# 11th-Linear Algebra

**Road Map**

**📌 DAY 1: Understanding Data Science & Its Fundamentals**

🔹 **What is Data Science?** → Learn about Data Science, its definition, components (Statistics, ML, AI, Big Data).

🔹 **Why is Data Science used?** → Understand its applications in different industries.

🔹 **How does Data Science work?** → Learn about the lifecycle (Data Collection → Cleaning → Analysis → Modeling → Deployment).

🔹 **Data Ethics** → Study key principles like fairness, privacy, and bias in data science.

🔹 **Data Science Workflow** → Understand CRISP-DM, KDD process, or any standard workflow.

🔹 **Different Types of Data** → Learn about structured, unstructured, semi-structured, categorical, numerical data.

🔹 **Challenges in Data Science** → Identify real-world challenges like data quality, scalability, and model explainability.

📌 **Output:** Notes summarizing all topics.

**📌 DAY 2: Linear Algebra – Matrices and Operations**

🔹 **Concepts of Matrices** → Learn about rows, columns, elements, dimensions.

🔹 **Matrix Representation & Transformations** → Study different ways to represent matrices in Python.

🔹 **Basic Matrix Operations** → Addition, subtraction, multiplication, transpose, determinant.

🔹 **Types of Matrices** → Square, diagonal, identity, symmetric, sparse, dense matrices.

📌 **Task:** Implement **Matrix Addition, Subtraction, and Transpose** using Python **without NumPy or libraries**.

**📌 DAY 3: Dot Product, Eigenvalues & Eigenvectors**

🔹 **Dot Product** → Learn what it is and how it works in vector multiplication.

🔹 **Eigenvalues & Eigenvectors** → Understand their significance in dimensionality reduction (PCA).

🔹 **Matrix Operations in Python** → Implement eigenvalue and eigenvector calculations manually.

📌 **Task:** Write Python scripts for **Dot Product** and **Eigenvalues/Eigenvectors calculation** without libraries.

**📌 DAY 4: Advanced Matrix Operations & Case Study**

🔹 **Advanced Matrix Operations** → Inverse, Rank, Determinant, Trace, Orthogonality.

🔹 **Python Implementation** → Write scripts for all these operations **without any libraries**.

🔹 **Data Science Case Study** → Pick a real-world case study and analyze how Data Science was used to solve a problem.

📌 **Task:** Write a Python script implementing **Matrix Inverse & Rank** from scratch.

**📌 DAY 5: Identify & Solve a Problem using Data Science**

🔹 **Find a Real-World Problem** → Research an issue (e.g., fraud detection, stock prediction, disease detection).

🔹 **Understand its Solution** → Learn what data is needed, what algorithms are used, and how the model is built.

📌 **Task:** Document the problem statement, datasets used, and approach.

**📌 DAY 6: Documentation & Final Review**

🔹 **Compile all your notes, Python scripts, and case study** into a structured document for future reference.

🔹 **Review your work** and make sure everything is well-documented.

📌 **Final Output:** A **well-organized document** covering all **Data Science concepts, Linear Algebra, Case Study, and Problem Solving Approach.**

**First Note**

**Data Science**

**📌 What is Data Science?**

**🔹 Definition of Data Science**

Data Science is an interdisciplinary field that combines **statistics, mathematics, programming, and domain knowledge** to extract meaningful insights from structured and unstructured data. It involves **collecting, cleaning, analyzing, visualizing, and interpreting data** to make data-driven decisions.

**🔹 Why is Data Science Important?**

* Helps businesses make **informed decisions** using data.
* Powers **AI & Machine Learning models** for automation.
* Identifies **patterns & trends** in large datasets.
* Solves real-world problems in **healthcare, finance, marketing, and more**.

**📌 Key Components of Data Science**

**1️⃣ Statistics & Mathematics**

* **Statistics** helps analyze data distributions, relationships, and trends.
* **Mathematics (Linear Algebra, Probability, Calculus)** is essential for building ML models.

📌 Example: Probability is used in fraud detection to predict if a transaction is fraudulent.

**2️⃣ Machine Learning (ML)**

* **Machine Learning** enables computers to learn from data without explicit programming.
* Uses algorithms like **Linear Regression, Decision Trees, Neural Networks, etc.**

📌 Example: Netflix recommends movies using ML models trained on user preferences.

**3️⃣ Artificial Intelligence (AI)**

* AI refers to machines mimicking human intelligence.
* **Machine Learning is a subset of AI.**
* AI models use **Deep Learning & NLP** for advanced decision-making.

📌 Example: Chatbots like Siri, Alexa, and ChatGPT use AI for human-like conversations.

**4️⃣ Big Data Technologies**

* **Big Data** refers to massive datasets that traditional systems can’t handle.
* Technologies like **Hadoop, Spark, and NoSQL databases** manage big data.

📌 Example: Google processes **petabytes** of data daily to rank search results efficiently.

**5️⃣ Data Engineering**

* Focuses on **data collection, storage, and processing.**
* Uses tools like **SQL, ETL Pipelines, Apache Spark, and Airflow**.

📌 Example: Facebook’s News Feed processes billions of user interactions in real-time.

**6️⃣ Data Visualization**

* Communicates insights using **graphs, charts, and dashboards**.
* Tools: **Matplotlib, Seaborn, Tableau, Power BI**.

📌 Example: A stock market dashboard showing trends over time.

**7️⃣ Domain Knowledge**

* Understanding the industry (finance, healthcare, e-commerce) helps in **problem-solving**.

📌 Example: A healthcare data scientist needs medical knowledge to analyze disease trends.

**📌 Real-World Applications of Data Science**

✅ **Healthcare:** Predicting diseases using AI.

✅ **Finance:** Fraud detection, credit scoring.

✅ **E-commerce:** Personalized recommendations (Amazon, Flipkart).

✅ **Marketing:** Customer segmentation, sentiment analysis.

✅ **Self-Driving Cars:** AI-based decision-making using data.

**📌 Summary**

✔ **Data Science = Statistics + ML + AI + Big Data + Engineering + Visualization + Domain Knowledge**

✔ It helps businesses **make better decisions** using data.

✔ It is used in **healthcare, finance, e-commerce, marketing, and more**.

**📌 Applications of Data Science in Different Industries**

**1️⃣ Healthcare 🏥**

✔ **Disease Prediction & Diagnosis:** AI models analyze patient data to predict diseases like cancer, diabetes, and heart conditions.

✔ **Medical Image Analysis:** Detects abnormalities in **X-rays, MRIs, and CT scans** using deep learning.

✔ **Drug Discovery:** Accelerates new drug development by analyzing chemical structures and biological reactions.

✔ **Personalized Medicine:** Uses patient history and genetics to recommend **customized treatments.**

✔ **Wearable Devices & Health Monitoring:** Smartwatches (like Fitbit, Apple Watch) track heart rate, sleep patterns, and oxygen levels.

📌 **Example:** IBM Watson analyzes medical data to assist doctors in diagnosing diseases faster.

**2️⃣ Finance & Banking 💰**

✔ **Fraud Detection:** Identifies suspicious transactions using pattern recognition.

✔ **Credit Risk Assessment:** Predicts loan default risk based on customer history.

✔ **Algorithmic Trading:** Uses AI to execute trades at lightning speed.

✔ **Customer Segmentation:** Analyzes spending behavior to offer **personalized financial services**.

✔ **Chatbots & Virtual Assistants:** AI-driven assistants help customers with banking queries.

📌 **Example:** PayPal and banks use AI to detect fraudulent transactions and block them in real-time.

**3️⃣ E-commerce & Retail 🛒**

✔ **Product Recommendations:** Suggests products based on user behavior (Amazon, Flipkart).

✔ **Dynamic Pricing:** Adjusts product prices in real-time based on demand and competitor prices.

✔ **Inventory Management:** Predicts stock demand and prevents shortages or overstocking.

✔ **Sentiment Analysis:** Analyzes customer reviews to improve products/services.

✔ **Customer Churn Prediction:** Identifies customers likely to leave and offers discounts to retain them.

📌 **Example:** Amazon’s "Customers who bought this also bought..." is powered by Data Science.

**4️⃣ Social Media 📱**

✔ **Content Recommendation:** Facebook, Instagram, TikTok, and YouTube use AI to show relevant posts and videos.

✔ **Fake News Detection:** Identifies and flags misleading content.

✔ **Sentiment Analysis:** Analyzes public reactions to events, brands, and influencers.

✔ **Ad Targeting:** Shows personalized ads based on browsing history.

📌 **Example:** YouTube recommends videos based on your watch history.

**5️⃣ Transportation & Logistics 🚗**

✔ **Route Optimization:** Google Maps and Uber suggest the best routes to avoid traffic.

✔ **Self-Driving Cars:** Tesla uses AI to process road and traffic data for autonomous driving.

✔ **Delivery Prediction:** Predicts when your online order will arrive.

✔ **Fleet Management:** Monitors and optimizes transportation networks.

📌 **Example:** Uber’s surge pricing is based on real-time demand using Data Science.

**6️⃣ Manufacturing 🏭**

✔ **Predictive Maintenance:** Detects machine failures before they happen.

✔ **Supply Chain Optimization:** Reduces delays and improves efficiency.

✔ **Quality Control:** Uses AI to inspect products for defects.

✔ **Demand Forecasting:** Predicts production needs based on historical trends.

📌 **Example:** Tesla uses AI-powered robots for automated car manufacturing.

**7️⃣ Sports & Entertainment ⚽🎬**

✔ **Player Performance Analysis:** Tracks athlete movements and predicts injuries.

✔ **Match Outcome Prediction:** AI analyzes team performance and suggests game strategies.

✔ **Audience Engagement:** Streaming platforms (Netflix, Spotify) recommend movies, songs, and shows.

✔ **Social Media Analysis:** Monitors fan reactions and engagement.

📌 **Example:** Netflix uses Data Science to recommend shows you might like.

**8️⃣ Cybersecurity 🔒**

✔ **Intrusion Detection:** Identifies and prevents hacking attempts.

✔ **Spam & Malware Detection:** Filters phishing emails and harmful software.

✔ **User Authentication:** AI-based face and fingerprint recognition enhance security.

📌 **Example:** Gmail uses AI to filter spam and phishing emails.

**📌 Summary: Why is Data Science Used?**

✅ **Predicts future trends** (Stock market, customer demand, weather forecasting).

✅ **Automates decision-making** (AI chatbots, self-driving cars, recommendation systems).

✅ **Improves efficiency** (Healthcare, banking, supply chain, manufacturing).

✅ **Reduces risks & fraud** (Cybersecurity, fraud detection).

✅ **Enhances customer experience** (Personalized ads, product recommendations).

**📌 How Does Data Science Work?**

Data Science follows a structured **lifecycle** to extract insights from data and make data-driven decisions. This lifecycle consists of **five main stages:**

🔹 **1️⃣ Data Collection**

🔹 **2️⃣ Data Cleaning & Preprocessing**

🔹 **3️⃣ Data Analysis & Exploration**

🔹 **4️⃣ Machine Learning & Modeling**

🔹 **5️⃣ Model Deployment & Monitoring**

Let’s break each step down in detail!

**🔹 1️⃣ Data Collection (Gathering Raw Data)**

The first step in Data Science is to **collect data** from various sources. Data can come from:

✔ **Databases** (SQL, NoSQL, Data Warehouses)

✔ **APIs** (Google Maps, Twitter API, OpenWeather)

✔ **Web Scraping** (BeautifulSoup, Scrapy)

✔ **IoT Sensors** (Smart devices, industrial machines)

✔ **Surveys & Forms** (Google Forms, Excel)

✔ **Logs & Transactions** (Website traffic, bank transactions)

📌 **Example:**

A retail company collects data from **customer purchases, website clicks, and social media interactions.**

**🔹 2️⃣ Data Cleaning & Preprocessing (Fixing Messy Data)**

Raw data is often **incomplete, inconsistent, or contains errors**.

Data Cleaning ensures that the data is **ready for analysis** by:

✔ **Handling Missing Values** (Filling with mean, median, mode, or removing them)

✔ **Removing Duplicates** (Avoid redundant records)

✔ **Fixing Incorrect Data Types** (e.g., converting dates, numbers, and categories)

✔ **Standardization & Normalization** (Scaling values to a similar range)

✔ **Dealing with Outliers** (Removing or transforming extreme values)

📌 **Example:**

If a dataset contains customer ages but some entries have **"???"**, we replace missing values with the **average age**.

