Below is a **battle-tested blueprint** for building an end-to-end AI system that (a) predicts marketing KPIs with ≥ 95 % accuracy, (b) quantifies *diminishing returns* so you can say exactly “how high is high?”, and (c) exposes everything to business users through a Streamlit dashboard.

**0. Project framing**

| **Goal** | **Success metric** |
| --- | --- |
| Predict revenue, purchases & ROAS for any (platform × funnel × budget × period) scenario. | **R² ≥ 0.95** *or* **MAPE ≤ 5 %** on an unseen hold-out set. |
| Detect the *“knee” budget* (point where marginal ROAS falls below a chosen threshold). | Knee error ≤ ±10 % when back-tested on historical data. |
| Recommend budget splits that maximise total revenue at a given spend ceiling. | ≥ 3 pp ROAS uplift vs. current allocation in an A/B test. |

Use those metrics in QA and in the dashboard to prove the model’s value.

**1. Data foundation**

**1.1 Raw tables you’ll need**

* **Media log** – one row per day, platform, funnel: *costs, impressions, clicks, purchases, revenue*
* **Meta-data** – platform type, objective, placement, creative format, audience size, etc.
* **Calendar features** – holiday flags, pay-day flags, seasonality indices.

**1.2 Cleaning & augmentation**

1. **Sanity checks** – negative spends, non-monotonic cumulative metrics, extreme outliers (clip to P1/P99).
2. **Lagged & ad-stock variables** (esp. for brand/upper funnel):

AdStockt=Spendt+λ AdStockt−1\text{AdStock}\_t = \text{Spend}\_t + \lambda \,\text{AdStock}\_{t-1}AdStockt​=Spendt​+λAdStockt−1​

Tune λ during modelling.

1. **Non-linear transforms** – log\_costs, sqrt\_impressions, interaction dummies (platform × funnel).

*Tip*: keep a reproducible data-prep notebook; feed exactly the same features to training and to Streamlit.

**2. Modelling pipeline**

**2.1 Base predictor**

* **Model type**: Gradient-boosted trees (LightGBM or XGBoost) wrapped in MultiOutputRegressor for *(revenue, purchases, clicks, impressions)*.
* **Hyper-parameter search**: run Optuna/Bayesian optimisation on 5-fold **time-series** CV.
* **Feature importance**: store SHAP values – exposed later for transparency.

**2.2 Accuracy ≥ 95 %**

* **Lag-aware CV** – sliding-window split so the test fold is always future-dated. Prevents leakage and over-optimistic R².
* **Ensembling** – average 5 models trained on different seeds; gains ~2-4 pp R².
* **Error diagnostics** – plot residuals vs. spend & calendar. If heavy heteroscedasticity, train on *log* targets plus a second model to predict variance.

**2.3 Diminishing-return layer**

Even with XGB, predictions can stay linear outside the data range. Append a **response-curve fitting** step:

python

CopyEdit

def hill(spend, V\_max, k, n):

return (V\_max \* spend\*\*n) / (k\*\*n + spend\*\*n) # logistic/Hill

for plat\_funnel in combos:

spend\_grid = np.linspace(0, max\_seen\*3, 200)

rev\_pred = base\_model.predict(...)

popt, \*\_ = curve\_fit(hill, spend\_grid, rev\_pred, bounds=(0, np.inf))

curves[plat\_funnel] = popt # save params

*The Hill (or negative-exponential) function guarantees saturation and differentiability.*

**Marginal ROAS** = derivative of the curve ÷ cost —> lets you pin-point where incremental returns drop below, say, 1:1.

**3. Budget-allocation optimiser**

Define decision variables sp,fs\_{p,f}sp,f​ (spend by platform p, funnel f).

vbnet

CopyEdit

Maximise Σ Revenue\_p,f(s\_p,f)

Subject to Σ s\_p,f = TotalBudget

s\_p,f ≥ 0

*Small budgets (< 5 platforms) → use scipy.optimize.minimize with SLSQP.*  
*Larger (dozens of channels) → use Google OR-Tools linear solver with piecewise-linear curve approximation.*

Return:

python

CopyEdit

pd.DataFrame({

'Platform': ...,

'Funnel': ...,

'Optimal Spend': ...,

'Expected Revenue': ...,

'ROAS': ...

})

**4. Streamlit front-end**

**4.1 Layout concept**

mathematica

CopyEdit

┌───────────────────────────────────────────────┐

│ Sidebar │

│ • Date range selector │

│ • Total budget slider │

│ • Funnel weight sliders (Upper/Mid/Lower) │

│ • Check-box: “show diminishing-return curve” │

└───────────────────────────────────────────────┘

┌──────────────┬───────────────────────────────┐

│ Tab 1: KPI │ Tab 2: Spend vs. Return │

│ Forecast │ Tab 3: Optimal Allocation │

│ │ Tab 4: Model Diagnostics │

└──────────────┴───────────────────────────────┘

**4.2 Key interactions**

* **Predict KPIs** (st.button) → calls predict\_kpis() with sidebar inputs.
* **Dynamic curve plot** – use Plotly: x = spend slider, y = predicted revenue; overlay marginal ROAS line.
* **“How high is high”** – show a vertical dashed line at the knee and text:

“Spending beyond SAR X drives < Y % incremental ROAS – consider reallocating.”

**4.3 Performance tricks**

1. @st.cache\_resource for loading model & curve params.
2. SessionState to remember user scenarios across tabs.
3. Use st.data\_editor() so power users can paste custom spend plans and watch KPIs recalc in real time.

**5. Validation & monitoring**

* **A/B guard-rails** – before rollout, run a split test: model-guided allocation vs. business-as-usual.
* **MLflow** – track each re-train’s R², MAE, curve parameters, feature schema hash.
* **Drift alert** – if weekly MAPE > 6 %, trigger automatic re-train & push new weights to Streamlit (GitHub Actions + Streamlit Cloud).

**6. Deliverables checklist**

* Reproducible data-prep notebook (01\_data\_prep.ipynb).
* Python package marketing\_ai with model.py, curves.py, optimizer.py.
* Streamlit app (app.py) & requirements.txt.
* Unit tests (PyTest) covering feature generation, curve fitting, optimiser.
* README with setup + expected accuracy benchmarks.
* One-pager methodology deck for stakeholders.