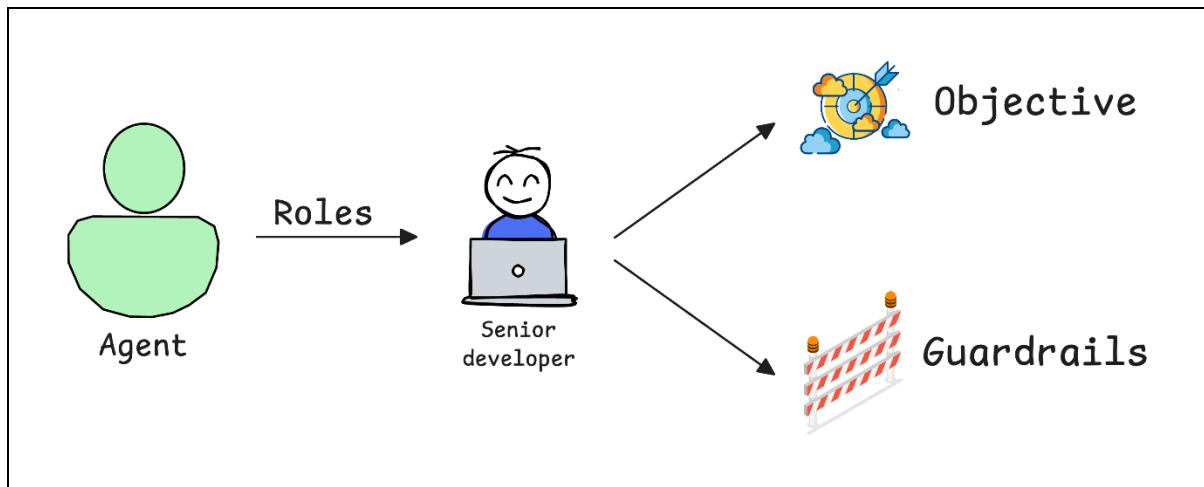
6.4. Cooperation

- ✓ Enabling Multi-Agent Collaboration
- ✓ Enables AI agents to work together efficiently.
- ✓ Enhances performance through collaboration and feedback exchange.



6.5. Guardrails

- ✓ Keeping AI Agents Reliable
- ✓ Implement safety measures for controlled behavior.
- ✓ Prevent false information, infinite loops, and unreliable decisions.

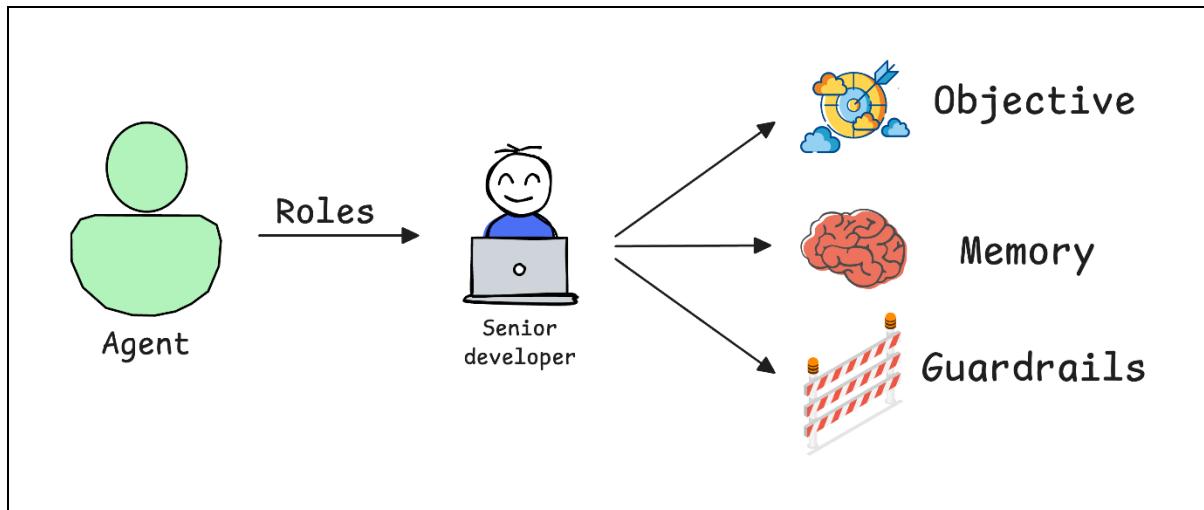


Examples of Effective Guardrails

- ✓ **Limit Tool Usage:** Prevents excessive API calls or irrelevant queries.
- ✓ **Validation Checkpoints:** Ensures outputs meet quality standards before proceeding.
- ✓ **Fallback Mechanisms:** Enables human or agent intervention when needed.

6.6. Memory

- ✓ **Without memory:** Agents reset every session, leading to repetitive interactions.
- ✓ **With memory:** Agents retain context, refine responses, and improve decision-making over time.



7. CrewAI Introduction

- ✓ It's a python framework for building multi-agent AI systems.
- ✓ By using this we can create specialized agents with unique roles, collaborate, specialize, and automate tasks.

How CrewAI Enhances AI Agents

- ✓ **Multi-Agent Collaboration:** Agents work together efficiently.
- ✓ **Specialization:** Assigns specific roles for better performance.
- ✓ **Task Automation:** Reduces manual intervention.
- ✓ **Memory & Context:** Enables continuous learning.
- ✓ **Tool Integration:** Uses external tools for enhanced capabilities.
- ✓ **Accuracy & Efficiency:** Delivers structured and reliable outputs.

Adaptive, goal-driven AI agents

- ✓ With CrewAI, we can efficiently design **adaptive, goal-driven AI agents** that collaborate and execute complex tasks independently.

8. ollama Introduction

- ✓ Ollama enables running and customizing LLMs locally.
- ✓ Avoids cloud dependency by supporting open-source AI models.
- ✓ Optimized for efficient local deployment.

How Ollama Enhances AI Agents?

- ✓ **Local Execution:** No cloud dependency, ensuring privacy.
- ✓ **Optimized Performance:** Faster inference with minimal resources.
- ✓ **Seamless Model Management:** Load, switch, and fine-tune easily.
- ✓ **Lightweight & Scalable:** Works on both edge devices and enterprises.
- ✓ **AI Agent Compatibility:** Supports autonomous agents with specialized models.

9. Creating AI Agents

- ✓ Now, let's move to implementation! Using CrewAI, an open-source framework, we can:
 - **Assign Roles:** Specialize agents for specific tasks.
 - **Set Goals:** Define clear objectives for each agent.
 - **Integrate Tools:** Equip agents with the right resources.
 - **Use Any LLM:** Seamlessly work with various language models.

10. Environment Setup

- ✓ To get started, install CrewAI as follows:

- pip install crewai
 - pip install crewai-tools

Choosing the Right LLM for CrewAI

- ✓ Ollama serves LLMs locally for seamless performance.
- ✓ CrewAI supports multiple providers, including:
 - OpenAI
 - Gemini
 - Groq
 - Azure
 - Fireworks AI
 - Cerebras
 - SambaNova & more!
- ✓ This flexibility lets you pick the best model for your needs.

Setting Up OpenAI with CrewAI

- ✓ To configure OpenAI as your LLM provider, create a `.env` file in your project's directory and add your API key like this:

- `OPENAI_API_KEY = your-api-key-here`

- ✓ Make sure to replace `your-api-key-here` with your actual OpenAI API key.
- ✓ This allows CrewAI to authenticate and interact with OpenAI's models seamlessly.

Installing and Running Ollama

- ✓ Download Ollama: Get it from official website <https://ollama.com>
- ✓ Install – Follow the on-screen setup instructions.
- ✓ Verify Installation run: `ollama --version`
- ✓ Download a Model: `ollama pull llama3.2`
- ✓ Run Locally: `ollama run llama3.2`
- ✓ Ready to Use – Serve LLMs locally without external APIs. Now, let's build our AI agents!

11. Creating Agents in CrewAI

- ✓ In CrewAI, Agents are autonomous entities with:
 - **Role:** Defines function (e.g., "Senior Technical Writer").
 - **Goal:** Specifies objective (e.g., "Write a publication-ready article").
 - **Backstory:** Adds expertise for better interactions.
- ✓ These attributes enable agents to perform tasks efficiently and independently.

12. Single-Agent vs. Multi-Agent Workflow in CrewAI

12.1. Single-Agent Workflow

- ✓ Only one agent is assigned a task.
- ✓ Handles the entire process independently.
- ✓ Simple, efficient, and best for focused tasks (e.g., writing an article).

12.2. Multi-Agent Workflow

- ✓ Multiple agents collaborate on different tasks.
- ✓ Workflows are divided (e.g., research, writing, review).
- ✓ Ideal for complex projects requiring diverse expertise.

13. AI Agents technical flow

- ✓ **Step 1:** Import Libraries.
- ✓ **Step 2:** Configure the LLM
- ✓ **Step 3:** Create Agents:
 - **Agent 1**
 - **Agent 2, etc**
- ✓ **Step 4:** Define Tasks
- ✓ **Step 5:** Create a Crew
- ✓ **Step 6:** Kickoff the Crew
- ✓ **Step 7:** Check the Response

14. Hello World Example: AI Research & Writing with CrewAI

- ✓ Simple AI workflow using CrewAI to generate a friendly greeting.
- ✓ We are using **a single-agent** workflow.

14.1. AI Agents technical flow for Simple Greeting AI using CrewAI

- ✓ **Step 1:** Import Libraries, Essential for using CrewAI and LLMs.
- ✓ **Step 2:** Configure the LLM, Define the Ollama LLaMA 3.2 language model.
- ✓ **Step 3:** Create Agent, Define its role, goal, and backstory.
 - **Agent 1:** Friendly AI, Responds with a simple greeting.
- ✓ **Step 4:** Define Task, Assign a specific task to the agent.
 - **Task:** Say "Hello, World!"
- ✓ **Step 5:** Create a Crew, Group the agent and assign the task.
- ✓ **Step 6:** Kickoff the Crew, Execute the workflow.
- ✓ **Step 7:** Check the Response, Validate and display the greeting.

Program Name Steps from 1 to 7, Simple Greeting AI using CrewAI
greetings_demo.py

```
print("Step 1: Importing the libraries")

from crewai import Agent
from crewai import Task
from crewai import Crew
from crewai import LLM

print("Step 2: Configure the llm")

llm = LLM(
    model = "ollama/llama3.2",
    base_url = "http://localhost:11434"
)

print("Step 3: Create Agent")

simple_agent = Agent(
    role = "Friendly AI",
    goal = "Respond with a simple greeting.",
    backstory = "You are a helpful AI that loves saying hello.",
    verbose = True
)

print("Step 4: Define Tasks")

greeting_task = Task(
    description = "Say 'Hello, World!'",  

    agent = simple_agent,  

    expected_output = "A friendly greeting."
)
```

```
print("Step 5: Create a Crew")

crew = Crew(
    agents = [simple_agent],
    tasks = [greeting_task],
    verbose = True
)

print("Step 6: Kickoff the Crew")

response = crew.kickoff()

print("Step 7: Check the Response")

print(response)
```

Output

Step 1: Importing the libraries
Step 2: Configure the llm
Step 3: Create Agent
Step 4: Define Tasks
Step 5: Create a Crew
Step 6: Kickoff the Crew
Step 7: Check the Response

Hello, World!

15. Use Case 1: Automated Content Creation using CrewAI

- ✓ Uses CrewAI to generate structured content.
- ✓ Senior Technical Writer (Ollama LLaMA 3.2) writes an engaging article on AI Agents.
- ✓ Ensures accuracy, efficiency, and scalability.
- ✓ We are using a **single-agent** workflow.

15.1. AI Agents technical flow for Automated content creation using CrewAI

- ✓ **Step 1:** Import Libraries - Essential for using CrewAI and LLMs.
- ✓ **Step 2:** Configure the LLM - Define the language model before creating agents.
- ✓ **Step 3:** Create Agents, Define their role, goal, and backstory.
 - Agent 1: Senior Technical Writer
- ✓ **Step 4:** Define Tasks, Assign specific tasks to each agent.
- ✓ **Step 5:** Create a Crew, assign tasks.
- ✓ **Step 6:** Kickoff the Crew, Execute the workflow.
- ✓ **Step 7:** Check the Response, Validate and analyze the output.

Program Name Steps from 1 to 7, Automated Content Creation with CrewAI
senior_tech_writer.py

```
print("Step 1: Importing the libraries")
```

```
from crewai import Agent
from crewai import Task
from crewai import Crew
from crewai import LLM
```

```
print("Step 2: Configure the llm")
```

```
llm = LLM(
    model = "ollama/llama3.2",
    base_url = "http://localhost:11434"
)
```

```
print("Step 3: Create Agent")
```

```
senior_technical_writer = Agent(
    role = "Senior Technical Writer",
    goal = """Craft clear, engaging, and well-structured
technical content based on research findings""",
    backstory = """You are an experienced technical writer
with expertise in simplifying complex concepts,
structuring content for readability, and ensuring accuracy
in documentation""",
    llm = llm,
    verbose = True
)
```

```
print("Step 4: Create Task")

writing_task = Task(
    description = """Write a well-structured, engaging,
    and technically accurate article
    on {topic}.""",
    agent = senior_technical_writer,
    expected_output = """A polished, detailed, and easy-to-
    read article on the given topic."""
)

print("Step 5: Create a Crew ")

crew = Crew(
    agents = [senior_technical_writer],
    tasks = [writing_task],
    verbose = True
)

print("Step 6: Run the Crew")

response = crew.kickoff(inputs = {"topic" : "AI Agents"})

print("Step 7: Check the Response")

print(response)
```

Output

- Step 1:** Importing the libraries
- Step 2:** Configure the llm
- Step 3:** Create Agent
- Step 4:** Create Task
- Step 5:** Create a Crew
- Step 6:** Run the Crew
- Step 7:** Check the Response

AI Agents: The Future of Intelligent Systems

Artificial intelligence (AI) has revolutionized numerous industries, transforming the way we live and work. At the heart of this technological advancement are AI agents, intelligent systems that can perform tasks autonomously, making decisions based on complex algorithms and machine learning models. In this article, we will delve into the world of AI agents, exploring their capabilities, types, and potential applications.

Understanding AI Agents

AI agents are software-based entities that operate independently, executing tasks without human intervention. They use machine learning techniques to analyze data, identify patterns, and make predictions or decisions. AI agents can be classified into two primary categories: rule-based and machine learning-based systems.

Rule-Based Systems

Rule-based systems rely on pre-defined rules and decision-making processes to guide their actions. These systems are typically used in applications where a high degree of accuracy is required, such as in healthcare and finance. Rule-based AI

AI Agents Part - 1

agents can be programmed to recognize specific patterns or anomalies, allowing them to make predictions or recommendations.

Machine Learning-Based Systems

Machine learning-based systems, on the other hand, use machine learning algorithms to analyze data and improve their performance over time. These systems are more flexible than rule-based systems, as they can adapt to changing circumstances and learn from experience. Machine learning-based AI agents are commonly used in applications such as natural language processing and computer vision.

Types of AI Agents

AI agents can be categorized into several types based on their capabilities and functionality:

1. **Simple Reflex Agents**: These agents respond to a specific set of input actions, using pre-defined rules to guide their behavior.
2. **Model-Based Reflex Agents**: These agents use knowledge-based systems to reason about the world, making decisions based on a combination of rules and machine learning models.
3. **Utility Function Agents**: These agents use utility functions to evaluate the consequences of their actions, optimizing their performance based on a set of goals or objectives.

Applications of AI Agents

AI Agents Part - 1

AI agents have numerous applications across various industries, including:

1. **Virtual Assistants**: AI agents are used in virtual assistants such as Siri and Alexa to provide personalized recommendations and perform tasks.
2. **Robotics**: AI agents control robots, allowing them to navigate complex environments and interact with humans.
3. **Healthcare**: AI agents are used in medical diagnosis and treatment, analysing patient data and providing insights to healthcare professionals.
4. **Finance**: AI agents are used in trading and investment, analysing market trends and making predictions.

Advantages of AI Agents

AI agents offer several advantages over traditional systems, including:

1. **Increased Efficiency**: AI agents can automate tasks, reducing the need for human intervention and increasing productivity.
2. **Improved Accuracy**: AI agents use machine learning algorithms to analyze data, reducing errors and improving accuracy.
3. **Personalization**: AI agents can provide personalized recommendations and services, enhancing the user experience.

Challenges and Limitations of AI Agents

While AI agents offer numerous benefits, they also pose several challenges and limitations, including:

AI Agents Part - 1

1. ****Explainability**:** AI agents can be difficult to understand, making it challenging to explain their decision-making processes.
2. ****Bias**:** AI agents can inherit biases from the data used to train them, leading to unfair outcomes.
3. ****Security**:** AI agents can pose security risks if not designed with proper safeguards.

Conclusion

AI agents are intelligent systems that have revolutionized numerous industries, transforming the way we live and work. With their capabilities in machine learning, natural language processing, and computer vision, AI agents offer numerous advantages over traditional systems. However, they also pose several challenges and limitations, including explainability, bias, and security concerns. As AI continues to evolve, it is essential to address these challenges and ensure that AI agents are designed with proper safeguards to ensure their safe and responsible deployment.

Final Thoughts

The future of AI agents holds much promise, with potential applications in numerous industries. However, as we continue to develop and deploy AI systems, it is crucial to prioritize transparency, accountability, and fairness. By doing so, we can unlock the full potential of AI agents, creating a brighter future for humanity.

In conclusion, AI agents are intelligent systems that have transformed numerous industries. With their capabilities in machine learning, natural language processing, and computer vision, they offer numerous advantages over traditional

AI Agents Part - 1

systems. However, it is essential to address the challenges and limitations associated with AI agents, ensuring that they are designed with proper safeguards to ensure their safe and responsible deployment.

As we move forward, it is crucial to prioritize transparency, accountability, and fairness in the development and deployment of AI agents. By doing so, we can unlock the full potential of these systems, creating a brighter future for humanity.

Enhancing Performance with Stronger LLMs

- ✓ Weaker LLMs may produce less accurate responses.
- ✓ Upgrade to DeepSeek or GPT-4 for better accuracy & efficiency.
- ✓ Model switching steps were covered earlier.

16. Use Case 2: Automated Healthcare AI Summary with CrewAI

- ✓ Automates research and writing on AI in healthcare.
- ✓ **Multiple agents** gather insights and creates a structured summary.
- ✓ Ensures efficiency, accuracy, and time-saving.

16.1. AI Agents technical flow for healthcare content creation using CrewAI

- ✓ **Step 1:** Import Libraries, Essential for using CrewAI and LLMs.
- ✓ **Step 2:** Configure the LLM, Define the language model before creating agents.
- ✓ **Step 3:** Create Agents, Define their role, goal, and backstory.
 - **Agent 1:** Researcher, Gathers information.
 - **Agent 2:** Writer, Summarizes research findings.
- ✓ **Step 4:** Define Tasks, Assign specific tasks to each agent.
- ✓ **Step 5:** Create a Crew, group agents and assign tasks.
- ✓ **Step 6:** Kickoff the Crew, Execute the workflow.
- ✓ **Step 7:** Check the Response – Validate and analyze the output.

AI Agents Part - 1

Program Step from 1 to 7: Healthcare use case
Name health_care_demo.py

```
print("Step 1: Importing the libraries")

from crewai import Agent
from crewai import Task
from crewai import Crew
from crewai import LLM

print("Step 2: Configure the llm")

llm = LLM(
    model = "ollama/llama3.2",
    base_url = "http://localhost:11434"
)

print("Step 3.1: Create Researcher Agent")

researcher = Agent(
    role = "Researcher",
    goal = "Gather information about a given topic.",
    backstory = "A detail-oriented AI skilled in finding
    accurate information.",
    verbose = True,
    allow_delegation = False
)
```

AI Agents Part - 1

```
print("Step 3.2: Create Writer Agent")

writer = Agent(
    role = "Writer",
    goal = "Summarize information into a concise and
    readable format.",
    backstory = "A creative AI that enjoys writing clear and
    engaging summaries.",
    verbose = True,
    allow_delegation = False
)

print("Step 4.1: Create Research Task")

research_task = Task(
    description = "Find key details about the benefits of AI in
    healthcare.",
    agent = researcher,
    expected_output = "A list of key benefits of AI in
    healthcare."
)

print("Step 4.2: Create Writing Task")

writing_task = Task(
    description = "Summarize the research findings into a
    short article.",
    agent = writer,
    expected_output = "A well-structured summary of AI's
    benefits in healthcare."
)
```

```
print("Step 5: Create a Crew")

crew = Crew(
    agents = [researcher, writer],
    tasks = [research_task, writing_task],
    verbose = True
)

print("Step 6: Run the Crew")

response = crew.kickoff()

print("Step 7: Check the Response")

print(response)
```

Output

Step 1: Importing the libraries
Step 2: Configure the llm
Step 3.1: Create Researcher Agent
Step 3.2: Create Writer Agent
Step 4.1: Create Research Task
Step 4.2: Create Writing Task
Step 5: Create a Crew
Step 6: Run the Crew
Step 7: Check the Response

The Transformative Benefits of AI in Healthcare

Artificial Intelligence (AI) is revolutionizing the healthcare sector through a range of innovative applications that enhance diagnostic accuracy, personalize medicine, and streamline operations. Here are some key benefits of AI in healthcare that demonstrate its transformative potential:

1. **Enhanced Diagnostic Accuracy**: AI algorithms, especially those utilizing deep learning techniques, surpass traditional diagnostic methods by providing more accurate analyses of medical images, such as X-rays, MRIs, and CT scans. Research indicates that AI can often match or even outperform human radiologists in identifying diseases, including cancer.
2. **Personalized Medicine**: By analysing extensive patient data, AI helps in crafting personalized treatment plans. It predicts individual responses to various therapies based on genetic profiles, lifestyle factors, and disease characteristics, resulting in more tailored and effective healthcare.
3. **Predictive Analytics**: AI's ability to process large datasets enables it to recognize patterns and forecast patient outcomes. Machine learning models can predict disease outbreaks or assess which patients are at a higher risk of readmission, facilitating timely preventive care.
4. **Operational Efficiency**: AI enhances healthcare operations through automation of routine tasks such as patient scheduling, flow management, and billing processes. This efficiency not only streamlines healthcare delivery but also alleviates administrative workloads on healthcare staff.
5. **Drug Discovery and Development**: The integration of AI in drug discovery accelerates the identification of potential drug candidates and assesses their efficacy by analysing biological data. This significantly reduces the time and financial investment needed to bring new medications to market.
6. **Remote Monitoring and Telemedicine**: AI-powered tools allow for continuous health monitoring through wearable devices. These applications enable healthcare providers to track

AI Agents Part - 1

vital signs and address emerging health issues early on, particularly beneficial for managing chronic illnesses.

7. ****Enhanced Patient Engagement**:** Virtual assistants and chatbots driven by AI offer patients immediate access to information and support, fostering better communication and engagement with healthcare providers. This assists patients in managing conditions effectively and adhering to treatment plans.
8. ****Integration of Data from Multiple Sources**:** AI systems excel at synthesizing health data from diverse sources like electronic health records (EHRs), genetic data, and wearables. This comprehensive analysis equips healthcare professionals with a holistic view for informed decision-making.
9. ****Efficiency in Clinical Trials**:** AI optimizes various aspects of clinical trials, including design, patient recruitment, and monitoring, thereby expediting the overall process. It helps in identifying suitable candidates and predicting responses, which enhances the likelihood of trial success.
10. ****Supporting Healthcare Professionals**:** By helping with administrative tasks and processing complex medical data, AI supports healthcare providers in focusing more on their patients. It offers evidence-based recommendations that improve the quality of care.

In summary, the applications of AI in healthcare showcase its incredible ability to improve diagnostic processes, enhance patient outcomes, and create operational efficiencies. As AI continues to evolve, it holds the promise of transforming healthcare delivery into a more accurate, efficient, and patient-centered system.

17. Use Case 3: Multi-Agent Research Assistant with CrewAI

- ✓ Uses CrewAI to automate and enhance research tasks.
- ✓ **Multiple Agents**
 - Research Agent gathers real-time, relevant information.
 - Summarization Agent structures findings into a clear summary.
 - Fact-Checking Agent verifies accuracy for reliability.
 - Improves accuracy, efficiency, and scalability compared to a single-agent system

17.1. AI Agents technical flow for Multi-Agent Research Assistant with CrewAI

- ✓ **Step 1:** Import Libraries – Load CrewAI, dotenv, and SerperDevTool.
- ✓ **Step 2:** Load Environment & Enable Web Search – Use `load_dotenv()` and SerperDevTool for real-time search.
- ✓ **Step 3:** Create Agents – Define roles and goals.
 - **Step 3.1:** Research Agent – Gathers the latest information using Serper Dev Tool.
 - **Step 3.2:** Summarization Agent – Extracts and summarizes key insights.
 - **Step 3.3:** Fact-Checking Agent – Verifies and corrects inaccuracies.
- ✓ **Step 4:** Define Tasks – Assign tasks to agents.
 - **Step 4.1:** Research Task – Use SerperDevTool to gather insights.
 - **Step 4.2:** Summarization Task – Convert findings into a structured summary.
 - **Step 4.3:** Fact-Checking Task – Validate and correct the summary.
- ✓ **Step 5:** Create a Crew – Group agents, assign tasks, and set sequential execution.
- ✓ **Step 6:** Run the Crew – Execute the workflow with an input topic.
- ✓ **Step 7:** Check the Response – Display the fact-checked, verified summary.

Enabling Web Search

- ✓ Integrate Serper.dev for real-time web searches.
 - Get a free API key from Serper.dev.
 - Set the API key in the .env file for authentication.
- ✓ Next step: Define agents and their tasks!

```
OPENAI_API_KEY="sk-38381...."
```

```
SERPER_API_KEY="96f29..."
```

AI Agents Part - 1

Program Name Steps from 1 to 7, Multi-Agent Research Assistant with CrewAI
multi_agent_demo.py

```
print("Step 1: Importing the libraries")

from crewai import Agent, Task
from dotenv import load_dotenv
from crewai import Crew, Process

from crewai_tools import SerperDevTool

print("Step Special 1: Load Environment & Enable Web Search –
      Use load_dotenv() and SerperDevTool for real-time search.")

load_dotenv()

serper_dev_tool = SerperDevTool()

print("Step 2: Configure the llm: Using default LLM now")

print("Step 3.1: Research Agent ")

research_agent = Agent(
    role = "Internet Researcher",
    goal = "Find the most relevant and recent information
          about a given topic.",
    backstory = """You are a skilled researcher, adept at
                  navigating the internet and gathering high-quality,
                  reliable information.""",
    tools = [serper_dev_tool],
    verbose = True
)
```

AI Agents Part - 1

```
print("Step 3.2: Summarization Agent")

summarizer_agent = Agent(
    role = "Content Summarizer",
    goal = "Condense the key insights from research into a
short and informative summary.",
    backstory = """You are an expert in distilling complex
information into concise, easy-to-read summaries.""",
    verbose = True
)

print("Step 3.3: Fact-Checking Agent")

fact_checker_agent = Agent(
    role = "Fact-Checking Specialist",
    goal = "Verify the accuracy of information and remove
any misleading or false claims.",
    backstory = """You are an investigative journalist with a
knack for validating facts,
ensuring that only accurate information is published."""",
    tools = [serper_dev_tool],
    verbose = True
)
```

```
print("Step 4.1: Research Task")

research_task = Task(
    description = """Use the SerperDevTool to search for the
    most relevant and recent data about {topic}."""
    "Extract the key insights from multiple sources.",
    agent = research_agent,
    tools = [serper_dev_tool],
    expected_output = "A detailed research report with key
    insights and source references."
)

print("Step 4.2: Summarization Task")

summarization_task = Task(
    description = "Summarize the research report into a
    concise and informative paragraph. "
    "Ensure clarity, coherence, and completeness.",
    agent = summarizer_agent,
    expected_output = "A well-structured summary with the
    most important insights."
)

print("Step 4.3: Fact-Checking Task")

fact_checking_task = Task(
    description = "Verify the summarized information for
    accuracy using the SerperDevTool. "
    "Cross-check facts with reliable sources and correct any
    errors.",
    agent = fact_checker_agent,
    tools = [serper_dev_tool],
    expected_output = "A fact-checked, verified summary of
    the research topic."
)
```

```
print("Step 5: Create a Crew")

research_crew = Crew(
    agents = [research_agent, summarizer_agent,
              fact_checker_agent],
    tasks = [research_task, summarization_task,
              fact_checking_task],
    process = Process.sequential,
    verbose = True
)

print("Step 6: Run the Crew")

result = research_crew.kickoff(inputs={"topic": "The impact of AI
on job markets"})

print("Step 7: Check the Response")

print("\nFinal Verified Summary:\n", result)
```

Output

- Step 1:** Importing the libraries
- Step Special 1:** Load Environment & Enable Web Search – Use load_dotenv() and SerperDevTool for real-time search
- Step 2:** Configure the llm: Using default LLM now
- Step 3.1:** Research Agent
- Step 3.2:** Summarization Agent
- Step 3.3:** Fact-Checking Agent
- Step 4.1:** Research Task
- Step 4.2:** Summarization Task
- Step 4.3:** Fact-Checking Task
- Step 5:** Create a Crew
- Step 6:** Execute the Crew

Step 7: Check the Response

Final Verified Summary:

The impact of Artificial Intelligence (AI) on job markets is significant, reshaping job roles and enhancing productivity. With AI automating traditional positions, particularly in technical fields, new job categories are emerging, forecasted to create 20 to 50 million jobs globally. AI is expected to boost workplace efficiency by up to 40%. However, this transition will vary across industries, with notable increases in AI job listings and demand for AI-literate professionals. Policymakers are urged to implement proactive measures, as AI could affect 40% of jobs worldwide, necessitating a fair transition into an AI-driven economy. Education will play a crucial role in preparing the workforce, emphasizing specialized skills in technology and AI. Overall, while AI presents challenges, it also offers opportunities for economic growth and workforce development if adequately managed.

1. Data Science – Machine Learning – Introduction

Contents

1. Machine learning.....	2
2. Is machine learning hard?	2
3. Program	2
4. What is an algorithm?	2
5. What is machine learning algorithm?.....	3
6. Diff b/n machine learning algorithm and normal algorithm?	3
7. Why Machine learning?	3
8. Machine learning definitions	4
9. Artificial intelligence vs Machine Learning vs Deep Learning.....	5
9.1. Machine learning	6
9.2. Deep learning.....	7
9.3. Artificial intelligence	8
10. Is computer identifying pictures on image like cat/dog?	9
11. Is human identifying picture on image like cat/dog?	9
12. Human vs computer	10
13. How a machine can take the decisions?.....	11
14. Past couple of decades.....	11
15. Human's follows this formula.....	11
16. Generally.....	12
17. How do machines think?.....	13
18. Real time examples	14
19. Gmail application	14
20. Banking loan application / credit card	15
21. Where machine learning helps?	16
22. Few of machine learning applications.....	16
23. Why machine learning required?	16
24. Technically speaking.....	17

1. Data Science – Machine Learning – Introduction

1. Machine learning

- ✓ It is a technique which enables computers to learn automatically from past data.
- ✓ It is an approach to train the models.
- ✓ It is helpful to predict the future values.

2. Is machine learning hard?

- ✓ No, never
- ✓ Machine learning is very easy.
- ✓ It requires imagination, creativity, and a visual mind.
- ✓ Key point is, we should learn how to play with data and applying logic on data.
- ✓ This material helps how to do all these things.

3. Program

- ✓ A program is a group of instructions to perform the task; computer can execute these instructions to finish the task.
- ✓ This is very good approach for simple task

4. What is an algorithm?

- ✓ An algorithm is program which having group of instructions to perform a task.
 - Input + Logic = Output
 - Data + Program = Result

5. What is machine learning algorithm?

- ✓ Machine Learning algorithm is an application.
- ✓ It **learns** knowledge (patterns) automatically **from the data** without being explicitly programmed.
 - Data + Result = Program(Model)

6. Diff b/n machine learning algorithm and normal algorithm?

- ✓ Machine Learning algorithm learns knowledge (patterns) from the Data.
- ✓ Normal algorithm cannot learn anything on its own.

7. Why Machine learning?

- ✓ Currently every company is generating huge amount of the data.
- ✓ So, the volume of data is increasing on every day.
- ✓ Machine learning algorithms can extract information from data.
- ✓ The major use cases of Machine learning,
 - Creating models.
 - Getting deep insights & etc.
- ✓ Machine learning is going to be the center point of all fields.

8. Machine learning definitions



“ Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.

~ Tom Mitchell,
Machine Learning, McGraw Hill, 1997

Carnegie Mellon University
Machine Learning

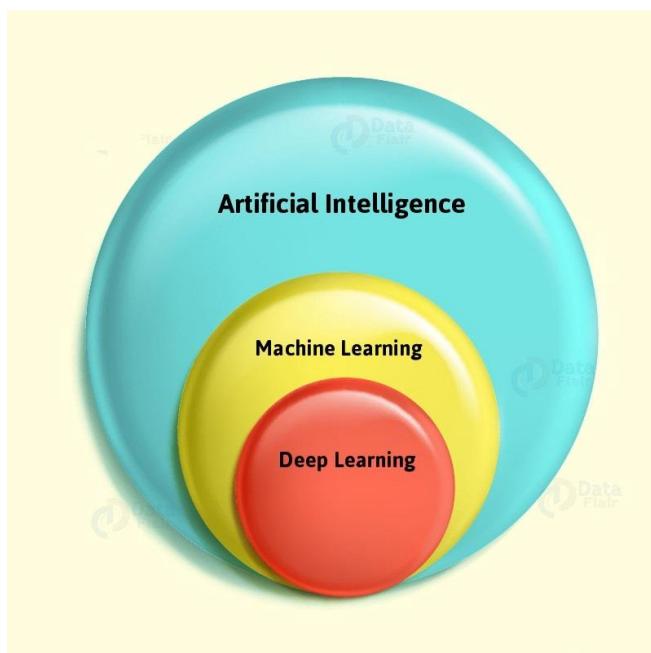
Arthur Samuel definition

- ✓ Machine learning term was introduced by Arthur Samuel in the year of 1959 and he defined it this way,
 - Machine learning is,
 - Enables a machine to learn automatically from data,
 - Improve performance from experiences,
 - Predict things without being explicitly programmed

Daniel's definition

- ✓ Teaching computers how to learn by using data and experience, rather than by instructions.

9. Artificial intelligence vs Machine Learning vs Deep Learning



9.1. Machine learning

- ✓ Machine learning is a part of artificial intelligence
- ✓ Machine Learning is a technique of understand the data, learn from data and then apply what they have learnt to take next decision.

Examples

- ✓ Amazon using machine learning to give better product choice recommendations to their costumers based on their preferences.
- ✓ Netflix uses machine learning to give better suggestions to their users of the TV series or movie or shows that they would like to watch & many more

9.2. Deep learning

- ✓ Deep learning is actually a subset of machine learning.
- ✓ The main difference between deep and machine learning is, machine learning models works better but the model still needs some guidance.
- ✓ If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly.
- ✓ In the case of deep learning, the model does it by itself.

Example

- ✓ Automatic car driving system is a good example of deep learning.

9.3. Artificial intelligence

- ✓ AI is an ability of computer program to function like a human brain.
- ✓ Machine learning and deep learning are the subsets of AI
- ✓ The MOTO of AI is to replicate a human brain, the way a human brain thinks, works and functions.
- ✓ Currently AI is not yet fully implemented but we are very close to establish that too.

Example

- ✓ Sophia, the most advanced AI model present today.

10. Is computer identifying pictures on image like cat/dog?

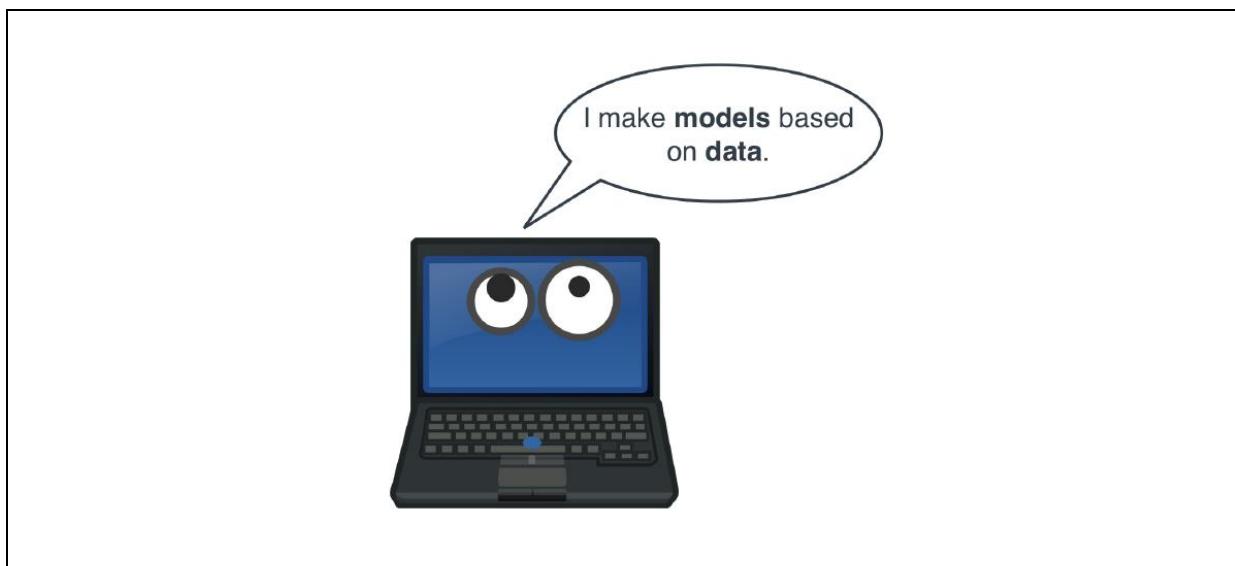
- ✓ Yes and it is really amazing task.
- ✓ If we giving lot of images to computer then computer can try to learn about the attributes to recognize images
- ✓ This is called as machine learning.
 - It is something like teaching to computer how to think like a human

11. Is human identifying picture on image like cat/dog?

- ✓ Absolutely YES right.
- ✓ During my childhood parents/teachers taught like how to recognize a cat/dog/tiger/lion/etc
- ✓ So, in real life, if we see cat/dog/tiger/lion (sorry tiger/lion I have seen in Zoo but not directly) then we can easy to recognize without much effort.
- ✓ So, a human can take the decision based on experience.

12. Human vs computer

- ✓ Humans take the **decisions** based on the **experience**
- ✓ Computer can create **models** by using data



13. How a machine can take the decisions?

- ✓ First we need to understand human decision making process before understanding about how a computer takes the decision.

14. Past couple of decades

- ✓ From past two decades most of the companies are digitalized.

15. Human's follows this formula...

- ✓ All humans used to follow some patterns or formulas to take the next best action.

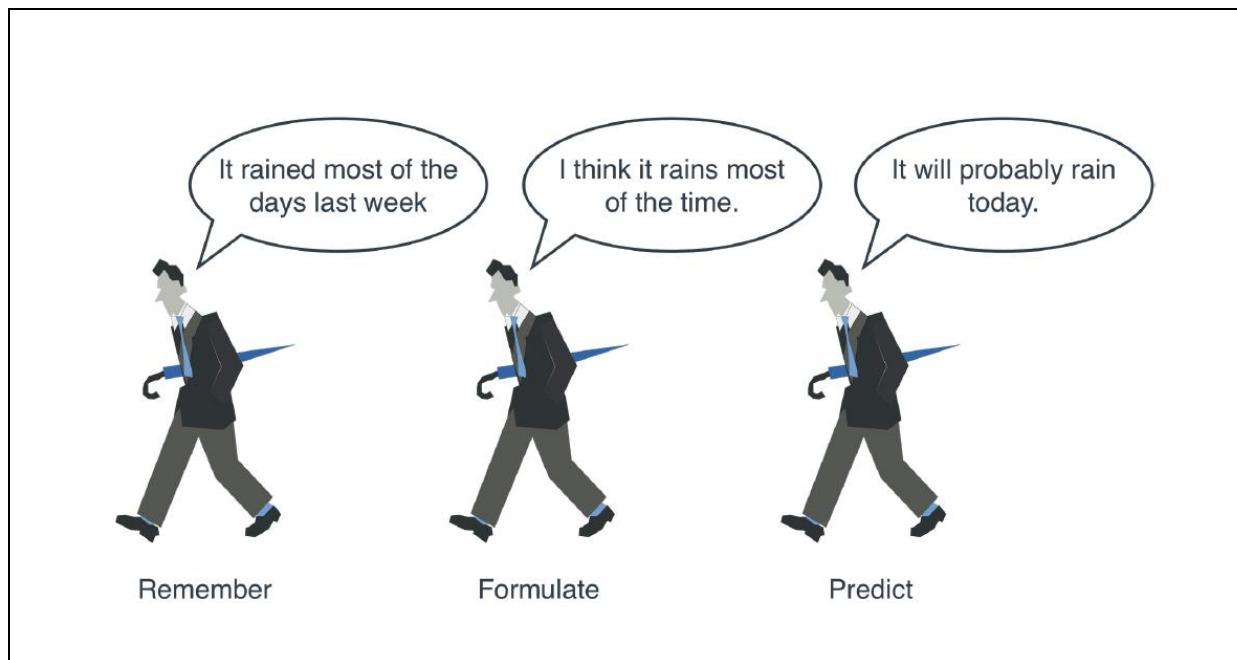
- Remember
- Formulate
- Predict

16. Generally

- ✓ We **remember** past situations that were similar.
- ✓ We **formulate** a general rule.
- ✓ We use this rule to **predict** what will happen if we take a certain decision.

Example

- ✓ For supposed to answer for a question like, "**Will it rain today? Or not**", then we will follow one **process** to answer this question right.
- ✓ We may be right or wrong, but at least, we are trying to make an accurate prediction.



17. How do machines think?

- ✓ The goal is to make the computer think the way we think, namely, how we use the remember formulate-predict framework.
- ✓ Here are the points what the computer can understand the points

Steps

- ✓ Remember
 - Look at the given **HUGE** data.
- ✓ Formulate
 - Go through rules and formulas, and check which one fits for the data
- ✓ Predict
 - Use the rule to make predictions about future data.

18. Real time examples

- ✓ Gmail application
- ✓ Banking loan application or credit card eligibility & etc

19. Gmail application

- ✓ *Gmail* is the very well-known example.
- ✓ Hope we are all known about Spam folder too
- ✓ **Spam**
 - It is the common term used for junk or unwanted email, such as chain letters, promotions, and so on
- ✓ **Ham**
 - It is the common term used for non-spam mail means useful mail
- ✓ In this scenario machine learning algorithm is working to filter spam or not
- ✓ Machine learning algorithm,
 - Understand the previous mails(data)
 - Based on the previous mail creates some models.
 - These models applies on upcoming mails and predict whether it is spam or not

20. Banking loan application / credit card

- ✓ Banking loan application, this is the very common application hope everyone knows
- ✓ Once we applied for loan by submitting required documents(data) like pay slip, past 6 months banking statement and etc
- ✓ Then banking guys will run one application over the given data.
- ✓ This is Machine learning algorithm
- ✓ This algorithm passes through the data and predicts the weather the loan/credit card sanctioned or not.

Note

- ✓ Many other applications are using machine learning
- ✓ I would like to say, Machine learning is everywhere 😊

21. Where machine learning helps?

- ✓ From past two decades most of the companies are digitalized.
- ✓ So, here data is generated more and more.
- ✓ So, in this case machine learning helps us to develop predictive models and automate several things.
- ✓ Assuming that we had past couple of year's **amazon** transactions data.
- ✓ Some questions:
 - Wanted to know how the business was in last couple of years
 - Wanted to take best decisions to improve the business and etc
- ✓ To answer above questions, we need to,
 - Gather the data
 - Process the data
 - Take the decisions.

22. Few of machine learning applications

- ✓ Text processing
- ✓ Speech Recognition
- ✓ Traffic prediction
- ✓ Product recommendations
- ✓ Self-driving cars
- ✓ Email spam and malware filtering
- ✓ Online fraud detection
- ✓ Weather forecasting and prediction & etc

23. Why machine learning required?

- ✓ As a human we cannot access huge amount of data manually to process.
- ✓ We can train machine learning algorithms by providing huge amount of data.
- ✓ Machine learning algorithm travel through this data in order to learn about data, it create the model and predict results automatically for new data

24. Technically speaking...

- ✓ Technically speaking,
 - WITH THE help of historical data
 - MACHINE LEARNING algorithms
 - BUILD a mathematical MODEL
 - that helps in making PREDICTIONS
 - Without being EXPLICITELY PROGRAMMER

1. Data Science – Machine Learning – Introduction

Contents

1. Machine learning.....	2
2. Is machine learning hard?	2
3. Program	2
4. What is an algorithm?	2
5. What is machine learning algorithm?.....	3
6. Diff b/n machine learning algorithm and normal algorithm?	3
7. Why Machine learning?	3
8. Machine learning definitions	4
9. Artificial intelligence vs Machine Learning vs Deep Learning.....	5
9.1. Machine learning	6
9.2. Deep learning.....	7
9.3. Artificial intelligence	8
10. Is computer identifying pictures on image like cat/dog?	9
11. Is human identifying picture on image like cat/dog?	9
12. Human vs computer	10
13. How a machine can take the decisions?.....	11
14. Past couple of decades.....	11
15. Human's follows this formula.....	11
16. Generally.....	12
17. How do machines think?.....	13
18. Real time examples	14
19. Gmail application	14
20. Banking loan application / credit card	15
21. Where machine learning helps?	16
22. Few of machine learning applications.....	16
23. Why machine learning required?	16
24. Technically speaking.....	17

1. Data Science – Machine Learning – Introduction

1. Machine learning

- ✓ It is a technique which enables computers to learn automatically from past data.
- ✓ It is an approach to train the models.
- ✓ It is helpful to predict the future values.

2. Is machine learning hard?

- ✓ No, never
- ✓ Machine learning is very easy.
- ✓ It requires imagination, creativity, and a visual mind.
- ✓ Key point is, we should learn how to play with data and applying logic on data.
- ✓ This material helps how to do all these things.

3. Program

- ✓ A program is a group of instructions to perform the task; computer can execute these instructions to finish the task.
- ✓ This is very good approach for simple task

4. What is an algorithm?

- ✓ An algorithm is program which having group of instructions to perform a task.
 - Input + Logic = Output
 - Data + Program = Result

5. What is machine learning algorithm?

- ✓ Machine Learning algorithm is an application.
- ✓ It **learns** knowledge (patterns) automatically **from the data** without being explicitly programmed.
 - Data + Result = Program(Model)

6. Diff b/n machine learning algorithm and normal algorithm?

- ✓ Machine Learning algorithm learns knowledge (patterns) from the Data.
- ✓ Normal algorithm cannot learn anything on its own.

7. Why Machine learning?

- ✓ Currently every company is generating huge amount of the data.
- ✓ So, the volume of data is increasing on every day.
- ✓ Machine learning algorithms can extract information from data.
- ✓ The major use cases of Machine learning,
 - Creating models.
 - Getting deep insights & etc.
- ✓ Machine learning is going to be the center point of all fields.

8. Machine learning definitions



“ Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.

~ Tom Mitchell,
Machine Learning, McGraw Hill, 1997

Carnegie Mellon University
Machine Learning

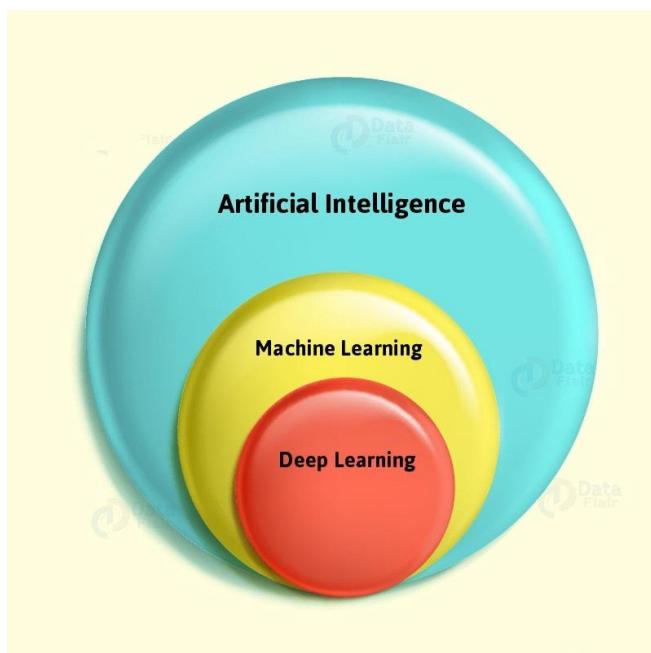
Arthur Samuel definition

- ✓ Machine learning term was introduced by Arthur Samuel in the year of 1959 and he defined it this way,
 - Machine learning is,
 - Enables a machine to learn automatically from data,
 - Improve performance from experiences,
 - Predict things without being explicitly programmed

Daniel's definition

- ✓ Teaching computers how to learn by using data and experience, rather than by instructions.

9. Artificial intelligence vs Machine Learning vs Deep Learning



9.1. Machine learning

- ✓ Machine learning is a part of artificial intelligence
- ✓ Machine Learning is a technique of understand the data, learn from data and then apply what they have learnt to take next decision.

Examples

- ✓ Amazon using machine learning to give better product choice recommendations to their costumers based on their preferences.
- ✓ Netflix uses machine learning to give better suggestions to their users of the TV series or movie or shows that they would like to watch & many more

9.2. Deep learning

- ✓ Deep learning is actually a subset of machine learning.
- ✓ The main difference between deep and machine learning is, machine learning models works better but the model still needs some guidance.
- ✓ If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly.
- ✓ In the case of deep learning, the model does it by itself.

Example

- ✓ Automatic car driving system is a good example of deep learning.

9.3. Artificial intelligence

- ✓ AI is an ability of computer program to function like a human brain.
- ✓ Machine learning and deep learning are the subsets of AI
- ✓ The MOTO of AI is to replicate a human brain, the way a human brain thinks, works and functions.
- ✓ Currently AI is not yet fully implemented but we are very close to establish that too.

Example

- ✓ Sophia, the most advanced AI model present today.

10. Is computer identifying pictures on image like cat/dog?

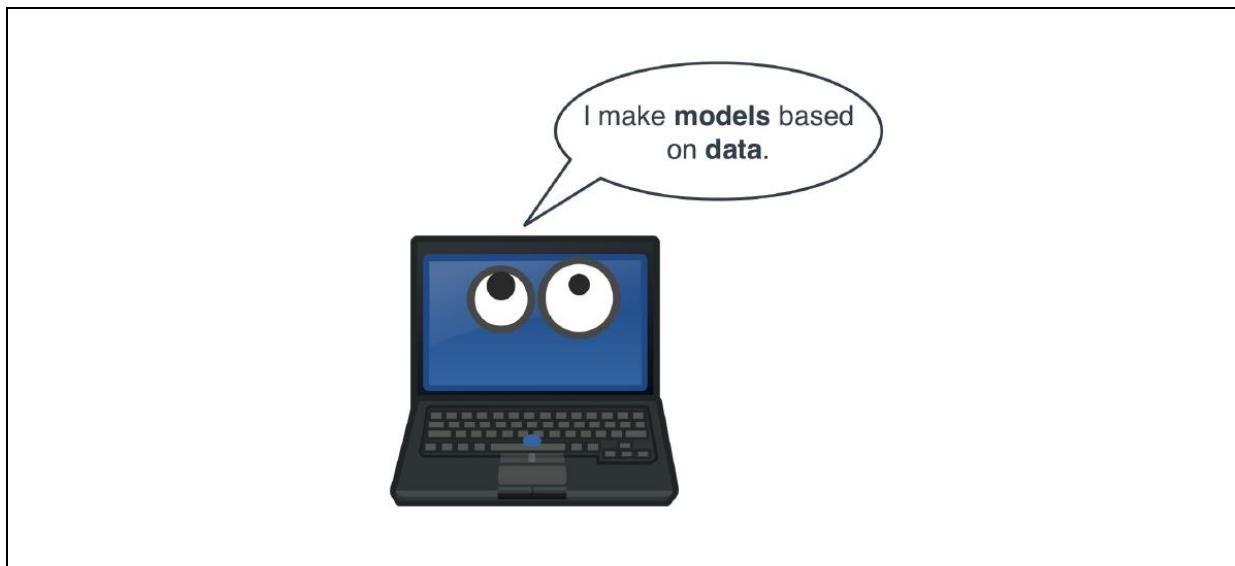
- ✓ Yes and it is really amazing task.
- ✓ If we giving lot of images to computer then computer can try to learn about the attributes to recognize images
- ✓ This is called as machine learning.
 - It is something like teaching to computer how to think like a human

11. Is human identifying picture on image like cat/dog?

- ✓ Absolutely YES right.
- ✓ During my childhood parents/teachers taught like how to recognize a cat/dog/tiger/lion/etc
- ✓ So, in real life, if we see cat/dog/tiger/lion (sorry tiger/lion I have seen in Zoo but not directly) then we can easy to recognize without much effort.
- ✓ So, a human can take the decision based on experience.

12. Human vs computer

- ✓ Humans take the **decisions** based on the **experience**
- ✓ Computer can create **models** by using data



13. How a machine can take the decisions?

- ✓ First we need to understand human decision making process before understanding about how a computer takes the decision.

14. Past couple of decades

- ✓ From past two decades most of the companies are digitalized.

15. Human's follows this formula...

- ✓ All humans used to follow some patterns or formulas to take the next best action.

