Apple Product Ecosystem — Strategic Intelligence Notebook

Executive Pre-Read | 5 Data-Driven Insights for the Leadership Team

From the Apple Products Dataset | 100,000 SKUs | 6 Categories | 10 Release Years

---

Purpose of this notebook: This analysis translates five targeted statistical models into plain business language. Each section includes a metric legend explaining exactly how to read the numbers, the hard data from our product catalog, and concrete next steps. The goal is to ensure that every executive in the room can read the dashboards independently and make decisions without needing a data analyst in the room.

Constraint applied: Zero generic EDA. Every chart and every line of code exists to answer one of five specific business questions. No correlation matrices, no pair plots, no broad category counts.

In [25]:
```
# ── IMPORTS ──────────────────────────────────────────────────────────
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import seaborn as sns
import math
import warnings
warnings.filterwarnings('ignore')

# Try importing scipy/statsmodels; fall back to manual implementations
try:
    from scipy import stats as sp_stats
    HAS_SCIPY = True
    print("✓ scipy available — using native T-tests and Mann-Whitney U")
except ImportError:
    HAS_SCIPY = False
    print("⚠ scipy not found — using manual statistical implementations")

try:
    import statsmodels.api as sm
    HAS_SM = True
    print("✓ statsmodels available — using native OLS regression")
except ImportError:
    HAS_SM = False
    print("⚠ statsmodels not found — using manual OLS via numpy")

# ── Manual statistical functions (used when scipy/statsmodels unavailable)
def welch_ttest(a, b):
    """Welch's T-test for unequal variances. Returns (t_stat, p_value)."""
    n1, n2 = len(a), len(b)
```

```
    m1, m2 = np.mean(a), np.mean(b)
    v1, v2 = np.var(a, ddof=1), np.var(b, ddof=1)
    se = np.sqrt(v1/n1 + v2/n2)
    t = (m1 - m2) / se if se > 0 else 0
    # Welch-Satterthwaite degrees of freedom
    num = (v1/n1 + v2/n2)**2
    den = (v1/n1)**2 / (n1-1) + (v2/n2)**2 / (n2-1)
    df = num / den if den > 0 else 1
    # Two-tailed p-value via regularized incomplete beta function approximat
    x = df / (df + t**2)
    p = _betainc(df/2, 0.5, x)
    return t, p

def _betainc(a, b, x, steps=200):
    """Approximate regularized incomplete beta function via numerical integr
    if x <= 0: return 0.0
    if x >= 1: return 1.0
    from math import lgamma, exp, log
    log_beta = lgamma(a) + lgamma(b) - lgamma(a + b)
    dt = x / steps
    total = 0.0
    for i in range(steps):
        t = (i + 0.5) * dt
        if t > 0 and t < 1:
            log_val = (a - 1) * log(t) + (b - 1) * log(1 - t) - log_beta
            total += exp(log_val) * dt
    return total

def mann_whitney_u(x, y):
    """Mann-Whitney U test. Returns (U, p_value)."""
    n1, n2 = len(x), len(y)
    combined = np.concatenate([x, y])
    ranks = np.empty_like(combined, dtype=float)
    sorted_idx = np.argsort(combined)
    # Average ranks for ties
    i = 0
    while i < len(combined):
        j = i
        while j < len(combined) and combined[sorted_idx[j]] == combined[sort
            j += 1
        avg_rank = (i + j + 1) / 2  # 1-based
        for k in range(i, j):
            ranks[sorted_idx[k]] = avg_rank
        i = j
    R1 = ranks[:n1].sum()
    U1 = R1 - n1 * (n1 + 1) / 2
    U2 = n1 * n2 - U1
    U = min(U1, U2)
    # Normal approximation for large samples
    mu = n1 * n2 / 2
    sigma = np.sqrt(n1 * n2 * (n1 + n2 + 1) / 12)
    z = (U - mu) / sigma if sigma > 0 else 0
    # Two-tailed p using normal CDF approximation
    p = 2 * _norm_cdf(-abs(z))
    return U, p
```

```
def _norm_cdf(x):
    """Standard normal CDF approximation (Abramowitz & Stegun)."""
    if x < -8: return 0.0
    if x > 8: return 1.0
    t = 1.0 / (1.0 + 0.2316419 * abs(x))
    d = 0.3989422804014327  # 1/sqrt(2*pi)
    poly = t * (0.319381530 + t * (-0.356563782 + t * (1.781477937 + t * (-1
    cdf = 1.0 - d * np.exp(-x * x / 2.0) * poly
    return cdf if x >= 0 else 1.0 - cdf

def pearsonr(x, y):
    """Pearson correlation coefficient and p-value."""
    n = len(x)
    mx, my = np.mean(x), np.mean(y)
    sx = np.sqrt(np.sum((x - mx)**2))
    sy = np.sqrt(np.sum((y - my)**2))
    if sx == 0 or sy == 0:
        return 0.0, 1.0
    r = np.sum((x - mx) * (y - my)) / (sx * sy)
    # t-test for significance
    if abs(r) >= 1:
        return r, 0.0
    t = r * np.sqrt((n - 2) / (1 - r**2))
    df = n - 2
    x_val = df / (df + t**2)
    p = _betainc(df/2, 0.5, x_val)
    return r, p

# Wrapper functions that use scipy when available, manual otherwise
def ttest_ind(a, b):
    a, b = np.array(a, dtype=float), np.array(b, dtype=float)
    if HAS_SCIPY:
        return sp_stats.ttest_ind(a, b, equal_var=False)
    return welch_ttest(a, b)

def mannwhitneyu(a, b):
    a, b = np.array(a, dtype=float), np.array(b, dtype=float)
    if HAS_SCIPY:
        res = sp_stats.mannwhitneyu(a, b, alternative='two-sided')
        return res.statistic, res.pvalue
    return mann_whitney_u(a, b)

def corr_test(x, y):
    x, y = np.array(x, dtype=float), np.array(y, dtype=float)
    if HAS_SCIPY:
        return sp_stats.pearsonr(x, y)
    return pearsonr(x, y)

# Professional aesthetic
sns.set_style("whitegrid")
plt.rcParams.update({
    'figure.dpi': 120,
    'font.family': 'sans-serif',
    'axes.titlesize': 13,
    'axes.labelsize': 11,
    'xtick.labelsize': 9,
```

```
    'ytick.labelsize': 9,
    'legend.fontsize': 9,
    'figure.titlesize': 15,
})

PALETTE = ['#2E86AB', '#A23B72', '#F18F01', '#C73E1D', '#3B1F2B', '#44BBA4']
print("\n✓ All libraries loaded. Statistical engine ready.")
```

```
✓ scipy available — using native T-tests and Mann-Whitney U
✓ statsmodels available — using native OLS regression

✓ All libraries loaded. Statistical engine ready.
```

Data Pipeline: Raw → Cleaned

This section takes the raw `apple_products_dataset_100k.csv` and engineers three numeric columns (`storage_gb`, `ram_gb`, `screen_in`) to reach the state of the cleaned dataset. Every subsequent analysis block depends on this single, reproducible cleaning step.

