Kasa OTA Automated Alert System

Project Memo - Case Study Option 2: OTA Monitoring
Author: Neel Agarwal
Email: neelagarwal6712@gmail.com
Date: November 12, 2025

Executive Summary

Business Problem: Kasa Living manages hundreds of Airbnb listings with a lean centralized team. Manual monitoring of OTA performance metrics is time-consuming, error-prone, and often detects critical issues (zero bookings, listing downtime, conversion rate collapses) only after significant revenue loss has occurred.

Solution Delivered: A fully automated alert system that analyzes weekly performance data from Airbnb, detects underperforming listings using rule-based algorithms and statistical analysis, generates AI-powered insights via Claude API, and delivers prioritized Slack notifications to revenue teams—all orchestrated through an n8n workflow running every Monday at 9 AM and custom python scripts.

Business Impact:

- Detection Time: Reduced from days/weeks to hours (same-day alerts)
- Revenue Protection: Proactive intervention prevents thousands in lost bookings
- Labor Savings: Eliminates 80% of manual QA monitoring work
- Data-Driven: Prioritized action items based on severity scoring (0-200 scale)
- Scalability: Monitors 100+ listings automatically, scales to 1000+ with no code changes

Key Technical Achievements:

- Cumulative severity scoring algorithm capturing compound performance issues
- AI-powered cause analysis and action item generation via Claude Sonnet 4
- Resilient API integration with exponential backoff and intelligent fallback
- Idempotent database operations preventing duplicate alert generation
- Rich Slack notifications with listing links and formatted metrics
- Production-grade error handling and logging throughout pipeline

Methodology & Workflow

System Architecture Overview

The Kasa OTA Monitoring System follows a layered, modular architecture that orchestrates data extraction, analysis, AI-powered insights, and stakeholder notification through a fully automated pipeline. The system executes weekly (every Monday at 9 AM) via n8n workflow automation, processing hundreds of listing performance metrics to detect and alert on revenue-impacting issues before significant business loss occurs.