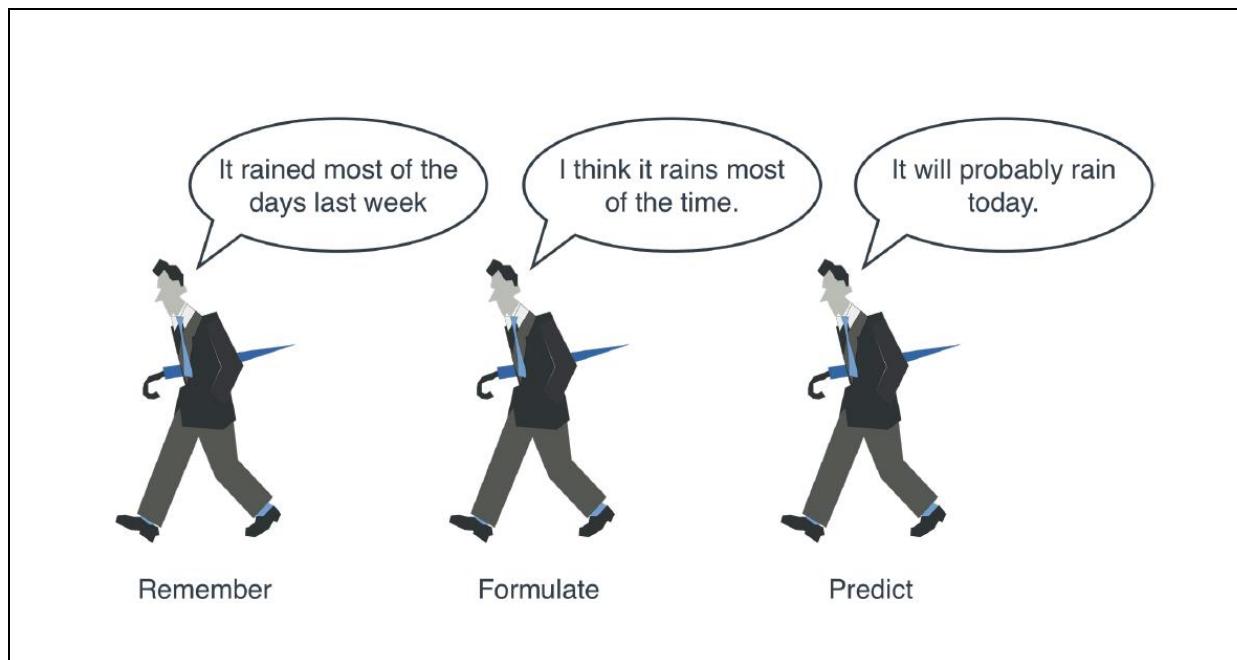
- Remember
- Formulate
- Predict

16. Generally

- ✓ We **remember** past situations that were similar.
- ✓ We **formulate** a general rule.
- ✓ We use this rule to **predict** what will happen if we take a certain decision.

Example

- ✓ For supposed to answer for a question like, "**Will it rain today? Or not**", then we will follow one **process** to answer this question right.
- ✓ We may be right or wrong, but at least, we are trying to make an accurate prediction.



17. How do machines think?

- ✓ The goal is to make the computer think the way we think, namely, how we use the remember formulate-predict framework.
- ✓ Here are the points what the computer can understand the points

Steps

- ✓ Remember
 - Look at the given **HUGE** data.
- ✓ Formulate
 - Go through rules and formulas, and check which one fits for the data
- ✓ Predict
 - Use the rule to make predictions about future data.

18. Real time examples

- ✓ Gmail application
- ✓ Banking loan application or credit card eligibility & etc

19. Gmail application

- ✓ *Gmail* is the very well-known example.
- ✓ Hope we are all known about Spam folder too
- ✓ **Spam**
 - It is the common term used for junk or unwanted email, such as chain letters, promotions, and so on
- ✓ **Ham**
 - It is the common term used for non-spam mail means useful mail
- ✓ In this scenario machine learning algorithm is working to filter spam or not
- ✓ Machine learning algorithm,
 - Understand the previous mails(data)
 - Based on the previous mail creates some models.
 - These models applies on upcoming mails and predict whether it is spam or not

20. Banking loan application / credit card

- ✓ Banking loan application, this is the very common application hope everyone knows
- ✓ Once we applied for loan by submitting required documents(data) like pay slip, past 6 months banking statement and etc
- ✓ Then banking guys will run one application over the given data.
- ✓ This is Machine learning algorithm
- ✓ This algorithm passes through the data and predicts the weather the loan/credit card sanctioned or not.

Note

- ✓ Many other applications are using machine learning
- ✓ I would like to say, Machine learning is everywhere 😊

21. Where machine learning helps?

- ✓ From past two decades most of the companies are digitalized.
- ✓ So, here data is generated more and more.
- ✓ So, in this case machine learning helps us to develop predictive models and automate several things.
- ✓ Assuming that we had past couple of year's **amazon** transactions data.
- ✓ Some questions:
 - Wanted to know how the business was in last couple of years
 - Wanted to take best decisions to improve the business and etc
- ✓ To answer above questions, we need to,
 - Gather the data
 - Process the data
 - Take the decisions.

22. Few of machine learning applications

- ✓ Text processing
- ✓ Speech Recognition
- ✓ Traffic prediction
- ✓ Product recommendations
- ✓ Self-driving cars
- ✓ Email spam and malware filtering
- ✓ Online fraud detection
- ✓ Weather forecasting and prediction & etc

23. Why machine learning required?

- ✓ As a human we cannot access huge amount of data manually to process.
- ✓ We can train machine learning algorithms by providing huge amount of data.
- ✓ Machine learning algorithm travel through this data in order to learn about data, it create the model and predict results automatically for new data

24. Technically speaking...

- ✓ Technically speaking,
 - WITH THE help of historical data
 - MACHINE LEARNING algorithms
 - BUILD a mathematical MODEL
 - that helps in making PREDICTIONS
 - Without being EXPLICITELY PROGRAMMER

2. Data Science – Machine Learning – Terminology

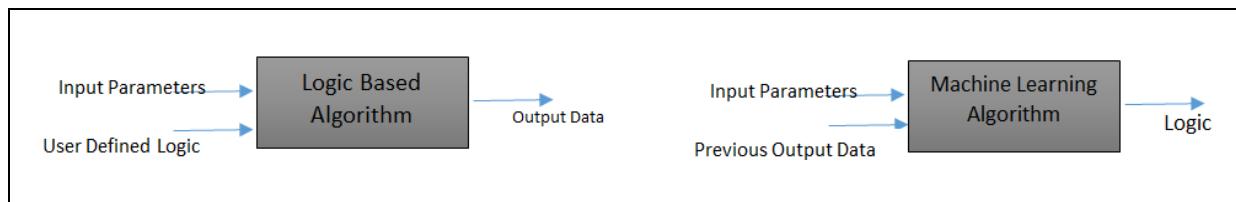
Contents

1. Machine learning.....	2
2. Model.....	2
3. Train	3
4. How to train?.....	3
5. Model learns.....	3
6. Pattern	3
7. Before trained	3
8. After trained	3
9. Test the model.....	3
10. Model deployment.....	4
11. Towards target value or classifier.....	4
12. General example	4
13. Smart application	4
14. Normal application.....	5
15. Smart application	6
16. Use case	6
17. Accuracy testing	7

2. Data Science – Machine Learning – Terminology

1. Machine learning

- ✓ Machine learning is a technique which enables computers to learn automatically from past data.
- ✓ Machine Learning is an approach to train the **models**.
- ✓ Machine learning is helpful to predict the future values.



2. Model

- ✓ In simple terms, a model is a representation of reality.
- ✓ A model is created to do a specific purpose

Example

- ✓ General example, a world map is a model for this physical world or universe.
 - The creator of world map have done deep analysis, understands, visualized with simple representation.

In terms of Machine learning,

- ✓ A model can be a,
 - A piece of code or program
 - A mathematical formula
 - A program + mathematical formula

3. Train

- ✓ We need to **train** the model

4. How to train?

- ✓ We need to use **data** to train the model

5. Model learns

- ✓ Model will learn **pattern** from data.

6. Pattern

- ✓ Patterns means a simple knowledge gained during training.
- ✓ So, model learnt patterns from data during training

7. Before trained

- ✓ Before training model, model doesn't know about the patterns.

8. After trained

- ✓ After training model, model knows about the patterns.

9. Test the model

- ✓ Once after model got trained then that model has to be tested to **check the accuracy**.
- ✓ If accuracy is good than the model works well

10. Model deployment

- ✓ Finally we need to deploy knowledge acquired by the model
- ✓ These models also called as model fit

11. Towards target value or classifier

- ✓ Models learnt patterns from data towards target value.
- ✓ So, let's try to understand what it means.

12. General example

- ✓ Assuming that one of my friend is introducing his friend as
 - His name is Prasad working in X Company and having 3 years of experience.
- ✓ When I heard about his experience I can guess/predict his salary (target value).
- ✓ So, our brain is already trained to guess some attributes from past data(old friends)
- ✓ So, we have patterns towards to target value

13. Smart application

- ✓ Once accuracy satisfied then we need to deploy the model.
- ✓ Once model deployed then that application will become as smart application

14. Normal application

- ✓ Assuming that we have created an application by using Java/dotnet/Python.
- ✓ It's just called as normal software application because this application is just works on instruction based.
- ✓ Instruct based means,
 - If x value is greater than y value, then prints x value
 - If x value is less than y value, then prints y value

Make a note

- ✓ Normal application cannot think like human to take decisions
- ✓ Normal applications cannot take own decisions

15. Smart application

- ✓ A smart application is an application which thinks like a human to take the decisions.
- ✓ Smart application is intelligence based.

16. Use case

- ✓ Whenever you send email automatically Gmail application is predicting whether it is a spam or not spam.
- ✓ That means Gmail application is well trained and captures the patterns about mail type like,
 - Spam mail
 - Promotion mail
 - Social media mail & etc
- ✓ So, Gmail application knows patterns about to recognize the type of mail.

17. Accuracy testing

- ✓ Before deploying the model, it needs to be tested in lot of dimensions.
- ✓ If model is getting good predictions then we need to deploy.

Process in machine learning

- ✓ We need to **train** the model.
- ✓ During training model acquires the **knowledge**.
- ✓ This knowledge is called as **model fit**
- ✓ Once after model fit produced we need to **test** to check the **accuracy**.
- ✓ If satisfied with accuracy then we need to **deploy** the model

Data is important

- ✓ We need to train the model with data
- ✓ During this training model will go through the data and understand the patterns about data.
- ✓ These patterns helps to predict the result on **new data**

3. Data Science – Machine Learning – Data & ML Algorithm

Contents

1. Data	2
2. Data in table	3
3. Row	3
4. Column	3
5. Cell.....	3
6. Statistical Learning Perspective	4
6.1. Example.....	4
7. Why Machine learning algorithms learn this function?	6
8. Terminology in statistics,	7
9. Computer Science Perspective	8
10. Models and Algorithms	9

3. Data Science – Machine Learning – Data & ML Algorithm

1. Data

- ✓ It is a collection of facts.
- ✓ Facts mean,
 - Alphabets
 - Numbers
 - Alphanumeric
 - Symbol

◆	A	B	C	D
1		Column 1	Column 2	Column 3
2	Row 1	2.2	2.3	1
3	Row 2	2.3	2.6	0
4	Row 3	2.1	2	1
5				

2. Data in table

- ✓ Generally, tables having data like rows, columns and cells
- ✓ Tabular data is very easy to understand.

3. Row

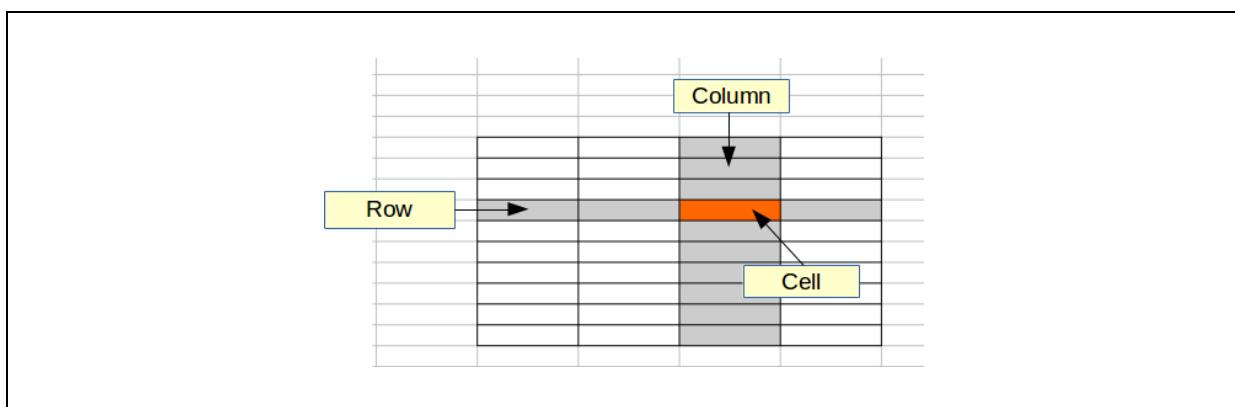
- ✓ A row describes a single entity or observation.
- ✓ A row is also called a record.

4. Column

- ✓ A column is a vertical group of cells within a table.
- ✓ A column having same type of values.
- ✓ In column we can store the data like weights, heights and prices etc info.

5. Cell

- ✓ A cell is a single value in a row and column.
- ✓ It may be a real value (1.5) an integer (2) or a category (red).



6. Statistical Learning Perspective

- ✓ In statistical learning perspective data means,
 - It is the context of a **hypothetical function** (f) that the machine learning algorithm is **trying to learn**.

6.1. Example

- ✓ Assuming that, there is a farmer in village and initially he had 1 acre land.
- ✓ While forming rice it get yields as below,

Area	Rice packets
1	10
2	20
3	30

If former having 4 acres land then how many rice packets it yields?

Answer

- ✓ A non-technical guy also can answer as, we will get **40** packets for **4** acres of land.

Area	Rice packets
1	10
2	20
3	30
4	?

Simple calculation

- ✓ Formula is,

○ 1 acre	=>	1×10	=	10
○ 2 acre	=>	2×10	=	20
○ 3 acre	=>	3×10	=	30
○ 4 acre	=>	4×10	=	40

- ✓ Simple Formula is,
 - Rice_yield = land_size x 10

7. Why Machine learning algorithms learn this function?

- ✓ To predict output for the given input.

Output = f(Input)

- ✓ Columns are called as input variables.
- ✓ Predicted result is called as output variable or response variable.

Output Variable = f(Input Variables)

◆	A	B	C
1	X1	X2	Y
2	2.2	2.3	1
3	2.3	2.6	0
4	2.1	2	1
5			

- ✓ Input data may have more than one input variable.
- ✓ The group of input variables are called as input vector.

Output Variable = f(Input Vector)

8. Terminology in statistics,

- ✓ Input variables are called as Independent variables.
- ✓ Output variable are called as the Dependent variable.
- ✓ Here the output is dependent on a function of input.

Dependent Variable = $f(\text{Independent Variables})$

- ✓ Input variable representing with X
- ✓ Output variable representing with y

$Y = f(X)$

- ✓ If we have multiple input variables then it's represent as input vector x_1, x_2, x_3 for the data

9. Computer Science Perspective

- ✓ A row means it's an entity/observation-instance/object in a table.
- ✓ A Column is called as an attribute.
- ✓ During modeling and predictions,
 - Input is called as input attribute.
 - Output is called as output attributes.

Output Attribute = Program(Input Attributes)

◆	A	B	C	D
1		Attribute 1	Attribute 2	Output Attribute
2	Instance 1	2.2	2.3	1
3	Instance 2	2.3	2.6	0
4	Instance 3	2.1	2	1
5				

Output = Program(Input Features)

Prediction output = Program(Instance)

10. Models and Algorithms

- ✓ A model is a specific representation learned **from data** and the algorithm as the process of learning it.

Model = Algorithm(Data)

4. Data Science – Machine Learning – Learning Function

Contents

1. Learning a Function	2
2. Purpose of learning-function	2
3. Machine learning algorithms	3

4. Data Science – Machine Learning – Learning Function

1. Learning a Function

- ✓ Machine learning algorithms are described as,
 - Learning a target function (f) which maps input variables (X) to output variable (Y)

$$\text{Output} = f(\text{Input})$$

$$Y = f(X)$$

- ✓ This function also having some error.
- ✓ This error is independent to the input data

$$Y = f(X) + \text{error}$$

2. Purpose of learning-function

- ✓ The purpose of this learning-function is to make predictions.
- ✓ The function which is having less error, that function will make the accurate predictions.
- ✓ This is called predictive modeling.

3. Machine learning algorithms

- ✓ Machine learning algorithms are techniques for estimating the target function (f).
- ✓ This function helps to predict the **output variable (Y)** for the given **input variables (X)**.
- ✓ Different machine learning algorithms having different assumptions about the form of the function, such as whether it is linear or nonlinear.

5. Data Science – Machine Learning – Types of the Models

Contents

1. Feature and label.....	2
1.1. Features	2
1.2. Label	3
2. Label Example	3
3. Labelled and unlabeled data.....	4
3.1. Labelled Data:	4
3.2. Unlabeled Data	5
4. Supervised learning definition.....	6
5. Unsupervised learning definition	7
6. Supervised learning example.....	8
7. Remember – Formulate - Predict.	9
8. Types of Supervised learning models.....	11
8.1. Regression models	12
8.2. Classification models.....	12
9. Unsupervised learning example	13
10. Types of unsupervised learning models.....	14
10.1. Clustering	15
10.2. Dimensionality reduction.....	16

5. Data Science – Machine Learning – Types of the Models

- ✓ In Machine learning there are mainly 3 types of models exists,

- ✓ Supervised learning
- ✓ Unsupervised learning
- ✓ Reinforcement learning

Best tip

- ✓ Before understanding the types of algorithms let's try to understand terminology

1. Feature and label

1.1. Features

- ✓ Features are simply the columns of the table
- ✓ These features describes the about the data
- ✓ In the given data, Age, Gender, Experience and Salary are features

Age	Gender	Experience	Salary
20	M	4	40000
24	F	5	50000

1.2. Label

- ✓ The output we will get after training the model is called as a **Label**

- ✓ Requirement
 - Suppose I wanted to predict the salary who had 6 years of experience
 - We prepared a model and that model predicted the salary.
 - Here salary is called as a **label**

- ✓ Simple
 - We are trying to predict a feature based on the others, that feature is the **label**.

2. Label Example

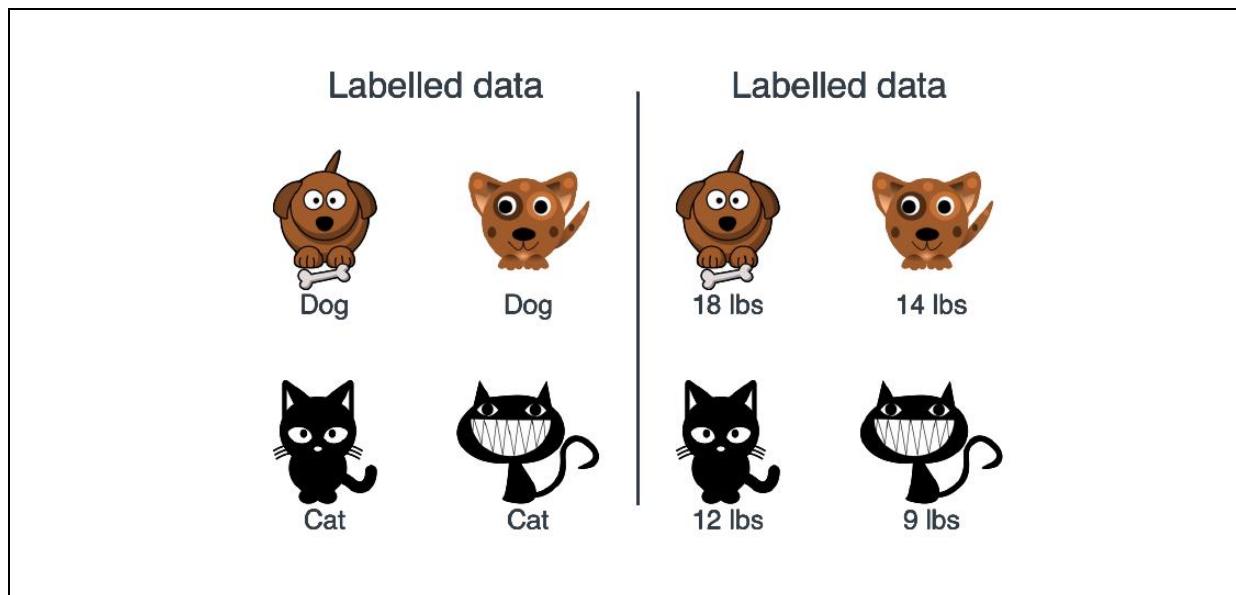
- ✓ If we are trying to predict the **type of pet** for example cat or dog based on information then that is the **label**.
- ✓ If we are trying to predict if the pet is **sick or healthy** based on symptoms and other information, then that is the **label**.
- ✓ If we are trying to predict the **age of the pet**, then the age is the **label**.

3. Labelled and unlabeled data

- ✓ According to the label, data is divided into two types
 - Labelled data
 - Unlabeled data

3.1. Labelled Data:

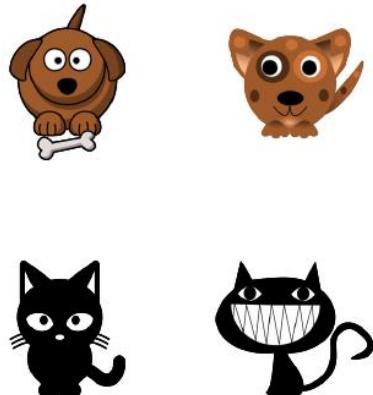
- ✓ Labelled data comes with a tag or label, like a name, a type, or a number.



3.2. Unlabeled Data

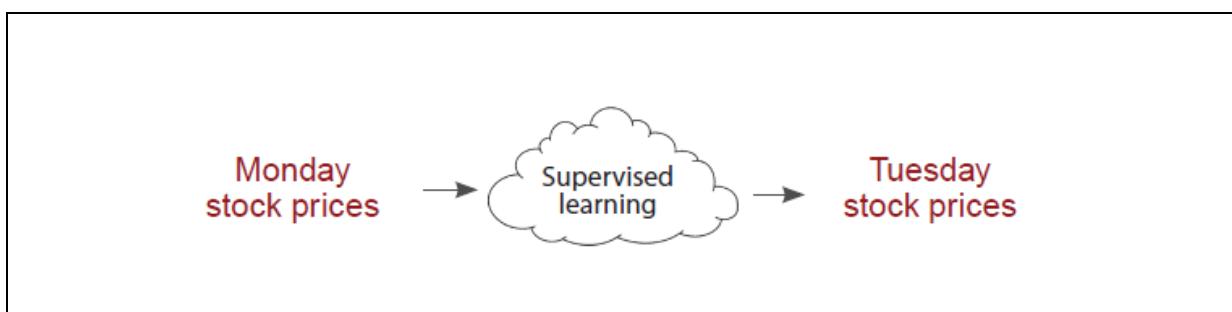
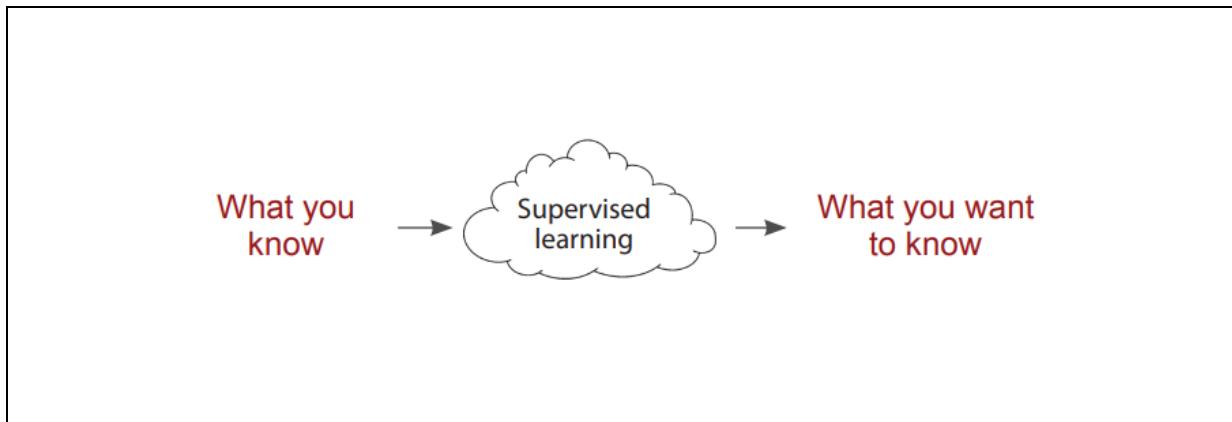
- ✓ Unlabeled data have no tag or label

Unlabelled data



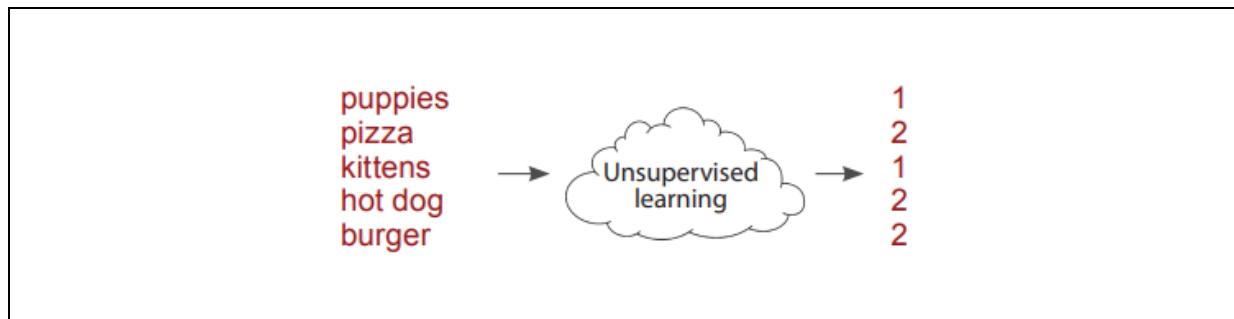
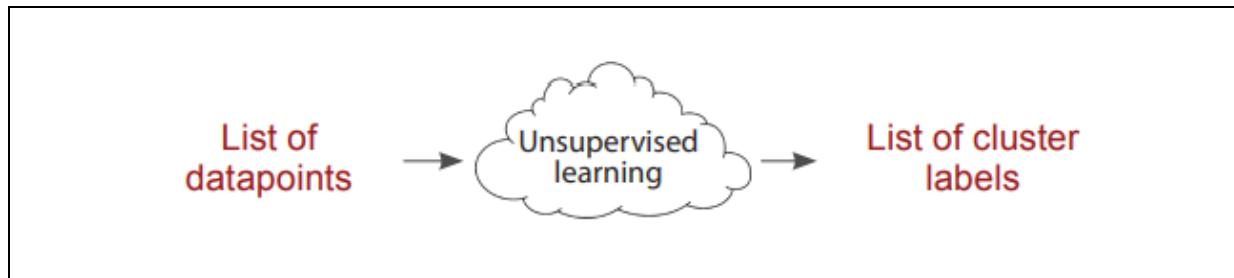
4. Supervised learning definition

- ✓ In supervised learning, we will train the models by using input **features** and **labels**



5. Unsupervised learning definition

- ✓ In unsupervised learning, we will train the models by using only input features and there is not labels



Unsupervised machine learning
Unsupervised learning groups your data.

6. Supervised learning example

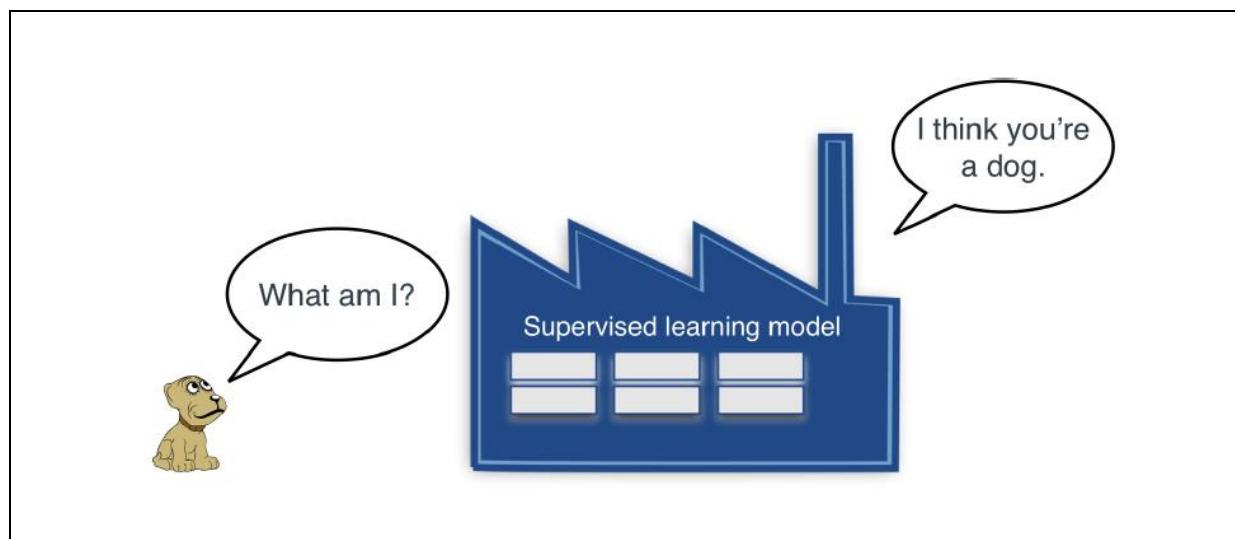
- ✓ Now a days it is very commonly using for all applications
- ✓ Supervised learning = **features + label**.

Examples

- ✓ Image recognition,
- ✓ Text processing,
- ✓ Recommendation systems & many more.

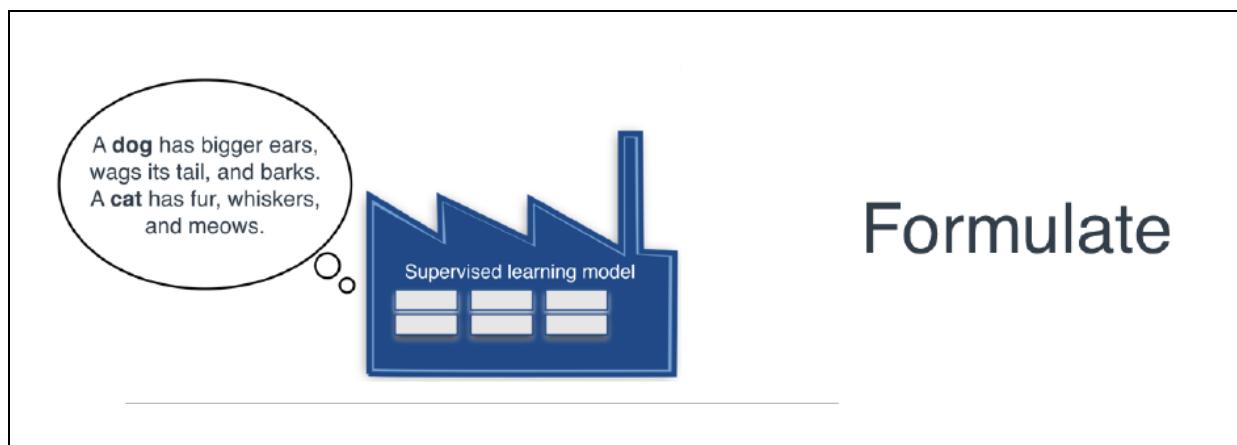
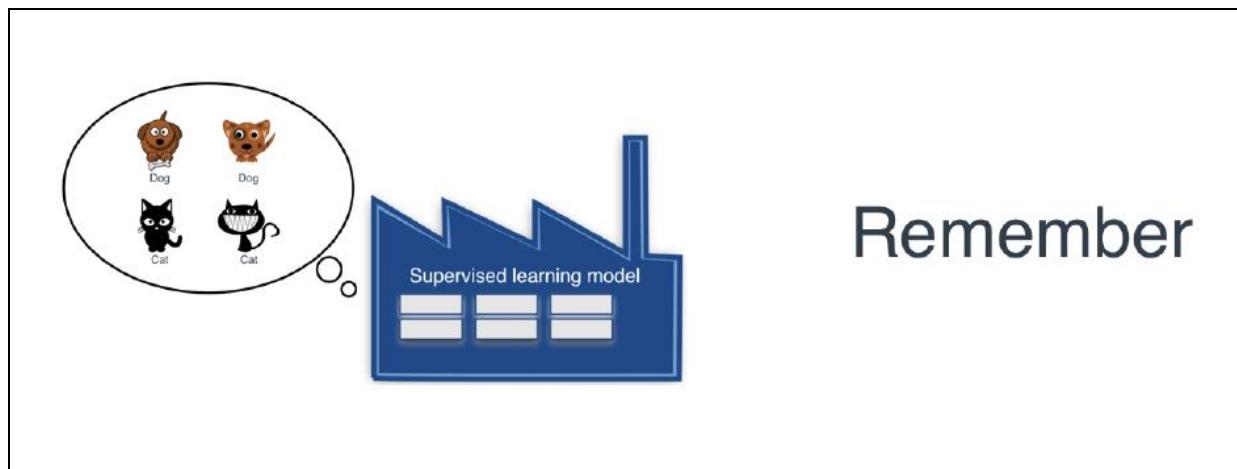
Scenario

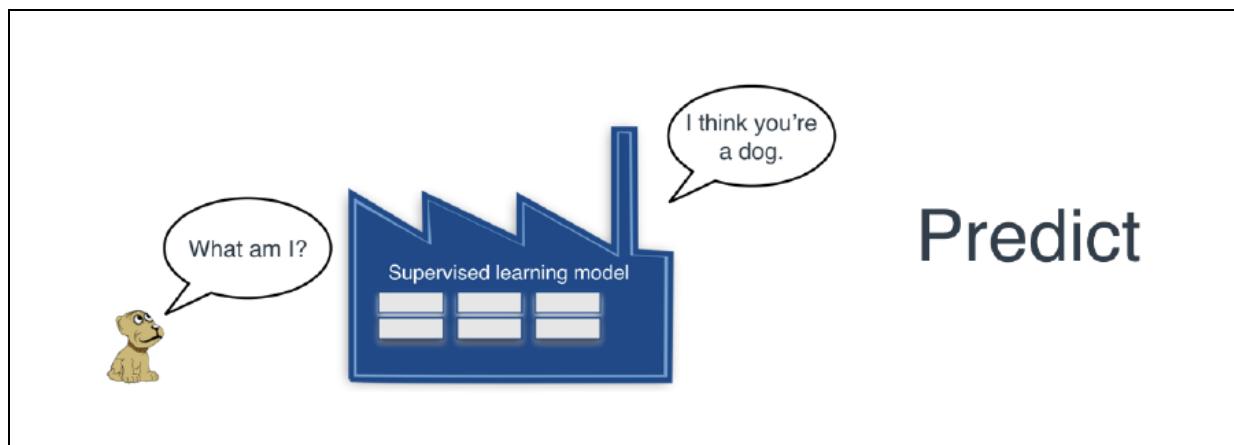
- ✓ In previous images, we have an images data about dogs and cats.
- ✓ **Labels** in the image are '**dog**' and '**cat**'.
- ✓ The machine learning model use previous data in order to predict the label of **new data points**.



7. Remember – Formulate - Predict.

- ✓ Supervised learning works in remember – formulate – predict
- ✓ The model first remembers the dataset of dogs and cats.
- ✓ Then model formulates a model for **what is a dog** and **what is a cat**.
- ✓ Whenever a **new image comes** in, the model makes a prediction about what the label of the image weather cat or dot





8. Types of Supervised learning models

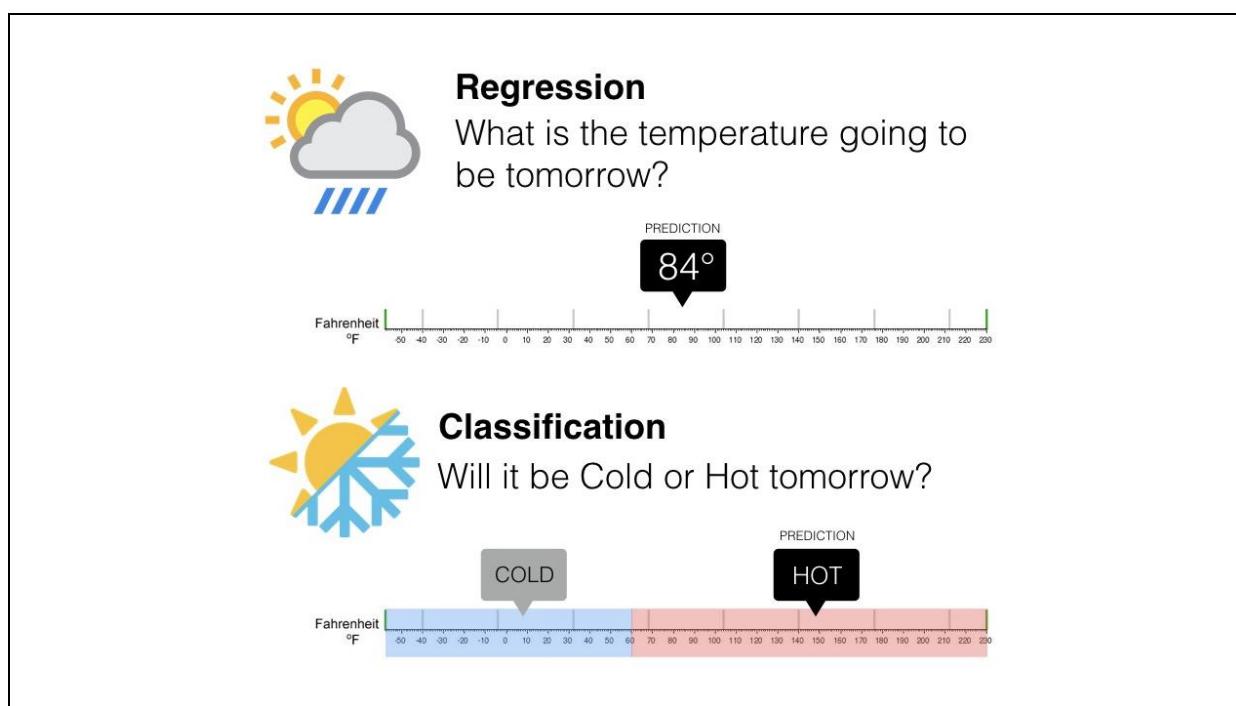
- ✓ Supervised learning models are divided into two types
 - Regression models
 - Classification models

8.1. Regression models

- ✓ Regression models used to predict a **number**
- ✓ The output of a regression model is a **continuous**, since the prediction can be any real value.
- ✓ Examples:
 - Weight of the animal
 - Employee salary
 - Students marks
 - Stock market
 - Number of sales
 - Predicting price of house & etc

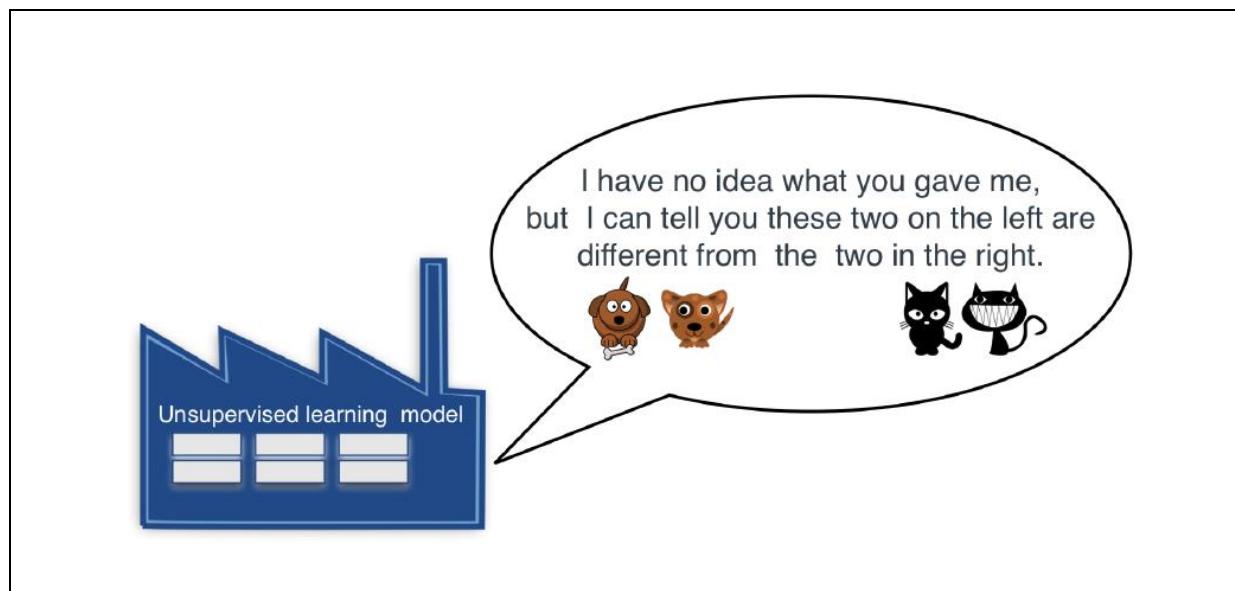
8.2. Classification models

- ✓ These are the types of models that predict the **class or state**.
- ✓ Examples
 - Type of animal (cat or dog),
 - Type of human being means male or female,
 - Biryani taste: good or bad or not good
 - Mail id spam or ham



9. Unsupervised learning example

- ✓ In unsupervised learning, we will train the models by using only input features but there are no labels
- ✓ Unsupervised learning is grouping the data based on similarities

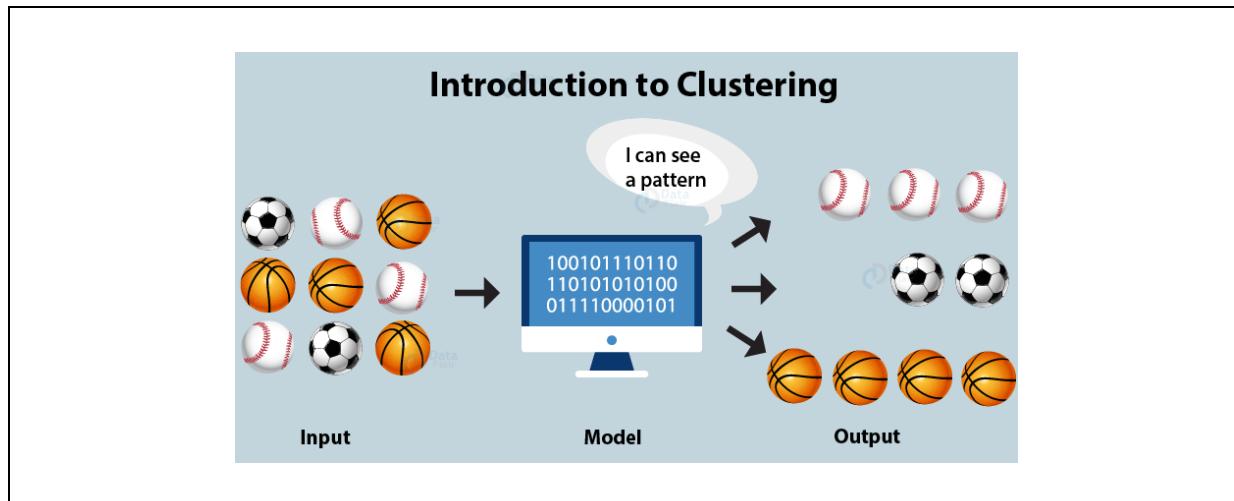
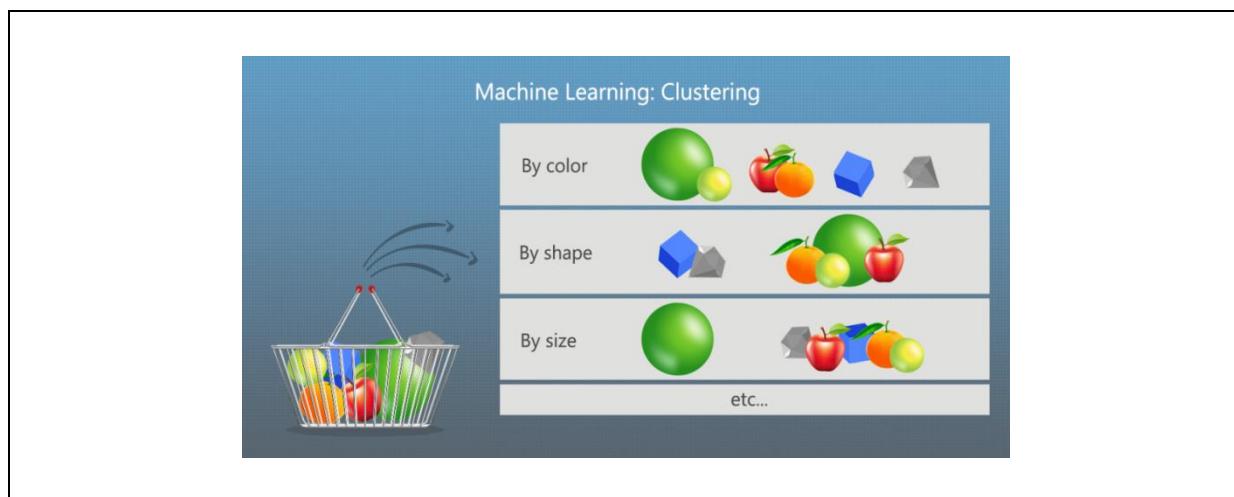
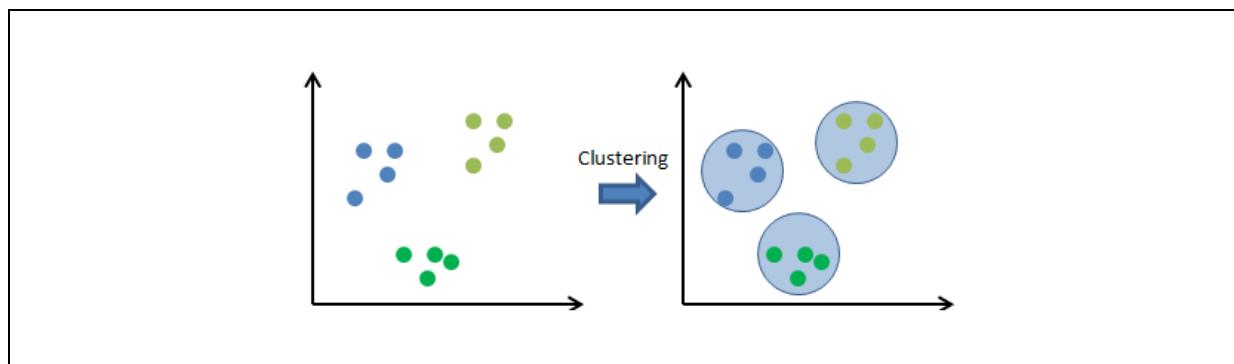


10. Types of unsupervised learning models

- ✓ Unsupervised learning models are divided into two types
 - Clustering
 - Dimensionality reduction

10.1. Clustering

- ✓ This is the task of grouping our data into clusters based on similarity.



10.2. Dimensionality reduction

- ✓ This is the task of simplifying our data and describing it with fewer features, without losing much generality.
- ✓ The dimensionality reduction algorithms will find ways that group them, losing as little information as possible.

6. Data Science – Machine Learning – Life Cycle

Contents

1. Machine learning life cycle	2
1.1. Data Collection.....	3
1.2. Data Preparation.....	3
1.3. Data Wrangling	3
1.4. Train the model.....	3
1.5. Test the model	4
1.6. Model deployment	4

6. Data Science – Machine Learning – Life Cycle

1. Machine learning life cycle

- ✓ There are mainly six steps involved here,
 - Data collection
 - Data preparation
 - Data Wrangling
 - Train the model
 - Test the model
 - Model deployment

1.1. Data Collection

- ✓ Data Gathering is the first step in machine learning life cycle.
- ✓ In this step, we need to identify the different data sources.
- ✓ Data can be collected from different sources such as files, database, web & etc.

1.2. Data Preparation

- ✓ During data preparation we should understand about the data format, quality of data & etc.
- ✓ A better understanding of data leads to an effective outcome.
- ✓ We will find correlations, general trends, and outliers.

1.3. Data Wrangling

- ✓ Data wrangling is the process of cleaning and converting raw data into a useable format.
- ✓ It is the process of,
 - Cleaning the data
 - Missing Values
 - Duplicate data
 - Invalid data
 - Selecting the variable to use,
- ✓ It is not necessary that data we have collected is always of our use as some of the data may not be useful.

1.4. Train the model

- ✓ During training, the model can learn different patterns and rules.
- ✓ We need to use train dataset to train the model.

1.5. Test the model

- ✓ Once model trained then testing is required to check the accuracy.
- ✓ We need to use test dataset to test the model.

1.6. Model deployment

- ✓ Once model trained and tested, if model is producing good results then we can deploy the model in the real time.

7. Data Science – Machine Learning – Train & Test Datasets

Contents

1. Types of Datasets in machine learning.....	2
2. Train dataset.....	3
3. Test dataset	3
4. How to decide size of these 3 sets?	3
5. Creating array	4
6. train_test_split(p) function.....	5
7. train_test_split(p, random_state = 0) function.....	18

7. Data Science – Machine Learning – Train & Test Datasets

1. Types of Datasets in machine learning

- ✓ There are mainly 3 types of datasets used in Machine learning,
 - Train dataset
 - Test dataset
 - Validation dataset

Note

- ✓ Here validation dataset is an optional but training and test datasets are mandatory



2. Train dataset

- ✓ Train dataset is used to train the models.
- ✓ This is the part of dataset which is used to train the model.
- ✓ Typically, training set contains about 60-70% of total dataset.
- ✓ First step is, model should get train with training dataset, during training model learns the parameters and underlying concepts from dataset.

3. Test dataset

- ✓ Test dataset is used to test the models.
- ✓ Once model training is done then we need to test the model with test dataset.
- ✓ During testing the model we will understand about model performance either good or not.
- ✓ Size of Test set is about 15-30% of total dataset.

4. How to decide size of these 3 sets?

- ✓ There is no thumb rule about choosing the size of these sets, but according to the experts' 70-30 or 60-40 is a good size for train and test set respectively.

5. Creating array

- ✓ We can create an array and split that array

Program Creating an array
Name demo1.py

```
import numpy as np

dataset = np.arange(10)

print(dataset)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
```

6. train_test_split(p) function

- ✓ train_test_split(p) is a predefined function in sklearn.model_selection package
- ✓ We need to access this function from sklearn package.
- ✓ By using this function we can split the dataset into train dataset and test dataset.

