The Revenue Engine: MasterControl Predictive Modeling

Quantifying the Sales Differential for Mx Lead Prioritization

MSBA Capstone Group 3

2026-01-01

Table of contents

1 Executive Summary

The Mission: Transform EDA insights into a production-grade lead scoring model that quantifies the exact revenue lift MasterControl can achieve through intelligent lead prioritization.

Key Deliverables:

1. Four-model tournament (Logistic, Random Forest, XGBoost, Neural Network)
2. Business Lift Calculation: "How many additional SQLs per 1000 leads?"
3. SHAP-based interpretability for Sales enablement
4. Production-ready scoring pipeline

---

2 Phase 1: Production Environment Setup

Architect's Note: We use `joblib` for parallelization across all cores. Global seeds ensure reproducibility across all stochastic processes.

```
# ==============================================================================
# PRODUCTION ENVIRONMENT CONFIGURATION
# ==============================================================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import warnings
from pathlib import Path
from datetime import datetime

# Scikit-learn Core
from sklearn.model_selection import (
    train_test_split, cross_val_score, StratifiedKFold,
    GridSearchCV, cross_val_predict
)
from sklearn.preprocessing import (
    StandardScaler, OneHotEncoder, LabelEncoder,
    FunctionTransformer
)
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

# XGBoost
try:
    import xgboost as xgb
    XGBOOST_AVAILABLE = True
except ImportError:
    XGBOOST_AVAILABLE = False
    print("Warning: XGBoost not available. Will skip XGB model.")

# Metrics
from sklearn.metrics import (
    roc_auc_score, precision_recall_curve, auc,
    log_loss, classification_report, confusion_matrix,
    roc_curve, precision_score, recall_score, f1_score
)

# Interpretability
try:
    import shap
```

```
    SHAP_AVAILABLE = True
except ImportError:
    SHAP_AVAILABLE = False
    print("Warning: SHAP not available. Will skip SHAP analysis.")

# Parallelization
from joblib import parallel_backend, Parallel, delayed
import multiprocessing

# ============================================================================
# GLOBAL CONFIGURATION
# ============================================================================

RANDOM_STATE = 42
N_JOBS = -1  # Use all available cores
CV_FOLDS = 5
TEST_SIZE = 0.2

np.random.seed(RANDOM_STATE)

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

# Plotting configuration
sns.set_theme(style="whitegrid", context="talk")
plt.rcParams['figure.figsize'] = (14, 8)
plt.rcParams['axes.titleweight'] = 'bold'

warnings.filterwarnings('ignore')

print("=" * 70)
print("PRODUCTION MODELING ENVIRONMENT")
print("=" * 70)
print(f"  Random State: {RANDOM_STATE}")
print(f"  CPU Cores Available: {multiprocessing.cpu_count()}")
print(f"  Cross-Validation Folds: {CV_FOLDS}")
print(f"  XGBoost Available: {XGBOOST_AVAILABLE}")
print(f"  SHAP Available: {SHAP_AVAILABLE}")
print("=" * 70)
```

```
Warning: XGBoost not available. Will skip XGB model.
Warning: SHAP not available. Will skip SHAP analysis.
======================================================================
PRODUCTION MODELING ENVIRONMENT
======================================================================
  Random State: 42
  CPU Cores Available: 24
  Cross-Validation Folds: 5
  XGBoost Available: False
```

```
  SHAP Available: False
======================================================================
```

---

3 Phase 2: Data Loading & Feature Engineering

Architect's Note: We rebuild the features from the EDA phase programmatically to ensure the pipeline is self-contained and reproducible.

```python
# ============================================================================
# DATA LOADING WITH ROBUST PATH DETECTION
# ============================================================================

def load_data():
    """Load raw data with intelligent path detection."""
    possible_paths = [
        Path.cwd() / "data" / "QAL Performance for MSBA.csv",
        Path.cwd().parent / "data" / "QAL Performance for MSBA.csv",
        Path.cwd().parent.parent / "data" / "QAL Performance for MSBA.csv",
        Path.cwd().parent.parent.parent / "data" / "QAL Performance for MSBA.csv"
    ]

    for p in possible_paths:
        if p.exists():
            df = pd.read_csv(p)
            print(f"  Data loaded from: {p}")
            return df

    raise FileNotFoundError("Could not find QAL Performance for MSBA.csv")

df_raw = load_data()

# Standardize column names
df_raw.columns = [c.strip().lower().replace(' ', '_').replace('/', '_').replace('-', '_')
                  for c in df_raw.columns]

print(f"  Raw Data Shape: {df_raw.shape}")
```

```
  Data loaded from: C:\Users\thoma\Repos\MSBA-Capstone-MasterControl-GroupProject\data\QAL Performance for
  Raw Data Shape: (16815, 14)
```

```python
# ============================================================================
# FEATURE ENGINEERING PIPELINE (From EDA Insights)
# ============================================================================

def engineer_features(df):
    """
    Apply all feature engineering discovered in EDA phase.
    This creates a reproducible transformation pipeline.
    """
    df = df.copy()

    # ------------------------------------------------------------------------
    # 1. TARGET VARIABLE
```

```
# ------------------------------------------------------------------------
success_stages = ['SQL', 'SQO', 'Won']
df['is_success'] = df['next_stage__c'].isin(success_stages).astype(int)

# ------------------------------------------------------------------------
# 2. 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)

# ------------------------------------------------------------------------
# 3. TITLE PARSING (The Alpha Features from EDA)
# ------------------------------------------------------------------------
def parse_seniority(t):
    if pd.isna(t): return 'Unknown'
    t = str(t).lower()
    if re.search(r'\b(ceo|cfo|coo|cto|cio|chief|c-level|president|founder|owner)\b', t):
        return 'C-Suite'
    if re.search(r'\b(svp|senior vice president|evp)\b', t):
        return 'SVP'
    if re.search(r'\b(vp|vice president|head of)\b', t):
        return 'VP'
    if re.search(r'\b(director)\b', t):
        return 'Director'
    if re.search(r'\b(manager|mgr|lead|supervisor)\b', t):
        return 'Manager'
    if re.search(r'\b(analyst|engineer|specialist|associate|coordinator)\b', t):
        return 'IC'
    return 'Other'

def parse_function(t):
    if pd.isna(t): return 'Unknown'
    t = str(t).lower()
    if re.search(r'\b(manuf|prod|ops|plant|supply|site)\b', t):
        return 'Manufacturing_Ops'
    if re.search(r'\b(quality|qa|qc|qms|compliance|validation|capa)\b', t):
        return 'Quality_Reg'
    if re.search(r'\b(regulatory|reg affairs|submissions)\b', t):
        return 'Regulatory'
    if re.search(r'\b(it|info|sys|tech|data|soft)\b', t):
        return 'IT_Systems'
    if re.search(r'\b(lab|r&d|sci|dev|clin|research)\b', t):
        return 'R_D_Lab'
    return 'Other'

def parse_scope(t):
    if pd.isna(t): return 'Standard'
    t = str(t).lower()
    if re.search(r'\b(global|worldwide|international|corporate|enterprise|group)\b', t):
        return 'Global'
```

```
    if re.search(r'\b(regional|division)\b', t):
        return 'Regional'
    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_scope)
df['is_decision_maker'] = df['title_seniority'].isin(
    ['C-Suite', 'SVP', 'VP', 'Director']
).astype(int)

# ------------------------------------------------------------------------
# 4. INTERACTION FEATURE: The "Magic Quadrant" Effect
# ------------------------------------------------------------------------
# This was the strongest signal in EDA
df['industry_model_interaction'] = (
    df['acct_target_industry'].fillna('Unknown').astype(str) + '_X_' +
    df['acct_manufacturing_model'].fillna('Unknown').astype(str)
)

# ------------------------------------------------------------------------
# 5. RECORD COMPLETENESS (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(row[col]).lower() not in ['unknown', 'nan', ''])
    return filled / len(completeness_cols)

df['record_completeness'] = df.apply(calc_completeness, axis=1)

# ------------------------------------------------------------------------
# 6. TEMPORAL FEATURES
# ------------------------------------------------------------------------
df['cohort_date'] = pd.to_datetime(df['qal_cohort_date'], errors='coerce')
df['cohort_quarter'] = df['cohort_date'].dt.quarter.fillna(0).astype(int)
df['cohort_month'] = df['cohort_date'].dt.month.fillna(0).astype(int)

# ------------------------------------------------------------------------
# 7. IMPUTATION
# ------------------------------------------------------------------------
fill_cols = ['acct_target_industry', 'acct_manufacturing_model',
             'acct_territory_rollup', 'acct_primary_site_function']
for col in fill_cols:
    if col in df.columns:
        df[col] = df[col].fillna('Unknown')
```

```
    # Fill title if missing
    df['contact_lead_title'] = df['contact_lead_title'].fillna('Unknown Title')

    return df

# Apply feature engineering
df = engineer_features(df_raw)

print("=" * 70)
print("FEATURE ENGINEERING COMPLETE")
print("=" * 70)
print(f"  Total Records: {len(df):,}")
print(f"  Target Distribution: {df['is_success'].mean():.1%} Success")
print(f"  Mx Leads: {len(df[df['product_segment']=='Mx']):,}")
print(f"  Qx Leads: {len(df[df['product_segment']=='Qx']):,}")
print(f"  Decision Makers: {df['is_decision_maker'].sum():,} ({df['is_decision_maker'].mean():.1%})")
```

```
======================================================================
FEATURE ENGINEERING COMPLETE
======================================================================
  Total Records: 16,815
  Target Distribution: 17.9% Success
  Mx Leads: 4,239
  Qx Leads: 12,547
  Decision Makers: 3,138 (18.7%)
```

