Task 1: Prepare Datasets & Document the Process

Objective: To collect and prepare multiple datasets while documenting the process.

Dataset Requirements & Collection Process:

1. **1000 Research Papers on Varied Niches**
    - Identified niche areas: AI, cybersecurity, climate change, finance, etc.
    - Used APIs such as arXiv, Semantic Scholar, and Google Scholar to scrape research papers.
    - Extracted metadata: title, abstract, authors, publication year, etc.
    - Converted data into structured formats (CSV, JSON, or plain text).
2. **5000 Study Material PDFs for Trending Skills**
    - Identified trending skills: Data Science, AI, Blockchain, UI/UX, etc.
    - Scraped resources from platforms like Coursera, Udemy, MIT OpenCourseWare, and GitHub.
    - Filtered high-quality PDFs and stored them with metadata.
3. **5000 Food Images Scraped from Social Media**
    - Used Instagram, Twitter, and Pinterest APIs to scrape images.
    - Focused on hashtags like #FoodPhotography and #HomeCooking.
    - Applied image quality filters to remove irrelevant images.
4. **1000 Whitepapers on Varied Niches**
    - Scraped from Forrester, Gartner, company reports, and GitHub repositories.
    - Extracted content and metadata and structured them properly.

Tools & Technologies Used:

- **Python Libraries:** requests, BeautifulSoup, Selenium, Tweepy, PyPDF2, pdfplumber
- **APIs Used:** arXiv API, Google Scholar API, Instagram API, Twitter API
- **Data Storage:** CSV, JSON, MongoDB

Deliverables:

- Prepared datasets (CSV/JSON)
- Documentation of the data collection process

Task 5: Introspect & Improve the PDF Parser Code

Objective: To analyze the given PDF parser code and suggest improvements.

Steps for Analysis:

1. **Review Code Functionality:**
    - Checked how the parser extracts text (line-by-line, structured tables, metadata).
    - Verified handling of different PDF structures (scanned vs. digital PDFs).
    - Looked for parsing errors (encoding issues, missing data, etc.).
2. **Identified Potential Issues:**
    - **Lack of OCR Support:** If it doesn't handle scanned PDFs, added Tesseract OCR.
    - **Poor Handling of Multi-Column PDFs:** Used pdfplumber for better extraction.
    - **Inefficient Processing:** Checked performance on large PDFs.
3. **Suggested Improvements:**
    - **Improve Accuracy:** Implemented PyMuPDF for better text extraction.
    - **Enhance OCR Capabilities:** Integrated Tesseract-OCR for scanned documents.
    - **Optimize Performance:** Implemented parallel processing for handling large PDFs.
    - **Error Handling:** Improved exception handling for incomplete/malformed PDFs.

Tools & Technologies Used:

- **Python Libraries:** PyPDF2, pdfplumber, PyMuPDF, Tesseract-OCR

Deliverables:

- Identified issues in the PDF parser
- List of improvements & code enhancements

Task 9: Optimal Delivery Route System

Objective: To design a logic that assigns optimal routes to delivery vehicles while allowing real-time order allocation.

Steps to Build the System:

1. **Define Input Data:**
   - **Orders:** Pickup & drop-off locations, package weight, priority.
   - **Vehicles:** Capacity, location, speed, fuel efficiency.
   - **Traffic Conditions:** Real-time updates from Google Maps API.
2. **Route Optimization Algorithm:**
   - Used **Dijkstra’s Algorithm / A Algorithm*** for the shortest path.
   - Implemented **Dynamic Route Updates** when new orders arrive.
   - Minimized costs (time, fuel, vehicle capacity usage).
3. **Real-Time Order Assignment:**
   - Checked existing routes of vehicles.
   - If a new order matched an existing route, assigned it directly.
   - If not, dynamically adjusted the route.
4. **Technology Stack:**
   - **Google Maps API / OpenStreetMap** for routing.
   - **Python (Flask/Django)** for backend development.
   - **Machine Learning** for ETA prediction.

Deliverables:

- Route Optimization Logic
- Algorithm for dynamic order assignment
- API-based real-time routing