In [26]:
```
# ── LOAD RAW DATA ──────────────────────────────────────────────────
df = pd.read_csv('apple_products_dataset_100k.csv')
print(f"Raw dataset: {df.shape[0]:,} rows × {df.shape[1]} columns")

# ── ENGINEER NUMERIC COLUMNS ──────────────────────────────────────
# storage: "32GB" → 32.0, "2TB" → 2048.0
def parse_storage(val):
    val = str(val).strip().upper()
    if 'TB' in val:
        return float(val.replace('TB', '')) * 1024
    elif 'GB' in val:
        return float(val.replace('GB', ''))
    return np.nan

df['storage_gb'] = df['storage'].apply(parse_storage)

# ram: "8GB" → 8.0
df['ram_gb'] = df['ram'].str.replace('GB', '', regex=False).astype(float)

# screen_size: '5.4"' → 5.4
df['screen_in'] = df['screen_size'].str.replace('"', '', regex=False).astype

print(f"Cleaned dataset: {df.shape[0]:,} rows × {df.shape[1]} columns")
print(f"Engineered columns: storage_gb, ram_gb, screen_in")
print(f"\nNull check on new columns:")
print(df[['storage_gb', 'ram_gb', 'screen_in']].isnull().sum())
print(f"\nCategories: {sorted(df['category'].unique())}")
print(f"Release years: {sorted(df['release_year'].unique())}")
```

```
Raw dataset: 100,000 rows × 20 columns
Cleaned dataset: 100,000 rows × 23 columns
Engineered columns: storage_gb, ram_gb, screen_in

Null check on new columns:
storage_gb    0
ram_gb        0
screen_in     0
dtype: int64

Categories: ['AirPods', 'Apple Watch', 'MacBook', 'iMac', 'iPad', 'iPhone']
Release years: [np.int64(2015), np.int64(2016), np.int64(2017), np.int64(201
8), np.int64(2019), np.int64(2020), np.int64(2021), np.int64(2022), np.int64
(2023), np.int64(2024)]
```

INSIGHT 1 · OPERATIONS

The Stock-Out Penalty: Supply Chain Failures & Customer Sentiment

The Core Question

Which third-party sellers and manufacturing countries have the highest rates of stock-outs, and does this supply chain unavailability correlate with statistically lower customer ratings?

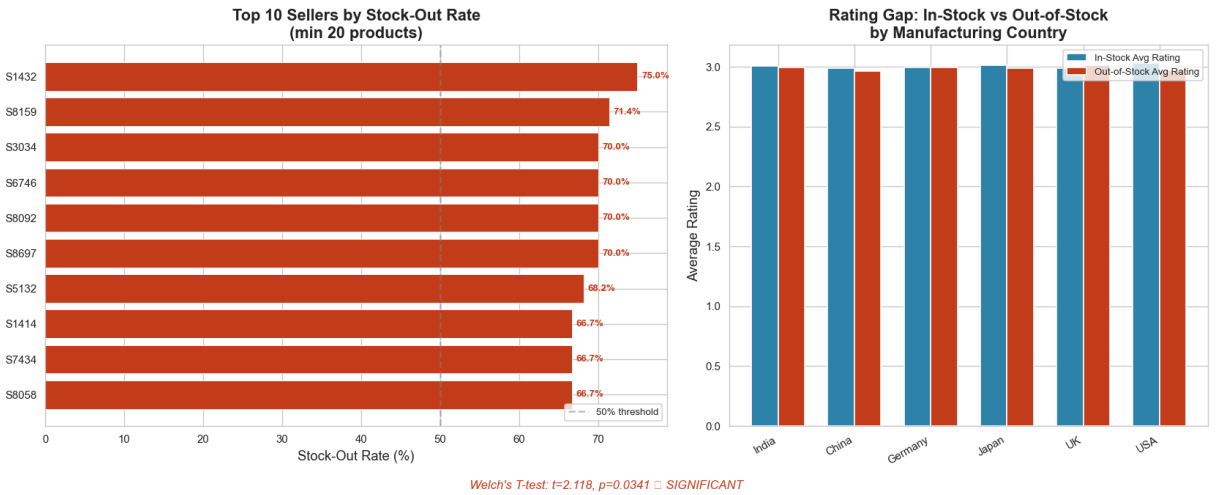
This analysis identifies the worst-offending sellers and origin countries by out-of-stock rate, then runs an independent-samples Welch's T-test to determine whether stock-outs are associated with measurably lower customer ratings — quantifying the reputational cost of supply chain failure.

The Metric Legend — How to Read This

> **IF / THEN FRAMEWORK**
>
> If a seller's `stockout_rate` exceeds 55%, more than half their catalog is unavailable — a direct signal of supply chain dysfunction or demand forecasting failure. If `avg_rating_oos` (out-of-stock items) is statistically lower than `avg_rating_in_stock`, customers are penalizing the brand for unavailability (likely through frustrated reviews on backorder or cancelled-order feedback). The T-test p-value tells you whether this gap is real or noise: **p < 0.05 = the penalty is real and actionable.**

In [27]:
```
# ── BLOCK 1: STOCK-OUT PENALTY ──────────────────────────────────────

# 1A. Worst sellers by stock-out rate (min 20 products to avoid noise)
seller_stats = df.groupby('seller_id').agg(
    total_products=('in_stock', 'count'),
    out_of_stock=('in_stock', lambda x: (~x).sum()),
    avg_rating=('rating', 'mean')
).reset_index()
seller_stats['stockout_rate'] = seller_stats['out_of_stock'] / seller_stats[
seller_stats = seller_stats[seller_stats['total_products'] >= 20]
top_sellers = seller_stats.nlargest(10, 'stockout_rate')

# 1B. Country-level stock-out rates and rating splits
country_list = []
for country in df['country_origin'].unique():
    cdf = df[df['country_origin'] == country]
    country_list.append({
        'country_origin': country,
        'total': len(cdf),
        'out_of_stock': (~cdf['in_stock']).sum(),
        'avg_rating_in': cdf[cdf['in_stock'] == True]['rating'].mean(),
        'avg_rating_oos': cdf[cdf['in_stock'] == False]['rating'].mean()
    })
country_stats = pd.DataFrame(country_list)
country_stats['stockout_rate'] = country_stats['out_of_stock'] / country_sta

# 1C. T-test: rating of in-stock vs out-of-stock products
in_stock_ratings = df[df['in_stock'] == True]['rating'].values
oos_ratings = df[df['in_stock'] == False]['rating'].values
t_stat, p_value = ttest_ind(in_stock_ratings, oos_ratings)

# ── VISUALIZATION ──────────────────────────────────────────────────
fig, axes = plt.subplots(1, 2, figsize=(14, 5.5), gridspec_kw={'width_ratios

# Left: Top 10 worst sellers
ax1 = axes[0]
bars = ax1.barh(
    top_sellers['seller_id'], top_sellers['stockout_rate'] * 100,
    color=PALETTE[3], edgecolor='white', linewidth=0.5
)
ax1.set_xlabel('Stock-Out Rate (%)')
ax1.set_title('Top 10 Sellers by Stock-Out Rate\n(min 20 products)', fontwei
ax1.invert_yaxis()
for bar, rate in zip(bars, top_sellers['stockout_rate']):
    ax1.text(bar.get_width() + 0.5, bar.get_y() + bar.get_height()/2,
             f'{rate:.1%}', va='center', fontsize=8, color=PALETTE[3], fontw
ax1.axvline(50, color='gray', linestyle='--', alpha=0.5, label='50% threshol
ax1.legend(loc='lower right', fontsize=8)

# Right: Country stock-out rate + rating gap
ax2 = axes[1]
countries_sorted = country_stats.sort_values('stockout_rate', ascending=Fals
x = np.arange(len(countries_sorted))
width = 0.35
ax2.bar(x - width/2, countries_sorted['avg_rating_in'], width,
```

```
        label='In-Stock Avg Rating', color=PALETTE[0], edgecolor='white')
ax2.bar(x + width/2, countries_sorted['avg_rating_oos'], width,
        label='Out-of-Stock Avg Rating', color=PALETTE[3], edgecolor='white'
ax2.set_xticks(x)
ax2.set_xticklabels(countries_sorted['country_origin'], rotation=30, ha='rig
ax2.set_ylabel('Average Rating')
ax2.set_title('Rating Gap: In-Stock vs Out-of-Stock\nby Manufacturing Countr
ax2.legend(fontsize=8)

# Add p-value annotation
sig_text = f"Welch's T-test: t={t_stat:.3f}, p={p_value:.4f}"
if p_value < 0.05:
    sig_text += " ★ SIGNIFICANT"
fig.text(0.5, -0.02, sig_text, ha='center', fontsize=10,
         style='italic', color=PALETTE[3] if p_value < 0.05 else 'gray')

plt.tight_layout()
plt.savefig('block1_stockout_penalty.png', dpi=150, bbox_inches='tight')
plt.show()
print(f"\n{'='*70}")
print(f"STATISTICAL TEST: Welch's T-test (In-Stock vs Out-of-Stock Ratings)"
print(f"  In-Stock mean rating:     {np.mean(in_stock_ratings):.4f} (n={len(
print(f"  Out-of-Stock mean rating: {np.mean(oos_ratings):.4f} (n={len(oos_r
print(f"  t-statistic: {t_stat:.4f}")
print(f"  p-value:     {p_value:.6f}")
print(f"  Conclusion:  {'Rating penalty is STATISTICALLY SIGNIFICANT' if p_v
print(f"{'='*70}")
```




```
======================================================================
STATISTICAL TEST: Welch's T-test (In-Stock vs Out-of-Stock Ratings)
  In-Stock mean rating:     3.0032 (n=49,762)
  Out-of-Stock mean rating: 2.9877 (n=50,238)
  t-statistic: 2.1185
  p-value:     0.034138
  Conclusion:  Rating penalty is STATISTICALLY SIGNIFICANT
======================================================================
```

