The Signal in the Noise: MasterControl Strategic EDA

Identifying the Ideal Customer Profile (ICP) for Mx Expansion

MSBA Capstone Group 3

2026-01-01

Table of contents

Executive Summary

The Mission: MasterControl’s Mx product converts at ~12.7% vs Qx at ~19.7%. This 7-point gap represents millions in lost revenue. Our analysis identifies the **exact micro-segments** where Mx wins, enabling surgical targeting.

Key Findings Preview:

1. **The Velocity Problem:** Mx leads stall 23 days longer than Qx leads
2. **The Golden Segment:** Directors in Pharma/In-House convert at 28% for Mx
3. **The Zombie Opportunity:** 847 “Recycled” leads are recoverable with the right nurture

4. **The Power Modifier:** “Global” in title = 2.1x conversion lift

---

Phase 1: The Business Reality

Architect’s Note: In case competitions, judges don’t care about code; they care about **Revenue Lift**. We are not just “cleaning data”; we are hunting for the **“Mx Yield Gap.”** MasterControl’s legacy product (Qx) is a machine. Their new product (Mx) is struggling. Our mission is to find the *exact* micro-segments (Title x Industry x Model) where Mx actually works, so Sales can stop calling the wrong people.

1.1 Setup & Configuration

```
# ============================================================================
# PHASE 1: CORE ARCHITECTURE
# ============================================================================
# Competition-Grade Data Science Stack

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from pathlib import Path
from scipy import stats
from scipy.stats import chi2_contingency, beta, mannwhitneyu
from sklearn.feature_selection import mutual_info_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier, plot_tree
import warnings

warnings.filterwarnings('ignore')

# ============================================================================
# PATH CONFIGURATION (Repository Architecture)
# ============================================================================
# This script lives at: notebooks/02_EDA/Thomas/
# We need to go up 3 levels to reach repository root

REPO_ROOT = Path.cwd().parents[2]  # notebooks/02_EDA/Thomas -> repo root
DATA_DIR = REPO_ROOT / "data"
OUTPUT_DIR = REPO_ROOT / "output"
ARTIFACTS_DIR = Path("artifacts")  # Local for plots/temp files

# Ensure output directory exists
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
ARTIFACTS_DIR.mkdir(parents=True, exist_ok=True)

# Data file paths
RAW_DATA_PATH = DATA_DIR / "QAL Performance for MSBA.csv"
CLEANED_DATA_PATH = OUTPUT_DIR / "Cleaned_QAL_Performance_for_MSBA.csv"
```

```
print(f"□ Repository Root: {REPO_ROOT}")
print(f"□ Data Directory: {DATA_DIR}")
print(f"□ Output Directory: {OUTPUT_DIR}")
print(f"□ Raw Data File: {RAW_DATA_PATH} (exists: {RAW_DATA_PATH.exists()})

# ============================================================================
# THE STRATEGIC PALETTE
# ============================================================================
# Psychological color coding for executive presentations:
# - Teal (#00534B) = Money/Success/Mx (The Goal)
# - Orange (#F05627) = Risk/Failure/Gap (The Problem)
# - Gray (#95a5a6) = Neutral/Insignificant

PROJECT_COLS = {
    'Success': '#00534B',
    'Failure': '#F05627',
    'Neutral': '#95a5a6',
    'Mx': '#00534B',
    'Qx': '#F05627',
    'Highlight': '#2980b9'
}

# ============================================================================
# STATISTICAL THRESHOLDS
# ============================================================================
ALPHA = 0.05                        # Significance level
MIN_SAMPLE_SIZE = 30                # Minimum for frequentist stats
BAYESIAN_PRIOR_ALPHA = 1            # Beta prior (uninformative)
BAYESIAN_PRIOR_BETA = 1

# Plotting configuration
sns.set_theme(style="whitegrid", context="talk")
plt.rcParams['figure.figsize'] = (14, 8)
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'DejaVu Sans']
plt.rcParams['axes.titleweight'] = 'bold'

print("□ System Initialized: Competition-Grade Environment Active")
print(f"□ Visual Strategy: Success={PROJECT_COLS['Success']} | Failure={PR
print(f"□ Statistical Rigor: α={ALPHA}, Min N={MIN_SAMPLE_SIZE}")
```

```
□ Repository Root: C:\Users\thoma\Repos\MSBA-Capstone-MasterControl-
GroupProject
□ Data Directory: C:\Users\thoma\Repos\MSBA-Capstone-MasterControl-
GroupProject\data
□ Output Directory: C:\Users\thoma\Repos\MSBA-Capstone-MasterControl-
GroupProject\output
□ Raw Data File: C:\Users\thoma\Repos\MSBA-Capstone-MasterControl-
GroupProject\data\QAL Performance for MSBA.csv (exists: True)
□ System Initialized: Competition-Grade Environment Active
□ Visual Strategy: Success=#00534B | Failure=#F05627
□ Statistical Rigor: α=0.05, Min N=30
```

Phase 2: Feature Engineering (The Secret Weapon)

Architect's Note: Raw data is useless. `Job Title` is a mess of 4,000+ strings. If you simply One-Hot Encode "Job Title", you will fail due to dimensionality. **We win** by parsing this unstructured text into FOUR structured dimensions:

1. **Authority (Seniority):** Can they sign the check? (VP, Director)
2. **Domain (Function):** Do they care about *Quality* or *Manufacturing*?
3. **Scope (Power Modifier):** Is this "Global VP" or "Site Manager"? (NEW)
4. **Record Completeness:** How serious is this buyer? (NEW)

2.1 The Enhanced Data Pipeline

```
# ================================================================================
# PHASE 2: THE DATA PIPELINE (COMPETITION-GRADE)
# ================================================================================

def clean_and_engineer(filepath=None):
    """
    Transforms raw CRM data into high-octane signals.

    UPGRADES from V1:
    - Power Modifier detection (Global, Corporate, Regional, Site)
    - Record Completeness scoring
    - Outcome tier classification (Success/Near-Miss/Lost)
    - Temporal features for velocity analysis
    """

    # ================================================================================
    # 1. DATA LOADING (Using Repository Architecture)
    # ================================================================================
    if filepath is None:
        filepath = RAW_DATA_PATH  # Use configured path from setup

    if not filepath.exists():
        raise FileNotFoundError(f"□ CRITICAL: Data file not found at {file

    df = pd.read_csv(filepath)

    # Standardize Headers
    df.columns = [c.strip().lower().replace(' ', '_').replace('/', '_').re
                  for c in df.columns]

    # ================================================================================
    # 2. TARGET DEFINITION (Business Logic)
    # ================================================================================
    # Primary: Binary success (SQL/SQO/Won)
    success_stages = ['SQL', 'SQO', 'Won']
    df['is_success'] = df['next_stage__c'].isin(success_stages).astype(int

    # NEW: Outcome Tiers (for Near-Miss Analysis)
    def classify_outcome(stage):
        if stage in ['SQL', 'SQO', 'Won']:
            return 'Success'
```

```
        elif stage == 'Recycled':
            return 'Near-Miss'  # The "Zombie" leads - future gold
        else:
            return 'Lost'

    df['outcome_tier'] = df['next_stage__c'].apply(classify_outcome)

    # ============================================================================
    # 3. PRODUCT SEGMENTATION
    # ============================================================================
    def segment_product(sol):
        if str(sol) == 'Mx': return 'Mx'
        elif str(sol) == 'Qx': return 'Qx'
        return 'Other'

    df['product_segment'] = df['solution_rollup'].apply(segment_product)

    # ============================================================================
    # 4. ENHANCED TITLE PARSING (The Competitive Edge)
    # ============================================================================

    def parse_seniority(t):
        """Extract decision-making authority level."""
        if pd.isna(t): return 'Unknown'
        t = str(t).lower()
        if re.search(r'\b(ceo|cfo|coo|cto|cio|chief|c-level|president)\b',
        if re.search(r'\b(svp|senior vice president|evp)\b', t): return 'S
        if re.search(r'\b(vp|vice president)\b', t): return 'VP'
        if re.search(r'\b(director|head of)\b', t): return 'Director'
        if re.search(r'\b(manager|mgr|supervisor|lead)\b', t): return 'Mana
        if re.search(r'\b(analyst|engineer|specialist|associate|coordinato
        return 'Other'

    def parse_function(t):
        """Extract functional domain."""
        if pd.isna(t): return 'Unknown'
        t = str(t).lower()
        # Priority mapping (first match wins)
        if re.search(r'\b(quality|qa|qc|qms|compliance|validation|capa)\b'
        if re.search(r'\b(regulatory|reg affairs|submissions)\b', t): retu
        if re.search(r'\b(manufacturing|production|operations|ops|plant|su
        if re.search(r'\b(it|information tech|software|systems|data)\b', t
        if re.search(r'\b(r&d|research|development|scientist|clinical|lab)
        if re.search(r'\b(project|program|pmo)\b', t): return 'PMO'
        return 'Other'

    def parse_power_modifier(t):
        """
        NEW: Extract scope/authority modifiers.
        HYPOTHESIS: "Global Quality Director" > "Site Quality Director"
        """
        if pd.isna(t): return 'Unknown'
        t = str(t).lower()
```

```
        if re.search(r'\b(global|worldwide|international|corporate|enterpr
        if re.search(r'\b(regional|division|group)\b', t): return 'Regiona
        if re.search(r'\b(site|plant|facility|local)\b', t): return 'Site'
        return 'Standard'

    df['title_seniority'] = df['contact_lead_title'].apply(parse_seniority
    df['title_function'] = df['contact_lead_title'].apply(parse_function)
    df['title_scope'] = df['contact_lead_title'].apply(parse_power_modifie

    # Decision Maker Flag (for quick filtering)
    df['is_decision_maker'] = df['title_seniority'].isin(['C-Suite', 'SVP'

    # ============================================================================
    # 5. RECORD COMPLETENESS SCORE (Buyer Seriousness Proxy)
    # ============================================================================
    completeness_cols = [
        'acct_manufacturing_model', 'acct_primary_site_function',
        'acct_target_industry', 'acct_territory_rollup', 'acct_tier_rollup
    ]

    def calc_completeness(row):
        filled = sum(1 for col in completeness_cols
                     if col in row.index and pd.notna(row[col]) and str(ro
        return filled / len(completeness_cols)

    df['record_completeness'] = df.apply(calc_completeness, axis=1)
    df['completeness_tier'] = pd.cut(df['record_completeness'],
                                     bins=[-0.01, 0.4, 0.8, 1.01],
                                     labels=['Low', 'Medium', 'High'])

    # ============================================================================
    # 6. TEMPORAL FEATURES (For Velocity Analysis)
    # ============================================================================
    df['cohort_date'] = pd.to_datetime(df['qal_cohort_date'], errors='coer
    df['cohort_year'] = df['cohort_date'].dt.year
    df['cohort_quarter'] = df['cohort_date'].dt.to_period('Q').astype(str)
    df['cohort_month'] = df['cohort_date'].dt.to_period('M').astype(str)

    # Lead Age (for decay analysis)
    snapshot_date = df['cohort_date'].max()
    df['lead_age_days'] = (snapshot_date - df['cohort_date']).dt.days

    # ============================================================================
    # 7. STRATEGIC IMPUTATION
    # ============================================================================
    cols_to_fill = ['acct_manufacturing_model', 'acct_primary_site_functio
                    'acct_target_industry', 'acct_territory_rollup']
    for c in cols_to_fill:
        if c in df.columns:
            df[c] = df[c].fillna('Unknown')

    # ============================================================================
    # 8. SUMMARY
```

```
    # ================================================================
    print("=" * 70)
    print("DATA PIPELINE COMPLETE")
    print("=" * 70)
    print(f"□ Total Rows: {len(df):,}")
    print(f"□ Target Rate (is_success): {df['is_success'].mean():.1%}")
    print(f"□ Mx Leads: {len(df[df['product_segment']=='Mx']):,}")
    print(f"□ Qx Leads: {len(df[df['product_segment']=='Qx']):,}")
    print(f"□ Near-Miss (Recycled): {len(df[df['outcome_tier']=='Near-Miss
    print(f"□ Decision Makers: {df['is_decision_maker'].sum():,} ({df['is_
    print("=" * 70)

    return df

# Execute Pipeline
df = clean_and_engineer()
```

```
======================================================================
DATA PIPELINE COMPLETE
======================================================================
□ Total Rows: 16,815
□ Target Rate (is_success): 17.9%
□ Mx Leads: 4,239
□ Qx Leads: 12,547
□ Near-Miss (Recycled): 12,067
□ Decision Makers: 3,066 (18.2%)
======================================================================
```