Program Name Creating an array and splitting dataset
demo2.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

result = train_test_split(dataset)

print(dataset)
print(result)
```

Output

```
C:\Users\Nireekshan\Desktop\PROGRAMS>py demo1.py
[0 1 2 3 4 5 6 7 8 9]
[array([0, 1, 8, 9, 2, 5, 7]), array([3, 6, 4])]

C:\Users\Nireekshan\Desktop\PROGRAMS>py demo1.py
[0 1 2 3 4 5 6 7 8 9]
[array([3, 2, 6, 4, 1, 5, 9]), array([7, 8, 0])]

C:\Users\Nireekshan\Desktop\PROGRAMS>py demo1.py
[0 1 2 3 4 5 6 7 8 9]
[array([7, 8, 4, 3, 0, 6, 2]), array([5, 1, 9])]
```

Program Name Creating an array and splitting
demo3.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
[0 6 5 1 4 9 7]
[8 3 2]
```

Program Name Creating an array and splitting demo4.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, test_size = 4)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
[6 5 7 9 3 1]
[2 0 4 8]
```

Program Name Creating an array and splitting demo5.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 6)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
[1 9 4 3 2 5]
[7 8 6 0]
```

Program Name Creating an array and splitting
demo6.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, test_size = 0.4)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
[9 5 2 0 8 7]
[1 4 3 6]
```

Program Name Creating an array and splitting demo7.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 0.6)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
[9 6 4 8 3 5]
[2 1 0 7]
```

Program Name Creating an array and splitting
demo8.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 4, test_size = 4)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
[0 9 3 6]
[4 7 2 5]
```

Program Name Creating an array and splitting demo9.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 4, test_size
= 10)

print(dataset)
print()
print(X_train)
print(X_test)
```

Output

```
ValueError: test_size=10 should be either positive and smaller than the number of samples 10 or a float
in the (0, 1) range
```

Program Name Creating an array and splitting
demo10.py

```
import numpy as np

dataset = np.arange(20)

print(dataset)
```

Output

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Program Name Creating an array and splitting
demo11.py

```
import numpy as np

dataset = np.arange(20).reshape(2, 10)

print(dataset)
```

Output

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]]
```

Program Name Creating an array and splitting
demo12.py

```
import numpy as np

dataset = np.arange(20).reshape(2, 10).T

print(dataset)
```

Output

```
[[ 0 10]
 [ 1 11]
 [ 2 12]
 [ 3 13]
 [ 4 14]
 [ 5 15]
 [ 6 16]
 [ 7 17]
 [ 8 18]
 [ 9 19]]
```

Program Name Creating an array and splitting
demo13.py

```
import numpy as np

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

print(X)
print()
print(y)
```

Output

```
[[ 0 10]
 [ 1 11]
 [ 2 12]
 [ 3 13]
 [ 4 14]
 [ 5 15]
 [ 6 16]
 [ 7 17]
 [ 8 18]
 [ 9 19]]

[0 1 2 3 4 5 6 7 8 9]
```

Program Name Creating an array and splitting
demo14.py

```
import numpy as np

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

print(X)
print()
print(y)
```

Output

```
[[ 0 10]
 [ 1 11]
 [ 2 12]
 [ 3 13]
 [ 4 14]
 [ 5 15]
 [ 6 16]
 [ 7 17]
 [ 8 18]
 [ 9 19]]

[0 1 2 3 4 5 6 7 8 9]
```

Program Name Creating an array and splitting
demo15.py

```
import numpy as np
from sklearn.model_selection import train_test_split

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

X_train, X_test, y_train, y_test = train_test_split(X, y)

print(X_train)
print()
print(X_test)
print()
print(y_train)
print()
print(y_test)
```

Output

```
[[ 9 19]
 [ 6 16]
 [ 2 12]
 [ 4 14]
 [ 3 13]
 [ 8 18]
 [ 7 17]]

[[ 5 15]
 [ 1 11]
 [ 0 10]]

[9 6 2 4 3 8 7]

[5 1 0]
```

7. `train_test_split(p, random_state = 0)` function

- ✓ `train_test_split(p, random_state = 0)` is a predefined function in `sklearn.model_selection` package
- ✓ We need to access this function from `sklearn` package.
- ✓ By using this function we can split the dataset into train dataset and test dataset.
- ✓ We will get the same train and test datasets across different executions.

Program Name Creating an array and splitting
demo16.py

```
import numpy as np
from sklearn.model_selection import train_test_split

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state = 0)

print(X_train)
print()
print(X_test)
print()
print(y_train)
print()
print(y_test)
```

Output

```
[[ 9 19]
 [ 1 11]
 [ 6 16]
 [ 7 17]
 [ 3 13]
 [ 0 10]
 [ 5 15]]

[[ 2 12]
 [ 8 18]
 [ 4 14]]

[9 1 6 7 3 0 5]

[2 8 4]
```

8. Data Science – Machine Learning – R value

Contents

1. Regression	2
2. A line	2
3. The goal	3
4. Can we use regression in everywhere?	4
5. r value	4
6. r value range.....	4
7. Calculate r value	5

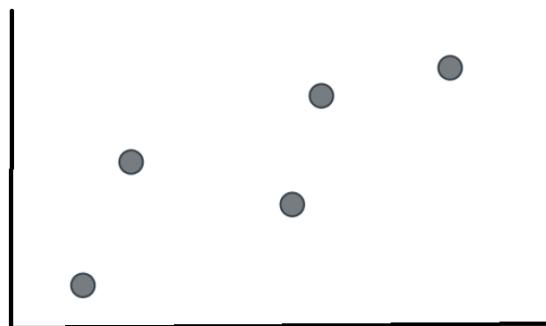
8. Data Science – Machine Learning – R value

1. Regression

- ✓ By using regression we can find the relationship between variables.
- ✓ Once we understand the relationship in between the variables then we can predict the future outcomes

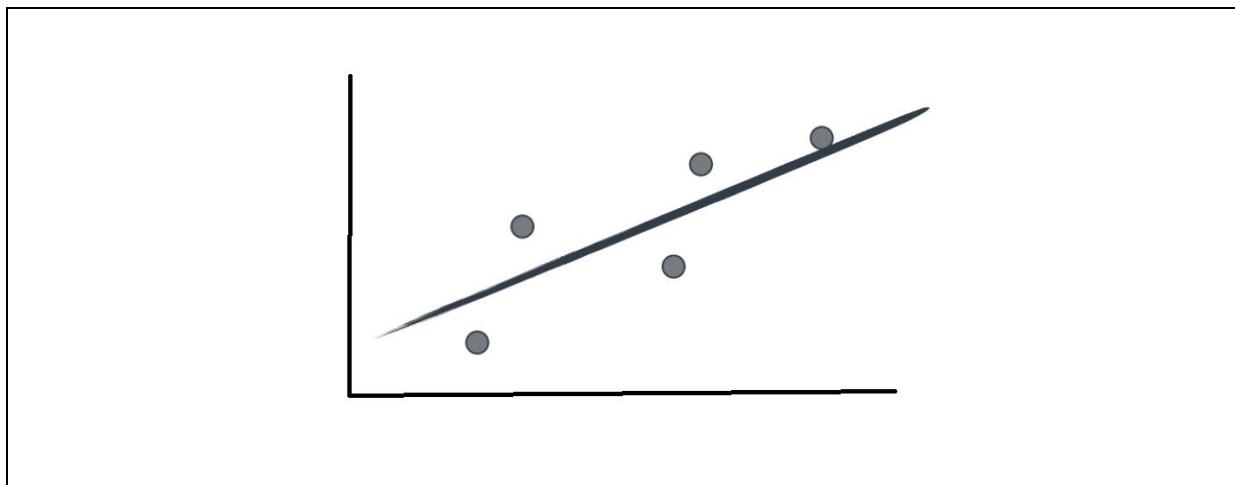
2. A line

- ✓ If two variables having relationship then if we draw this relationship in a two dimensional then we get a straight line.
- ✓ The picture of linear regression is simple.
- ✓ Let us say we have some points, a line will travel in between these points



3. The goal

- ✓ The goal of linear regression is to draw the best fitted line.
- ✓ Best fitted line means that the line which passes as close as possible to these points.



4. Can we use regression in everywhere?

- ✓ If there is a relationship in between the variables then only we can use linear regression algorithm.
- ✓ If there is no relationship in between the variables then we cannot use linear regression algorithm.

5. r value

- ✓ r value explains about how variables are related each other.
- ✓ This is very important step to recognise relationship in between values of x-axis and y-axis.

6. r value range

- ✓ The r values range, from -1 to 1.
- ✓ While calculating if we get r value as near to **-1** or **1** then those variables are **strongly related** each other.
- ✓ While calculating if we get r value as **0** then it confirms that there is **no relationship** in between the variables.

7. Calculate r value

- ✓ By using python scipy module we can calculate r value easily

Program Name Plotting x and y values
demo1.py

```
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt

d = {
    "area": [1, 2, 3, 4],
    "rice_yield": [10, 20, 30, 40]
}

df = pd.DataFrame(d)

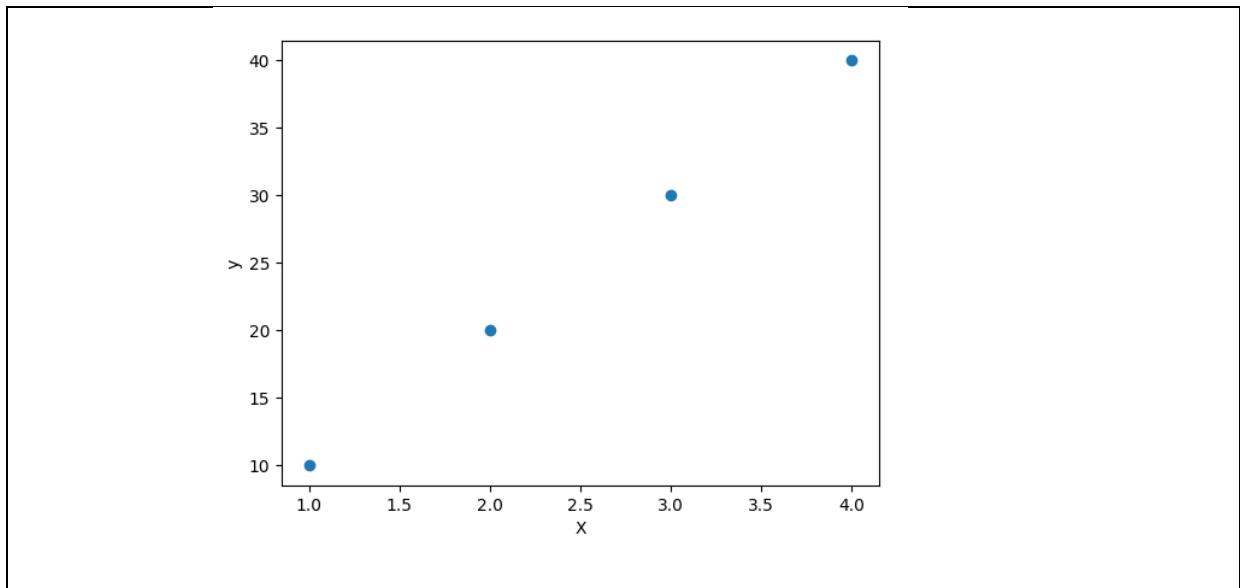
X = df.area.values
y = df.rice_yield.values

plt.xlabel("X")
plt.ylabel("y")

plt.scatter(X, y)
plt.show()
```

Output

Data Science – Machine Learning – R value



Program Name Calculating r value
demo2.py

```
import pandas as pd

d = {
    "area": [1, 2, 3, 4],
    "rice_yield": [10, 20, 30, 40]
}

df = pd.DataFrame(d)

r_value = df.corr()["rice_yield"]

print(r_value)
```

Output

1.0

Note:

- ✓ The result 1.0
- ✓ So we can confirm that there is strong a relationship.
- ✓ So we can apply linear regression algorithm on top of these variables for future prediction

Program Name Calculating r value
demo3.py

```
import pandas as pd

d = {
    "a": [600, 3000, 2, 3600, 4],
    "b": [550000, 565000, 610000, 680000, 725000]
}

df = pd.DataFrame(d)

r_value = df.corr()["b"]

print(r_value)
```

Output

-0.06533879637370224

Note:

- ✓ The result -0.06533879637370224 this value is very near to 0
- ✓ So we can confirm that there is no relationship.
- ✓ So we cannot apply linear regression algorithm on top of these variables, even if we apply then we will get wrong prediction results

9. Data Science – Machine Learning – Simple Linear Regression

Contents

1. Regression	2
2. A line	2
3. The goal	3
4. Linear Regression	4
5. Types of linear regression	4
6. Simple linear regression.....	4
7. Multiple linear regression.....	4
8. Simple linear regression example.....	5
8.1. Problem statement	5
8.2. The solution	5
9. Machine learning Terminology	6
9.1. Features	6
9.2. Label or target.....	6
9.3. Models	6
9.4. Prediction.....	6
9.5. Formula	6
10. Intercept and coefficient.....	17
11. $Y = m * X + b$ (m is coefficient and b is intercept)	18
12. Best fitted line	19
13. Predicting a group of home prices.....	21

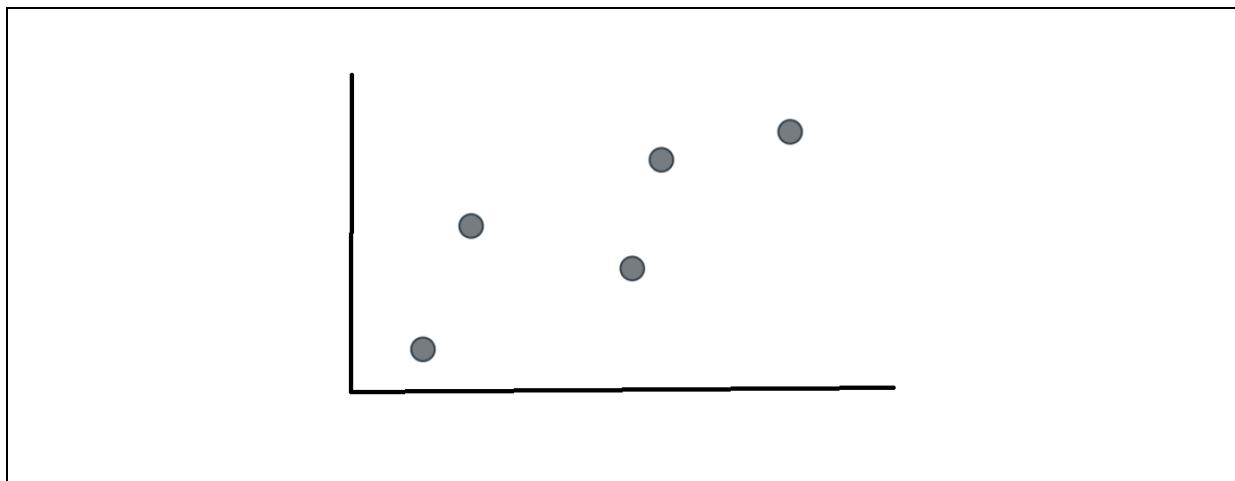
9. Data Science – Machine Learning – Simple Linear Regression

1. Regression

- ✓ Regression analysis is used to explain the relationship between two variables.
- ✓ Also called as it's a relationship in between dependent variable and one or more independent variables.

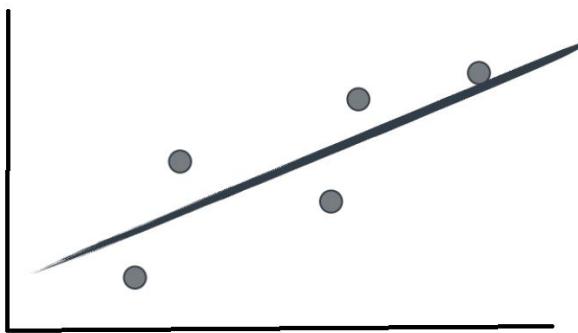
2. A line

- ✓ If two variables having relationship then if we draw this relationship in a two dimensional then we get a straight line.
- ✓ The picture of linear regression is simple.
- ✓ Let us say we have some points, a line will travel in between these points



3. The goal

- ✓ The goal of linear regression is to draw the best fitted line.
- ✓ Best fitted line means that the line which passes as close as possible to these points.



4. Linear Regression

- ✓ This is a technique and it explains the relationship between the dependent variable and independent variables

5. Types of linear regression

- ✓ There are two types of linear regression
 - Simple linear regression
 - Multiple linear regression

6. Simple linear regression

- ✓ When you have only 1 independent variable and 1 dependent variable, it is called simple linear regression.

7. Multiple linear regression

- ✓ When you have 2 or more independent variable and 1 dependent variable, it is called multiple linear regression.

8. Simple linear regression example

- ✓ When you have only 1 independent variable and 1 dependent variable, it is called simple linear regression.

8.1. Problem statement

- ✓ Assuming that we are planning to buy a new house and need to predict the price of a house

8.2. The solution

- ✓ While buying house first we need to check the area of the house

area	price
2600	550000
3000	565000
3200	610000
3600	680000
4000	725000

9. Machine learning Terminology

9.1. Features

- ✓ From the given problem the feature is **area** and **price**

9.2. Label or target

- ✓ Price of the house

9.3. Models

- ✓ A machine learning model is simply a rule, or a formula, which predicts a label from the features.
- ✓ In this case, the model is the equation we found for the price.

9.4. Prediction

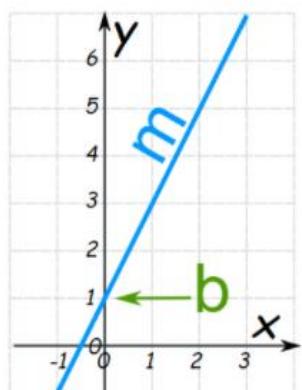
- ✓ The prediction is simply the output of the model.
- ✓ If the model gives the result as “Hey Guru I think the house with 36000 area is going to cost \$300”, then the prediction is 300.

9.5. Formula

- ✓ Home price = $m * (\text{area}) + b$

Reminder

- ✓ Once please walk through our maths regression chapter ([Chapter 7. Statistics - PART - 7 - Regression](#)) which we have already discussed, thanks



price = $m * \text{area} + b$

$$y = mx + b$$

Slope (or Gradient) Y Intercept

Program Name Loading house prices dataset
demo1.py

```
import pandas as pd

df = pd.read_csv("homeprices.csv")

print(df.head())
```

Output

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

Program Name Creating scatter plot using matplotlib
demo2.py

```
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv("homeprices.csv")

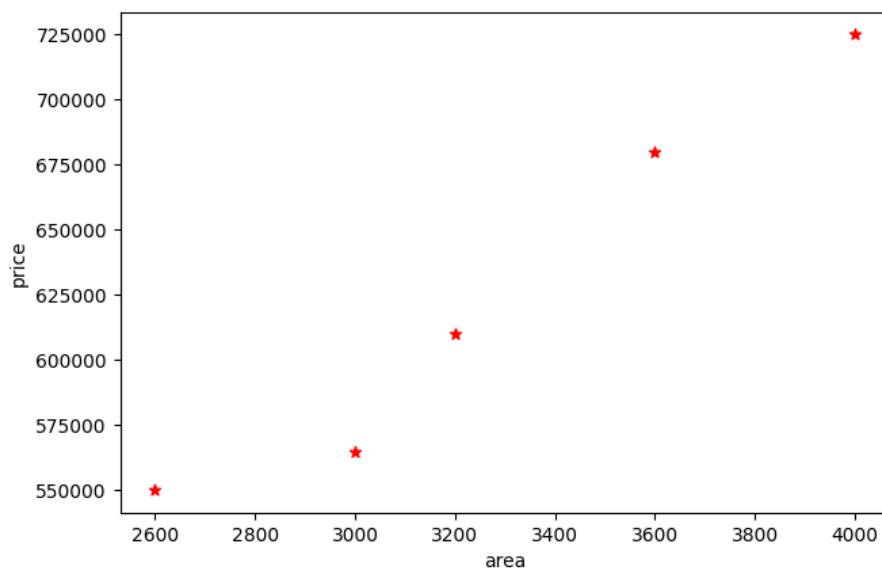
# plotting the dataset

plt.xlabel('area')
plt.ylabel('price')

plt.scatter(df.area, df.price, color = 'red', marker = '*')

plt.show()
```

Output



Program Name Loading the data set
demo3a.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis = 'columns')

print(df)
print()
print(new_df)
```

Output

```
    area    price
0  2600  550000
1  3000  565000
2  3200  610000
3  3600  680000
4  4000  725000

    area
0  2600
1  3000
2  3200
3  3600
4  4000
```

Program Name Loading the data set
demo3b.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis = 'columns')

print(new_df.values)
print()
print(df.price.values)
```

Output

```
[ [ 2600]
  [ 3000]
  [ 3200]
  [ 3600]
  [ 4000]]

[ 550000  565000  610000  680000  725000]
```

Program Name Creating LinearRegression object
demo3.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis = 'columns')

# Training the Algorithm
reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

print("Training the Algorithm")
```

Output

Training the Algorithm

Program Name Predict price of a home with area = 3300 sqr ft
demo4.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

# Making Predictions
print(reg.predict([[3300]]))
```

Output

```
[628715.75342466]
```

Program Name Predict price of a home with area = 5000 sqr ft
demo5.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

# Making Predictions
print(reg.predict([[5000]]))
```

Output

```
[859554.79452055]
```

Program Name Capture the coefficient from regression
demo6.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

print(reg.coef_)
```

Output

```
[135.78767123]
```

Program Name Capture the intercept from regression
demo7.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

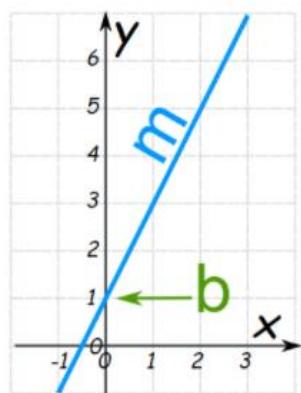
print(reg.intercept_)
```

Output

180616.43835616432

10. Intercept and coefficient

- ✓ Intercept = 180616.43835616432
- ✓ Coefficient = 135.78767123



$$\text{price} = m * \text{area} + b$$

$$y = mx + b$$

Slope (or Gradient) Y Intercept

11. $Y = m * X + b$ (m is coefficient and b is intercept)

- ✓ Let's calculate the above formula.
- ✓ In the given formula m is coefficient and b is intercept
- ✓ $Y = m * X + b$
- ✓ $Y = 135.78767123 * 3300 + 180616.43835616432$
- ✓ $Y = 628715.75342466$
- ✓ Awesome....!!!!

12. Best fitted line

- ✓ Let's calculate the above formula.
- ✓ We can draw a line

**Program
Name**

Drawing a best fitted line
demo8.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")

new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

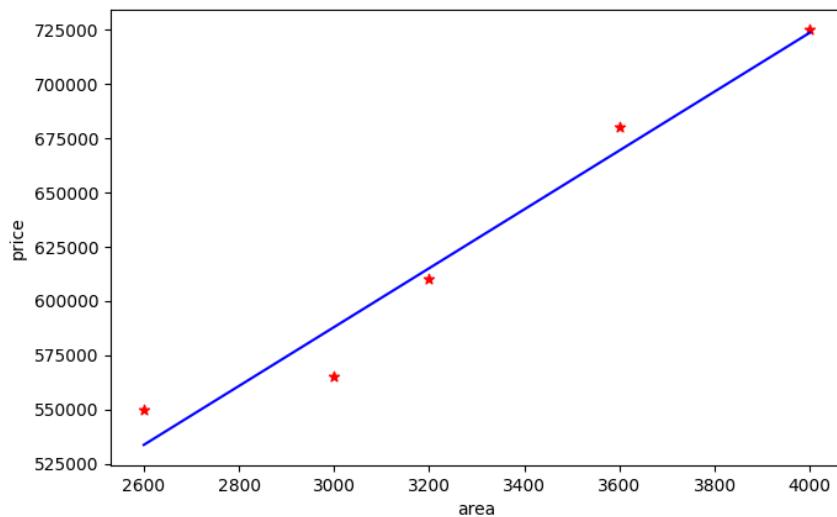
plt.xlabel('area')
plt.ylabel('price')

plt.scatter(df.area.values, df.price.values, color = 'red', marker = '*')

plt.plot(df.area.values, reg.predict(df[['area']].values), color = 'blue')

plt.show()
```

Output



13. Predicting a group of home prices

- ✓ By using above model we can predict the group of home prices as well

Program Name Loading a group of house areas
demo9.py

```
import pandas as pd
from sklearn import linear_model
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

area_df = pd.read_csv("areas.csv")
print(area_df)
```

Output

	area
0	1000
1	1500
2	2300
3	3540
4	4120
5	4560
6	5490
7	3460
8	4750
9	2300
10	9000
11	8600
12	7100

Program Name Predicting a group of home prices
demo10.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

area_df = pd.read_csv("areas.csv")

prices = reg.predict(area_df.values)
print(prices)
```

Output

```
[ 316404.10958904  384297.94520548  492928.08219178  661304.79452055
 740061.64383562  799808.21917808  926090.75342466  650441.78082192
 825607.87671233  492928.08219178  1402705.47945205  1348390.4109589
 1144708.90410959]
```

Program Name Create a csv file with predictions
demo11.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

area_df = pd.read_csv("areas.csv")
p = reg.predict(area_df.values)

area_df['prices'] = p
area_df.to_csv('output.csv')
print("Please check in current directory for output.csv")
```

Output

Please check in current directory for output.csv

9.1. Data Science – Machine Learning – Linear Regression Example

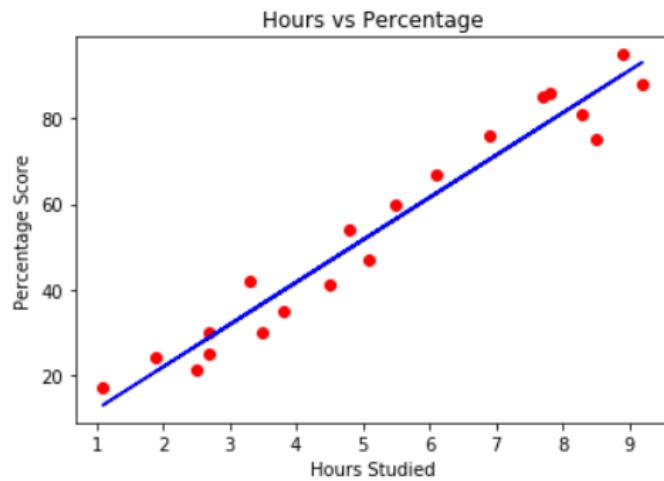
Contents

1. Scenario	2
2. Maths behind	3
3. Loading dataset	4
4. Plotting the data.....	5
5. Intercept & coefficient	10
6. Important info	12
7. Making Predictions.....	12
8. Evaluating the Algorithm	15
9. Evaluation metrics.....	15

9.1. Data Science – Machine Learning – Linear Regression Example

1. Scenario

- ✓ Let's find the relationship in between **marks** and **number of study hours**
- ✓ We want to find out the **marks** for given **number of study hours** to a student
- ✓ If we plot the independent variable (hours) on the x-axis and dependent variable (percentage) on the y-axis then linear regression gives us a straight line that best fits the data points.



2. Maths behind

- ✓ We know that the equation of a straight line is basically
 - $y = mx + b$
- ✓ Where **b** is the **intercept** and **m** is the **slope** of the line.
- ✓ So basically, the linear regression algorithm gives us the most optimal value for the **intercept** and the **slope**.
- ✓ The y and x variables remain the same
- ✓ There can be multiple straight lines depending upon the values of intercept and slope.
- ✓ Basically what the linear regression algorithm does is it fits multiple lines on the data points and returns the line that results in the least error.

3. Loading dataset

- ✓ Once we have dataset then we need to load by using pandas
- ✓ If we load dataset then it returns the DataFrame

Program Name Loading student_scores dataset
demo1.py

```
import pandas as pd

df = pd.read_csv('student_scores.csv')

print(df.head())
```

Output

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

4. Plotting the data

- ✓ Let's plot our data points to see the relationship between the data.

Program Name

Plotting the dataset

demo2.py

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('student_scores.csv')

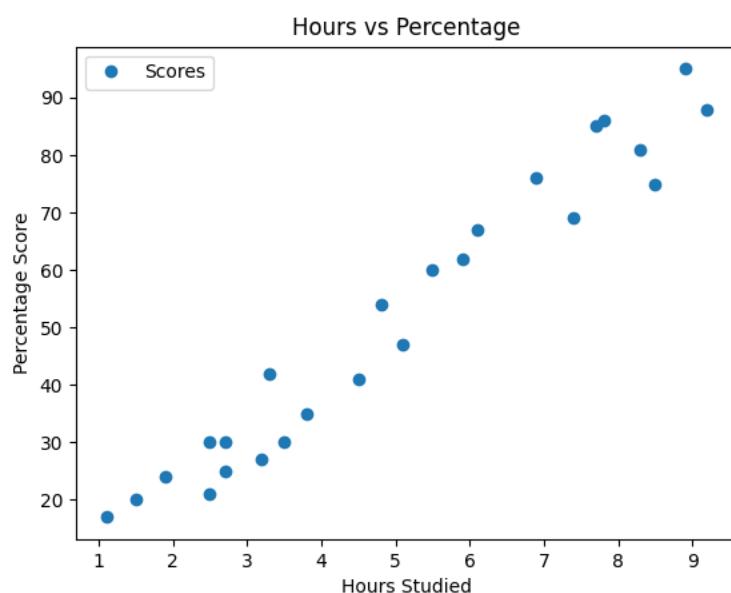
df.plot(x='Hours', y='Scores', style='o')

plt.title('Hours vs Percentage')

plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')

plt.show()
```

Output



Program Name Preparing the data
demo3.py

```
import pandas as pd

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

print(X)
print(y)
```

Output

```
[[2.5]
 [5.1]
 [3.2]
 [8.5]
 [3.5]
 [1.5]
 [9.2]
 [5.5]
 [8.3]
 [2.7]
 [7.7]
 [5.9]
 [4.5]
 [3.3]
 [1.1]
 [8.9]
 [2.5]
 [1.9]
 [6.1]
 [7.4]
 [2.7]
 [4.8]
 [3.8]
 [6.9]
 [7.8]]
[21 47 27 75 30 20 88 60 81 25 85 62 41 42 17 95 30 24 67 69 30 54 35 76
86]
```

Program Name Splitting the data
 demo4.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

print("X_train")
print(X_train)

print()
print("X_test")
print(X_test)

print()
print("Y_train")
print(y_train)

print()
print("y_test")
print(y_test)
```

Output

```
X_train
[[3.8]
 [1.9]
 [7.8]
 [6.9]
 [1.1]
 [5.1]
 [7.7]
 [3.3]
 [8.3]
 [9.2]
 [6.1]
 [3.5]
 [2.7]
 [5.5]
 [2.7]
 [8.5]
 [2.5]
 [4.8]
 [8.9]
 [4.5]]

X_test
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]]

Y_train
[35 24 86 76 17 47 85 42 81 88 67 30 25 60 30 75 21 54 95 41]

y_test
[20 27 69 30 62]
```

Program Name Training the model
demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Model got trained with data")
```

Output

Model got trained with data

5. Intercept & coefficient

- ✓ In the theory section we said that linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data.
- ✓ We can get the values of the intercept and slop from linear regression

Program

Name demo6.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.intercept_)
```

Output

2.018160041434669

Program Name Getting coefficient from created model
demo7.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.coef_)
```

Output

[9.91065648]

6. Important info

- ✓ This means that for every one unit of change in hours studied, the change in the score is about 9.91%.
- ✓ In simpler words, if a student studies **one hour more** than they previously studied for an exam, they can expect to achieve an increase of **9.91%** in the **score** achieved by the student previously.

7. Making Predictions

- ✓ Now successfully we have trained our algorithm.
- ✓ So, it's time to make some predictions.
- ✓ We need to use our test data and see how accurately our algorithm predicts the percentage score.

Program Name Making predictions
demo8.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
print(y_pred)
```

Output

```
[16.88414476 33.73226078 75.357018  26.79480124
 60.49103328]
```

Great

- ✓ The y_pred is a numpy array that contains all the predicted values for the input values in the X_test series.

Comparison

- ✓ To compare the actual output values for X_test with the predicted values.

Program Name Comparing the predicted result
demo9.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

d = {'Actual': y_test, 'Predicted': y_pred}

compare_df = pd.DataFrame(d)
print(compare_df)
```

Output

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

- ✓ Though our model is not very precise, the predicted percentages are close to the actual ones.

8. Evaluating the Algorithm

- ✓ We need to evaluate the performance of algorithm.
- ✓ This step is particularly important to compare how well different algorithms perform on a particular dataset.
- ✓ For regression algorithms there are three evaluation metrics are commonly used

9. Evaluation metrics

- ✓ Mean Absolute Error (**MAE**)
- ✓ Mean Squared Error (**MSE**)
- ✓ Root Mean Squared Error (**RMSE**)

Mean Absolute Error (MAE)

- ✓ Mean Absolute Error (MAE) is the mean of the absolute value of the errors.

Mean Squared Error (MSE)

- ✓ Mean Squared Error (MSE) is the mean of the squared errors.

Root Mean Squared Error (RMSE)

- ✓ Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors

$$MAE = \frac{1}{n} \sum \left| y - \hat{y} \right|$$

Divide by the total number of data points

Predicted output value

Actual output value

Sum of

The absolute value of the residual

$$MSE = \frac{1}{n} \sum \underbrace{\left(y - \hat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

We no need to calculate manually

- ✓ We don't have to perform these calculations manually.
- ✓ The scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Program Name	Loading student_scores dataset demo10.py
	<pre>import pandas as pd import numpy as np from sklearn.model_selection import train_test_split from sklearn.linear_model import LinearRegression from sklearn import metrics df = pd.read_csv('student_scores.csv') X = df.iloc[:, :-1].values y = df.iloc[:, 1].values X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0) regressor = LinearRegression() regressor.fit(X_train, y_train) y_pred = regressor.predict(X_test) mae = metrics.mean_absolute_error(y_test, y_pred) mse = metrics.mean_squared_error(y_test, y_pred) rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred)) print('Mean Absolute Error:', mae) print('Mean Squared Error:', mse) print('Root Mean Squared Error:', rmse)</pre>

Output

Mean Absolute Error: 4.18385989900298

Mean Squared Error: 21.598769307217413

Root Mean Squared Error: 4.647447612100368

9.2. Data Science – Machine Learning – Linear Regression Example

Program Name Loading salary dataset
demo1.py

```
import pandas as pd

dataset = pd.read_csv('Salary_Data.csv')

print(dataset.head())
```

Output

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

Program Name Preparing the data
demo2.py

```
import pandas as pd

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

print(X)
print(y)
```

Output

```
[[ 1.1]
 [ 1.3]
 [ 1.5]
 [ 2. ]
 [ 2.2]
 [ 2.9]
 [ 3. ]
 [ 3.2]
 [ 3.2]
 [ 3.7]
 [ 3.9]
 [ 4. ]
 [ 4. ]
 [ 4.1]
 [ 4.5]
 [ 4.9]
 [ 5.1]
 [ 5.3]
 [ 5.9]
 [ 6. ]
 [ 6.8]
 [ 7.1]
 [ 7.9]
 [ 8.2]
 [ 8.7]
 [ 9. ]
 [ 9.5]
 [ 9.6]
 [10.3]
 [10.5]]
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
 57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
 81363.  93940.  91738.  98273.  101302.  113812.  109431.  105582.  116969.
 112635. 122391. 121872.]
```

Program Name Splitting the dataset
demo3.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

print("X_train")
print(X_train)
print()

print("X_test")
print(X_test)
print()

print("y_train")
print(y_train)
print()

print("y_test")
print(y_test)
```

Output

```
X_train
[[ 2.9]
 [ 5.1]
 [ 3.2]
 [ 4.5]
 [ 8.2]
 [ 6.8]
 [ 1.3]
 [10.5]
 [ 3. ]
 [ 2.2]
 [ 5.9]
 [ 6.]
 [ 3.7]
 [ 3.2]
 [ 9.]
 [ 2.]
 [ 1.1]
 [ 7.1]
 [ 4.9]
 [ 4. ]]

X_test
[[ 1.5]
 [10.3]
 [ 4.1]
 [ 3.9]
 [ 9.5]
 [ 8.7]
 [ 9.6]
 [ 4.]
 [ 5.3]
 [ 7.9]]]

y_train
[ 56642.  66029.  64445.  61111.  113812.  91738.  46205.  121872.  60150.
 39891.  81363.  93940.  57189.  54445.  105582.  43525.  39343.  98273.
 67938.  56957.]]

y_test
[ 37731.  122391.  57081.  63218.  116969.  109431.  112635.  55794.  83088.
 101302.]
```

Program Name Training the model
demo4.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

print("Training the model")

regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Output

Training the model

Program Name Predicting the salaries
demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Predicting the salaries")

y_pred = regressor.predict(X_test)

print()
print(y_pred)
```

Output

```
Predicting the salaries
[ 40835.10590871 123079.39940819 65134.55626083 63265.36777221
 115602.64545369 108125.8914992 116537.23969801 64199.96201652
 76349.68719258 100649.1375447 ]
```

Program
Name

Plotting training dataset
demo6.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print("Visualizing Training dataset results ")

plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Salary vs Experience {Training set}')
plt.xlabel('Years of experience')
plt.ylabel('Salary')

plt.show()
```

Output

Visualizing Training dataset results



Program
Name

Plotting test dataset
demo7.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print("Visualizing Training dataset results ")

plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Salary vs Experience {Test set}')
plt.xlabel('Years of experience')
plt.ylabel('Salary')

plt.show()
```

Output

Visualizing Training dataset results



10. Data Science – Machine Learning – Polynomial Features

Contents

1. Polynomial Features for machine learning	2
2. Need of Polynomial Features.....	2
3. Equations	3
4. Lets create Polynomial features	19

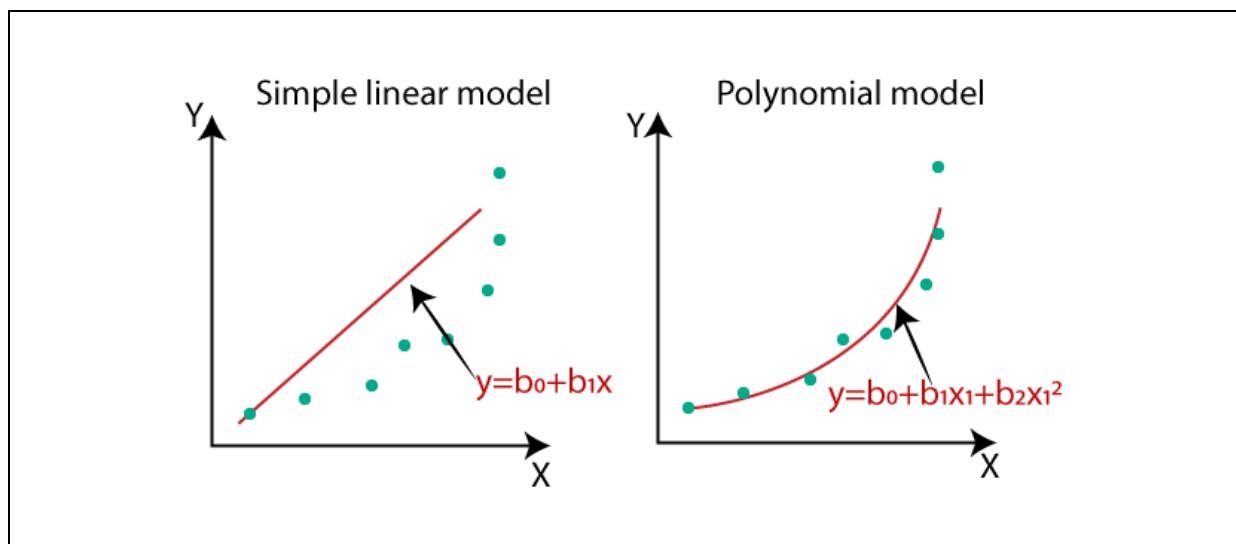
10. Data Science – Machine Learning – Polynomial Features

1. Polynomial Features for machine learning

- ✓ As such, polynomial features are a type of feature engineering means creating new input features based on the existing features.
- ✓ For example, if a dataset had one input feature X, then a polynomial feature would be the addition of a new feature (column) where values were calculated by squaring the values in X, e.g. X^2 .
- ✓ This process can be repeated for each input variable in the dataset, creating a transformed version of each.
- ✓ The "degree" of the polynomial is used to control the number of features added.

2. Need of Polynomial Features

- ✓ If we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression.
- ✓ If we apply the same model without any modification on a non-linear dataset, then it will produce wrong results
- ✓ Due to this,
 - The error rate will be high etc



3. Equations

Simple Linear Regression equation

✓ $y = b_0 + b_1 x$

Multiple Linear Regression equation

✓ $y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots + b_n x_n$

Polynomial Regression equation: $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + \dots + b_n x^n$

Program Name Creating an array
demo1.py

```
from numpy import asarray  
  
data = asarray([[2], [3], [4]])  
print(data)
```

Output

```
[[2]  
[3]  
[4]]
```

Program Name Creating feature from existing feature
demo2.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2],[3],[4]])

trans = PolynomialFeatures(degree = 1)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

Output

```
[[2]
 [3]
 [4]]

[[1. 2.]
 [1. 3.]
 [1. 4.]]
```

Program Name Creating feature from existing feature
demo3.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2],[3],[4]])

trans = PolynomialFeatures(degree = 2)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

Output

```
[[2]
 [3]
 [4]]

[[ 1.  2.  4.]
 [ 1.  3.  9.]
 [ 1.  4. 16.]]
```

Program Name Creating feature from existing feature
demo4.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2],[3],[4]])

trans = PolynomialFeatures(degree = 3)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

Output

```
[[2]
 [3]
 [4]]

[[ 1.  2.  4.  8.]
 [ 1.  3.  9. 27.]
 [ 1.  4. 16. 64.]]
```

Program Name Creating feature from existing feature
demo5.py

```
from numpy import asarray

data1 = asarray([[2, 3],[4, 5],[6, 7]])

print(data1)
```

Output

```
[[2 3]
 [4 5]
 [6 7]]
```

Program Name Creating feature from existing feature
demo6.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2, 3],[4, 5],[6, 7]])

trans = PolynomialFeatures(degree = 1)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

Output

```
[[2 3]
 [4 5]
 [6 7]]

[[1.  2.  3.]
 [1.  4.  5.]
 [1.  6.  7.]]
```

Program Name Creating feature from existing feature
demo7.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2, 3],[4, 5],[6, 7]])

trans = PolynomialFeatures(degree = 2)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

Output

```
[[2 3]
 [4 5]
 [6 7]]

[[ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]
 [ 1.  6.  7. 36. 42. 49.]]
```

Program Name Creating feature from existing feature
demo8.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2, 3],[4, 5],[6, 7]])

trans = PolynomialFeatures(degree = 3)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

Output

```
[[2 3]
 [4 5]
 [6 7]]

[[ 1.   2.   3.   4.   6.   9.   8.  12.  18.  27.]
 [ 1.   4.   5.  16.  20.  25.  64.  80. 100. 125.]
 [ 1.   6.   7.  36.  42.  49. 216. 252. 294. 343.]]
```

Program Name Loading dataset
demo9.py

```
import pandas as pd

df = pd.read_csv("poly_dataset.csv")

print(df)
```

Output

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

Program Name Data preparation
demo10.py

```
import pandas as pd

df = pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

print(X)
print()
print(y)
```

Output

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]

[ 45000   50000   60000   80000  110000  150000  200000  300000  500000
 1000000]
```

Program Name Plotting the dataset
demo11.py

```
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv("poly_dataset.csv")

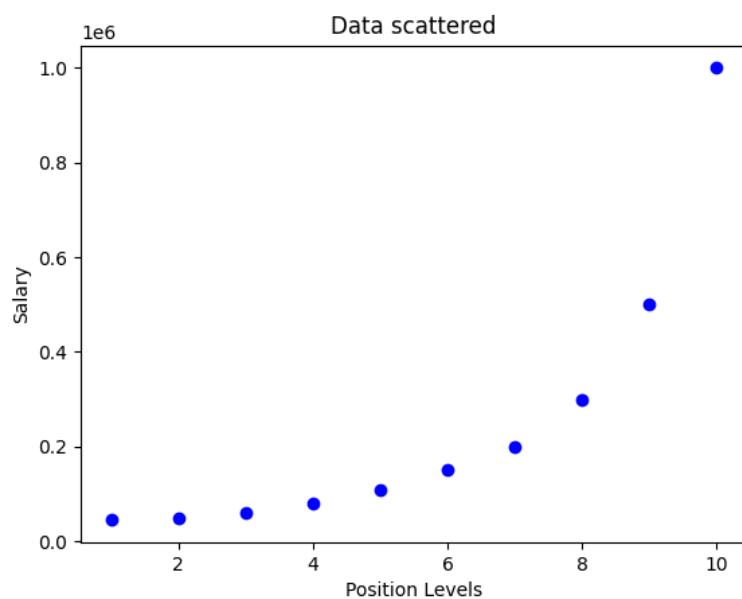
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

plt.scatter(X, y, color="blue")

plt.title("Data scattered")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

Output



Program Name Model training
demo12.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Loading the dataset
df = pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

# Model training
lin_regs= LinearRegression()
lin_regs.fit(X, y)

print("Model got trained")
```

Output

Model got trained

Program Name Model prediction
demo13.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Loading the dataset
df = pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

# Model training
lin_regs= LinearRegression()
lin_regs.fit(X, y)

print("Model got trained")
print(lin_regs.predict([[6.5]]))
```

Output

```
[330378.78787879]
```

Program Name Plotting the dataset
demo14.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

# Model training
lin_regs= LinearRegression()
lin_regs.fit(X, y)

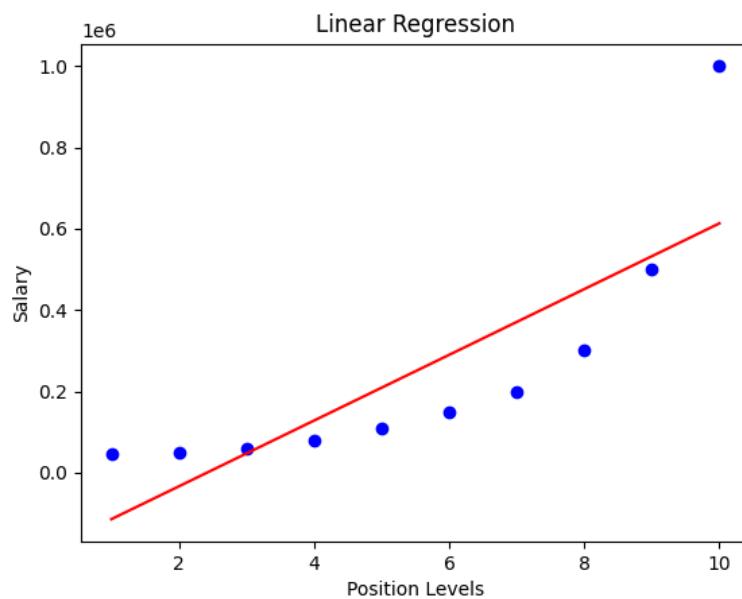
plt.scatter(X, y, color="blue")

plt.plot(X, lin_regs.predict(X), color = "red")

plt.title("Linear Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

Output



4. Lets create Polynomial features

- ✓ We need to use PolynomialFeatures class to get polynomial features.

Program

Name demo15.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

poly_regs= PolynomialFeatures(degree = 1)

x_poly= poly_regs.fit_transform(X)

print(x_poly)
```

Output

```
          Position  Level   Salary
0  Business Analyst      1    45000
1  Junior Consultant     2    50000
2  Senior Consultant     3    60000
3           Manager      4    80000
4  Country Manager       5   110000
5  Region Manager        6   150000
6        Partner         7  200000
7  Senior Partner        8  300000
8        C-level         9  500000
9         CEO          10 1000000
[[ 1.  1.]
 [ 1.  2.]
 [ 1.  3.]
 [ 1.  4.]
 [ 1.  5.]
 [ 1.  6.]
 [ 1.  7.]
 [ 1.  8.]
 [ 1.  9.]
 [ 1. 10.]]
```

Program Name Fitting the Polynomial regression to the dataset
demo16.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

poly_regs= PolynomialFeatures(degree = 2)

x_poly= poly_regs.fit_transform(X)

print(x_poly)
```

Output

```
          Position  Level   Salary
0  Business Analyst      1    45000
1  Junior Consultant     2    50000
2  Senior Consultant     3    60000
3            Manager     4    80000
4  Country Manager       5   110000
5  Region Manager        6   150000
6            Partner     7   200000
7  Senior Partner        8   300000
8            C-level     9   500000
9            CEO      10  1000000
[[ 1.  1.  1.]
 [ 1.  2.  4.]
 [ 1.  3.  9.]
 [ 1.  4. 16.]
 [ 1.  5. 25.]
 [ 1.  6. 36.]
 [ 1.  7. 49.]
 [ 1.  8. 64.]
 [ 1.  9. 81.]
 [ 1. 10. 100.]]
```

Program Name Fitting the Polynomial regression to the dataset
demo17.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

poly_regs= PolynomialFeatures(degree = 2)
x_poly= poly_regs.fit_transform(X)

model = LinearRegression()
model.fit(x_poly, y)

print("Fitting the Polynomial regression to the dataset ")
```

Output

Fitting the Polynomial regression to the dataset

Program Name Plotting Polynomial Regression features
demo18.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 2)
x_poly = poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Plotting Polynomial Regression

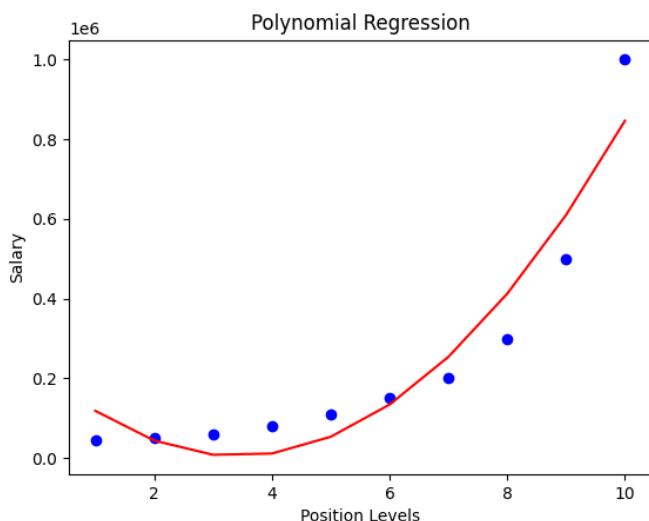
plt.scatter(X, y, color="blue")

plt.plot(X, model.predict(x_poly), color="red")

plt.title("Polynomial Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

Output



Program Name Plotting Polynomial Regression features
demo19.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 3)
x_poly = poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Plotting Polynomial Regression

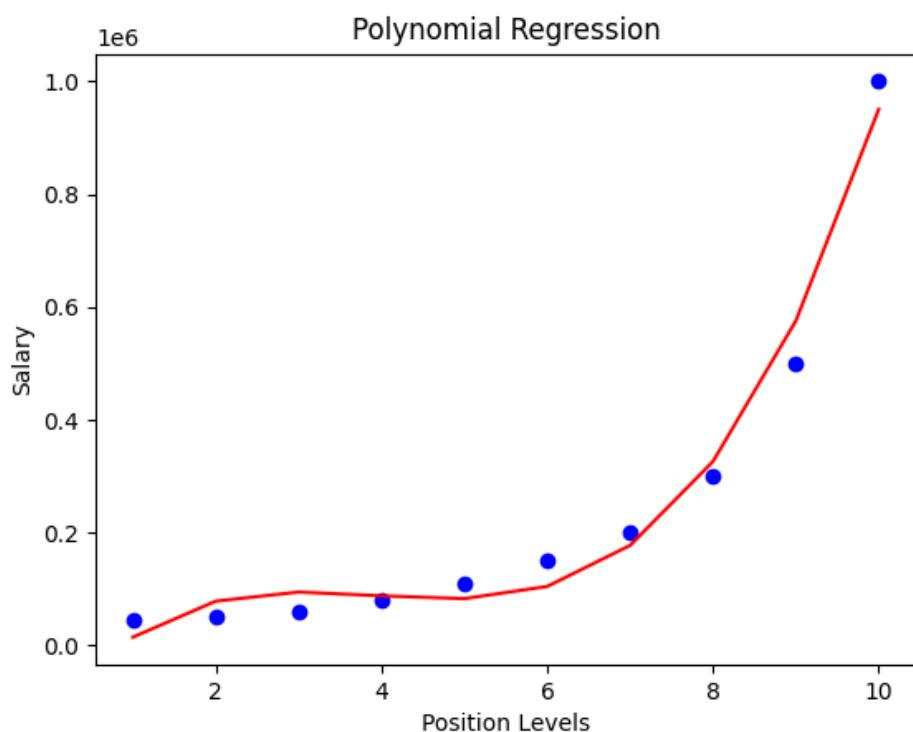
plt.scatter(X, y, color="blue")

plt.plot(X, model.predict(x_poly), color="red")

plt.title("Polynomial Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

Output



Program Name Plotting Polynomial Regression features
demo20.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 4)
x_poly = poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Plotting Polynomial Regression

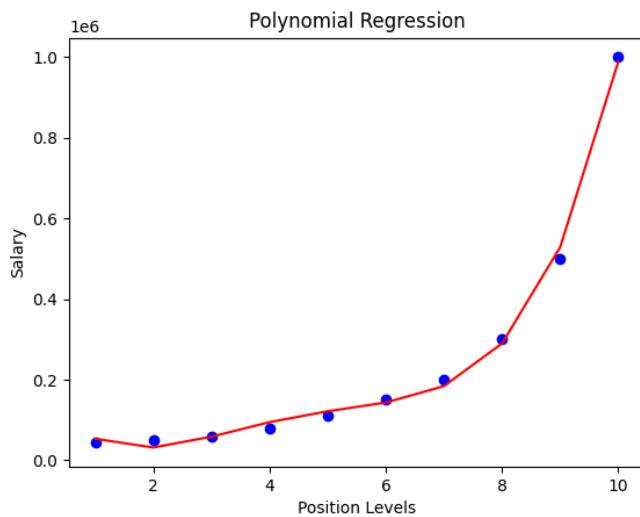
plt.scatter(X, y, color="blue")

plt.plot(X, model.predict(x_poly), color="red")

plt.title("Polynomial Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

Output



Program Name Predicting result
demo21.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 4)
x_poly= poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Prediction with Polynomial Regression
poly_pred = model.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

Output
[158862.45265155]

Note

- ✓ LinearRegression predicted output is : [330378.78787879]
- ✓ Polynomial Regression predicted output is : [158862.45265155]

Conclusion

- ✓ Polynomial Regression predicted output is the **accurate** one according to the discussion

11. Data Science – Machine Learning – Multiple Linear Regression

Contents

1. Multiple Linear Regression	2
2. Problem statement	2
3. Dataset.....	3
4. Machine learning Terminology	4
4.1. Features and label.....	4
4.2. Models	4
4.3. Prediction.....	4
4.4. Formula	5

11. Data Science – Machine Learning – Multiple Linear Regression

1. Multiple Linear Regression

- ✓ Multiple Linear Regression explains the relationship between a single dependent continuous variable and more than one independent variable.

2. Problem statement

- ✓ Assuming that we are planning to buy a new house and need to predict the price of a house.
- ✓ Here price depends on area (square feet), bed rooms and age of the home (in years).
- ✓ Given these prices we have to predict prices of new homes based on area, bed rooms and age
- ✓ Given these home prices find out price of a home that has,
 - 3000 sqr ft area, 3 bedrooms, 40 year old
 - 2500 sqr ft area, 4 bedrooms, 5 year old

3. Dataset

- ✓ homeprices1.csv is dataset we are using in this example
- ✓ This dataset contains columns as,
 - Area
 - Bedrooms
 - Age
 - Price

Area	Bedrooms	Age	price
2600	3	20	550000
3000	4	15	565000
3200		18	610000
3600	3	30	595000
4000	5	8	760000
4100	6	8	810000

4. Machine learning Terminology

4.1. Features and label

- ✓ Here **area**, **bedrooms**, **age** are called independent variables or features whereas price is a dependant variable

4.2. Models

- ✓ A machine learning model is simply a rule, or a formula, which predicts a label from the features.
- ✓ In this case, the model is the equation we found for the price.

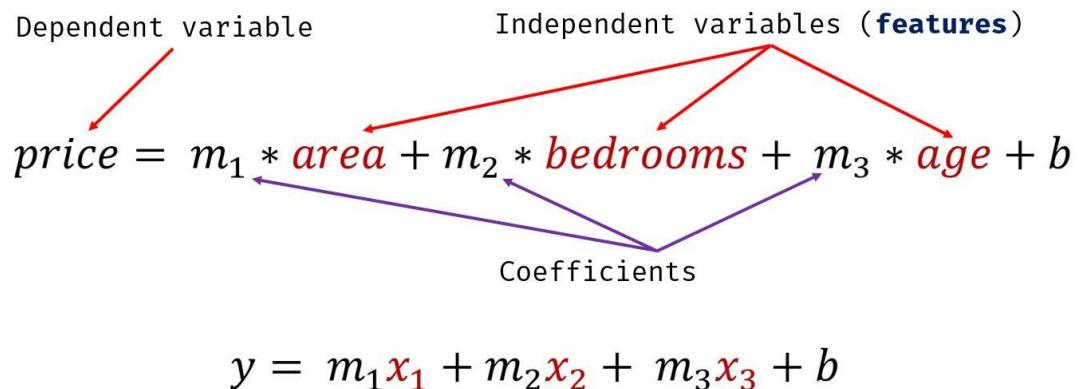
4.3. Prediction

- ✓ The prediction is simply the output of the model.

4.4. Formula

$$y = m_1 x_1 + m_2 x_2 + m_3 x_3 + b$$

$$price = m_1 * area + m_2 * bedrooms + m_3 * age + b$$



Program Name Loading house prices dataset
demo1.py

```
import pandas as pd

# Loading the dataset

df = pd.read_csv("homeprices1.csv")

print(df)
```

Output

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

**Program
Name**

Data pre-processing – Finding mean of bedrooms column
demo2.py

```
import pandas as pd

df = pd.read_csv("homeprices1.csv")

# Mean of the bedrooms
print("Mean of the bedrooms")
print(df.bedrooms.median())
```

Output

```
Mean of the bedrooms
4.0
```

Program Name Data pre-processing - Fill NA values with median value of a column
demo3.py

```
import pandas as pd

df = pd.read_csv("homeprices1.csv")

# Data loading
print("Filling missing value with mean\n")

# Data preprocessing

m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
print(df)
```

Output

```
Filling missing value with mean

      area  bedrooms   age   price
0    2600        3.0    20  550000
1    3000        4.0    15  565000
2    3200        4.0    18  610000
3    3600        3.0    30  595000
4    4000        5.0     8  760000
5    4100        6.0     8  810000
```

Program Name Model training
demo4.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)
print("Model trained")
```

Output
Model trained

**Program
Name**

Finding intercept

demo5.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df=pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

print("Intercept is:")
print(reg.intercept_)
```

Output

```
Intercept is:
221323.00186540408
```

Program Name Finding coefficients
demo6.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.vlaues, df.price)

print("Coefficients are:")
print(reg.coef_)
```

Output

```
Coefficients are:
[ 112.06244194 23388.88007794 -3231.71790863]
```

Program Name price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
demo7.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis='columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

# Prediction
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old")
print(reg.predict([[3000, 3, 40]]))
```

Output

```
price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
[498408.25158031]
```

Program Name price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
demo8.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

# Prediction
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old")

b = 112.06244194*3000 + 23388.88007794*3 + -
3231.71790863*40 + 221323.00186540384
print(b)
```

Output

```
price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
[498408.25158031]
```

Program Name price of home with 2500 sqr ft area, 4 bedrooms, 5 year old
demo9.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

# Prediction
print("price of home with 2500 sqr ft area, 4 bedrooms, 5 year old")
print(reg.predict([[2500, 4, 5]]))
```

Output

price of home with 2500 sqr ft area, 4 bedrooms, 5 year old
[578876.03748933]

12. Data Science – Machine Learning – Pickling and Unpickling

Contents

1. Pickling.....	2
2. Unpickling	2

12. Data Science – Machine Learning – Pickling and Unpickling

- ✓ Based on requirement we can write total state of object to the file and we can read total object information from the file.

1. Pickling

- ✓ The process of writing state of object to the file is called as **pickling**.
- ✓ We can implement pickling and unpickling by using python **pickle module**
- ✓ We can do pickling by using `dump(object, file)` predefined function in pickle module

▪ `pickle.dump(object, file)`

2. Unpickling

- ✓ The process of reading state of an object from the file is called **unpickling**.
- ✓ We can do unpickling by using `load(file)` predefined function in pickle module

▪ `obj = pickle.load(file)`

Program Name Pickling and unpickling example
demo1.py

```
import pickle

class Employee:
    def __init__(self, eno, ename, esal):
        self.eno = eno
        self.ename = ename
        self.esal = esal

    def display(self):
        print("Number is:", self.eno)
        print("Name is:", self.ename)
        print("Salary is:", self.esal)

with open("emp.dat", "wb") as f:
    e = Employee(100, "Daniel", 1000)
    pickle.dump(e, f)
    print("Pickling of Employee Object completed...\n")

with open("emp.dat", "rb") as f:
    obj = pickle.load(f)
    print("Printing Employee Information after unpickling")
    obj.display()
```

Output

Pickling of Employee Object completed...

Printing Employee Information after unpickling
Number is: 100
Name is: Daniel
Salary is: 1000

13. Data Science – Machine Learning – Saving model, Joblib & Pickling

Contents

1. Save model	2
2. Pickling.....	3

13. Data Science – Machine Learning – Saving model, Joblib & Pickling

1. Save model

- ✓ Once the model got created then we can save that model.
- ✓ There are two ways to save the model
 - By using python File IO pickle concept
 - By using Joblib library

Program Name Predict price of a home with area = 5000 sqr ft
demo1.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df, df.price)

# Predictions
print(reg.predict([[5000]]))
```

Output

```
[859554.79452055]
```

2. Pickling

- ✓ The process of writing state of object to the file is called as **pickling**.