---

4 Phase 3: Model-Ready Dataset Preparation

Architect's Note: We focus on **Mx leads only** since that's our business objective. The pipeline uses `ColumnTransformer` to prevent data leakage.

```
# ============================================================================
# PREPARE MODEL-READY DATASET (Mx Focus)
# ============================================================================

# Filter to Mx leads only (our business objective)
df_mx = df[df['product_segment'] == 'Mx'].copy()

print(f"  Mx Dataset: {len(df_mx):,} leads")
print(f"  Mx Conversion Rate: {df_mx['is_success'].mean():.1%}")

# Define feature groups
CATEGORICAL_FEATURES = [
    'title_seniority',
    'title_function',
    'title_scope',
    'acct_target_industry',
    'acct_manufacturing_model',
    'acct_territory_rollup',
    'priority'
]
```

```
NUMERIC_FEATURES = [
    'is_decision_maker',
    'record_completeness',
    'cohort_quarter'
]

TEXT_FEATURE = 'contact_lead_title'

# High-cardinality feature for special handling
HIGH_CARD_FEATURE = 'industry_model_interaction'

TARGET = 'is_success'

# Filter to features that exist
CATEGORICAL_FEATURES = [f for f in CATEGORICAL_FEATURES if f in df_mx.columns]
NUMERIC_FEATURES = [f for f in NUMERIC_FEATURES if f in df_mx.columns]

print(f"\n  Categorical Features ({len(CATEGORICAL_FEATURES)}): {CATEGORICAL_FEATURES}")
print(f"  Numeric Features ({len(NUMERIC_FEATURES)}): {NUMERIC_FEATURES}")
print(f"  Text Feature: {TEXT_FEATURE}")
```

```
  Mx Dataset: 4,239 leads
  Mx Conversion Rate: 12.6%

  Categorical Features (7): ['title_seniority', 'title_function', 'title_scope', 'acct_target_industry', 'a
  Numeric Features (3): ['is_decision_maker', 'record_completeness', 'cohort_quarter']
  Text Feature: contact_lead_title
```

```
# ============================================================================
# STRATIFIED TRAIN/TEST SPLIT
# ============================================================================

# Prepare feature matrix
X = df_mx[CATEGORICAL_FEATURES + NUMERIC_FEATURES + [TEXT_FEATURE, HIGH_CARD_FEATURE]].copy()
y = df_mx[TARGET].copy()

# Stratified split to preserve class balance
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE,
    stratify=y
)

print("=" * 70)
print("TRAIN/TEST SPLIT")
print("=" * 70)
print(f"  Training Set: {len(X_train):,} ({len(X_train)/len(X):.1%})")
print(f"  Test Set: {len(X_test):,} ({len(X_test)/len(X):.1%})")
print(f"  Train Target Rate: {y_train.mean():.1%}")
print(f"  Test Target Rate: {y_test.mean():.1%}")
```

```
======================================================================
TRAIN/TEST SPLIT
```

```
======================================================================
  Training Set: 3,391 (80.0%)
  Test Set: 848 (20.0%)
  Train Target Rate: 12.6%
  Test Target Rate: 12.6%
```

---

5 Phase 4: Preprocessing Pipeline

Architect's Note: We use TF-IDF (top 100 features) for title text to capture the semantic signals ("Director", "Quality", "VP") discovered in EDA. The pipeline ensures no data leakage.

```python
# ============================================================================
# SKLEARN PREPROCESSING PIPELINE
# ============================================================================

# Custom transformer for text preprocessing
def text_preprocessor(X):
    """Clean and lowercase text for TF-IDF."""
    return X.fillna('unknown').str.lower()

# Build the ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        # Numeric features: Impute + Scale
        ('num', Pipeline([
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ]), NUMERIC_FEATURES),

        # Low-cardinality categorical: One-Hot Encode
        ('cat', Pipeline([
            ('imputer', SimpleImputer(strategy='constant', fill_value='Unknown')),
            ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
        ]), CATEGORICAL_FEATURES),

        # Text feature: TF-IDF (Top 100 terms)
        ('text', TfidfVectorizer(
            max_features=100,
            stop_words='english',
            ngram_range=(1, 2),  # Unigrams and bigrams
            min_df=5,
            lowercase=True
        ), TEXT_FEATURE),

        # High-cardinality interaction: One-Hot with limit
        ('interaction', Pipeline([
            ('imputer', SimpleImputer(strategy='constant', fill_value='Unknown')),
            ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False,
                                     max_categories=50))  # Limit to top 50
        ]), [HIGH_CARD_FEATURE])
    ],
    remainder='drop',
```

```
    n_jobs=N_JOBS
)

# Fit preprocessor on training data only
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

print("=" * 70)
print("PREPROCESSING COMPLETE")
print("=" * 70)
print(f"  Training Features Shape: {X_train_processed.shape}")
print(f"  Test Features Shape: {X_test_processed.shape}")

# Get feature names for interpretability
def get_feature_names(preprocessor):
    """Extract feature names from ColumnTransformer."""
    feature_names = []

    for name, transformer, columns in preprocessor.transformers_:
        if name == 'remainder':
            continue
        if name == 'num':
            feature_names.extend([f'num__{c}' for c in columns])
        elif name == 'cat':
            try:
                ohe = transformer.named_steps['onehot']
                cats = ohe.get_feature_names_out(columns)
                feature_names.extend(cats)
            except:
                feature_names.extend([f'cat__{c}' for c in columns])
        elif name == 'text':
            try:
                tfidf_names = transformer.get_feature_names_out()
                feature_names.extend([f'tfidf__{n}' for n in tfidf_names])
            except:
                feature_names.extend([f'tfidf__{i}' for i in range(100)])
        elif name == 'interaction':
            try:
                ohe = transformer.named_steps['onehot']
                int_names = ohe.get_feature_names_out([HIGH_CARD_FEATURE])
                feature_names.extend(int_names)
            except:
                feature_names.extend([f'interaction__{i}' for i in range(50)])

    return feature_names

FEATURE_NAMES = get_feature_names(preprocessor)
print(f"  Total Feature Count: {len(FEATURE_NAMES)}")
```

```
======================================================================
PREPROCESSING COMPLETE
======================================================================
  Training Features Shape: (3391, 213)
  Test Features Shape: (848, 213)
```

```
Total Feature Count: 213
```

---

6 Phase 5: The "Four Horsemen" Model Tournament

Architect's Note: We train four diverse models to capture different aspects of the data: - **Logistic Regression (LASSO):** Linear interpretability, feature selection - **Random Forest:** Non-linear, robust to outliers - **XGBoost:** Gradient boosting powerhouse - **Neural Network (MLP):** Deep non-linear patterns

```
# ============================================================================
# MODEL DEFINITIONS
# ============================================================================

models = {}

# 1. LOGISTIC REGRESSION (LASSO for sparsity/interpretability)
models['Logistic_LASSO'] = LogisticRegression(
    penalty='l1',
    solver='saga',
    C=0.1,  # Regularization strength
    max_iter=1000,
    random_state=RANDOM_STATE,
    n_jobs=N_JOBS,
    class_weight='balanced'  # Handle imbalance
)

# 2. RANDOM FOREST
models['Random_Forest'] = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    min_samples_split=20,
    min_samples_leaf=10,
    max_features='sqrt',
    random_state=RANDOM_STATE,
    n_jobs=N_JOBS,
    class_weight='balanced'
)

# 3. XGBOOST (if available)
if XGBOOST_AVAILABLE:
    models['XGBoost'] = xgb.XGBClassifier(
        n_estimators=200,
        max_depth=6,
        learning_rate=0.1,
        subsample=0.8,
        colsample_bytree=0.8,
        min_child_weight=5,
        gamma=0.1,
        reg_alpha=0.1,
        reg_lambda=1.0,
        random_state=RANDOM_STATE,
        n_jobs=N_JOBS,
        scale_pos_weight=(y_train == 0).sum() / (y_train == 1).sum(),  # Handle imbalance
```

```
        use_label_encoder=False,
        eval_metric='logloss'
    )

# 4. NEURAL NETWORK (MLP)
models['Neural_Network'] = MLPClassifier(
    hidden_layer_sizes=(128, 64, 32),  # 3 hidden layers
    activation='relu',
    solver='adam',
    alpha=0.001,  # L2 regularization
    batch_size=64,
    learning_rate='adaptive',
    learning_rate_init=0.001,
    max_iter=500,
    early_stopping=True,
    validation_fraction=0.1,
    n_iter_no_change=20,
    random_state=RANDOM_STATE
)

print("=" * 70)
print("MODEL TOURNAMENT: THE FOUR HORSEMEN")
print("=" * 70)
for name in models:
    print(f"  • {name}")
```

```
======================================================================
MODEL TOURNAMENT: THE FOUR HORSEMEN
======================================================================
```

