Week 2 & 3 – Feature Engineering & Modelle (Guayas, Q1 2014)

Zielbild

Nach der Datenaufbereitung aus Week 1 wollte ich in Week 2/3 zwei Dinge erreichen:

1. ein **robustes, schnelles Baseline-Modell** für Daily-Forecasts im Guayas-Gebiet (Jan–März 2014) und

2. ein **sauberes Experiment-Setup**, das reproduzierbar ist (klare Splits, Artefakte, MLflow-Logging).

Ich habe mich zunächst bewusst **für XGBoost** entschieden. Die Gründe: sehr gute Performance auf tabellarischen Zeitreihen, wenig Tuning nötig, schnell in der Iteration – und ideal, um später in Streamlit zu deployen. Ein LSTM habe ich zusätzlich „als Vergleich" gebaut, aber mit klarer Erwartung, dass es hier nicht zwingend gewinnt (kurzes Fenster, viele klassische Kalender-/Lag-Features, kleine Validierungsphase).

Datengrundlage & Selektion

Als Trainingsbasis nutze ich das aus Week 1 erzeugte Parquet:
`guayas_Q1_2014_ml_ready_favorita.parquet` (Zeitraum 2014-01-01 bis 2014-03-31; ~996 k Zeilen).
Ich bleibe konsequent auf **Guayas-Stores** und schränke die Artikel auf die **Top-3 Familien** ein: *GROCERY I*, *BEVERAGES*, *CLEANING*. Das ist ein pragmatischer Trade-off: weniger Rauschen, trotzdem relevanter Abdeckungsgrad.

Zur **Validierung** habe ich mich bewusst gegen Shuffle und für harte Zeitfenster entschieden:

- **Train:** bis **2014-02-21** (Februar minus letzte 7 Tage)

- **Validation: 2014-02-22 bis 2014-02-28** (7-Tage-Fenster direkt vor März)

- **Test: 2014-03-01 bis 2014-03-31** (Hold-out für Reporting)

Damit verhindert der Split, dass die Modelle ungewollt in die Zukunft schauen. Außerdem ist die Validierung nah am Testfenster, was die Parametereinschätzung stabiler macht.

Feature Engineering (konservativ, aber wirkungsvoll)

Ich wollte die Pipeline **einfach & reproduzierbar** halten – und gleichzeitig die wichtigsten „sicheren“ Zeitreihen-Signale abgreifen.

Kalender & Struktur

- `day_of_week`, `month`, `is_weekend`
- Merge der **Item-Meta** (u. a. `perishable`, `class`) und **Store-Meta** (`cluster`, `store_type_id`) – beides rein numerisch encodiert.

Lags & Rollings

- Vorhanden aus Week 1: `lag_1`, `lag_7`, `lag_14`, `lag_30`, `roll_mean_7`, `roll_std_7`
- Ergänzt: `lag_2`, `lag_3`, `lag_5`, `lag_10`, `lag_21`
- Zwei einfache **Ratio-Features** als Robustheits-„Checks“: `ratio_1_7` (lag_1/roll_mean_7) und `ratio_7_30` (lag_7/lag_30)

Exogene Größen

- `transactions` (store-täglich) und `dcoilwtico` (nationaler Ölpreis; interpoliert) – beides als schwache/sekundäre Treiber gedacht.

Promotion

- `onpromotion` sauber nach 0/1 normalisiert (sofern vorhanden).

Die finale Featureliste, die ich auch exportiere, ist darum relativ kompakt, aber deckt die „klassischen“ Zeitreihen-Hebel gut ab.

Baseline-Modell: XGBoost (mit Early Stopping)

Ich starte mit einem nüchternen XGB-Setup (hist-Tree, `eta=0.07`, `max_depth=6`, `subsample/colsample=0.9`, L2-Reg, RMSE-Frühabbruch). Trainiert wird auf **Train(pure)**, das **7-Tage-Fenster** davor dient als **Early-Stopping-Validation**. Anschließend evaluiere ich auf dem **März-Test**.

Test-Ergebnis (März 2014, Summen über alle Serien):

- **MAE = 3.39**, **RMSE = 7.93**
- Bias sehr klein (-0.12), sMAPE ≈ **51.95%**
- (Hinweis: MAPE ist in dieser Domäne teilweise wenig aussagekräftig, v. a. wegen vieler sehr kleiner Y-Werte.)

Das ist für eine erste Baseline absolut solide – vor allem, weil sie **stabil** trainiert und reproduzierbar ist.

Schlankes Tuning & „Best Model“

Ich habe danach **Random-Search** über ein kleines Gitter laufen lassen (25 Stichproben) – weiter mit Early Stopping und demselben Validierungsfenster.
Die **beste Validierungskonfiguration** war:

- `eta=0.07`, `max_depth=4`, `subsample=0.7`, `colsample_bytree=0.7`, `reg_lambda=2.0`, `min_child_weight=3`.