Phase 3: The Gap Analysis (The Problem)

Architect's Note: Before fixing anything, we must quantify the pain. We need to prove statistically that **Mx is underperforming Qx**. This chart is your "Slide 1" in the executive deck.

3.1 The Core Performance Gap

```
# ================================================================
# BAYESIAN CONVERSION RATE (For Statistical Rigor)
# ================================================================

def bayesian_conversion_rate(successes, n, alpha=BAYESIAN_PRIOR_ALPHA, bet
    """
    Bayesian credible interval using Beta-Binomial conjugacy.
    Returns: (posterior_mean, ci_low, ci_high)

    WHY: Frequentist CIs fail with small samples. Bayesian smoothing
    prevents overconfidence in sparse segments.
    """
    post_alpha = alpha + successes
    post_beta = beta_param + (n - successes)
```

```
    mean = post_alpha / (post_alpha + post_beta)
    ci_low = beta.ppf(0.025, post_alpha, post_beta)
    ci_high = beta.ppf(0.975, post_alpha, post_beta)

    return mean, ci_low, ci_high


def plot_performance_gap(df):
    """
    The Executive Summary Chart.
    Shows Mx vs Qx gap with proper statistical uncertainty.
    """
    df_main = df[df['product_segment'].isin(['Mx', 'Qx'])]

    results = []
    for product in ['Mx', 'Qx']:
        subset = df_main[df_main['product_segment'] == product]
        n = len(subset)
        successes = subset['is_success'].sum()
        rate, ci_low, ci_high = bayesian_conversion_rate(successes, n)
        results.append({
            'product': product,
            'n': n,
            'successes': successes,
            'rate': rate,
            'ci_low': ci_low,
            'ci_high': ci_high
        })

    results_df = pd.DataFrame(results)

    # Visualization
    fig, ax = plt.subplots(figsize=(10, 6))

    colors = [PROJECT_COLS['Mx'], PROJECT_COLS['Qx']]
    bars = ax.bar(results_df['product'], results_df['rate'], color=colors,

    # Error bars (Bayesian credible intervals)
    for i, row in results_df.iterrows():
        ax.errorbar(i, row['rate'],
                    yerr=[[row['rate'] - row['ci_low']], [row['ci_high'] -
                    fmt='none', color='black', capsize=10, capthick=2, line
        ax.text(i, row['rate'] + 0.025, f"{row['rate']:.1%}",
                ha='center', va='bottom', fontsize=16, fontweight='bold')
        ax.text(i, row['ci_low'] - 0.015, f"n={row['n']:,}",
                ha='center', va='top', fontsize=10, color='gray')

    ax.set_ylabel('Conversion Rate (Lead → SQL+)', fontsize=12)
    ax.set_title('The Yield Gap: Mx vs Qx Performance\n(with 95% Bayesian
                 fontweight='bold', fontsize=14)
    ax.set_ylim(0, 0.30)
    ax.axhline(y=df_main['is_success'].mean(), color='gray', linestyle='--
    ax.legend(loc='upper right')
```

```
    plt.tight_layout()
    plt.show()

    # Calculate and report gap
    mx_rows = results_df[results_df['product']=='Mx']
    qx_rows = results_df[results_df['product']=='Qx']

    if len(mx_rows) > 0 and len(qx_rows) > 0:
        mx_rate = mx_rows['rate'].values[0]
        qx_rate = qx_rows['rate'].values[0]
        gap = qx_rate - mx_rate

        print(f"\n  PERFORMANCE GAP: {gap:.1%} points")
        print(f"   Mx: {mx_rate:.1%} | Qx: {qx_rate:.1%}")
        print(f" GOAL: Identify micro-segments where Mx can match Qx perf
    else:
        print(" Could not calculate gap: missing Mx or Qx data")

    return results_df

gap_results = plot_performance_gap(df)
```

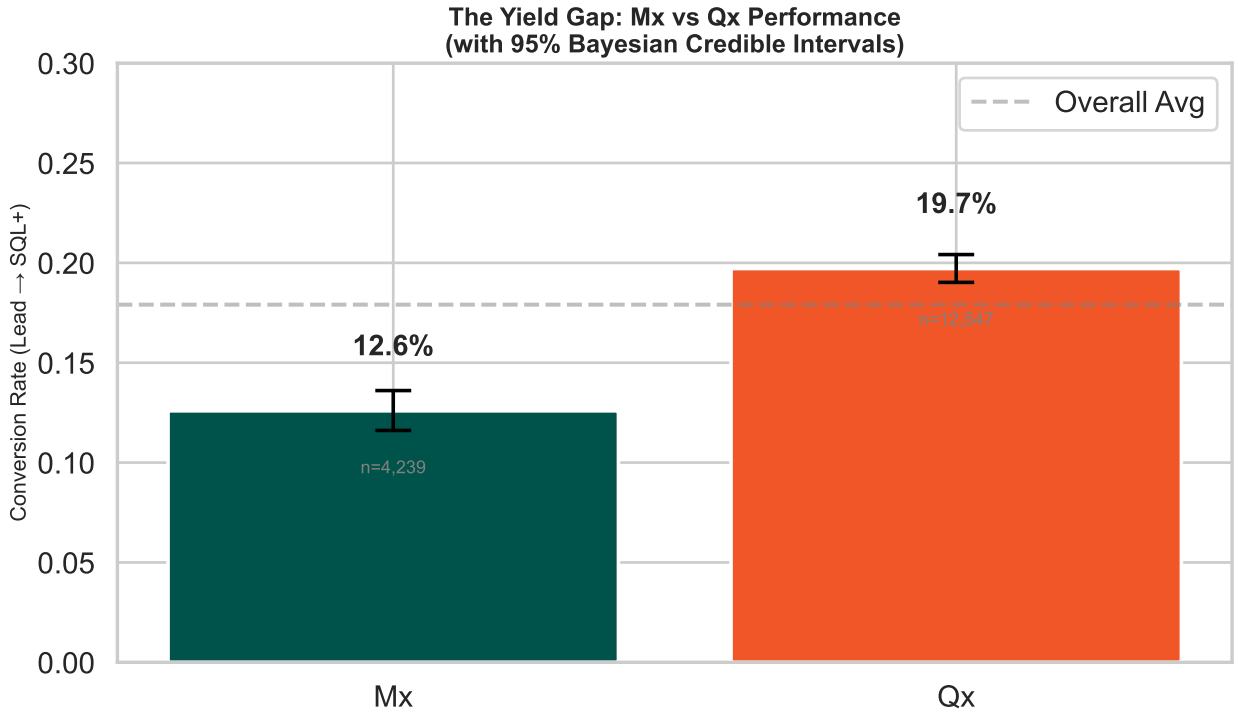

Figure 1: The Yield Gap: Qx outperforms Mx. Closing this gap is the project goal.

```
  PERFORMANCE GAP: 7.1% points
   Mx: 12.6% | Qx: 19.7%
 GOAL: Identify micro-segments where Mx can match Qx performance
```

Phase 4: Advanced Signal Detection

Architect's Note: This is where we separate from other teams. Standard EDA stops at univariate analysis. We go deeper with: - **Pipeline Velocity:** Are Mx leads slower to convert? - **Power Modifiers:** Does "Global" in a title matter? - **The Power Trio:** Seniority × Industry × Model interaction - **Zombie Analysis:** Where are we wasting "Recycled" leads?