Program Name Save the model into pickle file
demo2.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import pickle

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

model = LinearRegression()
model.fit(new_df.values, df.price.values)

with open('model_pickle', 'wb') as file:
    pickle.dump(model, file)

with open('model_pickle', 'rb') as file:
    model1 = pickle.load(file)
    print(model1.predict([[5000]]))
```

Output

```
[859554.79452055]
```

Program Name Doing prediction by using saved model
demo3.py

```
import pickle

with open('model_pickle', 'rb') as file:
    model1 = pickle.load(file)
    print(model1.predict([[5000]]))
```

Output

```
[859554.79452055]
```

Program Name Doing prediction by using saved model
demo4.py

```
import pickle

with open('model_pickle', 'rb') as file:
    model1 = pickle.load(file)
    print(model1.predict([[6000]]))
```

Output

```
[995342.46575342]
```

Program Name Save Trained Model Using Joblib
demo5.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import joblib

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

model = LinearRegression()
model.fit(new_df.values, df.price.values)

joblib.dump(model, 'model_joblib')

print("Model got saved")
```

Output

Model got saved

Program Name Do prediction with saved model
demo6.py

```
import joblib

mj = joblib.load('model_joblib')

print(mj.predict([[5000]]))
```

Output

```
[859554.79452055]
```

14. Data Science – Machine Learning – Cost function

Contents

1. Cost Functions in Machine Learning	2
2. Why Cost Functions in Machine Learning	2
3. How cost function works?.....	2
4. Goal of training.....	2
5. Useful sources	3

14. Data Science – Machine Learning – Cost function

1. Cost Functions in Machine Learning

- ✓ Cost functions are also called as Loss functions in machine learning.
- ✓ These are **optimization** techniques.
- ✓ There are many cost functions in machine learning
- ✓ There are different cost functions for regression and classification.

2. Why Cost Functions in Machine Learning

- ✓ During the training phase, the model assumes the initial weights randomly and tries to make a prediction on training data.
- ✓ So here, how the model gets to know how much "far" it was from the prediction?
- ✓ Model needs this information so that it can adjust the weight accordingly (using gradient descent) in the next iteration on training data.

3. How cost function works?

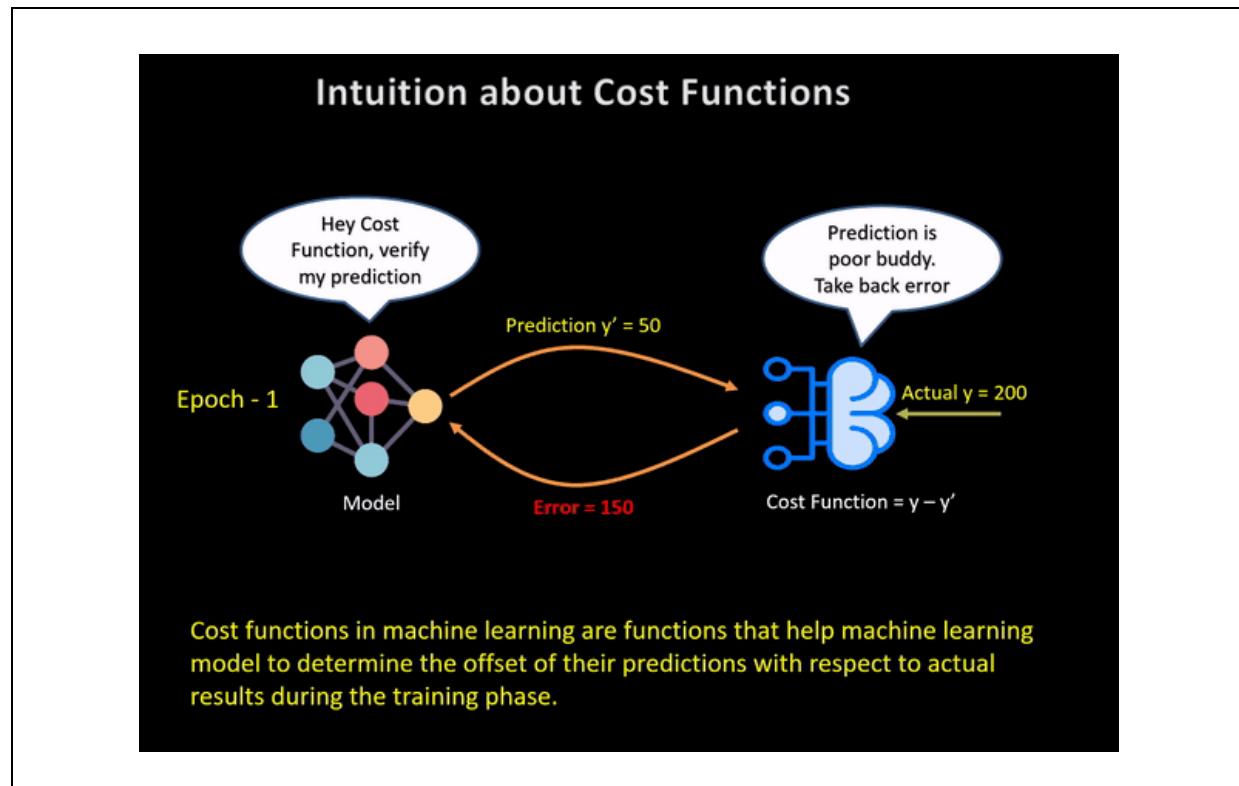
- ✓ Cost function is a function that takes both **predicted outputs** and **actual outputs** from model.
- ✓ Then it calculates how much error in the model to predict the good results.
- ✓ In next step the model tries to adjust this error for the next iteration on training data to reach optimization level

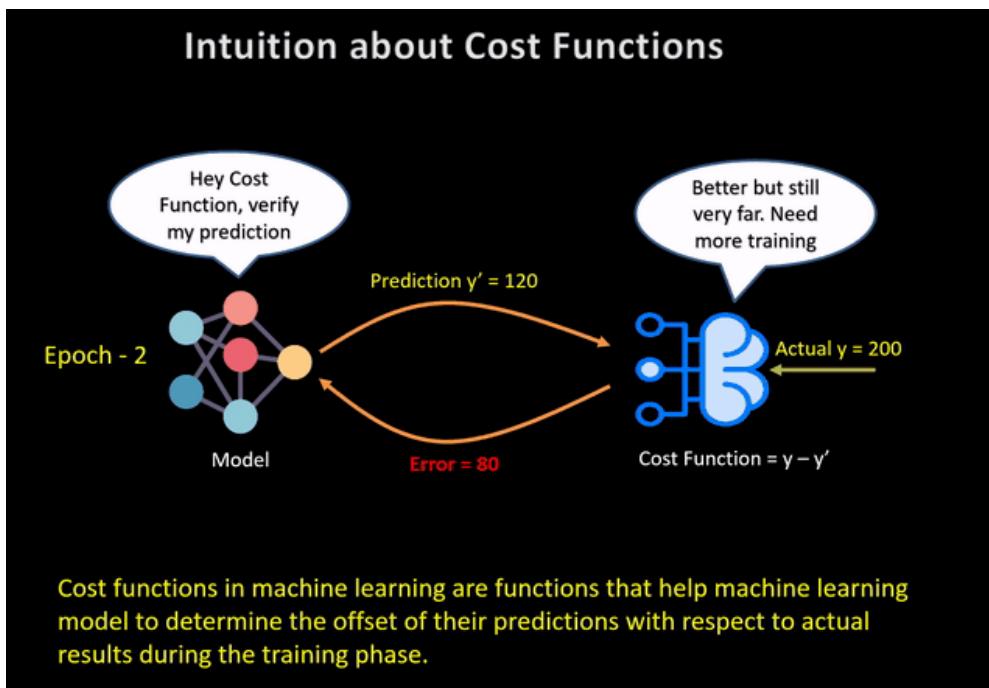
4. Goal of training

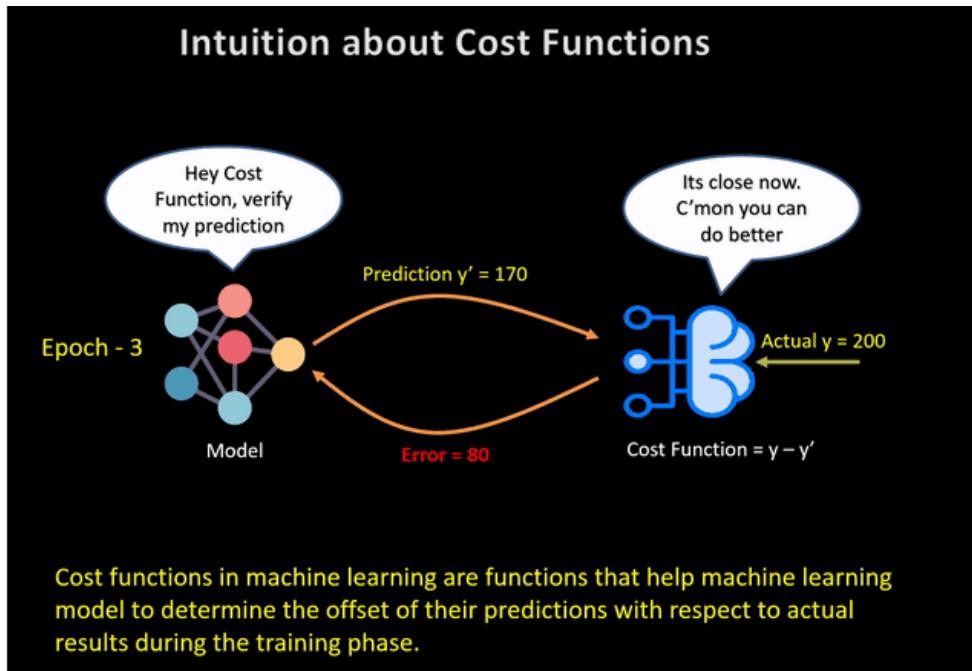
- ✓ The goal of the training phase is to come up with weights that minimize the **error in every iteration**.
- ✓ This way of training is required in machine learning to optimize model

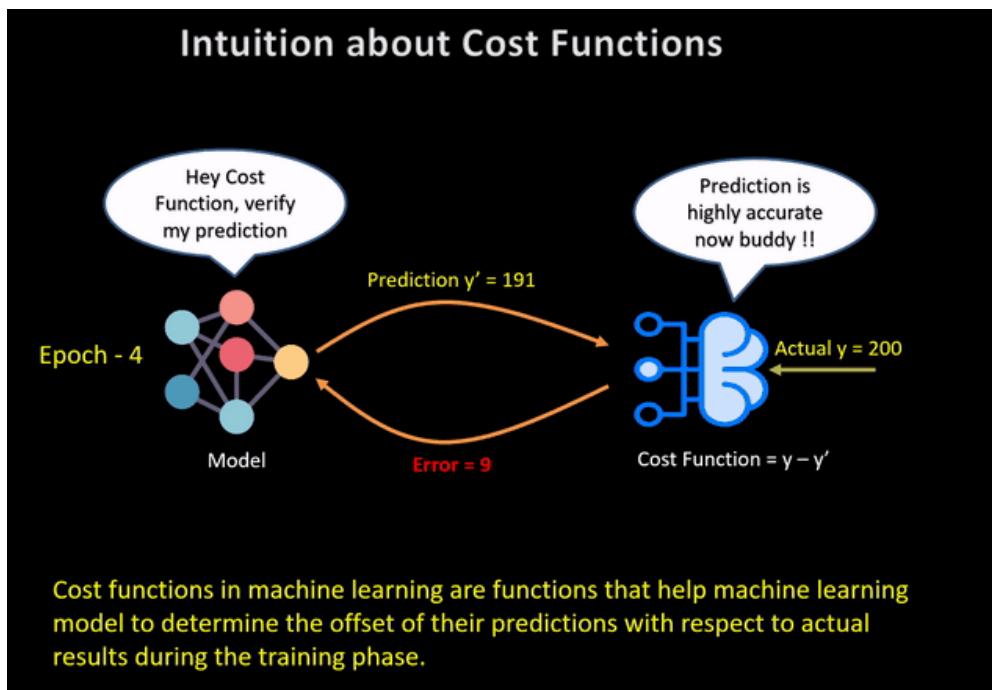
5. Useful sources

- ✓ These images are highly recommended to practice









15. Data Science – Machine Learning – Regression Cost functions

Contents

1. Cost Functions for Regression.....	2
2. Types of regression metrics	2
3. Distance Based Error	3
4. Mean Squared Error	4
5. Root Mean Squared Error	6
6. Mean Absolute Error.....	7
7. Formulas	8

15. Data Science – Machine Learning – Regression Cost functions

1. Cost Functions for Regression

- ✓ In regression, the model predicts an output value for each training data during the training phase.
- ✓ The cost functions for regression are calculated on **distance-based error**.

2. Types of regression metrics

- ✓ There are three metrics in regression,
 - Mean Squared Error (MSE).
 - Root Mean Squared Error (RMSE)
 - Mean Absolute Error (MAE)

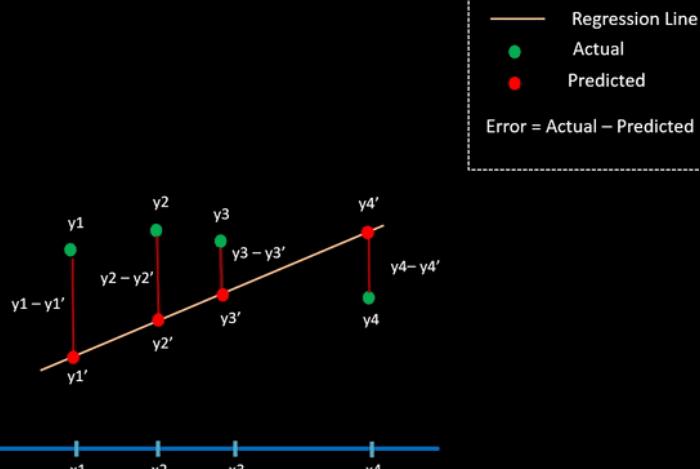
3. Distance Based Error

- Assuming that for a given set of input data, the **actual** output was y and our regression model **predicts y'** then the error in prediction is calculated as

$$\blacksquare \quad \text{Error} = y - y'$$

- This also known as **distance-based error** and it forms the basis of cost functions that are used in regression models.

Regression – Distance Based Error



4. Mean Squared Error

- ✓ The MSE is calculated as the **mean of the squared differences between predicted and expected target values** in a dataset.
- ✓ This value never be negative, since we are always squaring the errors.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Program Name Means Squared Error
demo1.py

```
from sklearn.metrics import mean_squared_error

expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]

mse_value = mean_squared_error(expected, predicted)

print(mse_value)
```

Output

```
0.3500000000000003
```

5. Root Mean Squared Error

- ✓ The Root Mean Squared Error, or RMSE, is an extension of the mean squared error.
 - $\text{RMSE} = \sqrt{\text{MSE}}$

<p>Program Name</p>	Root Means Squared Error demo2.py
	<pre>from sklearn.metrics import mean_squared_error expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0] predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0] rmse_value = mean_squared_error(expected, predicted, squared = False) print(rmse_value)</pre>
Output	0.5916079783099616

6. Mean Absolute Error

- ✓ MAE, average absolute difference between predicted and actual values.

Program Name Means Absolute Error
demo3.py

```
from sklearn.metrics import mean_absolute_error

expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]

mae_value = mean_absolute_error(expected, predicted)

print(mae_value)
```

Output

0.5

7. Formulas

Mean squared error

$$\text{MSE} = \frac{1}{n} \sum_{t=1}^n e_t^2$$

Root mean squared error

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{t=1}^n e_t^2}$$

Mean absolute error

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^n |e_t|$$

16. Data Science – ML – Dummy Variable & OneHotEncoding

Contents

1. Dataset : homeprices2.csv	2
2. Categorical data.....	3
3. How to handle text data in town column	4
4. Model may behaves like below,	4
5. Dummy variables.....	5

16. Data Science – ML – Dummy Variable & OneHotEncoding

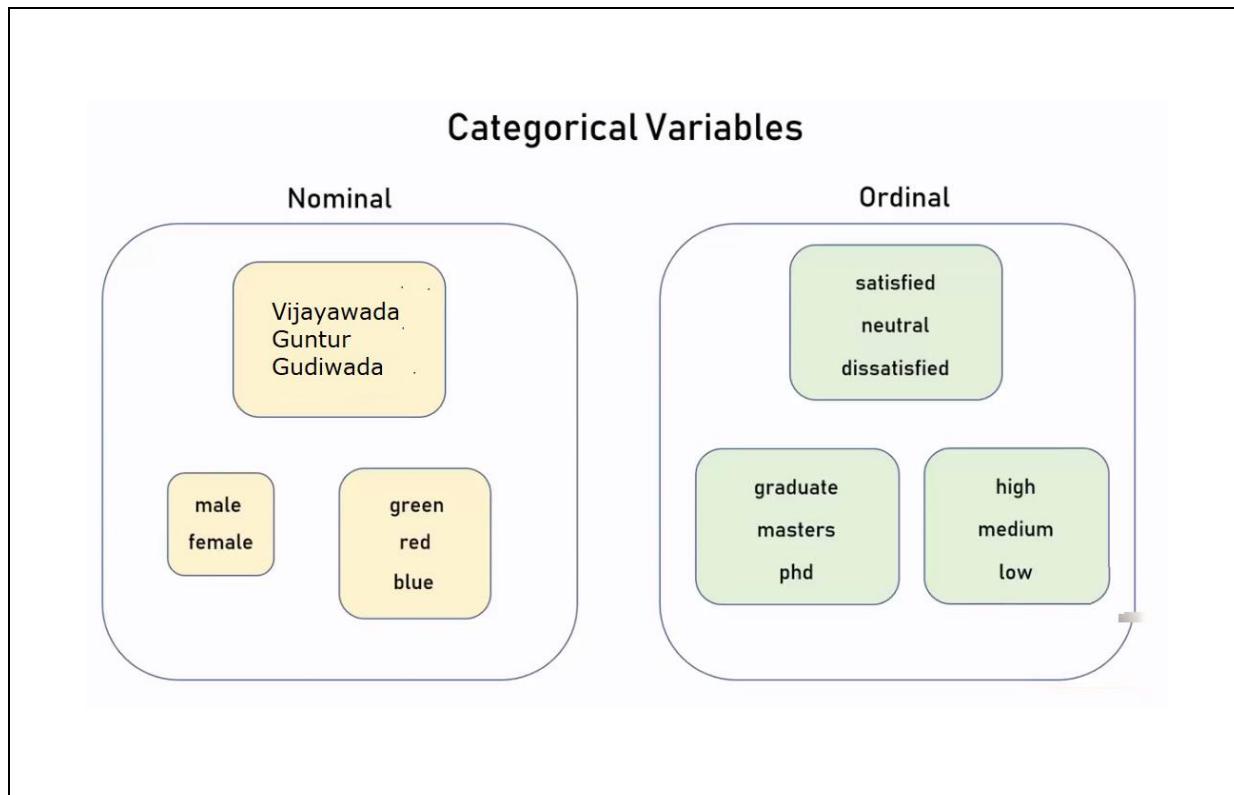
1. Dataset : homeprices2.csv

town	area	price
Vijayawada	2600	550000
Vijayawada	3000	565000
Vijayawada	3200	610000
Vijayawada	3600	680000
Vijayawada	4000	725000
Guntur	2600	585000
Guntur	2800	615000
Guntur	3300	650000
Guntur	3600	710000
Gudiwada	2600	575000
Gudiwada	2900	600000
Gudiwada	3100	620000
Gudiwada	3600	695000

Prediction

- ✓ With 3400 sqr ft area in Guntur
- ✓ With 2800 sqr ft area in Gudiwada

2. Categorical data



3. How to handle text data in town column

- ✓ Here town column having text data
- ✓ How to handle text data in numeric model?

4. Model may behaves like below,

- ✓ We can convert text data into number and can proceed to the next step.
 - Vijayawada - 1
 - Guntur - 2
 - Gudiwada - 3
- ✓ If we are giving data to the model then model assumes that the order like 1, 2, 3 and follow the below approach
 - Vijayawada < Guntur < Gudiwada
 - Or
 - Vijayawada + Guntur = Gudiwada

5. Dummy variables

- ✓ So, in this scenario we need to create dummy variable

Town	Area	Price	Gudiwada	Guntur	Vijayawada
Gudiwada	2600	575000	1	0	0
Gudiwada	2900	600000	1	0	0
Gudiwada	3100	620000	1	0	0
Gudiwada	3600	695000	1	0	0
Guntur	2600	585000	0	1	0
Guntur	2800	615000	0	1	0
Guntur	3300	650000	0	1	0
Guntur	3600	710000	0	1	0
Vijayawada	2600	550000	0	0	1
Vijayawada	3000	565000	0	0	1
Vijayawada	3200	610000	0	0	1
Vijayawada	3600	680000	0	0	1
Vijayawada	4000	725000	0	0	1

Program Name Loading dataset
demo1.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")

print(df)
```

Output

	town	area	price
0	Vijayawada	2600	550000
1	Vijayawada	3000	565000
2	Vijayawada	3200	610000
3	Vijayawada	3600	680000
4	Vijayawada	4000	725000
5	Guntur	2600	585000
6	Guntur	2800	615000
7	Guntur	3300	650000
8	Guntur	3600	710000
9	Gudiwada	2600	575000
10	Gudiwada	2900	600000
11	Gudiwada	3100	620000
12	Gudiwada	3600	695000

Program Name Creating dummy variables by using pandas
demo2.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)

print(dummies)
```

Output

	Gudiwada	Guntur	Vijayawada
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
5	0	1	0
6	0	1	0
7	0	1	0
8	0	1	0
9	1	0	0
10	1	0	0
11	1	0	0
12	1	0	0

Program Name Creating dummy variables and adding to the dataframe
demo3.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies], axis='columns')

print(merged)
```

Output

	town	area	price	Gudiwada	Guntur	Vijayawada
0	Vijayawada	2600	550000	0	0	1
1	Vijayawada	3000	565000	0	0	1
2	Vijayawada	3200	610000	0	0	1
3	Vijayawada	3600	680000	0	0	1
4	Vijayawada	4000	725000	0	0	1
5	Guntur	2600	585000	0	1	0
6	Guntur	2800	615000	0	1	0
7	Guntur	3300	650000	0	1	0
8	Guntur	3600	710000	0	1	0
9	Gudiwada	2600	575000	1	0	0
10	Gudiwada	2900	600000	1	0	0
11	Gudiwada	3100	620000	1	0	0
12	Gudiwada	3600	695000	1	0	0

Program Name Creating dummy variables and preparing the dataframe demo4.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)

merged = pd.concat([df, dummies], axis='columns')

final = merged.drop(['town'], axis='columns')

print(final)
```

Output

	area	price	Gudiwada	Guntur	Vijayawada
0	2600	550000	0	0	1
1	3000	565000	0	0	1
2	3200	610000	0	0	1
3	3600	680000	0	0	1
4	4000	725000	0	0	1
5	2600	585000	0	1	0
6	2800	615000	0	1	0
7	3300	650000	0	1	0
8	3600	710000	0	1	0
9	2600	575000	1	0	0
10	2900	600000	1	0	0
11	3100	620000	1	0	0
12	3600	695000	1	0	0

Program Name Preparing X and y
demo5.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies], axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

print(X)
print(y)
```

Output

```
area  Gudiyawada  Guntur  Vijayawada
0    2600        0        0        1
1    3000        0        0        1
2    3200        0        0        1
3    3600        0        0        1
4    4000        0        0        1
5    2600        0        1        0
6    2800        0        1        0
7    3300        0        1        0
8    3600        0        1        0
9    2600        1        0        0
10   2900        1        0        0
11   3100        1        0        0
12   3600        1        0        0
0      550000
1      565000
2      610000
3      680000
4      725000
5      585000
6      615000
7      650000
8      710000
9      575000
10     600000
11     620000
12     695000
Name: price, dtype: int64
```

Program Name Creating a model
demo6.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies], axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print("Model got trained")
```

Output

Model got trained

Program Name Predicting the house prices
demo7.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict(X))
```

Output

```
[539709.73984091 590468.71640508 615848.20468716 666607.18125133
 717366.1578155 579723.71533005 605103.20361214 668551.92431735
 706621.15674047 565396.15136531 603465.38378844 628844.87207052
 692293.59277574]
```

Program Name Checking the score
demo8.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.score(X, y))
```

Output

0.9573929037221873

Program Name Predicting house price in Vijayawada
demo9.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict([[3400, 0, 0, 1]]))
```

Output

```
[641227.69296925]
```

Program Name Predicting house price in Guntur
demo10.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict([[3400, 0, 1, 0]]))
```

Output

```
[681241.66845839]
```

Program Name Predicting house price in Gudiwada
demo11.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict([[3400, 1, 0, 0]]))
```

Output

[666914.10449365]

16. Data Science – Machine Learning – Gradient Descent

Contents

1. Gradient Descent.....	2
2. How Gradient Descent works	2
3. Convergence	3
4. Process behind the gradient	3
5. Steps	5
6. Follow below images.....	7
7. Fixed steps	11
8. Learning rate: Reaching minimum error	12

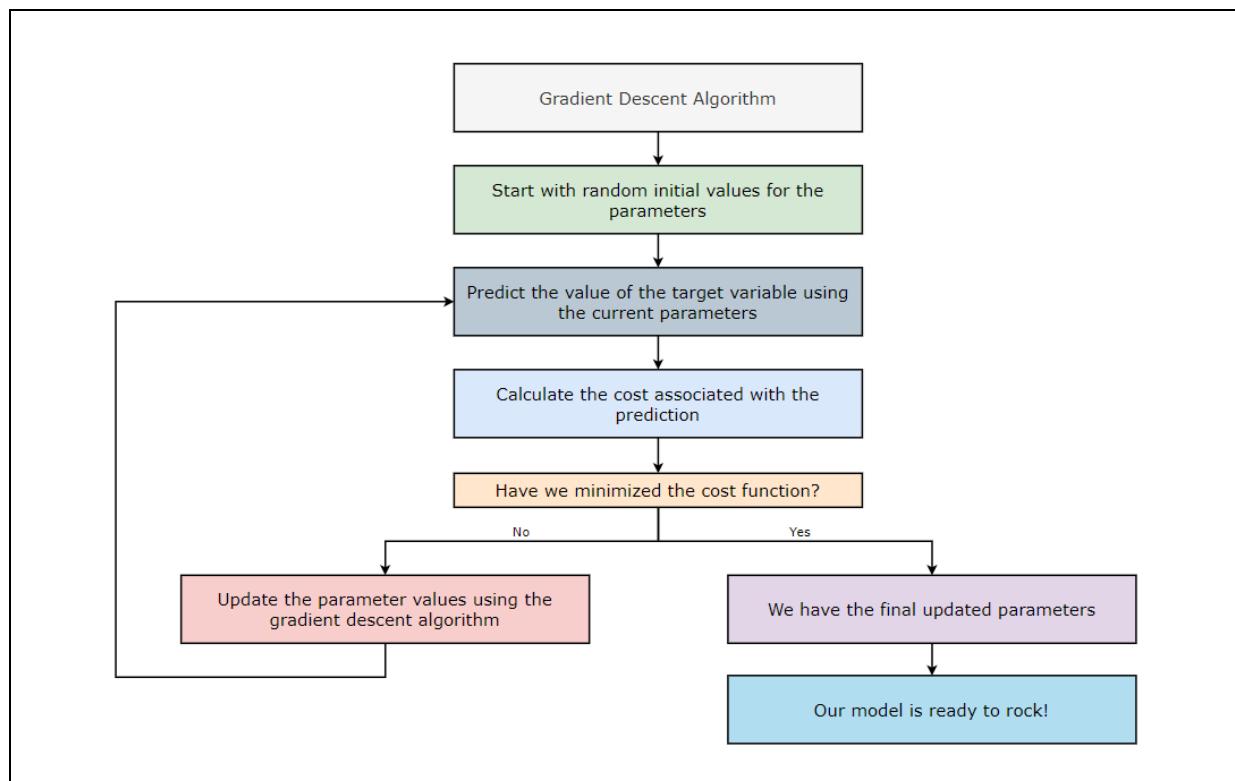
16. Data Science – Machine Learning – Gradient Descent

1. Gradient Descent

- ✓ Gradient descent is an optimization algorithm
- ✓ This algorithm find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

2. How Gradient Descent works

- ✓ 1. Start with random initial values for the parameters.
- ✓ 2. Predict the value of the target variable using the current parameters.
- ✓ 3. Calculate the cost associated with the prediction.
- ✓ 4. Is cost minimized?
 - If yes, then go to step no - 6.
 - If no, then go to step no - 5.
- ✓ 5. Update the parameter values using the gradient descent algorithm and return to step no - 2
- ✓ 6. We have our final updated parameters.
- ✓ 7. Our model is ready.

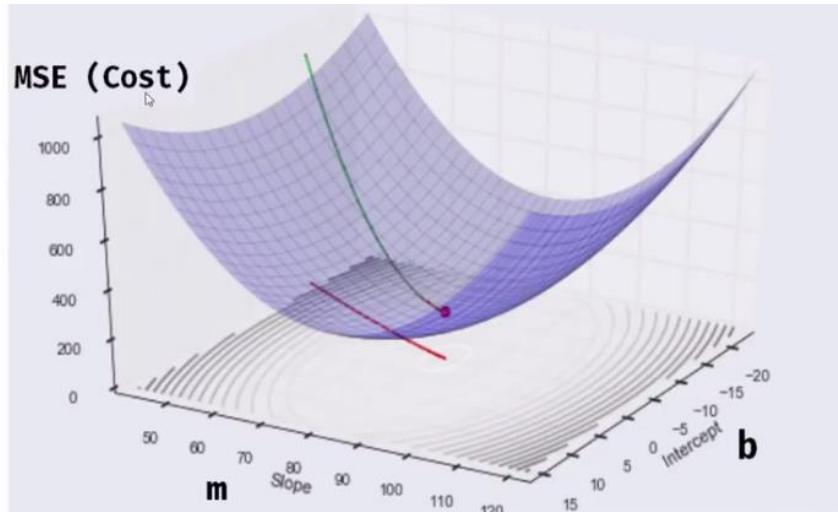


3. Convergence

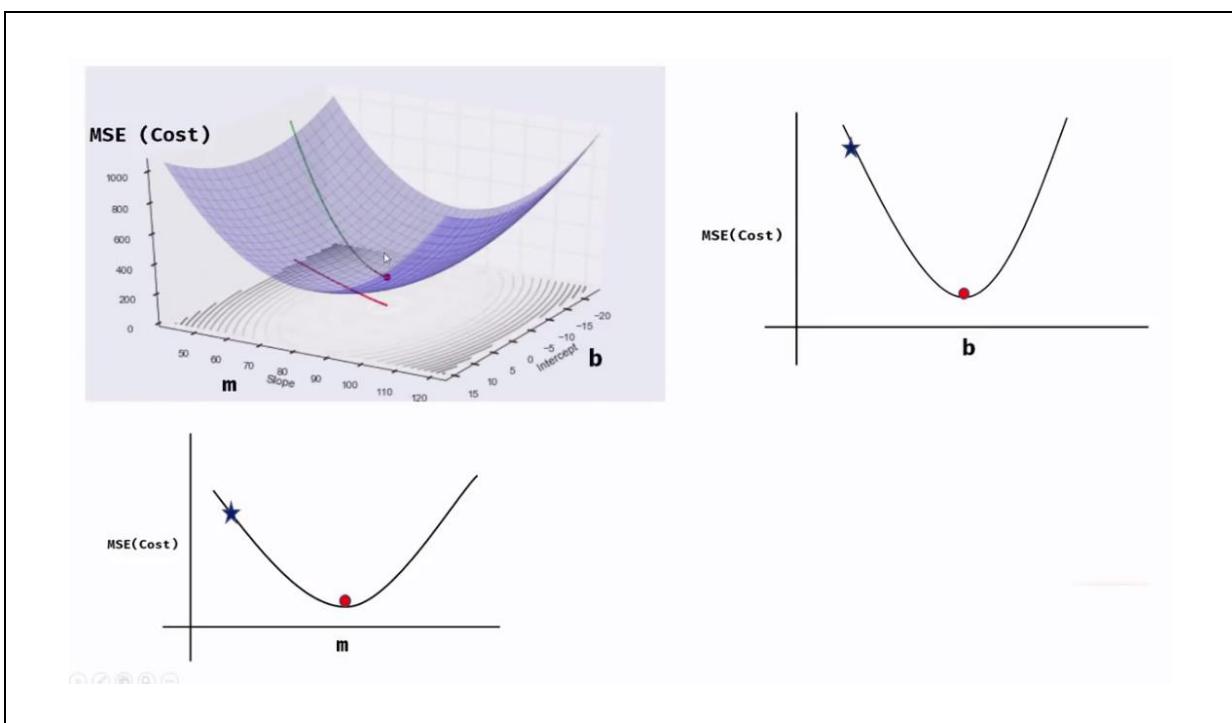
- ✓ Reaching a point in which gradient descent makes very small changes in your objective function is called convergence.
- ✓ It doesn't mean that, it has reached the optimal result but it is somewhat near to that.

4. Process behind the gradient

- ✓ Gradient descent is the algorithm that finds best fit line for given training data set.
- ✓ If we calculate for every value of m and b we will get **MSE**
- ✓ If we plot m and b against MSE then we will get below image

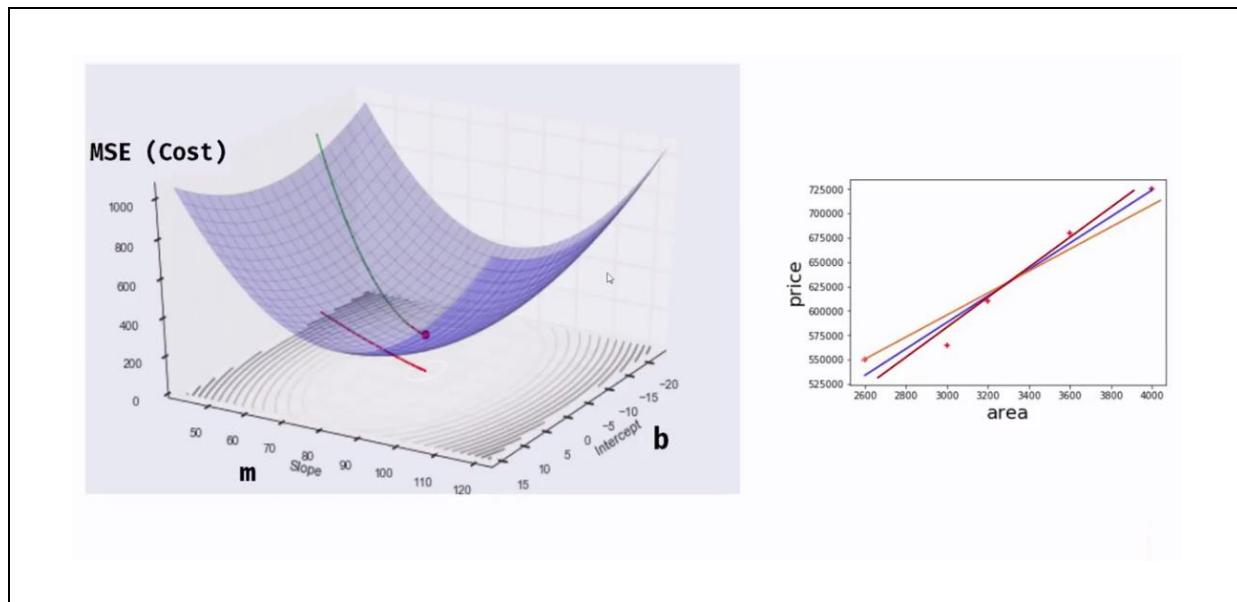


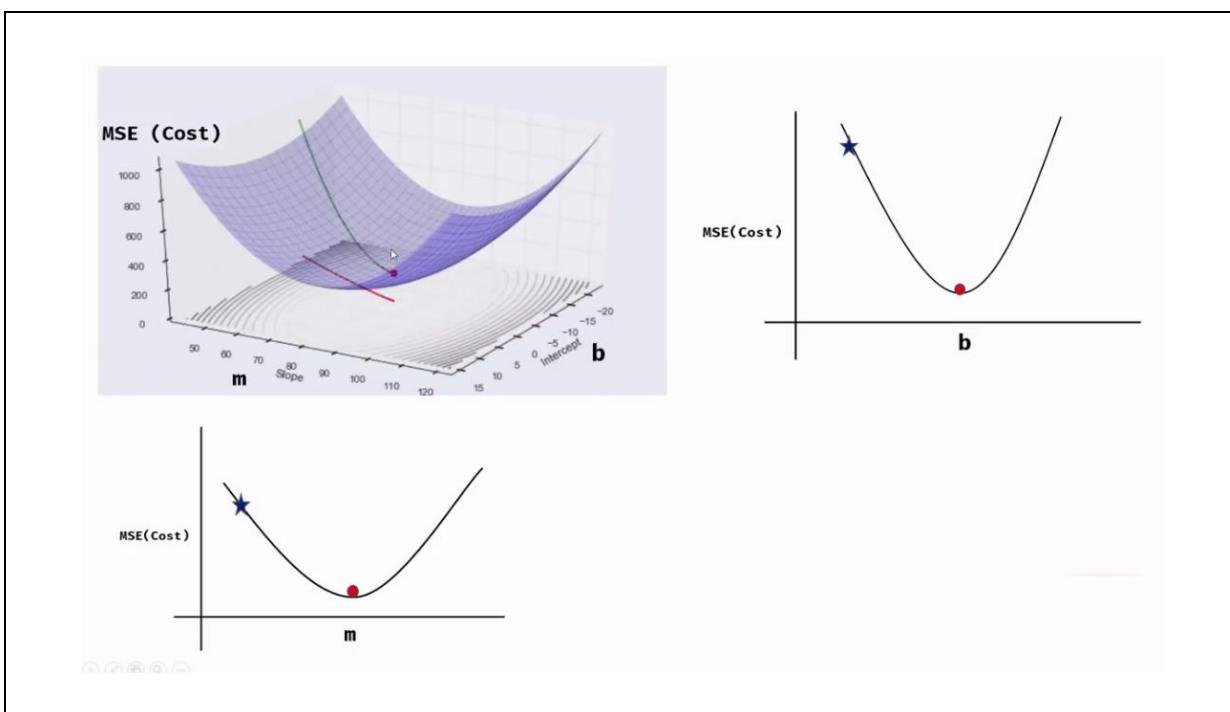
- ✓ Values
 - MSE values 0, 200, 400, 600, 800, 1000 etc
 - m values 50, 60, 70, 80, 90, 100, 110, 120 etc
 - b values 15, 10, 5, 0, -5, -10, -15 etc
- ✓ Here we will get an image which is like a bowl.
- ✓ We need to start some value for m and b , usually let's start from **zero** value



5. Steps

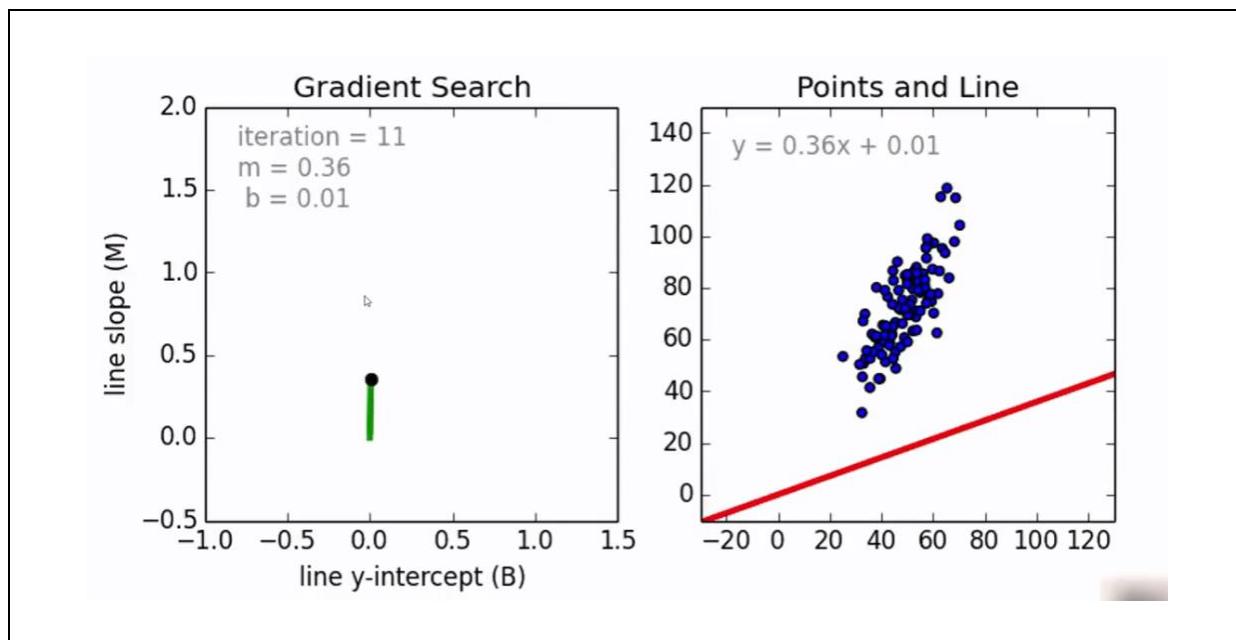
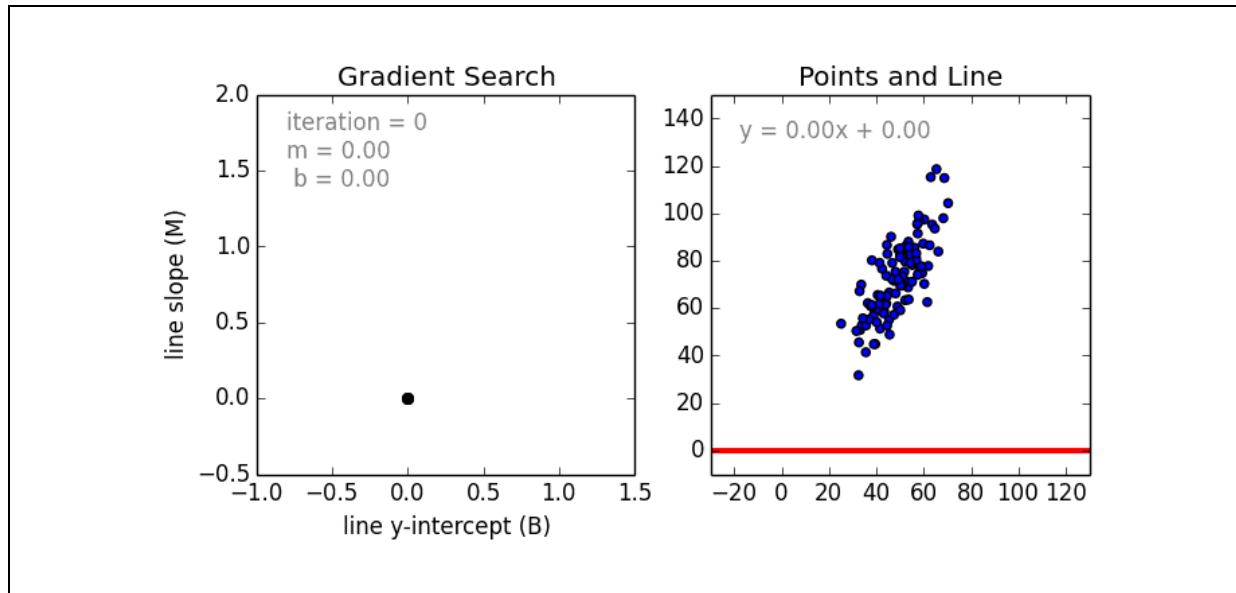
- ✓ Initially lets starts the point with $m = 0$ and $b = 0$
- ✓ Assuming that according to the diagram mse value is 1000 at m , b equals to zero
- ✓ Now just tune the values of m and b with some amount
- ✓ Now check mse value and the error will get reduce
- ✓ So, we need to keep on doing this process until to reach error level to minima(minimum)
- ✓ Once it reach the minima then we got our answer
- ✓ Here we need to use that m and b values, in the prediction function

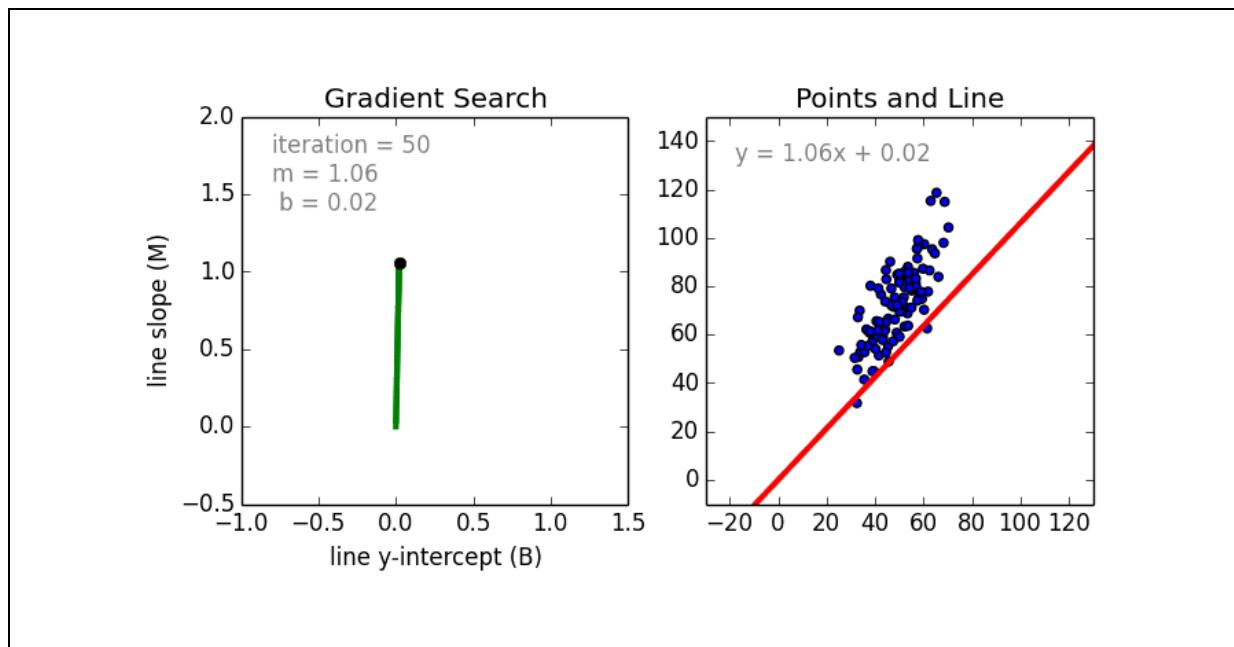
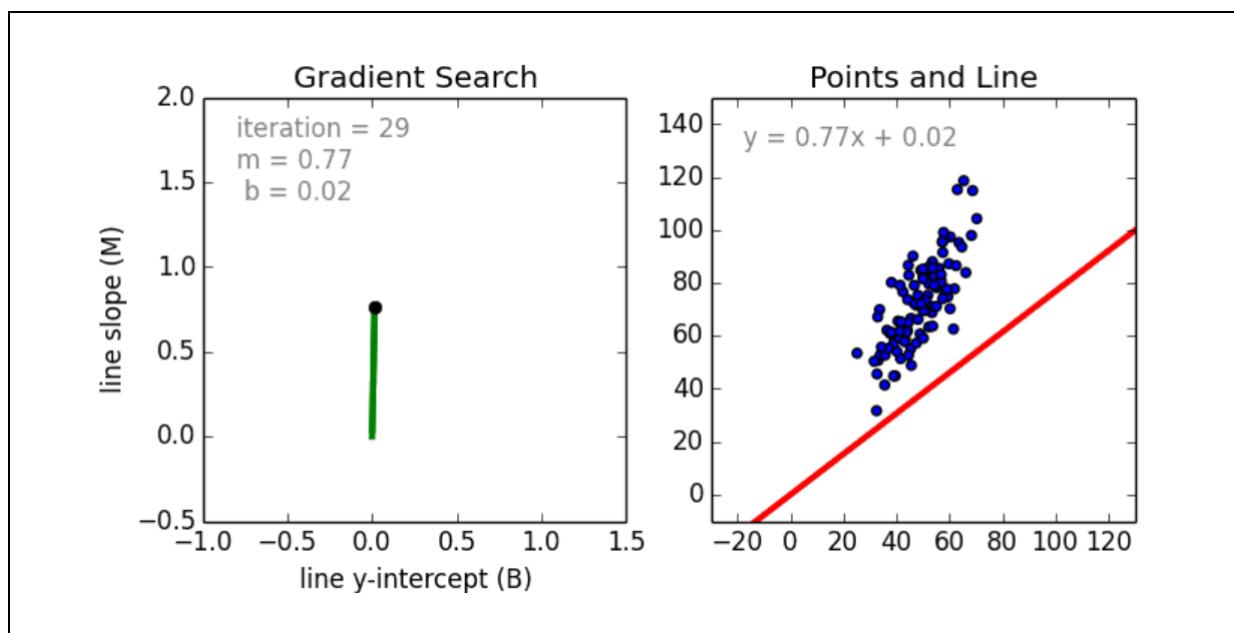


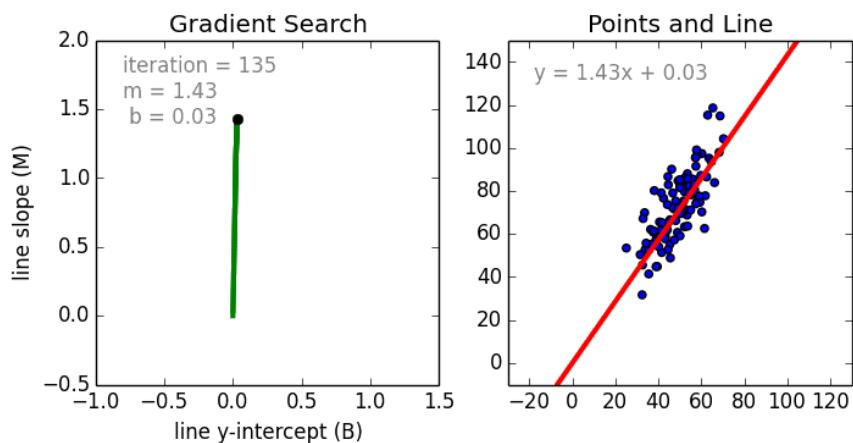
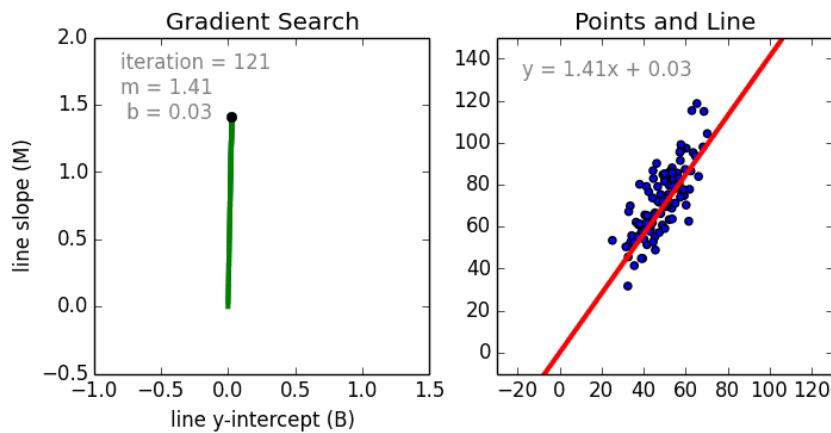


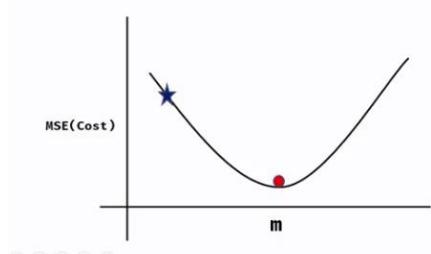
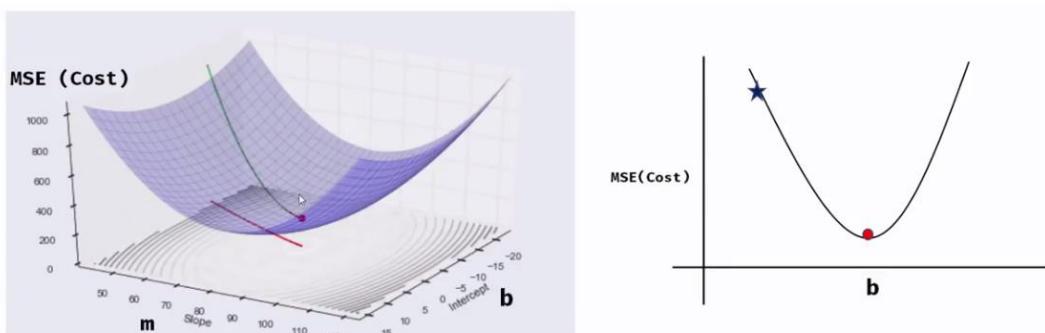
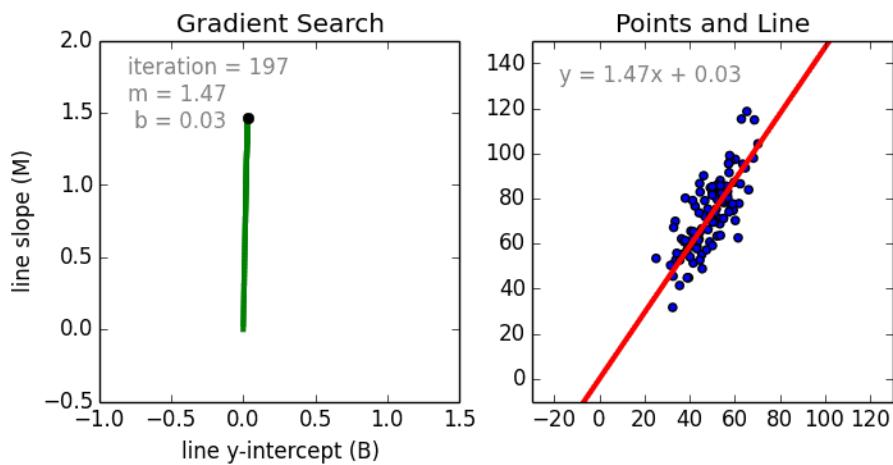
6. Follow below images

- ✓ Just try to understand below images for better understanding



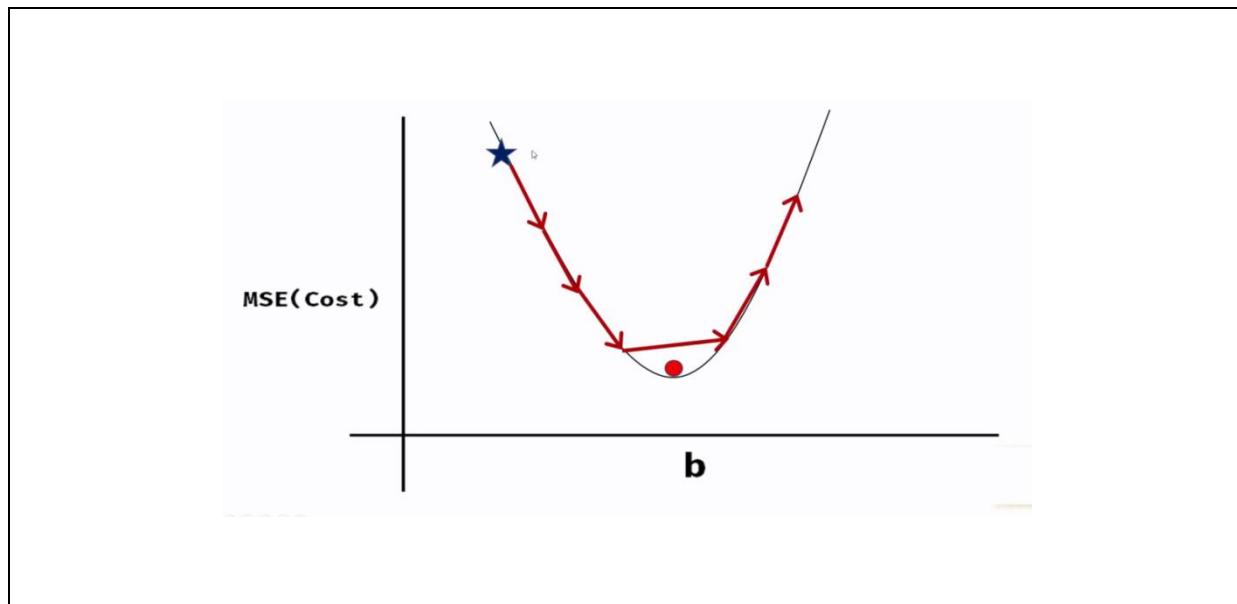






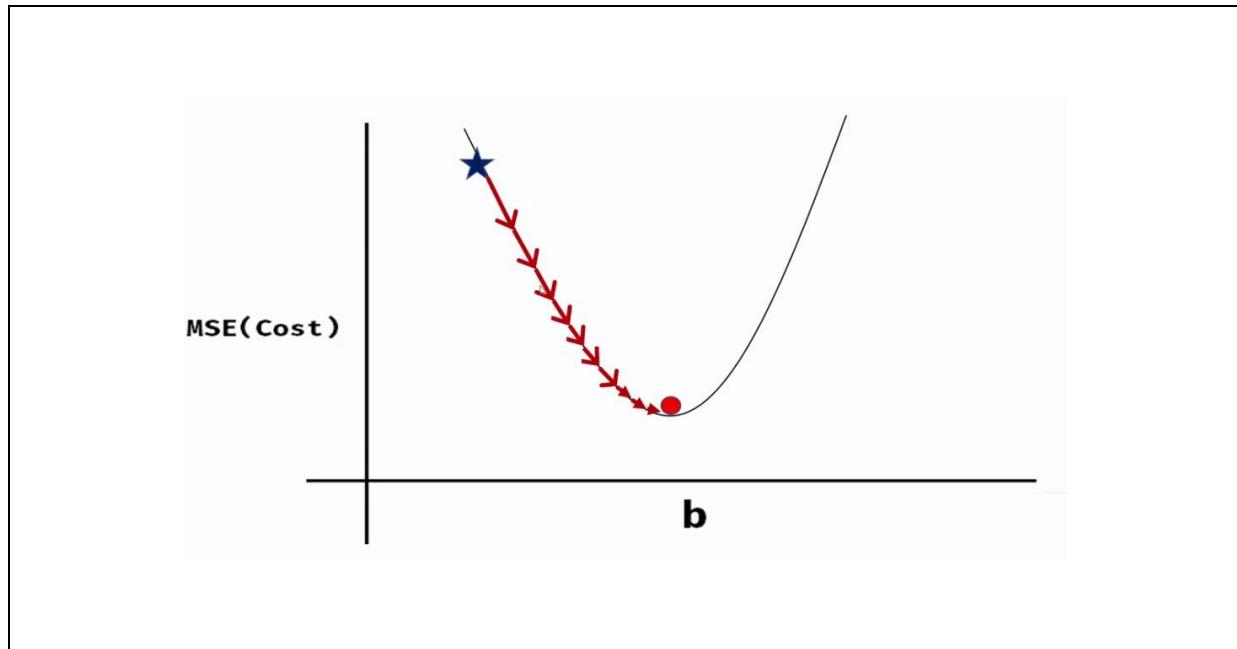
7. Fixed steps

- ✓ If we follow the fixed steps then we will get the below image
- ✓ By following this approach we may miss the global minima value.
- ✓ So, this approach is not going to work in well

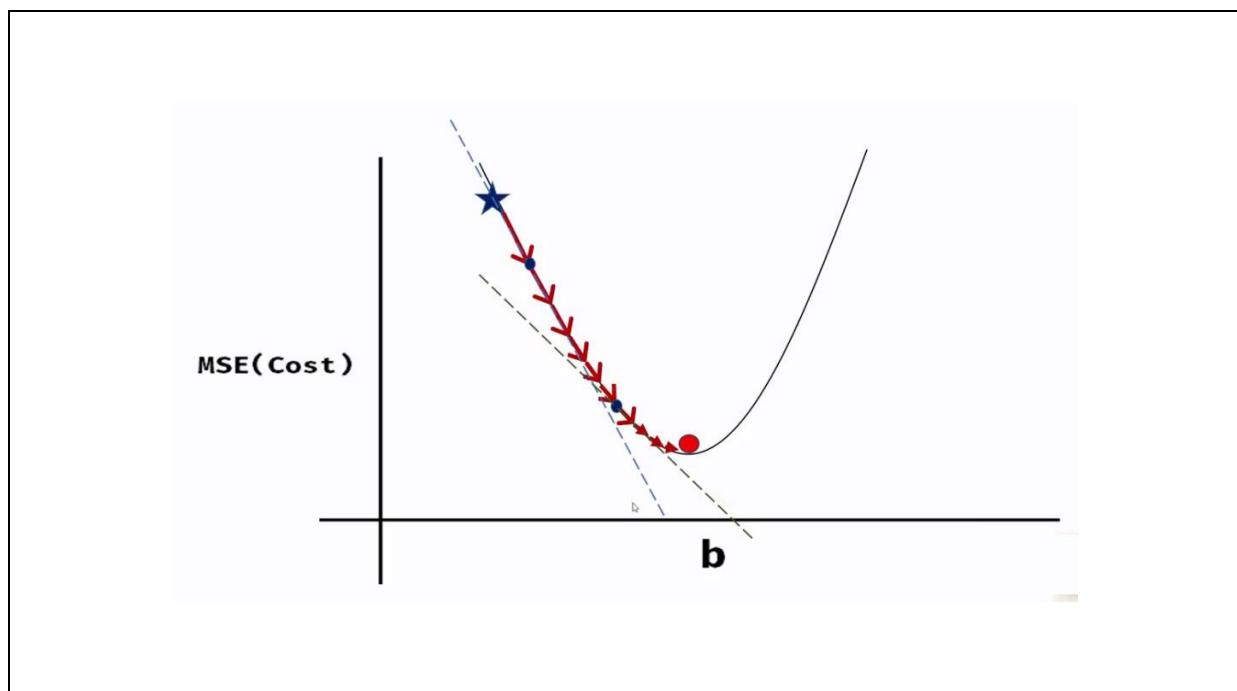


8. Learning rate: Reaching minimum error

- ✓ This approach seems to be good to catch minima value.
- ✓ The learning rate is a tuning parameter in an optimization algorithm.
- ✓ This determines the step size at each iteration while moving toward a minimum of a loss function.

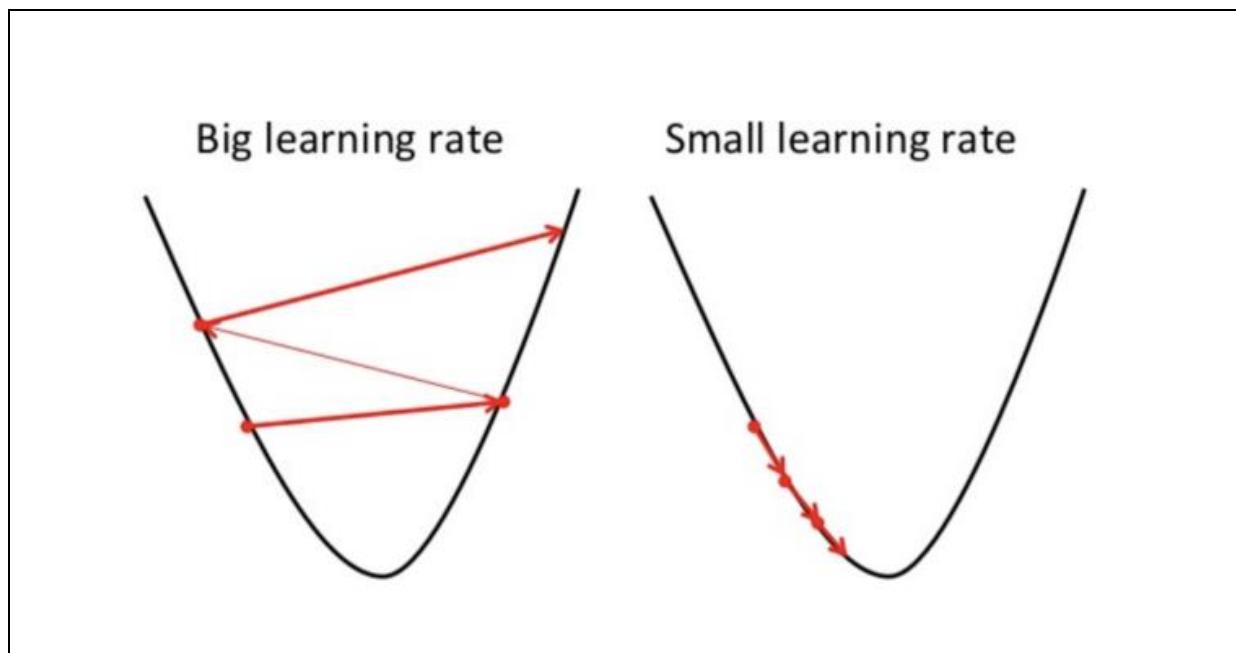


- ✓ At each point if we calculate the slope then we can reach the minimum value



9. Small and large learning rates

- ✓ A small leaning rate may lead to the model to take some time to learn
- ✓ A large learning rate will make the model converge as our pointer will shoot and we'll not be able to get to minima.



18. Data Science – Machine Learning – Logistic Regression

Contents

1. Logistic Regression	2
2. Types of logistic regression	2
3. Binary classification.....	2
4. Multiclass classification	2
5. Data set.....	3
6. Problem statement	3
7. Logistic function or Sigmoid.....	4

18. Data Science – Machine Learning – Logistic Regression

1. Logistic Regression

- ✓ Logistic regression comes under supervised Learning.
- ✓ It is a technique that is used to solve for classification problems.
- ✓ It is used for predicting the categorical dependent variable using a given set of independent variables.
- ✓ Examples
 - Email spam or not
 - Customer will buy product or not

2. Types of logistic regression

- ✓ Binary classification
 - This is having two classes
- ✓ Multiclass classification
 - This is having more than two classes

3. Binary classification

- ✓ In binary classification, there can be only two possible types of the dependent variables, such as,
 - 0 or 1
 - Pass or Fail
 - Yes or No etc.

4. Multiclass classification

- ✓ In multiclass classification, there can be 3 or more possible unordered types of the dependent variable, such as,
 - Ok, good, best
 - Cat, dot, sheep etc

5. Data set

- ✓ Its insurance dataset
 - ZERO means didn't buy the insurance
 - ONE means will buy the insurance
- ✓ We can understand one pattern here like, young people not buying the insurance
- ✓ Whereas person age increasing then that person more likely to buy the insurance

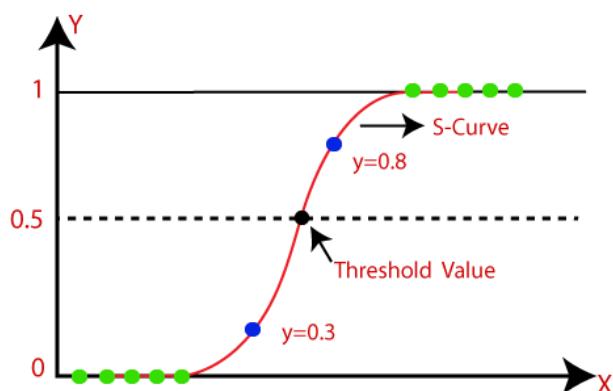
6. Problem statement

- ✓ Based on the age, we wanted to predict for persons will chose insurance or not.

age	Insurance status
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1

7. Logistic function or Sigmoid

- ✓ The logistic function, also called the sigmoid function.
- ✓ It maps any real value into another value within a range of 0 and 1.
- ✓ The value of the logistic regression must be between 0 and 1.
- ✓ In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1.



Formula

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

e = Euler's number ~ 2.71828

Sigmoid function converts input into range 0 to 1

Program Name Loading insurance dataset
demo1.py