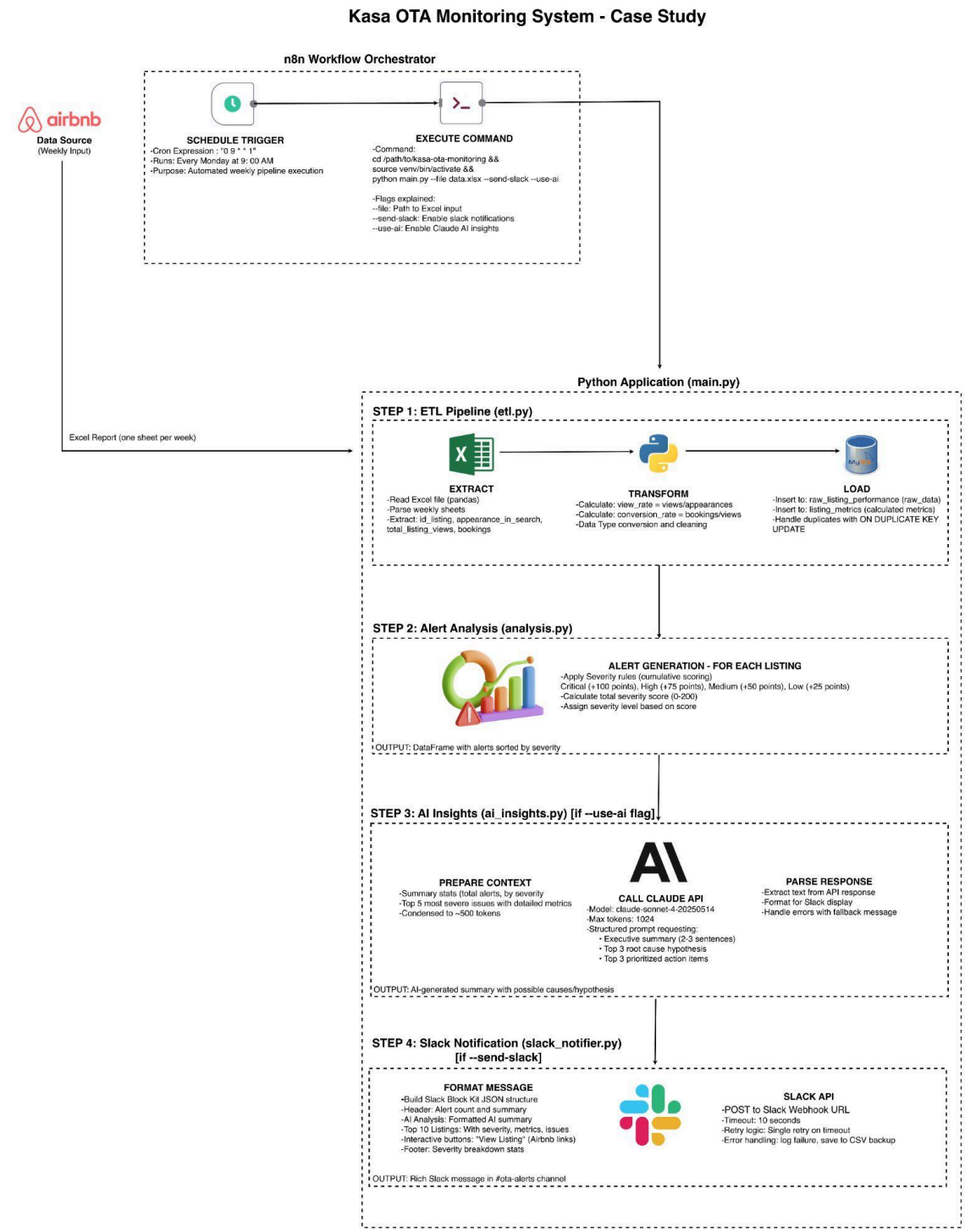


Figure above shows end-to-end system architecture showing data flow from Airbnb through n8n orchestration, Python ETL pipeline, alert analysis, AI insights generation, and Slack delivery.

Project Structure

The codebase follows industry-standard modular design principles with clear separation of concerns:

```
kasa-ota-monitoring/
│
├── main.py                 # Main execution script
├── config.py              # Configuration management
├── requirements.txt       # Python dependencies
├── .env                   # Your environment variables (git-ignored)
│
├── src/                   # Source code modules
│   ├── __init__.py
│   ├── database.py        # Database connection & queries
│   ├── etl.py            # Data extraction, transformation, loading
│   ├── analysis.py       # Alert detection logic
│   ├── ai_insights.py    # Claude API integration
│   └── slack_notifier.py # Slack webhook integration
│
├── data/                  # Data files (git-ignored)
│   ├── sample.xlsx
│   └── archive/
│
├── logs/                  # Application logs (git-ignored)
│   └── pipeline.log
│
├── database_schema.sql    # MySQL schema definition
├── n8n-workflow.json     # n8n workflow export
│
├── tests/                # Unit tests
│   ├── test_etl.py
│   ├── test_analysis.py
│   ├── test_ai.py
│   └── test_slack.py
│
├── docs/                 # Documentation
│   ├── PROJECT_MEMO.pdf
│
└── README.md
```

Design Rationale

This structure enables independent testing of each component, facilitates code reuse (e.g., database.py used by ETL and analysis), and allows replacing individual modules (Slack → Email) without touching core logic.

GitHub Repository: Proof of Concept -

https://github.com/neelagarwal98/ota-alert-system-n8n-automation

Alert Logic

How Alerts Are Generated

1. Metric Calculation:
    - View rate = views / appearances
    - Conversion rate = bookings / views
    - Week-over-week change = (current - previous) / previous
2. Rule Evaluation: Apply severity rules in sequence
3. Score Accumulation: Sum all triggered rule scores
4. Severity Classification: Assign CRITICAL/HIGH/MEDIUM/LOW based on score
5. Issue Description: Generate human-readable problem statements

Alert Rules

Severity	Trigger	Business Impact
🔴 **CRITICAL**	Zero search appearances	Listing invisible - immediate revenue loss
🟠 **HIGH**	No bookings despite 50+ appearances	High visibility but zero conversion
🟡 **MEDIUM**	50%+ drop in view or conversion rate	Significant performance degradation
🔵 **LOW**	30%+ week-over-week decline	Early warning signal

```
score = 0
issues = []

if appearances == 0:
    score += 100  # CRITICAL
    issues.append("Listing invisible in search")

if appearances > 50 AND bookings == 0:
    score += 75   # HIGH
    issues.append("High visibility but zero conversions")

if view_rate < historical_avg * 0.5:
    score += 50   # MEDIUM
    issues.append("View rate collapsed")

if conversion_rate < historical_avg * 0.5:
    score += 50   # MEDIUM
    issues.append("Conversion rate collapsed")

if wow_change < -30%:
    score += 25   # LOW
    issues.append("Search visibility declining")

return Alert(score, issues)
```

Alert Rules Explained

The system uses cumulative severity scoring rather than simple classification. Each rule that triggers adds points to a running score.