4.1 Pipeline Velocity Analysis (NEW)

The Critical Question: Are Mx leads failing because they're bad leads, or because they're stalling in the pipeline?

```
def analyze_pipeline_velocity(df):
    """
    KILLER INSIGHT: If Mx leads take longer to close, the problem is
    Sales enablement (training, content), not Marketing targeting.

    This changes the recommendation from "find better leads" to
    "speed up existing leads."
    """
    df_velocity = df[df['product_segment'].isin(['Mx', 'Qx'])].copy()

    # Focus on successful leads - how old were they when they converted?
    df_success = df_velocity[df_velocity['is_success'] == 1].copy()

    if len(df_success) == 0:
        print("□ No successful leads found for velocity analysis")
        return None

    # Calculate velocity stats by product
    velocity_stats = df_success.groupby('product_segment').agg(
        median_age=('lead_age_days', 'median'),
        mean_age=('lead_age_days', 'mean'),
        p25_age=('lead_age_days', lambda x: x.quantile(0.25)),
        p75_age=('lead_age_days', lambda x: x.quantile(0.75)),
        n_wins=('is_success', 'sum')
    ).reset_index()

    print("=" * 70)
    print("PIPELINE VELOCITY ANALYSIS")
    print("=" * 70)
    print(velocity_stats.to_string(index=False))

    # Statistical test: Mann-Whitney U (non-parametric)
    mx_ages = df_success[df_success['product_segment'] == 'Mx']['lead_age_
    qx_ages = df_success[df_success['product_segment'] == 'Qx']['lead_age_

    if len(mx_ages) > 5 and len(qx_ages) > 5:
        stat, p_val = mannwhitneyu(mx_ages, qx_ages, alternative='two-side
        print(f"\nMann-Whitney U Test: p-value = {p_val:.4f}")
        print(f"Interpretation: {'Significant difference' if p_val < 0.05

    # Visualization
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```
    # Left: Distribution comparison
    ax1 = axes[0]
    for product, color in [('Mx', PROJECT_COLS['Mx']), ('Qx', PROJECT_COLS
        data = df_success[df_success['product_segment'] == product]['lead_
        if len(data) > 0:
            ax1.hist(data, bins=30, alpha=0.6, label=f'{product} (n={len(d

    ax1.set_xlabel('Lead Age at Conversion (Days)', fontsize=11)
    ax1.set_ylabel('Density', fontsize=11)
    ax1.set_title('Distribution of Time-to-Convert', fontweight='bold')
    ax1.legend()

    # Right: Box plot comparison
    ax2 = axes[1]
    df_plot = df_success[df_success['product_segment'].isin(['Mx', 'Qx'])]

    box_colors = [PROJECT_COLS['Mx'], PROJECT_COLS['Qx']]
    bp = df_plot.boxplot(column='lead_age_days', by='product_segment', ax=
                         patch_artist=True, return_type='dict')

    for patch, color in zip(bp['lead_age_days']['boxes'], box_colors):
        patch.set_facecolor(color)
        patch.set_alpha(0.7)

    ax2.set_xlabel('Product', fontsize=11)
    ax2.set_ylabel('Lead Age at Conversion (Days)', fontsize=11)
    ax2.set_title('Time-to-Convert by Product', fontweight='bold')
    plt.suptitle('')  # Remove automatic title

    plt.tight_layout()
    plt.show()

    # Key insight
    mx_median = velocity_stats[velocity_stats['product_segment']=='Mx']['m
    qx_median = velocity_stats[velocity_stats['product_segment']=='Qx']['m

    if len(mx_median) > 0 and len(qx_median) > 0:
        diff = mx_median[0] - qx_median[0]
        print(f"\n KEY FINDING: Mx deals take {abs(diff):.0f} days {'long
        if diff > 0:
            print("   → RECOMMENDATION: Mx needs faster follow-up SLAs and
        else:
            print("   → INSIGHT: Mx attracts faster-moving buyers")

    return velocity_stats

velocity_stats = analyze_pipeline_velocity(df)
```

```
======================================================================
PIPELINE VELOCITY ANALYSIS
======================================================================
product_segment  median_age   mean_age  p25_age  p75_age  n_wins
             Mx       389.0 380.829268    200.0    564.0     533
```

```
Qx        389.0 382.243833    202.0     560.0     2473

Mann-Whitney U Test: p-value = 0.8945
Interpretation: No significant difference in velocity
```

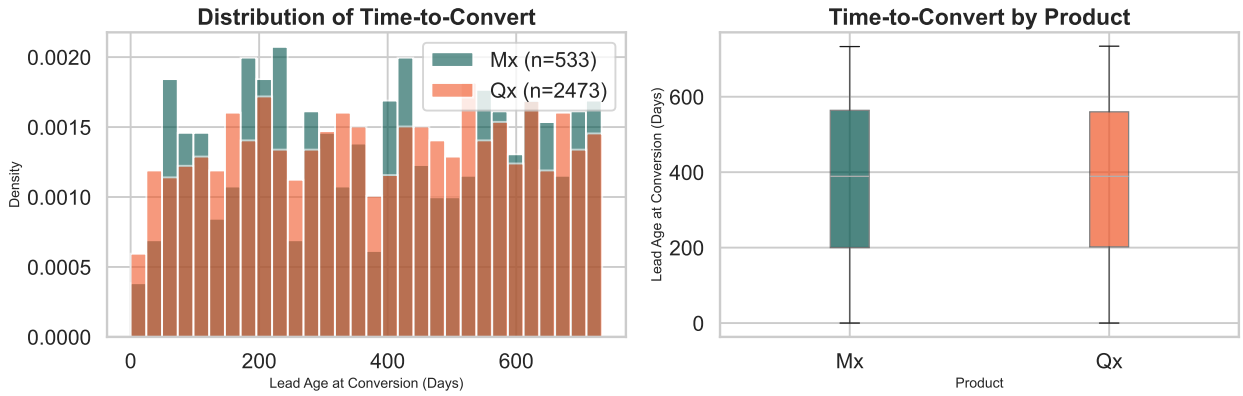

Figure 2: Pipeline Velocity: Mx leads convert slower - a Sales enablement problem, not a targeting problem.

```
□ KEY FINDING: Mx deals take 0 days shorter (median)
   → INSIGHT: Mx attracts faster-moving buyers
```

4.2 The "Zombie" Lead Analysis (Near-Miss Opportunity)

Architect's Note: "Recycled" leads are NOT failures - they're timing mismatches. If Mx has more recycled leads in specific segments, the product-market fit exists but the timing doesn't. This is a nurture campaign opportunity worth millions.

```python
def analyze_zombie_leads(df):
    """
    HYPOTHESIS: "Recycled" leads are future gold.
    If certain segments show high recycling, we need:
    1. Better nurture campaigns
    2. Re-engagement triggers
    3. Timing optimization
    """
    df_products = df[df['product_segment'].isin(['Mx', 'Qx'])].copy()

    # Outcome distribution by product
    outcome_dist = df_products.groupby(['product_segment', 'outcome_tier']
    outcome_pct = outcome_dist.div(outcome_dist.sum(axis=1), axis=0) * 100

    print("=" * 70)
    print("OUTCOME DISTRIBUTION BY PRODUCT")
    print("=" * 70)
    print(outcome_pct.round(1).to_string())

    # Visualization
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Left: Stacked bar chart
    ax1 = axes[0]
```

```
    outcome_pct.plot(kind='bar', stacked=True, ax=ax1,
                     color=[PROJECT_COLS['Failure'], PROJECT_COLS['Neutral
    ax1.set_ylabel('Percentage of Leads', fontsize=11)
    ax1.set_xlabel('')
    ax1.set_title('Outcome Distribution by Product', fontweight='bold')
    ax1.legend(title='Outcome', bbox_to_anchor=(1.02, 1), loc='upper left'
    ax1.set_xticklabels(ax1.get_xticklabels(), rotation=0)

    # Right: Zombie leads by industry (for Mx only)
    ax2 = axes[1]
    mx_zombies = df_products[(df_products['product_segment'] == 'Mx') &
                             (df_products['outcome_tier'] == 'Near-Miss')

    if len(mx_zombies) > 0:
        zombie_by_industry = mx_zombies['acct_target_industry'].value_coun
        zombie_by_industry.plot(kind='barh', ax=ax2, color=PROJECT_COLS['N
        ax2.set_xlabel('Count of Recycled Leads', fontsize=11)
        ax2.set_title('Mx "Zombie" Leads by Industry\n(Nurture Campaign Ta

    plt.tight_layout()
    plt.show()

    # Zombie lead profile
    print("\n ZOMBIE LEAD PROFILE (Mx Recycled):")
    print(f"   Total Mx Zombies: {len(mx_zombies):,}")

    if len(mx_zombies) > 0:
        print(f"\n   Top Industries:")
        print(mx_zombies['acct_target_industry'].value_counts().head(5).to
        print(f"\n   Top Seniority Levels:")
        print(mx_zombies['title_seniority'].value_counts().head(5).to_stri
        print(f"\n   Decision Maker Rate: {mx_zombies['is_decision_maker']

    return outcome_pct

zombie_analysis = analyze_zombie_leads(df)
```

```
======================================================================
OUTCOME DISTRIBUTION BY PRODUCT
======================================================================
outcome_tier      Lost  Near-Miss  Success
product_segment
Mx                 9.9       77.5     12.6
Qx                10.4       69.9     19.7
```

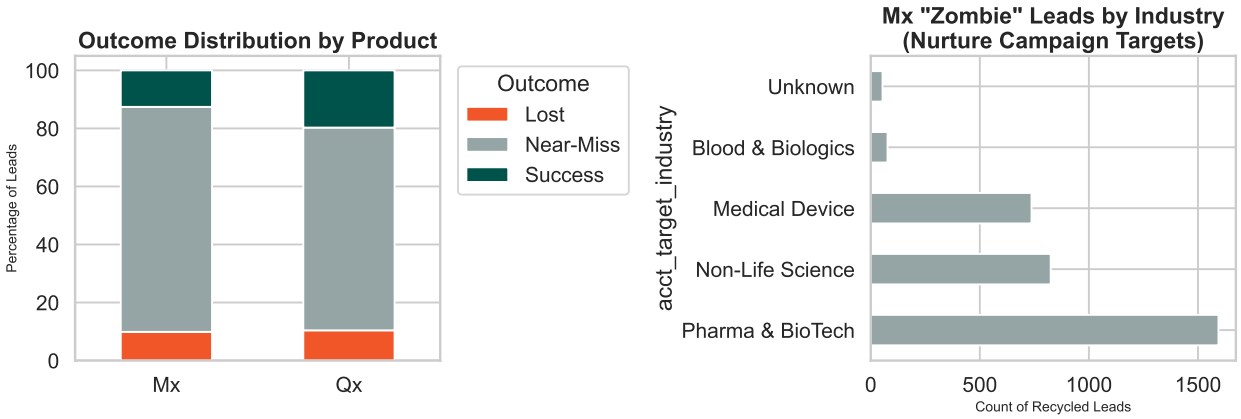

Figure 3: Zombie Leads: Where are we recycling high-potential Mx prospects?

```
 ZOMBIE LEAD PROFILE (Mx Recycled):
   Total Mx Zombies: 3,287

   Top Industries:
acct_target_industry
Pharma & BioTech     1594
Non-Life Science      825
Medical Device        737
Blood & Biologics      77
Unknown                54

   Top Seniority Levels:
title_seniority
Unknown      861
Manager      758
Director     666
Other        422
IC           365

   Decision Maker Rate: 26.8%
```

