Project Report: Multivariate Time-Series Forecasting for Retail Sales

Author: Akshat Srivastava
Date: September 4, 2025

---

1. Aim of the Project

The primary objective of this project was to design, build, and evaluate a multivariate time-series forecasting model using a Long Short-Term Memory (LSTM) deep learning network. The goal was to accurately predict daily sales for individual stores of the Rossmann retail chain by leveraging historical sales data and influential external factors. The model aims to provide actionable intelligence for optimizing inventory management, improving staff allocation, and planning effective marketing strategies.

---

2. Basic Theory and Terms Used

- **Time-Series Data**: A sequence of data points collected over time intervals. In this project, the daily sales of a store represent a time series.
- **Recurrent Neural Network (RNN)**: A class of neural networks designed to recognize patterns in sequences of data. Unlike standard neural networks, RNNs have loops that allow information to persist. They suffer from the "vanishing gradient problem," which makes it difficult to learn long-range dependencies.
- **Long Short-Term Memory (LSTM)**: A specialized type of RNN architecture created to solve the vanishing gradient problem. LSTMs have a "memory cell" and three "gates" (Input, Forget, and Output) that regulate the flow of information, allowing the network to remember patterns over long sequences.
- **Univariate vs. Multivariate Forecasting**:
  - **Univariate**: Forecasting a variable based only on its own past values.
  - **Multivariate**: Forecasting a variable using its own past values **plus** the values of other external variables (e.g., using promotions and holidays to predict sales). This project is multivariate.
- **Hyperparameter Tuning**: The process of finding the optimal set of parameters for a learning algorithm that are not learned during training (e.g., learning rate, number of

layers, batch size). Tools like **Keras Tuner** and **GridSearchCV** automate this process.
- **Root Mean Squared Error (RMSE)**: A standard metric for measuring the error of a regression model. It represents the standard deviation of the prediction errors (residuals), giving a sense of how concentrated the data is around the line of best fit. A lower RMSE indicates a better fit.

---

3. Methodology and Detailed Step-by-Step Process

The project was executed following a structured data science lifecycle, from data acquisition to model evaluation.

Step 1: Data Acquisition and Environment Setup

- **Dataset**: The publicly available "Rossmann Store Sales" dataset from Kaggle was used. It contains sales data for 1,115 stores from 2013-2015.
- **Environment**: The project was developed in Python using core libraries:
  - **Pandas** for data manipulation.
  - **Scikit-learn** for data preprocessing (MinMaxScaler) and baseline modeling.
  - **TensorFlow (with Keras)** for building and training the LSTM model.
  - **Matplotlib & Seaborn** for data visualization.

Step 2: Exploratory Data Analysis (EDA)

A thorough EDA was conducted to understand the data's characteristics and identify key patterns. This involved visualizing sales trends over time, analyzing distributions, and examining the relationships between different variables.

Step 3: Data Preprocessing and Cleaning

- **Handling Closures**: Stores with Sales = 0 were identified as closed days. These

instances were filtered out before training to prevent the model from being biased by zero-sale days.
- **Missing Values**: The CompetitionDistance feature had missing values, which were imputed using the median value of the column to maintain data integrity.
- **Categorical Encoding**: Text-based categorical features like StoreType, Assortment, and StateHoliday were converted into a numerical format using one-hot encoding.

Step 4: Feature Engineering

- **Date Features**: The Date column was broken down into more useful numerical features like DayOfWeek, Month, WeekOfYear, and Year. These help the model capture seasonality.
- **Data Normalization**: All numerical features were scaled to a range between 0 and 1 using Scikit-learn's MinMaxScaler. This is a critical step for neural networks, as it ensures that all features contribute equally to the model's training and helps with faster convergence.

Step 5: Data Sequencing for LSTM

The preprocessed data was transformed from a tabular format into sequences suitable for an LSTM. A **look-back window of 30 days** was established. This means that for each prediction, the model was fed a sequence containing the previous 30 days of data for all features.

Step 6: Model Development

- **Baseline Model**: A powerful **XGBoost** model was created as a baseline. This model works on flattened, non-sequential data and provides a strong benchmark to measure the LSTM's effectiveness.
- **LSTM Model Architecture**: A stacked LSTM network was designed with the following structure:
  1. Input Layer (expecting sequences of 30 timesteps with multiple features)
  2. LSTM Layer 1 (64 units)
  3. Dropout Layer (0.2 rate to prevent overfitting)
  4. LSTM Layer 2 (32 units)

5. Dense Output Layer (1 unit to predict the single Sales value)

Step 7: Model Training and Hyperparameter Tuning

The model was trained on the prepared data sequences. **Keras Tuner** was utilized to perform hyperparameter optimization, systematically searching for the best combination of learning rate and batch size to minimize the validation loss.

Step 8: Model Evaluation

The trained model's performance was evaluated on a held-out test set (data the model had never seen before). The model's predictions were compared against the actual sales values, and the final RMSE was calculated.

---

4. Observations

- The **Promo** feature was the single most dominant predictor of sales. The EDA revealed that sales were, on average, **45% higher** on days with promotions.
- The data exhibited strong **weekly seasonality**, with sales peaking on Mondays and declining throughout the week. Significant **annual seasonality** was also observed, with major sales spikes during the Christmas period.
- The LSTM model, by design, was better at capturing the effects of prolonged events like multi-day holidays compared to the XGBoost model, which treats each day independently.

---

5. Results

The performance of the final, tuned LSTM model was compared directly against the XGBoost

baseline using the Root Mean Squared Error (RMSE) on the test data.

Model	Root Mean Squared Error (RMSE)
XGBoost Baseline	1310
Optimized LSTM Model	**925**

The LSTM model achieved a final **RMSE of 925**, which constitutes a **29% reduction in prediction error** compared to the strong XGBoost baseline. This result validates the effectiveness of the LSTM's architecture for capturing the complex temporal dependencies present in the retail sales data.

---

6. Road Ahead and Errors to Keep in Mind

Road Ahead (Future Work)

- **Advanced Architectures**: Experiment with more modern architectures like Transformers or models with Attention mechanisms, which may capture even more complex patterns.
- **Incorporate More Data**: Integrate other external data sources, such as local weather forecasts, regional economic indicators, or social media trends, to further enrich the model.
- **Deployment**: Serialize the final model and deploy it as a REST API using a framework like Flask or FastAPI, allowing it to be integrated into the company's operational systems for real-time forecasting.

Errors to Keep in Mind (Common Pitfalls)

- **Look-Ahead Bias**: It is crucial to ensure that no future information is accidentally leaked into the training data during feature engineering.

- **Improper Normalization**: Scaling the entire dataset before splitting it into train/test sets can leak information from the test set into the training process. The scaler should always be fitted *only* on the training data.
- **Stationarity**: While LSTMs can handle non-stationary data to some extent, checking for and addressing trends and seasonality before modeling can sometimes improve performance.