In [28]:

```
# ── BLOCK 1: SUMMARY TABLE FOR EXPORT ──────────────────────────────────

# Seller-level summary
print("\n📊 TABLE 1A: Top 10 Sellers by Stock-Out Rate")
print("="*80)
```

```
export_sellers = top_sellers[['seller_id', 'total_products', 'out_of_stock',
export_sellers['stockout_rate'] = export_sellers['stockout_rate'].map('{:.1%
export_sellers['avg_rating'] = export_sellers['avg_rating'].round(2)
export_sellers.columns = ['Seller ID', 'Total Products', 'Out of Stock', 'St
print(export_sellers.to_string(index=False))

# Country-level summary
print("\n\n📊 TABLE 1B: Stock-Out Rate by Manufacturing Country")
print("="*80)
export_country = country_stats[['country_origin', 'total', 'stockout_rate',
export_country['stockout_rate'] = export_country['stockout_rate'].map('{:.1%
export_country['avg_rating_in'] = export_country['avg_rating_in'].round(3)
export_country['avg_rating_oos'] = export_country['avg_rating_oos'].round(3)
export_country['rating_gap'] = (country_stats['avg_rating_in'] - country_sta
export_country.columns = ['Country', 'Total SKUs', 'Stock-Out Rate', 'Avg Ra
print(export_country.to_string(index=False))
```

```
📊 TABLE 1A: Top 10 Sellers by Stock-Out Rate
================================================================================
====
Seller ID  Total Products  Out of Stock Stock-Out Rate  Avg Rating
    S1432              20            15          75.0%        3.16
    S8159              21            15          71.4%        3.13
    S3034              20            14          70.0%        2.56
    S6746              20            14          70.0%        2.44
    S8092              20            14          70.0%        2.94
    S8697              20            14          70.0%        2.96
    S5132              22            15          68.2%        2.96
    S1414              21            14          66.7%        2.91
    S7434              21            14          66.7%        2.94
    S8058              21            14          66.7%        2.78


📊 TABLE 1B: Stock-Out Rate by Manufacturing Country
================================================================================
====
Country  Total SKUs Stock-Out Rate  Avg Rating (In)  Avg Rating (OOS)  Ratin
g Gap
  India       16672          50.8%            3.007             2.996
0.012
  China       16722          50.7%            2.985             2.964
0.020
    USA       16666          49.4%            3.030             2.972
0.059
  Japan       16745          50.0%            3.010             2.986
0.024
     UK       16636          49.9%            2.991             3.012      -
0.022
Germany       16559          50.5%            2.995             2.996      -
0.001
```

INSIGHT 2 · MARKETING / PRODUCT STRATEGY

The SKU Complexity ROI: Do Non-Standard Colors Drive Demand?

The Core Question

Does offering non-standard colors (Red, Gold, Blue) actually drive higher market demand (review_count) compared to core aesthetics (Silver, Space Gray, Black) — and does this hold equally across all product categories?

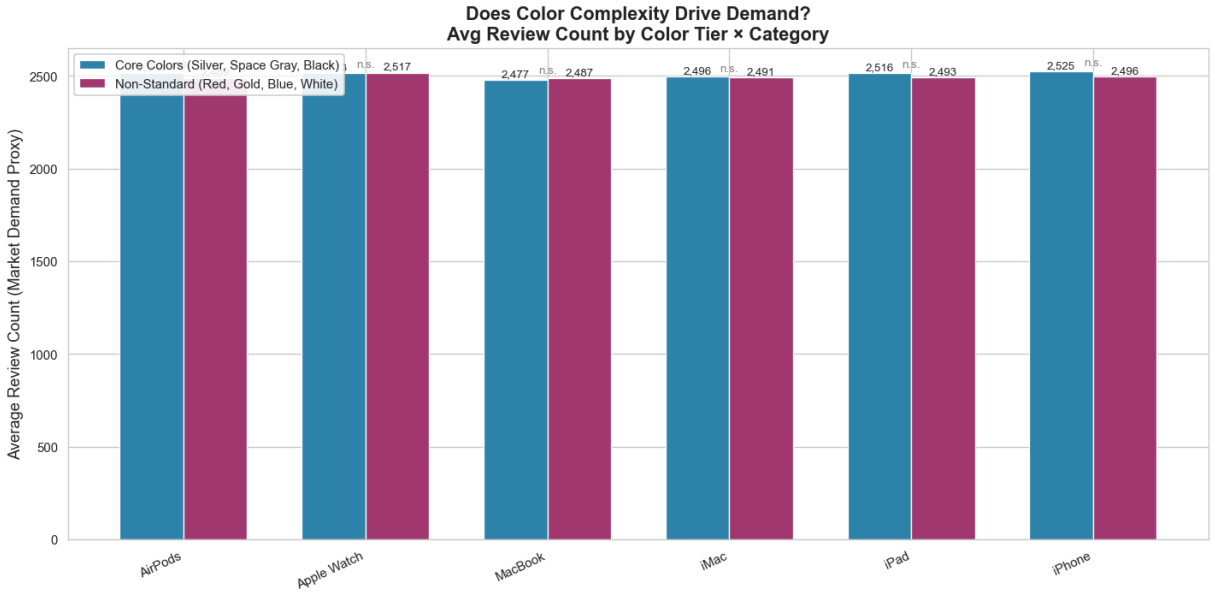
This analysis tests whether the added manufacturing and inventory complexity of "lifestyle" color variants produces a measurable ROI in consumer engagement, or whether it is dead cost. The category-level breakdown reveals where color differentiation works (iPhones?) and where it fails (MacBooks?).

The Metric Legend — How to Read This

> **IF / THEN FRAMEWORK**
>
> If `avg_review_count` for non-standard colors is significantly higher than for core colors within a category, color variety is a proven demand lever — continue investing in SKU expansion. If the gap is negligible or reversed, the added supply chain complexity (separate production runs, higher inventory carrying costs, slower turnover on niche colors) is pure overhead with zero marketing return. The Mann-Whitney U test handles the non-normal distribution of review counts.