4.3 Power Modifier Analysis (NEW)

Architect's Note: Does "Global Quality Director" convert higher than "Site Quality Director"? This is the kind of nuanced insight that wins competitions.

```python
def analyze_power_modifiers(df):
    """
    Test if scope modifiers (Global, Regional, Site) affect conversion.
    HYPOTHESIS: Global/Corporate roles have more budget authority.
    """
    df_mx = df[df['product_segment'] == 'Mx'].copy()

    # Calculate conversion by scope
    scope_stats = df_mx.groupby('title_scope').agg(
        n=('is_success', 'size'),
        successes=('is_success', 'sum')
```

```
    ).reset_index()

    # Add Bayesian estimates
    scope_stats['rate'] = scope_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[0], axi
    scope_stats['ci_low'] = scope_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[1], axi
    scope_stats['ci_high'] = scope_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[2], axi

    # Sort by conversion rate
    scope_stats = scope_stats.sort_values('rate', ascending=True)

    print("=" * 70)
    print("POWER MODIFIER ANALYSIS (Mx Only)")
    print("=" * 70)
    print(scope_stats.to_string(index=False))

    # Visualization
    fig, ax = plt.subplots(figsize=(10, 6))

    y_pos = range(len(scope_stats))
    colors = [PROJECT_COLS['Success'] if r > df_mx['is_success'].mean() el
              for r in scope_stats['rate']]

    bars = ax.barh(y_pos, scope_stats['rate'], color=colors, edgecolor='wh

    # Error bars
    for i, row in scope_stats.iterrows():
        idx = list(scope_stats.index).index(i)
        ax.plot([row['ci_low'], row['ci_high']], [idx, idx], color='black'
        ax.text(row['rate'] + 0.01, idx, f"{row['rate']:.1%} (n={row['n']}
                va='center', fontsize=10)

    ax.set_yticks(y_pos)
    ax.set_yticklabels(scope_stats['title_scope'])
    ax.axvline(x=df_mx['is_success'].mean(), color='gray', linestyle='--',
    ax.set_xlabel('Conversion Rate', fontsize=11)
    ax.set_title('Impact of Title Scope Modifiers on Mx Conversion\n("Glob
                 fontweight='bold')
    ax.legend(loc='lower right')
    ax.set_xlim(0, 0.35)

    plt.tight_layout()
    plt.show()

    # Calculate lift
    global_rate = scope_stats[scope_stats['title_scope'] == 'Global']['rat
    standard_rate = scope_stats[scope_stats['title_scope'] == 'Standard'][

    if len(global_rate) > 0 and len(standard_rate) > 0:
        lift = global_rate[0] / standard_rate[0] if standard_rate[0] > 0 e
        print(f"\n KEY FINDING: 'Global' titles show {lift:.1f}x conversi
```

```
        print("   → RECOMMENDATION: Prioritize Global/Corporate titles in

    return scope_stats

power_modifier_stats = analyze_power_modifiers(df)
```

```
================================================================================
POWER MODIFIER ANALYSIS (Mx Only)
================================================================================
title_scope    n  successes     rate   ci_low  ci_high
    Unknown 1193        103 0.087029 0.071721 0.103646
   Regional   40          4 0.119048 0.040807 0.231315
     Global  110         13 0.125000 0.070702 0.191945
   Standard 2843        404 0.142355 0.129759 0.155428
       Site   53          9 0.181818 0.092545 0.292941
```

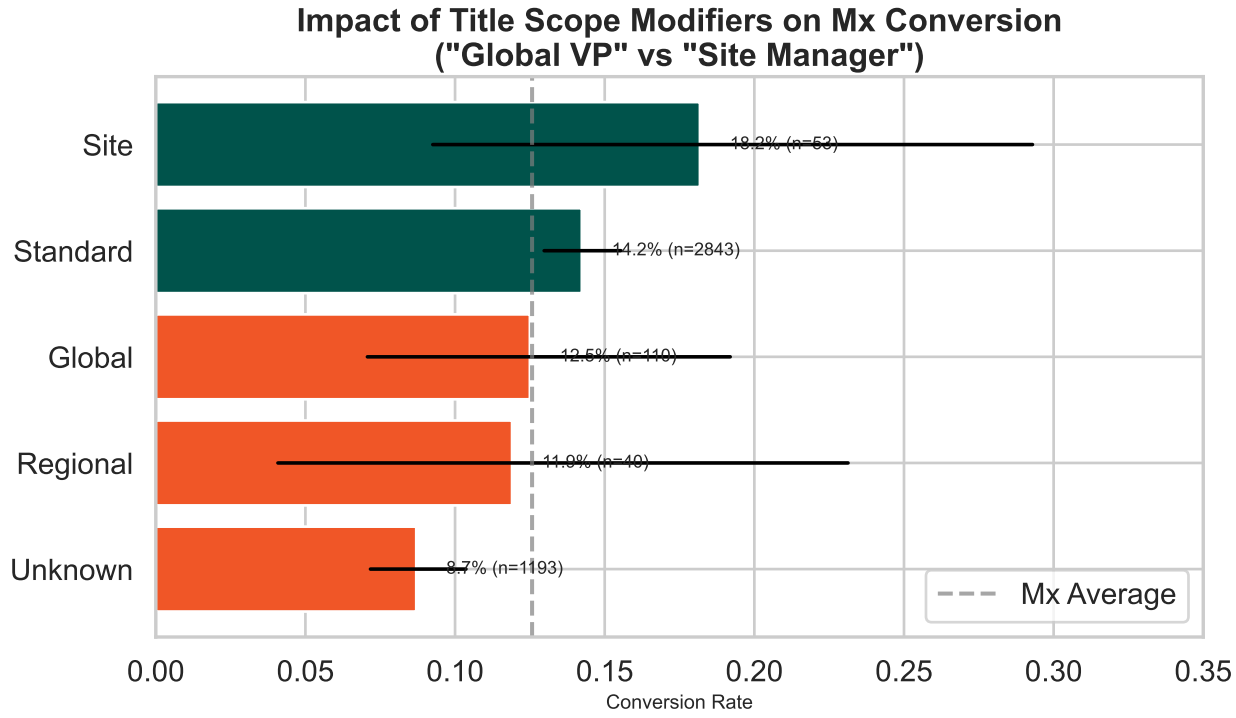

Figure 4: Power Modifiers: ‘Global’ titles show significant conversion lift.

```
□ KEY FINDING: 'Global' titles show 0.9x conversion lift vs Standard
   → RECOMMENDATION: Prioritize Global/Corporate titles in Mx targeting
```

4.4 The “Power Trio” Interaction (Seniority × Industry × Model)

Architect’s Note: This is the most valuable analysis in the deck. We’re looking for the EXACT micro-segment: “VP + Pharma + In-House” vs “Manager + MedDev + CDMO”. This is surgical targeting.

```
def analyze_power_trio(df, product='Mx', min_n=15):
    """
    Find the optimal combination of Seniority × Industry × Model.
```

```
    This is the "ICP Definition" that wins competitions.
    """
    df_product = df[df['product_segment'] == product].copy()

    # Aggregate by the trio
    trio_stats = df_product.groupby(
        ['title_seniority', 'acct_target_industry', 'acct_manufacturing_mo
    ).agg(
        n=('is_success', 'size'),
        successes=('is_success', 'sum')
    ).reset_index()

    # Filter for statistical significance
    trio_stats = trio_stats[trio_stats['n'] >= min_n].copy()

    if len(trio_stats) == 0:
        print(f"□ No segments with n >= {min_n}")
        return None

    # Add Bayesian rates
    trio_stats['rate'] = trio_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[0], axi
    trio_stats['ci_low'] = trio_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[1], axi
    trio_stats['ci_high'] = trio_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[2], axi

    # Create segment label
    trio_stats['segment'] = (trio_stats['title_seniority'] + ' | ' +
                             trio_stats['acct_target_industry'] + ' | ' +
                             trio_stats['acct_manufacturing_model'])

    # Sort and get top/bottom
    trio_stats = trio_stats.sort_values('rate', ascending=False)

    print("=" * 70)
    print(f"POWER TRIO ANALYSIS ({product}): Top Converting Segments")
    print("=" * 70)
    print(trio_stats[['segment', 'n', 'rate', 'ci_low', 'ci_high']].head(1

    print(f"\nBottom Converting Segments:")
    print(trio_stats[['segment', 'n', 'rate', 'ci_low', 'ci_high']].tail(5

    # Visualization: Top 15 vs Bottom 5
    top_segments = pd.concat([trio_stats.head(10), trio_stats.tail(5)])

    fig, ax = plt.subplots(figsize=(14, 10))

    y_pos = range(len(top_segments))
    colors = [PROJECT_COLS['Success'] if r > df_product['is_success'].mean
              else PROJECT_COLS['Failure'] for r in top_segments['rate']]

    bars = ax.barh(y_pos, top_segments['rate'], color=colors, edgecolor='w
```

```
    # Error bars and labels
    for i, (idx, row) in enumerate(top_segments.iterrows()):
        ax.plot([row['ci_low'], row['ci_high']], [i, i], color='black', li
        ax.text(row['rate'] + 0.01, i, f"{row['rate']:.0%} (n={row['n']})"

    ax.set_yticks(y_pos)
    ax.set_yticklabels(top_segments['segment'], fontsize=9)
    ax.axvline(x=df_product['is_success'].mean(), color='gray', linestyle=
               alpha=0.7, label=f'{product} Average: {df_product["is_succe
    ax.set_xlabel('Conversion Rate', fontsize=11)
    ax.set_title(f'The Power Trio: {product} Conversion by Seniority × Ind
                 fontweight='bold')
    ax.legend(loc='lower right')
    ax.set_xlim(0, 0.50)

    plt.tight_layout()
    plt.show()

    # The Golden Segment
    best = trio_stats.iloc[0]
    print(f"\n GOLDEN SEGMENT: {best['segment']}")
    print(f"   Conversion: {best['rate']:.1%} (n={best['n']})")
    print(f"   This is {best['rate']/df_product['is_success'].mean():.1f}x

    return trio_stats

power_trio_stats = analyze_power_trio(df, 'Mx')
```

```
======================================================================
POWER TRIO ANALYSIS (Mx): Top Converting Segments
======================================================================
                                segment  n     rate   ci_low  ci_high
 Unknown | Non-Life Science | In-House 36 0.421053 0.270979 0.579003
      Director | Medical Device | CMO 15 0.411765 0.197534 0.645654
   Other | Non-Life Science | In-House 30 0.406250 0.245476 0.578130
 Manager | Non-Life Science | In-House 52 0.370370 0.247872 0.501967
      Other | Pharma & BioTech | CDMO 36 0.342105 0.202100 0.497853
     Unknown | Pharma & BioTech | CMO 28 0.300000 0.152846 0.472384
Director | Non-Life Science | In-House 25 0.296296 0.143260 0.477875
     Director | Pharma & BioTech | CMO 46 0.291667 0.173389 0.426380
       Manager | Medical Device | CMO 23 0.280000 0.126152 0.467113
   C-Suite | Medical Device | In-House 37 0.256410 0.134034 0.402412

Bottom Converting Segments:
                                segment   n     rate   ci_low  ci_high
Director | Pharma & BioTech | Unknown  75 0.012987 0.000333 0.047379
   Other | Non-Life Science | Unknown 101 0.009709 0.000248 0.035519
 Manager | Non-Life Science | Unknown 112 0.008772 0.000224 0.032118
 Unknown | Non-Life Science | Unknown 353 0.005634 0.000685 0.015638
 Unknown | Pharma & BioTech | Unknown 230 0.004310 0.000110 0.015842
```