Auf dem **Test-März** liegt das getunte Modell bei:

- **MAE = 3.42**, **RMSE = 7.90**, Bias ≈ −0.10, sMAPE ≈ **53.06%**.

Unterm Strich: **kein großer Sprung** gegenüber der Baseline – was ich als gutes Zeichen werte. Die Baseline war bereits nahe am sweet spot; zu aggressive Tiefe/Subsampling hilft hier nicht weiter, weil die Serie stark durch **nahe Lags/Rollings + Kalender** geprägt ist.

LSTM – bewusst nur als „Beifang“

Ich habe zusätzlich ein kleines **PyTorch-LSTM** (2 LSTM-Layer, Dropout, 25 Epochen, Early Stopping) trainiert. Die Daten nutze ich ohne große Sequenz-Architektur-Optimierung; Fokus war eher „Signal prüfen“ als „SOTA bauen“.

Ergebnis (Test-Split in derselben Periode):

- **MAE ≈ 6.98**, **RMSE ≈ 13.49**, sMAPE ≈ **87.14%**.

Das bestätigt meine Erwartung: **ohne** längere Historienfenster, aufwendige Sequenzierung pro Store×Item und harte Regularisierung ist XGB hier klar im Vorteil. Ich habe das LSTM aber im MLflow dokumentiert – und optional als TorchScript exportiert, falls ich später noch einmal experimentieren will.

Backtesting, Erklärbarkeit & Diagnose

Rolling-Origin-Backtesting (XGB)
Mit `TimeSeriesSplit` (5 Folds) prüfe ich die Stabilität über das gesamte Q1-Fenster (immer nur Vergangenheit gegen Zukunft).
Ø-Werte (über Folds): MAE ≈ 3.39, **RMSE ≈ 8.02**, sMAPE ≈ **51.64%**.
→ Das deckt sich sehr gut mit der März-Performance und spricht für **robuste Generalisierung**.

SHAP-Analyse (ohne zusätzliches Paket, via `pred_contribs`)
Die globalen Beiträge zeigen genau das erwartete Bild:

- **Roll- und Lag-Features** (v. a. `roll_mean_7`, `lag_7/14/21/30`) dominieren,
- **Transactions** hat einen positiven, aber sekundären Einfluss,
- **Store-Identität** (z. B. `store_nbr`) wirkt als Proxy für dauerhafte Level-Unterschiede,
- **Ölpreis (`dcoilwtico`)** ist eher ein **Randfaktor** (kleine, teils negative Beiträge).

Residual-Checks
Ich werte die Fehler nach **Store**, **Item** und **Wochentag** aus.

- Stores mit besonders hohen medianen Abweichungen sind u. a. **51, 34, 24** – üblich, weil High-Volume-Standorte jede kleine prozentuale Abweichung in absoluten Zahlen verstärken.
- Bei einzelnen **Items** sieht man deutliche Median-Abweichungen; hier vermute ich **Sondereffekte** (sporadische Verkäufe, Promotions, Stockouts), die unser simples Feature-Set nicht vollständig einfängt.
- Nach **Wochentag** liegen die Mediane recht dicht zusammen; Wochenende ist erwartungsgemäß etwas volatiler.

Ausreißer-Erkennung
Mit einer robusten MAD-Schwelle (≈ 6×MAD) markiere ich ~12,5 k starke Ausreißer im

Backtest. Das nutze ich nicht zum Härten (keine harten Clamps), sondern als **„Watch-Liste“** – nützlich für spätere Business-Regeln/Overrides im Planner-Kontext.

Prognoseintervalle (Conformal)

Um die Kommunikation mit Planern praxisnäher zu machen, gebe ich **additive Konfidenz-Bänder** aus den Backtest-Residuen:

- **90 %-Halbbreite ≈ 7.22** → empirische Coverage **~90.6 %** (nahe am Ziel)
- zusätzlich sichere ich **95 %** als Referenz.
  Diese Quantile exportiere ich als JSON, damit die Streamlit-App später automatisch Intervalle mitschreiben kann.

Ensemble – nur vorsichtshalber

Ich habe testweise ein **lineares Ensemblé** aus XGB und LSTM über das Validierungsfenster gewichtet. Ergebnis: **bestes Gewicht α(LSTM)=0.000** – sprich, **XGB allein** schlägt die Mischung. Das bestätigt noch einmal, dass das LSTM in dieser Setup-Variante keinen Mehrwert bringt.

Data-Drift-Indikatoren (PSI)

Für ein Gefühl, ob sich zwischen Train (Jan–Feb) und Test (März) die Eingaben verschieben, berechne ich **PSI** je Feature.