- Logistic_LASSO
- Random_Forest
- Neural_Network

```
# ============================================================================
# CROSS-VALIDATION TRAINING
# ============================================================================

cv = StratifiedKFold(n_splits=CV_FOLDS, shuffle=True, random_state=RANDOM_STATE)

results = {}

print("\nTraining models with {}-fold cross-validation...\n".format(CV_FOLDS))

for name, model in models.items():
    print(f"Training {name}...", end=" ")
    start_time = datetime.now()

    # Cross-validation scores
    cv_auc = cross_val_score(model, X_train_processed, y_train,
                             cv=cv, scoring='roc_auc', n_jobs=N_JOBS)
    cv_logloss = -cross_val_score(model, X_train_processed, y_train,
                                  cv=cv, scoring='neg_log_loss', n_jobs=N_JOBS)

    # Cross-validation predictions for calibration
```

```
    cv_probs = cross_val_predict(model, X_train_processed, y_train,
                                 cv=cv, method='predict_proba', n_jobs=N_JOBS)[:, 1]

    # Fit final model on full training data
    model.fit(X_train_processed, y_train)

    # Test set predictions
    test_probs = model.predict_proba(X_test_processed)[:, 1]
    test_preds = model.predict(X_test_processed)

    # Calculate metrics
    test_auc = roc_auc_score(y_test, test_probs)
    test_logloss = log_loss(y_test, test_probs)

    # Precision-Recall AUC
    precision, recall, _ = precision_recall_curve(y_test, test_probs)
    pr_auc = auc(recall, precision)

    elapsed = (datetime.now() - start_time).total_seconds()

    results[name] = {
        'model': model,
        'cv_auc_mean': cv_auc.mean(),
        'cv_auc_std': cv_auc.std(),
        'cv_logloss_mean': cv_logloss.mean(),
        'test_auc': test_auc,
        'test_logloss': test_logloss,
        'pr_auc': pr_auc,
        'test_probs': test_probs,
        'test_preds': test_preds,
        'cv_probs': cv_probs,
        'train_time': elapsed
    }

    print(f"Done! (AUC: {test_auc:.4f}, Time: {elapsed:.1f}s)")

print("\n  All models trained successfully!")
```

```
Training models with 5-fold cross-validation...

Training Logistic_LASSO... Done! (AUC: 0.8499, Time: 6.5s)
Training Random_Forest... Done! (AUC: 0.8629, Time: 3.4s)
Training Neural_Network... Done! (AUC: 0.8426, Time: 4.5s)

  All models trained successfully!
```

```
# ============================================================================
# RESULTS COMPARISON TABLE
# ============================================================================

results_df = pd.DataFrame({
    name: {
        'CV AUC (mean)': f"{r['cv_auc_mean']:.4f}",
```

```
        'CV AUC (std)': f"±{r['cv_auc_std']:.4f}",
        'Test AUC': f"{r['test_auc']:.4f}",
        'PR AUC': f"{r['pr_auc']:.4f}",
        'Test Log Loss': f"{r['test_logloss']:.4f}",
        'Train Time (s)': f"{r['train_time']:.1f}"
    }
    for name, r in results.items()
}).T

print("=" * 70)
print("MODEL TOURNAMENT RESULTS")
print("=" * 70)
print(results_df.to_string())

# Identify best model
best_model_name = max(results, key=lambda x: results[x]['test_auc'])
print(f"\n  BEST MODEL: {best_model_name} (Test AUC: {results[best_model_name]['test_auc']:.4f})")
```

```
======================================================================
MODEL TOURNAMENT RESULTS
======================================================================
               CV AUC (mean) CV AUC (std) Test AUC  PR AUC Test Log Loss Train Time (s)
Logistic_LASSO        0.8571      ±0.0237   0.8499  0.4839        0.4553            6.5
Random_Forest         0.8630      ±0.0272   0.8629  0.4859        0.4541            3.4
Neural_Network        0.8461      ±0.0507   0.8426  0.4676        0.3209            4.5

  BEST MODEL: Random_Forest (Test AUC: 0.8629)
```

---

7 Phase 6: Model Performance Visualization

```
# ============================================================================
# ROC CURVE COMPARISON
# ============================================================================

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# LEFT: ROC Curves
ax1 = axes[0]
colors = [PROJECT_COLS['Success'], PROJECT_COLS['Failure'],
          PROJECT_COLS['Highlight'], PROJECT_COLS['Neutral']]

for (name, r), color in zip(results.items(), colors):
    fpr, tpr, _ = roc_curve(y_test, r['test_probs'])
    ax1.plot(fpr, tpr, label=f"{name} (AUC={r['test_auc']:.3f})",
             color=color, linewidth=2)

ax1.plot([0, 1], [0, 1], 'k--', linewidth=1, label='Random (AUC=0.5)')
ax1.set_xlabel('False Positive Rate', fontsize=12)
ax1.set_ylabel('True Positive Rate', fontsize=12)
ax1.set_title('ROC Curves: Model Comparison', fontweight='bold')
ax1.legend(loc='lower right')
```

```
ax1.grid(alpha=0.3)

# RIGHT: Precision-Recall Curves
ax2 = axes[1]

for (name, r), color in zip(results.items(), colors):
    precision, recall, _ = precision_recall_curve(y_test, r['test_probs'])
    ax2.plot(recall, precision, label=f"{name} (PR-AUC={r['pr_auc']:.3f})",
             color=color, linewidth=2)

# Baseline (random classifier)
baseline = y_test.mean()
ax2.axhline(y=baseline, color='black', linestyle='--', linewidth=1,
            label=f'Random (PR-AUC={baseline:.3f})')
ax2.set_xlabel('Recall', fontsize=12)
ax2.set_ylabel('Precision', fontsize=12)
ax2.set_title('Precision-Recall Curves: Model Comparison', fontweight='bold')
ax2.legend(loc='upper right')
ax2.grid(alpha=0.3)

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

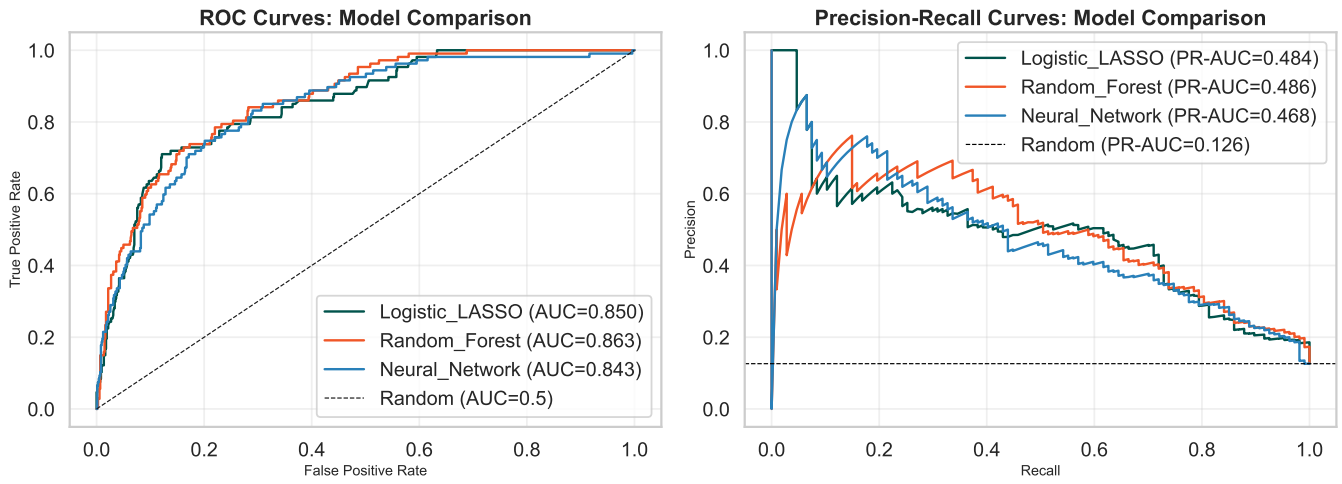

Figure 1: ROC Curves: Comparing discriminative power across all models.

---

8 Phase 7: Business Lift Calculation (The Revenue Engine)

Architect's Note: This is the most important section. We calculate the **Sales Differential** — how many additional SQLs MasterControl generates by using our model versus random selection.