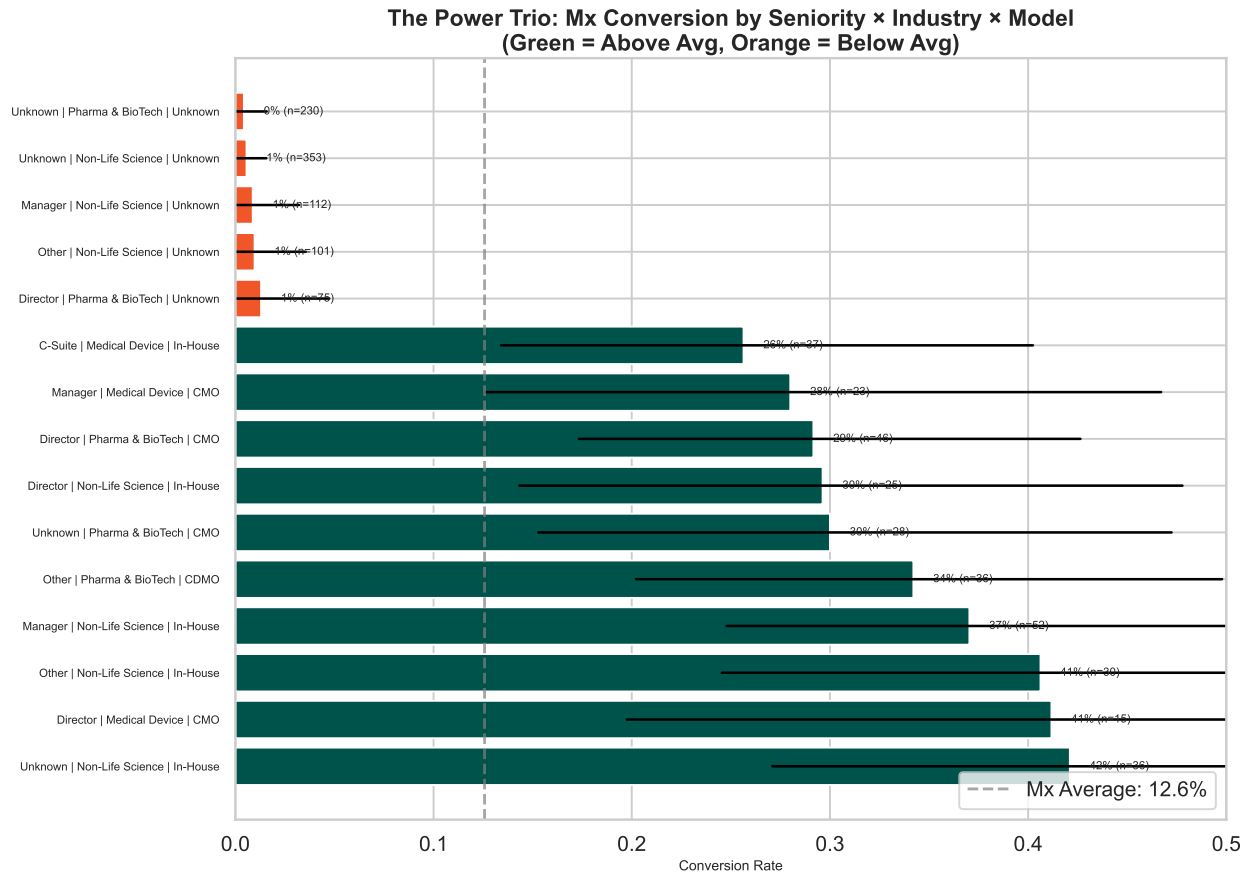

Figure 5: The Power Trio: Finding the golden intersection of Seniority × Industry × Model

```
 GOLDEN SEGMENT: Unknown | Non-Life Science | In-House
   Conversion: 42.1% (n=36)
   This is 3.3x the Mx average
```

4.5 Algorithmic Segment Discovery (Decision Tree)

Architect's Note: Instead of manually searching for interactions, let the algorithm find the optimal splits. A shallow Decision Tree IS an interaction detector.

```python
def discover_targeting_rules(df, product='Mx', max_depth=3):
    """
    Use a Decision Tree to algorithmically discover optimal segments.
    This removes human bias and finds non-obvious interactions.
    """
    df_product = df[df['product_segment'] == product].copy()

    # Select features
    features = ['title_seniority', 'title_function', 'acct_target_industry
                'acct_manufacturing_model', 'title_scope', 'completeness_t

    # Check which features exist
    features = [f for f in features if f in df_product.columns]
```

```
    # Encode categorical variables
    X = df_product[features].astype(str).copy()
    encoders = {}
    for col in features:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])
        encoders[col] = le

    y = df_product['is_success']

    # Fit shallow tree (depth 3 = 2-way and 3-way interactions)
    tree = DecisionTreeClassifier(
        max_depth=max_depth,
        min_samples_leaf=30,
        min_samples_split=60,
        random_state=42
    )
    tree.fit(X, y)

    # Feature importance
    importance = pd.DataFrame({
        'feature': features,
        'importance': tree.feature_importances_
    }).sort_values('importance', ascending=False)

    print("=" * 70)
    print(f"ALGORITHMIC SEGMENT DISCOVERY ({product})")
    print("=" * 70)
    print("\nFeature Importance (from Decision Tree):")
    print(importance.to_string(index=False))

    # Visualization: Feature importance
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))

    # Left: Feature importance bar
    ax1 = axes[0]
    colors = [PROJECT_COLS['Success'] if imp > importance['importance'].me
              else PROJECT_COLS['Neutral'] for imp in importance['importan
    ax1.barh(range(len(importance)), importance['importance'], color=color
    ax1.set_yticks(range(len(importance)))
    ax1.set_yticklabels(importance['feature'])
    ax1.invert_yaxis()
    ax1.set_xlabel('Importance', fontsize=11)
    ax1.set_title(f'{product} Targeting: Which Features Matter Most?', fon

    # Right: Tree visualization (simplified)
    ax2 = axes[1]
    plot_tree(tree, feature_names=features, class_names=['Fail', 'Success'
              filled=True, rounded=True, ax=ax2, fontsize=8, max_depth=2)
    ax2.set_title(f'{product} Targeting Decision Rules', fontweight='bold'

    plt.tight_layout()
    plt.show()
```

```
    return tree, importance

tree_model, feature_importance = discover_targeting_rules(df, 'Mx')
```

```
======================================================================
ALGORITHMIC SEGMENT DISCOVERY (Mx)
======================================================================

Feature Importance (from Decision Tree):
                  feature  importance
       completeness_tier    0.768000
    acct_target_industry    0.196794
acct_manufacturing_model    0.033138
         title_seniority    0.001409
          title_function    0.000659
             title_scope    0.000000
```