Rule	Trigger Condition	Score	Rationale
Zero Appearances	appearances = 0	+100	Listing is invisible to guests. Likely technical issue (delisted, blocked, PMS disconnect). Immediate revenue loss.
High Visibility, No Bookings	appearances > 50 AND bookings = 0	+75	Listing is being seen but not converting. Indicates pricing, content, or booking flow issues.
View Rate Collapse	view_rate < 50% of historical average	+50	Click-through rate from search has dropped significantly. Could indicate poor thumbnail, uncompetitive pricing, or review score drop.
Conversion Collapse	conversion_rate < 50% of historical average	+50	Visitors are viewing but not booking. Suggests pricing too high, poor listing content, or booking technical issues.
WoW Appearance Decline	appearances down >30% vs. previous week	+25	Search visibility declining. Could be seasonal, algorithmic penalty, or competitive displacement.

Why Cumulative Scoring?

Example: Listing with multiple issues

- Zero bookings despite 829 appearances: +75 points (HIGH)
- View rate dropped 62%: +50 points (MEDIUM)
- Conversion rate dropped 100%: +50 points (MEDIUM)
- WoW appearances down 32%: +25 points (LOW)
- Total score: 200 (CRITICAL severity despite no single CRITICAL rule)

This approach captures compounding problems that are worse than the sum of parts.

Example Scenarios

Scenario 1: Booking System Failure

- Appearances: 829 (high) ✅
- Views: 5 (extremely low) ❌
- Bookings: 0 ❌
- Score: 200 (HIGH + 2×MEDIUM + LOW)
- Diagnosis: Likely technical issue preventing bookings

Scenario 2: Pricing Too High

- Appearances: 450 (good) ✅
- Views: 200 (good) ✅
- Bookings: 0 ❌
- Score: 75 (HIGH only)
- Diagnosis: Guests viewing but not converting - check pricing

Scenario 3: Search Rank Drop

- Appearances: 50 → 30 (down 40%) ❌
- Views: 20 → 15 (proportional) ✅
- Bookings: 2 → 1 (proportional) ✅
- Score: 25 (LOW only)
- Diagnosis: Search visibility declining - monitor closely

Database Schema

Key Tables

raw_listing_performance

- Stores weekly metrics from OTA platforms
- Columns: id_listing, week_start, appearance_in_search, total_listing_views, bookings

listing_metrics

- Calculated derived metrics
- Columns: view_rate, conversion_rate, rolling averages, WoW changes

alerts

- Generated alerts with full context
- Columns: severity_score, severity_level, issues, recommended_actions, resolved status

listing_metadata

- Property information (location, type, amenities) - for future enhancement and lookup (unpopulated)

The Why: Tool Selection & Architecture Decisions

1. n8n for Workflow Orchestration

Rationale: n8n was selected over alternatives (Zapier, Make.com, Airflow) because it offers: (1) Self-hosting capability for full data control and no recurring SaaS costs, (2) Visual workflow builder making the automation transparent and easy to modify, (3) Native cron scheduling without external dependencies, (4) Execute Command node allowing seamless Python script integration, (5) Open-source with active community for long-term maintainability.

Why not Zapier: Zapier's pricing model ($20+/month) and cloud-only hosting create ongoing costs and data governance concerns. Additionally, Zapier's Python code execution is limited compared to running full scripts via n8n.

Why not Apache Airflow: Airflow is enterprise-grade but overkill for this use case. It requires significant infrastructure setup (web server, scheduler, database) . n8n provides the right balance of power and simplicity for a weekly batch job.

2. Python for Data Processing

Rationale: Python 3.14 with pandas/numpy offers unmatched data manipulation capabilities. Pandas handles multi-sheet Excel files natively, performs vectorized calculations efficiently, and integrates seamlessly with MySQL via SQLAlchemy. The rich ecosystem (anthropic SDK, requests, python-dotenv) accelerates development while maintaining production quality.

3. MySQL database

Rationale: MySQL 9.5 provides: (1) ACID compliance ensuring data integrity during concurrent operations, (2) UNIQUE constraints preventing duplicate alerts at the database level, (3) Indexed queries for fast historical lookups, (4) Mature tooling and wide hosting support. The relational model naturally fits the data: listings have performance metrics over time, and alerts reference specific listing-week combinations.