**🔹 3️⃣ Data Analysis & Exploration (Understanding the Data)**

Now that we have clean data, we **explore patterns, trends, and insights** using:

✔ **Descriptive Statistics** (Mean, Median, Mode, Variance, Standard Deviation)

✔ **Data Visualization** (Matplotlib, Seaborn, Tableau)

✔ **Correlation Analysis** (Finding relationships between features)

✔ **Feature Engineering** (Creating new meaningful features)

📌 **Example:**

An **e-commerce business** may analyze which products are sold most often **during weekends** compared to weekdays.

**🔹 4️⃣ Machine Learning & Modeling (Building Predictions)**

After understanding the data, we create **models** that can **predict, classify, or group data**.

✔ **Supervised Learning** (Regression & Classification)

✔ **Unsupervised Learning** (Clustering & Anomaly Detection)

✔ **Deep Learning (Neural Networks)** (Image recognition, NLP)

📌 **Example:**

An **ML model** predicts whether a **customer will buy a product** based on their browsing history.

**🔹 5️⃣ Model Deployment & Monitoring (Putting the Model into Action)**

Once a model is trained, it is **deployed into production** where real-world users interact with it.

✔ **Deploy models on cloud platforms** (AWS, Azure, GCP)

✔ **Monitor performance & accuracy** (Fix model drift)

✔ **Update with new data** (Retrain the model when needed)

📌 **Example:**

Netflix’s recommendation system is an **ML model** that continuously learns from user preferences to suggest better content.

**📌 Data Science Lifecycle Diagram**

pgsql

CopyEdit

+-------------------------+

| 1️⃣ Data Collection | → (APIs, Databases, IoT)

+-------------------------+

↓

+-------------------------+

| 2️⃣ Data Cleaning | → (Handle Missing Data, Outliers)

+-------------------------+

↓

+-------------------------+

| 3️⃣ Data Analysis | → (EDA, Visualization)

+-------------------------+

↓

+-------------------------+

| 4️⃣ ML & Modeling | → (Train, Test, Validate Models)

+-------------------------+

↓

+-------------------------+

| 5️⃣ Deployment | → (Put into Production)

+-------------------------+

**📌 Summary**

✅ **Data Science Workflow** follows a **step-by-step process** to extract value from data.

✅ **Real-world applications** include finance, healthcare, e-commerce, social media, and more.

✅ **ML models help in predictions & automation** for businesses.

**📌 Data Ethics in Data Science**

Data Ethics ensures that **data is collected, processed, and used responsibly** without harming individuals or groups. It focuses on **fairness, transparency, privacy, and accountability** to prevent bias, discrimination, and misuse.

**🔹 Why is Data Ethics Important?**

Without ethical guidelines, **AI and Data Science models can be harmful** by:

❌ **Violating privacy** (e.g., unauthorized tracking of users)

❌ **Introducing bias** (e.g., AI hiring models favoring certain demographics)

❌ **Being unfair** (e.g., loan approval models rejecting people unfairly)

❌ **Manipulating decisions** (e.g., fake news spreading on social media)

📌 **Example:**

🔴 **Amazon’s AI-based hiring system** was scrapped because it **favored male candidates** over women due to biased training data.

**🔹 Key Principles of Data Ethics**

**1️⃣ Fairness & Bias-Free Models**

* AI and ML models should not **discriminate** against individuals or groups.
* **Bias in Data:** If training data is biased, the model will also be biased.
* **Fair Decision-Making:** Models should provide equal opportunities to everyone.

📌 **Example:**

🟢 **Fair:** Loan approval models considering **income, credit score, and expenses**.

🔴 **Unfair:** Loan approval models rejecting applications based on **race, gender, or ZIP codes**.

**2️⃣ Privacy & Data Protection**

* Organizations must protect **user data from misuse or unauthorized access**.
* Follow laws like **GDPR (Europe) & CCPA (California)** to ensure privacy.
* **Anonymization & Encryption** should be used to protect personal data.

📌 **Example:**

🔴 **Facebook-Cambridge Analytica Scandal** → Facebook misused user data for political ads without consent.

🟢 **Apple’s Privacy Focus** → Apple encrypts user data and gives control over app tracking.

**3️⃣ Transparency & Explainability**

* AI systems should be **explainable**, so users understand **how decisions are made**.
* Black-box models (like deep learning) can be **hard to interpret**, leading to mistrust.
* **Users should know when they are interacting with AI** (e.g., AI chatbots vs. humans).

📌 **Example:**

🔴 **Bad Transparency:** A credit card company reduces a customer’s credit limit **without explanation**.

🟢 **Good Transparency:** AI loan models explain why a loan was approved or denied.

**4️⃣ Accountability & Responsibility**

* Companies & data scientists should **take responsibility** for AI-driven decisions.
* If an AI system **makes a mistake**, someone should be accountable for fixing it.
* Ethical AI practices should be monitored & updated regularly.

📌 **Example:**

🔴 **Self-Driving Car Accidents** → Who is responsible when an AI-driven car crashes? The manufacturer, the software developer, or the driver?

🟢 **Human-AI Collaboration** → AI should assist humans, not replace critical decision-making.

**5️⃣ Informed Consent & Data Usage**

* Users must **agree** before their data is collected or used.
* Companies should provide **clear terms & conditions** instead of hidden policies.

📌 **Example:**

🟢 **Good Practice:** Websites asking for **cookie permissions** before tracking users.

🔴 **Bad Practice:** Apps collecting **location & contact data** without user consent.

**🔹 Ethical Challenges in Data Science**

| **Challenge** | **Example** |
| --- | --- |
| **Bias in AI** | AI hiring system favors one gender. |
| **Data Privacy** | Social media apps selling user data. |
| **Fake News** | AI-generated deepfake videos spreading misinformation. |
| **Job Displacement** | Automation replacing human workers unfairly. |

**📌 Summary**

✅ **Data Ethics ensures fairness, transparency, and privacy in AI & Data Science.**

✅ **Bias & unfair decision-making can be reduced with ethical data collection.**

✅ **Privacy laws like GDPR & CCPA protect user data from misuse.**

✅ **Companies must be accountable for AI-driven decisions to prevent harm.**

**📌 Data Science Workflow**

The **Data Science Workflow** provides a structured approach to solving real-world problems using data. Several standardized workflows exist, but the **two most common frameworks** are:

1️⃣ **CRISP-DM (Cross-Industry Standard Process for Data Mining)**

2️⃣ **KDD (Knowledge Discovery in Databases) Process**

Both methods guide **how data is collected, processed, analyzed, and used to generate insights**. Let's explore them! 🚀

**🔹 1️⃣ CRISP-DM Framework (Most Popular)**

CRISP-DM is a widely used workflow in Data Science that consists of **six key phases**:

🔹 **1. Business Understanding** (Define problem & objectives)

🔹 **2. Data Understanding** (Explore and analyze raw data)

🔹 **3. Data Preparation** (Clean, transform, and preprocess data)

🔹 **4. Modeling** (Apply ML algorithms & build predictive models)

🔹 **5. Evaluation** (Check model accuracy & performance)

🔹 **6. Deployment** (Put model into production & monitor)

📌 **Example:** Predicting customer churn for an online streaming service like Netflix.

| **Phase** | **Task** | **Example** |
| --- | --- | --- |
| **1. Business Understanding** | Define problem | “Why are customers unsubscribing?” |
| **2. Data Understanding** | Collect & analyze data | Customer history, subscription patterns, complaints |
| **3. Data Preparation** | Clean & preprocess | Remove duplicates, handle missing values |
| **4. Modeling** | Train ML models | Random Forest, Logistic Regression |
| **5. Evaluation** | Measure performance | Accuracy, precision, recall |
| **6. Deployment** | Put into production | Integrate model with website/app |

🔹 **Diagram of CRISP-DM Workflow**

diff

CopyEdit

+-------------------------+

| 1️⃣ Business Understanding |

+-------------------------+

↓

+-------------------------+

| 2️⃣ Data Understanding |

+-------------------------+

↓

+-------------------------+

| 3️⃣ Data Preparation |

+-------------------------+

↓

+-------------------------+

| 4️⃣ Modeling |

+-------------------------+

↓

+-------------------------+

| 5️⃣ Evaluation |

+-------------------------+

↓

+-------------------------+

| 6️⃣ Deployment |

+-------------------------+

CRISP-DM is **iterative**, meaning **if a model fails**, you go back and refine the previous steps! 🔄

**🔹 2️⃣ KDD (Knowledge Discovery in Databases) Process**

The KDD process focuses on **extracting useful patterns from large datasets**. It consists of five major steps:

1️⃣ **Selection** (Choose relevant data for analysis)

2️⃣ **Preprocessing** (Clean and prepare the data)

3️⃣ **Transformation** (Feature engineering & dimensionality reduction)

4️⃣ **Data Mining** (Apply ML & statistical models)

5️⃣ **Interpretation/Evaluation** (Analyze results & generate insights)

📌 **Example:** Detecting fraudulent transactions in banking.

| **Phase** | **Task** | **Example** |
| --- | --- | --- |
| **1. Selection** | Gather data | Transaction history, customer details |
| **2. Preprocessing** | Clean & filter | Remove duplicate transactions |
| **3. Transformation** | Feature engineering | Convert timestamps to time-of-day patterns |
| **4. Data Mining** | Apply ML models | Anomaly detection with clustering |
| **5. Interpretation** | Evaluate insights | Flag suspicious transactions |

**🔹 Comparison of CRISP-DM vs KDD**

| **Feature** | **CRISP-DM** | **KDD** |
| --- | --- | --- |
| **Focus** | End-to-end process | Pattern discovery |
| **Industry Usage** | Business & AI applications | Research & large databases |
| **Iterative?** | Yes | No (linear) |

👉 **CRISP-DM is more practical for Data Science projects**, while **KDD is useful for discovering hidden patterns in large datasets.**

**📌 Summary**

✅ **CRISP-DM is the most widely used Data Science workflow** for real-world applications.

✅ **KDD is best for uncovering patterns in massive datasets** like financial transactions.

✅ **Both workflows emphasize data cleaning, transformation, modeling, and evaluation.**

✅ **CRISP-DM is iterative, meaning we can refine steps if needed.**

**📌 Different Types of Data in Data Science**

Data is the **fuel** for Data Science. To analyze and process it effectively, we classify data into different types:

1️⃣ **Based on Structure:**

* **Structured Data** 📊
* **Unstructured Data** 🖼️📄
* **Semi-structured Data** 📂

2️⃣ **Based on Nature (Statistical Analysis):**

* **Categorical Data** 🏷️
* **Numerical Data** 🔢

**🔹 1️⃣ Data Based on Structure**

**📊 Structured Data (Well-organized, tabular data)**

* Stored in a **fixed format** (rows & columns) like databases, Excel, SQL tables.
* Easy to **search, process, and analyze** using SQL, Pandas, etc.
* Uses **schemas (predefined structure)**.

📌 **Examples:**

✅ Customer database (Name, Age, Email, Purchase History)

✅ Banking transactions (Date, Amount, Account ID)

✅ Employee records (ID, Salary, Department)

📌 **Tools to Handle:** SQL, Pandas (Python), Excel

**🖼️📄 Unstructured Data (Raw, unorganized data)**

* Does **not have a predefined format** (hard to store in tables).
* Requires **NLP, Computer Vision, or Deep Learning** to analyze.

📌 **Examples:**

✅ Images (Selfies, X-rays, Satellite photos)

✅ Videos (YouTube, Surveillance footage)

✅ Text files (Emails, Social Media comments, PDFs)

✅ Audio (Podcasts, Voice recordings)

📌 **Tools to Handle:** NLP (Text), OpenCV (Images), Deep Learning

**📂 Semi-Structured Data (Hybrid format)**

* **Not fully tabular, but still has some structure**.
* Contains tags, metadata, or labels that organize the data.

📌 **Examples:**

✅ JSON, XML (Used in APIs, Web Data)

✅ Emails (Subject & sender = structured, body text = unstructured)

✅ Log files (System & application logs)

📌 **Tools to Handle:** JSON Parsers, XML Parsers

**🔹 2️⃣ Data Based on Nature (For Analysis & ML Models)**

**🏷️ Categorical Data (Qualitative, Non-Numerical)**

* **Describes labels, categories, or groups**.
* Cannot be measured in numbers but can be classified.

📌 **Types of Categorical Data:**

🔹 **Nominal Data (No Order, Just Labels)**

* Examples: Gender (Male, Female), Blood Group (A, B, O)
* **Can’t perform mathematical operations** (e.g., "Male + Female" doesn’t make sense).