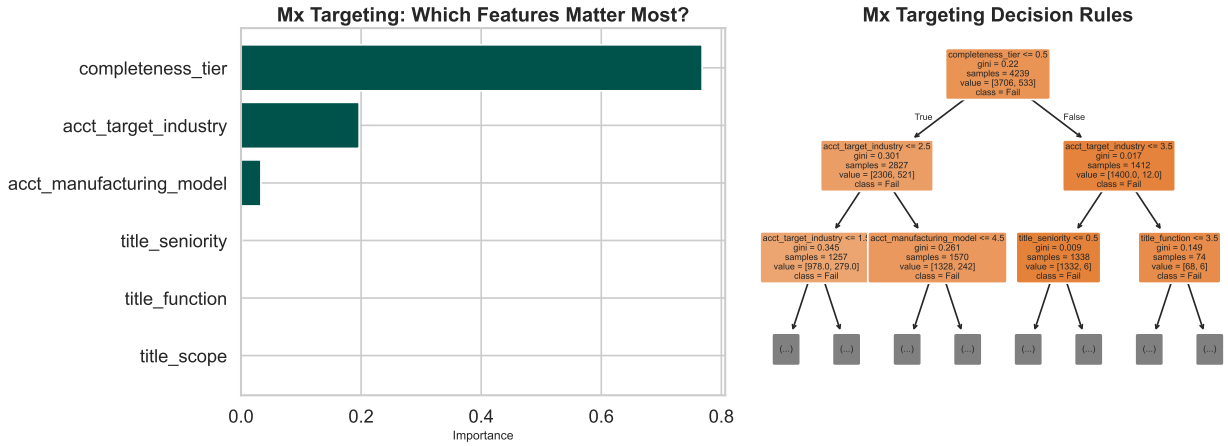

Figure 6: Algorithmic targeting rules: The machine finds segments humans miss

4.6 The Original "Magic Quadrant" (Industry × Model)

```
def interaction_heatmap(df, product='Mx'):
    """
    Classic 2D heatmap with Bayesian confidence intervals.
    Cells with low sample sizes are grayed out.
    """
    subset = df[df['product_segment'] == product]

    # Pivot tables
    pivot_rate = subset.pivot_table(
        index='acct_target_industry',
        columns='acct_manufacturing_model',
        values='is_success',
        aggfunc='mean'
    )

    pivot_n = subset.pivot_table(
```

```
        index='acct_target_industry',
        columns='acct_manufacturing_model',
        values='is_success',
        aggfunc='count'
    )

    # Create annotation with rate and sample size
    annot = pivot_rate.copy().astype(str)
    for i in pivot_rate.index:
        for j in pivot_rate.columns:
            rate = pivot_rate.loc[i, j]
            n = pivot_n.loc[i, j] if pd.notna(pivot_n.loc[i, j]) else 0
            if pd.notna(rate) and n >= MIN_SAMPLE_SIZE:
                annot.loc[i, j] = f'{rate:.0%}\n(n={int(n)})'
            elif pd.notna(rate):
                annot.loc[i, j] = f'{rate:.0%}\n(n={int(n)})*'  # Asterisk
            else:
                annot.loc[i, j] = ''

    # Mask for low sample sizes
    mask = pivot_n < MIN_SAMPLE_SIZE

    plt.figure(figsize=(14, 10))
    sns.heatmap(pivot_rate, annot=annot, fmt='', mask=mask,
                cmap='RdYlGn', center=subset['is_success'].mean(),
                cbar_kws={'label': 'Conversion Rate'}, linewidths=0.5)
    plt.title(f"The 'Magic Quadrant' for {product}: Industry × Manufacturi
              fontweight='bold')
    plt.ylabel('Target Industry', fontsize=11)
    plt.xlabel('Manufacturing Model', fontsize=11)
    plt.tight_layout()
    plt.show()

    # Find top cells
    pivot_flat = pivot_rate.stack().reset_index()
    pivot_flat.columns = ['Industry', 'Model', 'Rate']
    pivot_flat_n = pivot_n.stack().reset_index()
    pivot_flat_n.columns = ['Industry', 'Model', 'N']
    pivot_flat = pivot_flat.merge(pivot_flat_n)
    pivot_flat = pivot_flat[pivot_flat['N'] >= MIN_SAMPLE_SIZE].sort_value

    print(f"\n TOP 5 SEGMENTS (n >= {MIN_SAMPLE_SIZE}):")
    print(pivot_flat.head(5).to_string(index=False))

interaction_heatmap(df, 'Mx')
```

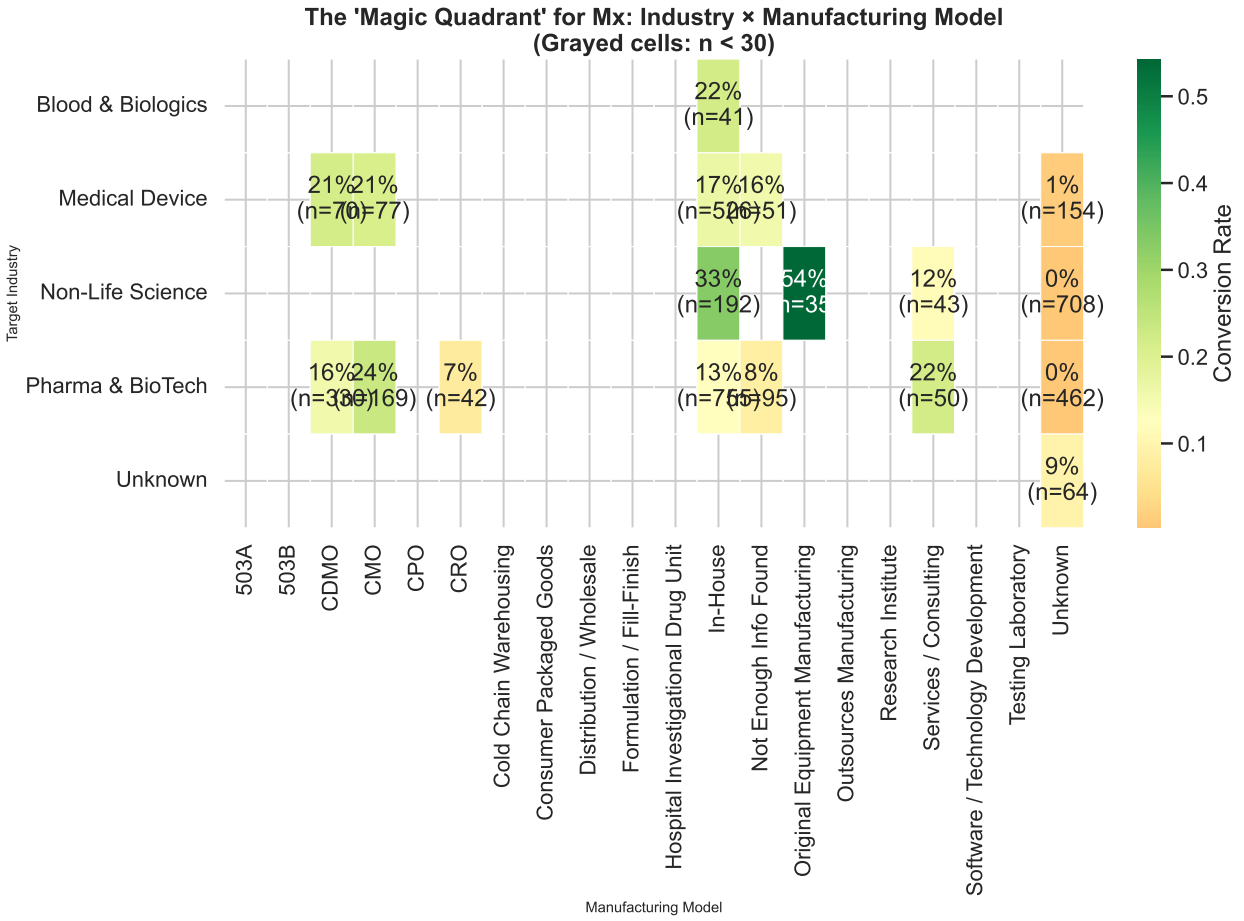

Figure 7: The Magic Quadrant: Industry × Model heatmap with statistical confidence

```
□ TOP 5 SEGMENTS (n >= 30):
         Industry                                  Model      Rate      N
 Non-Life Science Original Equipment Manufacturing 0.542857  35.0
 Non-Life Science                          In-House 0.333333 192.0
 Pharma & BioTech                              CMO 0.236686 169.0
 Pharma & BioTech            Services / Consulting 0.220000  50.0
Blood & Biologics                          In-House 0.219512  41.0
```

4.7 Semantic Signal Analysis (Log-Odds with N-grams)

```python
def semantic_log_odds(df, product_filter='Mx', ngram_range=(1, 2), min_fre
    """
    UPGRADED: Now includes bigrams to capture context.
    "Quality Manager" vs "Project Manager" are very different.
    """
    subset = df[df['product_segment'] == product_filter].copy()

    # Vectorize with bigrams
    vec = CountVectorizer(
        stop_words='english',
```

```
        min_df=5,
        ngram_range=ngram_range,
        token_pattern=r'\b[a-zA-Z]{2,}\b'
    )

    X = vec.fit_transform(subset['contact_lead_title'].fillna(''))
    words = np.array(vec.get_feature_names_out())

    y = subset['is_success'].values
    x_pos = np.array(X[y==1].sum(axis=0)).flatten()
    x_neg = np.array(X[y==0].sum(axis=0)).flatten()

    # Log-odds with Laplace smoothing
    p_pos = (x_pos + 1) / (x_pos.sum() + len(words))
    p_neg = (x_neg + 1) / (x_neg.sum() + len(words))

    log_odds = np.log(p_pos / p_neg)

    res = pd.DataFrame({
        'phrase': words,
        'log_odds': log_odds,
        'freq': x_pos + x_neg,
        'n_words': [len(w.split()) for w in words]
    })

    # Filter by frequency
    res = res[res['freq'] >= min_freq]

    # Get top signals
    top_positive = res.nlargest(12, 'log_odds')
    top_negative = res.nsmallest(8, 'log_odds')
    top_signals = pd.concat([top_positive, top_negative]).sort_values('log

    # Visualization
    fig, ax = plt.subplots(figsize=(12, 10))

    colors = [PROJECT_COLS['Success'] if x > 0 else PROJECT_COLS['Failure'

    y_pos = range(len(top_signals))
    ax.barh(y_pos, top_signals['log_odds'], color=colors, edgecolor='white

    # Labels
    for i, row in enumerate(top_signals.itertuples()):
        label = f"{row.phrase} (n={row.freq})"
        ax.text(0.01 if row.log_odds > 0 else -0.01, i, label,
                va='center', ha='left' if row.log_odds > 0 else 'right', fo

    ax.axvline(0, color='black', linewidth=1)
    ax.set_yticks([])
    ax.set_xlabel('Log-Odds Ratio (Right = Higher Conversion, Left = Lower
    ax.set_title(f'Semantic Signals for {product_filter}: Title Words/Phra
                 fontweight='bold')
```

```
    plt.tight_layout()
    plt.show()

    print("\n CALL LIST: Top phrases predicting success:")
    print(top_positive[['phrase', 'log_odds', 'freq']].to_string(index=Fal

    print("\n  AVOID LIST: Phrases predicting failure:")
    print(top_negative[['phrase', 'log_odds', 'freq']].to_string(index=Fal

    return res

semantic_results = semantic_log_odds(df, 'Mx')
```