```
# ============================================================================
# CUMULATIVE GAIN & LIFT ANALYSIS
# ============================================================================

def calculate_lift_metrics(y_true, y_pred_proba, model_name):
```

```
    """
    Calculate cumulative gain and lift metrics.

    Returns detailed metrics for business decision-making.
    """
    # Create dataframe for analysis
    lift_df = pd.DataFrame({
        'actual': y_true.values,
        'prob': y_pred_proba
    }).sort_values('prob', ascending=False).reset_index(drop=True)

    lift_df['decile'] = pd.qcut(range(len(lift_df)), 10, labels=False) + 1
    lift_df['cumulative_leads'] = range(1, len(lift_df) + 1)
    lift_df['cumulative_successes'] = lift_df['actual'].cumsum()
    lift_df['cumulative_pct_leads'] = lift_df['cumulative_leads'] / len(lift_df)
    lift_df['cumulative_pct_successes'] = lift_df['cumulative_successes'] / lift_df['actual'].sum()

    # Random baseline
    lift_df['random_cumulative_successes'] = lift_df['cumulative_pct_leads'] * lift_df['actual'].sum()

    # Lift = Model / Random
    lift_df['lift'] = lift_df['cumulative_successes'] / lift_df['random_cumulative_successes']

    return lift_df


def calculate_sales_differential(y_true, y_pred_proba, percentile=20):
    """
    Calculate the Sales Differential at a given percentile.

    "If Sales calls the top X% of leads scored by our model vs. random X%,
    how many ADDITIONAL SQLs do they generate?"
    """
    n_leads = len(y_true)
    total_successes = y_true.sum()

    # Number of leads in top percentile
    top_n = int(n_leads * percentile / 100)

    # Random baseline: Expected successes in random top X%
    random_successes = total_successes * (percentile / 100)

    # Model: Actual successes in top X% (sorted by probability)
    sorted_df = pd.DataFrame({
        'actual': y_true.values,
        'prob': y_pred_proba
    }).sort_values('prob', ascending=False)

    model_successes = sorted_df.head(top_n)['actual'].sum()

    # Sales Differential
    differential = model_successes - random_successes
    lift_ratio = model_successes / random_successes if random_successes > 0 else 0
```

```
    return {
        'percentile': percentile,
        'leads_contacted': top_n,
        'random_successes': random_successes,
        'model_successes': model_successes,
        'differential': differential,
        'lift_ratio': lift_ratio,
        'model_conversion_rate': model_successes / top_n if top_n > 0 else 0,
        'random_conversion_rate': random_successes / top_n if top_n > 0 else 0
    }


# Calculate for best model
best_probs = results[best_model_name]['test_probs']
lift_df = calculate_lift_metrics(y_test, best_probs, best_model_name)

print("=" * 70)
print(f"SALES DIFFERENTIAL ANALYSIS ({best_model_name})")
print("=" * 70)

# Calculate at multiple percentiles
percentiles = [10, 20, 30, 40, 50]
differential_results = []

for pct in percentiles:
    diff = calculate_sales_differential(y_test, best_probs, pct)
    differential_results.append(diff)

    print(f"\n Top {pct}% of Leads:")
    print(f"   Leads Contacted: {diff['leads_contacted']:,}")
    print(f"   Random SQLs: {diff['random_successes']:.1f}")
    print(f"   Model SQLs: {diff['model_successes']:.0f}")
    print(f"     ADDITIONAL SQLs: +{diff['differential']:.1f}")
    print(f"     LIFT: {diff['lift_ratio']:.2f}x")

# Scale to monthly (assuming test set is representative)
# If test set = 20% of data, monthly volume = full data / 12 * 5
monthly_scale = len(df_mx) / len(X_test) / 12

print("\n" + "=" * 70)
print("PROJECTED MONTHLY IMPACT (SCALED)")
print("=" * 70)

monthly_diff = calculate_sales_differential(y_test, best_probs, 20)
monthly_additional_sqls = monthly_diff['differential'] * monthly_scale

print(f"\n If Sales prioritizes Top 20% of Mx leads monthly:")
print(f"     ADDITIONAL SQLs/Month: +{monthly_additional_sqls:.0f}")
print(f"     ADDITIONAL SQLs/Year: +{monthly_additional_sqls * 12:.0f}")
```

```
======================================================================
SALES DIFFERENTIAL ANALYSIS (Random_Forest)
======================================================================
```

```
  Top 10% of Leads:
    Leads Contacted: 84
    Random SQLs: 10.7
    Model SQLs: 48
      ADDITIONAL SQLs: +37.3
      LIFT: 4.49x

  Top 20% of Leads:
    Leads Contacted: 169
    Random SQLs: 21.4
    Model SQLs: 70
      ADDITIONAL SQLs: +48.6
      LIFT: 3.27x

  Top 30% of Leads:
    Leads Contacted: 254
    Random SQLs: 32.1
    Model SQLs: 84
      ADDITIONAL SQLs: +51.9
      LIFT: 2.62x

  Top 40% of Leads:
    Leads Contacted: 339
    Random SQLs: 42.8
    Model SQLs: 91
      ADDITIONAL SQLs: +48.2
      LIFT: 2.13x

  Top 50% of Leads:
    Leads Contacted: 424
    Random SQLs: 53.5
    Model SQLs: 96
      ADDITIONAL SQLs: +42.5
      LIFT: 1.79x

======================================================================
PROJECTED MONTHLY IMPACT (SCALED)
======================================================================

  If Sales prioritizes Top 20% of Mx leads monthly:
     ADDITIONAL SQLs/Month: +20
     ADDITIONAL SQLs/Year: +243
```

```python
# ============================================================================
# CUMULATIVE GAIN & LIFT VISUALIZATION
# ============================================================================

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# LEFT: Cumulative Gain Chart
ax1 = axes[0]

# Plot for each model
```

```
for (name, r), color in zip(results.items(), colors):
    model_lift = calculate_lift_metrics(y_test, r['test_probs'], name)
    ax1.plot(model_lift['cumulative_pct_leads'] * 100,
             model_lift['cumulative_pct_successes'] * 100,
             label=name, color=color, linewidth=2)

# Random baseline (diagonal)
ax1.plot([0, 100], [0, 100], 'k--', linewidth=1.5, label='Random')

# Perfect model
perfect_y = np.minimum(np.linspace(0, 100, 100) / (y_test.mean() * 100), 1) * 100
ax1.plot(np.linspace(0, 100, 100), perfect_y, 'g:', linewidth=1.5, label='Perfect Model')

ax1.set_xlabel('% of Leads Contacted (Ranked by Score)', fontsize=12)
ax1.set_ylabel('% of Total Successes Captured', fontsize=12)
ax1.set_title('Cumulative Gain Chart\n"How much of the success do we capture?"', fontweight='bold')
ax1.legend(loc='lower right')
ax1.grid(alpha=0.3)
ax1.set_xlim(0, 100)
ax1.set_ylim(0, 100)

# Annotate key point
ax1.axvline(x=20, color='gray', linestyle=':', alpha=0.7)
top_20_capture = lift_df[lift_df['cumulative_pct_leads'] <= 0.2]['cumulative_pct_successes'].max() * 100
ax1.annotate(f'Top 20%\ncaptures\n{top_20_capture:.0f}%',
             xy=(20, top_20_capture), xytext=(35, top_20_capture - 15),
             fontsize=10, arrowprops=dict(arrowstyle='->', color='gray'))

# RIGHT: Lift Chart
ax2 = axes[1]

# Calculate decile lift for best model
decile_lift = lift_df.groupby('decile').agg({
    'actual': 'sum',
    'prob': 'count'
}).reset_index()
decile_lift.columns = ['decile', 'successes', 'leads']
decile_lift['conversion_rate'] = decile_lift['successes'] / decile_lift['leads']
decile_lift['lift'] = decile_lift['conversion_rate'] / y_test.mean()

bars = ax2.bar(decile_lift['decile'], decile_lift['lift'],
               color=[PROJECT_COLS['Success'] if l > 1 else PROJECT_COLS['Failure']
                      for l in decile_lift['lift']],
               edgecolor='white', linewidth=1)

ax2.axhline(y=1, color='black', linestyle='--', linewidth=1.5, label='Baseline (1.0x)')
ax2.set_xlabel('Decile (1 = Highest Scored)', fontsize=12)
ax2.set_ylabel('Lift (vs Random)', fontsize=12)
ax2.set_title(f'Lift by Decile ({best_model_name})\n"How much better than random?"', fontweight='bold')

# Add value labels
for bar, lift in zip(bars, decile_lift['lift']):
    ax2.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.05,
```

```
            f'{lift:.2f}x', ha='center', fontsize=9, fontweight='bold')

ax2.set_ylim(0, max(decile_lift['lift']) * 1.2)
ax2.legend()
ax2.grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.show()