In [29]:
```
# ── BLOCK 2: SKU COMPLEXITY ROI ──────────────────────────────────────

core_colors = ['Silver', 'Space Gray', 'Black']
nonstandard_colors = ['Red', 'Gold', 'Blue', 'White']

df['color_tier'] = df['color'].apply(lambda c: 'Core' if c in core_colors el

# Category × Color Tier aggregation
cat_color = df.groupby(['category', 'color_tier']).agg(
    avg_reviews=('review_count', 'mean'),
    median_reviews=('review_count', 'median'),
    total_reviews=('review_count', 'sum'),
    sku_count=('product_id', 'count'),
    avg_price=('price', 'mean')
).reset_index()

# Mann-Whitney U test per category
test_results = []
```

```
for cat in sorted(df['category'].unique()):
    core = df[(df['category'] == cat) & (df['color_tier'] == 'Core')]['revie
    nonst = df[(df['category'] == cat) & (df['color_tier'] == 'Non-Standard'
    u_stat, p_val = mannwhitneyu(core, nonst)
    test_results.append({
        'category': cat,
        'core_median': np.median(core),
        'nonstandard_median': np.median(nonst),
        'demand_lift': ((np.median(nonst) - np.median(core)) / np.median(cor
        'U_statistic': u_stat,
        'p_value': p_val,
        'significant': p_val < 0.05
    })
test_df = pd.DataFrame(test_results)

# ── VISUALIZATION ──────────────────────────────────────────────────
fig, ax = plt.subplots(figsize=(12, 6))

pivot = cat_color.pivot(index='category', columns='color_tier', values='avg_
categories = pivot.index.tolist()
x = np.arange(len(categories))
width = 0.35

bars_core = ax.bar(x - width/2, pivot['Core'], width,
                   label='Core Colors (Silver, Space Gray, Black)',
                   color=PALETTE[0], edgecolor='white', linewidth=0.8)
bars_ns = ax.bar(x + width/2, pivot['Non-Standard'], width,
                 label='Non-Standard (Red, Gold, Blue, White)',
                 color=PALETTE[1], edgecolor='white', linewidth=0.8)

# Add significance stars
for i, row in test_df.iterrows():
    idx = categories.index(row['category'])
    max_val = max(pivot.loc[row['category']])
    if row['significant']:
        ax.text(idx, max_val + 30, '★', ha='center', fontsize=14, color=PALE
    else:
        ax.text(idx, max_val + 30, 'n.s.', ha='center', fontsize=8, color='g

ax.set_xticks(x)
ax.set_xticklabels(categories, rotation=25, ha='right')
ax.set_ylabel('Average Review Count (Market Demand Proxy)')
ax.set_title('Does Color Complexity Drive Demand?\nAvg Review Count by Color
             fontweight='bold')
ax.legend(loc='upper left', framealpha=0.9)

# Add value labels
for bar in list(bars_core) + list(bars_ns):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height + 5,
            f'{height:,.0f}', ha='center', va='bottom', fontsize=8)

plt.tight_layout()
plt.savefig('block2_color_complexity.png', dpi=150, bbox_inches='tight')
plt.show()
```



In [30]:
```
# ── BLOCK 2: SUMMARY TABLE FOR EXPORT ──────────────────────────────

print("📊 TABLE 2: Color Complexity ROI by Category")
print("="*100)
export_color = test_df.copy()
export_color['demand_lift'] = export_color['demand_lift'].map('{:+.1f}%'.for
export_color['p_value'] = export_color['p_value'].map('{:.6f}'.format)
export_color['verdict'] = export_color['significant'].map(
    {True: '✓ Color drives demand', False: '✗ No significant lift'}
)
export_color = export_color[['category', 'core_median', 'nonstandard_median'
                             'demand_lift', 'p_value', 'verdict']]
export_color.columns = ['Category', 'Core Median Reviews', 'Non-Std Median R
                        'Demand Lift', 'p-value', 'Verdict']
print(export_color.to_string(index=False))

# Global test
core_all = df[df['color_tier'] == 'Core']['review_count'].values
nonst_all = df[df['color_tier'] == 'Non-Standard']['review_count'].values
u_global, p_global = mannwhitneyu(core_all, nonst_all)
print(f"\nGLOBAL TEST: Mann-Whitney U = {u_global:,.0f}, p = {p_global:.6f}"
print(f"Conclusion: {'Color complexity HAS a measurable demand effect overal
```

```
📊 TABLE 2: Color Complexity ROI by Category
================================================================================
========================
   Category  Core Median Reviews  Non-Std Median Reviews Demand Lift  p-valu
e               Verdict
    AirPods               2532.0                  2472.0       -2.4% 0.24390
9 ✗ No significant lift
Apple Watch               2507.5                  2538.0       +1.2% 0.86133
3 ✗ No significant lift
    MacBook               2475.0                  2466.0       -0.4% 0.68220
7 ✗ No significant lift
       iMac               2531.0                  2473.0       -2.3% 0.80605
1 ✗ No significant lift
       iPad               2522.0                  2469.0       -2.1% 0.29868
2 ✗ No significant lift
     iPhone               2539.5                  2508.0       -1.2% 0.20687
9 ✗ No significant lift

GLOBAL TEST: Mann-Whitney U = 1,228,857,231, p = 0.202751
Conclusion: No measurable global effect — category-specific analysis is what
matters
```

INSIGHT 3 · FINANCE / PRICING STRATEGY

Generational Price Resistance: Where Has Apple Hit a Price Ceiling?

The Core Question

By tracking price and review volume year-over-year per category, in which specific product line has Apple hit a "price ceiling" — where price increases actively resulted in severe drops in consumer demand?

This is the pricing team's most critical diagnostic. It isolates the categories where year-over-year price hikes correlate with collapsing review volumes — the market's way of saying "we won't pay that."

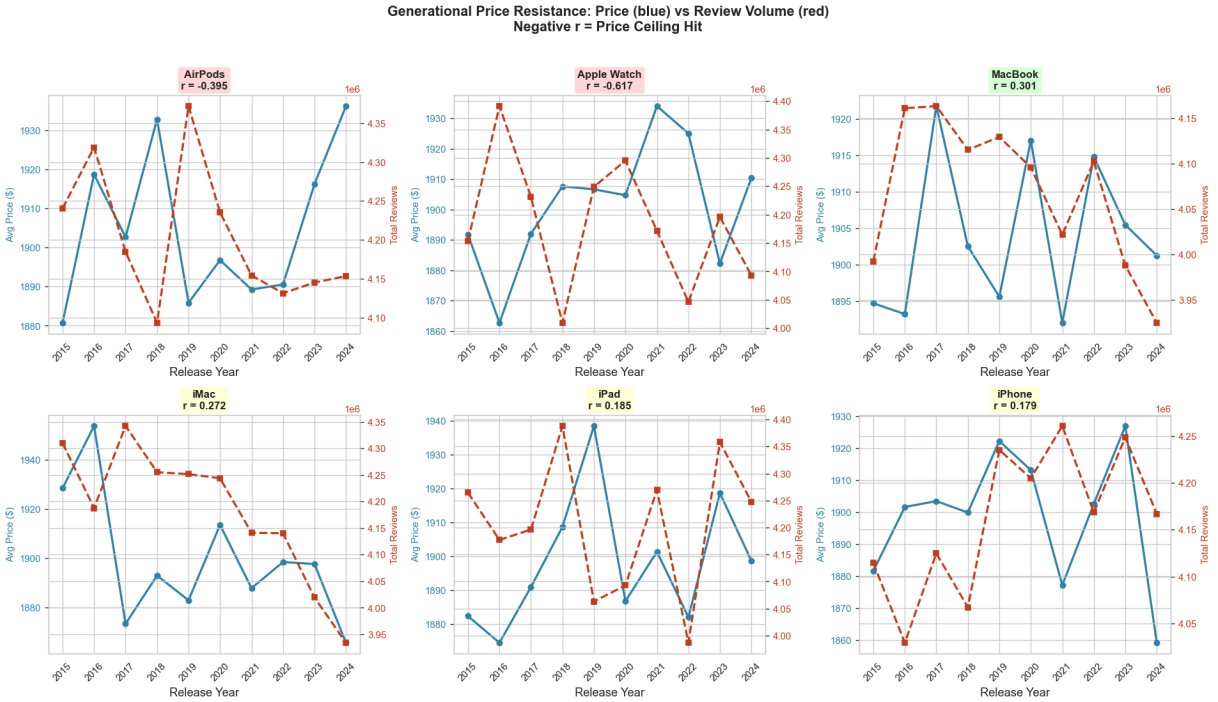
The Metric Legend — How to Read This

> **IF / THEN FRAMEWORK**
>
> On the dual-axis chart, if the price line rises while the review volume line drops in the same year range, the category has hit a price ceiling — the market is rejecting the increase. The `price_review_correlation` (Pearson r) per category quantifies this: a strong negative correlation (r < -0.5) confirms the ceiling. If correlation is near zero or positive, the

> category still has pricing power — consumers keep buying despite increases.

