Deep Dive: Understanding the Parameters Behind ARIMA and XGBoost Forecasting Models

When it comes to forecasting business outcomes like sales, everyone talks about **which model performs better** — ARIMA or XGBoost.
But few people actually explain **why** a model performs better, and **what each parameter is really doing behind the scenes.**

So let's decode the mechanics of the four forecasting models used in our experiment — from the classical ARIMA to the tree-powered XGBoost — and understand their parameters like professionals.

🧩1️⃣ARIMA (No Exogenous Variables)

Code:

```
model_arima = ARIMA(
    endog = train['sales_units'],
    order = (2, 0, 2),      # ARIMA(p,d,q)
    trend = 'ct',           # constant + linear trend
    enforce_stationarity = True,
    enforce_invertibility = True
)
fit_arima = model_arima.fit()
forecast_arima = fit_arima.forecast(steps=len(test))
```

🔍 Parameter Meaning

Parameter	Purpose	Explanation
endog	Target series	The variable being forecasted — here `sales_units`

order=(p,d,q)	ARIMA order	`(p)` = AR lag count, `(d)` = differencing, `(q)` = MA lag count
p (2)	Autoregressive term	Looks back two time steps to model persistence or memory
d (0)	Differencing term	Since the series is stable, no differencing is applied
q (2)	Moving average term	Corrects based on last two forecast residuals (smoothing noise)
trend='ct'	Deterministic trend	Adds a constant and linear time slope
enforce_stationarity	Model stability	Keeps AR roots within the unit circle (prevents blow-up)
enforce_invertibility	Residual stability	Ensures the MA process can be inverted for forecasting

🧠 Interpretation

ARIMA(2,0,2) learns a combination of past sales and their recent errors while adding a mild time-based slope.

✅ Best for smooth, trend-driven data
❌ Limited when external factors affect sales

🧩2️⃣ ARIMA (With Exogenous Variables)

Code:

```
exog_train = train[['calls_made','is_holiday_week']]
exog_test  = test[['calls_made','is_holiday_week']]

model_arima = ARIMA(
    endog = train['sales_units'],
    exog  = exog_train,
    order = (2, 0, 2),
    trend = 'ct',
    enforce_stationarity = True,
```

```
    enforce_invertibility = True
)
fit_arima = model_arima.fit()
forecast_arima = fit_arima.forecast(steps=len(test), exog=exog_test)
```

🔍 New Parameter Introduced

Parameter	Purpose	Explanation
exog	External regressors	Other features that influence the target (e.g., marketing activity, holidays)
forecast(..., exog=exog_test)	Future exog inputs	Provides known future values for external variables during forecasting

🧠 How It Works

This version adds a regression layer over ARIMA — combining **temporal memory** (p,d,q) with **external context** (`calls_made`, `is_holiday_week`):

$$Sales_t = \alpha + \beta_1 t + \phi_1 Sales_{t-1} + \gamma_1 Calls_t + \gamma_2 Holiday_t + \epsilon_t$$

✅ Adds real business intelligence
✅ Coefficients explain cause and effect
❌ Still linear — doesn't capture complex interactions

🌳3️⃣XGBoost (No Exogenous Variables)

Code:

```
model_xgb = XGBRegressor(
    objective='reg:squarederror',
    n_estimators=400,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
```

```
)
model_xgb.fit(X_train, y_train)
forecast_xgb = model_xgb.predict(X_test)
```

🔍 Parameter Meaning

Parameter	Purpose	Explanation
objective='reg:squarederror'	Task type	Regression for continuous targets
n_estimators=400	Number of trees	Total boosting iterations (each tree corrects previous errors)
learning_rate=0.05	Step size	Controls how fast trees learn — smaller = slower but more stable
max_depth=5	Tree complexity	Max number of splits per tree; deeper trees capture more interactions
subsample=0.8	Row sampling	Uses 80% of data rows per iteration (adds randomness to reduce overfit)
colsample_bytree=0.8	Column sampling	Uses 80% of features per tree (improves generalization)
random_state=42	Reproducibility	Ensures consistent results

🧠 Interpretation

- Builds **400 small decision trees**, each improving upon the previous.
- Uses **lag features (`lag_1`, `lag_2`, `rolling_mean_3`)** to mimic temporal patterns.
- Nonlinear and flexible — but not inherently time-aware.

✅ Great for nonlinear relationships
❌ Needs large data and good feature engineering

🌳4️⃣XGBoost (With Exogenous Variables)

Code:

```
features =
['lag_1','lag_2','rolling_mean_3','calls_made','is_holiday_week']

model_xgb = XGBRegressor(
    objective='reg:squarederror',
    n_estimators=400,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)
model_xgb.fit(train_xgb[features], train_xgb['sales_units'])
forecast_xgb = model_xgb.predict(test_xgb[features])
```

🔍 Feature Meaning

Feature	Purpose	Explanation
lag_1, lag_2	Temporal memory	Previous sales observations
rolling_mean_3	Local trend	3-week average for smoothing
calls_made	Marketing driver	High calls → likely sales lift
is_holiday_week	Event flag	Holidays → expected drop

🧠 Interpretation

XGBoost now blends **temporal** and **business context** to forecast:

> "If recent sales were strong and call volume is high, expect an uptick — unless it's a holiday week."

✅ Best performance when data is rich & dynamic
❌ Harder to interpret — uses hundreds of split conditions

⚖️ Parameter Summary Table

Model	Type	Core Parameters	Core Idea
ARIMA (No Exog)	Statistical	`(p,d,q)`, `trend`	Learns from past values
ARIMA (With Exog)	Regression + Statistical	`(p,d,q)`, `trend`, `exog`	Past + external effects
XGBoost (No Exog)	ML (nonlinear)	`n_estimators`, `learning_rate`, `max_depth`	Learns nonlinear lag patterns
XGBoost (With Exog)	ML + Regression	Above + external features	Learns complex relationships between time and context

---

🧠 How They Differ

Aspect	ARIMA	XGBoost
Model Nature	Statistical	Machine Learning
Relationship Type	Linear	Nonlinear
Needs Lag Features?	No	Yes
Can Use Exogenous Data?	Yes	Yes
Interpretability	High	Medium
Performance on Smooth Data	Better	Lower
Performance on Noisy Data	Lower	Better

🧩 Final Thought

> ARIMA **explains the story** behind your forecast.
> XGBoost **adapts to the unknowns** your business throws at it.

They aren't rivals — they're **tools for different types of data**.
Understanding the parameters behind each is how you move from "model building" to **true analytical storytelling.**