**Project Write-Up: *Sales Forecasting – Improving Accuracy with Integrated Data and Machine Learning***

**Overview and Motivation**

This project focused on developing a robust retail sales forecasting system that integrates multiple internal and external data sources such as transactions, holidays, oil prices, and store information to improve prediction accuracy and support data-driven business decisions. My goal was to move beyond simple time-series forecasting and build a machine-learning pipeline capable of learning complex seasonal and contextual patterns influencing sales. The project was implemented across seven structured weeks, progressing from data setup to feature engineering, model tuning, and final evaluation.

Sales Forecasting – Improving Accuracy with Integrated Data and Machine Learning

This project focuses on enhancing retail sales forecasting accuracy by integrating internal and external data sources and applying advanced machine learning models. The pipeline automates data ingestion, transformation, feature engineering, and model training for both traditional and deep learning approaches.

**KEY FEATURES**

• Automated ETL pipeline with SQLite database integration  
• Feature engineering for lag, rolling, and seasonal trends  
• Advanced machine learning models and deep learning (LSTM)  
• Hyperparameter tuning using Optuna  
• Model evaluation and comparison via RMSE metrics  
• Visualization and explainability using SHAP and Prophet

**PROJECT STRUCTURE**

sales\_forecasting/  
│  
├── data/  
│ ├── raw/ Raw CSVs (not included in repo)  
│ ├── processed/ Cleaned & feature-engineered datasets  
│  
├── models/ Trained model artifacts  
├── reports/ Outputs and visualizations  
├── notebooks/ Jupyter notebooks (week-wise progress)  
├── src/ Python modules (week1–week7)  
│ ├── utils.py  
│ ├── week1\_setup.py  
│ ├── week2\_cleaning.py  
│ ├── week3\_features.py  
│ ├── week4\_baseline.py  
│ ├── week5\_models.py  
│ ├── week6\_tuning.py  
│ └── week7\_final.py  
│  
├── main.py Orchestrates the weekly workflow  
├── requirements.txt List of dependencies  
├── .gitignore  
└── README.txt This document

**DATASET**

This project uses the "Store Sales – Time Series Forecasting" dataset from Kaggle.  
Because of size restrictions, the dataset is not included in this repository.  
You can download it from:  
<https://www.kaggle.com/competitions/store-sales-time-series-forecasting/data>

If you prefer a pre-cleaned version, you can download it from Google Drive:  
<https://drive.google.com/file/d/1dWMelQK5sJEislX363Dzlcv5Ek13QbBA/view?usp=drive_link>

**SETUP INSTRUCTIONS**

1. Clone the repository  
   git clone <https://github.com/your-username/sales_forecasting_final.git>  
   cd sales\_forecasting\_final
2. Create and activate a virtual environment  
   python -m venv venv  
   venv\Scripts\activate (Windows)  
   source venv/bin/activate (Mac/Linux)
3. Install dependencies  
   pip install -r requirements.txt

**HOW TO RUN**

Option 1: Run full pipeline - python main.py

Option 2: Run by week  
python -m src.week1\_setup  
python -m src.week2\_cleaning  
python -m src.week3\_features  
python -m src.week4\_baseline  
python -m src.week5\_models  
python -m src.week6\_tuning  
python -m src.week7\_final

Option 3: Run interactively in Jupyter Notebook  
Open notebooks/Sales\_Forecasting.ipynb and execute week-by-week cells.

**Week-by-Week Process and Reasoning**

**Week 1 – Setup and Planning**  
I began by creating a structured project environment with organized directories for raw data, processed outputs, models, reports, and notebooks. This step ensured smooth workflow and reproducibility. I also initialized a local SQLite database to hold the integrated datasets. At this stage, I outlined the overall process in a GitHub repository so that version control and documentation stayed consistent throughout development.

**Week 2 – ETL and Data Cleaning**  
This week was devoted to extracting, transforming, and loading the raw Kaggle datasets into the database. I encountered initial challenges with inconsistent date formats and missing values—especially in oil prices and holiday transfers. I overcame this by interpolating continuous series (like oil) and filtering irrelevant or duplicated records. The ETL logic was built entirely in SQL, which gave me both efficiency and transparency for data validation.

**Week 3 – Feature Engineering**  
Once the base tables were clean, I aggregated them to monthly granularity and introduced engineered features such as lag variables, rolling averages, and seasonal indicators. I realized that without these features, most models performed poorly on validation data. I also merged macro-economic and transactional signals to better capture real-world patterns. One debugging challenge here was handling missing rolling windows after grouping; I resolved it using group-wise shift and rolling logic in pandas.

**Week 4 – Baseline Models and EDA**  
With features ready, I explored the data visually to confirm seasonality and overall growth trends. Baseline forecasting was performed using a simple 12-month lag, ARIMA, and Prophet. Installing Prophet initially caused dependency conflicts with NumPy 2.0, which I fixed by adjusting the environment version. This week helped establish baseline performance for later model comparisons.

**Week 5 – Advanced Models**  
Here I implemented machine-learning-based regressors such as Random Forest, XGBoost, and an LSTM network for sequence learning. The Random Forest served as a strong non-linear benchmark, while XGBoost handled multivariate relationships effectively. I learned the importance of scaling input data for deep-learning models and monitored overfitting through training history. This week was also where I began saving trained models and metrics for later analysis.

**Week 6 – Hyperparameter Tuning**  
To refine the results, I integrated Optuna for automated hyperparameter optimization of Random Forest and XGBoost. Managing database-backed study storage ensured reproducibility of experiments. I realized how tuning parameters like learning rate and tree depth significantly improved RMSE performance. This week taught me how to balance model complexity with generalization.

**Week 7 – Final Evaluation and Ensembling**  
The final week focused on comparing all models, visualizing their results, and building an ensemble that averaged predictions from Random Forest, XGBoost, and LSTM. This approach consistently reduced validation error compared to individual models. I also generated SHAP value visualizations to interpret feature impact and created Prophet forecasts for longer-term projections. Additional plots—leaderboards, actual-vs-predicted comparisons, and correlation heatmaps—helped communicate model performance effectively.

**Difficulties and How I Overcame Them**

Several technical and structural roadblocks arose throughout the project. Prophet’s incompatibility with the latest NumPy version caused repeated import failures; I resolved this by pinning dependencies. Handling large datasets in pandas occasionally led to memory issues, so I adopted chunked SQL loading. Debugging indentation and module imports across weekly scripts was another challenge, which I solved by standardizing a single run() function structure for all modules and managing shared utilities through a utils.py file. These issues reinforced the importance of consistent coding standards and modular design.

**What I Learned**

This project deepened my understanding of both the theory and practice of time-series forecasting. I learned how to build complete data pipelines, from raw ingestion to model deployment, and how to integrate statistical and machine-learning methods in one cohesive workflow. The experience also strengthened my skills in feature engineering, hyperparameter tuning, model interpretation, and visualization. More broadly, I gained confidence in diagnosing errors, version-controlling iterative experiments, and maintaining clear, reproducible code.

**Reflection and Takeaways**

The most valuable lesson was recognizing how data quality and structure drive model performance. Adding engineered features and aligning temporal relationships improved results far more than tuning algorithms alone. The modular weekly structure mirrored a real-world data-science workflow—beginning with messy data and ending with a usable, validated product. This project combined creativity, technical rigor, and persistence, and it gave me a complete perspective on developing data-driven forecasting systems from scratch.