🔹 **Ordinal Data (Ordered, But No Fixed Difference)**

* Examples: Movie Ratings (★☆☆☆☆ to ★★★★★), Education Level (High School, Bachelor’s, Master’s)
* **Order matters, but differences aren’t measurable** (★★★★★ isn't exactly "5 times" better than ★☆☆☆☆).

📌 **How to Handle?** One-Hot Encoding, Label Encoding

**🔢 Numerical Data (Quantitative, Measurable in Numbers)**

* **Can be counted or measured**.
* Used in mathematical operations & ML models.

📌 **Types of Numerical Data:**

🔹 **Discrete Data (Whole Numbers, Countable)**

* Example: Number of students in a class (25, 30, 45)
* **Only takes integer values** (can’t have 2.5 students).

🔹 **Continuous Data (Decimal, Measurable)**

* Example: Height (5.8 feet), Temperature (37.5°C)
* **Can take any value in a range** (e.g., 98.6°F, 98.65°F, 98.7°F).

📌 **How to Handle?** Scaling, Normalization, Standardization

**📌 Summary**

| **Data Type** | **Definition** | **Examples** |
| --- | --- | --- |
| **Structured** | Organized, Tabular | Databases, Excel |
| **Unstructured** | No fixed format | Images, Videos, Text |
| **Semi-structured** | Hybrid format | JSON, Emails, XML |
| **Categorical (Nominal)** | Labels without order | Colors, Gender |
| **Categorical (Ordinal)** | Ordered categories | Education Level, Ratings |
| **Numerical (Discrete)** | Countable, Whole Numbers | Number of students, Orders placed |
| **Numerical (Continuous)** | Measurable, Decimal | Temperature, Height |

**🚀 Final Thought**

Understanding data types is **crucial** in Data Science because:

✅ **It helps in choosing the right preprocessing techniques** (e.g., Encoding for categorical data).

✅ **It affects ML model selection** (e.g., Regression needs numerical data, Classification works with categorical).

✅ **Improves data storage & efficiency** (Structured vs. Unstructured).

**🚧 Challenges in Data Science**

Data Science is powerful, but it's not always smooth sailing. Let's explore some real-world challenges that Data Scientists face:

**🔹 1️⃣ Data Quality Issues (Garbage In, Garbage Out)**

📌 **Problem:** Raw data is often messy, incomplete, or inconsistent.

📌 **Challenges:**

* Missing values (e.g., some customers didn’t fill in their age).
* Duplicates (e.g., the same order recorded twice).
* Incorrect or inconsistent formats (e.g., dates written as "01/03/25" vs. "March 1, 2025").

💡 **Solution:**

✅ Data Cleaning techniques (Handling missing values, standardizing formats).

✅ Use Python tools like **Pandas** to clean data efficiently.

**🔹 2️⃣ Scalability & Big Data Handling**

📌 **Problem:** Data is growing at **massive** scales (TBs & PBs). Processing large datasets can be slow & expensive.

📌 **Challenges:**

* Handling **real-time streaming data** (e.g., Stock Market, IoT Sensors).
* Managing **huge datasets efficiently** (e.g., Amazon customer purchases).
* Expensive storage & processing costs.

💡 **Solution:**

✅ Use **Big Data tools** like Hadoop, Spark, and Dask.

✅ Optimize code for efficiency (avoid loops, use vectorized operations).

**🔹 3️⃣ Bias & Fairness in Data**

📌 **Problem:** Data & algorithms can unintentionally **reinforce biases**.

📌 **Challenges:**

* If a hiring model is trained on past data where fewer women were hired, it may **unfairly reject female candidates**.
* Face recognition models may work well for some races but fail for others.

💡 **Solution:**

✅ Regularly **audit models** for bias.

✅ Ensure **diverse and representative data** for training.

✅ Use **Fairness AI techniques** (IBM AI Fairness 360, SHAP for model explainability).

**🔹 4️⃣ Data Privacy & Security**

📌 **Problem:** Handling sensitive user data **without violating privacy laws** (GDPR, HIPAA).

📌 **Challenges:**

* **User consent & data collection transparency**.
* **Risk of data breaches & hacking**.
* **Anonymizing personal information**.

💡 **Solution:**

✅ Use **encryption & access controls** (only authorized users can access data).

✅ Implement **differential privacy** to prevent user identification.

✅ Follow **legal compliance** (GDPR, CCPA, HIPAA).

**🔹 5️⃣ Model Interpretability & Explainability**

📌 **Problem:** Some AI models (like Deep Learning) are **black boxes**—hard to understand.

📌 **Challenges:**

* How do we **explain AI decisions**? (e.g., "Why did the bank reject my loan?")
* Regulatory industries (Healthcare, Finance) **require transparency**.

💡 **Solution:**

✅ Use **explainable AI tools** (SHAP, LIME) to understand model predictions.

✅ Prefer simpler models (Decision Trees, Linear Regression) when possible.

**🔹 6️⃣ Choosing the Right Model**

📌 **Problem:** There are **too many algorithms** to choose from!

📌 **Challenges:**

* **Which model is best?** (Linear Regression vs. Random Forest vs. Deep Learning).
* **Avoiding overfitting** (when a model memorizes instead of generalizing).
* **Feature selection** (which data points actually matter?).

💡 **Solution:**

✅ Compare models using **cross-validation & metrics** (accuracy, precision, recall).

✅ Use **Grid Search / AutoML** to find the best parameters.

✅ Regularization techniques (L1/L2) to reduce overfitting.

**🔹 7️⃣ Deployment & Maintenance Issues**

📌 **Problem:** Training a model is easy, but making it **work in the real world** is hard!

📌 **Challenges:**

* **Deploying AI models** to production environments.
* **Model drift** (as time passes, data patterns change, and models degrade).
* **Slow inference speed** (AI should be fast in real-time applications).

💡 **Solution:**

✅ Use cloud services (**AWS, Google Cloud AI, Azure ML**) for deployment.

✅ Set up **continuous model monitoring** (retrain when accuracy drops).

✅ Optimize models using **quantization & pruning** for faster inference.

**🚀 Final Thoughts**

Data Science is **not just about coding**—it’s about tackling these real-world challenges!

✔️ **Data Cleaning & Privacy → Build trustworthy models.**

✔️ **Scalability & Bias → Ensure fairness & efficiency.**

✔️ **Interpretability & Deployment → Make AI usable & reliable.**

**Linear Algebra**

applicaions of matrix each

* how linear algebra connected to data science

**Why is it important?**

* + **Data Representation** → Data is often stored in **matrices** (tables with rows & columns).
  + **Transformations** → Scaling, rotations, PCA (dimensionality reduction) all use matrix operations.
  + **ML Algorithms** → Models like **Linear Regression, PCA, Neural Networks** rely heavily on matrix operations.
  + **Optimization** → Finding the best model parameters involves solving equations using Linear Algebra.
  + **Graphs & Networks** → Social networks, recommendation systems use adjacency matrices for connections.

**How does it help in Data Science?**

* + **Speeds up computations** (using matrix multiplication instead of loops).
  + **Helps in feature selection & reduction** (like PCA).
  + **Powers Deep Learning** (Neural Networks use vectors & matrices for forward/backpropagation).

**Linear Algebra = Vectors + Matrices + Transformations**

It deals with **scalars (single numbers), vectors (1D arrays), matrices (2D arrays), and tensors (multi-dimensional arrays)** and how they interact.

**Key Concepts in Linear Algebra for Data Science:**

* + **Vectors** → Represent features (e.g., [height, weight, age] of a person).
  + **Matrices** → Store data (e.g., dataset with rows as samples, columns as features).
  + **Matrix Operations** → Addition, multiplication, inverse (used in ML).
  + **Eigenvalues & Eigenvectors** → Used in PCA for dimensionality reduction.
  + **Linear Transformations** → Rotation, scaling, and projection of data.

**📌 Linear Algebra in Data Science**

**Linear Algebra** is a fundamental mathematical concept used in **Data Science, Machine Learning, and AI**. It helps in **handling, transforming, and analyzing data** efficiently. Most of the mathematical operations in ML algorithms, such as **neural networks, PCA, regression models, and recommendation systems**, rely on **linear algebra**.

**📌 Why is Linear Algebra Important in Data Science?**

Linear algebra is used for:

✔ **Data Representation** → Organizing data in the form of matrices.

✔ **Dimensionality Reduction** → Techniques like **Principal Component Analysis (PCA)**.

✔ **Machine Learning Models** → Algorithms like **Linear Regression, Logistic Regression, and Neural Networks** use matrices.

✔ **Transformations** → Rotating, scaling, and manipulating data (used in images, NLP, etc.).

✔ **Optimization** → Gradient Descent and backpropagation in Deep Learning.

**📌 Key Linear Algebra Concepts in Data Science**

**1️⃣ Scalars, Vectors, Matrices, and Tensors**

* **Scalar** → A single numerical value.a=5

a=5a = 5

* **Vector** → A **1D array** (row or column of numbers).

v=235 2 3 5

* **Matrix** → A **2D array** (rows × columns of numbers).

M=[123456] 1 & 2 & 3 4 & 5 & 6

* **Tensor** → A **multi-dimensional array** (3D or higher). Used in Deep Learning.

**2️⃣ Matrix Operations**

* **Addition & Subtraction** → Element-wise operations.
* **Multiplication** → Dot product, Hadamard product.
* **Transpose** → Flipping a matrix across its diagonal.
* **Determinant & Inverse** → Used in solving equations.

**3️⃣ Eigenvalues & Eigenvectors**

Used in **Principal Component Analysis (PCA)** to reduce dimensionality in datasets.

Av=λv

where:

✔ **A** is a matrix,

✔ **v** is an eigenvector,

✔ **λ** is an eigenvalue.

Eigenvectors help in **finding patterns and reducing noise** in data.

**📌 Applications of Linear Algebra in Data Science**

✔ **Image Processing** → Matrices are used to store pixel values.

✔ **Natural Language Processing (NLP)** → Text data is represented as vectors.

✔ **Recommendation Systems** → User-item matrices help in predicting preferences.

✔ **Neural Networks** → Weights and activations are stored as matrices.

**🔗 Final Thoughts**

Linear algebra is the **foundation** of data science and machine learning. Understanding matrices, vectors, and transformations helps in **building efficient models** and performing **complex computations** easily. 🚀

**📌 Concepts of Matrices**

**🔹 What is a Matrix?**

A **matrix** is a rectangular array of numbers arranged in **rows** and **columns**.

📌 **Example of a Matrix:**

A=[123456789]

1 & 2 & 3

4 & 5 & 6

7 & 8 & 9

This is a **3×3** matrix (3 rows, 3 columns).

**🔹 Basic Terms in Matrices**

| **Term** | **Description** | **Example** |
| --- | --- | --- |
| **Element** | Each individual number in a matrix | In matrix A, **5** is an element (2nd row, 2nd column) |
| **Row** | A horizontal collection of elements | 1st row: **[1, 2, 3]** |
| **Column** | A vertical collection of elements | 2nd column: **[2, 5, 8]** |
| **Dimension (Order)** | Number of rows × Number of columns | Matrix A has **3×3 dimensions** |
| **Square Matrix** | Rows = Columns | **3×3, 4×4** matrices |
| **Rectangular Matrix** | Rows ≠ Columns | **2×3, 3×4** matrices |
| **Row Vector** | A matrix with only one row | **[1, 2, 3]** (1×3) |
| **Column Vector** | A matrix with only one column | **[4, 5, 6]ᵀ** (3×1) |
| **Zero Matrix (O)** | A matrix where all elements are 0 | **O =** [0000] |
| 0 & 0 |  |  |
| 0 & 0 |  |  |
| **Identity Matrix (I)** | A square matrix with 1s on the diagonal | I = |
| 1 & 0 & 0 |  |  |
| 0 & 1 & 0 |  |  |
| 0 & 0 & 1 |  |  |
|  |  |  |
| **Diagonal Matrix** | A square matrix where only the diagonal has values | a & 0 & 0 |
| 0 & b & 0 |  |  |
| 0 & 0 & c |  |  |
| **Scalar Matrix** | A diagonal matrix where all diagonal values are the same | k & 0 & 0 |
| 0 & k & 0 |  |  |
| 0 & 0 & k |  |  |
| **Triangular Matrix** | A matrix where all elements **above or below** the diagonal are zero | a & b & c |
| 0 & d & e |  |  |
| 0 & 0 & f |  |  |
| **Symmetric Matrix** | A matrix where A=A^T | a & b & c |
| b & d & e |  |  |
| c & e & f |  |  |

**🔹 Matrix Representation in Python (Without Libraries)**

python

CopyEdit

# Defining a 3x3 matrix manually

matrix\_A = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

# Accessing elements (Example: Element at 2nd row, 2nd column)

element = matrix\_A[1][1] # Indexing starts from 0, so matrix\_A[1][1] gives 5

# Printing the matrix

for row in matrix\_A:

print(row)

**🔹 Output:**

csharp

CopyEdit

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

**🚀 Summary**

✔️ A **matrix** is a 2D array of numbers with **rows and columns**.