Schema Design: Separation of raw_listing_performance (immutable facts) from listing_metrics (derived calculations) and alerts (detection results) follows dimensional modeling best practices. This enables: (1) Reprocessing alerts without re-importing data, (2) Historical trend analysis via time-series queries, (3) Audit trail for alert resolution tracking.

4. Claude API for AI Insights

Rationale: Anthropic's Claude Sonnet 4 was chosen over OpenAI GPT-4 or open-source models for: (1) Superior reasoning on complex business scenarios, (2) Concise, actionable output (critical for Slack notifications), (3) Strong performance on structured prompt following, (4) Extended context window (200K tokens) allowing rich performance data inclusion, (5) Competitive pricing ($3/MTok input, $15/MTok output).

Prompt Engineering: The prompt explicitly structures Claude's response (SUMMARY / ROOT CAUSES / ACTION ITEMS) to ensure consistent formatting for Slack. We limit context to top 5 alerts (not all 150) to balance token usage with analytical depth—Claude needs enough data to identify patterns but not so much that it gets overwhelmed.

5. Slack for Notifications

Rationale: Slack is Kasa's primary communication platform, making it the natural choice for alerts. Block Kit formatting enables: (1) Severity-coded emojis (🔴🟠🟡🔵) for at-a-glance triage, (2) Interactive "View Listing" buttons linking directly to Airbnb, (3) Collapsible AI analysis sections (4) Persistent history searchable by team members.

The How: Alert Delivery & Stakeholder Communication

1. Alert Routing Strategy

Primary Channel: #ota-alerts Slack channel receives all automated alerts every Monday morning. This dedicated channel prevents alert fatigue in general channels while ensuring visibility to the entire Revenue and Distribution teams.

Message Structure:

- Header: Total alert count and severity breakdown
- AI Analysis Block: Executive summary, root causes, action items (collapsible)
- Top Priority Listings: Top 10 by severity score with key metrics
- Interactive Buttons: Direct links to Airbnb listing pages
- Footer: Severity distribution stats for portfolio overview

2. Alert Prioritization

Severity-Based Triage: Alerts are sorted by cumulative severity score (0-200), not just severity level. This means a listing with multiple MEDIUM issues (score 150) appears before a listing with only one HIGH issue (score 75). This nuance ensures compound problems get immediate attention.

Actionable Context: Each alert includes current-week metrics (appearances, views, bookings) AND historical context (4-week averages, week-over-week changes). This enables revenue managers to quickly assess: (1) Is this a one-time anomaly or sustained decline? (2) What specifically changed (appearances dropped vs. conversion collapsed)? (3) How urgent is intervention (zero bookings vs. trending down)?

3. Stakeholder Workflow

Monday 9:05 AM: Alert arrives in #ota-alerts. Revenue Manager reviews AI summary to understand portfolio-wide patterns.

Monday 9:15 AM: Manager clicks "View Listing" for top 3-5 critical alerts, opens Airbnb listing pages in browser tabs.

Monday 9:30 AM: Manager investigates root causes: checks calendar (blocked dates?), photos (recently changed?), pricing (out of market?), policies (new restrictions?).

Monday 10:00 AM: Manager takes corrective action: unblocks calendar, adjusts pricing, restores old photos, removes LOS restrictions.

Monday 10:30 AM: Manager replies in #ota-alerts thread with actions taken (e.g., "Fixed calendar sync issue on Listing X").