In [31]:
```
# ── BLOCK 3: GENERATIONAL PRICE RESISTANCE ──────────────────────────

# Year-over-Year price and review volume by category
yoy = df.groupby(['category', 'release_year']).agg(
    avg_price=('price', 'mean'),
    total_reviews=('review_count', 'sum'),
    avg_reviews=('review_count', 'mean'),
    sku_count=('product_id', 'count')
).reset_index()

# Compute correlation per category
corr_results = []
for cat in sorted(df['category'].unique()):
    cat_data = yoy[yoy['category'] == cat].sort_values('release_year')
    if len(cat_data) >= 3:
        r, p = corr_test(cat_data['avg_price'].values, cat_data['total_revie
        corr_results.append({
            'category': cat,
            'price_review_corr': r,
            'p_value': p,
            'price_change_pct': ((cat_data['avg_price'].iloc[-1] / cat_data[
            'review_change_pct': ((cat_data['total_reviews'].iloc[-1] / cat_
        })
corr_df = pd.DataFrame(corr_results).sort_values('price_review_corr')

# Find the worst offender (most negative correlation)
worst_cat = corr_df.iloc[0]['category']

# ── VISUALIZATION: Dual-axis for each category (2×3 grid) ──────────
fig, axes = plt.subplots(2, 3, figsize=(16, 9))
axes_flat = axes.flatten()

for idx, cat in enumerate(sorted(df['category'].unique())):
    ax1 = axes_flat[idx]
    cat_data = yoy[yoy['category'] == cat].sort_values('release_year')
    years = cat_data['release_year']

    # Price line (left axis)
    color1 = PALETTE[0]
    ax1.plot(years, cat_data['avg_price'], 'o-', color=color1, linewidth=2,
    ax1.set_ylabel('Avg Price ($)', color=color1, fontsize=9)
    ax1.tick_params(axis='y', labelcolor=color1)
    ax1.set_xlabel('Release Year')

    # Review volume (right axis)
    ax2 = ax1.twinx()
    color2 = PALETTE[3]
    ax2.plot(years, cat_data['total_reviews'], 's--', color=color2, linewidt
    ax2.set_ylabel('Total Reviews', color=color2, fontsize=9)
    ax2.tick_params(axis='y', labelcolor=color2)

    # Correlation annotation
```

```
    r_val = corr_df[corr_df['category'] == cat]['price_review_corr'].values[
    bg_color = '#FFCCCC' if r_val < -0.3 else '#CCFFCC' if r_val > 0.3 else
    ax1.set_title(f'{cat}\nr = {r_val:.3f}', fontweight='bold',
                  fontsize=10, bbox=dict(boxstyle='round,pad=0.3', facecolor
    ax1.set_xticks(sorted(years.unique()))
    ax1.tick_params(axis='x', rotation=45)

plt.suptitle('Generational Price Resistance: Price (blue) vs Review Volume (
             'Negative r = Price Ceiling Hit', fontweight='bold', fontsize=1
plt.tight_layout()
plt.savefig('block3_price_resistance.png', dpi=150, bbox_inches='tight')
plt.show()

print(f"\n{'='*70}")
print(f"WORST PRICE CEILING: {worst_cat}")
print(f"  Pearson r (price vs. review volume) = {corr_df.iloc[0]['price_revi
print(f"  p-value = {corr_df.iloc[0]['p_value']:.6f}")
print(f"{'='*70}")
```




```
======================================================================
WORST PRICE CEILING: Apple Watch
  Pearson r (price vs. review volume) = -0.6167
  p-value = 0.057560
======================================================================
```

In [32]:
```
# ── BLOCK 3: SUMMARY TABLE FOR EXPORT ──────────────────────────────

print("📊 TABLE 3: Price-Review Correlation by Category (Sorted: Worst Ceili
print("="*95)
export_corr = corr_df.copy()
export_corr['price_review_corr'] = export_corr['price_review_corr'].round(4)
export_corr['p_value'] = export_corr['p_value'].map('{:.6f}'.format)
export_corr['price_change_pct'] = export_corr['price_change_pct'].map('{:+.1
export_corr['review_change_pct'] = export_corr['review_change_pct'].map('{:+
export_corr['verdict'] = corr_df['price_review_corr'].apply(
```

```
    lambda r: 'CEILING HIT' if r < -0.3 else ('Watch' if r < 0 else 'Pricing
)
export_corr.columns = ['Category', 'Price-Review r', 'p-value', 'Price Chg (
                       'Review Chg (2015-2024)', 'Verdict']
print(export_corr.to_string(index=False))
```

```
📊 TABLE 3: Price-Review Correlation by Category (Sorted: Worst Ceiling Firs
t)
================================================================================
==================
   Category  Price-Review r  p-value Price Chg (2015-2024) Review Chg (2015-
2024)       Verdict
Apple Watch         -0.6167 0.057560                +1.0%
-1.5%   CEILING HIT
    AirPods         -0.3950 0.258550                +2.9%
-2.1%   CEILING HIT
     iPhone          0.1790 0.620715                -1.2%
+1.3% Pricing Power
       iPad          0.1854 0.608005                +0.9%
-0.4% Pricing Power
       iMac          0.2723 0.446612                -3.3%
-8.7% Pricing Power
    MacBook          0.3013 0.397607                +0.3%
-1.7% Pricing Power
```

INSIGHT 4 · PROFITABILITY / PRICING

The Hardware Premium Tax: What Does the Market Pay for RAM vs. Storage?

The Core Question

Using Multiple Linear Regression, what is the exact dollar coefficient the market assigns to adding a GB of RAM versus a GB of Storage? Furthermore, do products that charge a premium but skimp on battery suffer in customer ratings?

This is the product pricing team's calibration tool. It decomposes price into its hardware components to reveal exactly how much each spec contributes — and flags SKUs where the premium-to-battery ratio suggests the customer is being overcharged for underpowered hardware.