- `dcoilwtico` (Ölpreis) zeigt erwartungsgemäß einen **starken Drift** (PSI > 5) – nicht kritisch, weil der Einfluss ohnehin klein ist.
- `transactions` hat **moderaten Drift** (~0.26) – das ist plausibel, weil saisonale Nachfrage im März leicht verschoben ist.
- Die **Lag/Roll-Features** bleiben weitgehend stabil (PSI << 0.2).

Für die Praxis heißt das: Öl kann man für Deployment sogar weglassen, ohne Performance-Verlust zu riskieren; `transactions` sollte man aber weiter im Blick behalten.

Ergebnisse & kleine Bilanz

- **Bestes Modell: XGBoost** (getunt)
    - **Test März 2014: MAE ~ 3.42**, **RMSE ~ 7.90**, **sMAPE ~ 53 %**
    - Sehr ähnliche Werte im **Rolling-Backtest** → **robust**.
- **LSTM** (baseline-artig) ist **klar schwächer** (RMSE ~ 13.5) – für dieses Setup kein Gewinn.
- **Wichtigste Treiber**: kurze **Lags/Rollings** + simple **Kalender-Signale**.
- **Conformal-Intervalle** funktionieren gut (Coverage ≈ 90 %).
- **Drift** ist unkritisch außer beim Öl, das ohnehin wenig treibt.

Warum ich bei XGB bleibe:
Ich bekomme **gute Güte bei hoher Geschwindigkeit** und kann das Modell mit sehr wenig Ballast in eine **Streamlit-App** integrieren. Außerdem ist die Erklärung (SHAP) für Planner einfacher zu vermitteln („starker Einfluss der Vorwoche“, „Wochenend-Effekt“).

Artefakte & Reproduce-Story

Ich habe alles so abgelegt, dass ich es ohne Notebooks benutzen kann:

- **Modelle:**
    - `artifacts_week2_3/xgb_booster.json` (finales Booster-Modell)
    - `models/xgb_best.json`, `models/xgb_baseline.json` (für Vergleich/Archiv)
- **Features & Meta:**
    - `artifacts_week2_3/features.json` (Feature-Liste + Target)
    - `artifacts_week2_3/preprocessing_meta.json` (Fenster, Dataset, usw.)
- **Unsicherheit & Tools:**

    - `artifacts_week2_3/conformal_intervals.json` (90/95 %-Hüllen)
    - `artifacts_week2_3/predict.py` (kleiner Inferenz-Helper)
    - `artifacts_week2_3/streamlit_app_stub.py` (Starter-App: CSV laden → predict → CSV export)
- **Experiment-Tracking:**
    - **MLflow-Runs** für Baseline, Tuning-Winner, LSTM und Backtest-Aggregat (Metriken + Plots).

Damit ist die Brücke zu Week 4 praktisch fertig: App kann die Artefakte direkt laden, CSVs mit bereits erzeugten Features entgegennehmen und Vorhersagen inkl. 90 %-Intervall exportieren.

Was ich bewusst nicht gemacht habe (und warum)

- **Komplexe hierarchische Modelle** (pro Store×Item separate Fits): wäre aufwändig und bringt in diesem engen Zeitfenster wenig Zusatznutzen.
- **Aufwendige LSTM/Seq2Seq**: dafür brauche ich längere Historien, geschickte Aggregation/Normalisierung pro Serie und mehr Trainingstime.
- **Feature-Explosion** (z. B. Dutzende rolling Fenster): die Gefahr von Leakage/Instabilität steigt; mein Fokus war **saubere, faire Splits** und **wenige, starke Features**.

Nächste Schritte (falls Zeit bleibt)

- **Promo-Signale verfeinern** (z. B. Lag-gedämpfte Promo-Effekte, Dauer einer Promo als Feature).
- **Store-/Item-Clustering** und pro Cluster leichte **Model-Abzweigungen**.
- **Kalender-Holidays** sauberer mappen (lokale Feiertage in Ecuador, Transfer-Holidays).

- **Planner-Friendly UI** in Streamlit (Store/Item-Picker, N-Day-Forecast, Export + Intervall-Bänder).

Kurzfazit

Für den Bewertungsfokus von Week 2/3 habe ich eine **stabile und gut erklärbare XGBoost-Pipeline** aufgebaut, mit **klaren Zeit-Splits**, **Backtesting**, **Erklärbarkeit** (SHAP), **Unsicherheitsbändern** (Conformal) und einem **sauberen Artefakt-Handover** Richtung Streamlit.
Das Ergebnis ist absichtlich pragmatisch: **einfach, schnell, reproduzierbar** – und genau das macht es für die spätere App und für Demand-Planner im Alltag nutzbar.