✔️ **Order of a matrix** is given as (Rows × Columns).

✔️ Special matrices include **Identity, Zero, Diagonal, and Triangular matrices**.

✔️ We can **represent matrices in Python** using lists.

**📌 Matrix Representation & Transformations**

**🔹 Different Ways to Represent Matrices in Python (Without Libraries)**

**1️⃣ Using Lists (Nested Lists)**

Python lists can be used to represent matrices by storing each row as a list inside a larger list.

python

CopyEdit

# 3x3 Matrix representation using nested lists

matrix\_A = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

# Accessing elements (Example: 2nd row, 3rd column)

element = matrix\_A[1][2] # Indexing starts from 0, so matrix\_A[1][2] gives 6

# Printing the matrix

for row in matrix\_A:

print(row)

**🔹 Output:**

csharp

CopyEdit

[1, 2, 3]

[4, 5, 6]

[7, 8, 9]

**2️⃣ Using Dictionary Representation**

Instead of using lists, we can use dictionaries to store matrix values with keys as coordinates.

python

CopyEdit

# Dictionary representation of a sparse matrix

matrix\_dict = {

(0, 0): 1, (0, 2): 3,

(1, 1): 5,

(2, 0): 7, (2, 2): 9

}

# Accessing an element (Example: 2nd row, 2nd column)

element = matrix\_dict.get((1, 1), 0) # Returns 5 if exists, else returns 0

print(f"Element at (1,1): {element}")

**🔹 Output:**

java

CopyEdit

Element at (1,1): 5

👉 **Useful for sparse matrices where most values are zero.**

**🔹 Matrix Transformations**

Matrix transformations involve modifying matrices, such as **transposition, rotation, scaling, reflection, etc.**

**1️⃣ Transpose of a Matrix**

Transposing swaps rows with columns.

Original Matrix (A): [1, 2] [3, 4]

Transposed Matrix (A^T): [1, 3] [2, 4]

**Python Implementation:**

python

CopyEdit

# Transposing a matrix manually

matrix\_A = [

[1, 2, 3],

[4, 5, 6]

]

# Transpose (Flipping rows to columns)

transpose\_A = [[matrix\_A[j][i] for j in range(len(matrix\_A))] for i in range(len(matrix\_A[0]))]

# Printing the transposed matrix

for row in transpose\_A:

print(row)

**🔹 Output:**

csharp

CopyEdit

[1, 4]

[2, 5]

[3, 6]

**2️⃣ Rotating a Matrix (90° Clockwise)**

Rotating a matrix by **90° clockwise** is done by transposing it and then reversing each row.

python

CopyEdit

# 90-degree rotation function

def rotate\_90(matrix):

return [list(reversed(col)) for col in zip(\*matrix)]

matrix\_B = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

rotated\_matrix = rotate\_90(matrix\_B)

# Printing rotated matrix

for row in rotated\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[7, 4, 1]

[8, 5, 2]

[9, 6, 3]

**3️⃣ Scaling a Matrix (Multiplication by a Scalar)**

Scaling multiplies all elements by a scalar value.

python

CopyEdit

# Scaling a matrix

scalar = 3

matrix\_C = [

[1, 2],

[3, 4]

]

scaled\_matrix = [[scalar \* element for element in row] for row in matrix\_C]

# Printing the scaled matrix

for row in scaled\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[3, 6]

[9, 12]

**4️⃣ Reflection (Flipping a Matrix)**

Reflecting a matrix along an axis, such as **horizontal or vertical**.

python

CopyEdit

# Horizontal Flip

matrix\_B = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

def flip\_horizontal(matrix):

return matrix[::-1]

flipped\_matrix = flip\_horizontal(matrix\_B)

# Printing flipped matrix

for row in flipped\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[7, 8, 9]

[4, 5, 6]

[1, 2, 3]

**🚀 Summary**

✔️ Matrices can be represented using **lists, dictionaries (for sparse matrices), or tuples**.

✔️ **Transformations include:**

* **Transpose** (flipping rows & columns)
* **Rotation** (90° clockwise/counterclockwise)
* **Scaling** (multiplying all elements by a scalar)
* **Reflection** (flipping horizontally or vertically)

**📌 Basic Matrix Operations**

Let's cover fundamental matrix operations **without using any libraries** (No NumPy, No SciPy). 🚀

**🔹 1. Matrix Addition**

Adding two matrices is done **element-wise**:

C[i][j]=A[i][j]+B[i][j] C[i][j] = A[i][j] + B[i][j]

C[i][j]=A[i][j]+B[i][j]

🔸 **Conditions:**

✔️ Matrices must have the **same dimensions** (same number of rows & columns).

🔹 **Python Code (Matrix Addition)**

python

CopyEdit

def matrix\_addition(A, B):

rows, cols = len(A), len(A[0])

result = [[A[i][j] + B[i][j] for j in range(cols)] for i in range(rows)]

return result

# Example Matrices

A = [[1, 2, 3], [4, 5, 6]]

B = [[7, 8, 9], [10, 11, 12]]

# Perform Addition

sum\_matrix = matrix\_addition(A, B)

# Print Result

for row in sum\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[8, 10, 12]

[14, 16, 18]

**🔹 2. Matrix Subtraction**

Subtracting two matrices is also **element-wise**:

C[i][j]=A[i][j]−B[i][j] C[i][j] = A[i][j] - B[i][j]

C[i][j]=A[i][j]−B[i][j]

🔸 **Conditions:**

✔️ Matrices must have the **same dimensions**.

🔹 **Python Code (Matrix Subtraction)**

python

CopyEdit

def matrix\_subtraction(A, B):

rows, cols = len(A), len(A[0])

result = [[A[i][j] - B[i][j] for j in range(cols)] for i in range(rows)]

return result

# Example Matrices

A = [[1, 2, 3], [4, 5, 6]]

B = [[7, 8, 9], [10, 11, 12]]

# Perform Addition

sum\_matrix = matrix\_subtraction(A, B)

# Print Result

for row in sum\_matrix:

print(row)

**🔹 Output:**

css

CopyEdit

[-6, -6, -6]

[-6, -6, -6]

**🔹 3. Matrix Multiplication**

* **Types of Matrix Multiplications**
  1. **Scalar Multiplication** 🔢
     + Multiply each element of the matrix by a scalar (single number).
     + Example: 2×[1324]=[2648]

2×[1234]=[2468]2 \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}

* 1. **Dot Product (Scalar Product)** 🎯
     + For vectors: Multiply corresponding elements and sum them up.
     + Example: [1,2,3]⋅[4,5,6]=(1×4)+(2×5)+(3×6)=32

[1,2,3]⋅[4,5,6]=(1×4)+(2×5)+(3×6)=32[1, 2, 3] \cdot [4, 5, 6] = (1×4) + (2×5) + (3×6) = 32

* 1. **Matrix Multiplication (Dot Product for Matrices)** 🏗
     + Multiply **rows of first matrix** with **columns of second matrix**.
     + Example: [1324]×[5768]=[19432250]

[1234]×[5678]=[19224350]\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}

* 1. **Hadamard Product (Element-wise Multiplication)** 🤝
     + Multiply corresponding elements of two matrices of the same size.
     + Example: [1324]∘[5768]=[5211232]

[1234]∘[5678]=[5122132]\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \circ \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}

* 1. **Kronecker Product** 🔄
     + Expands one matrix by multiplying it with every element of the other matrix.
     + Example (conceptually): A⊗B Results in a **larger** block matrix.

A⊗BA \otimes B

* 🔹 **Key Points:**
  1. **Matrix multiplication** follows rules like **row × column**, not element-wise.
  2. **Hadamard product** is element-wise, but **dot product** follows matrix rules.
  3. Used in **machine learning, physics, graphics, and engineering**!

**A. Scalar Multiplication**

Each element is multiplied by a scalar value:

C[i][j]=k×A[i][j] C[i][j] = k \times A[i][j]

C[i][j]=k×A[i][j]

🔹 **Python Code (Scalar Multiplication)**

python

CopyEdit

def scalar\_multiplication(matrix, scalar):

return [[scalar \* element for element in row] for row in matrix]

A = [[1, 2, 3], [4, 5, 6]]

# Example Scalar Multiplication

scaled\_matrix = scalar\_multiplication(A, 2)

# Print Result

for row in scaled\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[2, 4, 6]

[8, 10, 12]

**B. Matrix Multiplication (Dot Product)**

Matrix multiplication is NOT element-wise. Instead, it follows this rule:

C[i][j]=k=0∑mA[i][k]×B[k][j]

🔸 **Conditions:**

✔️ **A's columns must match B's rows**.

✔️ If **A is (m × n) and B is (n × p)**, then the result **C will be (m × p)**.

🔹 **Python Code (Matrix Multiplication)**

python

CopyEdit

def matrix\_multiplication(A, B):

rows\_A, cols\_A = len(A), len(A[0])

rows\_B, cols\_B = len(B), len(B[0])

# Ensure A's columns = B's rows

if cols\_A != rows\_B:

raise ValueError("Matrix multiplication not possible: A's columns must match B's rows")

# Multiply Matrices

result = [[sum(A[i][k] \* B[k][j] for k in range(cols\_A)) for j in range(cols\_B)] for i in range(rows\_A)]

return result

# Example Matrices

A = [[1, 2], [3, 4]]

B = [[5, 6], [7, 8]]

# Perform Multiplication

product\_matrix = matrix\_multiplication(A, B)

# Print Result

for row in product\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[19, 22]

[43, 50]

**🔹 4. Matrix Transpose**

Swapping rows and columns:

AT[i][j]=A[j][i] A^T[i][j] = A[j][i]

AT[i][j]=A[j][i]

🔹 **Python Code (Matrix Transpose)**

python

CopyEdit

def transpose(matrix):

return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

A = [[1, 2], [3, 4]]

# Perform Transpose

transpose\_matrix = transpose(A)

# Print Result

for row in transpose\_matrix:

print(row)

**🔹 Output:**

csharp

CopyEdit

[1, 3]

[2, 4]

**🔹 5. Matrix Determinant (Only for 2×2 Matrices)**

For a **2×2 matrix**:

det(A)=(a×d)−(b×c)

🔹 **Python Code (Determinant of 2×2 Matrix)**

python

CopyEdit

def determinant\_2x2(matrix):

if len(matrix) != 2 or len(matrix[0]) != 2:

raise ValueError("Determinant can only be calculated for 2x2 matrices")

return (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0])

# Example

A = [[1, 2], [3, 4]]

det\_A = determinant\_2x2(A)

print(f"Determinant of A: {det\_A}")

**🔹 Output:**

css

CopyEdit

Determinant of A: -2

**🚀 Summary**

✔ **Addition & Subtraction** → Only for same-sized matrices (Element-wise).