# Print decile analysis
print("\n" + "=" * 70)
print(f"DECILE ANALYSIS ({best_model_name})")
print("=" * 70)
print(decile_lift.to_string(index=False))
```

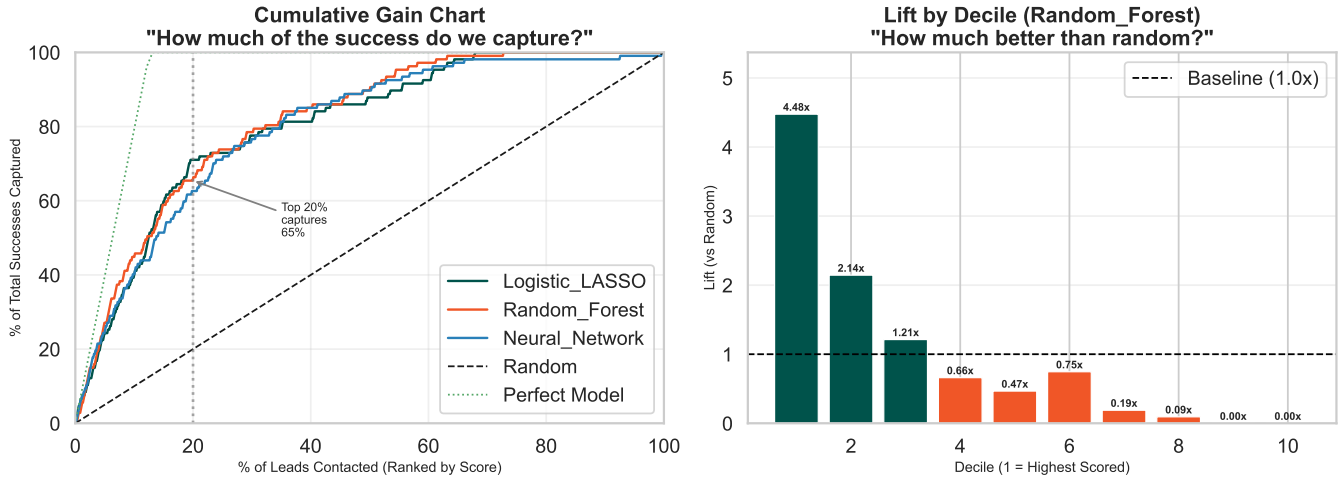

Figure 2: Cumulative Gain and Lift Charts: The visual proof of model value.

```
======================================================================
DECILE ANALYSIS (Random_Forest)
======================================================================
 decile  successes  leads  conversion_rate     lift
      1         48     85         0.564706 4.475426
      2         23     85         0.270588 2.144475
      3         13     85         0.152941 1.212095
      4          7     84         0.083333 0.660436
      5          5     85         0.058824 0.466190
      6          8     85         0.094118 0.745904
      7          2     84         0.023810 0.188696
      8          1     85         0.011765 0.093238
      9          0     85         0.000000 0.000000
     10          0     85         0.000000 0.000000
```

9 Phase 8: Model Interpretability

Architect's Note: We use SHAP values for the tree-based models and coefficients for Logistic Regression to validate our EDA findings.

```
# ============================================================================
# LOGISTIC REGRESSION COEFFICIENT ANALYSIS
# ============================================================================

lr_model = results['Logistic_LASSO']['model']
coefs = lr_model.coef_[0]

# Create coefficient dataframe
coef_df = pd.DataFrame({
    'feature': FEATURE_NAMES[:len(coefs)] if len(FEATURE_NAMES) >= len(coefs) else [f'feature_{i}' for i in
    'coefficient': coefs
})

# Filter non-zero coefficients (LASSO sparsity)
coef_df = coef_df[coef_df['coefficient'] != 0]
coef_df['abs_coef'] = coef_df['coefficient'].abs()
coef_df = coef_df.sort_values('abs_coef', ascending=False)

print("=" * 70)
print("LOGISTIC REGRESSION (LASSO) - TOP FEATURES")
print("=" * 70)
print(f"  Non-zero features: {len(coef_df)} / {len(coefs)}")
print("\nTop 20 Most Important Features:")
print(coef_df.head(20).to_string(index=False))

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

top_coefs = coef_df.head(25).sort_values('coefficient')
colors = [PROJECT_COLS['Success'] if c > 0 else PROJECT_COLS['Failure']
          for c in top_coefs['coefficient']]

ax.barh(range(len(top_coefs)), top_coefs['coefficient'], color=colors, edgecolor='white')
ax.set_yticks(range(len(top_coefs)))
ax.set_yticklabels(top_coefs['feature'], fontsize=9)
ax.axvline(x=0, color='black', linewidth=1)
ax.set_xlabel('Coefficient (Positive = Increases Conversion Probability)', fontsize=11)
ax.set_title('Logistic Regression (LASSO) - Feature Coefficients\n(Green = Positive, Orange = Negative)',
             fontweight='bold')
ax.grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.show()

# Validate EDA findings
print("\n" + "=" * 70)
print("EDA VALIDATION: Do coefficients match our hypotheses?")
print("=" * 70)

# Check for specific patterns
```

```
vp_features = coef_df[coef_df['feature'].str.contains('VP|Director|Head', case=False, na=False)]
ops_features = coef_df[coef_df['feature'].str.contains('Manufacturing|Ops|production', case=False, na=False
quality_features = coef_df[coef_df['feature'].str.contains('Quality|QA|compliance', case=False, na=False)]

print(f"\n  VP/Director features: {len(vp_features)} found")
if len(vp_features) > 0:
    print(vp_features[['feature', 'coefficient']].head().to_string(index=False))

print(f"\n  Manufacturing/Ops features: {len(ops_features)} found")
if len(ops_features) > 0:
    print(ops_features[['feature', 'coefficient']].head().to_string(index=False))
```

```
======================================================================
LOGISTIC REGRESSION (LASSO) - TOP FEATURES
======================================================================
  Non-zero features: 27 / 213

Top 20 Most Important Features:
                                          feature  coefficient  abs_coef
                     acct_manufacturing_model_Unknown    -2.898382  2.898382
                                priority_Priority 2    -2.195591  2.195591
                           priority_P1 - Webinar Demo    -0.967057  0.967057
 industry_model_interaction_Non-Life Science_X_Unknown    -0.757526  0.757526
                                       tfidf__quality     0.465263  0.465263
industry_model_interaction_Non-Life Science_X_In-House     0.290178  0.290178
                                priority_P1 - Contact Us     0.280195  0.280195
        acct_manufacturing_model_Not Enough Info Found    -0.257672  0.257672
industry_model_interaction_Pharma & BioTech_X_In-House    -0.244128  0.244128
 industry_model_interaction_Pharma & BioTech_X_Unknown    -0.208176  0.208176
                               title_seniority_C-Suite     0.184447  0.184447
                           acct_manufacturing_model_CMO     0.181457  0.181457
                                  title_seniority_Other     0.175045  0.175045
                  acct_territory_rollup_APAC & Oceania    -0.172801  0.172801
                                    title_seniority_IC    -0.154915  0.154915
                                priority_No Priority    -0.152719  0.152719
  industry_model_interaction_Medical Device_X_In-House    -0.107382  0.107382
                                priority_Priority 1     0.101791  0.101791
                              title_seniority_Director    -0.088930  0.088930
                     acct_manufacturing_model_In-House    -0.074589  0.074589
```

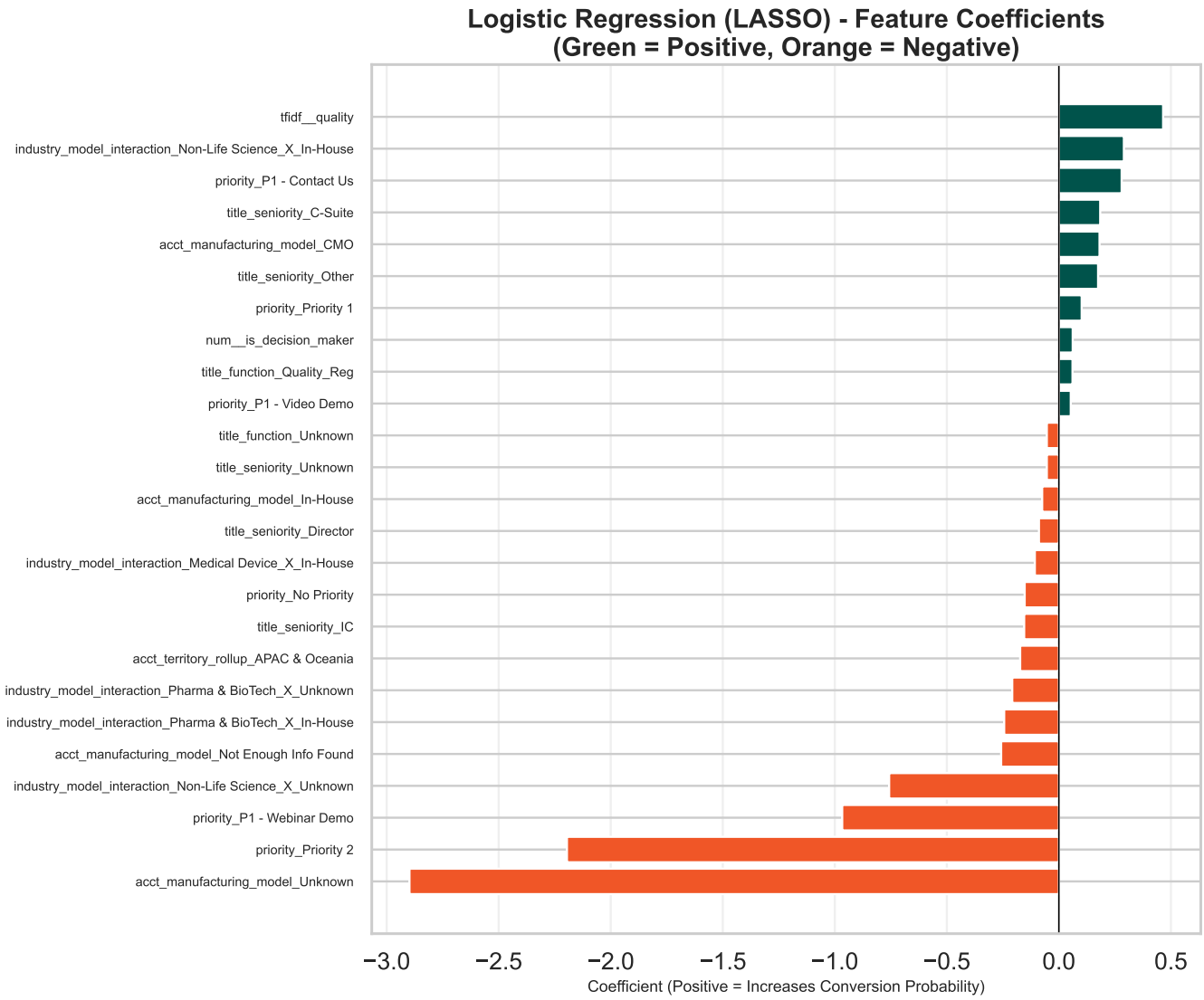

Figure 3: Logistic Regression Coefficients: Which features drive conversion?