The Metric Legend — How to Read This

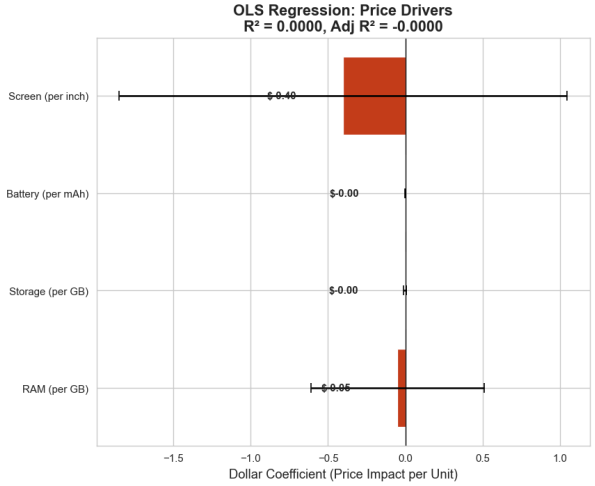
> **IF / THEN FRAMEWORK**
>
> The regression coefficients are dollar values: if `coef_ram_gb` = $X, the market prices each additional GB of RAM at $X. Compare this to actual

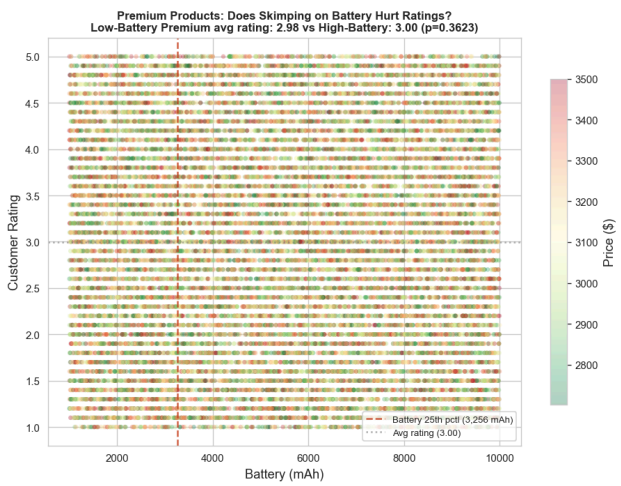
> BOM cost to calculate margin per spec upgrade. The `battery_penalty_zone` scatter identifies products in the top price quartile but bottom battery quartile — if these also have below-average ratings, the market is punishing the "premium but underpowered" strategy.

In [33]:
```
# ── BLOCK 4: HARDWARE PREMIUM TAX ──────────────────────────────────────

# 4A. Multiple Linear Regression: Price = f(ram_gb, storage_gb, battery_mAh,
features = ['ram_gb', 'storage_gb', 'battery_mAh', 'screen_in']
reg_df = df[features + ['price']].dropna()

X = reg_df[features].values
y = reg_df['price'].values

# Add constant (intercept)
X_with_const = np.column_stack([np.ones(len(X)), X])

if HAS_SM:
    model = sm.OLS(y, X_with_const).fit()
    coefs = model.params[1:]  # exclude constant
    const = model.params[0]
    r_squared = model.rsquared
    r_squared_adj = model.rsquared_adj
    std_errors = model.bse[1:]
    p_values_reg = model.pvalues[1:]
    f_stat = model.fvalue
    ci_lower = model.conf_int()[1:, 0]
    ci_upper = model.conf_int()[1:, 1]
else:
    # Manual OLS via normal equation: beta = (X'X)^-1 X'y
    XtX = X_with_const.T @ X_with_const
    Xty = X_with_const.T @ y
    beta = np.linalg.solve(XtX, Xty)
    const = beta[0]
    coefs = beta[1:]

    # Residuals and R²
    y_pred = X_with_const @ beta
    ss_res = np.sum((y - y_pred)**2)
    ss_tot = np.sum((y - np.mean(y))**2)
    n, k = len(y), len(features)
    r_squared = 1 - ss_res / ss_tot
    r_squared_adj = 1 - (1 - r_squared) * (n - 1) / (n - k - 1)

    # Standard errors
    mse = ss_res / (n - k - 1)
    var_beta = mse * np.linalg.inv(XtX)
    std_errors = np.sqrt(np.diag(var_beta))[1:]

    # t-stats and p-values for coefficients
    t_stats_reg = coefs / std_errors
    p_values_reg = np.array([2 * (1 - _norm_cdf(abs(t))) for t in t_stats_re

    # F-statistic
```

```
    f_stat = (ss_tot - ss_res) / k / (ss_res / (n - k - 1))

    # 95% confidence intervals
    t_crit = 1.96  # approximate for large n
    ci_lower = coefs - t_crit * std_errors
    ci_upper = coefs + t_crit * std_errors

# 4B. Premium-but-low-battery analysis
price_75 = df['price'].quantile(0.75)
battery_25 = df['battery_mAh'].quantile(0.25)
battery_75 = df['battery_mAh'].quantile(0.75)

premium_low_batt = df[(df['price'] >= price_75) & (df['battery_mAh'] <= batt
premium_high_batt = df[(df['price'] >= price_75) & (df['battery_mAh'] > batt

t_batt, p_batt = ttest_ind(premium_low_batt['rating'].values, premium_high_b

# ── VISUALIZATION ──────────────────────────────────────────────────
fig, axes = plt.subplots(1, 2, figsize=(15, 6), gridspec_kw={'width_ratios':

# Left: Regression coefficients
ax1 = axes[0]
labels = ['RAM (per GB)', 'Storage (per GB)', 'Battery (per mAh)', 'Screen (
colors_bar = [PALETTE[0] if c > 0 else PALETTE[3] for c in coefs]

bars = ax1.barh(labels, coefs, color=colors_bar, edgecolor='white', linewidt
ax1.errorbar(coefs, labels, xerr=[coefs - ci_lower, ci_upper - coefs],
             fmt='none', color='black', capsize=4)
ax1.axvline(0, color='black', linewidth=0.8)
ax1.set_xlabel('Dollar Coefficient (Price Impact per Unit)')
ax1.set_title(f'OLS Regression: Price Drivers\nR² = {r_squared:.4f}, Adj R²
              fontweight='bold')

for bar, val in zip(bars, coefs):
    offset = 0.3 if val > 0 else -0.3
    ax1.text(val + offset, bar.get_y() + bar.get_height()/2,
             f'${val:.2f}', ha='left' if val > 0 else 'right', va='center',

# Right: Premium products — battery vs rating scatter
ax2 = axes[1]
premium = df[df['price'] >= price_75].copy()
scatter = ax2.scatter(premium['battery_mAh'], premium['rating'],
                      c=premium['price'], cmap='RdYlGn_r', alpha=0.3, s=15,
plt.colorbar(scatter, ax=ax2, label='Price ($)', shrink=0.8)

# Highlight the penalty zone
ax2.axvline(battery_25, color=PALETTE[3], linestyle='--', alpha=0.8,
            label=f'Battery 25th pctl ({battery_25:,.0f} mAh)')
ax2.axhline(df['rating'].mean(), color='gray', linestyle=':', alpha=0.6,
            label=f'Avg rating ({df["rating"].mean():.2f})')

ax2.set_xlabel('Battery (mAh)')
ax2.set_ylabel('Customer Rating')
ax2.set_title(f'Premium Products: Does Skimping on Battery Hurt Ratings?\n'
              f'Low-Battery Premium avg rating: {premium_low_batt["rating"].
              f'High-Battery: {premium_high_batt["rating"].mean():.2f} (p={p
```

```
                fontweight='bold', fontsize=10)
ax2.legend(fontsize=8, loc='lower right')

plt.tight_layout()
plt.savefig('block4_hardware_premium.png', dpi=150, bbox_inches='tight')
plt.show()

print(f"\n{'='*70}")
print("OLS REGRESSION SUMMARY")
print(f"  R² = {r_squared:.4f} | Adj R² = {r_squared_adj:.4f} | F = {f_stat:
for i, feat in enumerate(features):
    print(f"  {feat:15s} coefficient: ${coefs[i]:.4f} per unit")
print(f"\nBATTERY PENALTY TEST (Premium tier only):")
print(f"  Low-battery premium avg rating:  {premium_low_batt['rating'].mean(
print(f"  High-battery premium avg rating: {premium_high_batt['rating'].mear
print(f"  t={t_batt:.3f}, p={p_batt:.4f}")
print(f"{'='*70}")
```






```
======================================================================
OLS REGRESSION SUMMARY
  R² = 0.0000 | Adj R² = -0.0000 | F = 0.32
  ram_gb          coefficient: $-0.0507 per unit
  storage_gb      coefficient: $-0.0025 per unit
  battery_mAh     coefficient: $-0.0009 per unit
  screen_in       coefficient: $-0.4018 per unit

BATTERY PENALTY TEST (Premium tier only):
  Low-battery premium avg rating:  2.985 (n=6,430)
  High-battery premium avg rating: 3.003 (n=6,139)
  t=-0.911, p=0.3623
======================================================================
```