✔ **Multiplication**:

* **Scalar Multiplication** → Each element multiplied by a constant.
* **Matrix Multiplication** → Uses dot product (A’s columns must match B’s rows).✔ **Transpose** → Swap rows and columns.✔ **Determinant** → Defined for **square matrices (2×2 example given).**

**📌 Types of Matrices**

Matrices come in different forms based on their **structure, elements, and properties**. Let’s explore the most important ones! 🚀

**🔹 1. Square Matrix**

A **square matrix** has the **same number of rows and columns**:

An×nA\_{n \times n}

An×n

✅ **Example (3×3 Square Matrix):**

A=[123456789] 1 & 2 & 3 4 & 5 & 6 7 & 8 & 9

🔹 **Python Code:**

python

CopyEdit

def is\_square\_matrix(matrix):

return len(matrix) == len(matrix[0])

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(is\_square\_matrix(A)) # Output: True

**🔹 2. Diagonal Matrix**

A **diagonal matrix** has **non-zero elements only on the diagonal**, and **all other elements are zero**: 1 & 0 & 0 0 & 2 & 0 0 & 0 & 3

🔹 **Python Code:**

python

CopyEdit

def is\_diagonal(matrix):

for i in range(len(matrix)):

for j in range(len(matrix[0])):

if i != j and matrix[i][j] != 0:

return False

return True

D = [[5, 0, 0], [0, 3, 0], [0, 0, 7]]

print(is\_diagonal(D)) # Output: True

**🔹 3. Identity Matrix (Unit Matrix, I)**

An **identity matrix** is a special **diagonal matrix where all diagonal elements are 1**: 1 & 0 & 0 0 & 1 & 0 0 & 0 & 1

🔹 **Python Code:**

python

CopyEdit

def is\_identity(matrix):

for i in range(len(matrix)):

for j in range(len(matrix[0])):

if (i == j and matrix[i][j] != 1) or (i != j and matrix[i][j] != 0):

return False

return True

I = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

print(is\_identity(I)) # Output: True

**🔹 4. Symmetric Matrix**

A **symmetric matrix** is equal to its **transpose**: A = A^T

✅ **Example:** 1 & 2 & 3 2 & 4 & 5 3 & 5 & 6

🔹 **Python Code:**

python

CopyEdit

def is\_symmetric(matrix):

for i in range(len(matrix)):

for j in range(len(matrix)):

if matrix[i][j] != matrix[j][i]:

return False

return True

S = [[1, 2, 3], [2, 4, 5], [3, 5, 6]]

print(is\_symmetric(S)) # Output: True

**🔹 5. Sparse Matrix**

A **sparse matrix** has mostly **zero** elements: 0 & 0 & 0 & 1 0 & 2 & 0 & 0 0 & 0 & 0 & 0

🔹 Used in **machine learning**, **graphs**, **image processing**.

🔹 **Python Code:**

python

CopyEdit

def is\_sparse(matrix):

total\_elements = len(matrix) \* len(matrix[0])

zero\_count = sum(matrix[i][j] == 0 for i in range(len(matrix)) for j in range(len(matrix[0])))

return zero\_count > (total\_elements / 2) # More than 50% zeros

S = [[0, 0, 0, 1], [0, 2, 0, 0], [0, 0, 0, 0]]

print(is\_sparse(S)) # Output: True

**🔹 6. Dense Matrix**

A **dense matrix** has **few or no** zero elements: 1 & 2 & 3 4 & 5 & 6 7 & 8 & 9

🔹 Opposite of a **sparse matrix**.

🔹 Most real-world datasets **start as dense** but can be optimized into sparse matrices.

🔹 **Python Code:**

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

def check(matrix):

total\_elements=len(matrix)\*len(matrix[0])

zeros=sum(matrix[i][j]==0 for i in range(len(matrix)) for j in range(len(matrix[0])))

return zeros<(total\_elements/2)

print(check(A))

**🚀 Summary**

| **Matrix Type** | **Properties** |
| --- | --- |
| **Square Matrix** | Same rows & columns (n×n) |
| **Diagonal Matrix** | Non-zero only on the diagonal |
| **Identity Matrix** | Diagonal = 1, all else = 0 |
| **Symmetric Matrix** | A = Aᵀ (Transpose is same) |
| **Sparse Matrix** | Mostly zero elements (>50%) |
| **Dense Matrix** | Few or no zero elements |

**Dot Product, Eigenvalues & Eigenvectors**

**Dot Product**

The **dot product** (also called **scalar product**) is a mathematical operation that **takes two equal-length vectors and returns a single scalar (number).**

**1️⃣ Formula of Dot Product**

For two vectors **A** and **B**:

A · B = a₁b₁ + a₂b₂ + ... + aₙbₙ

In **summation notation**:

A · B = ∑(aᵢ \* bᵢ) for i = 1 to n

**2️⃣ Example Calculation**

Let’s say we have two vectors:

A = [2, 3, 4]

B = [1, 0, -1]

The **dot product** is:

A · B = (2 × 1) + (3 × 0) + (4 × -1) = 2 + 0 - 4 = -2

**3️⃣ Geometric Interpretation**

The dot product also relates to the **angle (θ) between two vectors**:

A · B = |A| \* |B| \* cos(θ)

where:

* **|A|** and **|B|** are the magnitudes (lengths) of vectors **A** and **B**
* **θ** is the angle between them
* **If A · B = 0, the vectors are perpendicular (orthogonal)**

**4️⃣ Python Implementation (Without NumPy)**

python

CopyEdit

def dot\_product(A, B):

return sum(a \* b for a, b in zip(A, B))

A = [2, 3, 4]

B = [1, 0, -1]

result = dot\_product(A, B)

print("Dot Product:", result) # Output: -2

**5️⃣ Key Properties of Dot Product**

1. **Commutative Property**:

A · B = B · A

1. **Distributive Property**:

A · (B + C) = A · B + A · C

1. **If the dot product is zero**:

A · B = 0 → Vectors are **perpendicular (orthogonal)**

🔥 **Conclusion:**

The **dot product** helps in **vector projections, measuring angles, and machine learning applications (e.g., cosine similarity in NLP & recommendation systems).** 🚀

**Eigenvalues & Eigenvectors**

Eigenvalues and eigenvectors are fundamental concepts in linear algebra that help in dimensionality reduction (PCA), stability analysis, and transformations in machine learning & data science.

**1. What are Eigenvalues & Eigenvectors?**

For a square matrix A, an eigenvector **v** and an eigenvalue **λ** satisfy:

A \* v = λ \* v

* **Eigenvector (v):** A nonzero vector whose direction remains unchanged after applying the matrix transformation.
* **Eigenvalue (λ):** A scalar factor by which the eigenvector is scaled.

**2. Why are They Important?**

* **Principal Component Analysis (PCA)** – Used for dimensionality reduction by selecting important features.
* **Data Compression** – Helps in reducing redundant information in high-dimensional data.
* **Stability Analysis** – Used in dynamical systems and optimization.
* **Markov Chains & Graphs** – Helps in network analysis, PageRank, and recommendation systems.

**3. How to Compute Eigenvalues & Eigenvectors?**

**Step 1: Compute the Characteristic Equation**

For a matrix A, solve:

det(A - λI) = 0

where **I** is the identity matrix and **det()** represents the determinant.

**Step 2: Solve for Eigenvalues (λ)**

Find the roots of the determinant equation.

**Step 3: Solve for Eigenvectors (v)**

For each eigenvalue **λ**, solve:

(A - λI)v = 0

to get the eigenvectors.

**4. Example Calculation**

Given matrix:

A = | 4 -2 |

| 1 1 |

**Step 1: Characteristic Equation**

Compute determinant of (A - λI)

| (4 - λ) -2 |

| 1 (1 - λ) | = 0

# Expanding the determinant:

(4 - λ)(1 - λ) - (-2)(1) = 0

4 - 4λ - λ + λ² + 2 = 0

λ² - 5λ + 6 = 0

# Solving for λ:

(λ - 2)(λ - 3) = 0

# Eigenvalues:

λ₁ = 2, λ₂ = 3

**Step 2: Solve for Eigenvectors**

For **λ = 2**: Solve (A - 2 I )v = 0

(A - 2I) =

| 2 -2 |

| 1 -1 |

# Solve:

2x - 2y = 0 → x = y

# Eigenvector:

v₁ = [ 1, 1 ]

# For λ = 3, solve (A - 3I)v = 0

(A - 3I) =

| 1 -2 |

| 1 -2 |

# Solve:

x - 2y = 0 → x = 2y

# Eigenvector:

v₂ = [ 2, 1 ]x

**5. Python Implementation**

python

CopyEdit

import numpy as np

# Define matrix A

A = np.array([[4, -2],

[1, 1]])

# Compute eigenvalues & eigenvectors

eigenvalues, eigenvectors = np.linalg.eig(A)

print("Eigenvalues:", eigenvalues)

print("Eigenvectors:\\n", eigenvectors)

**6. Eigenvalues & PCA (Dimensionality Reduction)**

* **PCA (Principal Component Analysis)** uses eigenvalues & eigenvectors to find principal components that retain the most variance in data.
* Higher eigenvalues indicate important directions (features) in the data.
* By keeping only the top-k eigenvectors, we can reduce dimensions while preserving key information.

**Conclusion**

Eigenvalues and eigenvectors help transform data, reduce dimensionality, and analyze stability in machine learning and data science. They play a key role in PCA, Markov Chains, Graph Theory, and Quantum Computing.

**Eigenvalues & Eigenvectors Calculation**

**Step 1: Compute Eigenvalues**

Eigenvalues **λ** satisfy the characteristic equation:

det(A - λI) = 0

For a **2×2 matrix**:

A = | a b |

| c d |

The characteristic equation is:

| a-λ b |

| c d-λ | = 0

Expanding the determinant:

(a - λ)(d - λ) - (b \* c) = 0

Solving this quadratic equation gives the **eigenvalues**.

**Step 2: Compute Eigenvectors**

For each eigenvalue **λ**, solve:

(A - λI) \* v = 0

where **v** is the eigenvector.

**Python Implementation (Manual Computation)**

import numpy as np

def compute\_eigenvalues(A):

"""Compute eigenvalues manually by solving det(A - λI) = 0"""

a, b, c, d = A[0, 0], A[0, 1], A[1, 0], A[1, 1]

# Solve the characteristic equation λ^2 - (a+d)λ + (ad - bc) = 0

trace = a + d # tr(A)

determinant = (a \* d) - (b \* c) # det(A)

# Solve for λ using the quadratic formula

eigenvalue1 = (trace + np.sqrt(trace\*\*2 - 4 \* determinant)) / 2

eigenvalue2 = (trace - np.sqrt(trace\*\*2 - 4 \* determinant)) / 2

return eigenvalue1, eigenvalue2

def compute\_eigenvector(A, eigenvalue):

"""Compute eigenvector manually by solving (A - λI)v = 0"""

I = np.eye(A.shape[0]) # Identity matrix

A\_lambda\_I = A - eigenvalue \* I # (A - λI)

# Solve for v in (A - λI)v = 0 (Ax = 0)

# Choose one equation (row 1) to solve for x, y

if A\_lambda\_I[0, 0] != 0:

x = 1

y = -A\_lambda\_I[0, 1] / A\_lambda\_I[0, 0] # Solve for y

else:

x = -A\_lambda\_I[1, 1] / A\_lambda\_I[1, 0]

y = 1

eigenvector = np.array([x, y])

return eigenvector / np.linalg.norm(eigenvector) # Normalize the eigenvector

# Example matrix

A = np.array([[4, -2],

[1, 1]])

# Compute eigenvalues

eigenvalues = compute\_eigenvalues(A)

print("Eigenvalues:", eigenvalues)

# Compute eigenvectors

eigenvectors = [compute\_eigenvector(A, ev) for ev in eigenvalues]

print("Eigenvectors:")

for v in eigenvectors:

print(v)

**Example Output (For A = [[4, -2], [1, 1]])**

Eigenvalues: (3.0, 2.0)

Eigenvectors:

[0.894, 0.447] # Eigenvector for λ = 3

[0.707, 0.707] # Eigenvector for λ = 2

This manually calculates the **eigenvalues and eigenvectors** without numpy.linalg.eig(). 🚀

**Advanced Matrix Operations & Case Study**

**📌 Advanced Matrix Operations**

In this section, we’ll cover some advanced matrix operations essential for **Linear Algebra in Data Science and Machine Learning**. Let’s break them down one by one! 🚀

**🔹 1. Matrix Inverse (A⁻¹)**

The **inverse of a matrix** A is a matrix A⁻¹ such that:

**A × A⁻¹ = I** (Identity Matrix)

🔹 **Only square matrices with non-zero determinants have an inverse.**

🔹 If **det(A) = 0**, the inverse **does not exist** (singular matrix).

**✅ Example: Find the Inverse of a 2×2 Matrix**

Let’s take a matrix:

makefile

CopyEdit

A = | 4 7 |

| 2 6 |

**Step 1: Find the determinant of A**

markdown

CopyEdit

det(A) = (4 × 6) - (7 × 2)

= 24 - 14

= 10

**Step 2: Use the formula for a 2×2 matrix inverse**

For any 2×2 matrix:

makefile

CopyEdit

A = | a b |

| c d |

The inverse is calculated as:

markdown

CopyEdit

A⁻¹ = (1 / det(A)) × | d -b |

| -c a |

**Step 3: Apply the formula**

markdown

CopyEdit

A⁻¹ = (1 / 10) × | 6 -7 |

| -2 4 |

A⁻¹ = | 0.6 -0.7 |

| -0.2 0.4 |

🔹 **Python Code (without libraries!):**

python

CopyEdit

def determinant\_2x2(matrix):

return (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0])

def inverse\_2x2(matrix):

det = determinant\_2x2(matrix)

if det == 0:

return "Inverse does not exist"

return [[matrix[1][1] / det, -matrix[0][1] / det],

[-matrix[1][0] / det, matrix[0][0] / det]]

A = [[4, 7], [2, 6]]

A\_inv = inverse\_2x2(A)

print(A\_inv)

**🔹 2. Matrix Rank**

**Rank** is the number of **linearly independent rows or columns** in a matrix. It tells us:

✔ Whether a system of linear equations has a **unique solution**.