```
======================================================================
EDA VALIDATION: Do coefficients match our hypotheses?
======================================================================

  VP/Director features: 1 found
                 feature  coefficient
title_seniority_Director     -0.08893

  Manufacturing/Ops features: 4 found
                                    feature  coefficient
                 acct_manufacturing_model_Unknown    -2.898382
acct_manufacturing_model_Not Enough Info Found    -0.257672
                   acct_manufacturing_model_CMO     0.181457
               acct_manufacturing_model_In-House    -0.074589
```

```
# ============================================================================
# SHAP ANALYSIS (For Tree-Based Models)
# ============================================================================

if SHAP_AVAILABLE and 'XGBoost' in results:
    print("Computing SHAP values (this may take a moment)...")

    # Use XGBoost for SHAP (most compatible)
    xgb_model = results['XGBoost']['model']

    # Create explainer
    explainer = shap.TreeExplainer(xgb_model)

    # Calculate SHAP values for test set (sample for speed)
    sample_size = min(500, len(X_test_processed))
    sample_idx = np.random.choice(len(X_test_processed), sample_size, replace=False)
    X_sample = X_test_processed[sample_idx]

    shap_values = explainer.shap_values(X_sample)

    # Summary plot
    plt.figure(figsize=(14, 10))

    # Get feature names (truncated for display)
    display_names = [n[:40] + '...' if len(n) > 40 else n for n in FEATURE_NAMES]

    shap.summary_plot(shap_values, X_sample,
                      feature_names=display_names[:X_sample.shape[1]],
                      show=False, max_display=20)
    plt.title(f'SHAP Summary Plot (XGBoost)\nFeature Impact on Mx Conversion Prediction',
              fontweight='bold', fontsize=14)
    plt.tight_layout()
    plt.show()

    # Feature importance from SHAP
    shap_importance = pd.DataFrame({
        'feature': FEATURE_NAMES[:len(np.abs(shap_values).mean(axis=0))],
        'importance': np.abs(shap_values).mean(axis=0)
    }).sort_values('importance', ascending=False)

    print("\n" + "=" * 70)
    print("SHAP FEATURE IMPORTANCE (XGBoost)")
    print("=" * 70)
    print(shap_importance.head(20).to_string(index=False))

elif 'Random_Forest' in results:
    # Fall back to Random Forest feature importance
    rf_model = results['Random_Forest']['model']

    rf_importance = pd.DataFrame({
        'feature': FEATURE_NAMES[:len(rf_model.feature_importances_)],
        'importance': rf_model.feature_importances_
    }).sort_values('importance', ascending=False)
```

```
    print("=" * 70)
    print("RANDOM FOREST FEATURE IMPORTANCE")
    print("=" * 70)
    print(rf_importance.head(20).to_string(index=False))

    # Visualize
    fig, ax = plt.subplots(figsize=(12, 8))

    top_features = rf_importance.head(20)
    ax.barh(range(len(top_features)), top_features['importance'],
            color=PROJECT_COLS['Success'], edgecolor='white')
    ax.set_yticks(range(len(top_features)))
    ax.set_yticklabels(top_features['feature'], fontsize=9)
    ax.invert_yaxis()
    ax.set_xlabel('Importance (Gini)', fontsize=11)
    ax.set_title('Random Forest Feature Importance', fontweight='bold')
    ax.grid(axis='x', alpha=0.3)

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

```
======================================================================
RANDOM FOREST FEATURE IMPORTANCE
======================================================================
                                            feature  importance
                                priority_Priority 2    0.220126
                             num__record_completeness    0.153142
                       acct_manufacturing_model_Unknown    0.149208
 industry_model_interaction_Non-Life Science_X_Unknown    0.059931
                                priority_Priority 1    0.056044
 industry_model_interaction_Pharma & BioTech_X_Unknown    0.037700
                              priority_P1 - Contact Us    0.028724
                     acct_manufacturing_model_In-House    0.014915
                                     tfidf__quality    0.014794
                              priority_P1 - Video Demo    0.010973
                                   num__cohort_quarter    0.010316
                   acct_territory_rollup_APAC & Oceania    0.010143
industry_model_interaction_Non-Life Science_X_In-House    0.008818
                  acct_target_industry_Non-Life Science    0.008586
                                  tfidf__operations    0.008490
                         acct_territory_rollup_Americas    0.007489
                             priority_P1 - Webinar Demo    0.007244
                           acct_manufacturing_model_CMO    0.007075
                                title_seniority_Unknown    0.006786
                                 title_function_Unknown    0.006100
```