```
import pandas as pd

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

print(df.head(10))
```

Output

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1

Program Name Plotting the dataset
demo2.py

```
import pandas as pd
from matplotlib import pyplot as plt

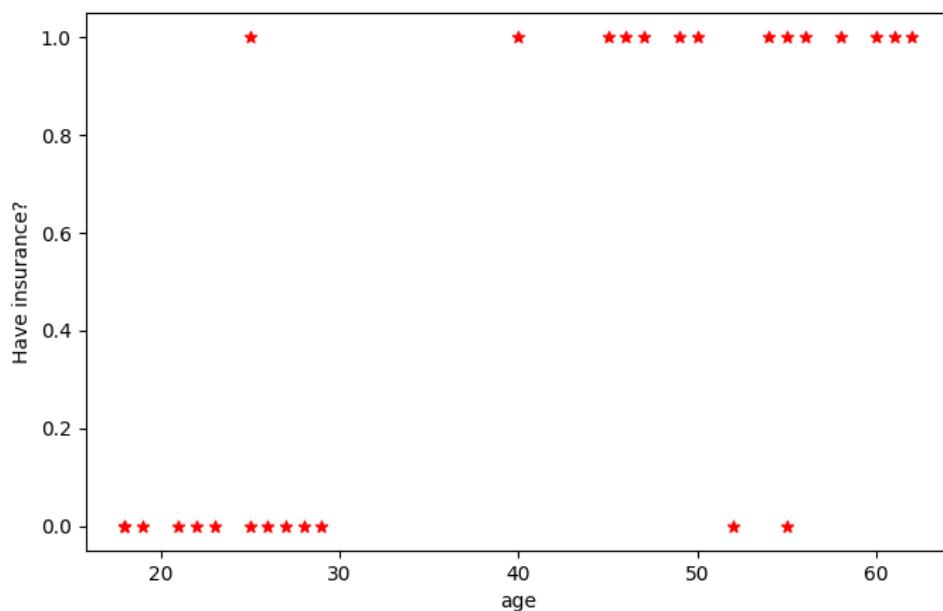
# Loading the dataset
df = pd.read_csv("insurance_data.csv")

# plotting the data
plt.scatter(df.age, df.bought_insurance, marker = '*', color = 'red')

plt.xlabel('age')
plt.ylabel('Have insurance?')

plt.show()
```

Output



Program Name Splitting the dataset
 demo3.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Loading the dataset
df = pd.read_csv("insurance_data.csv")
X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state=52)

print("X_train", '\n')
print(X_train, '\n')
```

Output

```
X_train  
  
      age  
14    49  
4     46  
12    27  
2     47  
8     62  
6     55  
19    18  
26    23  
24    50  
20    21  
15    55  
16    25  
1     25  
17    58  
3     52  
25    54  
10    18  
5     56  
0     22  
22    40  
23    45  
13    29  
11    28  
21    26  
  
X_test  
  
      age  
7     60  
9     61  
18    19
```

Program Name Splitting the dataset
 demo4.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 52)

print("y_train", '\n')
print(y_train,'n')
```

Output

```
y_train  
14      1  
4       1  
12      0  
2       1  
8       1  
6       0  
19      0  
26      0  
24      1  
20      0  
15      1  
16      1  
1       0  
17      1  
3       0  
25      1  
10      0  
5       1  
0       0  
22      1  
23      1  
13      0  
11      0  
21      0  
Name: bought_insurance, dtype: int64  
  
y_test  
7      1  
9      1  
18     0  
Name: bought_insurance, dtype: int64
```

Program Name Training the model
demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

print("Model got trained")
```

Output

Model got trained

Program Name

Prediction with single value

demo6.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction
print(model.predict([[50]]))
print(model.predict([[25]]))
```

Output

```
[1]
[0]
```

Program Name Prediction the result
demo7.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction
y_predicted = model.predict(X_test)

print("X_test data is \n")
print(X_test, '\n')

print("Prediction for X_test data \n")
print(y_predicted)
```

Output

```
X_test data is

      age
7      60
9      61
18     19

Prediction for X_test data

[1 1 0]
```

Prediction

- ✓ The one who has 19 years age he will not buy the insurance
- ✓ Both who has 60 and 61 years age persons will buy the insurance

Program Name

Prediction score
demo8.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df=pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction score
print(model.score(X_test, y_test))
```

Output

1.0

Program Name Prediction probability demo9.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

print("X_test")
print(X_test)

print()

# Prediction probability
print(model.predict_proba(X_test))
```

Output

```
X_test
    age
7      60
9      61
18     19

[[0.06266647  0.93733353]
 [0.05553734  0.94446266]
 [0.92804604  0.07195396]]
```

19. Data Science – ML – Logistic Regression – Multi class classification

Contents

1. Logistic Regression	2
2. Types of logistic regression	2
3. Binary classification examples	2
4. Multiclass classification examples	2
5. Practical example	3
6. Problem	4
7. Load dataset	5

19. Data Science – ML – Logistic Regression – Multi class classification

1. Logistic Regression

- ✓ Logistic regression comes under supervised Learning.
- ✓ It is a technique that is used to solve for classification problems.
- ✓ It is used for predicting the categorical dependent variable using a given set of independent variables.

2. Types of logistic regression

- ✓ Binary classification
 - This is having two classes
- ✓ Multiclass classification
 - This is having more than two classes

3. Binary classification examples

- ✓ In binary classification, there can be only two possible types of the dependent variables, such as,
 - 0 or 1
 - Pass or Fail
 - Yes or No etc.

4. Multiclass classification examples

- ✓ In multiclass classification, there can be 3 or more possible unordered types of the dependent variable, such as,
 - Ok, good, best
 - Cat, dot, sheep etc

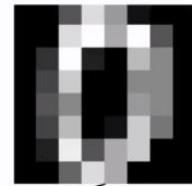
5. Practical example

- ✓ Identify hand written digits
 - If image having ZERO number then output should be Zero.
 - If image having ONE number then output should be ONE.
 - If image having TWO number then output should be TWO.
 -
 - If image having NINE number then output should be NINE.

6. Problem

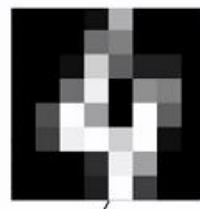
- ✓ Recognising the number.

Identify hand written digits recognition



0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Identify hand written digits recognition



0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

7. Load dataset

- ✓ Sklearn library comes up with built-in datasets.
- ✓ One of the dataset names is called as digits dataset.

Program Name Loading digits dataset
demo1.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(len(digits.data))
```

Output
1797

Program Name Loading digits dataset and checking attributes
demo2.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(dir(digits))
```

Output
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target',
'target_names']

Program Name Loading digits dataset and accessing DESCR attribute
demo3.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.DESCR)
```

Output

```
C:\Users\Nireekshan\Desktop\PROGRAMS>py test.py
.. _digits_dataset:

Optical recognition of handwritten digits dataset
-----
**Data Set Characteristics:**

:Number of Instances: 1797
:Number of Attributes: 64
:Attribute Information: 8x8 image of integer pixels in the range 0..16.
:Missing Attribute Values: None
:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
:Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.
```

Program Name Loading digits dataset and accessing data attribute
demo4.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.data)
```

Output

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
```

Program Name Loading digits dataset, checking the first value
demo5.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.data[0])
```

Output

```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
 0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
 0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
```

Program Name Loading digits dataset, checking the second value
demo6.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.data[1])
```

Output

```
[ 0.  0.  0. 12. 13.  5.  0.  0.  0.  0. 11. 16.  9.  0.  0.  0.  0.
 3. 15. 16.  6.  0.  0.  0.  7. 15. 16. 16.  2.  0.  0.  0.  0.  1. 16.
16.  3.  0.  0.  0.  1. 16. 16.  6.  0.  0.  0.  0.  0.  1. 16. 16.  6.
 0.  0.  0.  0. 11. 16. 10.  0.  0.]
```

Program Name Loading digits dataset and accessing images attribute
demo7.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.images)
```

Output

```
[[[ 0.  0.  5. ...  1.  0.  0.]
 [ 0.  0.  13. ... 15.  5.  0.]
 [ 0.  3.  15. ... 11.  8.  0.]
 ...
 [ 0.  4.  11. ... 12.  7.  0.]
 [ 0.  2.  14. ... 12.  0.  0.]
 [ 0.  0.  6. ...  0.  0.  0.]]]

[[ 0.  0.  0. ...  5.  0.  0.]
 [ 0.  0.  0. ...  9.  0.  0.]
 [ 0.  0.  3. ...  6.  0.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]]]

[[ 0.  0.  0. ... 12.  0.  0.]
 [ 0.  0.  3. ... 14.  0.  0.]
 [ 0.  0.  8. ... 16.  0.  0.]
 ...]
```

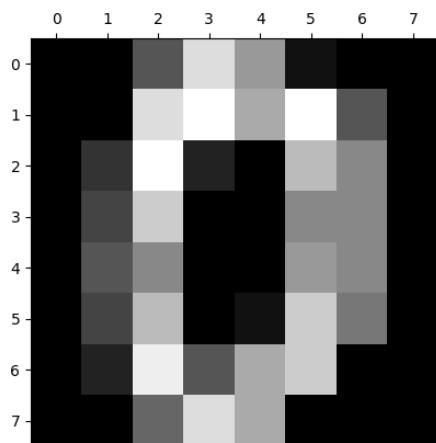
Program Name Loading digits dataset, displaying image
demo8.py

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Loading the dataset
digits = load_digits()

plt.matshow(digits.images[0])
plt.gray()
plt.show()
```

Output



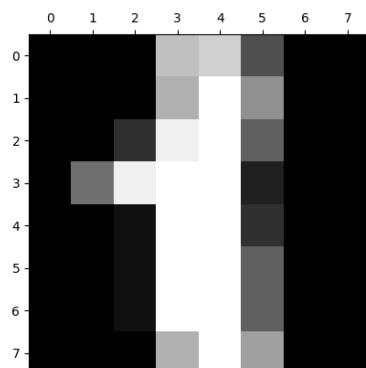
Program Name Loading digits dataset, displaying image
demo9.py

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Loading the dataset
digits = load_digits()

plt.matshow(digits.images[1])
plt.gray()
plt.show()
```

Output



Program Name Loading digits dataset, displaying image
demo10.py

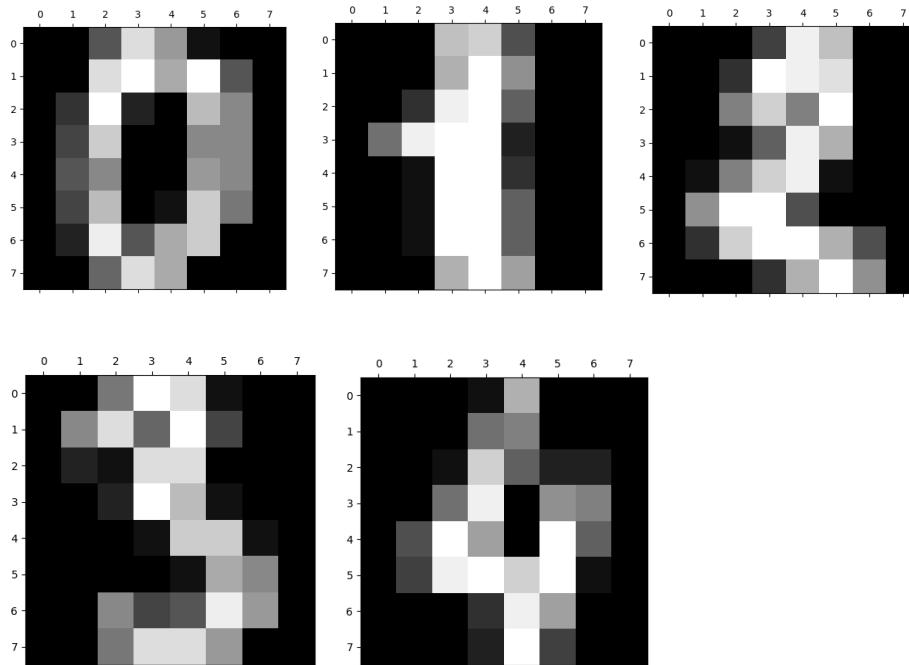
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Loading the dataset
digits = load_digits()

plt.gray()

for i in range(5):
    plt.matshow(digits.images[i])
    plt.show()
```

Output



Program Name Loading digits dataset and accessing target attribute
demo11.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.target)
```

Output

```
[0 1 2 ... 8 9 8]
```

Program Name Loading digits dataset and accessing target attribute
demo12.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.target[0:5])
```

Output

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

Program Name Splitting the dataset
demo13.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

print("Splitting the dataset")
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size = 0.2)
```

Output

```
Splitting the dataset
```

Program Name

Model creation
demo14.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

print("Model creation")
model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)
```

Output

Model creation

Note

- ✓ lbfgs stand for: "Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm".
- ✓ It is one of the solvers' algorithms provided by Scikit-Learn Library.

Program Name Model score
demo15.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

Output

0.9611111111111111

Program Name Model prediction
demo16.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size = 0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict([digits.data[6]]))
```

Output

[6]

Program Name Model prediction
demo17.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict([digits.data[9]]))
```

Output

[9]

Program Name Model prediction
demo18.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict([digits.data[20]]))
```

Output

[0]

Program Name Model prediction
demo19.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict(digits.data[0:5]))
```

Output

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

20. Data Science – Machine Learning – Decision Tree

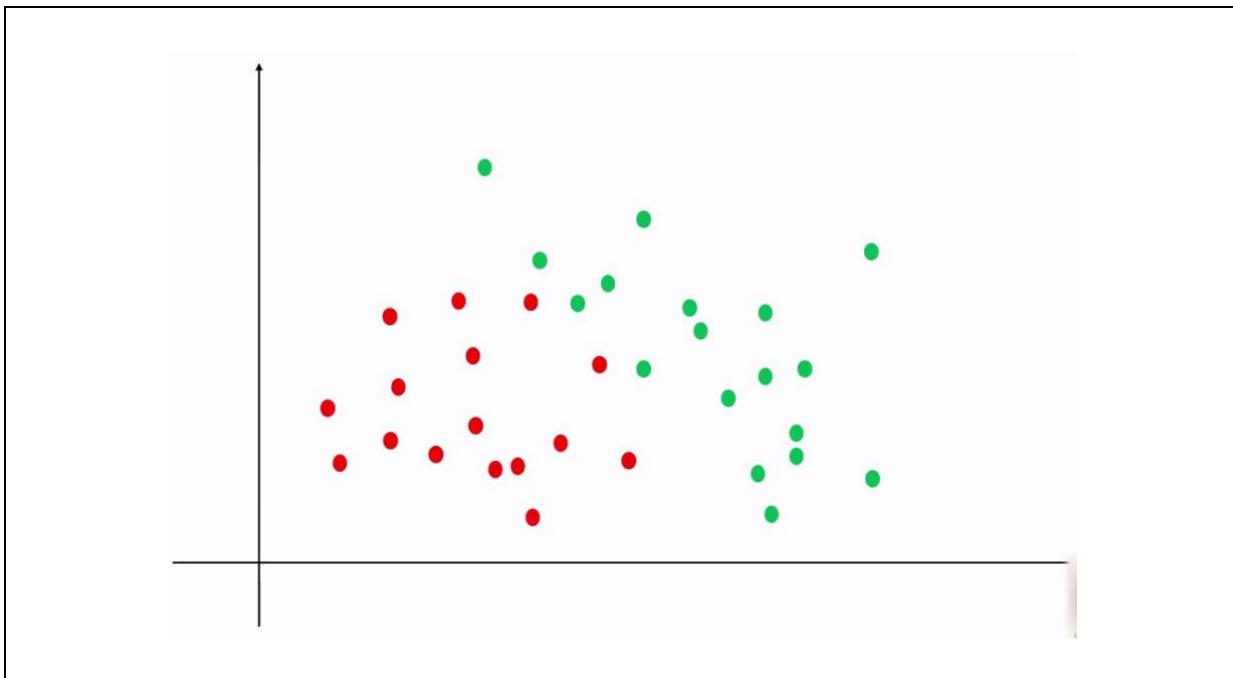
Contents

1. If dataset is like below example1	2
2. If dataset is as below example2.....	3
3. Decision Tree Classification Algorithm	4
3.1. Decision Node	4
3.2. Leaf nodes.....	4
4. CART algorithm.....	5
5. Why use Decision Trees?.....	5
6. Decision Tree Terminologies.....	6
6.1. Root Node	6
6.2. Leaf Node	6
6.3. Splitting	6
6.4. Branch/Sub Tree	6
6.5. Pruning.....	6
6.6. Parent/Child node.....	6
7. How does the Decision Tree algorithm Work?.....	7
7.1. Example.....	8
8. Problem	9
9. DecisionTreeClassifier class	17
9.1. fit(X_train, y_train) method.....	18
9.2. predict(p) method.....	20

20. Data Science – Machine Learning – Decision Tree

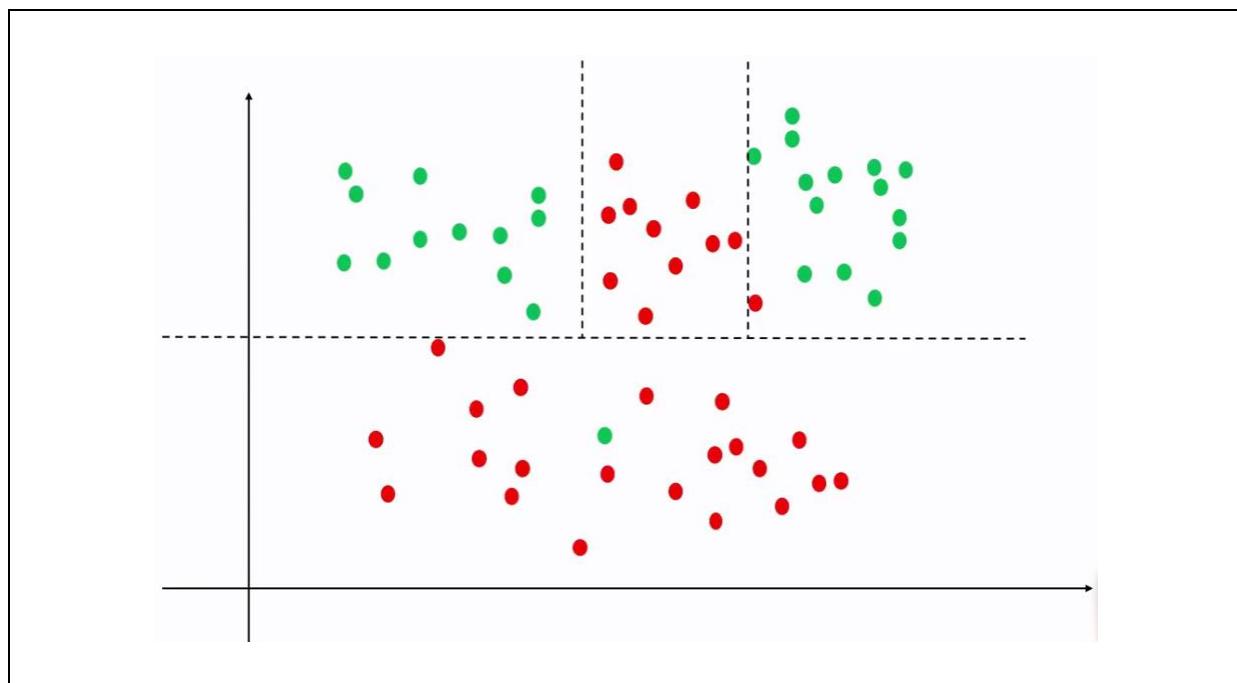
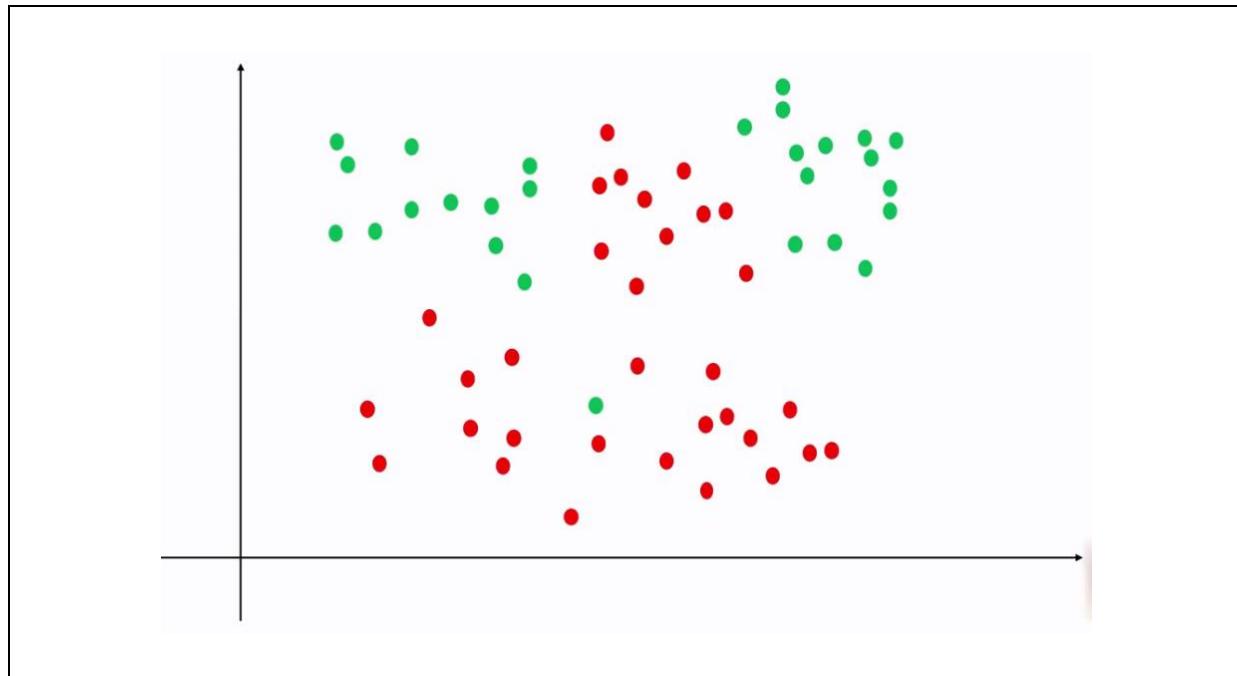
1. If dataset is like below example1

- ✓ We can draw a good separation line by using regression algorithm



2. If dataset is as below example2

- ✓ In the given below scenario, dataset is very complex.
- ✓ Then a single line may not fit for the given dataset
- ✓ Here decision tree comes into the picture



3. Decision Tree Classification Algorithm

- ✓ Decision Tree is a supervised learning technique.
- ✓ It can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- ✓ It is called a decision tree because similar to a tree structure like it starts with the root node and expands on further branches and constructs a tree-like structure.
- ✓ In a Decision tree there are two nodes,
 - Decision Node
 - Leaf Node.

3.1. Decision Node

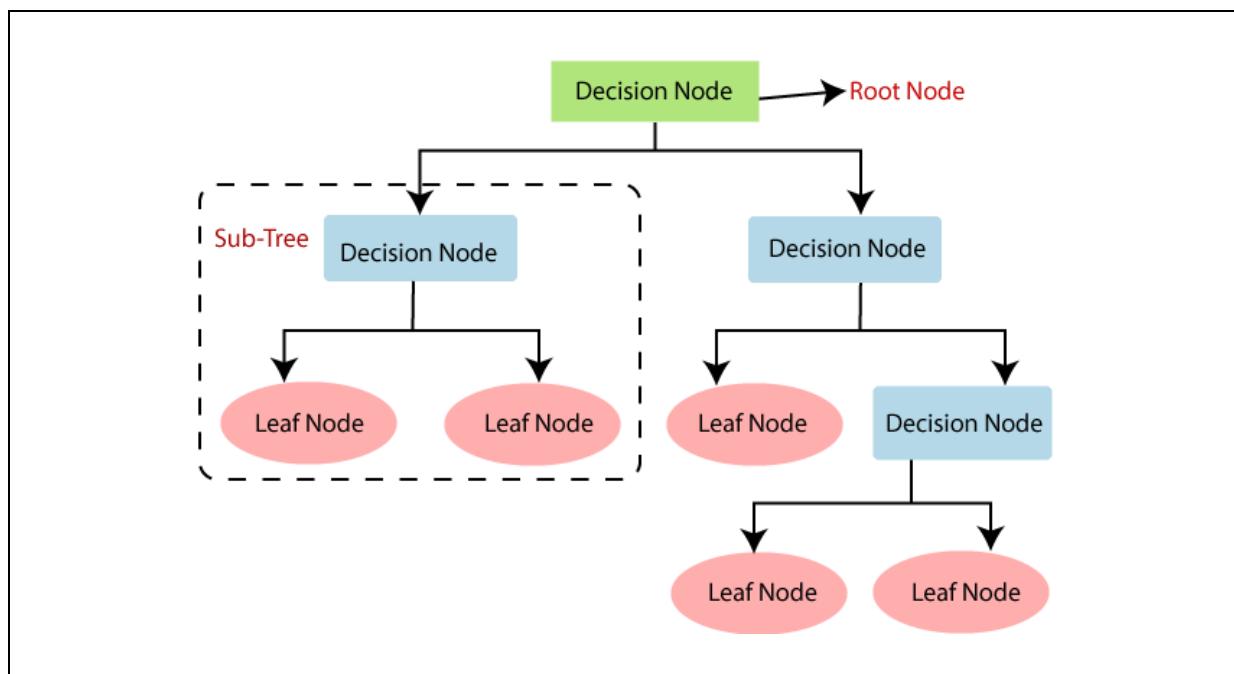
- ✓ Decision nodes are used to make any decision and have multiple branches.

3.2. Leaf nodes

- ✓ Leaf nodes are the output of those decisions and do not contain any further branches.

4. CART algorithm

- ✓ In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- ✓ A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



5. Why use Decision Trees?

- ✓ Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- ✓ The logic behind the decision tree can be easily understood because it shows a tree-like structure.

6. Decision Tree Terminologies

6.1. Root Node

- ✓ Root node is from where the decision tree starts.
- ✓ It represents the entire dataset, which further gets divided into two or more homogeneous sets.

6.2. Leaf Node

- ✓ Leaf nodes are the final output node.
- ✓ The tree cannot be segregated further after getting a leaf node.

6.3. Splitting

- ✓ Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

6.4. Branch/Sub Tree

- ✓ A tree formed by splitting the tree.

6.5. Pruning

- ✓ Pruning is the process of removing the unwanted branches from the tree.

6.6. Parent/Child node

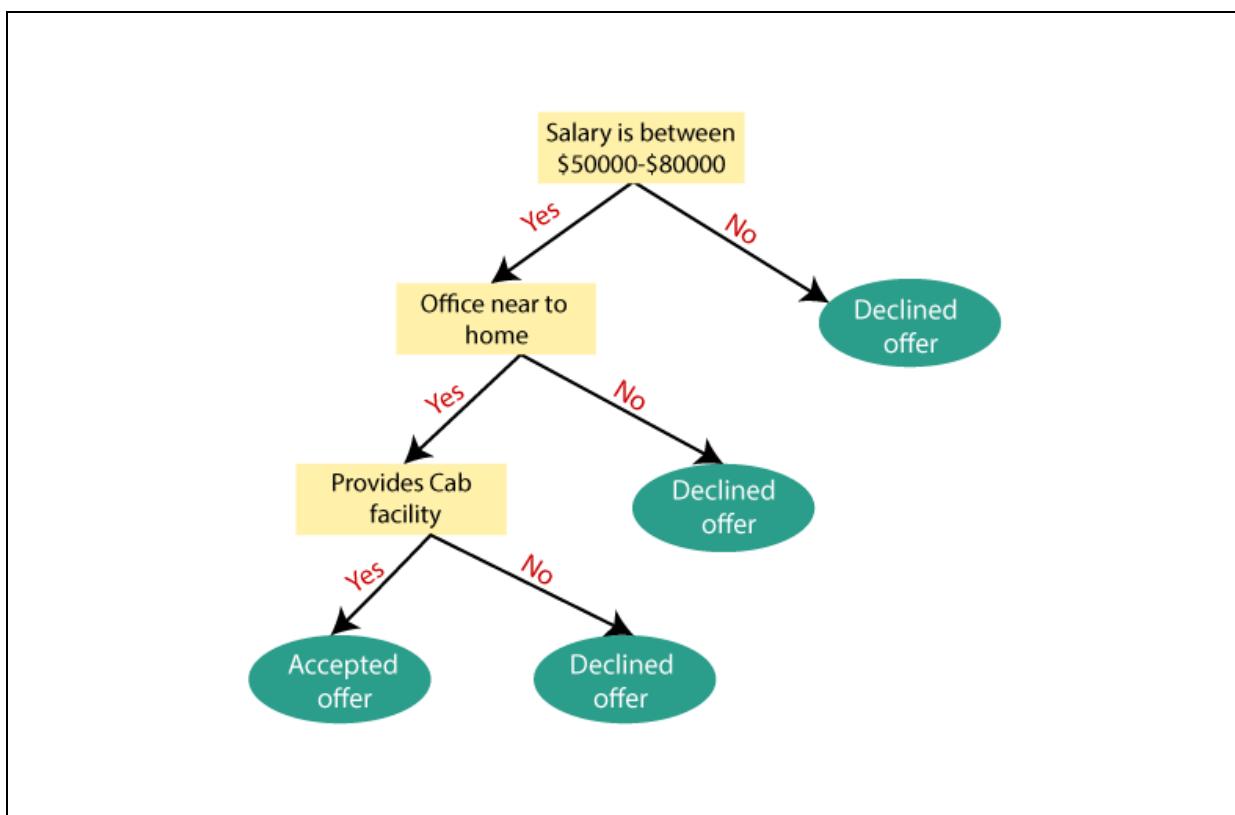
- ✓ The root node of the tree is called the parent node, and other nodes are called the child nodes.

7. How does the Decision Tree algorithm Work?

- ✓ Step-1: Begin the tree with the root node, which contains the complete dataset.
- ✓ Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- ✓ Step-3: Divide the dataset into subsets that contains possible values for the best attributes.
- ✓ Step-4: Generate the decision tree node, which contains the best attribute.
- ✓ Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

7.1. Example

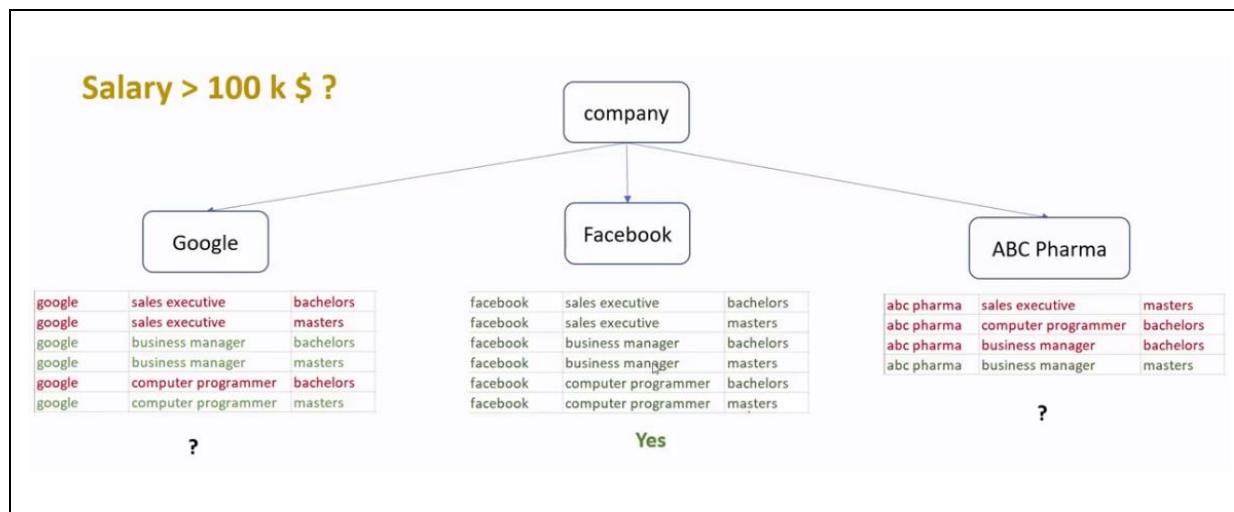
- ✓ Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not.
- ✓ So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM).
- ✓ The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels.
- ✓ The next decision node further gets split into one decision node (Cab facility) and one leaf node.
- ✓ Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer).



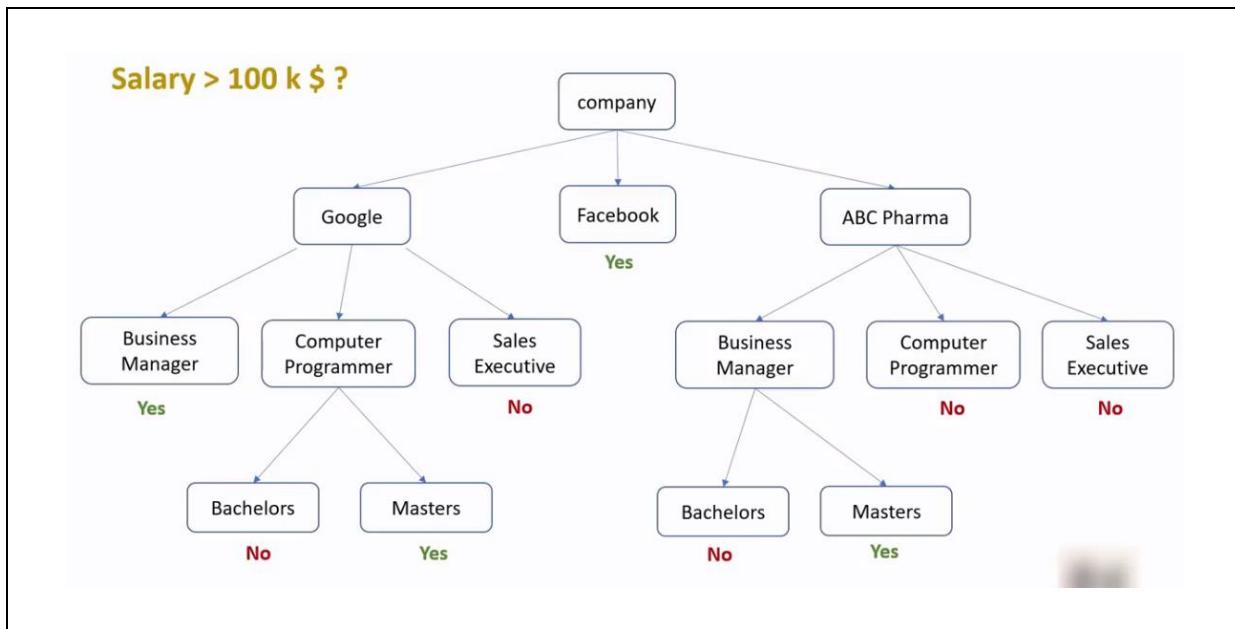
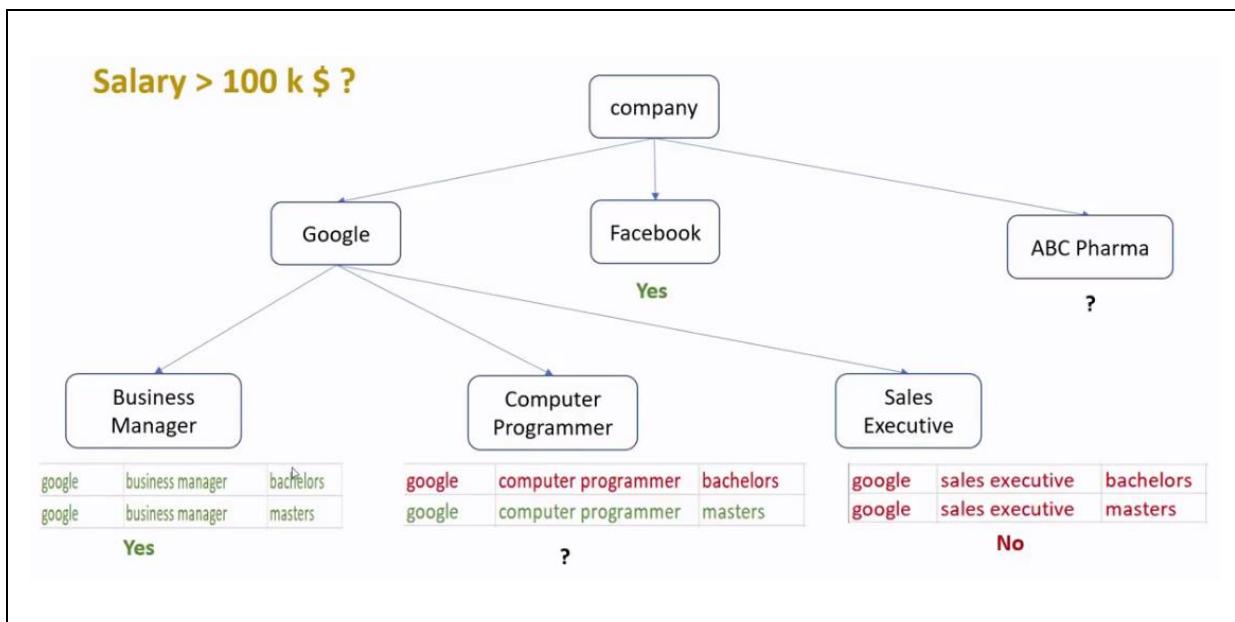
Data Science – Machine Learning – Decision Tree

8. Problem

Company	Job	Degree	Salary_more_then_100k
google	sales executive	bachelors	0
google	sales executive	masters	0
google	business manager	bachelors	1
google	business manager	masters	1
google	computer programmer	bachelors	0
google	computer programmer	masters	1
abc pharma	sales executive	masters	0
abc pharma	computer programmer	bachelors	0
abc pharma	business manager	bachelors	0
abc pharma	business manager	masters	1
facebook	sales executive	bachelors	1
facebook	sales executive	masters	1
facebook	business manager	bachelors	1
facebook	business manager	masters	1
facebook	computer programmer	bachelors	1
facebook	computer programmer	masters	1



Data Science – Machine Learning – Decision Tree



Program Name Loading dataset
demo1.py

```
import pandas as pd

df = pd.read_csv("salaries.csv")
print(df)
```

Output

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0
5	google	computer programmer	masters	1
6	abc pharma	sales executive	masters	0
7	abc pharma	computer programmer	bachelors	0
8	abc pharma	business manager	bachelors	0
9	abc pharma	business manager	masters	1
10	facebook	sales executive	bachelors	1
11	facebook	sales executive	masters	1
12	facebook	business manager	bachelors	1
13	facebook	business manager	masters	1
14	facebook	computer programmer	bachelors	1
15	facebook	computer programmer	masters	1

Program Name Preparing input and target
demo2.py

```
import pandas as pd

df = pd.read_csv("salaries.csv")

inputs = df.drop('salary_more_then_100k', axis = 'columns')
target = df['salary_more_then_100k']

print("Input")
print(inputs.head())

print("Target")
print(target.head())
```

Output

```
Input
   company           job      degree
0  google    sales executive bachelors
1  google    sales executive    masters
2  google  business manager bachelors
3  google  business manager    masters
4  google  computer programmer bachelors
Target
0    0
1    0
2    1
3    1
4    0
Name: salary_more_then_100k, dtype: int64
```

Program Name Transforming input
demo3.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")

inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])

print(inputs)
```

Output

	company	job	degree	company_n
0	google	sales executive	bachelors	2
1	google	sales executive	masters	2
2	google	business manager	bachelors	2
3	google	business manager	masters	2
4	google	computer programmer	bachelors	2
5	google	computer programmer	masters	2
6	abc pharma	sales executive	masters	0
7	abc pharma	computer programmer	bachelors	0
8	abc pharma	business manager	bachelors	0
9	abc pharma	business manager	masters	0
10	facebook	sales executive	bachelors	1
11	facebook	sales executive	masters	1
12	facebook	business manager	bachelors	1
13	facebook	business manager	masters	1
14	facebook	computer programmer	bachelors	1
15	facebook	computer programmer	masters	1

Program Name Transforming input
demo4.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")

inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])

print(inputs)
```

Output

	company	job	degree	company_n	job_n
0	google	sales executive	bachelors	2	2
1	google	sales executive	masters	2	2
2	google	business manager	bachelors	2	0
3	google	business manager	masters	2	0
4	google	computer programmer	bachelors	2	1
5	google	computer programmer	masters	2	1
6	abc pharma	sales executive	masters	0	2
7	abc pharma	computer programmer	bachelors	0	1
8	abc pharma	business manager	bachelors	0	0
9	abc pharma	business manager	masters	0	0
10	facebook	sales executive	bachelors	1	2
11	facebook	sales executive	masters	1	2
12	facebook	business manager	bachelors	1	0
13	facebook	business manager	masters	1	0
14	facebook	computer programmer	bachelors	1	1
15	facebook	computer programmer	masters	1	1

Program Name Transforming input
demo5.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

print(inputs)
```

Output

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

Program Name Transforming input
demo6.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print(inputs_n)
```

Output

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

9. DecisionTreeClassifier class

- ✓ **DecisionTreeClassifier** is predefined class in **sklearn.tree** package
- ✓ We need to import this class from **sklearn.tree** package
- ✓ Once we imported then we need to **create an object** to **DecisionTreeClassifier** class.

Program

Name Model creation

demo7.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

model = DecisionTreeClassifier()
print("DecisionTreeClassifier object created")
```

Output

DecisionTreeClassifier object created

9.1. fit(X_train, y_train) method

- ✓ fit(X_train, y_train) is predefined method in DecisionTreeClassifier class.
- ✓ We should access this method by using DecisionTreeClassifier object only.
- ✓ By using this method we need to train the model.

