High-Potential HR Service Leads Prediction Model

# Introduction

This document outlines the approach taken to preprocess data, engineer features, and build predictive models for identifying high-potential leads for HR services. The objective was to predict whether a company is a "hot lead" based on its funding and hiring trends.

# Data Preprocessing

## Handling Missing Values

* No missing values were found in the dataset, so no imputation was required.

## Handling Infinite Values

* Detected inf values in the funding\_per\_employee column (caused by division by zero).
* Dropped all rows containing inf values to maintain data integrity

## Encoding Categorical Variables

* Applied **Label Encoding** for categorical columns.
* Stored encoders to maintain consistent transformations across train, test, and holdout datasets.
* Mapped unseen categories in test/holdout data to -1 to prevent errors.

## Date Type Conversion

* Converted the last\_funding\_date column from **string (object)** to **datetime** format using pd.to\_datetime().
* This ensured accurate time-based calculations and prevented potential issues during feature engineering.

## Outlier Detection and Handling

* Used **Z-score analysis** to detect extreme values in numerical columns.
* Found significant outliers in funding\_per\_employee and growth\_momentum.
* **Decision:** Outliers were **not removed** as they provided meaningful business insights based on domain knowledge.

# Feature Engineering

To enhance model performance, we derived new features from existing data

## Date-Based Features (last\_funding\_date)

* Extracted key time-based features:
  + **funding\_year** (year of last funding round).
  + **funding\_month** (seasonality impact).
  + **funding\_day** (day of last funding round).
  + **funding\_weekday** (weekday of last funding round, 0 = Monday, 6 = Sunday).
  + **funding\_quarter** (quarter in which the funding occurred).
* **Dropped the original** **last\_funding\_date column** after extracting these features.

# Model Selection & Training

## Applying SMOTE for Class Balancing

* Used **Synthetic Minority Over-Sampling Technique (SMOTE)** on df\_train to generate synthetic samples for the minority class (is\_hot\_lead=1).
* Ensured the dataset was balanced before training.

## Feature Scaling (Standardization)

* Applied **StandardScaler** to normalize all numerical features.
* This ensured that all models, especially **Logistic Regression and** **XGBoost**, performed optimally.

## Models Used for Training

To compare performance, we trained multiple models:

|  |  |
| --- | --- |
| **Model** | **Description** |
| **Logistic Regression** | Baseline model, interpretable but weak on non-linearity. |
| **Random Forest** | Ensemble model, handles non-linearity well. |
| **XGBoost** | Boosting model, optimized for structured data. |
| **Gradient Boosting** | Uses gradient descent to minimize errors iteratively. |
| **Decision Tree** | Simple yet prone to overfitting. |

## Model Evaluation Metrics

* **Primary Metric:** F1-Score (Balances Precision & Recall).
* **Secondary Metrics:**
  + **Accuracy** → Overall correct predictions.
  + **Precision** → How many predicted positives were actual positives?
  + **Recall** → How many actual positives were identified correctly?
  + **Confusion Matrix** → Used to analyze misclassifications.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **Confusion Matrix** |
| **Logistic Regression** | 0.8792 | 0.5981 | 0.5844 | 0.5911 | [[3953, 293], [310, 436]] |
| **Random Forest** | 0.9511 | 0.8031 | 0.8914 | 0.8449 | [[4083, 163], [81, 665]] |
| **XGBoost** | 0.9587 | 0.8199 | 0.9276 | 0.8704 | [[4094, 152], [54, 692]] |
| **Decision Tree** | 0.9193 | 0.7372 | 0.7144 | 0.7257 | [[4056, 190], [213, 533]] |
| **Gradient Boosting** | 0.9469 | 0.7675 | 0.9249 | 0.8389 | [[4056, 190], [213, 533]] |

* **XGBoost** outperformed all other models in terms of **accuracy, precision, recall, and F1-score**, making it the best model for this task.

# Hyperparameter Tuning (XGBoost Optimization)

## Randomized Search for Hyperparameter Tuning

* To further improve XGBoost's performance, we performed Randomized Hyperparameter Tuning using RandomizedSearchCV.

## Best Hyperparameters for XGBoost

* The optimized hyperparameters obtained were:

best\_params = {

'subsample': 0.7,

'reg\_lambda': 2.0,

'reg\_alpha': 0.5,

'n\_estimators': 400,

'max\_depth': 9,

'learning\_rate': 0.1788888888888889,

'gamma': 0.0,

'colsample\_bytree': 0.7

}

* The final **tuned** **XGBoost model** was trained with these parameters and stored as best\_xgboost\_model.pkl.

## Hyperparameter Tuning Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **Confusion Matrix** |
| **XGBoost (Tuned)** | 0.969351 | 0.863804 | 0.9437 | 0.901986 | [[4135, 111],  [42, 704]] |

* **Before Hyperparameter Tuning:** The model was **highly recall-focused**, meaning it caught most hot leads but misclassified some non-hot leads.
* **After Hyperparameter Tuning:** The model achieved a **better balance**, with **fewer false positives and higher precision** while maintaining high recall.

# Final Predictions & Submission

## Predicting on Holdout Dataset

* The holdout dataset was preprocessed the same way as train and test data.
* Standardization was applied using the previously saved StandardScaler.
* The best XGBoost model was used for final predictions.

# Conclusion

* **XGBoost (Tuned) was the best model**, achieving the highest F1-score (**0.9019**).
* **Randomized Hyperparameter Tuning** further improved model performance.
* **Final holdout predictions were generated and saved as submission.csv**.