✔ Whether the matrix is **full rank** or **singular** (dependent rows).

✅ **Example:**

Let’s take a **3×3 matrix A**:

makefile

CopyEdit

A =

| 1 2 3 |

| 4 5 6 |

| 7 8 9 |

If we take the **transpose (Aᵀ)**, we swap the rows and columns:

CopyEdit

Aᵀ =

| 1 4 7 |

| 2 5 8 |

| 3 6 9 |

Here, row 3 is a linear combination of rows 1 & 2 → **Rank = 2**.

**Why Does Rank Matter?**

✔ If a matrix has **full rank**, it means it holds the maximum possible independent information.

✔ If a matrix has **low rank**, it means some rows/columns are redundant (they don't add new information).

🔹 **Python Code (Manual Check for Rank):**

python

CopyEdit

def rank\_manual(matrix):

rows = len(matrix)

cols = len(matrix[0])

return min(rows, cols) # Simplified approach

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(rank\_manual(A)) # Output: 2 (Approx)

def row\_echelon\_form(matrix):

"""Convert matrix to row echelon form to determine rank"""

rows, cols = len(matrix), len(matrix[0])

rank = 0

for i in range(min(rows, cols)):

if matrix[i][i] == 0:

for j in range(i+1, rows):

if matrix[j][i] != 0:

matrix[i], matrix[j] = matrix[j], matrix[i] # Swap rows

break

if matrix[i][i] != 0:

rank += 1

for j in range(i+1, rows):

factor = matrix[j][i] / matrix[i][i]

for k in range(cols):

matrix[j][k] -= factor \* matrix[i][k]

return rank

A = [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

print("Rank of A:", row\_echelon\_form(A)) # Output: 2

**🔹 3. Determinant**

The **determinant** tells us:

✔ Whether a matrix is **invertible** (det ≠ 0)

✔ The **scaling factor** of transformations

✅ **Formula for a 2×2 matrix:**

det(A) = ad - bc

A = [ a b ] [ c d ]

✅ **Formula for a 3×3 matrix:**

det(A) = a(ei − fh) - b(di − fg) + c(dh − eg)

A = | a b c | | d e f | | g h i |

🔹 **Python Code (Without Libraries):**

python

CopyEdit

def determinant\_3x3(matrix):

return (matrix[0][0] \* (matrix[1][1] \* matrix[2][2] - matrix[1][2] \* matrix[2][1]) -

matrix[0][1] \* (matrix[1][0] \* matrix[2][2] - matrix[1][2] \* matrix[2][0]) +

matrix[0][2] \* (matrix[1][0] \* matrix[2][1] - matrix[1][1] \* matrix[2][0]))

A = [[2, 3, 1], [4, 5, 6], [7, 8, 9]]

print(determinant\_3x3(A)) # Output: Determinant value

**🔹 4. Matrix Trace**

The **trace** of a square matrix is **the sum of its diagonal elements**.

✅ **Example:**

A = | a11 a12 a13 | | a21 a22 a23 | | a31 a32 a33 |

tr(A) = a11 + a22 + a33

🔹 **Python Code (Without Libraries):**

python

CopyEdit

def trace(matrix):

return sum(matrix[i][i] for i in range(len(matrix)))

A = [[3, 2, 1], [4, 5, 6], [7, 8, 9]]

print(trace(A)) # Output: 3 + 5 + 9 = 17

**🔹 5. Orthogonality of Matrices**

A matrix is **orthogonal** if its **transpose is equal to its inverse**:

A⁻¹ = Aᵀ

**Aᵀ × A= I** (Identity Matrix).

✔ If **A is orthogonal**, its **columns are unit vectors and perpendicular to each other**.

✔ Important for **QR decomposition** and **dimensionality reduction**.

**Example:**

Matrix **Q**:

makefile

CopyEdit

Q = | 0 -1 |

| 1 0 |

Transpose of **Q** (Qᵀ):

markdown

CopyEdit

Qᵀ = | 0 1 |

| -1 0 |

Now, multiplying **Qᵀ × Q**:

markdown

CopyEdit

Qᵀ × Q = | 0 1 | × | 0 -1 |

| -1 0 | | 1 0 |

= | (0×0 + 1×1) (0×-1 + 1×0) |

| (-1×0 + 0×1) (-1×-1 + 0×0) |

= | 1 0 |

| 0 1 |

This is the **Identity Matrix (I)**, proving that **Q is an orthogonal matrix**.

🔹 **Python Code (Check if a matrix is orthogonal):**

python

CopyEdit

def transpose(matrix):

return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

def multiply\_matrices(A, B):

return [[sum(A[i][k] \* B[k][j] for k in range(len(A))) for j in range(len(B[0]))] for i in range(len(A))]

def is\_orthogonal(matrix):

transposed = transpose(matrix)

identity = multiply\_matrices(matrix, transposed)

return identity == [[1 if i == j else 0 for j in range(len(matrix))] for i in range(len(matrix))]

Q = [[0, -1], [1, 0]]

print(is\_orthogonal(Q)) # Output: True

**🚀 Summary of Advanced Operations**

| **Operation** | **Purpose** |
| --- | --- |
| **Inverse** | Used to solve systems of equations (only for non-singular matrices) |
| **Rank** | Number of linearly independent rows/columns |
| **Determinant** | Checks if the matrix is invertible (det ≠ 0) |
| **Trace** | Sum of diagonal elements (used in eigenvalues, transformations) |
| **Orthogonality** | Used in QR decomposition and data transformations |

**📌 Data Science Case Study: Fraud Detection in Banking**

**🟢 Problem Statement**

Banks and financial institutions face a major issue with **credit card fraud**. Every year, millions of dollars are lost due to fraudulent transactions. Traditional rule-based fraud detection systems often fail to detect **new types of fraud**.

**🔍 How Data Science Solves This?**

By using **Machine Learning models**, banks can detect fraudulent transactions **in real time** by analyzing transaction patterns and user behavior.

**📌 Step-by-Step Data Science Workflow**

**🔹 Step 1: Data Collection**

Banks collect massive transaction data, including:

✔ Transaction Amount

✔ Location (Country, City)

✔ Time of Transaction

✔ Type of Merchant (Retail, Online, ATM, etc.)

✔ Cardholder Details (Age, Past Transactions, Credit Score)

🔹 Example Dataset:

| **Transaction ID** | **Amount ($)** | **Location** | **Time** | **Merchant Type** | **Fraud (0/1)** |
| --- | --- | --- | --- | --- | --- |
| 1001 | 500 | New York | 11:30 AM | Online Store | 0 |
| 1002 | 1500 | California | 2:00 AM | ATM Withdrawal | 1 |
| 1003 | 100 | Texas | 6:00 PM | Grocery Store | 0 |
| 1004 | 5000 | Russia | 3:45 AM | Electronics Store | 1 |

**🔹 Step 2: Data Cleaning & Preprocessing**

✔ **Handle Missing Data:** Fill missing values in location, merchant type, or amount.

✔ **Remove Duplicates:** If the same transaction appears twice, remove one.

✔ **Convert Categorical Data:** Change **Location, Merchant Type** into numerical format.

✔ **Feature Engineering:** Create new features like **Transaction Frequency, Daily Spend Limit** to help models detect fraud.

🔹 **Python Code for Data Cleaning (Without Pandas):**

python

CopyEdit

transactions = [

{"id": 1001, "amount": 500, "location": "New York", "time": "11:30 AM", "merchant": "Online Store", "fraud": 0},

{"id": 1002, "amount": 1500, "location": "California", "time": "2:00 AM", "merchant": "ATM Withdrawal", "fraud": 1},

{"id": 1003, "amount": 100, "location": "Texas", "time": "6:00 PM", "merchant": "Grocery Store", "fraud": 0},

{"id": 1004, "amount": 5000, "location": "Russia", "time": "3:45 AM", "merchant": "Electronics Store", "fraud": 1}

]

# Convert time to 24-hour format for better analysis

for txn in transactions:

time\_parts = txn["time"].split()

hour, minute = map(int, time\_parts[0].split(":"))

if time\_parts[1] == "PM" and hour != 12:

hour += 12

txn["time"] = hour

print(transactions)

**🔹 Step 3: Exploratory Data Analysis (EDA)**

✔ Check **Fraudulent vs. Non-Fraudulent** transactions

✔ Find **High-Risk Locations & Merchant Types**

✔ Analyze **Transaction Timing** (Frauds often occur late at night!)

🔹 **Example Insights from EDA:**

✔ **Fraudulent transactions are 80% more likely to happen between 12 AM - 5 AM.**

✔ **Transactions above $2000 have a 60% higher fraud risk.**

✔ **ATM withdrawals in foreign locations have a 75% fraud probability.**

**🔹 Step 4: Model Building (Machine Learning for Fraud Detection)**

We can use models like:

✔ **Logistic Regression** (for binary fraud detection)

✔ **Random Forest** (for better accuracy with multiple features)

✔ **Neural Networks** (for high-volume fraud detection)

🔹 **Simple Fraud Detection Logic (Without ML, Just Rules-Based):**

python

CopyEdit

def detect\_fraud(transaction):

if transaction["amount"] > 2000 or transaction["time"] < 5:

return 1 # Fraud

return 0 # Not Fraud

for txn in transactions:

txn["predicted\_fraud"] = detect\_fraud(txn)

print(transactions)

**🔹 Step 5: Model Deployment**

Once a Machine Learning model is trained, it can be deployed on a **real-time fraud detection system**.

✔ If fraud is detected, **send an alert** to the bank & customer.

✔ Use APIs to integrate ML models with **banking applications**.

**📌 Results & Impact of Data Science in Fraud Detection**

✔ **80%+ accuracy** in fraud detection compared to old rule-based systems.

✔ **Faster detection** (real-time processing with ML models).

✔ **Fewer False Alarms**, as ML models learn real customer behavior.

✅ **Real-World Example:**

**J.P. Morgan & Visa** use AI-powered fraud detection models, saving millions in fraudulent transactions yearly.

**🚀 Summary of the Case Study**

| **Step** | **Process** |
| --- | --- |
| **1. Data Collection** | Gather transactions (amount, time, location, type) |
| **2. Data Cleaning** | Handle missing values, encode categorical data |
| **3. EDA** | Analyze fraud patterns (timing, amount, location) |
| **4. Model Building** | Use ML (Logistic Regression, Random Forest, Neural Networks) |
| **5. Deployment** | Integrate with real-time banking systems |

**Additional**

* Mathematics fundamentals

Mathematics is the backbone of data science! Here are the fundamental topics you need to master:

**1️⃣ Linear Algebra (Essential for ML & Deep Learning)**

* + **Vectors & Matrices**: Basics of vectors, matrices, and operations (addition, multiplication, transpose, inverse, determinant).
  + **Eigenvalues & Eigenvectors**: Used in PCA (dimensionality reduction).
  + **Matrix Factorization**: SVD, LU, and QR decomposition (important for recommendation systems).
  + **Dot Product & Norms**: Measures similarity between vectors.

📌 *Applications:* Image processing, NLP, recommendation systems.

**2️⃣ Probability & Statistics (For Data Analysis & ML)**

* + **Basic Probability**: Random variables, conditional probability, Bayes’ Theorem.
  + **Distributions**: Normal, Binomial, Poisson, Uniform (understanding data spread).
  + **Expectation & Variance**: Mean, median, mode, standard deviation.
  + **Hypothesis Testing**: p-value, confidence intervals, t-tests, chi-square test.
  + **Bayesian Inference**: Prior, likelihood, posterior (important in ML & A/B testing).

📌 *Applications:* A/B testing, predictive modeling, uncertainty estimation.

**3️⃣ Calculus (For Optimization & ML Models)**

* + **Derivatives & Gradients**: Helps in understanding how models learn.
  + **Partial Derivatives**: Used in multivariable optimization (Gradient Descent).
  + **Gradient Descent**: Core of ML algorithms like Neural Networks.
  + **Integrals**: Used in probability distributions and Bayesian inference.

📌 *Applications:* Training ML models, deep learning optimization.