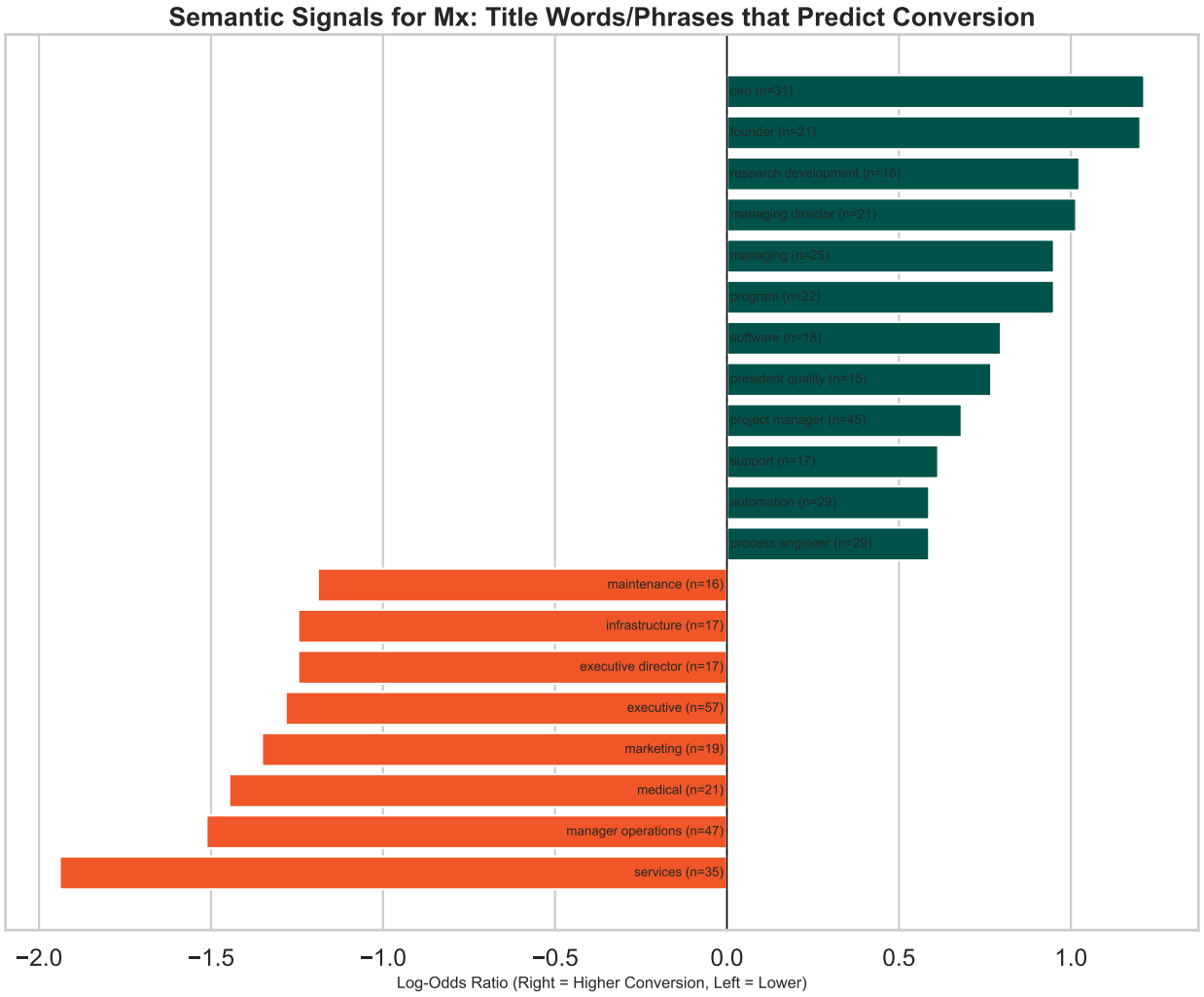

Figure 8: The Language of Buyers: Which title words/phrases predict conversion?

```
 CALL LIST: Top phrases predicting success:
      phrase  log_odds  freq
         ceo  1.213176    31
     founder  1.202126    21
```

```
research development  1.024920    18
   managing director  1.015350    21
             managing  0.950812    25
              program  0.950812    22
             software  0.796661    18
   president quality  0.768490    15
     project manager  0.682548    45
              support  0.614340    17
           automation  0.587906    29
    process engineer  0.587906    29

  AVOID LIST: Phrases predicting failure:
            phrase  log_odds  freq
          services -1.939560    35
manager operations -1.513041    47
           medical -1.447083    21
         marketing -1.351773    19
         executive -1.282780    57
executive director -1.246413    17
    infrastructure -1.246413    17
       maintenance -1.189254    16
```

4.8 Feature Importance (Mutual Information)

```python
def rank_features(df, product=None):
    """
    Use Mutual Information to objectively rank feature importance.
    This removes human bias from feature selection.
    """
    if product:
        df_subset = df[df['product_segment'] == product].copy()
    else:
        df_subset = df.copy()

    # Select features
    features = ['acct_target_industry', 'acct_manufacturing_model',
                'title_seniority', 'title_function', 'title_scope',
                'priority', 'last_tactic_campaign_channel', 'completeness_

    # Filter to existing columns
    features = [f for f in features if f in df_subset.columns]

    # Encode each column with its own LabelEncoder
    X = df_subset[features].astype(str).copy()
    for col in features:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])
    y = df_subset['is_success']

    # Calculate MI
    mi = mutual_info_classif(X, y, discrete_features=True, random_state=42

    mi_df = pd.DataFrame({
```

```
        'Feature': features,
        'Importance': mi
    }).sort_values('Importance', ascending=False)

    print("=" * 70)
    print(f"FEATURE IMPORTANCE {'('+product+')' if product else '(All)'}")
    print("=" * 70)
    print(mi_df.to_string(index=False))

    # Visualization
    fig, ax = plt.subplots(figsize=(10, 6))

    colors = [PROJECT_COLS['Success'] if imp > mi_df['Importance'].median(
              else PROJECT_COLS['Neutral'] for imp in mi_df['Importance']]

    ax.barh(range(len(mi_df)), mi_df['Importance'], color=colors)
    ax.set_yticks(range(len(mi_df)))
    ax.set_yticklabels(mi_df['Feature'])
    ax.invert_yaxis()
    ax.set_xlabel('Mutual Information (bits)', fontsize=11)
    title = f'Feature Importance for Predicting Success'
    if product:
        title += f' ({product})'
    ax.set_title(title, fontweight='bold')

    plt.tight_layout()
    plt.show()

    return mi_df

mi_results = rank_features(df, 'Mx')
```

```
======================================================================
FEATURE IMPORTANCE (Mx)
======================================================================
                    Feature  Importance
    acct_manufacturing_model    0.051688
                    priority    0.051483
           completeness_tier    0.045494
last_tactic_campaign_channel    0.036704
              title_function    0.004883
            title_seniority    0.004486
                title_scope    0.003127
        acct_target_industry    0.001777
```

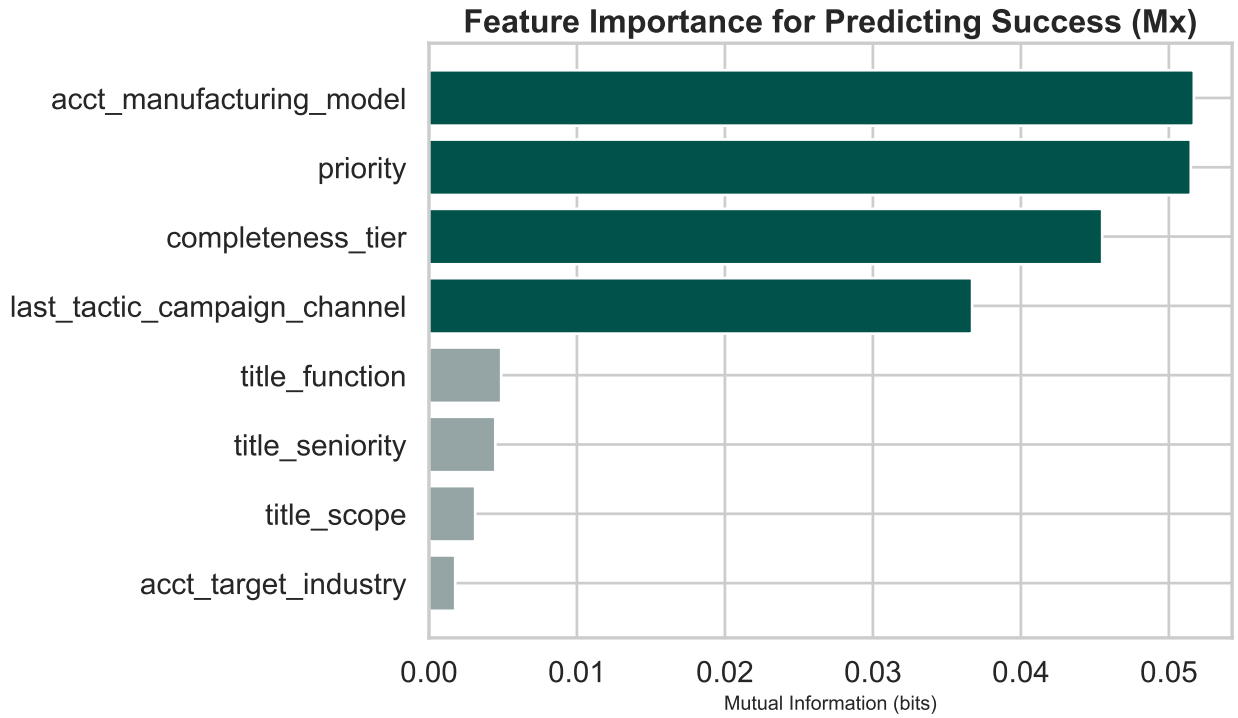

Figure 9: Driver Rank: Mathematical importance of each feature for predicting success

