**EPIC 1: Database Foundations (MySQL & MongoDB)**  
 Description:  
 Design and build the foundational databases for storing orders, suppliers, inventory, and shipment logs to support all supply chain operations.

**User Story 1.1: Create MySQL Schema for Orders, Suppliers, and Inventory**  
 As a data engineer, I want to design and implement MySQL tables for orders, suppliers, and inventory so that structured data can be efficiently stored and queried.

Acceptance Criteria:

* Tables created with proper relationships and keys.
* CRUD operations tested successfully.

Tasks:

* Create orders, suppliers, and inventory tables in MySQL.
* Define foreign key relationships.
* Insert and validate sample data.
* Test SELECT, INSERT, UPDATE, DELETE operations.

**User Story 1.2: Implement Stored Procedure for Auto Reorder**  
 As a backend developer, I want to create a stored procedure to auto-reorder items when inventory is below threshold so that restocking is automated.

Acceptance Criteria:

* Procedure executes on condition trigger.
* Low stock items trigger reorder log entry.

Tasks:

* Write SQL stored procedure auto\_reorder().
* Test the trigger logic.
* Document reorder thresholds and parameters.

**User Story 1.3: Store Shipment Logs in MongoDB**  
 As a developer, I want to store shipment logs in MongoDB collections so that real-time and semi-structured data can be tracked efficiently.

Acceptance Criteria:

* Shipment logs inserted successfully.
* Index created for faster query performance.

Tasks:

* Create MongoDB collection for shipment logs.
* Insert sample shipment data.
* Create index on shipment date and supplier ID.

**EPIC 2: Data Collection and Preprocessing in Python**  
 Description:  
 Collect and preprocess supply chain data from APIs or files to prepare it for analytics and storage.

**User Story 2.1: Fetch Data from API or Local Files**  
 As a developer, I want to fetch supply chain data using Python requests or CSV input so that the latest order and shipment information can be processed.

Acceptance Criteria:

* Data fetched without errors.
* Valid JSON or CSV structure.

Tasks:

* Write Python script using requests.get() or pd.read\_csv().
* Validate fetched data structure.
* Save raw data to local storage.

**User Story 2.2: Clean and Preprocess Data**  
 As a data analyst, I want to clean and format the collected data so that it can be used reliably for downstream analysis.

Acceptance Criteria:

* Nulls dropped, dates formatted, duplicates removed.
* Cleaned data verified for consistency.

Tasks:

* Drop missing or null values.
* Format delivery\_date as datetime.
* Normalize supplier names and order IDs.
* Save cleaned DataFrame to CSV.

**User Story 2.3: Compute Metrics Using NumPy**  
 As a data analyst, I want to calculate delivery delays and stock levels so that I can generate insights into supplier performance.

Acceptance Criteria:

* Delay days calculated accurately.
* New columns added for delay flags.

Tasks:

* Compute delay\_days using NumPy.
* Add is\_delayed column.
* Print summary of delayed vs on-time deliveries.

**EPIC 3: Process Big Data with PySpark**  
 Description:  
 Use PySpark for scalable supply chain data processing to handle large datasets efficiently.

**User Story 3.1: Load Data into PySpark**  
 As a data engineer, I want to load CSV data into PySpark DataFrames so that it can be processed in a distributed environment.

Acceptance Criteria:

* Data loaded successfully from CSV.
* Schema inferred correctly.

Tasks:

* Initialize Spark session.
* Load orders data from CSV.
* Display first few records for verification.

**User Story 3.2: Filter and Group Data**  
 As a data analyst, I want to filter delayed shipments and group them by supplier so that I can identify performance issues.

Acceptance Criteria:

* Delayed shipments correctly filtered.
* Grouped summary output verified.

Tasks:

* Apply filter where is\_delayed = 1.
* Group by supplier\_id and count delayed orders.
* Save results to DataFrame.

**User Story 3.3: Save Processed Data**  
 As a developer, I want to save processed data as CSV or Parquet so that it can be reused for reporting.

Acceptance Criteria:

* File saved without errors.
* Output verified in target directory.

Tasks:

* Write DataFrame to output/supplier\_summary.csv.
* Validate saved file size and content.

**EPIC 4: Simple ETL in Azure Databricks**  
 Description:  
 Build a Databricks notebook to perform ETL (Extract, Transform, Load) operations and store clean data for analytics.

**User Story 4.1: Upload CSVs to Databricks Workspace**  
 As a data engineer, I want to upload source data to Databricks FileStore so that it can be processed within notebooks.

Acceptance Criteria:

* CSVs uploaded and accessible via /dbfs/FileStore/.

Tasks:

* Use Databricks file upload UI or CLI.
* Verify file path accessibility using %fs ls.

**User Story 4.2: Transform Data in Databricks**  
 As a developer, I want to clean and filter order data using PySpark in Databricks so that I can generate insights-ready datasets.

Acceptance Criteria:

* Data cleaned, filtered, and saved.

Tasks:

* Load CSV into DataFrame.
* Remove null and delayed shipments.
* Save as Delta table or CSV.

**User Story 4.3: Run SQL Queries for Validation**  
 As a data analyst, I want to execute SQL queries on transformed data so that I can validate data quality and business metrics.

Acceptance Criteria:

* Queries run successfully.
* Results match expected counts.

Tasks:

* Create temporary view.
* Run SELECT queries.
* Export results to CSV.

**EPIC 5: CI/CD Automation in Azure DevOps**  
 Description:  
 Automate execution of the data processing scripts using Azure DevOps pipelines for continuous integration and delivery.

**User Story 5.1: Set Up Azure DevOps Pipeline**  
 As a DevOps engineer, I want to configure a pipeline that executes Python scripts automatically so that I can ensure consistency in builds and deployments.

Acceptance Criteria:

* Pipeline triggers on every commit.
* Job runs successfully in DevOps console.

Tasks:

* Create azure-pipelines.yml.
* Set up ubuntu-latest pool.
* Define steps to install dependencies and run run\_pipeline.py.

**User Story 5.2: Install Dependencies Automatically**  
 As a developer, I want the pipeline to install packages from requirements.txt so that it can run smoothly across environments.

Acceptance Criteria:

* All dependencies installed without errors.

Tasks:

* Add pip install -r requirements.txt to YAML.
* Validate install logs in pipeline output.

**User Story 5.3: Monitor Pipeline Logs and Results**  
 As a project manager, I want to track the execution and outcome of each pipeline run so that I can monitor deployment health.

Acceptance Criteria:

* Console logs visible for each run.
* Success or failure status recorded.

Tasks:

* Check pipeline run logs.
* Export or screenshot execution output.

**User Story 5.4: Notify Team on Completion**  
 As a team lead, I want to receive email notifications on pipeline completion so that I’m informed about run outcomes automatically.

Acceptance Criteria:

* Email or Teams notification received.

Tasks:

* Configure DevOps notifications in Project Settings.
* Verify test email upon successful run.

**Screenshots:**