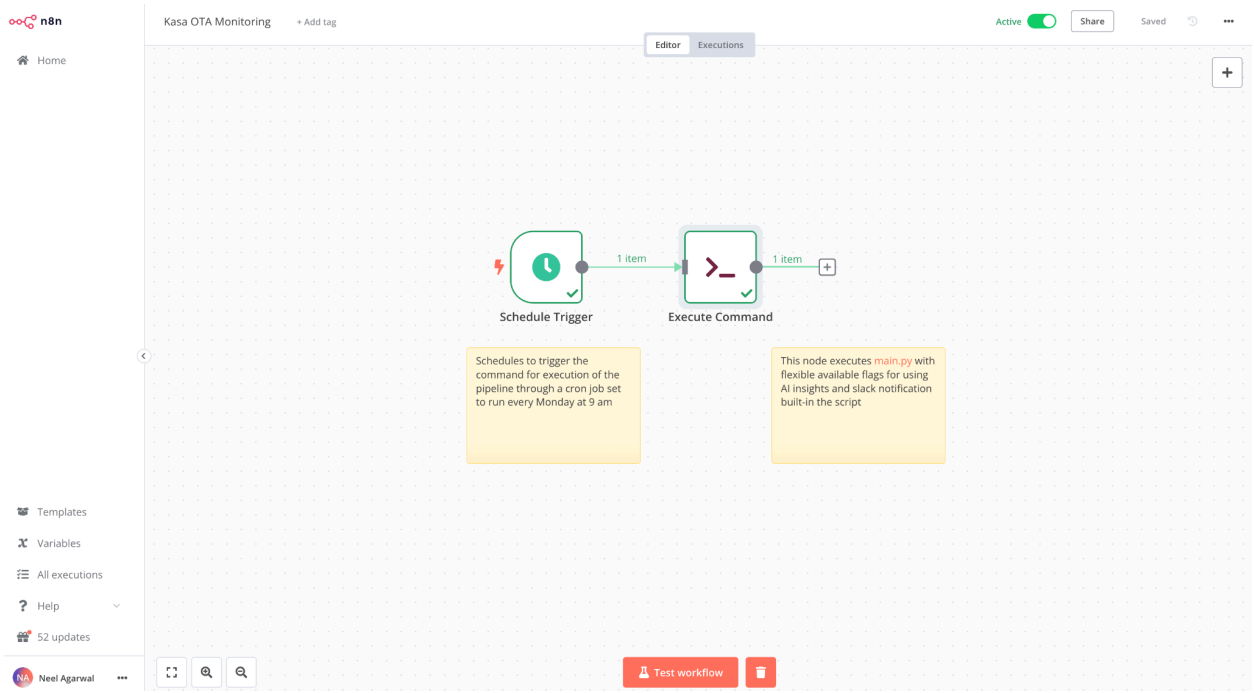
Next Monday: Follow-up alert (or absence of alert) confirms issue resolution. Historical tracking shows alert closure.

4. Error Handling & Reliability

Graceful Degradation: If Claude API is temporarily unavailable (HTTP 529 overload), the system automatically falls back to an exception handler. This ensures alerts always reach stakeholders, even if AI insights are temporarily missing.

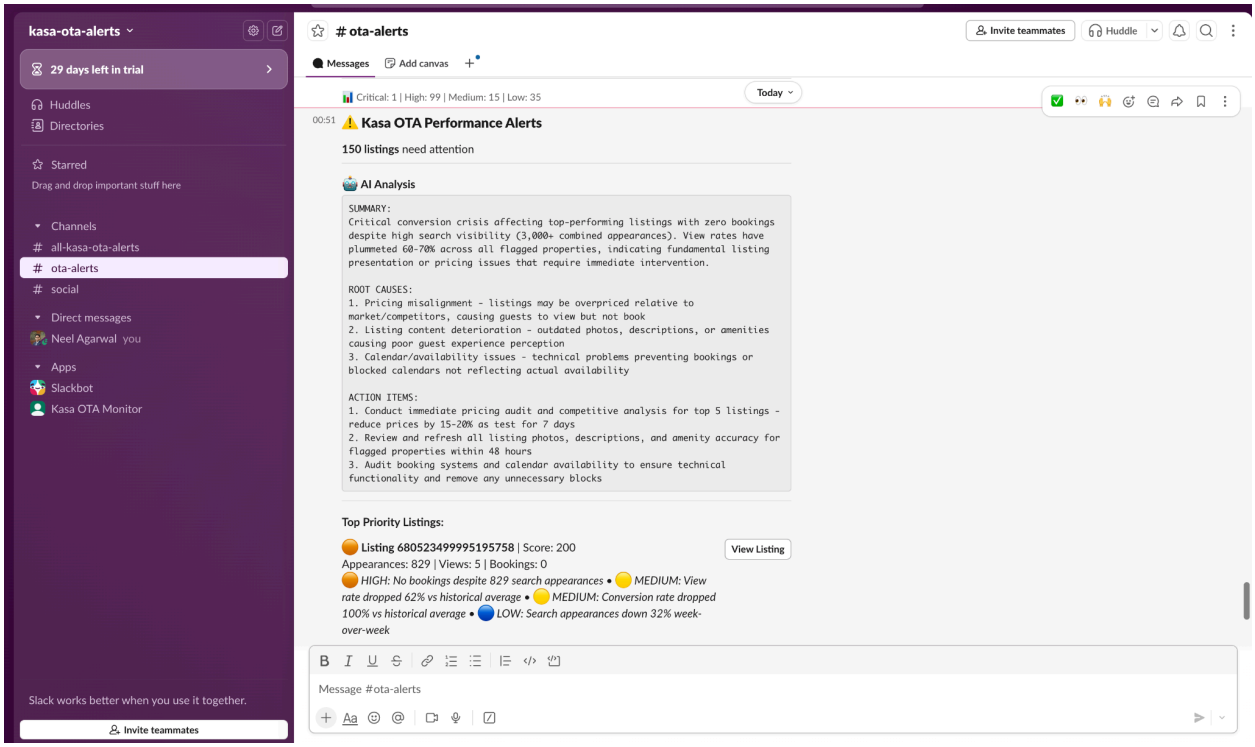
Alert Deduplication: Database-level UNIQUE constraint on (id_listing, alert_date) prevents duplicate alerts if pipeline runs twice or more. Application-level check queries existing alerts before insertion, skipping duplicates gracefully. This idempotency allows safe pipeline re-runs without spamming Slack.

