**Project: Predicting Product Demand for Supply Chain Optimization using Generative AI**

**Problem Statement**

In the retail and manufacturing industries, efficient inventory and supply chain management are crucial to reducing costs and meeting customer demand. Misestimating demand can lead to overstocking (increased holding costs) or stockouts (lost sales and customer dissatisfaction). Traditional demand forecasting methods struggle with:

* Variability in demand patterns due to seasonality, promotions, and market changes.
* High-dimensional, time-series data from multiple sources (e.g., sales data, market data, holidays, and promotions).

**Objective:** Develop an advanced Generative AI model for demand forecasting. The model will synthesize demand scenarios based on historical data and other relevant variables, providing more robust predictions under uncertain and dynamic conditions.

**Metrics for Success:**

1. **Forecast Accuracy**: Measured by Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE).
2. **Inventory Optimization**: Reduction in overstocking and stockouts by a target percentage.
3. **Cost Savings**: Reduction in holding and stockout costs, validated by historical baseline comparisons.

**Solution Outline**

This solution uses a Conditional Generative Adversarial Network (cGAN) to generate realistic demand scenarios and combines this with a predictive model to forecast future demand accurately.

**Project Breakdown**

**Step 1: Data Collection and Preprocessing**

1. **Data Sources**:
   * Historical sales data (daily or weekly data).
   * Product metadata (e.g., product category, seasonality, and price).
   * External factors (e.g., holidays, weather, promotions).
2. **Data Cleaning**:
   * Handle missing values and outliers in demand and price data.
   * Normalize or standardize numerical data.
   * Convert categorical data (e.g., product categories) to embeddings.
3. **Feature Engineering**:
   * Create time-based features (e.g., day of the week, month).
   * Use lagged demand values to capture seasonality.
   * Aggregate demand at different granularities to capture patterns.
4. **Train-Test Split**:
   * Split data into training and testing sets, ensuring chronological separation to prevent data leakage.

**Step 2: Model Development**

1. **Demand Generation Model** (cGAN):
   * **Generator**: Takes in conditions (e.g., product ID, week) and noise to generate synthetic demand values.
   * **Discriminator**: Differentiates between real and synthetic demand, learning complex patterns in the data.
   * **Conditioning Variables**: Include product details, time features, and external factors.
   * **Training**: Train the cGAN on historical data to create synthetic scenarios for rare events (e.g., high demand due to holidays).
2. **Forecasting Model** (Hybrid CNN-LSTM):
   * Use the synthetic demand scenarios generated by the cGAN as additional training samples to improve the robustness of the forecasting model.
   * **CNN**: Extract spatial features from past demand trends.
   * **LSTM**: Capture temporal dependencies and predict future demand.
   * **Output Layer**: Dense layer with ReLU activation for demand output.
3. **Evaluation Metrics**:
   * Track MAPE and RMSE on validation data.
   * Compare forecast accuracy against a baseline (e.g., traditional time-series forecasting).

**Step 3: Training and Validation**

1. **Training**:
   * Train the cGAN model separately until it can generate realistic demand distributions.
   * Train the CNN-LSTM model on both real and generated demand scenarios.
2. **Validation**:
   * Use a rolling forecast method for validation, retraining on the most recent data periodically.
   * Evaluate MAPE and RMSE on validation data.
3. **Hyperparameter Tuning**:
   * Optimize key parameters (e.g., number of layers, units in LSTM) to minimize RMSE on the validation set.

**Step 4: Model Deployment**

1. **API Development**:
   * Deploy the trained model as a RESTful API using Flask or FastAPI.
   * Integrate endpoints for batch and real-time demand prediction.
2. **Automated Pipelines**:
   * Set up a pipeline with Airflow for regular retraining using new data.
   * Deploy on AWS or Google Cloud for scalability, with automated scaling and monitoring.
3. **Monitoring and Logging**:
   * Set up logging to track forecast accuracy and model drift.
   * Use monitoring tools like Prometheus and Grafana to ensure API reliability.