**4️⃣ Optimization Techniques (For Model Training)**

* + **Gradient Descent Variants**: SGD, Adam, RMSProp.
  + **Convex Optimization**: Finding optimal solutions efficiently.
  + **Lagrange Multipliers**: Used in constrained optimization problems.

📌 *Applications:* Hyperparameter tuning, loss minimization.

**5️⃣ Discrete Mathematics (For Algorithms & Data Structures)**

* + **Combinatorics**: Permutations & combinations (useful in probability).
  + **Graph Theory**: Nodes, edges, BFS, DFS (used in social networks, pathfinding).
  + **Set Theory**: Unions, intersections (important for database queries & ML).

📌 *Applications:* Search algorithms, recommendation engines, NLP.

**6️⃣ Linear Regression & Correlation (For Predictions & ML)**

* + **Correlation vs. Causation**: Pearson correlation, Spearman correlation.
  + **Linear Regression**: Least squares, cost function.
  + **R-squared & Adjusted R-squared**: Evaluating model fit.

📌 *Applications:* Predictive analytics, time-series forecasting.

* **Curse of Dimensionality**

**Curse of Dimensionality – The Hidden Danger in High Dimensions**

The **curse of dimensionality** refers to the problems that arise when working with high-dimensional data. As dimensions (features) increase, data becomes sparse, distances lose meaning, and models become inefficient.

**Why is it a Problem?**

1️⃣ **Distance Measures Break Down**

* + In higher dimensions, all points tend to become **equidistant**, making distance-based algorithms (like k-NN, clustering) less effective.
  + Example: In 2D, two random points might be far apart, but in 100D, they are **almost equally far from all other points**.

2️⃣ **Increased Computational Cost**

* + More dimensions = **More parameters** = **More computations**.
  + Training models in high dimensions **requires exponentially more data**.

3️⃣ **Overfitting in Machine Learning**

* + High dimensions allow models to memorize data instead of generalizing.
  + Without enough data, models learn **noise instead of patterns**.

4️⃣ **Data Becomes Sparse**

* + In high dimensions, data points are spread out too thinly, making clustering, density estimation, and nearest neighbor searches ineffective.
  + Example: If you randomly distribute 100 points in a **1D line**, some points will be close. But in a **100D space**, points are **isolated**.

**How to Handle the Curse of Dimensionality?**

✅ **Dimensionality Reduction Techniques**

* + **PCA (Principal Component Analysis)** – Projects data onto fewer dimensions while preserving variance.
  + **t-SNE & UMAP** – Non-linear techniques for visualization and clustering.
  + **Autoencoders** – Neural networks that learn compressed feature representations.

✅ **Feature Selection**

* + Remove **irrelevant** or **correlated** features to reduce dimensionality.

✅ **Regularization**

* + **L1 (Lasso) & L2 (Ridge) Regression** help prevent overfitting in high dimensions.

✅ **Curse-Aware Algorithms**

* + Some models (like Decision Trees and Random Forests) handle high-dimensional data better than distance-based methods.

**Real-World Example**

📌 *Face Recognition:*

* + A raw image has thousands of pixels (features).
  + Without dimensionality reduction, face recognition models would require **massive** datasets to learn effectively.
  + PCA can reduce dimensions while keeping key facial features.
* Applications of Dot Product & Cross Product in Data Science

**1️⃣ Dot Product Applications in Data Science**

✅ **1. Cosine Similarity (Text & Recommendation Systems)**

* + Used in **NLP** and **recommendation systems** to measure similarity between text documents or user preferences.
  + If two vectors (documents, users) have a high dot product, they are **more similar**.

**Formula:**

cos⁡(θ)=A⋅B∥A∥∥B∥\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}

cos(θ)=∥A∥∥B∥A⋅B

**Example:**

* + In **Netflix recommendations**, user preference vectors are compared to movie feature vectors.
  + In **Google Search**, cosine similarity is used to rank search results based on query-document similarity.

✅ **2. Principal Component Analysis (PCA) - Dimensionality Reduction**

* + PCA projects high-dimensional data onto a lower-dimensional space.
  + The **dot product** is used to compute projections of data points onto principal components (eigenvectors).

**Example:**

* + Reducing **image dimensions** while keeping important features.
  + Compressing high-dimensional **customer data** into fewer meaningful features.

✅ **3. Neural Networks (Perceptrons & Weights Computation)**

* + Each neuron in a neural network computes the **dot product of inputs and weights**.
  + This forms the foundation of **forward propagation** in deep learning.

**Formula:**

Z=WTX+bZ = W^T X + b

Z=WTX+b

**Example:**

* + **Image classification (CNNs)**
  + **Sentiment analysis (RNNs, LSTMs)**

✅ **4. Linear Regression & Gradient Computation**

* + In **linear regression**, predictions are made using a dot product: Y=X⋅W+b

Y=X⋅W+bY = X \cdot W + b

* + In **gradient descent**, the dot product is used to compute gradients for weight updates.

**Example:**

* + Predicting house prices based on features like size, location, and number of rooms.

**2️⃣ Cross Product Applications in Data Science**

✅ **1. Feature Engineering for 3D Data**

* + In **computer vision and image processing**, the cross product helps compute **surface normals** from 3D point clouds.
  + Used in **object detection** and **scene reconstruction**.

**Example:**

* + **Autonomous cars** use LiDAR to map the environment in 3D.
  + **Medical imaging** (MRI, CT scans) uses 3D vectors to reconstruct body structures.

✅ **2. Computer Vision – Object Detection & Orientation**

* + In **pose estimation**, the cross product helps find **perpendicular vectors** to align objects correctly.
  + Used in **face recognition** and **gesture recognition**.

**Example:**

* + **Self-driving cars** use cross products to detect lane boundaries.
  + **Augmented reality (AR)** applications align virtual objects in a real-world scene.

✅ **3. Data Visualization – 3D Graphing**

* + When plotting high-dimensional data, the cross product helps compute **viewing angles** and rotations for better visualization.
  + Used in **t-SNE, UMAP, PCA visualizations**.

**Example:**

* + Rotating a **3D scatter plot** in **seaborn/matplotlib** for better insight into clusters.

**🔥 Summary**

| **Concept** | **Dot Product** | **Cross Product** |
| --- | --- | --- |
| **Measures?** | Similarity & Projection | Perpendicular Direction |
| **Used in?** | NLP, Recommendation Systems, PCA, Neural Networks | 3D Vision, Feature Engineering, Robotics |
| **Formula** | ( A \cdot B = | A |
| **Example** | Cosine similarity, regression, neural networks | 3D object detection, self-driving cars |

**Practical**

**Matrix Transformations**

class Matrix:

def \_\_init\_\_(self, data):

self.data = data

self.rows = len(data)

self.cols = len(data[0])

def display(self):

for row in self.data:

print(row)

def transpose(self):

result = [[self.data[j][i] for j in range(self.rows)] for i in range(self.cols)]

return Matrix(result)

def rotate(self):

result = [[self.data[j][i] for j in range(self.rows - 1, -1, -1)] for i in range(self.cols)]

return Matrix(result)

def flip\_horizontal(self):

"""Flips the matrix horizontally (left-right)."""

result = [row[::-1] for row in self.data]

return Matrix(result)

def flip\_vertical(self):

"""Flips the matrix vertically (top-bottom)."""

result = self.data[::-1]

return Matrix(result)

first = Matrix([[2, 3, 5], [7, 9, 4]])

print("\\nOriginal Matrix:")

first.display()

print("\\nTransposed Matrix:")

first.transpose().display()

print("\\nRotated Matrix (90° Clockwise):")

first.rotate().display()

print("\\nFlipped Horizontally (Left-Right):")

first.flip\_horizontal().display()

print("\\nFlipped Vertically (Top-Bottom):")

first.flip\_vertical().display()

**1️⃣ Transpose of a Matrix**

python

CopyEdit

def transpose(matrix):

return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("Transpose:")

for row in transpose(matrix):

print(row)

**2️⃣ Rotating a Matrix (90° Clockwise)**

python

CopyEdit

def rotate\_90\_clockwise(matrix):

return [[matrix[j][i] for j in range(len(matrix)-1, -1, -1)] for i in range(len(matrix[0]))]

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("90° Rotated:")

for row in rotate\_90\_clockwise(matrix):

print(row)

**3️⃣ Scaling a Matrix (Multiplication by a Scalar)**

python

CopyEdit

def scale\_matrix(matrix, scalar):

return [[element \* scalar for element in row] for row in matrix]

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

scalar = 2

print("Scaled Matrix:")

for row in scale\_matrix(matrix, scalar):

print(row)

**4️⃣ Reflection (Flipping a Matrix)**

**Horizontally (Left-Right Flip)**

python

CopyEdit

def flip\_horizontal(matrix):

return [row[::-1] for row in matrix]

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("Horizontally Flipped:")

for row in flip\_horizontal(matrix):

print(row)

**Vertically (Top-Bottom Flip)**

python

CopyEdit

def flip\_vertical(matrix):

return matrix[::-1]

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("Vertically Flipped:")

for row in flip\_vertical(matrix):

print(row)

**Matrix Operations**

class Matrix:

def \_\_init\_\_(self,data):

self.data=data

self.rows=len(data)

self.cols=len(data[0])

def display(self):

for i in self.data:

print(i)

def addition(self,other):

if self.rows!= other.rows or self.cols!=other.cols:

return 'error'

result=[[self.data[i][j]+other.data[i][j] for j in range(self.cols)]for i in range(self.rows)]

return Matrix(result)

def substraction(self,other):

if self.rows!=other.rows or self.cols!=other.cols:

return "error"

result=[]

for i in range(self.rows):

rows=[]

for j in range(self.cols):

rows.append(self.data[i][j]-other.data[i][j])

result.append(rows)

return Matrix(result)

def scalar\_multiplication(self,other):

result=[[self.data[i][j]\*other for j in range(self.cols)]for i in range(self.rows)]

return Matrix(result)

def dot\_product(self,other):

if self.cols!=other.rows:

return "needs to be in same dimension"

result=[[sum(self.data[i][k] \* other.data[k][j] for k in range(self.cols))for j in range(other.cols)]for i in range(self.rows)]

return Matrix(result)

def determinant(self):

if self.rows!=self.cols:

return 'need to be in same dimension'

if self.rows==2:

return self.data[0][0]\*self.data[1][1]-self.data[0][1]\*self.data[1][0]

det=0

for i in range(self.cols):

sub\_matrix=[row[:i]+row[i+1:] for row in self.data[1:]]

det+=((-1)\*\*i)\*self.data[0][i]\*Matrix(sub\_matrix).determinant()

return det

first=Matrix([[1,2,3],[1,2,3],[2,5,4]])

first.display()

second=Matrix([[1,4,2],[1,2,4],[5,3,5]])

add=first.addition(second)

add.display()

sub=first.substraction(second)

sub.display()

scal=first.scalar\_multiplication(3)

scal.display()

dot=first.dot\_product(second)

dot.display()

print(first.determinant())

**1️⃣ Matrix Addition**

python

CopyEdit

def add\_matrices(A, B):

return [[A[i][j] + B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

B = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

print("Matrix Addition:")

for row in add\_matrices(A, B):

print(row)

**2️⃣ Matrix Subtraction**

python

CopyEdit

def subtract\_matrices(A, B):

return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

print("Matrix Subtraction:")

for row in subtract\_matrices(A, B):

print(row)

**3️⃣ Matrix Multiplication**

**A. Scalar Multiplication**

python

CopyEdit

def scalar\_multiply(matrix, scalar):

return [[element \* scalar for element in row] for row in matrix]

scalar = 2

print("Scalar Multiplication:")

for row in scalar\_multiply(A, scalar):

print(row)

**B. Matrix Multiplication (Dot Product)**

python

CopyEdit

def matrix\_multiply(A, B):

result = [[sum(A[i][k] \* B[k][j] for k in range(len(A[0]))) for j in range(len(B[0]))] for i in range(len(A))]

return result

C = [[1, 2], [3, 4], [5, 6]] # 3x2 matrix

D = [[7, 8, 9], [10, 11, 12]] # 2x3 matrix

print("Matrix Multiplication (Dot Product):")

for row in matrix\_multiply(C, D):

print(row)

**4️⃣ Matrix Determinant**

**A. Determinant of a 2×2 Matrix**

python

CopyEdit

def determinant\_2x2(matrix):

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

M2 = [[4, 6], [3, 8]]

print("Determinant of 2x2 Matrix:", determinant\_2x2(M2))