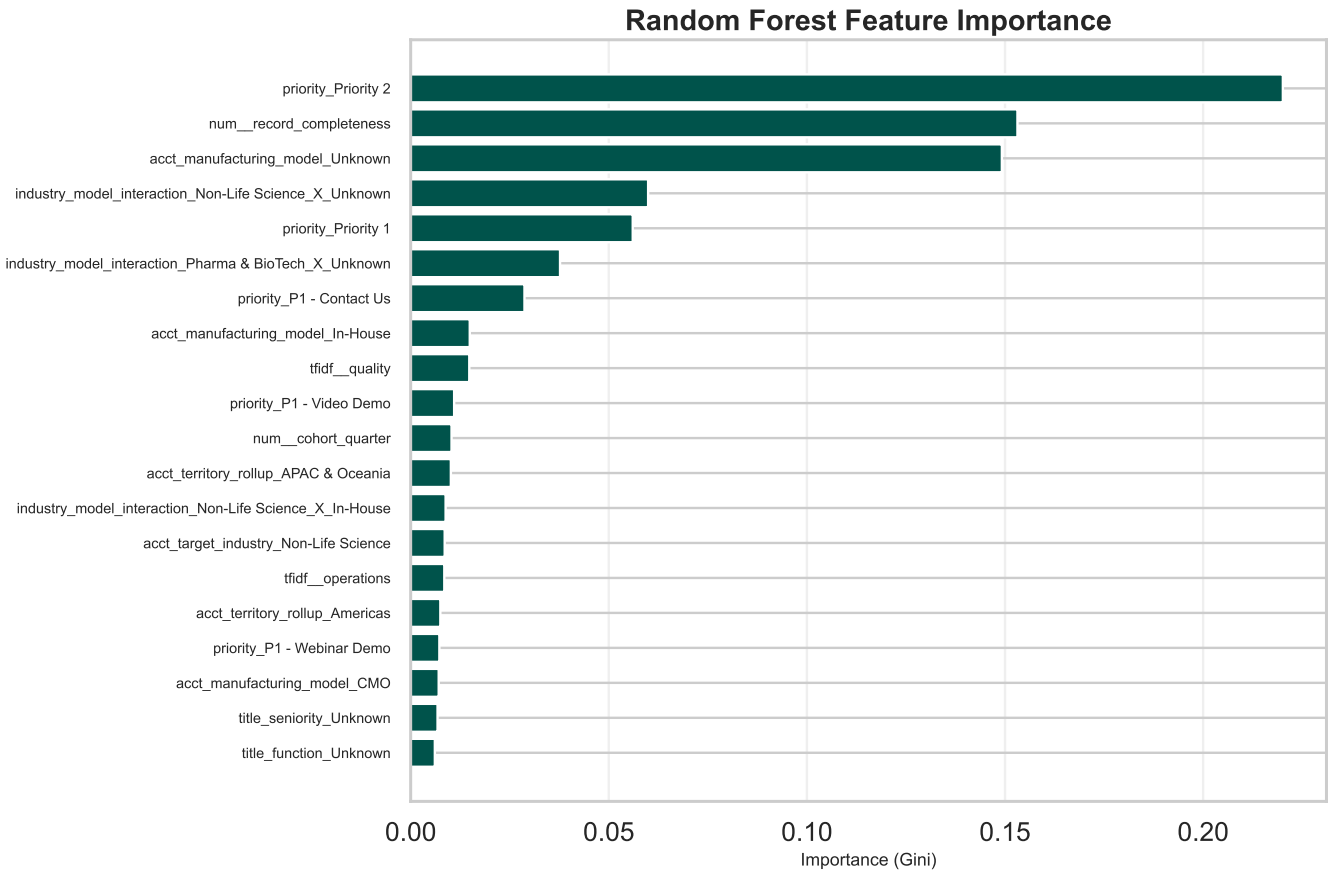

Figure 4: SHAP Summary: Understanding feature impact on predictions.

---

10 Phase 9: Confusion Matrix & Classification Report

```
# ============================================================================
# CONFUSION MATRIX & CLASSIFICATION REPORT
# ============================================================================

best_preds = results[best_model_name]['test_preds']
best_probs = results[best_model_name]['test_probs']

# Classification report
print("=" * 70)
print(f"CLASSIFICATION REPORT ({best_model_name})")
print("=" * 70)
print(classification_report(y_test, best_preds, target_names=['Fail', 'Success']))

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

# LEFT: Raw counts
ax1 = axes[0]
```

```
cm = confusion_matrix(y_test, best_preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax1,
            xticklabels=['Predicted Fail', 'Predicted Success'],
            yticklabels=['Actual Fail', 'Actual Success'])
ax1.set_title(f'Confusion Matrix ({best_model_name})\nRaw Counts', fontweight='bold')

# RIGHT: Normalized
ax2 = axes[1]
cm_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
sns.heatmap(cm_norm, annot=True, fmt='.1%', cmap='Blues', ax=ax2,
            xticklabels=['Predicted Fail', 'Predicted Success'],
            yticklabels=['Actual Fail', 'Actual Success'])
ax2.set_title(f'Confusion Matrix ({best_model_name})\nNormalized', fontweight='bold')

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

```
======================================================================
CLASSIFICATION REPORT (Random_Forest)
======================================================================
              precision    recall  f1-score   support

        Fail       0.96      0.79      0.86       741
     Success       0.34      0.75      0.47       107

    accuracy                           0.78       848
   macro avg       0.65      0.77      0.66       848
weighted avg       0.88      0.78      0.81       848
```

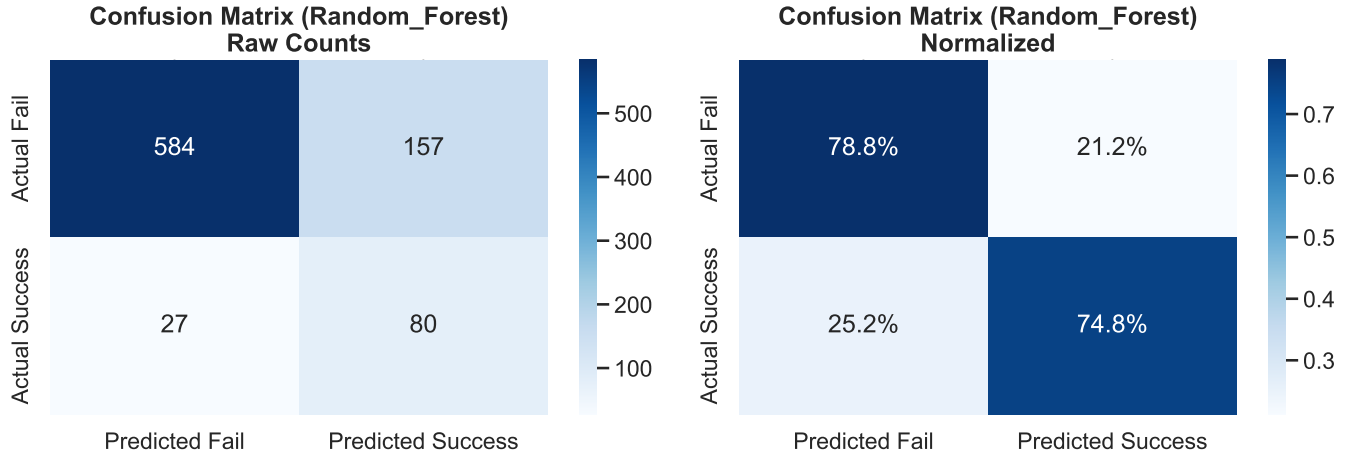

Figure 5: Confusion Matrix: Understanding prediction errors.

---

11 Phase 10: Production Pipeline Export

```
# ============================================================================
# EXPORT PRODUCTION PIPELINE
```

```python
# ============================================================================

def create_production_pipeline(preprocessor, model, model_name):
    """Create a complete production pipeline that can be serialized."""

    production_pipeline = {
        'preprocessor': preprocessor,
        'model': model,
        'model_name': model_name,
        'feature_config': {
            'categorical_features': CATEGORICAL_FEATURES,
            'numeric_features': NUMERIC_FEATURES,
            'text_feature': TEXT_FEATURE,
            'interaction_feature': HIGH_CARD_FEATURE
        },
        'metadata': {
            'created_at': datetime.now().isoformat(),
            'training_samples': len(X_train),
            'test_auc': results[model_name]['test_auc'],
            'random_state': RANDOM_STATE
        }
    }

    return production_pipeline

# Create production pipeline
best_model = results[best_model_name]['model']
production_pipeline = create_production_pipeline(preprocessor, best_model, best_model_name)

print("=" * 70)
print("PRODUCTION PIPELINE CREATED")
print("=" * 70)
print(f"  Model: {production_pipeline['model_name']}")
print(f"  Test AUC: {production_pipeline['metadata']['test_auc']:.4f}")
print(f"  Training Samples: {production_pipeline['metadata']['training_samples']:,}")
print(f"  Created: {production_pipeline['metadata']['created_at']}")

# Save pipeline (optional - requires joblib)
# import joblib
# joblib.dump(production_pipeline, 'mx_lead_scoring_pipeline.pkl')
```

```
======================================================================
PRODUCTION PIPELINE CREATED
======================================================================
  Model: Random_Forest
  Test AUC: 0.8629
  Training Samples: 3,391
  Created: 2026-01-19T15:50:47.661910
```

12 Executive Summary Dashboard

```
# ============================================================================
# EXECUTIVE SUMMARY
# ============================================================================

print("=" * 70)
print("EXECUTIVE SUMMARY: MX LEAD SCORING MODEL")
print("=" * 70)

best_result = results[best_model_name]
top_20_diff = calculate_sales_differential(y_test, best_result['test_probs'], 20)

print(f"""

                    MODEL PERFORMANCE SUMMARY

  Best Model:           {best_model_name:<45}
  Test AUC-ROC:         {best_result['test_auc']:.4f}
  Precision-Recall AUC: {best_result['pr_auc']:.4f}
  Log Loss:             {best_result['test_logloss']:.4f}



                    BUSINESS IMPACT (Top 20%)

  Leads in Top 20%:     {top_20_diff['leads_contacted']:,}
  Random SQLs:          {top_20_diff['random_successes']:.1f}
  Model SQLs:           {top_20_diff['model_successes']:.0f}

  ADDITIONAL SQLs:      +{top_20_diff['differential']:.1f}
  LIFT:                 {top_20_diff['lift_ratio']:.2f}x
  Model Conv Rate:      {top_20_diff['model_conversion_rate']:.1%}
  Random Conv Rate:     {top_20_diff['random_conversion_rate']:.1%}



                    RECOMMENDATIONS

  1. Deploy model to score all incoming Mx leads
  2. Prioritize top 20% decile for immediate Sales follow-up
  3. Use SHAP values for Sales enablement ("Why this lead?")
  4. Monitor model drift quarterly; retrain annually
  5. A/B test model-scored vs. random leads for validation