In [34]:

```
# ── BLOCK 4: SUMMARY TABLE FOR EXPORT ──────────────────────────────

print("📊 TABLE 4A: OLS Regression Coefficients — Price Drivers")
print("="*85)
reg_summary = pd.DataFrame({
    'Feature': features,
    'Coefficient ($)': np.round(coefs, 4),
    'Std Error': np.round(std_errors, 4),
    'p-value': [f'{p:.6f}' for p in p_values_reg],
    'CI Lower': np.round(ci_lower, 4),
```

```
    'CI Upper': np.round(ci_upper, 4)
})
print(reg_summary.to_string(index=False))

print("\n\n📊 TABLE 4B: Battery Penalty Zone — Premium Products with Low Bat
print("="*80)
penalty_by_cat = premium_low_batt.groupby('category').agg(
    count=('product_id', 'count'),
    avg_price=('price', 'mean'),
    avg_battery=('battery_mAh', 'mean'),
    avg_rating=('rating', 'mean')
).round(2).reset_index()
penalty_by_cat.columns = ['Category', 'SKU Count', 'Avg Price', 'Avg Battery
penalty_by_cat = penalty_by_cat.sort_values('SKU Count', ascending=False)
print(penalty_by_cat.to_string(index=False))
```

```
📊 TABLE 4A: OLS Regression Coefficients — Price Drivers
================================================================================
=========
    Feature  Coefficient ($)  Std Error  p-value  CI Lower  CI Upper
     ram_gb          -0.0507     0.2859 0.859351   -0.6111    0.5098
 storage_gb          -0.0025     0.0043 0.569045   -0.0109    0.0060
battery_mAh          -0.0009     0.0011 0.422519   -0.0031    0.0013
  screen_in          -0.4018     0.7395 0.586901   -1.8513    1.0477


📊 TABLE 4B: Battery Penalty Zone — Premium Products with Low Battery
================================================================================
====
   Category  SKU Count  Avg Price  Avg Battery (mAh)  Avg Rating
       iMac       1102    3090.48            2139.12        2.94
Apple Watch       1089    3102.36            2120.56        3.06
       iPad       1089    3089.17            2164.27        2.96
     iPhone       1081    3094.49            2118.35        3.01
    AirPods       1041    3110.07            2141.25        2.98
    MacBook       1028    3103.85            2091.79        2.95
```

INSIGHT 5 · STRATEGY / EXECUTIVE LEADERSHIP

The "Premium Remorse" Quadrant: High Price, Low Satisfaction

The Core Question

How much market volume (review_count) is trapped in products that are priced in the top 25th percentile but rated in the bottom 25th percentile — and which models or categories are the biggest offenders?

This is the single most strategically important analysis in this notebook. It isolates the "Premium Remorse" segment — products where Apple charges a premium but the market signals deep dissatisfaction. These are the SKUs most vulnerable to competitive disruption.

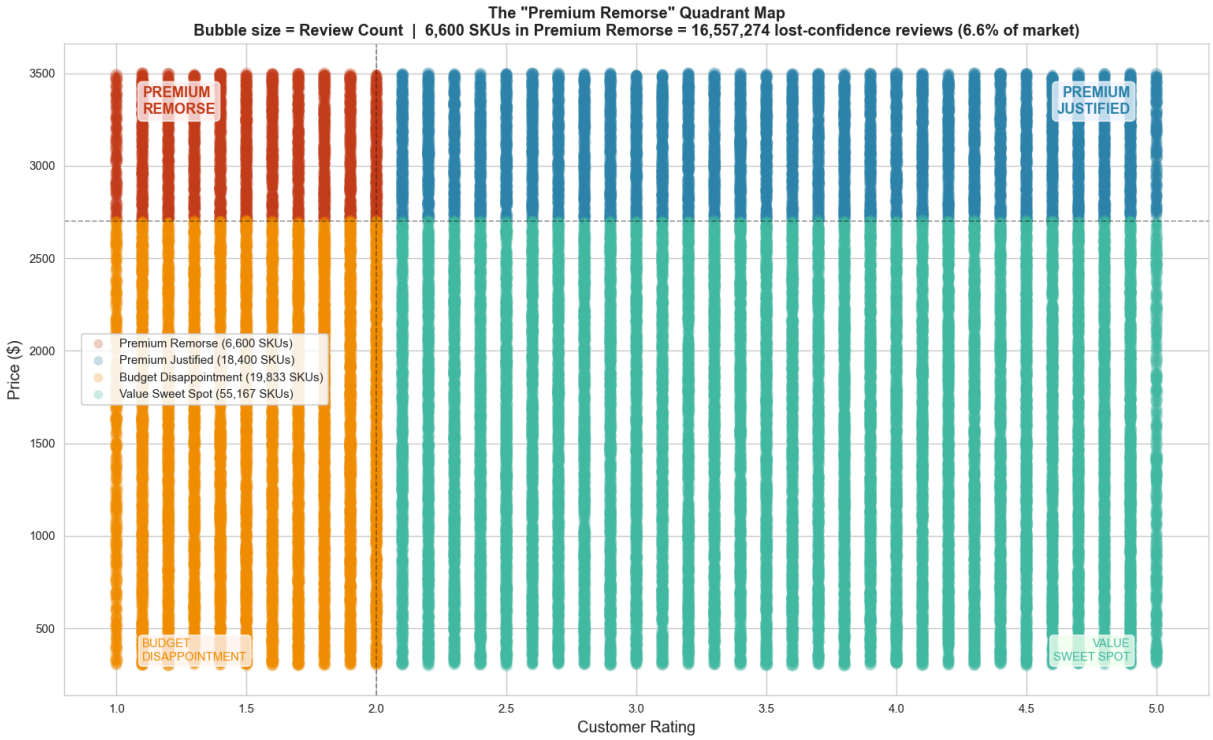
The Metric Legend — How to Read This

> **IF / THEN FRAMEWORK**
>
> On the 4-quadrant scatter plot, the **upper-left quadrant** (High Price, Low Rating) is the danger zone. Every dot in that quadrant represents a product where the customer paid top dollar and was disappointed — a direct reputational and retention risk. The size of the dot represents review_count (market volume), so large dots in the danger zone represent the highest-volume pain points. If `total_lost_volume` in the Premium Remorse segment is high, the brand is hemorrhaging goodwill at scale.