Program

Name Model creation

demo8.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print("Model got trained")
```

Output

Model got trained

Program Name Model score
demo9.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print("Model got trained")
model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print(model.score(inputs_n, target))
```

Output

1.0

9.2. predict(p) method

- ✓ predict(p) is predefined method in DecisionTreeClassifier class.
- ✓ We should access this method by using DecisionTreeClassifier object only.
- ✓ By using this method we can predict the results.

Program

Name

Model prediction

demo10.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print("Model got trained")
model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print(model.predict([[2, 1, 0]]))
```

Output

[0]

Program Name

Model prediction

demo11.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print("Model got trained")
model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print(model.predict([[2, 1, 1]]))
```

Output

[1]

21. Data Science – Machine Learning – Confusion Matrix

Contents

1. Introduction	2
2. Confusion Matrix	2
3. Name of the each combination	3
4. Table1	4
5. Scenario	5
5.1. True positive	6
5.2. True Negative.....	6
5.3. False Positive.....	7
5.4. False Negative	7
6. Type I and Type II errors	8
7. Confusion Matrix	9
8. Table2	9
9. Confusion matrix example	10
10. Performance Metrics	11
11. Accuracy	12
12. Accuracy can be misleading	13
13. Recall or Sensitivity or True Positive Rate	14
14. Specificity or True Negative Rate	15
15. Precision	16
16. F1 Score	18

21. Data Science – Machine Learning – Confusion Matrix

1. Introduction

- ✓ Confusion matrix is a method to explain the results of the classification model.

2. Confusion Matrix

- ✓ In binary classification, the outcomes are,
 - True
 - False
- ✓ To make it more clear let us consider an example of a binary classifier that scans the **MRI** images and **predicts whether a person has cancer or not**.
- ✓ The outcomes predicted by classifier and the actual outcomes can only have the following four combinations

Predicted	Actual
Cancer = Yes	Cancer = Yes
Cancer = Yes	Cancer = No
Cancer = No	Cancer = Yes
Cancer = No	Cancer = No

3. Name of the each combination

- ✓ Now let's understand clearly by comparing each other

Name	Predicted	Actual
True Positive	Cancer = Yes	Cancer = Yes
False Positive	Cancer = Yes	Cancer = No
False Negative	Cancer = No	Cancer = Yes
True Negative	Cancer = No	Cancer = No

4. Table1

Name	Description
1. True Positive	✓ The model predicts that the patient is having cancer (positive) and indeed he has cancer (true prediction).
2. False Positive	✓ The model predicts that the patient is having cancer (positive) but actually he is not having cancer (false prediction)
3. False Negative	✓ The model predicts that the patient is not having cancer (negative) but he actually has cancer (false prediction).
4. True Negative	✓ The model predicts that the patient is not having cancer (negative) and indeed he does not have cancer (true prediction).

5. Scenario

- ✓ Now consider the above classification (pregnant or not pregnant) carried out by a machine learning algorithm. The output of the machine learning algorithm can be mapped to one of the following categories.

5.1. True positive

- ✓ A person who is actually pregnant (positive) and classified as pregnant (positive). This is called TRUE POSITIVE (TP).



5.2. True Negative

- ✓ A person who is actually not pregnant (negative) and classified as not pregnant (negative). This is called TRUE NEGATIVE (TN).



5.3. False Positive

- ✓ A person who is actually not pregnant (negative) and classified as pregnant (positive). This is called FALSE POSITIVE (FP).



5.4. False Negative

- ✓ A person who is actually pregnant (positive) and classified as not pregnant (negative). This is called FALSE NEGATIVE (FN).



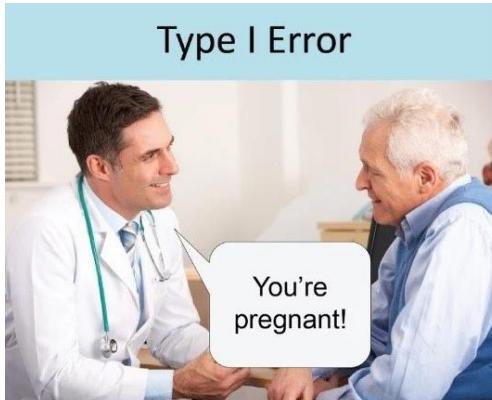
6. Type I and Type II errors

Type I error

- ✓ Type I error also called as False Positive.
- ✓ Type I error occurs if you reject the null hypothesis when it was true.

Type II error

- ✓ Type II error also called as False Negative
- ✓ Type II error occurs if you accept the null hypothesis when it was false.



7. Confusion Matrix

- ✓ We can represent above discussion by using matrix also

		Actual Cancer = Yes	Actual Cancer = No	
Predicted Cancer = Yes	True Positive (TP)	False Positive (FP)		
	False Negative (FN)	True Negative (TN)		
Predicted Cancer = No				

8. Table2

- ✓ True Positive = No. of True Positives from total predictions
- ✓ False Positive = No. of False Positives from total predictions
- ✓ False Negative = No. of False Negatives from total predictions
- ✓ True Negative = No. of True Negatives from total predictions
- ✓ Total number of prediction = $TP + FP + FN + TN$

9. Confusion matrix example

- ✓ Let's try to represent this confusion matrix with real numbers.

		Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive	False Positive	
	57	14	
Predicted Cancer = No	False Negative	True Negative	
	23	171	

- 57 = No. of True Positives from total predictions
- 14 = No. of False Positives from total predictions
- 23 = No. of False Negatives from total predictions
- 171 = No. of True Negatives from total predictions
- $57+14+23+171 = 265$ = Total no. of predictions

10. Performance Metrics

- ✓ If we have confusion matrix then we can use it to calculate various performance metrics to measure your classification models.

11. Accuracy

- ✓ Accuracy is a measure of the fraction of times the model predicted correctly (both true positive and true negative) out of total no. of predictions.

		Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive	False Positive 14	
	57		
Predicted Cancer = No	False Negative 23	True Negative 171	

$$\text{Accuracy} = \frac{TP+TN}{\text{Total Predictions}}$$

- ✓ This is the simple and very intuitive metrics.
- ✓ Let us consider the accuracy of the classifier in our example above.

$$\text{Accuracy} = \frac{57+171}{265} = 0.86$$

- ✓ So this means our model can classify cancer and non-cancer cases with 86% accuracy.

12. Accuracy can be misleading

- ✓ We felt like accuracy looks good to measure the performance of the model, but there is a catch!!
- ✓ Assume that your machine model trains badly and only identifies negative scenarios and miss to identify true positive completely. (This happens when there is a class imbalance in training data.)

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 0	False Positive 0
Predicted Cancer = No	False Negative 80	True Negative 185

- ✓ So in our example, say the classifier completely misses to predict any case of cancer and marks all cases as non-cancer, this is how the confusion matrix will look like.
- ✓ So the accuracy of our classifier will be

$$\text{Accuracy} = \frac{185}{265} = 0.70$$

- ✓ So here model could not predict a single cancer case but we got accuracy is 70% which is not so good😊.
- ✓ So, only believing Accuracy is not good way
- ✓ This is why we have several other metrics to measure the performances of the machine learning model.

13. Recall or Sensitivity or True Positive Rate

- ✓ Recall is the fraction of times the model predicts positive cases correctly from the total number of actual positive cases.

$$\text{Recall} = \frac{TP}{TP+FN}$$

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 57	False Positive 14
Predicted Cancer = No	False Negative 23	True Negative 171

- ✓ Let's understand, TP + FN is the total number of actual positive cases that the classifier should have identified.
- ✓ In our example, TP = 57 cancer cases classifier predicted correctly.
- ✓ FN = 23 cancer cases our classifier missed.
- ✓ SO, the total cancer cases were TP + FN = 57+23 = 80.

$$\text{Recall} = \frac{57}{57+23} = \frac{57}{80} = 0.71$$

- ✓ So this means our model has 71% recall capability to predict cancer cases from total actual cancer cases.

14. Specificity or True Negative Rate

- ✓ Specificity is the fraction of times the classifier predicts negatives cases correctly from the total number of actual negative cases.
- ✓ This metrics is the opposite of Recall/Sensitivity that we understood above.

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 57	False Positive 14
Predicted Cancer = No	False Negative 23	True Negative 171

$$\text{Specificity} = \frac{TN}{TN+FP}$$

- ✓ Here $TN+FP$ is the total number of actual negative cases.
- ✓ In our example, $TN = 171$ non-cancer cases were predicted correctly by the classifier and $FP = 14$ non-cancer cases were incorrectly identified as cancer.
- ✓ So, the total non-cancer cases were $TN+FP = 171+14 = 185$.

$$\text{Specificity} = \frac{171}{171+14} = \frac{171}{185} = 0.92$$

- ✓ So this means our model has 92% capability to identify negative cases from total number of actual negative cases.

15. Precision

- ✓ Precision calculates the fraction of times the model predicts positive cases correctly from the total number of positive cases it predicted.

$$\text{Precision} = \frac{TP}{TP+FP}$$

- ✓ Let's try to understand TP+FP is the total number of cases predicted by the classifier as positive.
- ✓ TP is the total number of cases that are actually positive.

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 57	False Positive 14
Predicted Cancer = No	False Negative 23	True Negative 171

- ✓ In our example, the classifier predicted $TP+FP = 71$ cases of cancer.
- ✓ Out of these 71 cases, only $TP = 57$ cases were correct.

$$\text{Precision} = \frac{57}{57+14} = \frac{57}{71} = 0.80$$

- ✓ So this means our classifier has a precision or correctness of 80% when it predicts cases as cancer.

16. F1 Score

- ✓ So, Recall and precision are the best way to choose.
- ✓ Instead of using many metrics, Can't we chose one metric?
- ✓ Yes we can, here F1 score helps actually
- ✓ The F1 score is a measure of a model's accuracy.

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- ✓ Using F1 scores we can now evaluate and compare the performance of the models easily now.

22. Data Science – Machine Learning – Bias Variance Trade Off

Contents

1. Bias.....	2
2. Variance.....	2
3. Remember	2
4. The goal of supervised algorithm	2
5. Error	2
6. Prediction error	2
7. Bias Error	3
8. Low Bias.....	3
9. High-Bias.....	3
10. Examples.....	3
11. Variance Error.....	4
12. Low variance:.....	4
13. High variance	4
14. Examples.....	4
15. Bias-Variance Trade-Off	5
16. Configuring the bias-variance trade-off for specific algorithms.....	5
17. The relationship b/w bias and variance	5
18. In reality.....	5
19. Bias and variance.....	6

22. Data Science – Machine Learning – Bias Variance Trade Off

1. Bias

- ✓ Bias is the error or difference between prediction results and actual values.

2. Variance

- ✓ Variance is the error that occurs due to small changes in the training set.

3. Remember

- ✓ The bias and variance provide the information to understand the performance of machine learning algorithms while prediction.

4. The goal of supervised algorithm

- ✓ In supervised learning technique, an algorithm learns a model from training data.
- ✓ The goal of any supervised learning algorithm is,
 - Find the best mapping function/target function (f) for the output variable (Y) for the given input data (X).

5. Error

- ✓ Error means some wrong calculation.
- ✓ Error can be small error or big error.
- ✓ Small error can be acceptable and big error should be minimize

6. Prediction error

- ✓ Prediction error means, we made some prediction but there are some errors are existing.
- ✓ The prediction error for any machine learning algorithm can be divided into two parts
 - Bias Error
 - Variance Error

7. Bias Error

- ✓ Bias are the simplifying assumptions made by a model to make the target function easy to learn.

8. Low Bias

- ✓ Low bias value suggests **more assumptions** about the form of the target function.

9. High-Bias

- ✓ High bias value suggests **fewer assumptions** about the form of the target function.

10. Examples

- ✓ Low-bias machine learning algorithms are,

- Decision Trees
- k-Nearest Neighbors
- Support Vector Machines.

- ✓ High-bias machine learning algorithms are,

- Linear Regression
- Linear Discriminant Analysis
- Logistic Regression.

11. Variance Error

- ✓ Variance is the error that occurs due to small changes in the training set.
- ✓ The target function is estimated from the training data by a machine learning algorithm, we can expect the algorithm may have some variance.
- ✓ Ideally, it should not change too much from one training dataset to the other.

12. Low variance:

- ✓ Low variance value suggests small changes to the estimate of the target function with changes to the training dataset.

13. High variance

- ✓ High variance value suggests large changes to the estimate of the target function with changes to the training dataset.

14. Examples

- ✓ Low-variance machine learning algorithms are,
 - Linear Regression
 - Linear Discriminant Analysis
 - Logistic Regression.
- ✓ High-variance machine learning algorithms are,
 - Decision Trees
 - k-Nearest Neighbors
 - Support Vector Machines.

15. Bias-Variance Trade-Off

- ✓ The goal of any supervised machine learning algorithm is to achieve the sweet spot of **low bias and low variance**.
- ✓ It means, the algorithm should achieve good prediction performance.

Note 1

- ✓ Parametric or linear machine learning algorithms often have a **high bias** but a **low variance**.
- ✓ Nonparametric or nonlinear machine learning algorithms often have a **low bias** but a **high variance**.

16. Configuring the bias-variance trade-off for specific algorithms

- ✓ Below are two examples of configuring the bias-variance trade-off for specific algorithms
 - The **k-nearest neighbour's algorithm** has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute to the prediction and in turn increases the bias of the model.
 - The **support vector machine algorithm** has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

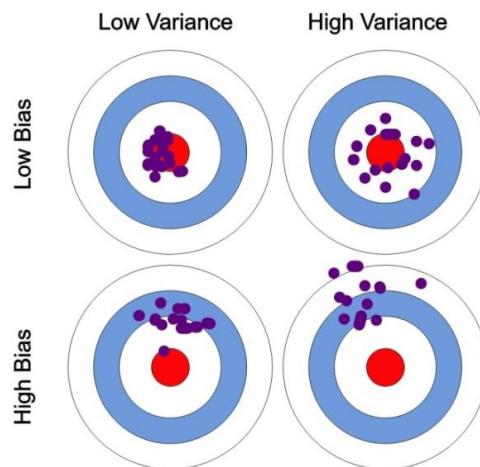
17. The relationship b/w bias and variance

- ✓ Increasing the bias will decrease the variance.
- ✓ Increasing the variance will decrease the bias.

18. In reality

- ✓ In reality we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function.

19. Bias and variance



Error

- ✓ The error is calculated as the difference between predicted and actual value.

Low bias and low variance

- ✓ This is also called as Bias-Variance Tradeoff.
- ✓ If low bias and low variance then the error is very less.
- ✓ So, in this scenario the model is very accurate

Low bias and high variance

- ✓ So, in this scenario the model having lower accuracy.

High bias and low variance

- ✓ So, in this scenario the model having lower accuracy.

High bias and High variance

- ✓ So, in this scenario the model having lower accuracy.

```
pip install mlxtend
```

Program Name Calculate bias and variance values
demo1.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

model = LinearRegression()

mse, bias, var = bias_variance_decomp(model, X_train, y_train,
X_test, y_test, loss='mse', num_rounds = 200, random_seed = 1)

print('MSE:', mse)
print('Bias:', bias)
print('Variance:', var)
```

Output

```
MSE: 25.994907724912153
Bias: 22.41211837451609
Variance: 3.5827893503960495
```

- ✓ In this case, we can see that the model has a high bias and a low variance.

23. Data Science – Machine Learning – Random Forest Algorithm

Contents

1. Random Forest	2
2. Ensemble learning	2
3. Process behind in Random Forest	2
5. Why use Random Forest?	3
6. How does Random Forest algorithm work?.....	3
7. Steps in Random Forest	4
8. Use case	5

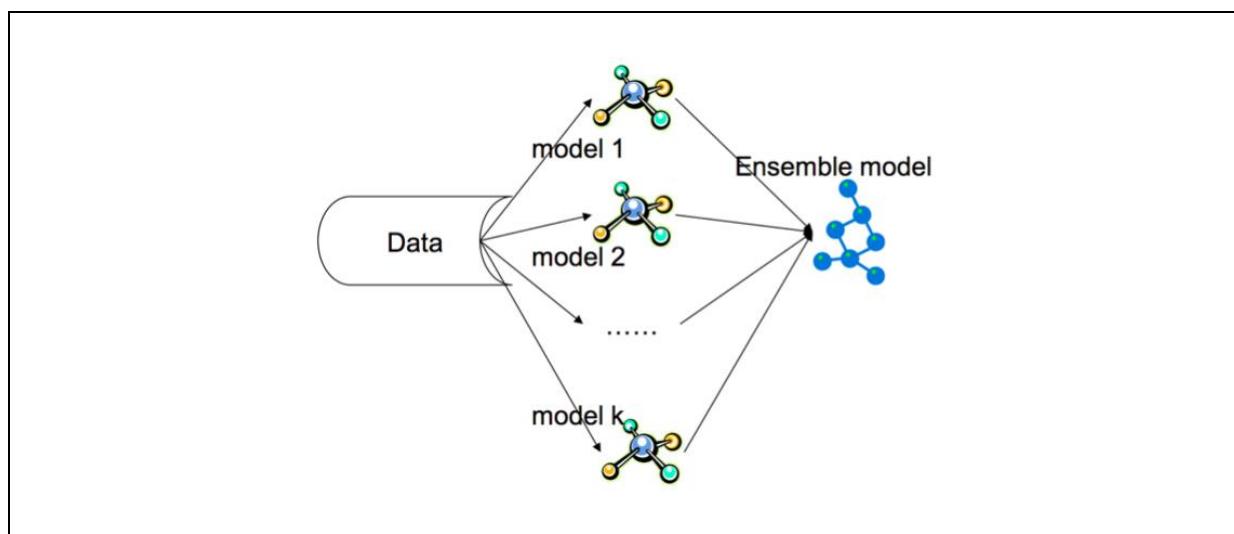
23. Data Science – Machine Learning – Random Forest Algorithm

1. Random Forest

- ✓ Random Forest is supervised learning technique.
- ✓ It can be used for both Classification and Regression problems in ML.
- ✓ Random forest is the concept of ensemble learning.

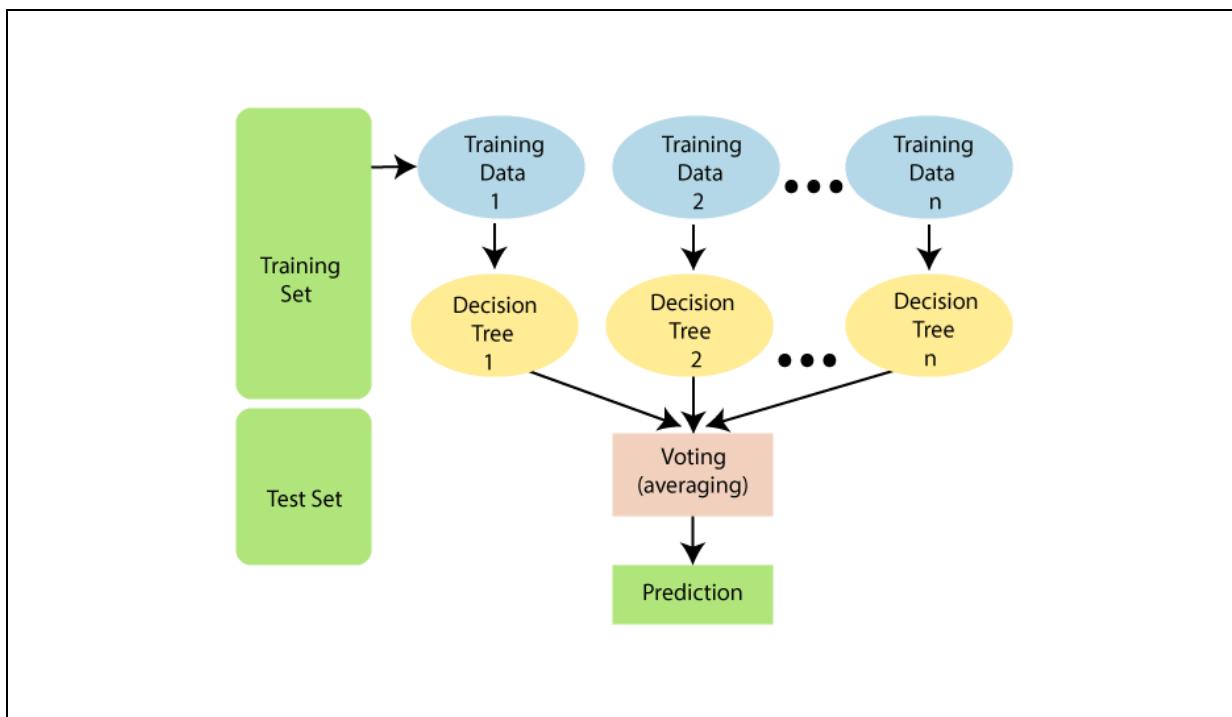
2. Ensemble learning

- ✓ This is the process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.



3. Process behind in Random Forest

- ✓ Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset.
- ✓ It takes the average of all decision trees to improve the predictive accuracy of that dataset.
 - Instead of believing on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- ✓ The greater number of trees in the forest leads to higher accuracy.



5. Why use Random Forest?

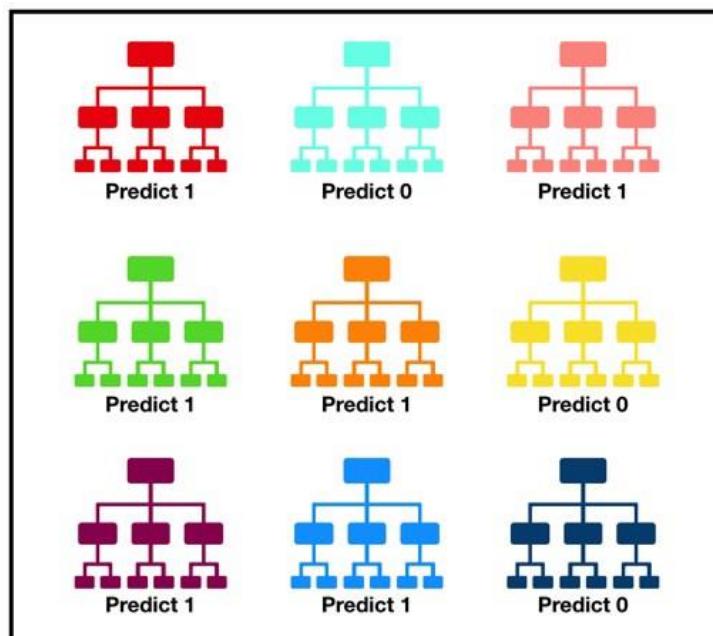
- ✓ It takes less training time as compared to other algorithms.
- ✓ It predicts output with high accuracy, even for the large dataset it runs efficiently.
- ✓ It can also maintain accuracy when a large proportion of data is missing.

6. How does Random Forest algorithm work?

- ✓ Random Forest works in two-phase,
 - First it creates the random forest by combining N decision tree.
 - Second is to make predictions for each tree created in the first phase.

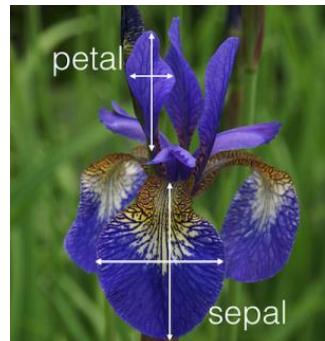
7. Steps in Random Forest

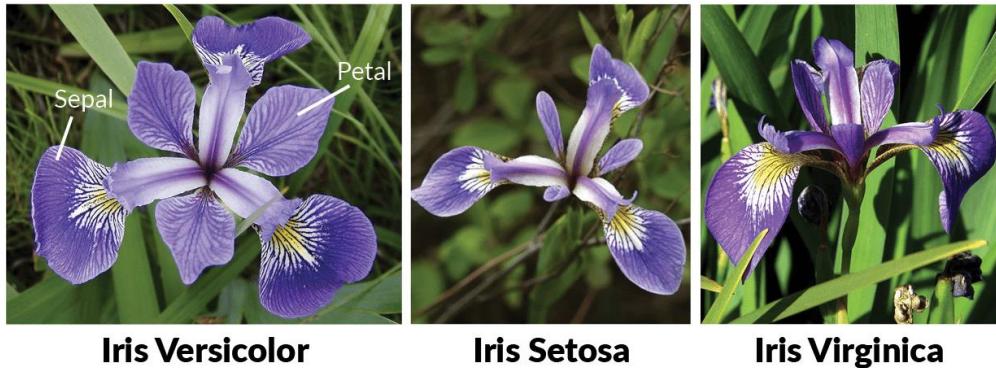
- ✓ Step-1: Select random K data points from the training set.
- ✓ Step-2: Build the decision trees associated with the selected data points
- ✓ Step-3: Choose the number N for decision trees that you want to build.
- ✓ Step-4: Repeat Step 1 & 2.
- ✓ Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.



8. Use case

- ✓ Assuming that Abhi had a hobby which is interested in distinguishing the species of some iris flowers that he has found
- ✓ He has collected some measurements associated with each iris, which are:
 - The **length** and **width** of the **petals**
 - The **length** and **width** of the **sepals**, all measured in centimetres.
- ✓ He also has the measurements of some irises that have been previously identified to the species
 - setosa,
 - versicolor
 - virginica
- ✓ The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known.
- ✓ So that we can predict the species for the new irises that she has found.





Flower codes

- ✓ Setosa - 0
- ✓ Versicolor - 1
- ✓ Virginica - 2

Program Name Loading iris dataset
demo1.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(dir(iris))
```

Output

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target',
'target_names']
```

Program Name Displaying feature names
demo2.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
```

Output

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
```

Program Name Displaying target names
demo3.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.target_names)
```

Output

```
['setosa' 'versicolor' 'virginica']
```

Program Name Displaying data
demo4.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.data)
```

Output

```
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```

Program Name Length of the data
demo5.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(len(iris.data))
```

Output
150

Program Name Create a Dataframe by using data and features
demo6.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

print(df)
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8
[150 rows x 4 columns]				

Program Name Adding target column to the dataframe
demo7.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(df.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Program Name Displaying target == 0 flowers
demo8.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(df[df.target == 0].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Program Name Displaying length of the target == 0 flowers
demo9.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(len(df[df.target == 0]))
```

Output

50

Program Name Displaying target == 1 flowers
demo10.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target == 1].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

Program Name Displaying length of the target == 1 flowers
demo11.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==1]))
```

Output

50

Program Name Displaying target == 2 flowers
demo12.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(df[df.target==2].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

Program Name Displaying length of the target == 2 flowers
demo13.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(len(df[df.target == 2]))
```

Output

50

Program Name Displaying the flower names
demo14.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)
print(df)
```

Output

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target flower_name
0            5.1          3.5            1.4           0.2       0    setosa
1            4.9          3.0            1.4           0.2       0    setosa
2            4.7          3.2            1.3           0.2       0    setosa
3            4.6          3.1            1.5           0.2       0    setosa
4            5.0          3.6            1.4           0.2       0    setosa
..           ...
145           6.7          3.0            5.2           2.3       2  virginica
146           6.3          2.5            5.0           1.9       2  virginica
147           6.5          3.0            5.2           2.0       2  virginica
148           6.2          3.4            5.4           2.3       2  virginica
149           5.9          3.0            5.1           1.8       2  virginica
[150 rows x 6 columns]
```

Program Name All setosa flowers
demo15.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

setosa_50 = df[:50]
print(setosa_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Program Name All versicolor flowers
demo16.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

versicolor_50 = df[50:100]
print(versicolor_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

Program Name All virginica flowers
demo17.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

virginica_50 = df[100:]
print(virginica_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

Program Name Splitting the data
 demo18.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis = 'columns')
y = df.target

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

print("Splitting the data")
```

Output

Splitting the data

Program Name Model training
demo19.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

# Train Using RandomForestClassifier

model = RandomForestClassifier(n_estimators=40)
model.fit(X_train, y_train)
print("Model got trained")
```

Output

Model got trained

Program Name Model score
demo20.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

model = RandomForestClassifier(n_estimators=40)
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

Output

0.9666666666666667

Note

- ✓ `n_estimators=40` parameter defines the number of trees in the random forest.

Program Name Model prediction
demo21.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

model = RandomForestClassifier(n_estimators = 40)

model.fit(X_train.values, y_train)

print(model.predict([[4.8,3.0,1.5,0.3]]))
```

Output

[0]

24. Data Science – Machine Learning – K Fold Cross Validation

Contents

1. K-fold cross validation.....	2
2. Ways to training the model	2
3. Scenario 1	3
4. Scenario 2	4
5. Limitation.....	5
6. K fold cross validation	5
7. Iteration 1:.....	5
8. Iteration 2:.....	6
9. Last iteration:	7
10. Average of scores	7

24. Data Science – Machine Learning – K Fold Cross Validation

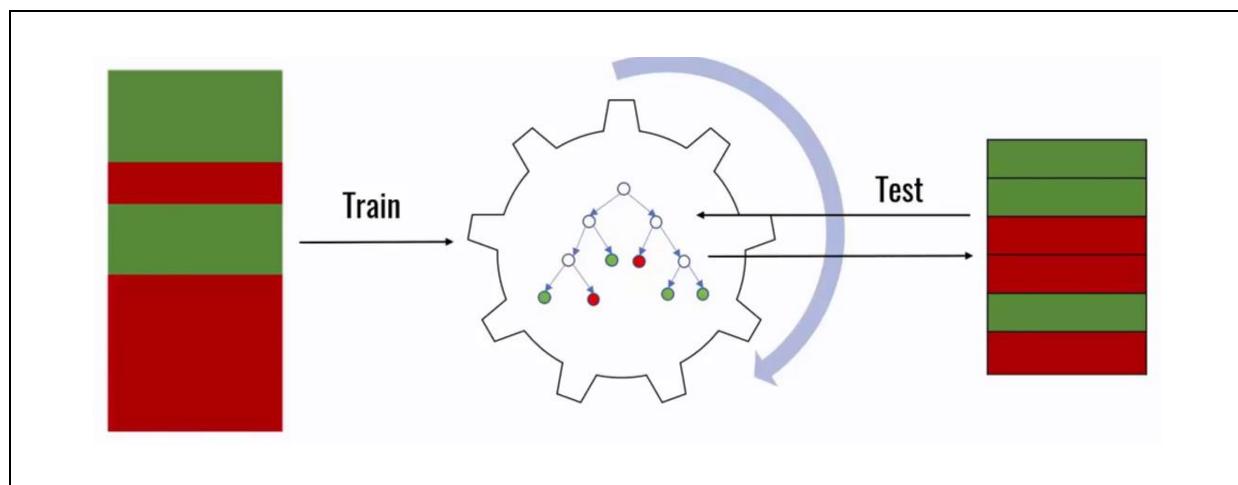
- ✓ To solve a problem we do have different machine learning algorithm for same problem
- ✓ So, we need to understand clearly which model is the best to use

1. K-fold cross validation

- ✓ Cross-validation is a technique, it evaluate the model performance.

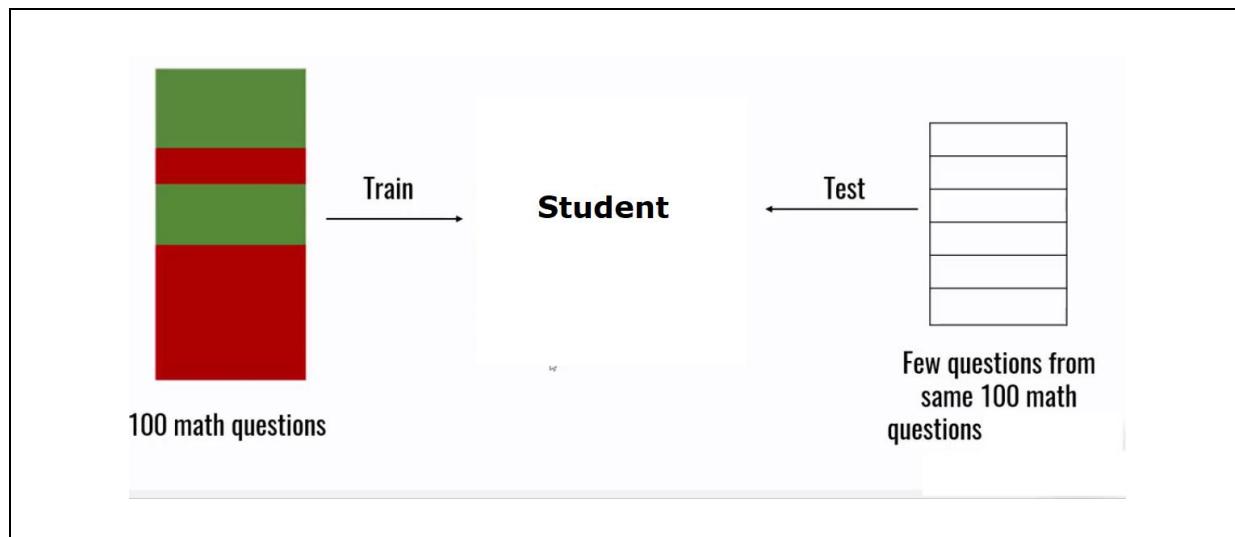
2. Ways to training the model

- ✓ So far we learned to spilt the data into train and test datasets
- ✓ Once the model got trained then we need to test the model



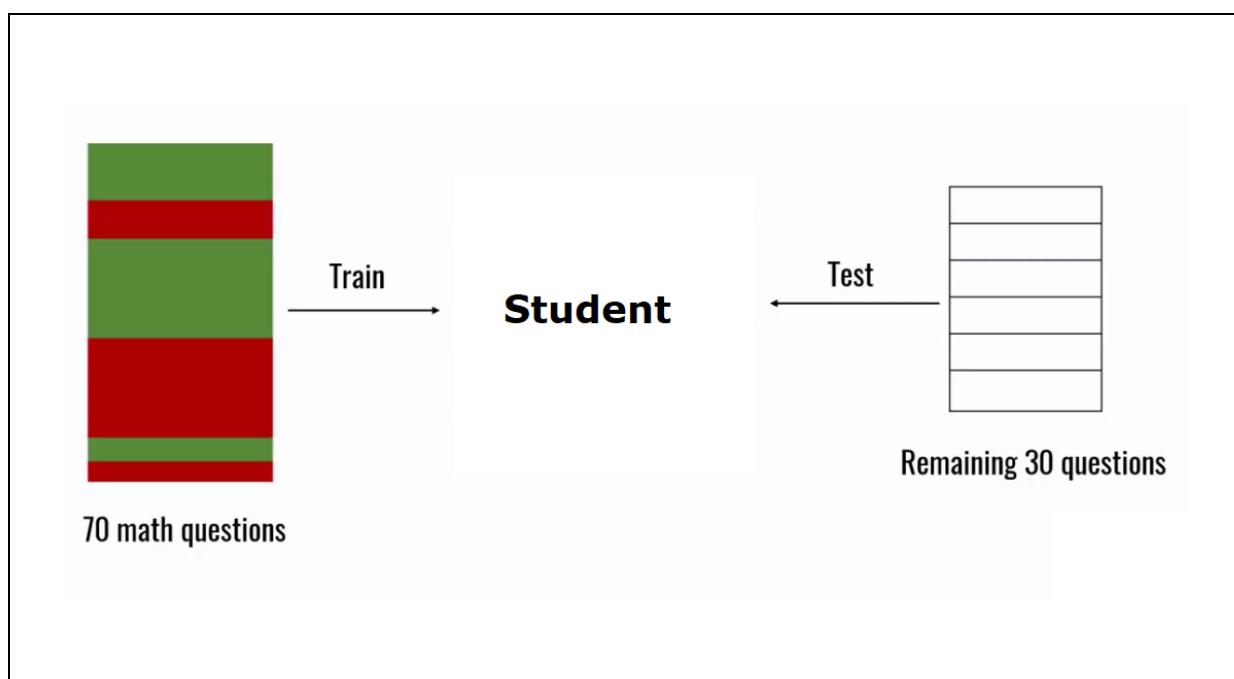
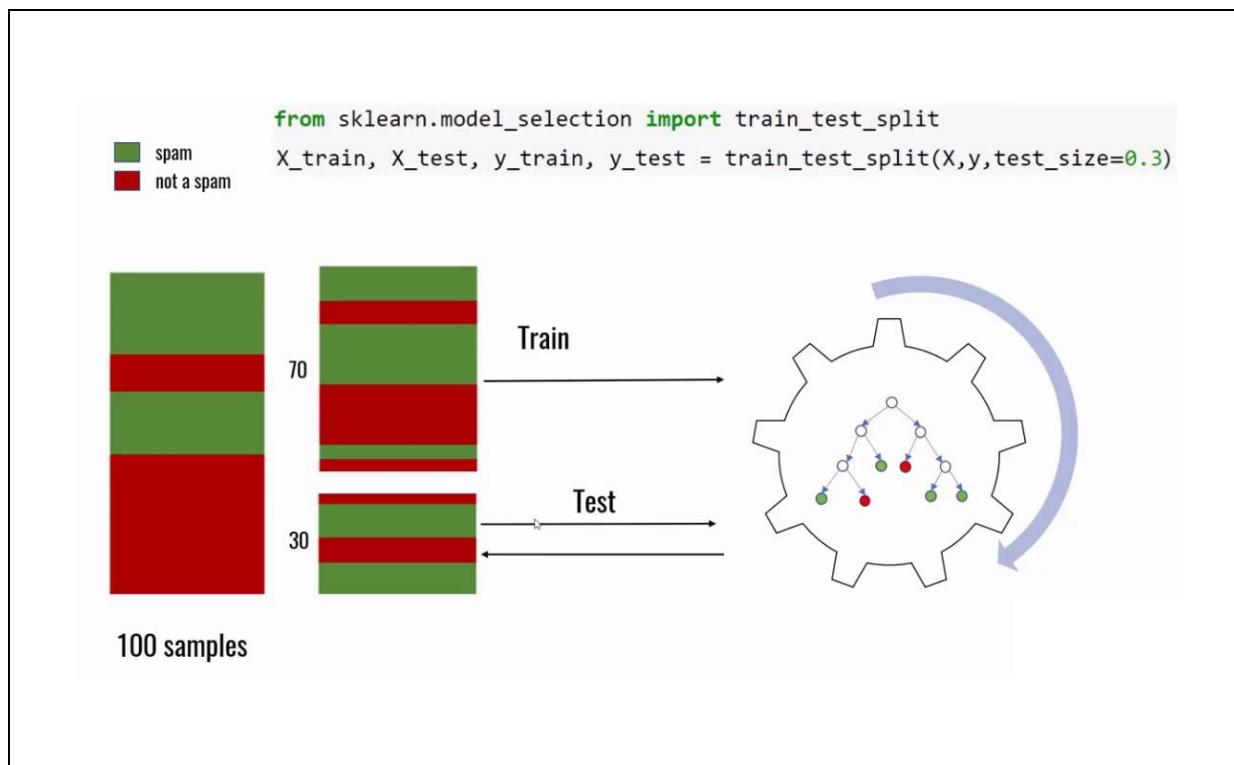
3. Scenario 1

- ✓ Use all dataset to train and test the model



4. Scenario 2

- ✓ Split available dataset into training and testing to test the model



5. Limitation

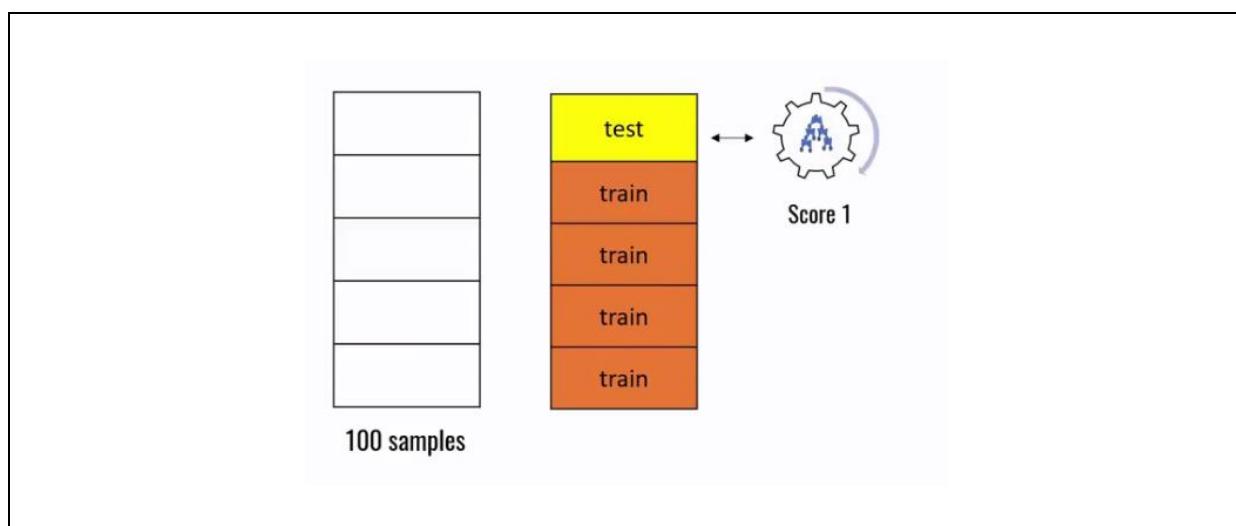
- ✓ During testing the model with test dataset, model may fail in few scenarios because model need to face new scenarios right

6. K fold cross validation

- ✓ Cross-validation is a technique, it evaluate the model performance.
- ✓ We used to divide 100 samples into folds, each contains 20 samples
- ✓ Then now we can start iterations to test the model in different ways

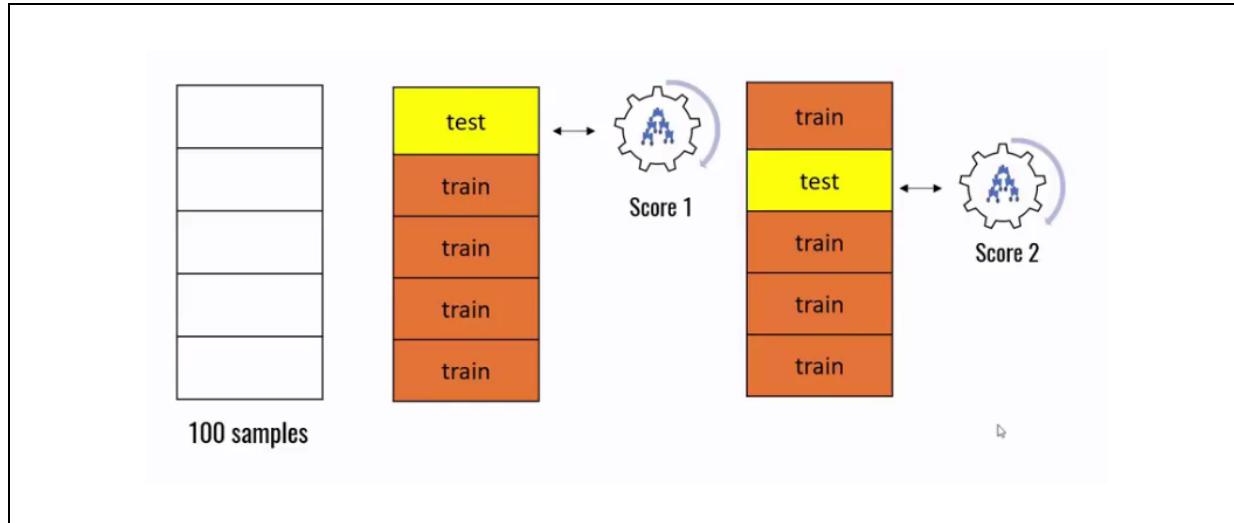
7. Iteration 1

- ✓ Use the first fold to test the model
- ✓ Use the remaining folds to training



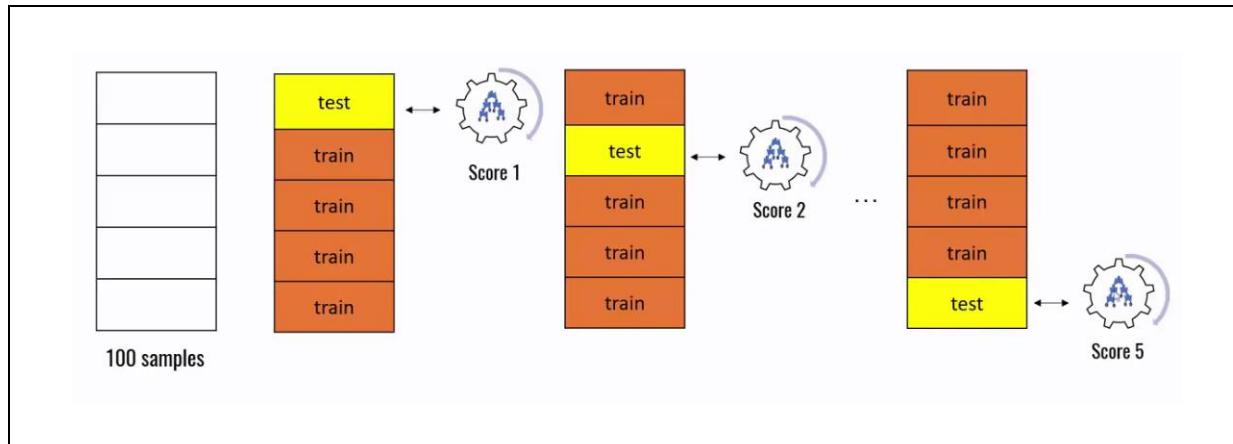
8. Iteration 2

- ✓ Use the second fold to test the model
- ✓ Use the remaining folds to training



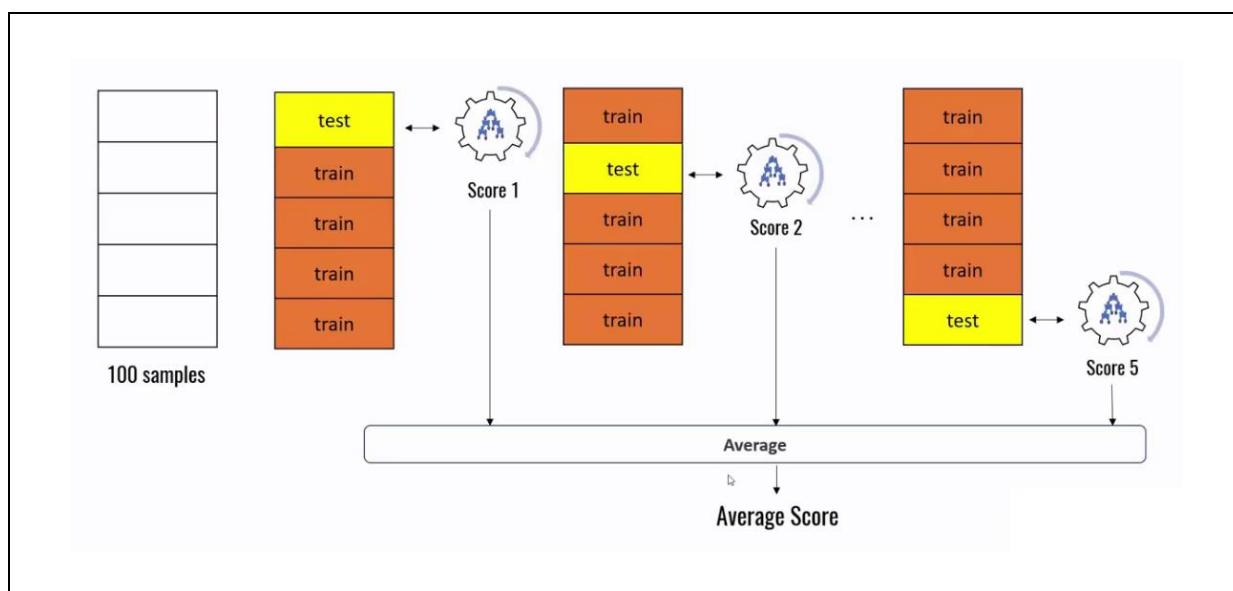
9. Last iteration

- ✓ Repeat the process till to the last fold



10. Average of scores

- ✓ This technique is good to calculate the average of all iterations scores



Program Name Dataset is loading
demo1.py

```
from sklearn.datasets import load_digits  
  
digits = load_digits()  
  
print("Dataset is loading")
```

Output Dataset is loading

Program Name Splitting the data into train and test datasets
demo2.py

```
from sklearn.datasets import load_digits  
from sklearn.model_selection import train_test_split  
  
digits = load_digits()  
X_train, X_test, y_train, y_test = train_test_split(digits.data,  
digits.target, test_size = 0.3)  
  
print("Splitting the data into train and test")
```

Output Splitting the data into train and test

Program

Name demo3.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)

lr = LogisticRegression(solver = 'lbfgs', max_iter = 3000)
lr.fit(X_train, y_train)
print(lr.score(X_test, y_test))
```

Output

0.95

Program Name Applying SVM algorithm
demo4.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)

svm = SVC()
svm.fit(X_train, y_train)

print(svm.score(X_test, y_test))
```

Output
0.9907407407407407

Program Name Applying Random forest algorithm
demo5.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)

rf = RandomForestClassifier(n_estimators=40)
rf.fit(X_train, y_train)

print(rf.score(X_test, y_test))
```

Output
0.975925925925926

Program Name K fold cross validation
demo6.py

```
from sklearn.model_selection import KFold  
  
kf = KFold(n_splits=3)  
  
print(kf)
```

Output
KFold(n_splits=3, random_state=None, shuffle=False)

Program Name K fold cross validation: Example
demo7.py

```
from sklearn.model_selection import KFold  
  
kf = KFold(n_splits=3)  
  
for train_index, test_index in kf.split([1,2,3,4,5,6,7,8,9]):  
    print(train_index, test_index)
```

Output

```
[3 4 5 6 7 8] [0 1 2]  
[0 1 2 6 7 8] [3 4 5]  
[0 1 2 3 4 5] [6 7 8]
```

Program Name Logistic regression model performance using cross_val_score
demo8.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression

digits = load_digits()

a = LogisticRegression(solver = 'lbfgs', max_iter = 5000)
scores = cross_val_score(a, digits.data, digits.target, cv=3)

print(scores)
```

Output

```
[0.92153589 0.94156928 0.91652755]
```

Program

Name demo9.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.svm import SVC

digits = load_digits()

b = SVC()
scores = cross_val_score(b, digits.data, digits.target, cv=3)

print(scores)
```

Output

```
[0.92153589 0.94156928 0.91652755]
```

**Program
Name**

Random forest model performance using cross_val_score
demo10.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier

digits = load_digits()

c = RandomForestClassifier(n_estimators=40)
scores = cross_val_score(c, digits.data, digits.target, cv=3)

print(scores)
```

Output

```
[0.93823038 0.94156928 0.92821369]
```

Program Name Checking average of all model scores
demo11.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import numpy as np

digits = load_digits()

a = LogisticRegression(solver = 'lbfgs', max_iter = 5000)
b = SVC()
c = RandomForestClassifier(n_estimators=40)

scores1 = cross_val_score(a, digits.data, digits.target, cv=3)
scores2 = cross_val_score(b, digits.data, digits.target, cv=3)
scores3 = cross_val_score(c, digits.data, digits.target, cv=3)

print(np.average(scores1))
print(np.average(scores2))
print(np.average(scores3))
```

Output

```
0.9265442404006677
0.9699499165275459
0.9315525876460767
```

25. Data Science – Machine Learning – Support Vector Machine

Contents

1. Support Vector Machine	2
2. Decision boundary.....	2
3. Why SVM name is SVM.....	2
4. Usage	2
5. Types of SVM	2
6. Linear SVM.....	3
7. Non-linear SVM	3
8. 2 features forms straight line & 3 features forms plane.....	3
9. How does SVM works?.....	4
10. Hyperplane and Support vectors.....	5
11. Use case	6

25. Data Science – Machine Learning – Support Vector Machine

1. Support Vector Machine

- ✓ Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms.
- ✓ It is used for Classification as well as Regression problems.
- ✓ Mostly used for Classification problems in Machine Learning.

2. Decision boundary

- ✓ The goal of the SVM algorithm is to create the best line or decision boundary that can separate n-dimensional space into classes.
- ✓ This best decision boundary is called a hyperplane.