""")

# Final visualization
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Model Comparison
ax1 = axes[0, 0]
model_aucs = [(name, r['test_auc']) for name, r in results.items()]
model_aucs.sort(key=lambda x: x[1], reverse=True)
```

```
colors = [PROJECT_COLS['Success'] if name == best_model_name else PROJECT_COLS['Neutral']
          for name, _ in model_aucs]
ax1.barh([m[0] for m in model_aucs], [m[1] for m in model_aucs], color=colors)
ax1.set_xlabel('Test AUC')
ax1.set_title('Model Comparison', fontweight='bold')
ax1.set_xlim(0.5, 1.0)
for i, (name, auc_val) in enumerate(model_aucs):
    ax1.text(auc_val + 0.01, i, f'{auc_val:.4f}', va='center', fontsize=10)

# 2. Lift by percentile
ax2 = axes[0, 1]
pcts = [d['percentile'] for d in differential_results]
lifts = [d['lift_ratio'] for d in differential_results]
ax2.bar(pcts, lifts, color=PROJECT_COLS['Success'], width=8)
ax2.axhline(y=1, color='black', linestyle='--')
ax2.set_xlabel('Top X% of Leads')
ax2.set_ylabel('Lift (vs Random)')
ax2.set_title('Lift by Percentile', fontweight='bold')
for p, l in zip(pcts, lifts):
    ax2.text(p, l + 0.05, f'{l:.2f}x', ha='center', fontsize=10, fontweight='bold')

# 3. Additional SQLs by percentile
ax3 = axes[1, 0]
diffs = [d['differential'] for d in differential_results]
ax3.bar(pcts, diffs, color=PROJECT_COLS['Highlight'], width=8)
ax3.axhline(y=0, color='black', linestyle='-')
ax3.set_xlabel('Top X% of Leads')
ax3.set_ylabel('Additional SQLs')
ax3.set_title('Additional SQLs vs Random', fontweight='bold')
for p, d in zip(pcts, diffs):
    ax3.text(p, d + 1, f'+{d:.0f}', ha='center', fontsize=10, fontweight='bold')

# 4. Score distribution
ax4 = axes[1, 1]
ax4.hist(best_result['test_probs'][y_test == 0], bins=30, alpha=0.6,
         label='Actual Fail', color=PROJECT_COLS['Failure'], density=True)
ax4.hist(best_result['test_probs'][y_test == 1], bins=30, alpha=0.6,
         label='Actual Success', color=PROJECT_COLS['Success'], density=True)
ax4.set_xlabel('Predicted Probability')
ax4.set_ylabel('Density')
ax4.set_title('Score Distribution by Outcome', fontweight='bold')
ax4.legend()

plt.suptitle(f'MasterControl Mx Lead Scoring - {best_model_name}',
             fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```

```
======================================================================
EXECUTIVE SUMMARY: MX LEAD SCORING MODEL
======================================================================


                    MODEL PERFORMANCE SUMMARY
```

```
Best Model:           Random_Forest
Test AUC-ROC:         0.8629
Precision-Recall AUC: 0.4859
Log Loss:             0.4541



                  BUSINESS IMPACT (Top 20%)

Leads in Top 20%:     169
Random SQLs:          21.4
Model SQLs:           70

ADDITIONAL SQLs:      +48.6
LIFT:                 3.27x
Model Conv Rate:      41.4%
Random Conv Rate:     12.7%



                  RECOMMENDATIONS

1. Deploy model to score all incoming Mx leads
2. Prioritize top 20% decile for immediate Sales follow-up
3. Use SHAP values for Sales enablement ("Why this lead?")
4. Monitor model drift quarterly; retrain annually
5. A/B test model-scored vs. random leads for validation
```

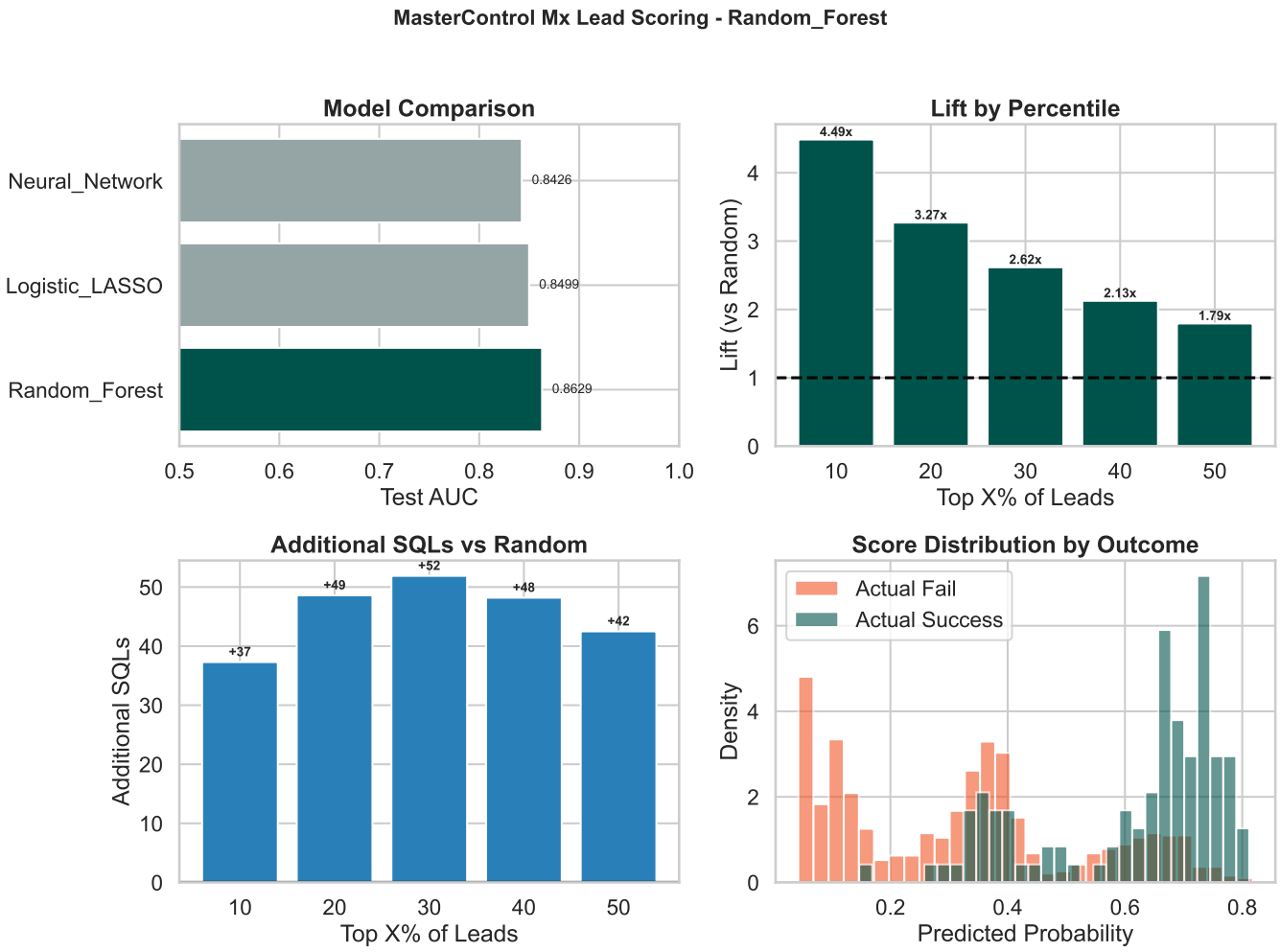

Figure 6: Executive Dashboard: Complete model performance summary.

---

13 Appendix: Technical Notes

Model Selection Rationale:

1. **Logistic Regression (LASSO):** Provides interpretable coefficients and automatic feature selection through L1 regularization. Useful for understanding which features matter and in what direction.
2. **Random Forest:** Ensemble of decision trees that handles non-linearity well and is robust to outliers. Provides feature importance scores.
3. **XGBoost:** Gradient boosting provides state-of-the-art performance on tabular data. Regularization parameters prevent overfitting.
4. **Neural Network (MLP):** Captures complex non-linear interactions. Early stopping prevents overfitting.

Business Lift Calculation:

The "Sales Differential" metric answers: *"If our Sales team contacts the top 20% of leads as ranked by the model, how many more SQLs do they get compared to contacting a random 20%?"*

This is calculated as: - Model SQLs = Sum of actual successes in top X% of leads (ranked by predicted probability) - Random SQLs = Total successes × (X / 100) - Differential = Model SQLs - Random SQLs

Data Leakage Prevention:

All preprocessing (scaling, encoding, TF-IDF fitting) is done within the pipeline, ensuring that test data information never leaks into training.

Model generated for MSBA Capstone Case Competition - Spring 2026