**Project Title:** Intelligent Inventory Demand Forecasting and Reorder Alert System

**Duration:** 1 Month

**Project Overview**

This project aims to build a software-based intelligent inventory management solution that forecasts future product demand and automatically generates reorder alerts. Without hardware integration, the system will rely on historical sales and inventory datasets along with predefined lead times and supplier data stored in a relational database. The outcome will be a prediction model and a dashboard application that can assist retailers in making data-driven stocking decisions.

By working on this project, you will utilize your machine learning, programming, and database skills, and gain hands-on experience with modern data science and web frameworks.

**Disclaimer:** Use of any AI tool (including chatbots, code generators, or AI-assisted development environments) to complete this project is strictly prohibited. Any evidence of such usage will result in the immediate cancellation of the internship.

**Key Technical Objectives**

1. **Data Management & Preprocessing:**  
   * **Data Storage:**
     + Use **MySQL** or **PostgreSQL** for relational data storage. The intern will design database schemas and establish connections from the Python application.
   * **Data Import & Cleaning:**
     + Data (e.g., historical daily sales, SKU details, supplier lead times) will be provided in CSV format.
     + Use **Pandas** in Python to load, clean, and preprocess data (handling missing values, formatting date/time fields, and performing aggregations).
   * **Schema Design:**
     + Tables may include(this is an example you can also add other info):
       - **products**
       - **sales\_history**
       - **inventory\_levels**
       - **suppliers**
     + Connect these tables logically to facilitate efficient querying and model input preparation.
2. **Demand Forecasting Model:**
   * **Modelling Approaches:**
     + Time-series forecasting using libraries like **Facebook Prophet**, **stats models** (ARIMA, SARIMAX), or **scikit-learn** (if reframed as a regression problem with lag features).
   * **Feature Engineering:**
     + Create lag features (e.g., last week’s sales, last month’s sales), moving averages, and seasonal indicators (day of week, month) to improve model accuracy.
   * **Model Training & Validation:**
     + Split historical data into training and validation sets based on time (e.g., first 80% of data for training, last 20% for validation).
     + Use **Mean Absolute Error (MAE)** or **Mean Absolute Percentage Error (MAPE)** as the primary metric to evaluate forecast quality.
   * **Model Deployment Steps:**
     + Save the trained model using **joblib** or **pickle** for future inference.
     + Implement a scheduled script (Python cron job or Airflow DAG on local machine) to run forecasts daily/weekly and store predictions back into the database.
3. **Threshold & Automated Alerts:**
   * **Threshold Calculation:**
     + Dynamically calculate reorder thresholds based on forecasted demand and supplier lead times.
     + For example: safety\_stock = avg\_daily\_demand \* lead\_time \* safety\_factor, where safety\_factor might be a configurable parameter.
   * **Automated Alerts Generation:**
     + Write a Python script that queries current inventory levels and forecasted demand and determines if a reorder is needed.
     + Alerts could be stored in a alerts table within the database and displayed on the dashboard.
   * **Future Consideration:**
     + Implement logic to send email alerts using **SMTP** or integrate with messaging services (Slack, MS Teams) if time permits.
4. **Dashboard & Visualization:**
   * **Framework Selection:**
     + Use **Streamlit** for rapid dashboard development, or **Flask/FastAPI** + **Plotly/Dash** for a more customizable setup.
   * **Key Dashboard Components:**
     + **Sales History Visualization:** Interactive time-series plots (line charts) of historical sales per product using **Plotly**.
     + **Forecast Display:** Show next 2-4 weeks of predicted demand.
     + **Inventory Levels & Reorder Alerts:** Real-time summary table highlighting products that need restocking.
     + **Parameter Controls:** Simple sliders/inputs to adjust forecast horizon, safety factors, or select specific products.
   * **Backend Integration:**
     + Connect the dashboard to the database using SQLAlchemy or a similar ORM to fetch results dynamically.
5. **Documentation & Code Quality:**
   * **Code Organization:**
     + Create a clear directory structure:
       - data/ for input CSVs
       - src/ for Python scripts (EDA, preprocessing, model training, forecasting, alert scripts)
       - app/ for dashboard source code
       - docs/ for documentation
   * **Version Control & Collaboration:**
     + Use **Git** for version control.
     + Maintain a clear README with instructions on setup, database migrations, and running each component.
   * **Technical Documentation:**
     + Include a short architecture diagram (e.g., using Mermaid in markdown) showing data flow from database to model to dashboard.
   * **Final Presentation:**
     + A slide deck covering data exploration, model selection, performance metrics, system architecture, and a live demonstration of the dashboard.

**Proposed Timeline**

**Week 1: Data Infrastructure & Initial EDA**

* **Days 1-2:** Requirements finalization, environment setup (Python virtual environment, database initialization), and data schema design.
* **Days 3-5:** Data loading into the database, initial EDA with Pandas and Matplotlib/Seaborn, cleaning data, basic time-series plots.

**Week 2: Model Development & Validation**

* **Days 1-2:** Implement baseline forecasting models (Prophet/ARIMA), tune parameters.
* **Days 3-5:** Evaluate models against validation sets, select the best approach, and finalize a forecasting pipeline. Store trained model artifacts.

**Week 3: Integration & Dashboard Creation**

* **Days 1-2:** Implement logic for reorder threshold calculations and store alerts in the database.
* **Days 3-5:** Develop a Streamlit/Flask+Plotly-based dashboard. Integrate live data from DB (predictions, current inventory, alerts) into the front end.

**Week 4: Testing, Refinement & Documentation**

* **Days 1-3:** Test the entire pipeline end-to-end (data → model → alerts → dashboard). Optimize any slow queries or visualization performance issues.
* **Days 4-5:** Prepare final documentation, a short demo video or presentation slides, and ensure code is well-commented and versioned.

**Deliverables**

1. **Functional Codebase:**
   * Scripts for data ingestion, preprocessing, model training/forecasting, and generating alerts.
   * A fully operational dashboard application connected to the database and the predictive model.
2. **Database Schema & Data Management:**
   * SQL scripts or migrations defining the products, sales\_history, inventory\_levels, suppliers, and alerts tables.
   * Clear instructions on how to populate the database and run daily forecasts.
3. **Machine Learning Model Artifacts:**
   * A trained time-series forecasting model artifact (e.g., Prophet model saved using joblib).
   * Model evaluation reports (Jupyter notebooks or markdown reports) showing error metrics and plots.
4. **Dashboard Application:**
   * A user-friendly web interface to visualize historical sales, future forecasts, and identify products needing reorder.
5. **Documentation & Presentations:**
   * A README outlining setup instructions, dependencies (requirements.txt), and usage.
   * A system architecture document (in docs/) describing the data flow and technical stack.
   * A short presentation (PDF or PowerPoint) summarizing key insights, methodologies, and demonstration steps.