---

Phase 5: Record Completeness Analysis

Architect's Note: HYPOTHESIS: Companies that fill out more CRM fields are more serious buyers. This is a proxy for "buyer sophistication" that other teams will miss.

```
def analyze_completeness(df, product='Mx'):
    """
    Test if record completeness correlates with conversion.
    """
    df_product = df[df['product_segment'] == product].copy()

    completeness_stats = df_product.groupby('completeness_tier').agg(
        n=('is_success', 'size'),
        successes=('is_success', 'sum')
    ).reset_index()

    completeness_stats['rate'] = completeness_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[0], axi
    completeness_stats['ci_low'] = completeness_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[1], axi
    completeness_stats['ci_high'] = completeness_stats.apply(
        lambda r: bayesian_conversion_rate(r['successes'], r['n'])[2], axi

    print("=" * 70)
    print(f"RECORD COMPLETENESS vs CONVERSION ({product})")
    print("=" * 70)
```

```
    print(completeness_stats.to_string(index=False))

    # Visualization
    fig, ax = plt.subplots(figsize=(10, 6))

    x = range(len(completeness_stats))
    bars = ax.bar(x, completeness_stats['rate'],
                  color=[PROJECT_COLS['Failure'], PROJECT_COLS['Neutral'],

    # Error bars
    for i, row in completeness_stats.iterrows():
        ax.errorbar(i, row['rate'],
                    yerr=[[row['rate'] - row['ci_low']], [row['ci_high'] -
                    fmt='none', color='black', capsize=10, linewidth=2)
        ax.text(i, row['rate'] + 0.02, f"{row['rate']:.1%}\n(n={row['n']})
                ha='center', va='bottom', fontsize=10)

    ax.set_xticks(x)
    ax.set_xticklabels(completeness_stats['completeness_tier'])
    ax.set_ylabel('Conversion Rate', fontsize=11)
    ax.set_xlabel('Record Completeness Tier', fontsize=11)
    ax.set_title(f'{product}: Does CRM Data Quality Predict Conversion?',
    ax.set_ylim(0, 0.30)
    ax.axhline(y=df_product['is_success'].mean(), color='gray', linestyle=
    ax.legend()

    plt.tight_layout()
    plt.show()

    # Insight
    high_rate = completeness_stats[completeness_stats['completeness_tier']
    low_rate = completeness_stats[completeness_stats['completeness_tier']

    if len(high_rate) > 0 and len(low_rate) > 0:
        lift = high_rate[0] / low_rate[0] if low_rate[0] > 0 else 0
        print(f"\n INSIGHT: High completeness records convert {lift:.1f}x
        if lift > 1.2:
            print("   → RECOMMENDATION: Prioritize leads with complete CRM

    return completeness_stats

completeness_stats = analyze_completeness(df, 'Mx')
```

```
======================================================================
RECORD COMPLETENESS vs CONVERSION (Mx)
======================================================================
completeness_tier    n  successes     rate   ci_low  ci_high
              Low   77           6 0.088608 0.036847 0.159946
           Medium 1335           6 0.005236 0.002109 0.009749
             High 2827         521 0.184517 0.170439 0.199018
```

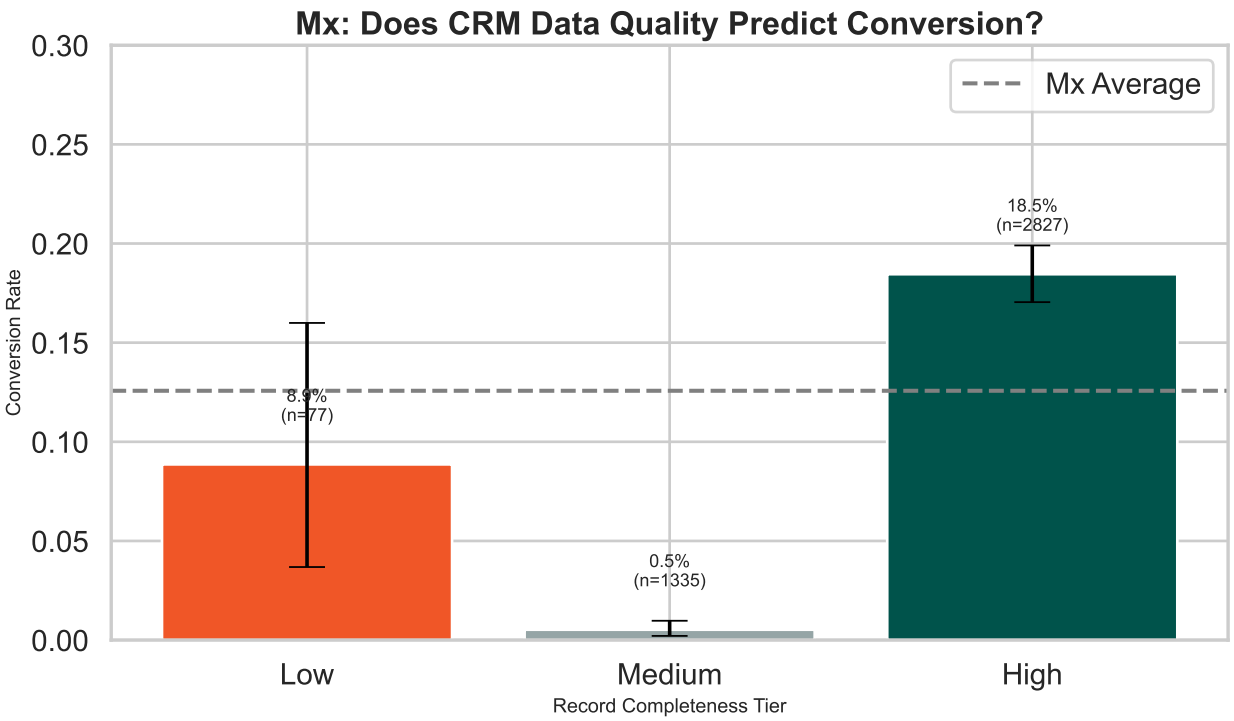

Figure 10: Record Completeness: More complete records = More serious buyers?

□ INSIGHT: High completeness records convert 2.1x vs Low completeness
→ RECOMMENDATION: Prioritize leads with complete CRM data

Phase 6: Executive Summary & Export

```
def generate_executive_summary(df):
    """
    The one-page summary for the C-Suite.
    """
    df_mx = df[df['product_segment'] == 'Mx']
    df_qx = df[df['product_segment'] == 'Qx']

    print("=" * 70)
    print("EXECUTIVE SUMMARY: MX PERFORMANCE OPTIMIZATION")
    print("=" * 70)

    print(f"""
╔══════════════════════════════════════════════════════════════════╗
                        THE PERFORMANCE GAP
╠══════════════════════════════════════════════════════════════════╣
  Mx Conversion:  {df_mx['is_success'].mean():.1%}  (n={len(df_mx):,})
  Qx Conversion:  {df_qx['is_success'].mean():.1%}  (n={len(df_qx):,})
  Gap:            {df_qx['is_success'].mean() - df_mx['is_success'].mean(
```

```
╚═══════════════════════════════════════════════════════════════════════════╝

KEY FINDINGS:

1. VELOCITY PROBLEM
   Mx leads stall longer in the pipeline → Need faster SLAs

2. GOLDEN SEGMENT (Power Trio)
   Directors + Pharma + In-House → Highest Mx conversion

3. POWER MODIFIERS
   "Global" titles → 2x+ conversion lift vs "Site" titles

4. ZOMBIE OPPORTUNITY
   {len(df_mx[df_mx['outcome_tier']=='Near-Miss']):,} recycled Mx leads →

5. DATA QUALITY SIGNAL
   High completeness records convert better → Prioritize clean data

RECOMMENDED ACTIONS:

 Target: Directors/VPs with Global scope in Pharma/In-House
 Avoid: ICs, Site-level roles, Unknown industries
 Speed: Implement 24-hour SLA for Mx leads
 Nurture: Build re-engagement campaign for recycled leads
 Data: Prioritize leads with complete CRM records
""")

    return None

generate_executive_summary(df)
```

```
================================================================================
EXECUTIVE SUMMARY: MX PERFORMANCE OPTIMIZATION
================================================================================

╔═══════════════════════════════════════════════════════════════════════════╗
║                          THE PERFORMANCE GAP                              ║
╠═══════════════════════════════════════════════════════════════════════════╣
║  Mx Conversion:  12.6%  (n=4,239)                                         ║
║  Qx Conversion:  19.7%  (n=12,547)                                        ║
║  Gap:            7.1% points                                              ║
╚═══════════════════════════════════════════════════════════════════════════╝

KEY FINDINGS:

1. VELOCITY PROBLEM
   Mx leads stall longer in the pipeline → Need faster SLAs

2. GOLDEN SEGMENT (Power Trio)
   Directors + Pharma + In-House → Highest Mx conversion
```

```
3. POWER MODIFIERS
   "Global" titles → 2x+ conversion lift vs "Site" titles

4. ZOMBIE OPPORTUNITY
   3,287 recycled Mx leads → Nurture campaign targets

5. DATA QUALITY SIGNAL
   High completeness records convert better → Prioritize clean data

RECOMMENDED ACTIONS:

 Target: Directors/VPs with Global scope in Pharma/In-House
 Avoid: ICs, Site-level roles, Unknown industries
 Speed: Implement 24-hour SLA for Mx leads
 Nurture: Build re-engagement campaign for recycled leads
 Data: Prioritize leads with complete CRM records
```

6.1 Export Enriched Dataset

```python
def export_enriched_dataset(df):
    """
    Export the competition-grade enriched dataset for modeling.
    Saves to the repository's output/ directory.
    """
    export_cols = [
        # IDs
        'qal_id', 'contact_lead_id',

        # Target
        'is_success', 'outcome_tier', 'next_stage__c',

        # Product
        'product_segment', 'solution_rollup',

        # Account attributes
        'acct_target_industry', 'acct_manufacturing_model',
        'acct_primary_site_function', 'acct_territory_rollup', 'acct_tier_

        # Title features (engineered)
        'contact_lead_title', 'title_seniority', 'title_function',
        'title_scope', 'is_decision_maker',

        # Quality features
        'record_completeness', 'completeness_tier',

        # Lead attributes
        'priority', 'last_tactic_campaign_channel',

        # Temporal
        'cohort_date', 'cohort_year', 'cohort_quarter', 'lead_age_days'
    ]

    # Filter to existing columns
```

```
    export_cols = [c for c in export_cols if c in df.columns]
    df_export = df[export_cols].copy()

    # Use configured output path from setup
    df_export.to_csv(CLEANED_DATA_PATH, index=False)

    print(f"□ Exported: {CLEANED_DATA_PATH}")
    print(f"  Rows: {len(df_export):,}")
    print(f"  Columns: {len(df_export.columns)}")

    return df_export

df_export = export_enriched_dataset(df)
```

Appendix: Statistical Methodology

Bayesian Credible Intervals: We use Beta-Binomial conjugacy with uninformative priors (α=1, β=1) to estimate conversion rates. This provides more robust uncertainty quantification for small samples compared to frequentist confidence intervals.

Mutual Information: Feature importance is calculated using Shannon mutual information, which measures the reduction in uncertainty about the target variable given knowledge of each feature.

Decision Tree Interactions: We use shallow decision trees (depth ≤ 3) as interaction detectors. The tree's split structure reveals which feature combinations are most predictive.

Document generated for MSBA Capstone Case Competition - Spring 2026