3. Why SVM name is SVM

- ✓ SVM chooses the extreme points/vectors that help in creating the hyperplane.
- ✓ These extreme cases are called as support vectors; hence this algorithm is called as Support Vector Machine.

4. Usage

- ✓ SVM algorithm can be used for,
 - Face detection
 - Image classification
 - Text categorization

5. Types of SVM

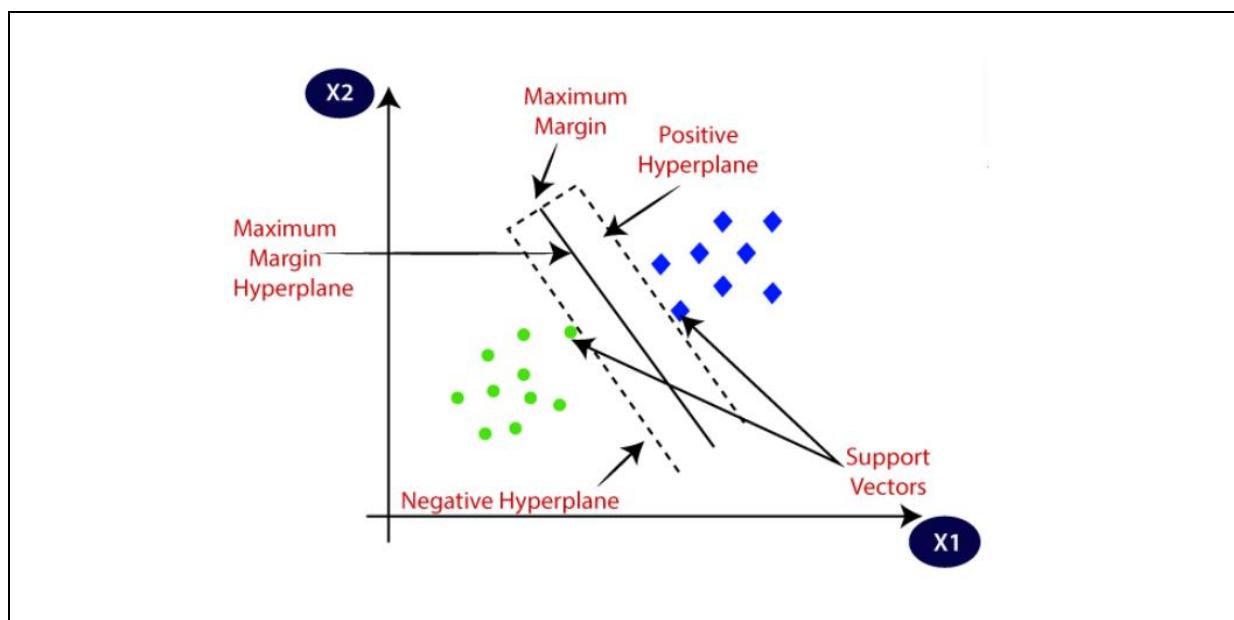
- ✓ Linear SVM
- ✓ Non-linear SVM

6. Linear SVM

- ✓ If the dataset can be classified into two classes by using a single straight line, then that is called as linearly separable data, and classifier is used called as Linear SVM classifier.

7. Non-linear SVM

- ✓ If the data set cannot be classified by using a straight line, then that is called as non-linear separable data, and classifier is used called as Non-linear SVM classifier.



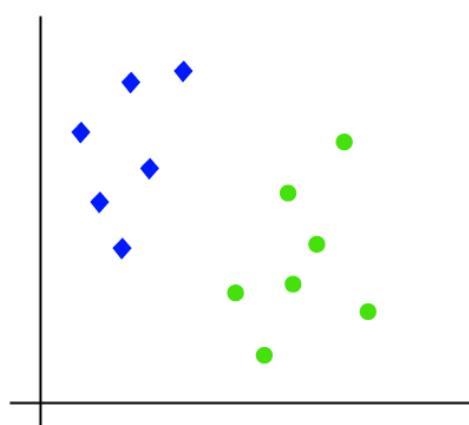
8. 2 features forms straight line & 3 features forms plane

- ✓ The dimensions of the hyperplane depend on the features present in the dataset,
 - If there are **2 features** then hyperplane will be a **straight line**
 - If there are **3 features** then hyperplane will be a **2-dimension plane**

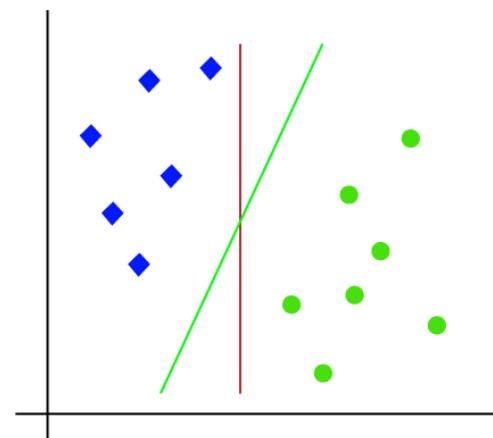
9. How does SVM works?

Linear SVM

- ✓ Assuming that we have a dataset that has two tags (green and blue), and the dataset has two features x_1 and x_2 .
- ✓ We want a classifier that can classify the pair (x_1, x_2) of coordinates in either green or blue.

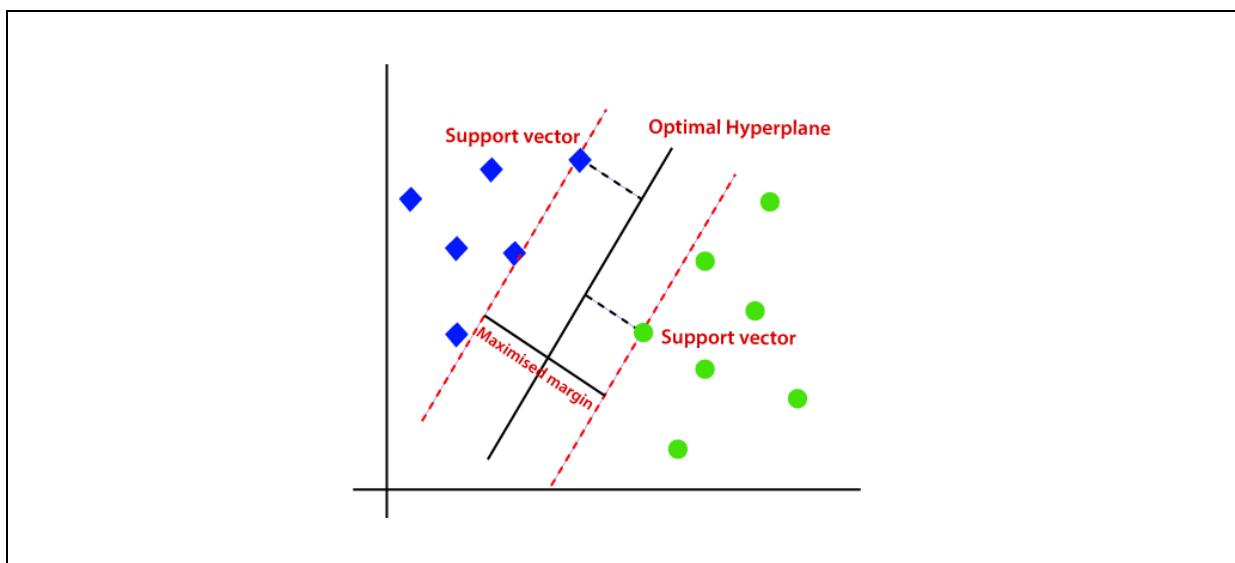


- ✓ Though it is 2-d space we can use straight line to separate these classes
- ✓ There can be multiple lines that can separate these classes too



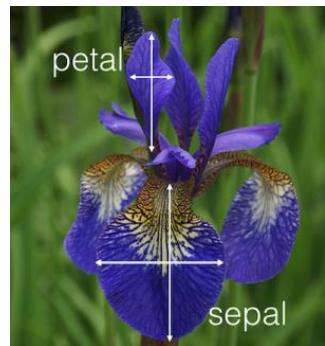
10. Hyperplane and Support vectors

- ✓ SVM algorithm helps to find the best line or decision boundary, this best boundary or region is called as a **hyperplane**.
- ✓ SVM algorithm finds the closest point of the lines from both the classes; these points are called **support vectors**.
- ✓ The **distance between** the vectors and the hyperplane is called as **margin**.
- ✓ The **goal** of SVM is to maximize this margin.
- ✓ The hyperplane with maximum margin is called the **optimal hyperplane**.



11. Use case

- ✓ Assuming that Abhi had a hobby which is interested in distinguishing the species of some iris flowers that he has found
- ✓ He has collected some measurements associated with each iris, which are:
 - The **length** and **width** of the **petals**
 - The **length** and **width** of the **sepals**, all measured in centimetres.
- ✓ She also has the measurements of some irises that have been previously identified to the species
 - **setosa**,
 - **versicolor**
 - **virginica**
- ✓ The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known.
- ✓ So that we can predict the species for the new irises that she has found.



Flower codes

- ✓ Setosa - 0
- ✓ Versicolor - 1
- ✓ Virginica - 2

Program Name Loading iris dataset
demo1.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(dir(iris))
```

Output

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target',
'target_names']
```

Program Name Displaying feature names
demo2.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
```

Output

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
```

Program Name Displaying target names
demo3.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.target_names)
```

Output

```
['setosa' 'versicolor' 'virginica']
```

Program Name Displaying data
demo4.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.data)
```

Output

```
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```

Program Name Length of the data
demo5.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(len(iris.data))
```

Output
150

Program Name Create a Dataframe by using data and features
demo6.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

print(df)
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

Program Name Adding target column to the dataframe
demo7.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Program Name Displaying target == 0 flowers
demo8.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==0].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Program Name Displaying length of the target == 0 flowers
demo9.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==0]))
```

Output

50

Program Name Displaying target == 1 flowers
demo10.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==1].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

Program Name Displaying length of the target == 0 flowers
demo11.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==1]))
```

Output

50

Program Name Displaying target == 2 flowers
demo12.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==2].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

Program Name Displaying length of the target == 2 flowers
demo13.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==2]))
```

Output

50

Program Name Displaying the flower names
demo14.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
print(df)
```

Output

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target flower_name
0           5.1          3.5            1.4            0.2      0    setosa
1           4.9          3.0            1.4            0.2      0    setosa
2           4.7          3.2            1.3            0.2      0    setosa
3           4.6          3.1            1.5            0.2      0    setosa
4           5.0          3.6            1.4            0.2      0    setosa
..          ...
145          6.7          3.0            5.2            2.3      2  virginica
146          6.3          2.5            5.0            1.9      2  virginica
147          6.5          3.0            5.2            2.0      2  virginica
148          6.2          3.4            5.4            2.3      2  virginica
149          5.9          3.0            5.1            1.8      2  virginica
[150 rows x 6 columns]
```

Program Name All setosa flowers
demo15.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

setosa_50 = df[:50]
print(setosa_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Program Name All versicolor flowers
demo16.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

versicolor_50 = df[50:100]
print(versicolor_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

Program Name All virginica flowers
demo17.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

virginica_50 = df[100:]
print(virginica_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

Program
Name

All types of flowers
demo18.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]

print("All types of flowers")
```

Output

All types of flowers

Program
Name

Plotting
demo19.py

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]

# Sepal length vs Sepal Width (Setosa vs Versicolor)

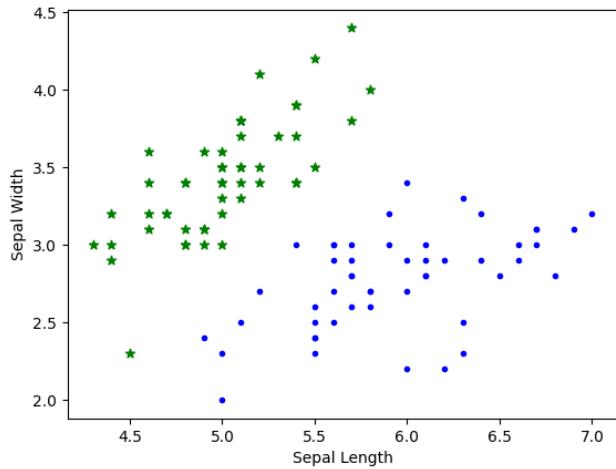
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')

plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],
color="green", marker='*')

plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],
color="blue", marker='.')

plt.show()
```

Output



Program Name Plotting
demo20.py

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

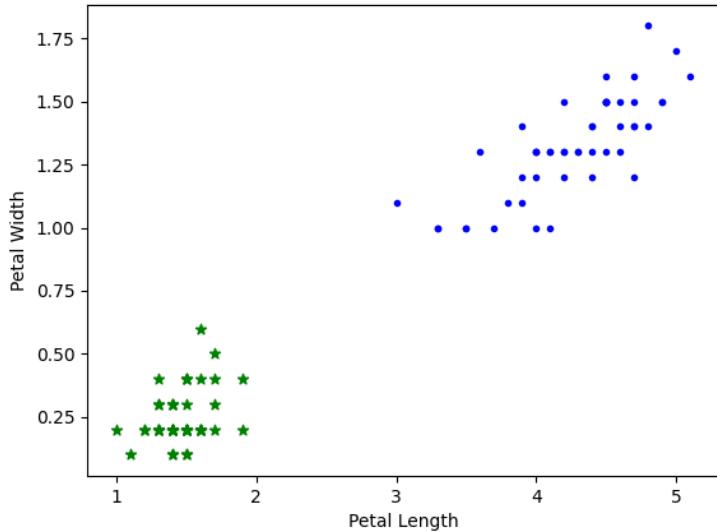
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]

# Petal length vs Petal Width (Setosa vs Versicolor)

plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],
color="green", marker='*')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],
color="blue", marker='.')

plt.show()
```

Output



Program Name Splitting the data
demo21.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

print("Splitting the data")
```

Output

Splitting the data

Program Name Model training
demo22.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

# Train Using Support Vector Machine (SVM)

model = SVC()
model.fit(X_train, y_train)
print("Model got trained")
```

Output

Model got trained

Program Name Model score
demo23.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

# Train Using Support Vector Machine (SVM)

model = SVC()
model.fit(X_train, y_train)

print(model.score(X_test, y_test))
```

Output

0.9666666666666667

Program Name Model prediction
demo24.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

# Train Using Support Vector Machine (SVM)

model = SVC()
model.fit(X_train, y_train)

print(model.predict([[4.8,3.0,1.5,0.3]]))
```

Output

[0]

26. Data Science – Machine Learning – Underfitting and Overfitting

Contents

1. Overfitting & Underfitting	2
2. Noise	2
3. Bias.....	2
4. Variance	2
5. Overfitting	3
6. Underfitting	4
7. Good fit model.....	4
8. Good example	5

26. Data Science – Machine Learning – Underfitting and Overfitting

1. Overfitting & Underfitting

- ✓ The main goal of every machine learning model is to **generalize** well.
- ✓ A generalized model provides a suitable output on unknown dataset
 - This means after providing training on the dataset, it can produce reliable and accurate output.
- ✓ Overfitting and Underfitting are the two main problems that occur in machine learning, because of these ML model performances may impact to reduce.

2. Noise

- ✓ Noise means irrelevant data; it reduces the performance of the model.

3. Bias

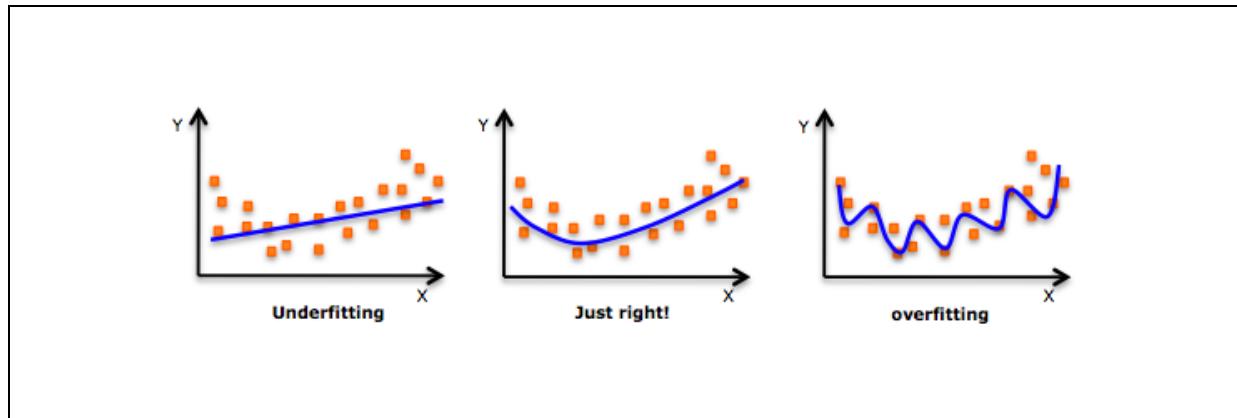
- ✓ Difference between the predicted values and the actual values.

4. Variance

- ✓ Machine learning model performs well with the training dataset, but does not perform well with the test dataset.

5. Overfitting

- ✓ Overfitting is the scenario where a machine learning model cannot generalize or fit well on unseen dataset.
- ✓ The over fitted model has **low bias** and **high variance**.



Note:

- ✓ We can avoid the Overfitting in Model by using cross-Validation, Training with more data, removing features, regularization, Ensemble

6. Underfitting

- ✓ During training if model unable to learn properly then this is called as Underfitting.
- ✓ So it reduces the accuracy and produces unreliable predictions.
- ✓ The under fitted model has **high bias** and **low variance**.

7. Good fit model

- ✓ If model predicts well on training dataset and unseen dataset then it's called as good fit model

8. Good example

- ✓ Assuming that there are three students have prepared for a mathematics examination.
- ✓ First student:
 - Prepared only Addition operations and skipped other math operations from textbook[X]
- ✓ Second student
 - Prepared all math operations from textbook[X]
- ✓ Third student
 - Prepared all math operations from textbook[X].
 - Practiced more on new topics from other math text books[Y, Z]

During exam

- ✓ First student:
 - He can answer only for addition related questions.
- ✓ Second student
 - He can answer to the questions which are from only textbook[X]
- ✓ Third student
 - He can answer to the questions which are from textbook[X, Y, Z]

Comparison

✓ First student	-	Under fitting
✓ Second student	-	Over fitting
✓ Third student	-	Good fit

27. Data Science – Machine Learning – Lasso & Ridge Regression

Contents

1. Linear Regression	2
2. Lasso Regression.....	2
3. Coefficients can be large	2
4. L1 penalty	2
5. L1 Regularization	3
6. How to avoid overfitting issue?	3
7. L1 Regularization and L2 Regularization.....	3
8. Dataset Details: Melbourne house sale price	6
9. Ridge Regression	34
10. Coefficients can be large	34
11. L2 penalty	34

27. Data Science – Machine Learning – Lasso & Ridge Regression

1. Linear Regression

- ✓ Linear Regression is a standard algorithm for regression and it is used to explain the relationship between two variables.
- ✓ Also called as it's a relationship between dependent variable and one or more independent variables.

2. Lasso Regression

- ✓ In linear regression with a single input variable, this relationship is a line, and with **higher dimensions**, this relationship can be hyperplane that connects the input variables to the target variable.
- ✓ The coefficients of the model are found via an optimization process which minimizes error.

3. Coefficients can be large

- ✓ A problem with linear regression, the estimated coefficients of the model can become large and may become model unstable
- ✓ One approach to address the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients.

4. L1 penalty

- ✓ **Lasso Regression** is a popular type of regularized linear regression that includes an L1 penalty.
- ✓ It penalizes a model based on the **sum of the absolute coefficient values**. This is called the L1 penalty.
- ✓ An L1 penalty minimizes the size of all coefficients and some coefficients to be minimized to the value zero, effectively removing input features from the model.

5. L1 Regularization

- ✓ A regression model that uses **L1** regularization technique is called **Lasso** Regression.
- ✓ Lasso full form is, Least Absolute Shrinkage and Selection Operator

- ✓ Minimization objective = LS Obj + $\alpha * (\text{sum of the absolute value of coefficients})$

6. How to avoid overfitting issue?

- ✓ Regularization is an important concept that is used to avoid overfitting of the data,
- ✓ We can address this issue by using L1 and L2 Regularization
- ✓ Regularization is implemented by adding a “penalty” term to the best fit derived from the trained data, to achieve a *lesser variance*

7. L1 Regularization and L2 Regularization

- ✓ When you have a large number of features in your dataset, some of the Regularization techniques used to address over-fitting.

Program Name Importing required libraries
demo1.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

print("Importing required libraries")
```

Output

Importing required libraries

Program Name Loading the dataset
demo2.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print(dataset.head())
```

Output

```
Suburb      Address  Rooms  ...  Longitude      Regionname  Propertycount
0  Abbotsford    68 Studley St    2  ...  144.9958  Northern Metropolitan  4019.0
1  Abbotsford    85 Turner St    2  ...  144.9984  Northern Metropolitan  4019.0
2  Abbotsford   25 Bloomburg St    2  ...  144.9934  Northern Metropolitan  4019.0
3  Abbotsford  18/659 Victoria St    3  ...  145.0116  Northern Metropolitan  4019.0
4  Abbotsford      5 Charles St    3  ...  144.9944  Northern Metropolitan  4019.0

[5 rows x 21 columns]
```

8. Dataset Details: Melbourne house sale price

- ✓ The dataset is about the housing market in Melbourne and contains information about the house sale price
- ✓ Notes on Specific Variables
 - Rooms: Number of rooms
 - Price: Price in dollars
 - Method: S - property sold; SP - property sold prior; PI - property passed in; PN - sold prior not disclosed; SN - sold not disclosed; NB - no bid; VB - vendor bid; W - withdrawn prior to auction; SA - sold after auction; SS - sold after auction price not disclosed. N/A - price or highest bid not available.
 - Type: br - bedroom(s); h - house, cottage, villa, semi, terrace; u - unit, duplex; t - townhouse; dev site - development site; o res - other residential.
 - SellerG: Real Estate Agent
 - Date: Date sold
 - Distance: Distance from CBD
 - Region name: General Region (West, North West, North, North east ...etc)
 - Property count: Number of properties that exist in the suburb (an outlying district of a city, especially a residential one.).
 - Bedroom2 : Scrapped # of Bedrooms (from different source)
 - Bathroom: Number of Bathrooms
 - Car: Number of cars/pots
 - Landsize: Land Size
 - BuildingArea: Building Size
 - Council Area: Governing council for the area

Program Name Rows and columns in DataFrame
demo3.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print("Rows and columns:", dataset.shape)
```

Output

Rows and columns: (34857, 21)

Program Name Unique values
demo4.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print(dataset.nunique())
```

Output

```
Suburb          351
Address         34009
Rooms           12
Type            3
Price           2871
Method          9
SellerG         388
Date             78
Distance        215
Postcode        211
Bedroom2        15
Bathroom        11
Car              15
Landsize        1684
BuildingArea    740
YearBuilt       160
CouncilArea     33
Latitude        13402
Longitude       14524
Regionname      8
Propertycount   342
dtype: int64
```

Program Name

Get the required columns

demo5.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print(dataset.shape)

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
               'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
               'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

print(dataset.head())
```

Output

```
      Suburb  Rooms  Type Method SellerG ... Bathroom  Car  Landsize BuildingArea  Price
0  Abbotsford     2     h    SS  Jellis   ...      1.0  1.0    126.0        NaN      NaN
1  Abbotsford     2     h     S  Biggin   ...      1.0  1.0    202.0        NaN  1480000.0
2  Abbotsford     2     h     S  Biggin   ...      1.0  0.0    156.0      79.0  1035000.0
3  Abbotsford     3     u    VB  Rounds   ...      2.0  1.0      0.0        NaN      NaN
4  Abbotsford     3     h    SP  Biggin   ...      2.0  0.0    134.0     150.0  1465000.0

[5 rows x 15 columns]
```

Program Name Rows and columns in DataFrame
demo6.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]
print("Rows and columns:", dataset.shape)
```

Output

Rows and columns: (34857, 15)

Program Name Checking NaN values
demo7.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

print(dataset.isna().sum())
```

Output

```
Suburb          0
Rooms           0
Type            0
Method           0
SellerG          0
Regionname       3
Propertycount    3
Distance         1
CouncilArea      3
Bedroom2        8217
Bathroom        8226
Car              8728
Landsize        11810
BuildingArea     21115
Price            7610
dtype: int64
```

Program Name Few of the columns filling with zero
demo8.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

print(dataset.isna().sum())
```

Output

```
Suburb          0
Rooms           0
Type            0
Method           0
SellerG          0
Regionname       3
Propertycount    0
Distance          0
CouncilArea       3
Bedroom2          0
Bathroom          0
Car              0
Landsize        11810
BuildingArea     21115
Price            7610
dtype: int64
```

Program Name Filling Landsize and BuildingArea columns with mean value
demo9.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

print(dataset.isna().sum())
```

Output

```
Suburb          0
Rooms           0
Type            0
Method          0
SellerG         0
Regionname      3
Propertycount   0
Distance        0
CouncilArea     3
Bedroom2        0
Bathroom        0
Car              0
Landsize         0
BuildingArea    0
Price           7610
dtype: int64
```

Program Name Dropping NaN values
demo10.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

print(dataset.isna().sum())
```

Output

```
Suburb      0
Rooms       0
Type        0
Method      0
SellerG     0
Regionname  0
Propertycount 0
Distance    0
CouncilArea 0
Bedroom2    0
Bathroom    0
Car         0
Landsize    0
BuildingArea 0
Price       0
dtype: int64
```

Program Name Creating dummy variables for characters data
demo11.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

print(dataset.head())
```

Output

```
   Rooms  PropertyCount ... CouncilArea_Yarra City Council CouncilArea_Yarra Ranges Shire Council
1      2          4019.0 ...
2      2          4019.0 ...
4      3          4019.0 ...
5      3          4019.0 ...
6      4          4019.0 ...
[5 rows x 745 columns]
```

Program Name Creating features and labels
demo12.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']

print("Created features and labels")
```

Output
Created features and labels

Program Name Splitting training and testing datasets
demo13.py

```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
print("Splitting train and test datasets")
```

Output

Splitting train and test datasets

Program Name

Creating LinearRegression model and training
demo14.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =
```

```
0.3, random_state=2)

reg = LinearRegression()

print("Created LinearRegression model and training")

reg.fit(train_X, train_y)
```

Output

Created LinearRegression model

Program Name Creating LinearRegression model
demo15.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
reg = LinearRegression()  
  
reg.fit(train_X, train_y)  
  
print("Training LinearRegression model")
```

Output

Training LinearRegression model

Program Name Linear Regression: Training, training dataset score
demo16.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
reg = LinearRegression()  
reg.fit(train_X, train_y)  
  
print("Training dataset score is:")  
print(reg.score(train_X, train_y))
```

Output

```
Training dataset score is:  
0.6827792395792723
```

Program Name Linear Regression: Training, test dataset score
demo17.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
reg = LinearRegression()  
reg.fit(train_X, train_y)  
  
print("Creating Linear Regression model and training with test  
dataset:")  
print(reg.score(test_X, test_y))
```

Output

Creating Linear Regression model and training with test dataset
0.1385368316157145

Note

- ✓ If training score is very good and test score is very low then it called as over fit

Program Name Lasso Regression
demo18.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
lasso_reg = Lasso(alpha = 50, max_iter = 100, tol = 0.1)  
lasso_reg.fit(train_X, train_y)  
  
print("Creating Lasso Regression model and training with train  
dataset")  
print(lasso_reg.score(train_X, train_y))
```

Output

```
Creating Lasso Regression model and training with train dataset  
0.6766985624766824
```

Program Name Lasso Regression: Training, testing dataset score
demo19.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
lasso_reg = Lasso(alpha = 50, max_iter = 100, tol = 0.1)  
lasso_reg.fit(train_X, train_y)  
  
print("Creating Lasso Regression model checking test dataset  
score")  
print(lasso_reg.score(test_X, test_y))
```

Output

```
Creating Lasso Regression model checking test dataset score  
0.6636111369404489
```

9. Ridge Regression

- ✓ In linear regression with a single input variable, this relationship is a line, and with **higher dimensions**, this relationship can be hyperplane that connects the input variables to the target variable.
- ✓ The coefficients of the model are found via an optimization process which minimize error.

10. Coefficients can be large

- ✓ A problem with linear regression, the estimated coefficients of the model can become large and may become model unstable
- ✓ One approach to address the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients.

11. L2 penalty

- ✓ **Ridge Regression** is a popular type of regularized linear regression that includes an **L2** penalty.
- ✓ It penalize a model based on **sum of the squared coefficient value**. This is called the L2 penalty.
- ✓ An L2 penalty minimizes the size of all coefficients and some coefficients to be minimized to the value zero, effectively removing input features from the model.

- ✓ Minimization objective = LS Obj + $\alpha * (\text{sum of square of coefficients})$

Program Name Ridge Regression: Training, training dataset score
demo20.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
ridge_reg = Ridge(alpha = 50, max_iter = 100, tol = 0.1)  
ridge_reg.fit(train_X, train_y)  
  
print("Ridge Regression model score with train dataset:")  
print(ridge_reg.score(train_X, train_y))
```

Output

```
Ridge Regression model score with train dataset:  
0.6670848945194958
```

Program Name Ridge Regression: Training, testing dataset score
demo21.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
ridge_reg = Ridge(alpha=50, max_iter=100, tol=0.1)  
ridge_reg.fit(train_X, train_y)  
  
print("Ridge Regression model score with test dataset:")  
print(ridge_reg.score(test_X, test_y))
```

Output

```
Ridge Regression model score with test dataset:  
0.6670848945194958
```

28. Data Science – Machine Learning – K – Means Clustering

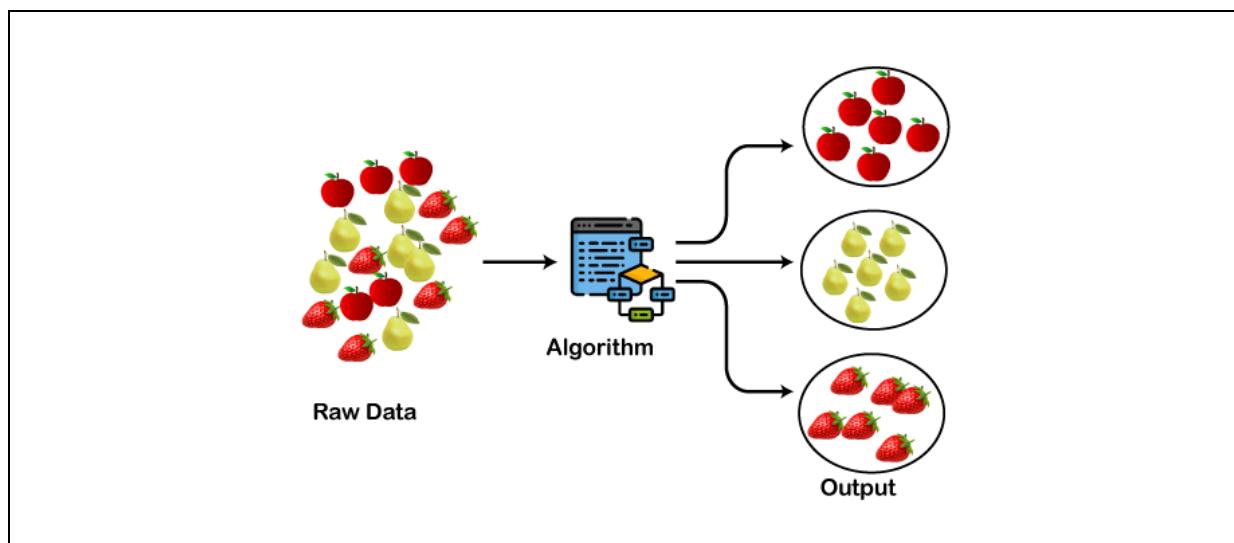
Contents

1. Clustering	2
2. K means clustering algorithm	2
3. Steps in K-Means Algorithm	3
4. Scenario	4
5. How to determine the correct number of clusters?	8
6. Elbow Method	10

28. Data Science – Machine Learning – K – Means Clustering

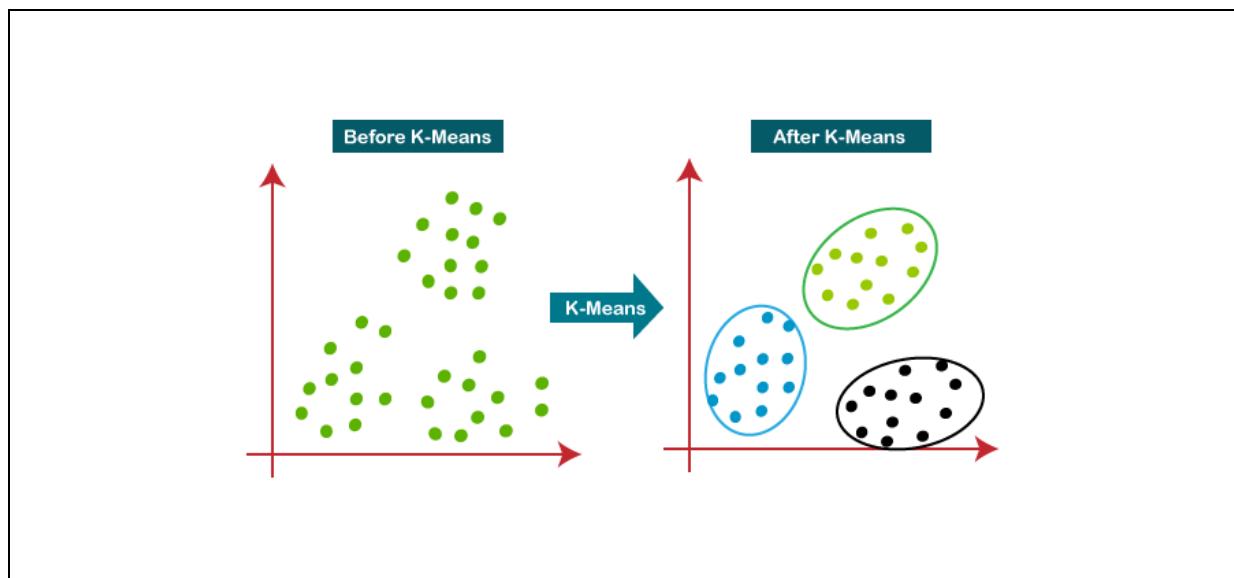
1. Clustering

- ✓ Clustering or cluster analysis is a machine learning technique.
- ✓ It groups the unlabelled dataset.
- ✓ It is a way of grouping the data points into different clusters based on similarities.



2. K means clustering algorithm

- ✓ K-Means Clustering is an Unsupervised Learning algorithm.
- ✓ This algorithm groups the unlabeled dataset into different clusters based on similar properties.
- ✓ Here K defines the number of pre-defined clusters that need to be created in the process,
 - If $K = 2$ then there will be two clusters,
 - If $K = 3$ then there will be three clusters etc.
- ✓ It is a centroid-based algorithm, where each cluster is associated with a centroid.

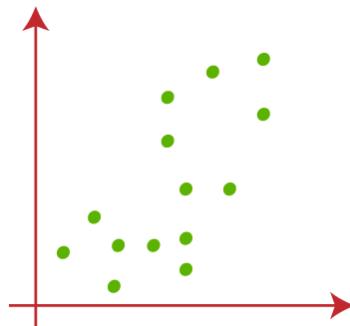


3. Steps in K-Means Algorithm

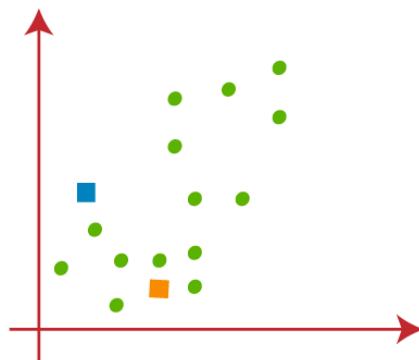
- ✓ Step-1: Select the number K to decide the number of clusters.
- ✓ Step-2: Select random K points or centroids.
- ✓ Step-3: Assign each data point to their closest centroid, which will form K clusters.
- ✓ Step-4: Calculate the variance and place a new centroid of each cluster.
- ✓ Step-5: Repeat the initial 3 steps, which mean reassign each data point to the new closest centroid of each cluster.
- ✓ Step-6: If any reassignment occurs, then go to step-4 else FINISH.
- ✓ Step-7: The model is ready.

4. Scenario

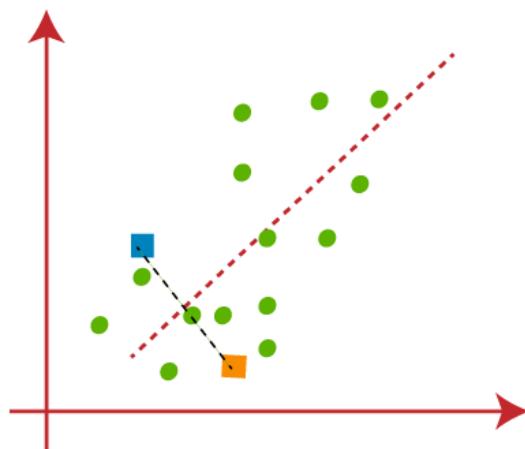
- ✓ Assuming that we have scattered two variables in x and y axis



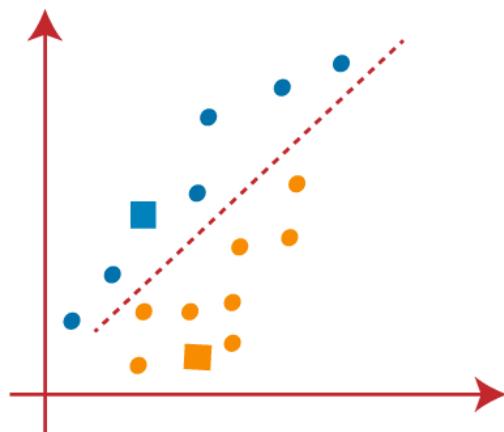
- ✓ Let's take some random k points or centroid to form the cluster.
- ✓ These points can be either the points from the dataset or any other point.
- ✓ So, here we are selecting the below two points as k points, which are not the part of our dataset.



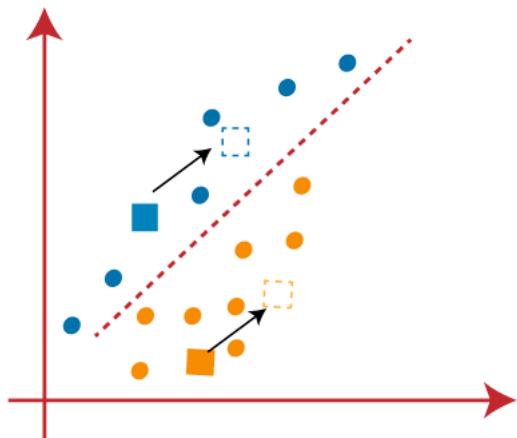
- ✓ Now we will assign each data point of the scatter plot to its closest K-point or centroid.
- ✓ Let's compute the distance between two points.
- ✓ So, we will draw a median between both the centroids.



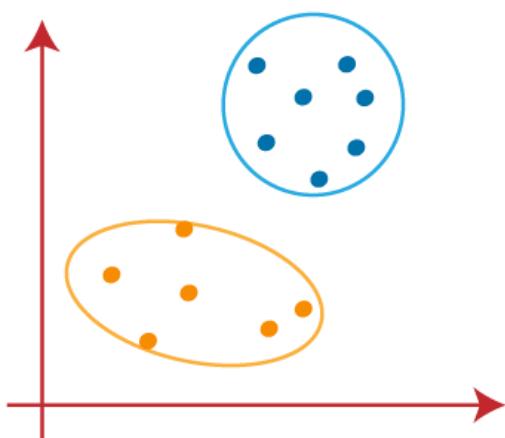
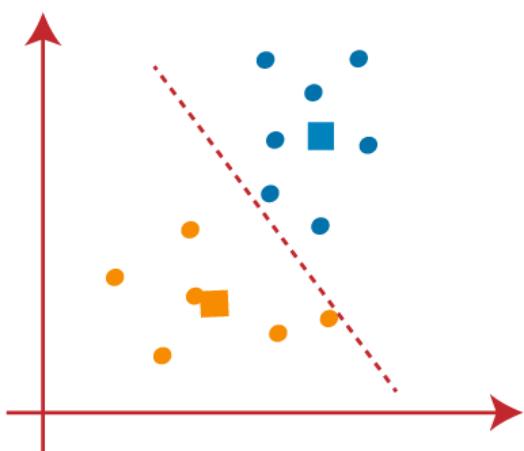
- ✓ From the above image, it is clear that points left side of the line is near to the K1 or blue centroid.
- ✓ Points to the right of the line are close to the yellow centroid.



- ✓ As we need to find the closest cluster, so we will repeat the process by choosing a new centroid.
- ✓ To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids.

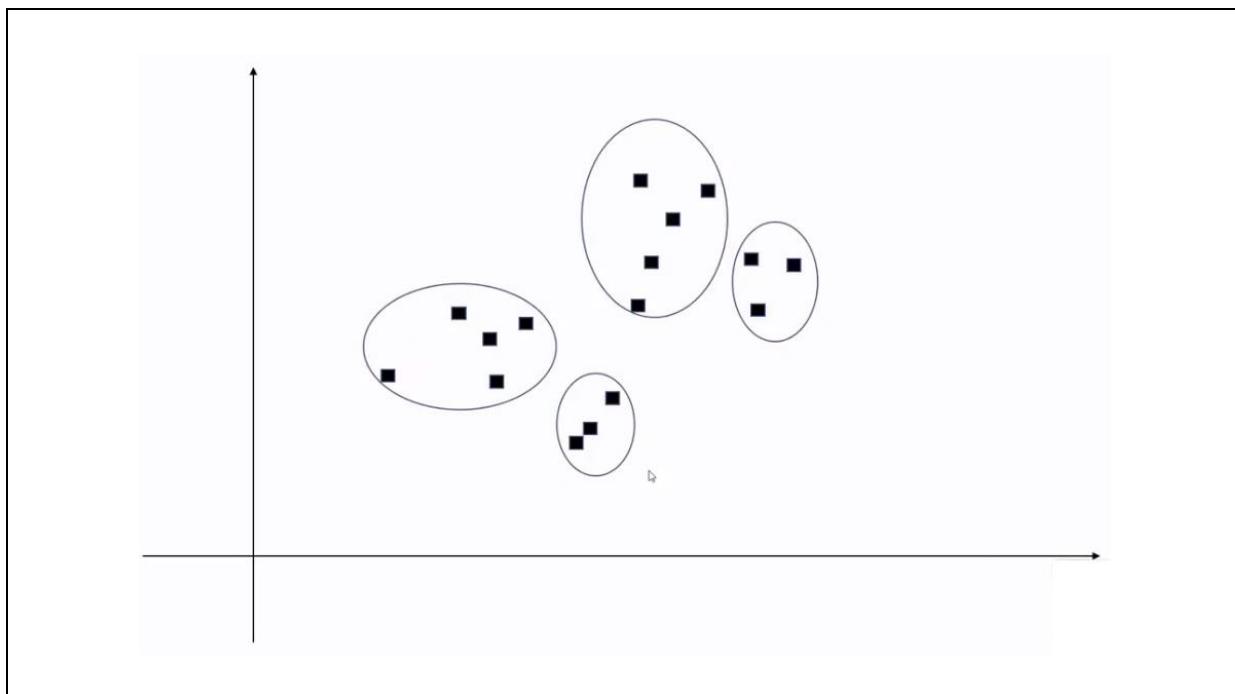
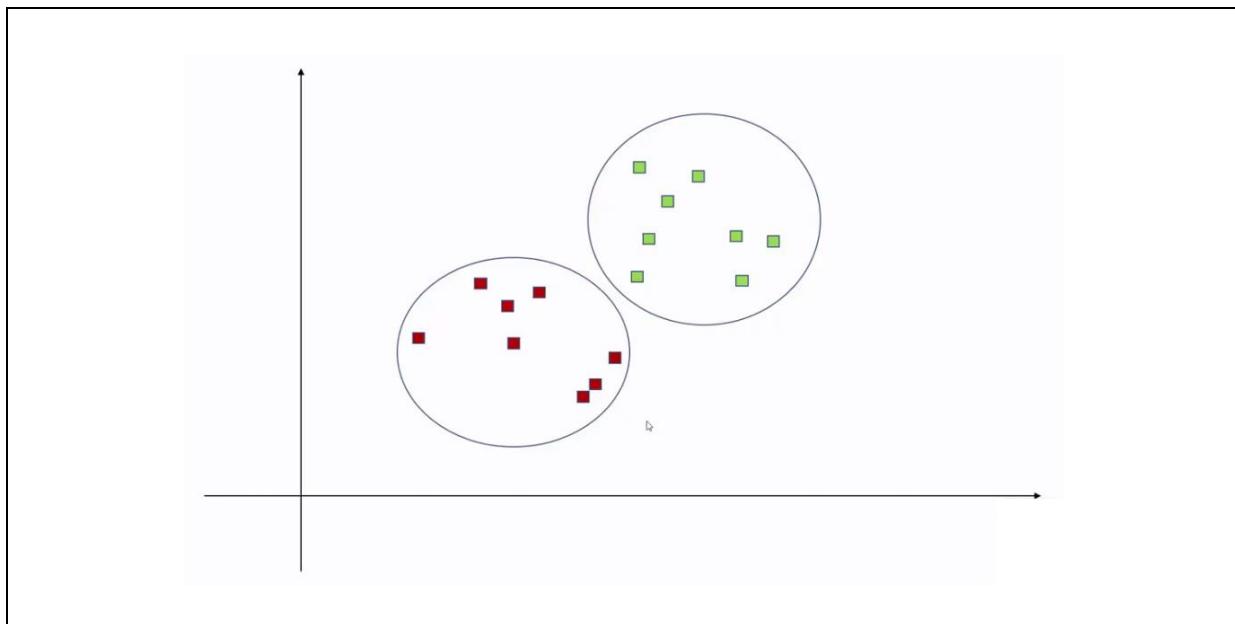


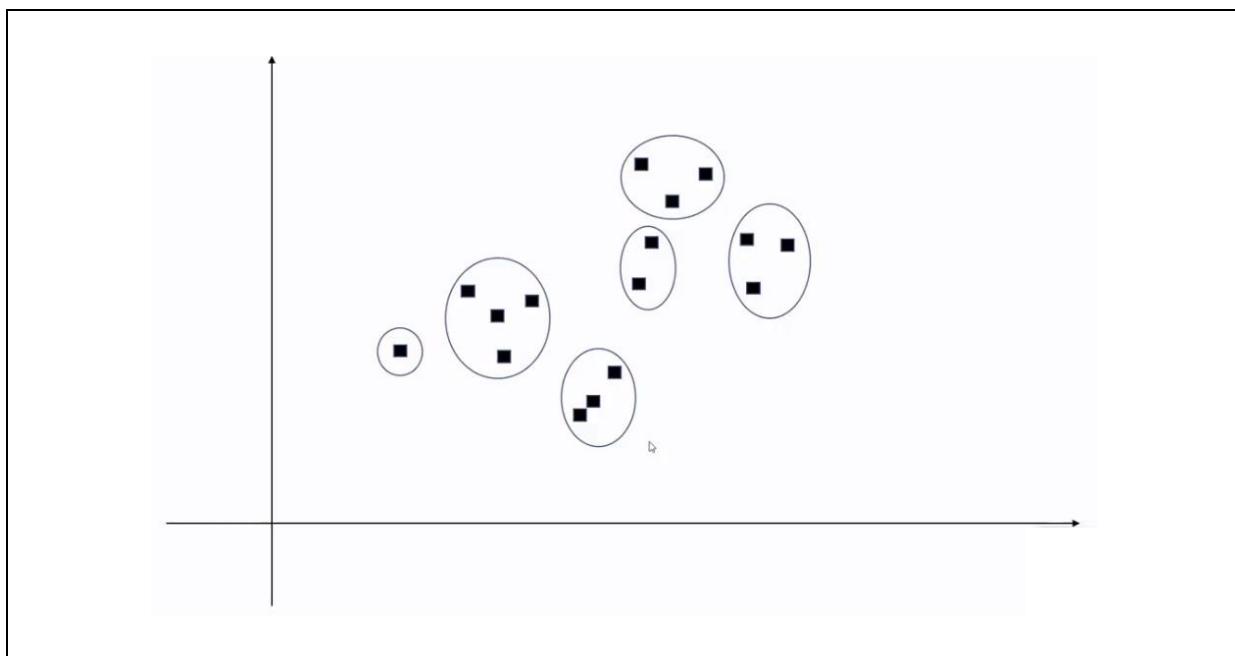
- ✓ As we got the new centroids so again will draw the median line and reassign the data points.



5. How to determine the correct number of clusters?

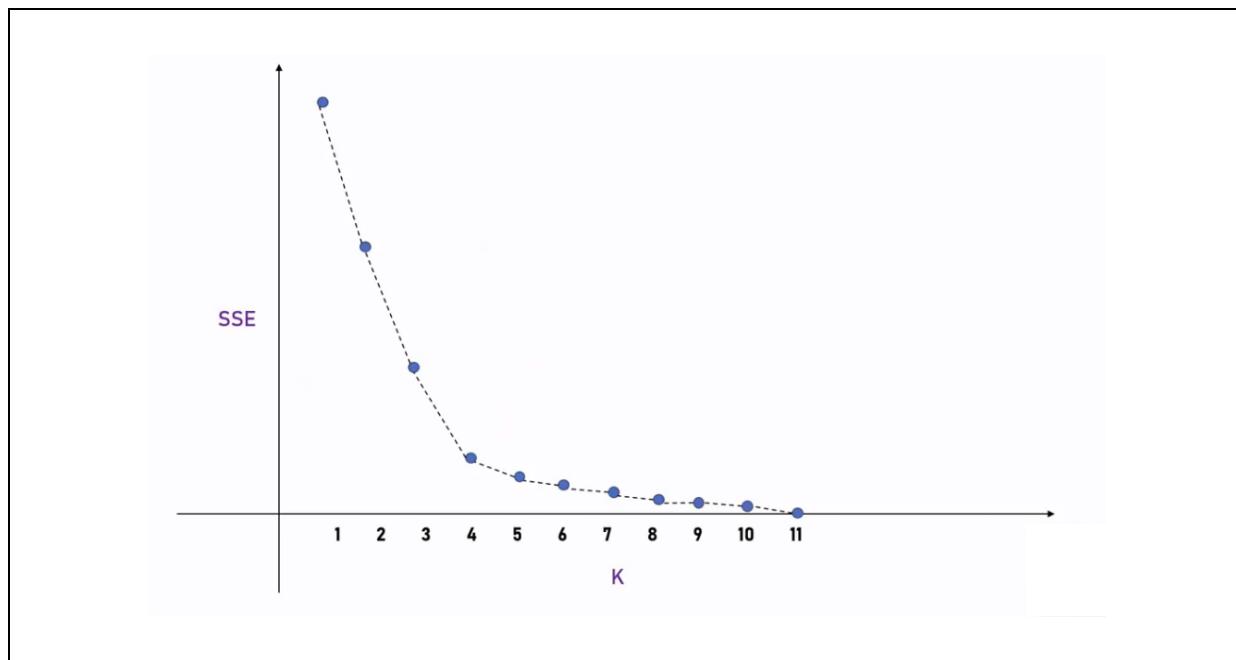
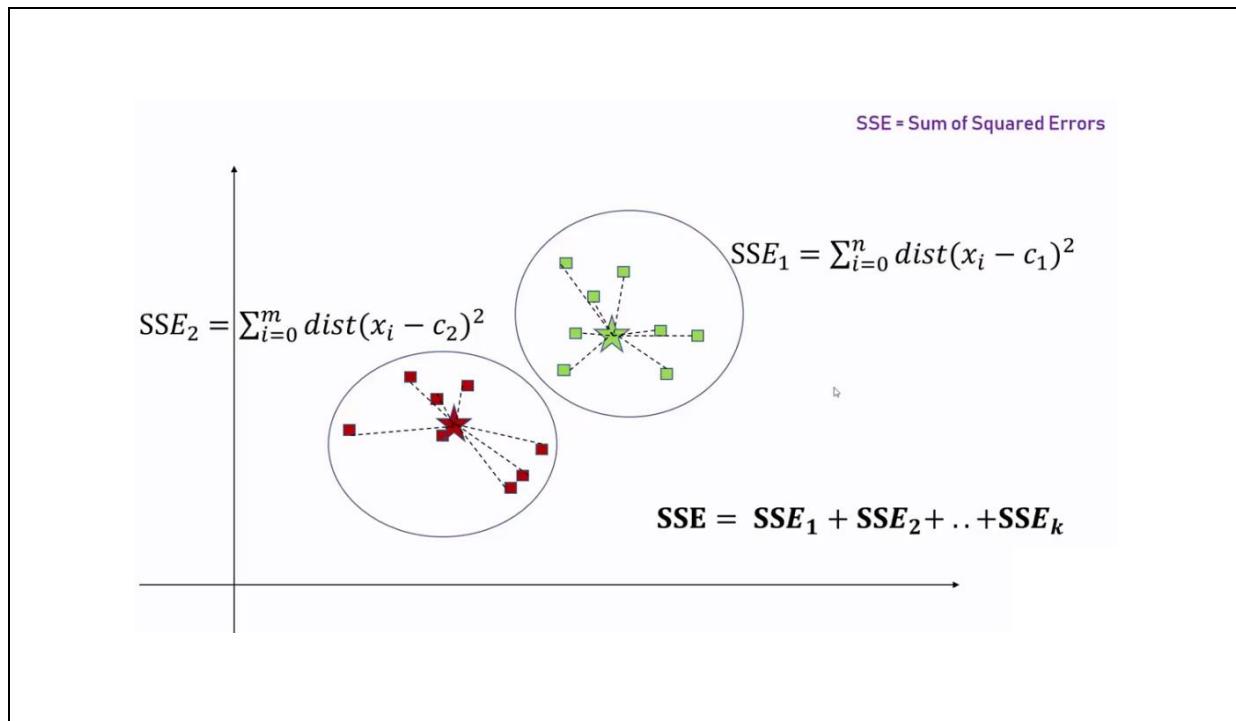
- ✓ Let's take few scenarios

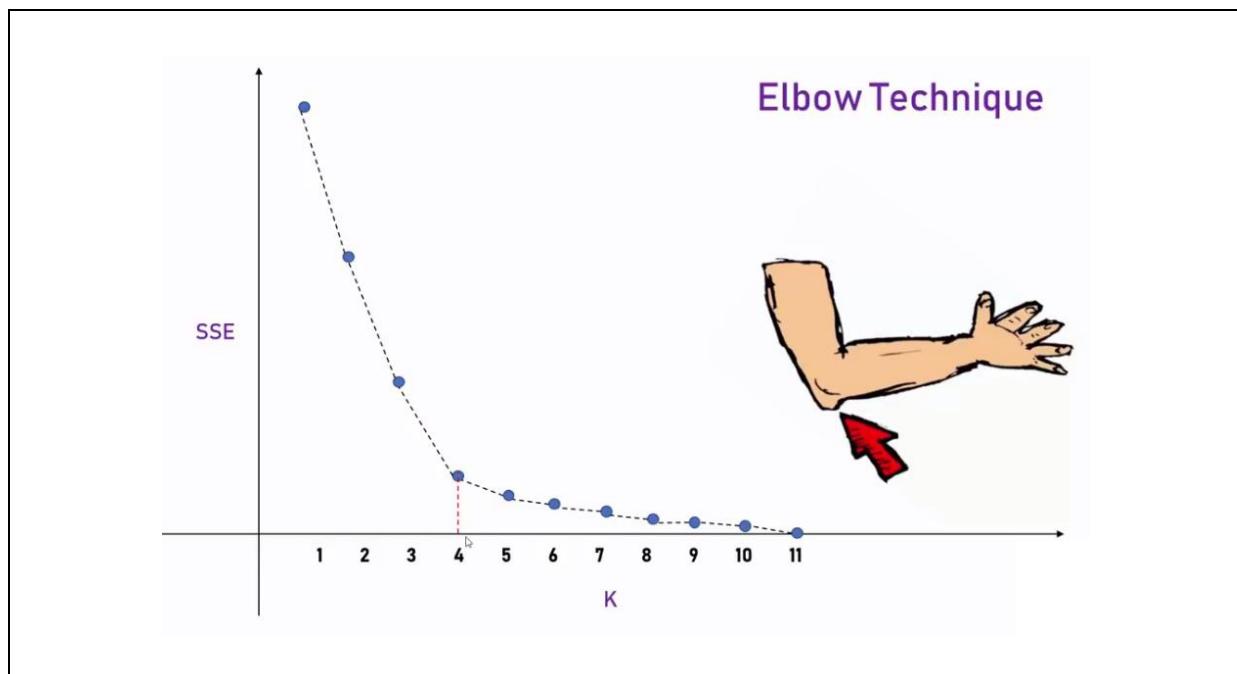




6. Elbow Method

- ✓ The Elbow method is one of the most popular ways to find the optimal number of clusters.
- ✓ This method uses the concept of Cluster Sum of Squares.
- ✓ It creates the total variations within a cluster.





