Pre-reqs & Setup

- *Create AWS account / student AWS credits (if available).*
- *Install local tools: Git, Python 3.9+, AWS CLI, Docker (optional).*
- *Create a GitHub repo and a project README (include architecture diagram and team members).*

Data & Schema

- *Obtain a sample sales CSV (can be synthetic): columns: date, sku, store_id, units_sold, price, promo_flag, region.*
- *Create a small external file for holidays/festivals and a simple weather CSV (date,location,temperature,precipitation).*
- *Validate data types and fill/flag missing values. Create data/ folder in repo.*

AWS Resource

- *Create an S3 bucket for raw and processed data: s3://<team>-forecasting-raw, s3://<team>-forecasting-processed.*
- *Create minimal IAM user/role for the project with S3, Lambda, SageMaker, and CloudWatch permissions.*

Data Ingestion (Upload)

- *Upload CSVs to S3 raw zone. Use CLI or console.*
- *Create a simple AWS Lambda + API Gateway endpoint to accept file uploads.*

Lightweight ETL / Preprocessing

- *Implement a Python script or Glue job that:*
    - *Reads raw CSV from S3*
    - *Cleans data (type casting, remove duplicates, fill or flag missing)*
    - *Generates basic features: day_of_week, is_holiday, lag_7, rolling_14_mean, price_change.*
    - *Writes processed CSV to S3 processed zone.*

- *Save the script in etl/ and include a requirements.txt.*

Simple Baseline Model

Option A — Local:

- *Build a simple baseline model in Python:*
    - *Aggregation to daily SKU-store level.*
    - *Train/test split using time-based split (e.g., last 20% as test).*

    - *Use a simple model: Prophet, or XGBoost on engineered features, or even an ARIMA baseline.*
    - *Save predictions as CSV and compute MAPE/MAE.*

Option B — SageMaker:

- *Containerize/training-job using built-in XGBoost or a simple script.*

Use Amazon Forecast

- *Prepare data in Forecast-friendly schema (timestamp, item_id, target_value, forecast_dimensions).*
- *Create dataset group, import data, train a predictor, and export forecast CSV to S3.*

Serving / API

- *Create a simple API to serve forecasts:*
    - *Option: API Gateway + Lambda (Python) function that reads forecast CSV from S3 and returns forecasts for a given sku and date_range.*
    - *Local alternative: Flask app that reads CSV and serves endpoints.*

Dashboard / Frontend

- *Build a minimal dashboard (one-page) to:*
    - *Upload CSVs (optional)*
    - *Select SKU & date range*
    - *Plot historical vs forecast (use Plotly/Chart.js)*
    - *Show simple metric summary (MAPE, MAE)*

- *Hosting options: static React app + fetch API, or simple notebook dashboard (Voila) for local demos.*

Evaluation & Backtesting

- *Implement a time-based backtest: rolling-window or holdout.*
- *Compute MAPE, MAE, and a simple bias metric (mean(pred-actual)/mean(actual)).*
- *Create a short report (Markdown) summarizing results and lessons.*

Logging, Monitoring & Basic Retraining Trigger

- *Add simple logging (CloudWatch for Lambda or file logs for local).*
- *Implement a simple retrain trigger:*
    - *Manual: run train.py script*

    - *Automated : Step Function or a cron-like EventBridge to trigger training job*

Extensions

- *Add Amazon Forecast or compare Forecast vs your model*.
- *Add more external features (weather impact test, festival uplift detection)*.
- *Implement feature importance (SHAP) and add explanation charts*.
- Add simple anomaly detection for sudden demand spikes.