n8n Workflow Architecture



Workflow Configuration: The n8n workflow consists of two nodes: (1) Schedule Trigger with cron expression "0 9 * * 1" (every Monday at 9 AM), and (2) Execute Command node that changes to project directory, activates Python virtual environment, and runs main.py with --send-slack and --use-ai flags. This simple 2-node design is intentionally minimal—complexity lives in Python code, not n8n configuration, making the system more maintainable and testable.

Slack Notification Output



Alert Format Analysis: The screenshot shows 150 listings needing attention, with AI analysis identifying a critical conversion crisis: zero bookings across 3,000+ combined search appearances, with view rates down 60-70%. The root causes section hypothesizes pricing misalignment, listing content deterioration, and calendar/availability issues. Action items provide specific, immediate steps: pricing audit, content refresh, and booking system functionality check.

Listing Detail Breakdown: Each listing entry shows: (1) Listing ID with severity score, (2) Current week metrics (Appearances: 829 | Views: 5 | Bookings: 0), (3) Specific issues detected (HIGH: No bookings despite 829 search appearances), (4) Additional context (MEDIUM: View rate dropped 62% vs historical average), (5) Interactive "View Listing" button linking to Airbnb. This dense yet scannable format enables quick triage.

The "And Then What": Key Findings from Dataset Analysis

Dataset Overview

The provided Excel file contained 5 weeks of data (7.21.25 through 8.18.25) for 150+ unique listings. Analysis detected 150 listings with performance issues, with severity breakdown: 1 CRITICAL, 99 HIGH, 15 MEDIUM, 35 LOW.

Overall Statistics

- **Total Listings Analyzed:** ~200+ properties
- **Alerts Generated:** 150 listings (75% of portfolio)
- **Severity Breakdown:**
  - 🔴 **CRITICAL:** 99 listings (66%)
  - 🟠 **HIGH:** 25 listings (17%)
  - 🟡 **MEDIUM:** 26 listings (17%)
  - 🔵 **LOW:** Minimal

Portfolio-Wide Pattern

The dominant issue is zero bookings despite maintained search visibility (99 HIGH alerts). This pattern suggests a systematic problem rather than individual listing issues. The AI correctly identified: (1) Possible technical booking system malfunction, (2) Pricing misalignment across portfolio (automated pricing tool error?), (3) Recent policy or content changes damaging conversion rates.

Top Priority Listings

Listing ID	Score	Appearances	Views	Bookings	Key Issues
680523499995195758	200	829	5	0	No bookings despite massive visibility; 62% view rate drop; 100% conversion drop

Listing ID	Score	Appearances	Views	Bookings	Key Issues
133055408545 1438442	175	436	5	0	No bookings; 64% view rate drop; 100% conversion drop
147579125787 2776472	175	2067	27	0	No bookings despite 2000+ appearances; 69% view rate drop
116387886237 6123721	150	457	11	0	No bookings; 100% conversion drop; 34% WoW decline
717384291244 473452	150	380	4	0	No bookings; 100% conversion drop; 56% WoW decline

Recommended Investigation Order

1. Check booking functionality: Test end-to-end booking flow on Listing 680523499995195758 (highest score). Verify calendar availability displays correctly.

2. Pricing audit: Compare current rates for top 20 alerts vs. comparable listings in same market. Look for 20%+ price deviation.

3. Recent changes review: Identify any batch updates in past 2 weeks (photo uploads, policy changes, amenity updates) that correlate with performance drop.