**B. Determinant of a Matrix (More than 2×2) Using Recursion**

python

CopyEdit

def determinant(matrix):

if len(matrix) == 1:

return matrix[0][0]

if len(matrix) == 2:

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

det = 0

for col in range(len(matrix)):

sub\_matrix = [row[:col] + row[col+1:] for row in matrix[1:]]

det += ((-1) \*\* col) \* matrix[0][col] \* determinant(sub\_matrix)

return det

M3 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Example 3x3 matrix

print("Determinant of 3x3 Matrix:", determinant(M3))

**Types of Matrices**

**1️⃣ Square Matrix**

python

CopyEdit

def is\_square(matrix):

return len(matrix) == len(matrix[0])

square\_matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("Is Square Matrix:", is\_square(square\_matrix))

**2️⃣ Diagonal Matrix**

python

CopyEdit

def is\_diagonal(matrix):

for i in range(len(matrix)):

for j in range(len(matrix[i])):

if i != j and matrix[i][j] != 0:

return False

return True

diagonal\_matrix = [[3, 0, 0], [0, 5, 0], [0, 0, 9]]

print("Is Diagonal Matrix:", is\_diagonal(diagonal\_matrix))

**3️⃣ Identity Matrix (Unit Matrix, I)**

python

CopyEdit

def is\_identity(matrix):

for i in range(len(matrix)):

for j in range(len(matrix[i])):

if (i == j and matrix[i][j] != 1) or (i != j and matrix[i][j] != 0):

return False

return True

identity\_matrix = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

print("Is Identity Matrix:", is\_identity(identity\_matrix))

**4️⃣ Symmetric Matrix**

def is\_symmetric(matrix):

for i in range(len(matrix)):

for j in range(len(matrix)):

if matrix[i][j] != matrix[j][i]:

return False

return True

S = [[1, 2, 3], [2, 4, 5], [3, 5, 6]]

print(is\_symmetric(S)) # Output: True

**5️⃣ Sparse Matrix**

python

CopyEdit

def is\_sparse(matrix):

zero\_count = sum(row.count(0) for row in matrix)

total\_elements = len(matrix) \* len(matrix[0])

return zero\_count > (total\_elements // 2) # More than half are zeros

sparse\_matrix = [[0, 0, 3], [0, 5, 0], [0, 0, 0]]

print("Is Sparse Matrix:", is\_sparse(sparse\_matrix))

**6️⃣ Dense Matrix**

python

CopyEdit

def is\_dense(matrix):

return not is\_sparse(matrix)

dense\_matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("Is Dense Matrix:", is\_dense(dense\_matrix))

**Inverse of an n×n matrix**

**🔹 Code for Any n×n Matrix**

python

CopyEdit

A⋅A⁻¹=I

A⁻¹ = (1 / det(A)) \* adj(A)

def determinant(matrix):

"""Compute determinant of an n×n matrix using recursion (Laplace Expansion)."""

size = len(matrix)

if size == 1:

return matrix[0][0]

if size == 2:

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

det = 0

for col in range(size):

sub\_matrix = [row[:col] + row[col+1:] for row in matrix[1:]] # Remove first row & current column

det += ((-1) \*\* col) \* matrix[0][col] \* determinant(sub\_matrix)

return det

def cofactor(matrix):->C\_ij = (-1)^(i+j) \* det(M\_ij)

"""Compute the cofactor matrix of an n×n matrix."""

size = len(matrix)

cofactor\_matrix = [[0] \* size for \_ in range(size)]

for i in range(size):

for j in range(size):

sub\_matrix = [row[:j] + row[j+1:] for row in (matrix[:i] + matrix[i+1:])]

cofactor\_matrix[i][j] = ((-1) \*\* (i + j)) \* determinant(sub\_matrix)

return cofactor\_matrix

def transpose(matrix):

"""Transpose a matrix."""

return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

def inverse(matrix):

"""Compute the inverse of an n×n matrix using determinant and cofactor method."""

det = determinant(matrix)

if det == 0:

return None # No inverse if determinant is 0

cofactor\_matrix = cofactor(matrix)

adjugate\_matrix = transpose(cofactor\_matrix)

# Multiply adjugate by 1/det

inverse\_matrix = [[adjugate\_matrix[i][j] / det for j in range(len(matrix))] for i in range(len(matrix))]

return inverse\_matrix

# Example 3×3 Matrix

A = [[4, 7, 2],

[3, 6, 1],

[2, 5, 3]]

# Compute Inverse

inv\_A = inverse(A)

if inv\_A:

print("Inverse Matrix:")

for row in inv\_A:

print(row)

else:

print("Matrix is non-invertible.")

**Matrix Rank, Matrix Trace and Orthogonality of Matrices**

**🔹 1 Matrix Rank (Using Row Reduction)**

**Rank** is the number of **nonzero rows** in a matrix’s **row echelon form**.

python

CopyEdit

def matrix\_rank(matrix):

"""Find the rank of a matrix using row reduction (Gaussian elimination)."""

rows, cols = len(matrix), len(matrix[0])

rank = 0

for r in range(rows):

# Find the first non-zero pivot element

for c in range(cols):

if matrix[r][c] != 0:

rank += 1

# Make other rows 0 in this column

for i in range(r + 1, rows):

if matrix[i][c] != 0:

factor = matrix[i][c] / matrix[r][c]

matrix[i] = [matrix[i][j] - factor \* matrix[r][j] for j in range(cols)]

break

return rank

# Example

A = [[2, 4, 1],

[1, 2, 1],

[3, 6, 2]]

print("Rank of matrix:", matrix\_rank(A))

**🔹 2 Matrix Trace**

The **trace** of a square matrix is the sum of its diagonal elements.

python

CopyEdit

def matrix\_trace(matrix):

"""Compute the trace of a square matrix."""

return sum(matrix[i][i] for i in range(len(matrix)))

# Example

A = [[3, 2, -1],

[2, -2, 4],

[-1, 0.5, -1]]

print("Matrix Trace:", matrix\_trace(A))

**🔹 3 Check Orthogonality of a Matrix**

A matrix **A** is **orthogonal** if **A × Aᵀ = I** (Identity Matrix).

python

CopyEdit

def transpose(matrix):

"""Compute transpose of a matrix."""

return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

def matrix\_multiply(A, B):

"""Multiply two matrices."""

rows\_A, cols\_A = len(A), len(A[0])

rows\_B, cols\_B = len(B), len(B[0])

if cols\_A != rows\_B:

return None # Cannot multiply if dimensions don't match

result = [[sum(A[i][k] \* B[k][j] for k in range(cols\_A)) for j in range(cols\_B)] for i in range(rows\_A)]

return result

def is\_orthogonal(matrix):

"""Check if a square matrix is orthogonal (A \* Aᵀ = I)."""

size = len(matrix)

identity = [[1 if i == j else 0 for j in range(size)] for i in range(size)]

A\_T = transpose(matrix)

product = matrix\_multiply(matrix, A\_T)

return product == identity

# Example Orthogonal Matrix

A = [[1, 0],

[0, -1]] # This is an orthogonal matrix

print("Is Orthogonal:", is\_orthogonal(A))

**Dot Product of Two Matrices**

**1️⃣ Using Nested Loops (Traditional Method)**

python

CopyEdit

def dot\_product\_loops(A, B):

if len(A[0]) != len(B): # Check if multiplication is possible

raise ValueError("Number of columns in A must be equal to number of rows in B")

result = [[0] \* len(B[0]) for \_ in range(len(A))] # Initialize result matrix

for i in range(len(A)):

for j in range(len(B[0])):

for k in range(len(A[0])):

result[i][j] += A[i][k] \* B[k][j]

return result

# Example Matrices

A = [[1, 2, 3],

[4, 5, 6]]

B = [[7, 8],

[9, 10],

[11, 12]]

# Performing Dot Product

result = dot\_product\_loops(A, B)

print("Dot Product using Nested Loops:")

for row in result:

print(row)

✅ **Pros:** Simple and easy to understand

❌ **Cons:** Slower for large matrices

**2️⃣ Using List Comprehension**

python

CopyEdit

def dot\_product\_list\_comprehension(A, B):

if len(A[0]) != len(B):

raise ValueError("Number of columns in A must be equal to number of rows in B")

return [[sum(A[i][k] \* B[k][j] for k in range(len(A[0]))) for j in range(len(B[0]))] for i in range(len(A))]

# Performing Dot Product

result = dot\_product\_list\_comprehension(A, B)

print("Dot Product using List Comprehension:")

for row in result:

print(row)

✅ **Pros:** Short and Pythonic

❌ **Cons:** Harder to read for beginners

**3️⃣ Using Zip for Transposing B**

python

CopyEdit

def dot\_product\_zip(A, B):

if len(A[0]) != len(B):

raise ValueError("Number of columns in A must be equal to number of rows in B")

B\_T = list(zip(\*B)) # Transpose B using zip

return [[sum(a \* b for a, b in zip(row\_A, col\_B)) for col\_B in B\_T] for row\_A in A]

# Performing Dot Product

result = dot\_product\_zip(A, B)

print("Dot Product using Zip:")

for row in result:

print(row)

✅ **Pros:** Efficient, avoids extra loops by using zip()

❌ **Cons:** May be less intuitive for absolute beginners

**Finding eigenvalues and eigenvectors**

**🔹 Step-by-Step Code (2×2 Matrix)**

def determinant(matrix):

return matrix[0][0]\*matrix[1][1]-matrix[0][1]\*matrix[1][0]

def eigan\_value(matrix):

a,b=matrix[0][0],matrix[0][1]

c,d=matrix[1][0],matrix[1][1]

trace=a+d

det=determinant(matrix)

discriminant=(trace\*\*2)-(4\*det)

if discriminant==0:

return []

sqr\_descriminant=discriminant\*\*0.5

one=(trace+sqr\_descriminant)/2

two=(trace-sqr\_descriminant)/2

return [one,two]

def eigan\_vector(matrix,eigan\_values):

vector={}

for k in eigan\_values:

mod\_matrix=[[matrix[i][j] - (k if i==j else 0)for j in range(2)]for i in range(2)]

if mod\_matrix[0][0]!=0:

x = -mod\_matrix[0][1] / mod\_matrix[0][0]

vector[k]=[x,1]

else:

vector[k] = [1, -mod\_matrix[1][0] / mod\_matrix[1][1]]

return vector

A = [[4, -2],

[1, 1]]

va=eigan\_value(A)

print(eigan\_value(A))

print(eigan\_vector(A,va))

**🔹 Code for Any n×n Matrix**

python

CopyEdit

def determinant(matrix):

"""Compute determinant of a matrix using recursion (Laplace Expansion)."""

size = len(matrix)

if size == 1:

return matrix[0][0]

if size == 2:

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

det = 0

for col in range(size):

sub\_matrix = [row[:col] + row[col+1:] for row in matrix[1:]] # Remove first row & current column

det += ((-1) \*\* col) \* matrix[0][col] \* determinant(sub\_matrix)

return det

def find\_eigenvalues(matrix):

"""Find eigenvalues by solving det(A - λI) = 0."""

size = len(matrix)

# Try integer values of λ (brute-force approach)

lambdas = range(-10, 10) # Testing integer values from -10 to 10

eigenvalues = []

for λ in lambdas:

mod\_matrix = [[matrix[i][j] - (λ if i == j else 0) for j in range(size)] for i in range(size)]

if determinant(mod\_matrix) == 0:

eigenvalues.append(λ)

return list(set(eigenvalues)) # Remove duplicates

def solve\_linear\_system(matrix):

"""Solve (A - λI)v = 0 using basic row operations."""

size = len(matrix)

solutions = []

for col in range(size):

if matrix[col][col] != 0:

solutions.append(1) # Assume 1 for free variable

else:

solutions.append(0)

return solutions

def find\_eigenvectors(matrix, eigenvalues):

"""Find eigenvectors for each eigenvalue."""

size = len(matrix)

eigenvectors = {}

for λ in eigenvalues:

mod\_matrix = [[matrix[i][j] - (λ if i == j else 0) for j in range(size)] for i in range(size)]

eigenvectors[λ] = solve\_linear\_system(mod\_matrix)

return eigenvectors

# Example 3×3 Matrix

A = [[6, 2, 1],

[2, 3, 1],

[1, 1, 1]]

# Compute Eigenvalues & Eigenvectors

eigenvalues = find\_eigenvalues(A)

eigenvectors = find\_eigenvectors(A, eigenvalues)

print("Eigenvalues:", eigenvalues)

for λ, vec in eigenvectors.items():

print(f"Eigenvector for λ = {λ}: {vec}")