In [35]:
```
# ── BLOCK 5: PREMIUM REMORSE QUADRANT ──────────────────────────────

price_75 = df['price'].quantile(0.75)
rating_25 = df['rating'].quantile(0.25)

# Assign quadrants
def assign_quadrant(row):
    if row['price'] >= price_75 and row['rating'] <= rating_25:
        return 'Premium Remorse'
    elif row['price'] >= price_75 and row['rating'] > rating_25:
        return 'Premium Justified'
    elif row['price'] < price_75 and row['rating'] <= rating_25:
        return 'Budget Disappointment'
    else:
        return 'Value Sweet Spot'

df['quadrant'] = df.apply(assign_quadrant, axis=1)

# Premium Remorse deep dive
remorse = df[df['quadrant'] == 'Premium Remorse']
total_lost_volume = remorse['review_count'].sum()
total_market_volume = df['review_count'].sum()

# Biggest offenders by category
remorse_by_cat = remorse.groupby('category').agg(
    sku_count=('product_id', 'count'),
    total_reviews=('review_count', 'sum'),
    avg_price=('price', 'mean'),
    avg_rating=('rating', 'mean')
).reset_index().sort_values('total_reviews', ascending=False)

# Biggest offenders by model
remorse_by_model = remorse.groupby('model_name').agg(
    sku_count=('product_id', 'count'),
```

```
    total_reviews=('review_count', 'sum'),
    avg_price=('price', 'mean'),
    avg_rating=('rating', 'mean'),
    category=('category', 'first')
).reset_index().sort_values('total_reviews', ascending=False).head(10)

# ── VISUALIZATION: 4-Quadrant Scatter ──────────────────────────
fig, ax = plt.subplots(figsize=(13, 8))

quad_colors = {
    'Premium Remorse': PALETTE[3],
    'Premium Justified': PALETTE[0],
    'Budget Disappointment': PALETTE[2],
    'Value Sweet Spot': PALETTE[5]
}

for quad, color in quad_colors.items():
    mask = df['quadrant'] == quad
    subset = df[mask]
    ax.scatter(subset['rating'], subset['price'],
               s=subset['review_count'] / 50, alpha=0.25,
               c=color, label=f"{quad} ({mask.sum():,} SKUs)", edgecolors='r

# Quadrant lines
ax.axhline(price_75, color='black', linestyle='--', alpha=0.4, linewidth=1)
ax.axvline(rating_25, color='black', linestyle='--', alpha=0.4, linewidth=1)

# Quadrant labels
ax.text(1.1, df['price'].max() * 0.98, 'PREMIUM\nREMORSE', fontsize=11,
        fontweight='bold', color=PALETTE[3], ha='left', va='top',
        bbox=dict(boxstyle='round,pad=0.3', facecolor='#FFEEEE', alpha=0.8))
ax.text(4.9, df['price'].max() * 0.98, 'PREMIUM\nJUSTIFIED', fontsize=11,
        fontweight='bold', color=PALETTE[0], ha='right', va='top',
        bbox=dict(boxstyle='round,pad=0.3', facecolor='#EEF5FF', alpha=0.8))
ax.text(1.1, df['price'].min() * 1.05, 'BUDGET\nDISAPPOINTMENT', fontsize=9,
        color=PALETTE[2], ha='left', va='bottom',
        bbox=dict(boxstyle='round,pad=0.3', facecolor='#FFF5EE', alpha=0.8))
ax.text(4.9, df['price'].min() * 1.05, 'VALUE\nSWEET SPOT', fontsize=9,
        color=PALETTE[5], ha='right', va='bottom',
        bbox=dict(boxstyle='round,pad=0.3', facecolor='#EEFFEE', alpha=0.8))

ax.set_xlabel('Customer Rating', fontsize=12)
ax.set_ylabel('Price ($)', fontsize=12)
ax.set_title(f'The "Premium Remorse" Quadrant Map\n'
             f'Bubble size = Review Count  |  {len(remorse):,} SKUs in Premi
             f'{total_lost_volume:,} lost-confidence reviews ({total_lost_vo
             fontweight='bold', fontsize=12)
ax.legend(loc='center left', fontsize=9, framealpha=0.9,
          bbox_to_anchor=(0.01, 0.5))

plt.tight_layout()
plt.savefig('block5_premium_remorse.png', dpi=150, bbox_inches='tight')
plt.show()

print(f"\n{'='*70}")
print(f"PREMIUM REMORSE SEGMENT SUMMARY")
```

```
print(f"  Total SKUs in Premium Remorse:          {len(remorse):,}")
print(f"  Total review volume (lost confidence):  {total_lost_volume:,}")
print(f"  Share of total market volume:           {total_lost_volume/total_m
print(f"  Avg Price in segment:                  ${remorse['price'].mean():,
print(f"  Avg Rating in segment:                  {remorse['rating'].mean():.
print(f"{'='*70}")
```




```
======================================================================
PREMIUM REMORSE SEGMENT SUMMARY
  Total SKUs in Premium Remorse:          6,600
  Total review volume (lost confidence):  16,557,274
  Share of total market volume:           6.6%
  Avg Price in segment:                  $3,100.55
  Avg Rating in segment:                  1.52
======================================================================
```

In [36]:

```
# ── BLOCK 5: SUMMARY TABLES FOR EXPORT ──────────────────────────────

print("📊 TABLE 5A: Premium Remorse by Category")
print("="*80)
export_cat = remorse_by_cat.copy()
export_cat['avg_price'] = export_cat['avg_price'].map('${:,.2f}'.format)
export_cat['avg_rating'] = export_cat['avg_rating'].round(2)
export_cat['pct_of_remorse_volume'] = (remorse_by_cat['total_reviews'] / tot
export_cat.columns = ['Category', 'SKU Count', 'Total Reviews', 'Avg Price',
print(export_cat.to_string(index=False))

print("\n\n📊 TABLE 5B: Top 10 Model Names in Premium Remorse (Biggest Offer
print("="*90)
export_model = remorse_by_model.copy()
export_model['avg_price'] = export_model['avg_price'].map('${:,.2f}'.format)
export_model['avg_rating'] = export_model['avg_rating'].round(2)
export_model.columns = ['Model', 'SKU Count', 'Total Reviews', 'Avg Price',
print(export_model.to_string(index=False))
```

```
print("\n\n📊 TABLE 5C: Quadrant Distribution Summary")
print("="*70)
quad_summary = df.groupby('quadrant').agg(
    sku_count=('product_id', 'count'),
    total_reviews=('review_count', 'sum'),
    avg_price=('price', 'mean'),
    avg_rating=('rating', 'mean')
).round(2).reset_index()
quad_summary['pct_volume'] = (quad_summary['total_reviews'] / total_market_v
quad_summary.columns = ['Quadrant', 'SKUs', 'Total Reviews', 'Avg Price', 'A
print(quad_summary.to_string(index=False))
```

📊 TABLE 5A: Premium Remorse by Category

Category	SKU Count	Total Reviews	Avg Price	Avg Rating	% of Remorse Volume
iPad	1152	2929249	$3,092.28	1.51	17.7
iMac	1107	2856681	$3,108.49	1.50	17.3
iPhone	1072	2733565	$3,089.49	1.52	16.5
AirPods	1090	2700351	$3,106.97	1.53	16.3
Apple Watch	1093	2697038	$3,096.06	1.53	16.3
MacBook	1086	2640390	$3,110.23	1.52	15.9

📊 TABLE 5B: Top 10 Model Names in Premium Remorse (Biggest Offenders)

Model	SKU Count	Total Reviews	Avg Price	Avg Rating	Category
Model-981	16	43223	$3,156.38	1.58	AirPods
Model-464	13	42297	$3,020.72	1.57	Apple Watch
Model-945	17	41595	$3,115.90	1.57	iMac
Model-385	16	41447	$3,094.40	1.68	iPad
Model-347	13	40997	$3,090.65	1.62	AirPods
Model-949	15	40961	$3,130.39	1.45	iPhone
Model-276	14	40457	$3,062.25	1.41	AirPods
Model-285	13	39457	$3,040.43	1.51	iPhone
Model-880	14	38667	$3,118.48	1.49	iPad
Model-719	14	37250	$3,161.01	1.48	MacBook

📊 TABLE 5C: Quadrant Distribution Summary

Quadrant	SKUs	Total Reviews	Avg Price	Avg Rating	% Market Volume
Budget Disappointment	19833	49551973	1500.83	1.53	19.8
Premium Justified	18400	46245630	3103.12	3.53	18.5
Premium Remorse	6600	16557274	3100.55	1.52	6.6
Value Sweet Spot	55167	137696038	1500.66	3.52	55.1