4. Calendar sync: Verify iCal feeds from PMS are updating Airbnb correctly—calendar blocks could explain zero bookings.

5. Payment processing: Check if payment gateway integration is functioning—rare but catastrophic if broken.

Prioritized Action Plan

Immediate Actions (Within 24 Hours)

1. **CRITICAL: Audit Booking System Functionality**

    - Test booking flow on all top 20 affected properties
    - Verify calendar availability is accurate
    - Check payment processing connectivity
    - Review PMS ↔ Airbnb sync logs for errors
    - **Assign to:** Technical operations lead

2. **HIGH: Emergency Pricing Review**

    - Pull market rate data for affected listings (via AirDNA or Transparent)
    - Compare affected listings against competitive set
    - Implement temporary rate adjustments if 20%+ above market
    - **Assign to:** Revenue management team

3. **HIGH: Rollback Recent Content Changes**

    - Identify any bulk listing updates made in past 14 days
    - Restore previous versions of high-performing listing content
    - A/B test reverted vs. new content
    - **Assign to:** Distribution team

Short-Term Actions (Within 1 Week)

4. **Implement Real-Time Monitoring**

    - Set up daily alert runs (vs. current weekly)
    - Add threshold alerts for zero bookings >3 days
    - Create executive dashboard showing portfolio health
    - **Assign to:** Data/analytics team

5. **Conduct Full Portfolio Audit**

    - Review all 99 CRITICAL listings for common factors
    - Check for geographic clustering (market-specific issues?)
    - Identify if specific property types more affected
    - **Assign to:** Operations + revenue management

6. **Enhance Alert Logic**

    - Add "successive weeks underperforming" flag
    - Incorporate review score changes

- Add competitor pricing context
- **Assign to:** AI innovation specialist (follow-up)

Long-Term Initiatives (Within 1 Month)

7. **Build Predictive Performance Model**

    - ML model to forecast bookings based on search/view patterns
    - Earlier detection of issues (predict next week's problems)
    - Anomaly detection for unusual patterns
    - **Assign to:** Data science team

8. **Create Automated Response Playbooks**

    - Auto-trigger rate adjustments for underperforming listings
    - Automated A/B testing of listing content
    - Integration with dynamic pricing tools
    - **Assign to:** Revenue systems team

9. **Expand to Multi-OTA Monitoring**

    - Integrate Expedia, Booking.com performance data
    - Cross-platform performance comparison
    - Identify if issues are Airbnb-specific or systemic
    - **Assign to:** Distribution + data engineering

Listing Prioritization Strategy

How to prioritize 150+ alerts:

1. **Severity Score (Primary):** Higher scores = investigate first
2. **Historical Revenue (Secondary):** Focus on traditionally high-performers
3. **Search Volume (Tertiary):** High appearances + zero bookings = bigger revenue loss
4. **Market Timing (Context):** High-season markets get priority

Recommended Daily Workflow:

- **Morning:** Review yesterday's new CRITICAL alerts
- **Mid-day:** Work through top 10 HIGH alerts
- **Weekly:** Bulk review MEDIUM alerts for patterns
- **Monthly:** Analyze LOW alerts for early warning signals

How I Used AI in This Project

Claude for Code development and debugging

Pair Programming: Throughout the code development, I used Claude (Anthropic) as an AI pair programmer. Specific examples: (1) Designing the SQLAlchemy database schema with proper indexing and constraints, (2) Debugging pandas operations for multi-sheet Excel parsing, (3) Crafting the cumulative scoring algorithm for alert severity, (4) Implementing exponential backoff retry logic for API resilience, (5) Building Slack Block Kit JSON for rich message formatting.

Code Review: I asked Claude to review critical sections for: edge case handling (division by zero, empty DataFrames), SQL injection vulnerabilities (using parameterized queries), memory efficiency (streaming large Excel files), and error message clarity (helpful for debugging during n8n execution).

Claude API for Production Insights

Strategic Analysis: The automation system uses Claude API (Sonnet 4) to analyze alert patterns. Claude receives: (1) Overall alert statistics (150 total, breakdown by severity), (2) Top 5 most severe issues with full metrics, (3) Structured prompt requesting executive summary + root causes + action items.

Why This Adds Value: Without AI, the alert would simply list 150 issues with no synthesis. Revenue managers would need to manually identify patterns (e.g., "zero bookings is a portfolio-wide problem, not individual listing issue"). Claude performs this pattern recognition automatically, surfacing systematic issues that would otherwise be obscured by individual alert noise. The root cause hypotheses (pricing misalignment, booking system malfunction, content deterioration) are non-obvious and require business context that pure rule-based systems can't provide.