Program Name Loading the dataset
demo1.py

```
import pandas as pd

df = pd.read_csv("income.csv")

print(df.head())
```

Output

	Name	Age	Income(\$)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

Program Name Plotting the data
demo2.py

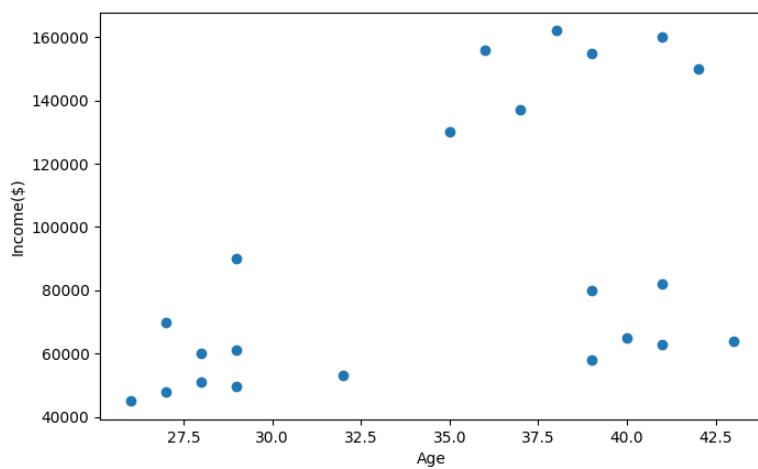
```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("income.csv")

plt.scatter(df.Age, df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')

plt.show()
```

Output



Program Name Creating clusters
demo3.py

```
import pandas as pd
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])
print(y_predicted)
```

Output

```
[1 1 2 2 0 0 0 0 0 0 2 2 2 2 2 2 2 1 1 2]
```

Program Name Predicting the cluster
demo4.py

```
import pandas as pd
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])
df['cluster']=y_predicted

print(df.head())
```

Output

	Name	Age	Income(\$)	cluster
0	Rob	27	70000	2
1	Michael	29	90000	2
2	Mohan	29	61000	0
3	Ismail	28	60000	0
4	Kory	42	150000	1

Program Name Cluster distance
demo5.py

```
import pandas as pd
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])
df['cluster']=y_predicted

print(km.cluster_centers_)
```

Output

```
[[3.29090909e+01 5.61363636e+04]
 [3.82857143e+01 1.50000000e+05]
 [3.40000000e+01 8.05000000e+04]]
```

Program Name Plotting the clusters
demo6.py

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])
df['cluster'] = y_predicted

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]

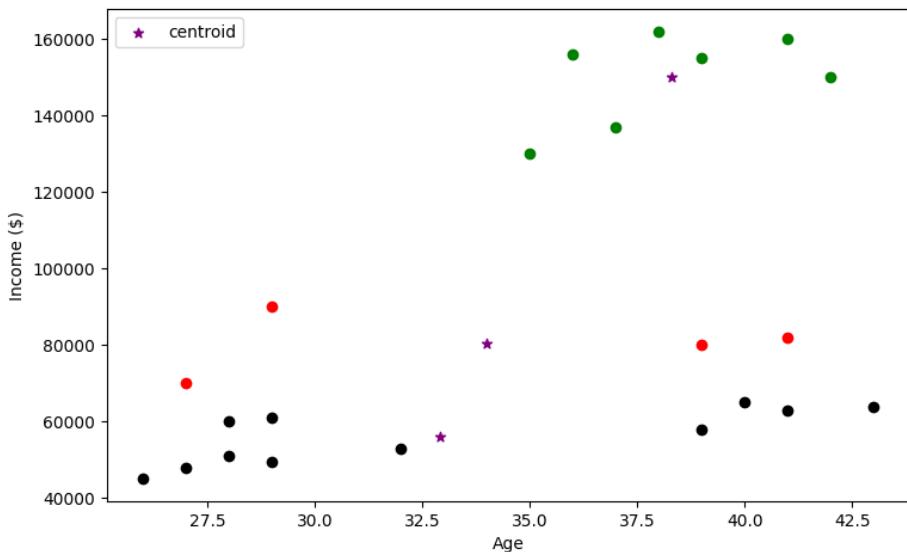
plt.scatter(df1.Age, df1['Income($)', color='green'])
plt.scatter(df2.Age, df2['Income($)', color='red'])
plt.scatter(df3.Age, df3['Income($)', color='black'])

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
            color = 'purple', marker='*', label='centroid')

plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()

plt.show()
```

Output



Program Name Features scaling
demo7.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

print(df.head())
```

Output

	Name	Age	Income(\$)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436

Program Name Plotting after features scaling
demo8.py

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

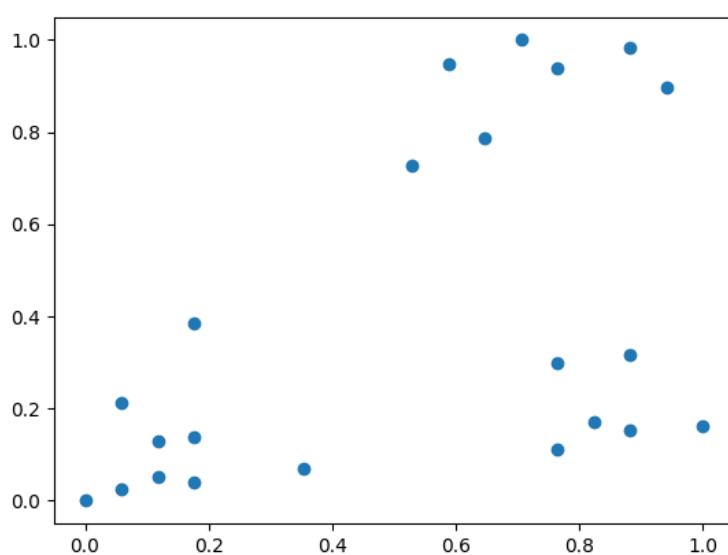
scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

plt.scatter(df.Age, df['Income($)])

plt.show()
```

Output



Program Name Prediction
demo9.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters = 3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])
print(y_predicted)
```

Output

```
[1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 0 0 0 0 0]
```

Program Name Prediction demo10.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])

df['cluster'] = y_predicted
print(df.head())
```

Output

	Name	Age	Income(\$)	cluster
0	Rob	0.058824	0.213675	1
1	Michael	0.176471	0.384615	1
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
4	Kory	0.941176	0.897436	0

Program Name Cluster distance
demo11.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])

df['cluster']=y_predicted
print(km.cluster_centers_)
```

Output

```
[[0.1372549  0.11633428]
 [0.72268908 0.8974359 ]
 [0.85294118 0.2022792 ]]
```

Program Name Plotting the clusters
demo12.py

```
import pandas as pd
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])

df['cluster']=y_predicted

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]

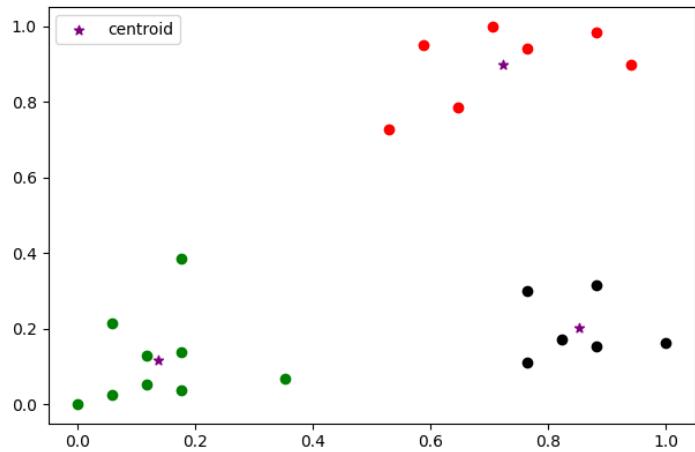
plt.scatter(df1.Age,df1['Income($)'),color='green')
plt.scatter(df2.Age,df2['Income($)'),color='red')
plt.scatter(df3.Age,df3['Income($)'),color='black')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
           color='purple', marker='*',label='centroid')

plt.legend()

plt.show()
```

Output



Program Name Elbow method
demo13.py

```
import pandas as pd
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

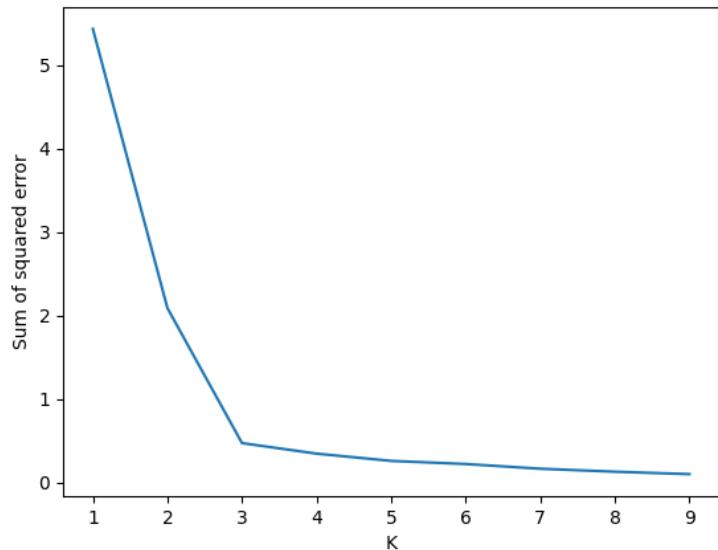
sse = []
k_rng = range(1,10)

for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age', 'Income($)']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng, sse)

plt.show()
```

Output



29. Data Science – Machine Learning – K Nearest Neighbor

Contents

1. K-Nearest Neighbor Algorithm	2
2. How it works?	2
3. Scenario	3
4. Use case	5

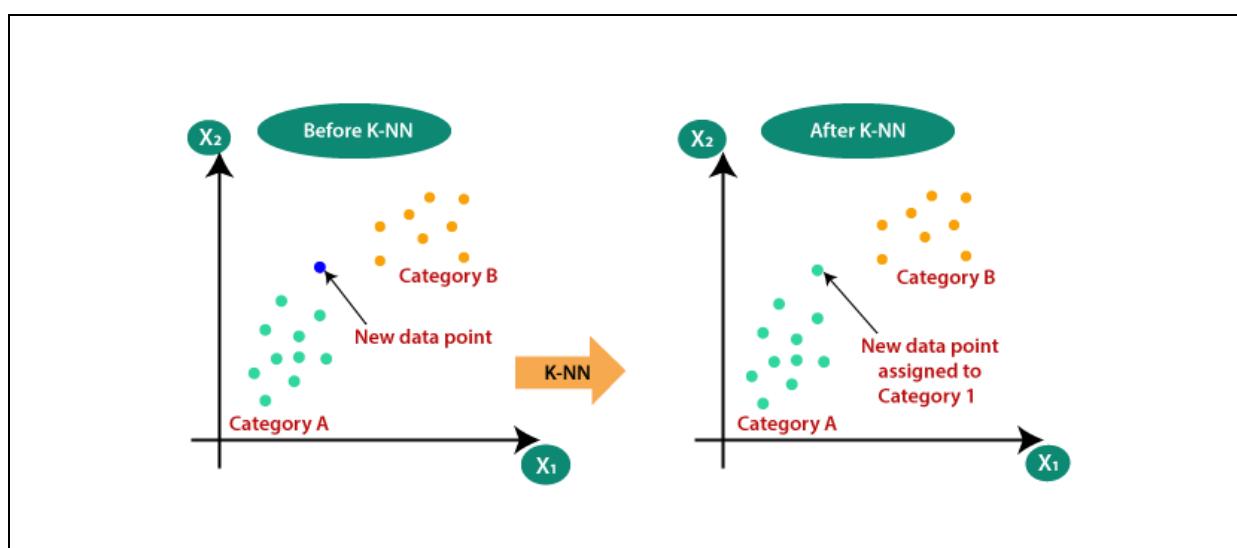
29. Data Science – Machine Learning – K Nearest Neighbor

1. K-Nearest Neighbor Algorithm

- ✓ K-Nearest Neighbor is a Supervised Learning technique.
- ✓ K-NN algorithm follow one basic rule that is, similar things are near to each other.
- ✓ It is also called a lazy learner algorithm because it does not learn from the training set immediately.
 - At the time of training phase this algorithm just stores the dataset
 - Whenever we get new data point then it classifies the category

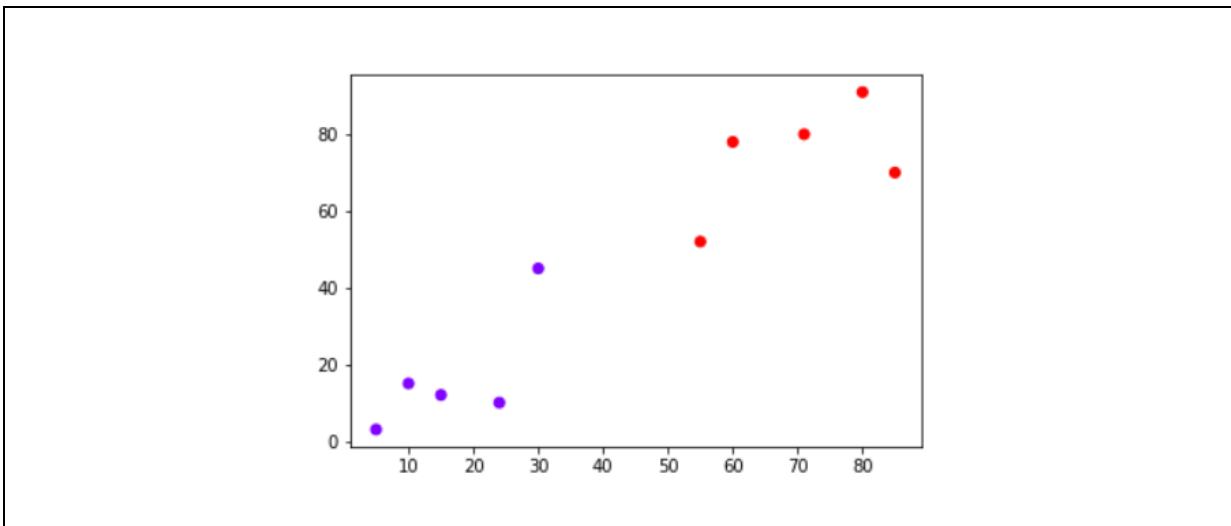
2. How it works?

- ✓ It simply calculates the distance of a new data point to all other training data points.
- ✓ The distance can be of any type e.g. Euclidean or Manhattan etc.
- ✓ It selects the K-nearest data points.
- ✓ Finally it assigns the data point to the class to which the majority of the K data points belong.

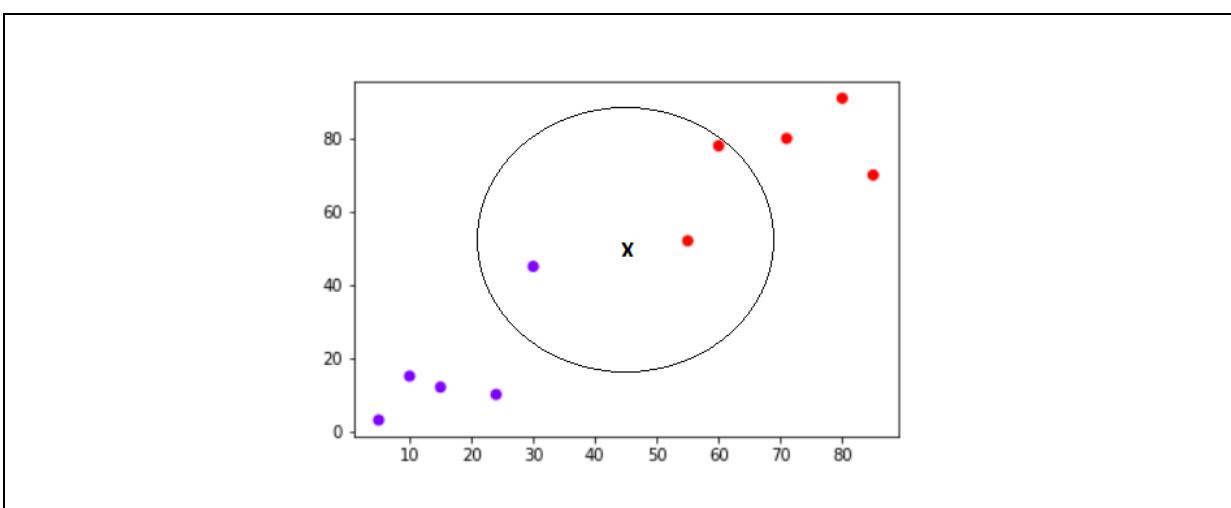


3. Scenario

- ✓ Suppose you have a dataset with two variables, which when plotted, looks like the one in the following figure.



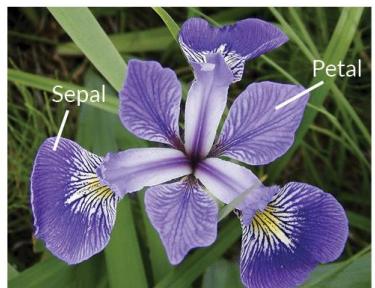
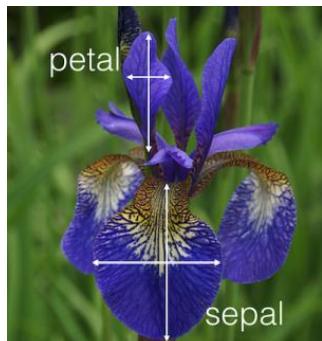
- ✓ Our task is to classify a new data point with 'X' into "Blue" class or "Red" class.
- ✓ Suppose the value of K is 3.
- ✓ The KNN algorithm starts by calculating the distance of point X from all the points.
- ✓ It then finds the 3 nearest points with least distance to point X.
- ✓ This is shown in the figure below; the three nearest points have been encircled.



- ✓ The final step of the KNN algorithm is to assign new point to the class to which majority of the three nearest points belong.
- ✓ From the above image we can see that the two of the three nearest points belong to the class "Red" while one belongs to the class "Blue".
- ✓ Therefore the new data point will be classified as "Red".

4. Use case

- ✓ Assuming that Abhi had a hobby which is interested in distinguishing the species of some iris flowers that he has found
- ✓ He has collected some measurements associated with each iris, which are:
 - The **length** and **width** of the **petals**
 - The **length** and **width** of the **sepals**, all measured in centimetres.
- ✓ She also has the measurements of some irises that have been previously identified to the species
 - setosa,
 - versicolor
 - virginica
- ✓ The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known.
- ✓ So that we can predict the species for the new irises that she has found.



Iris Versicolor



Iris Setosa



Iris Virginica

Flower codes

- ✓ Setosa - 0
- ✓ Versicolor - 1
- ✓ Virginica - 2

Program Name Loading iris dataset
demo1.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(dir(iris))
```

Output

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target',
'target_names']
```

Program Name Displaying feature names
demo2.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
```

Output

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Program Name Displaying target names
demo3.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.target_names)
```

Output

```
['setosa' 'versicolor' 'virginica']
```

Program Name Displaying data
demo4.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.data)
```

Output

```
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```

Program Name Length of the data
demo5.py

```
from sklearn.datasets import load_iris  
  
iris = load_iris()  
  
print(len(iris.data))
```

Output
150

Program Name Create a Dataframe by using data and features
demo6.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

print(df)
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

Program Name Adding target column to the dataframe
demo7.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Program Name Displaying target == 0 flowers
demo8.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==0].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Program Name Displaying length of the target == 0 flowers
demo9.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==0]))
```

Output

50

Program Name Displaying target == 1 flowers
demo10.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==1].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

Program Name Displaying length of the target == 0 flowers
demo11.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==1]))
```

Output

50

Program Name Displaying target == 2 flowers
demo12.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==2].head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

Program Name Displaying length of the target == 2 flowers
demo13.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==2]))
```

Output

50

Program Name Displaying the flower names
demo14.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
print(df)
```

Output

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target  flower_name
0           5.1          3.5            1.4            0.2      0    setosa
1           4.9          3.0            1.4            0.2      0    setosa
2           4.7          3.2            1.3            0.2      0    setosa
3           4.6          3.1            1.5            0.2      0    setosa
4           5.0          3.6            1.4            0.2      0    setosa
..          ...
145          6.7          3.0            5.2            2.3      2  virginica
146          6.3          2.5            5.0            1.9      2  virginica
147          6.5          3.0            5.2            2.0      2  virginica
148          6.2          3.4            5.4            2.3      2  virginica
149          5.9          3.0            5.1            1.8      2  virginica
[150 rows x 6 columns]
```

Program Name All setosa flowers
demo15.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

setosa_50 = df[:50]
print(setosa_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Program Name All versicolor flowers
demo16.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

versicolor_50 = df[50:100]
print(versicolor_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

Program Name All virginica flowers
demo17.py

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

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

verginica_50 = df[100:]
print(verginica_50.head())
```

Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

Program Name Splitting the data
 demo18.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

print("Splitting the data")
```

Output

Splitting the data

Program Name Model training
demo19.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

# Train Using K Neighbor classifier

classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)

print('Model got trained')
```

Output

Model got trained

Program Name Model score
demo20.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

print(classifier.score(X_test, y_test))
```

Output

0.9666666666666667

Program Name Model prediction
demo21.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

print(classifier.predict([[4.8, 3.0, 1.5, 0.3]]))
```

Output

[0]

Program Name Model prediction
demo22.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print(y_pred)
```

Output

```
[0 1 0 1 2 1 0 1 2 0 0 2 1 0 1 0 0 0 0 1 1 0 2 2 0 2 0 0 1 0]
```

Program Name Model evaluation
demo22.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,
confusion_matrix

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

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

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Output

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	1.00	0.78	0.88	9
2	0.80	1.00	0.89	8
accuracy			0.93	30
macro avg	0.93	0.93	0.92	30
weighted avg	0.95	0.93	0.93	30

30. Data Science – Machine Learning – Naïve Bayes Classifier

Contents

1. Naive Bayes Classifier.....	2
2. Why is it called Naïve Bayes?.....	2
3. Bayes theorem.....	3
4. Scenario	4
5. Conditional probability	6
6. Use case	9
7. Math problem and solution.....	10
8. Use cases.....	12

30. Data Science – Machine Learning – Naïve Bayes Classifier

1. Naïve Bayes Classifier

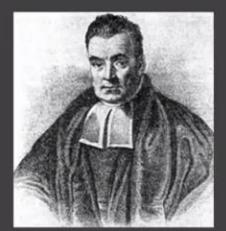
- ✓ Naïve Bayes algorithm is a supervised learning algorithm.
- ✓ This is based on Bayes theorem and used for solving classification problems.
- ✓ It is mainly used in text classification.
 - Examples like spam filtration, Sentimental analysis, and classifying articles etc.

2. Why is it called Naïve Bayes?

- ✓ **Naive:**
 - It is called Naive because it assumes that the occurrence of a certain feature is independent of the other features.
 - Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple.
 - Hence each feature individually contributes to identify that it is an apple without depending on each other.
- ✓ **Bayes:**
 - It is called Bayes because it depends on the principle of Bayes' Theorem.

3. Bayes theorem

- ✓ Bayes' theorem is also known as Bayes' Rule or Bayes' law.
- ✓ It is used to determine the probability of a hypothesis with prior knowledge.
- ✓ It depends on the conditional probability.



Thomas Bayes

$$P(A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

4. Scenario

- ✓ When we flip a coin, the probability of getting head or tail is $1/2$, because there are two possibilities of outcomes
- ✓ So the chance of getting head or tail is 50%



$$p(\text{head}) = 1/2$$

Pick a random card, what is the probability of getting a queen?



4 queens, 52 total cards

$$P(\text{queen}) = 4/52 = 1/13$$

5. Conditional probability

Pick a random card, you know it is a diamond. Now what is the probability of that card being a queen?



Total diamonds = 13

Queen = 1



Total diamonds = 13

Queen = 1

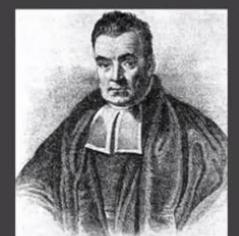
$P(\text{queen/diamond}) = 1/13$

Conditional Probability



$P(\text{queen/diamond}) = 1/13$

$P(A/B) = \text{Probability of event A knowing}$
 $\text{that event B has already occurred}$



Thomas Bayes

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

$$P(\text{queen/diamond}) = \frac{P(\text{diamond/queen}) * P(\text{queen})}{P(\text{diamond})}$$
$$\begin{aligned} P(\text{diamond/queen}) &= 1/4 &= \frac{1/4 * 1/13}{1/4} \\ P(\text{queen}) &= 1/13 &= 1/13 \end{aligned}$$

6. Use case

Passenger Id	Name	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
1	Braund, Mr. Owen Harris	3	male	22	1	0	21171	7.25	S		0
2	Cumings, Mrs. John Bradley	1	female	38	1	0	17599	71.2833	C85	C	1
3	Heikkinen, Miss. Laina	3	female	26	0	0	3101282	7.925		S	1
4	Futrelle, Mrs. Jacques Heath	1	female	35	1	0	113803	53.1	C123	S	1
5	Allen, Mr. William Henry	3	male	35	0	0	373450	8.05		S	0
6	Moran, Mr. James	3	male		0	0	330877	8.4583	Q		0
7	McCarthy, Mr. Timothy J	1	male	54	0	0	17463	51.8625	E46	S	0
8	Palsson, Master. Gosta Leonard	3	male	2	3	1	349909	21.075		S	0
9	Johnson, Mrs. Oscar	3	female	27	0	2	347742	11.1333		S	1
10	Nasser, Mrs. Nicholas	2	female	14	1	0	237736	30.0708	C		1

Passenger Id	Name	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
1	Braund, Mr. Owen Harris	3	male	22	1	0	21171	7.25	S		0
2	Cumings, Mrs. John Bradley	1	female	38	1	0	17599	71.2833	C85	C	1
3	Heikkinen, Miss. Laina	3	female	26	0	0	3101282	7.925		S	1
4	Futrelle, Mrs. Jacques Heath	1	female	35	1	0	113803	53.1	C123	S	1
5	Allen, Mr. William Henry	3	male	35	0	0	373450	8.05		S	0
6	Moran, Mr. James	3	male		0	0	330877	8.4583	Q		0
7	McCarthy, Mr. Timothy J	1	male	54	0	0	17463	51.8625	E46	S	0
8	Palsson, Master. Gosta Leonard	3	male	2	3	1	349909	21.075		S	0
9	Johnson, Mrs. Oscar	3	female	27	0	2	347742	11.1333		S	1
10	Nasser, Mrs. Nicholas	2	female	14	1	0	237736	30.0708	C		1

$$P\left(\frac{\text{Survived}}{\text{Male} \& \text{Class} \& \text{Age} \& \text{Cabin} \& \text{Fare}}\right)$$

Make a naïve assumption that features such as male, class, age , cabin, fare etc. are independent of each other

7. Math problem and solution

Problem

- ✓ If the weather is sunny, then the Player should play or not?

Solution

- ✓ To solve this, first consider the below dataset

Outlook		Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	$5/14 = 0.35$
Rainy	2	2	$4/14 = 0.29$
Sunny	2	3	$5/14 = 0.35$
All	$4/14 = 0.29$	$10/14 = 0.71$	

Applying Bayes' theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = 0.60$$

$$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{No}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = 0.41$$

So as we can see from the above calculation that $P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$

Hence on a Sunny day, Player can play the game.

8. Use cases



Program Name Loading the dataset
demo1.py

```
import pandas as pd

df = pd.read_csv("titanic.csv")

print(df.head())
```

Output

	Pclass	Sex	Age	Fare	Survived
0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

Program Name Preparing input
demo2.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])
inputs = df.drop('Survived', axis='columns')

print(inputs.head())
```

Output

	Pclass	Sex	Age	Fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500

Program Name Preparing target
demo3.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])
inputs = df.drop('Survived', axis='columns')

target = df.Survived

print(target)
```

Output

```
0      0
1      1
2      1
3      1
4      0
 ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Program Name Creating dummy variables
demo4.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])
inputs = df.drop('Survived', axis = 'columns')
target = df.Survived

dummies = pd.get_dummies(inputs.Sex, dtype = int)

print(dummies.head())
```

Output

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

Program Name Concatenating the dataframe
demo5.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)

inputs = pd.concat([inputs, dummies], axis = 'columns')

print(inputs.head())
```

Output

	Pclass	Sex	Age	Fare	female	male
0	3	male	22.0	7.2500	0	1
1	1	female	38.0	71.2833	1	0
2	3	female	26.0	7.9250	1	0
3	1	female	35.0	53.1000	1	0
4	3	male	35.0	8.0500	0	1

Program Name Dropping unnecessary column
demo6.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')

inputs.drop(['Sex', 'male'], axis = 'columns', inplace = True)

print(inputs.head())
```

Output

	Pclass	Age	Fare	female
0	3	22.0	7.2500	0
1	1	38.0	71.2833	1
2	3	26.0	7.9250	1
3	1	35.0	53.1000	1
4	3	35.0	8.0500	0

Program Name Checking empty values
demo7.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis = 'columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'], axis='columns', inplace = True)

print(inputs.columns[inputs.isna().any()])
```

Output

```
Index(['Age'], dtype='object')
```

Program Name Checking empty values
demo8.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)

inputs = pd.concat([inputs, dummies], axis = 'columns')

inputs.drop(['Sex', 'male'],axis='columns', inplace=True)

print(inputs.Age)
```

Output

```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
...
886    27.0
887    19.0
888    NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

Program Name Filling Age Nan values with mean
demo9.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

print(inputs)
```

Output

	Pclass	Age	Fare	female
0	3	22.000000	7.2500	0
1	1	38.000000	71.2833	1
2	3	26.000000	7.9250	1
3	1	35.000000	53.1000	1
4	3	35.000000	8.0500	0
..
886	2	27.000000	13.0000	0
887	1	19.000000	30.0000	1
888	3	29.699118	23.4500	1
889	1	26.000000	30.0000	0
890	3	32.000000	7.7500	0
[891 rows x 4 columns]				

Program Name Splitting the dataset
demo10.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

print("Splitting the dataset")
```

Output

Splitting the dataset

Program Name Creating model
demo11.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

print("Model got trained")
```

Output
Model got trained

Program Name Prediction demo12.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

print(X_test[0:5])
print(model.predict(X_test[0:5]))
```

Output

	Pclass	Age	Fare	female
445	1	4.00	81.8583	0
831	2	0.83	18.7500	0
294	3	24.00	7.8958	0
736	3	48.00	34.3750	1
525	3	40.50	7.7500	0
	[1 0 0 1 0]			

Program Name Prediction probability
demo13.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

print(model.predict(X_test[0:5]))
print(model.predict_proba(X_test[:10]))
```

Output

```
[0 1 1 1 0]
[[9.36162489e-01 6.38375110e-02]
 [4.15057870e-01 5.84942130e-01]
 [1.59954910e-10 1.00000000e+00]
 [2.61425161e-01 7.38574839e-01]
 [9.38899635e-01 6.11003653e-02]]
```

Program Name

Cross validation score

demo14.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

model.predict(X_test[0:5])

print(cross_val_score(GaussianNB(),X_train, y_train, cv=5))
```

Output

```
[ 0.768      0.792      0.776      0.80645161  0.75806452 ]
```

31. Data Science – Machine Learning – GridSearchCV

Contents

1. Introduction	2
2. Hyper parameter tuning	3
3. Problem and solution	4
4. Hyper parameter tuning	5
5. Kernel Functions	6
6. Ways to tune parameters	8
7. RandomizedSearchCV	18

31. Data Science – Machine Learning – GridSearchCV

1. Introduction

- ✓ While building a Machine learning model we always define two things that are,
 - Model parameters
 - Model hyperparameters of a predictive algorithm.
- ✓ Model parameters are the ones that are an internal part of the model and their value is computed automatically.
 - Support vector machine.
- ✓ Hyperparameters are the ones that can be manipulated by the programmer to improve the performance of the model like the learning rate.
- ✓ Different parts of a hyperparameter approaches,
 - GridSearchCV
 - RandomizedSearchCV

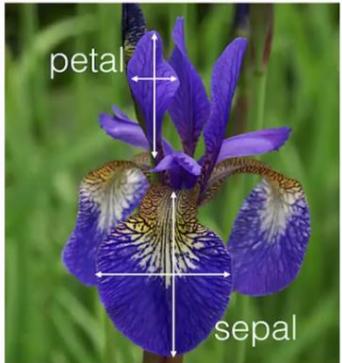
2. Hyper parameter tuning

- ✓ Hyperparameter tuning is the process of tuning the parameters while building machine learning models.
- ✓ These parameters are defined by programmer wish; machine learning algorithms never learn these parameters.
- ✓ If parameters are tuned then we will get good performance to the model
- ✓ We define the hyperparameter as shown below for the random forest classifier model.
- ✓ These parameters are tuned randomly and results are checked.

```
○ RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
criterion='mse', max_depth=None, max_features='auto',
max_leaf_nodes=None, max_samples=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

3. Problem and solution

sklearn iris flower dataset



features				target label
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5.0	3.3	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor

- ✓ Above problem we can solve by using different algorithms
 - LogisticRegression
 - SVM
 - Decision Tree
 - RandomForest
 - NaiveBayes
- ✓ If we are trying to use SVM then it having different parameters

4. Hyper parameter tuning

- ✓ The process of choosing parameters called as hyper parameter tuning

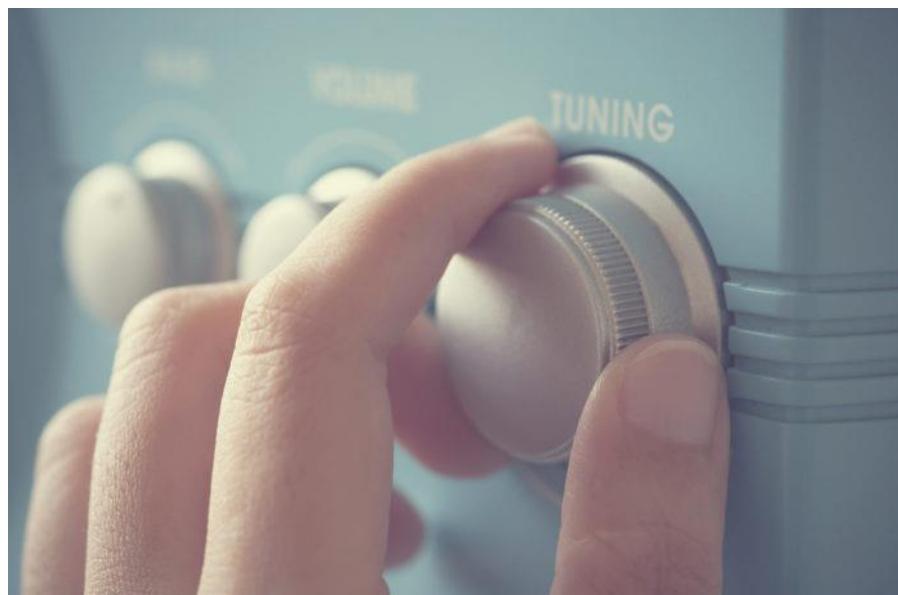
SVM

example `model = svm.SVC(kernel='rbf',C=30,gamma='auto')`

Parameter	Values
Kernel	'rbf', 'linear', 'poly'
C	Integer
Gamma	float

5. Kernel Functions

- ✓ SVM algorithms use a set of mathematical functions that are defined as the kernel.
 - The function of kernel is to take data as input and transform it into the required form.
 - For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.



Program Name Loading iris dataset
demo1.py

```
from sklearn import datasets
import pandas as pd

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

print(df)
```

Output

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  flower
0            5.1           3.5            1.4           0.2    setosa
1            4.9           3.0            1.4           0.2    setosa
2            4.7           3.2            1.3           0.2    setosa
3            4.6           3.1            1.5           0.2    setosa
4            5.0           3.6            1.4           0.2    setosa
..          ...
145           6.7           3.0            5.2           2.3  virginica
146           6.3           2.5            5.0           1.9  virginica
147           6.5           3.0            5.2           2.0  virginica
148           6.2           3.4            5.4           2.3  virginica
149           5.9           3.0            5.1           1.8  virginica
[150 rows x 5 columns]
```

6. Ways to tune parameters

- ✓ Approach 1: Use train_test_split and manually tune parameters by trial and error
- ✓ Approach 2: Use K Fold Cross validation
- ✓ Approach 3: Use GridSearchCV

Program Approach 1: Use train_test_split and manually tune parameters by trial and error
Name demo2.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size = 0.3)

model = svm.SVC(kernel = 'rbf', C = 30, gamma = 'auto')
model.fit(X_train, y_train)

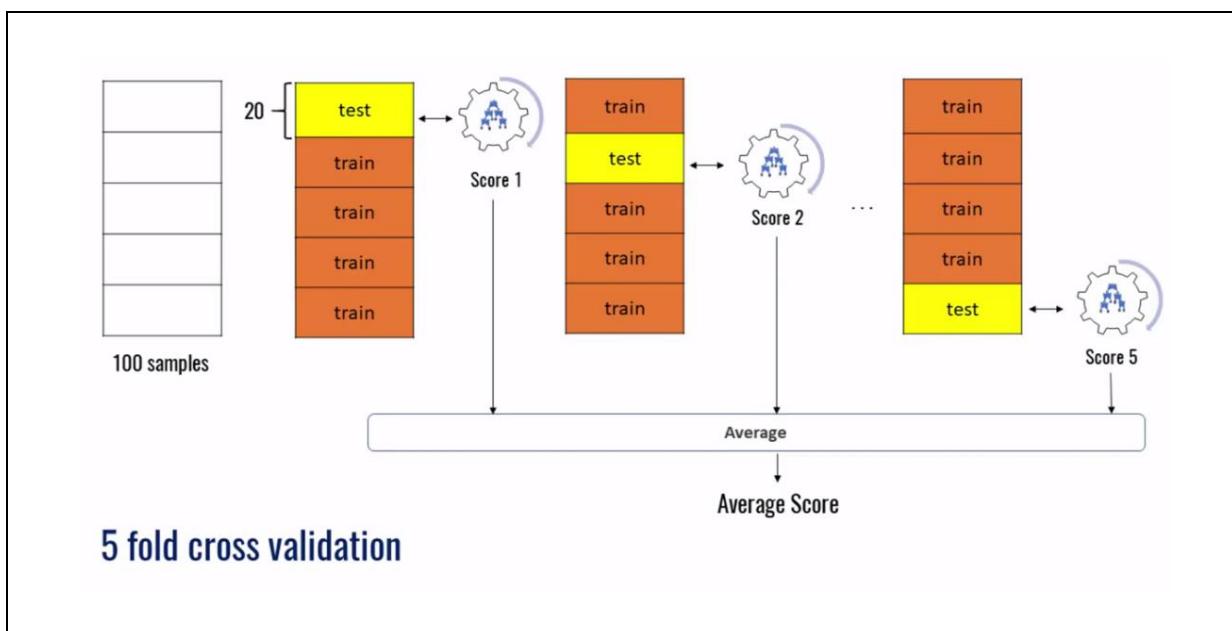
print(model.score(X_test, y_test))
```

Output

```
C:\Users\admin\Desktop>py demo.py
0.9111111111111111

C:\Users\admin\Desktop>py demo.py
0.9777777777777777

C:\Users\admin\Desktop>py demo.py
0.9555555555555556
```



Program Name

Approach 2: Use K Fold Cross validation

demo3.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc1 = cross_val_score(svm.SVC(kernel='linear', C=10,
gamma='auto'), iris.data, iris.target, cv=5)

print(sc1)
```

Output

```
[1.          1.          0.9         0.96666667 1.        ]
```

Program Name

Approach 2: Use K Fold Cross validation

demo4.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc2 = cross_val_score(svm.SVC(kernel='rbf', C=10, gamma='auto'),
iris.data, iris.target, cv=5)

print(sc2)
```

Output

```
[ 0.96666667 1.           0.96666667 0.96666667 1.           ]
```

Program Name

Approach 2: Use K Fold Cross validation
demo5.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc3 = cross_val_score(svm.SVC(kernel='rbf', C=20, gamma='auto'),
iris.data, iris.target, cv=5)

print(sc3)
```

Output

```
[ 0.96666667 1.           0.9           0.96666667 1.           ]
```

Program Name Approach: Use GridSearchCV
demo6.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1, 10, 20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv = 5,
return_train_score = False)

clf.fit(iris.data, iris.target)

print(clf.cv_results_)
```

Output

```
{'mean_fit_time': array([0.00124702, 0.          , 0.00060325, 0.00039907, 0.00067601,
   0.00019941]), 'std_fit_time': array([0.00203303, 0.          , 0.00120649, 0.00048875, 0.00093714,
   0.00039883]), 'mean_score_time': array([0.00019922, 0.00120854, 0.00019917, 0.00100141, 0.
   , 0.0012084 ]), 'std_score_time': array([0.00039845, 0.00195683, 0.00039835, 0.00200281, 0.
   , 0.00195764]), 'param_C': masked_array(data=[1, 1, 10, 10, 20, 20],
   mask=[False, False, False, False, False, False],
   fill_value='?'),
   dtype=object), 'param_kernel': masked_array(data=['rbf', 'linear', 'rbf', 'linear', 'rbf',
'linear'],
   mask=[False, False, False, False, False, False],
   fill_value='?'),
   dtype=object), 'params': [{ 'C': 1, 'kernel': 'rbf'}, { 'C': 1, 'kernel': 'linear'}, { 'C': 10
, 'kernel': 'rbf'}, { 'C': 10, 'kernel': 'linear'}, { 'C': 20, 'kernel': 'rbf'}, { 'C': 20, 'kernel': 'lin
ear'}], 'split0_test_score': array([0.96666667, 0.96666667, 0.96666667, 1.          , 0.96666667,
1.          ], 'split1_test_score': array([1., 1., 1., 1., 1.]), 'split2_test_score': array([
0.96666667, 0.96666667, 0.96666667, 0.9       , 0.9       ],
0.9       ), 'split3_test_score': array([0.96666667, 0.96666667, 0.96666667, 0.96666667, 0.9666
6667,
```

Program Name Approach: Use GridSearchCV, creating DataFrame with results
demo7.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1, 10, 20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma = 'auto'), d, cv = 5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(df)
```

Output

	mean_fit_time	std_fit_time	mean_score_time	...	mean_test_score	std_test_score	rank_test_score
0	0.000758	0.001515	0.001209	...	0.980000	0.016330	1
1	0.001207	0.001955	0.000000	...	0.980000	0.016330	1
2	0.000199	0.000399	0.000000	...	0.980000	0.016330	1
3	0.000419	0.000536	0.000000	...	0.973333	0.038873	4
4	0.001052	0.002105	0.000000	...	0.966667	0.036515	5
5	0.000197	0.000393	0.001008	...	0.966667	0.042164	6

[6 rows x 15 columns]

Program Name Approach: Use GridSearchCV, creating DataFrame with results
demo8.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1,10,20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

result = df[['param_C', 'param_kernel', 'mean_test_score']]

print(result)
```

Output

	param_C	param_kernel	mean_test_score
0	1	rbf	0.980000
1	1	linear	0.980000
2	10	rbf	0.980000
3	10	linear	0.973333
4	20	rbf	0.966667
5	20	linear	0.966667

Program Name Approach: Use GridSearchCV.
demo9.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1,10,20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(clf.best_params_)
```

Output

```
{'C': 1, 'kernel': 'rbf'}
```

Program Name Approach: Use GridSearchCV.
demo10.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1,10,20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(clf.best_score_)
```

Output

```
0.9800000000000001
```

7. RandomizedSearchCV

- ✓ Use RandomizedSearchCV to reduce number of iterations and with random combination of parameters.
- ✓ This is useful when you have too many parameters to try and your training time is longer. It helps reduce the cost of computation

Program Name

Approach: Use RandomizedSearchCV

demo11.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import RandomizedSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1, 10, 20],
    'kernel': ['rbf', 'linear']
}

rs = RandomizedSearchCV(svm.SVC(gamma = 'auto'), d, cv = 5,
                        return_train_score = False,
                        n_iter=2
)

rs.fit(iris.data, iris.target)
cols = ['param_C', 'param_kernel', 'mean_test_score']
df = pd.DataFrame(rs.cv_results_)[cols]

print(df)
```

Output

	param_C	param_kernel	mean_test_score
0	10	rbf	0.980000
1	20	rbf	0.966667

32. Data Science – Machine Learning – XGBoost

Contents

1. Introduction	2
2. Why Use XGBoost?	2
3. Install XGBoost	2
4. Dataset explanation	3
5. Input and output from the Dataset	4
5.1. Input Variables (X):.....	4
5.2. Output Variables (y):.....	4
6. Label Encoding	15
7. Feature Importance with XGBoost and Feature Selection	24

32. Data Science – Machine Learning – XGBoost

1. Introduction

- ✓ XGBoost stands for eXtreme Gradient Boosting.
- ✓ XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.
- ✓ It combines the predictions from two or more models.
- ✓ So that it gets the better performance.

2. Why Use XGBoost?

- ✓ The two reasons to use XGBoost are also the two goals of the project:
 - Execution Speed.
 - Model Performance.

3. Install XGBoost

- ✓ pip install xgboost

4. Dataset explanation

- ✓ We are going to work with **pima-indians-diabetes.csv**
- ✓ This Dataset is related to health care domain.
- ✓ Pima Indians are a Native American group that lives in Mexico and Arizona, USA.
- ✓ It describes patient medical record data for Pima Indians and whether they had a diabetes within five years.
- ✓ The Pima Indian Diabetes dataset consisting of Pima Indian females 21 years and older is a popular benchmark dataset.
- ✓ It is a binary classification problem (onset of diabetes as 1 or not as 0).
- ✓ All of the input variables that describe each patient are numerical.

Feature	Description	Data type	Range
Preg	Number of times pregnant	Numeric	[0, 17]
Gluc	Plasma glucose concentration at 2 Hours in an oral glucose tolerance test (GTIT)	Numeric	[0, 199]
BP	Diastolic Blood Pressure (mm Hg)	Numeric	[0, 122]
Skin	Triceps skin fold thickness (mm)	Numeric	[0, 99]
Insulin	2-Hour Serum insulin (μ h/ml)	Numeric	[0, 846]
BMI	Body mass index [weight in kg/(Height in m)]	Numeric	[0, 67.1]
DPF	Diabetes pedigree function	Numeric	[0.078, 2.42]
Age	Age (years)	Numeric	[21, 81]
Outcome	Binary value indicating non-diabetic /diabetic	Factor	[0,1]

5. Input and output from the Dataset

5.1. Input Variables (X):

- ✓ 1. Number of times pregnant
- ✓ 2. Plasma glucose concentration at 2 hours in an oral glucose tolerance test
- ✓ 3. Diastolic blood pressure (mm Hg)
- ✓ 4. Triceps skin fold thickness (mm)
- ✓ 5. 2-hour serum insulin (μ IU/ml)
- ✓ 6. Body mass index (weight in kg/(height in m))
- ✓ 7. Diabetes pedigree function
- ✓ 8. Age (years)

5.2. Output Variables (y):

- ✓ 1. Class variable (0 or 1)

Program Name Loading csv file
demo1.py
Input file pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

print(dataset)
```

Output

```
[[ 6.    148.    72.    ...   0.627   50.    1.    ]
 [ 1.    85.    66.    ...   0.351   31.    0.    ]
 [ 8.    183.    64.    ...   0.672   32.    1.    ]
 ...
 [ 5.    121.    72.    ...   0.245   30.    0.    ]
 [ 1.    126.    60.    ...   0.349   47.    1.    ]
 [ 1.    93.    70.    ...   0.315   23.    0.    ]]
```

Program Preparing input (X) and output (y) variables

Name demo2.py

Input file pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Input and output")
```

Output

Input and output

Program Name Splitting data into train and test datasets
Input file demo3.py
pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

print("Train and Test dataset")
```

Output

Train and Test dataset

Program Name Model creation
demo4.py
Input file pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
print("Model created")
```

Output

Model created

Program Name Train the model
Name demo5.py
Input file pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)
print("Model trained")
```

Output

Model created

Program Model prediction
Name demo6.py
Input file pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(X_test)
print(predictions)
```

Output

```
[[ 1.    90.    62.    ...   27.2    0.58   24.   ]
 [ 7.   181.    84.    ...   35.9    0.586   51.   ]
 [ 13.   152.    90.    ...   26.8    0.731   43.   ]
 ...
 [ 4.   118.    70.    ...   44.5    0.904   26.   ]
 [ 7.   152.    88.    ...   50.    0.337   36.   ]
 [ 7.   168.    88.    ...   38.2    0.787   40.   ]]
[0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0
0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0
1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1
1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1
1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1]
```

Program Model prediction
Name demo7.py
Input file pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(X_test)
print(predictions)
```

Output

```
[[ 1.    90.    62.    ...   27.2    0.58   24.   ]
 [ 7.   181.    84.    ...   35.9    0.586   51.   ]
 [ 13.   152.    90.    ...   26.8    0.731   43.   ]
 ...
 [ 4.   118.    70.    ...   44.5    0.904   26.   ]
 [ 7.   152.    88.    ...   50.    0.337   36.   ]
 [ 7.   168.    88.    ...   38.2    0.787   40.   ]]
[0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0
0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0
1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1
1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0
1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0]
```

Program Name Check model accuracy
Input file demo8.py
pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print(accuracy)
```

Output

0.740157

6. Label Encoding

- ✓ The iris flowers classification problem is an example of a problem that has a string class value.
- ✓ XGBoost cannot model this problem because output variables are strings, we need to convert output variables be numeric.
- ✓ We can easily convert the string values to integer values using the LabelEncoder.
- ✓ The three class values
 - (Iris-setosa, Iris-versicolor, Iris-virginica) are mapped to the integer values (0, 1, 2).

Program Name	Loading csv file demo1.py
Input file	iris.csv

```
from pandas import read_csv

data = read_csv('iris.csv', header = None)
dataset = data.values

print(dataset[0:5])
```

Output

```
[[5.1 3.5 1.4 0.2 'Iris-setosa']
 [4.9 3.0 1.4 0.2 'Iris-setosa']
 [4.7 3.2 1.3 0.2 'Iris-setosa']
 [4.6 3.1 1.5 0.2 'Iris-setosa']
 [5.0 3.6 1.4 0.2 'Iris-setosa']]
```

Program Data preparation
Name demo2.py
Input file iris.csv

```
from pandas import read_csv

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

print(X[0:5])
print()
print(Y[0:5])
```

Output

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.0 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.0 3.6 1.4 0.2]]

['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa']
```

Program	Encoding flower names
Name	demo3.py
Input file	iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

print(label_encoded_y)
```

Output

Program Name Splitting the data
Input file demo4.py
iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

print("Data splitting done")
```

Output

Data splitting done

Program Model creation
Name demo5.py
Input file iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()

print("Model created")
```

Output

Model created

Program Name Model training

Name demo6.py

Input file iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, y_train)

print("Model trained")
```

Output

Model trained

Program Prediction
Name demo7.py
Input file iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(X_test[0:5])
print()
print(predictions[0:5])
```

Output

```
[[5.9 3.0 5.1 1.8]
 [5.4 3.0 4.5 1.5]
 [5.0 3.5 1.3 0.3]
 [5.6 3.0 4.5 1.5]
 [4.9 2.5 4.5 1.7]]

[2 1 0 1 1]
```

Program Name Evaluate predictions
Input file demo8.py
iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Output

Accuracy: 92.00%

7. Feature Importance with XGBoost and Feature Selection

- ✓ XGBoost is helping to get important features from the existing features
- ✓ We can see like every features how much score it having.

Program Name Feature Importance with XGBoost

Input file demo1.py

pima-indians-diabetes.csv

```
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

model = XGBClassifier()

model.fit(X, y)

print(model.feature_importances_)
```

Output

```
[0.10621195 0.24240209 0.08803371 0.07818192 0.10381888
 0.14867325 0.10059208 0.13208608]
```

Program Name Plotting the Feature Importance with XGBoost
Input file demo2.py
pima-indians-diabetes.csv

```
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

model = XGBClassifier()

model.fit(X, y)

print(model.feature_importances_)

r = range(len(model.feature_importances_))
f_imp = model.feature_importances_

pyplot.bar(r, f_imp)
pyplot.show()
```

Output

```
[0.10621195 0.24240209 0.08803371 0.07818192 0.10381888  
 0.14867325 0.10059208 0.13208608]
```