AI for Documentation

README Generation: I used Claude to help structure and polish the README.md, ensuring comprehensive coverage of: installation steps, configuration options, usage examples, deployment strategies.

AI Limitations & Human Judgment

What AI Could Not Do:

- Business Logic Design: The alert rules (what constitutes CRITICAL vs HIGH) required domain expertise. AI suggested patterns but couldn't determine "50 appearances" as the threshold for high-visibility alerts.
- Architectural Decisions: Choosing n8n over Airflow, MySQL over PostgreSQL, or cumulative scoring over classification—these required weighing tradeoffs AI couldn't evaluate.
- Error Debugging: When pandas operations failed on edge cases (all-zero columns), I had to manually trace execution and understand the underlying data issue.
- Prompt Engineering: Crafting the Claude API prompt took 5+ iterations to get consistent, actionable output. AI couldn't self-optimize its own prompting.
- Stakeholder Communication: Deciding how to present findings (Slack vs email, daily vs weekly, top 10 vs all alerts) required understanding organizational context.

Code Quality & Engineering Practices

Modular Architecture

The codebase follows single-responsibility principle with clear separation:

- database.py: Pure data access layer (no business logic)
- etl.py: Stateless transformation functions (no database dependencies except loading)
- analysis.py: Alert detection logic isolated from I/O operations
- ai_insights.py: API client with retry logic, no knowledge of database schema
- slack_notifier.py: Message formatting separate from business logic
- config.py: Centralized configuration, single source of truth for thresholds

Benefits: This modularity enables: (1) Unit testing each component in isolation, (2) Replacing Slack with email without touching analysis code, (3) Swapping MySQL for PostgreSQL by only changing database.py, (4) Adjusting alert rules without understanding ETL logic.

Error Handling & Resilience

Database Operations: All database insertions use try-except blocks with specific error handling: Duplicate key errors log warnings but don't crash. Connection failures trigger immediate alerts via logging. Transactions use context managers ensuring proper cleanup even on exceptions.

Data Validation: ETL pipeline validates: (1) Non-empty DataFrames before processing, (2) Numeric types for calculations (coerce errors to NaN), (3) Date parsing with fallback to current date if sheet name format invalid, (4) Division by zero protection (replace 0 denominators with 1).

Configuration Management

Environment-Based Config: All credentials and thresholds live in .env file (git-ignored). config.py validates required variables on startup, failing fast with clear error messages. This enables: (1) Dev/staging/prod environments with different databases, (2) Easy threshold tuning without code changes, (3) Secure credential storage (never committed to git).

Sensible Defaults: Config class provides defaults for all non-secret values: CRITICAL_THRESHOLD=100, HISTORICAL_WEEKS=4, etc. This makes initial setup easier (only secrets required in .env) while allowing full customization.

Logging & Observability

Comprehensive logging at INFO level provides: (1) Pipeline progress ("Running ETL...", "Analyzing 150 listings"), (2) Alert summary ("Found 99 HIGH alerts"), (3) API call status ("Retrying Claude API in 4s"), (4) Error details with stack traces. Logs write to stdout for n8n capture and future log aggregation (Datadog, CloudWatch).

Next Steps & Future Enhancements

Current Deployment (Proof of Concept)

Local Execution: The system currently runs on my MacBook Pro: Python 3.14 in venv, MySQL 9.5 via Homebrew, n8n self-hosted on localhost:5678. This setup demonstrates full functionality but isn't production-ready (single point of failure, no redundancy, manual Excel file updates). Codebase accessible in :https://github.com/neelagarwal98/ota-alert-system-n8n-automation

Expandable enhancements

- Web **Dashboard**: Streamlit or Flask app showing: (1) Current active alerts table (sortable/filterable), (2) Historical alert trends chart, (3) Listing performance over time (line charts), (4) Alert resolution rate KPI.
- Instead of slack, the alert notification can go to a custom built application. This application being interactive with a **chatbot**, can integrate simple **text-to-SQ**L and vice-versa to interact with the database for further addressing the alerts.
- Taking the above a step further, instead of just querying, **Agentic Workflow** can be integrated to pick up the alert notifications and then decide to call relevant tools to address the given alerts accordingly without human involvement.

This existing system demonstrates that thoughtful automation—combining data engineering, machine learning, and workflow orchestration—can deliver immediate business value while laying groundwork for advanced capabilities and on-the-go implementation. The modular architecture and comprehensive documentation ensure the system is maintainable, extensible, and production-ready for Kasa Living's operations team.

— Neel Agarwal

November 2025