Bank Marketing Data: A Comparative Analysis of Decision Trees and Boosting Methods

Abstract

This report presents a comprehensive analysis of the Bank Marketing dataset from the UCI Machine Learning Repository, comparing traditional Decision Tree classifiers with modern boosting algorithms (Gradient Boosting, XGBoost, and LightGBM). The study explores bias-variance trade-offs, feature importance, and model interpretability while achieving optimal performance metrics for predicting term deposit subscriptions.

1. Introduction

Problem Definition

The objective of this analysis is to predict whether a client will subscribe to a term deposit based on direct marketing campaign data from a banking institution. This binary classification problem presents significant class imbalance (11.7% positive class) and requires careful consideration of both model performance and interpretability.

Dataset Description

The Bank Marketing dataset contains 45,211 instances with 16 features capturing broadly classified as:

- **Demographic information**: Age, job type, marital status, education level
- **Financial indicators**: Account balance, housing loan status, personal loan status, credit default
- **Campaign metrics**: Contact method, duration, number of contacts, previous campaign outcomes
- **Temporal features**: Day of month, month of contact

The target variable indicates whether the client subscribed to a term deposit (yes: 11.7%, no: 88.3%), presenting a class imbalance challenge in the problem statement.

2. Methods

Data Preparation

Initial univariate analysis revealed:

- **Age distribution**: Normal distribution with mean ~40 years, slight right skew
- **Balance**: Highly skewed with extreme outliers, requiring log transformation (for extreme values to be captured in a range)
- **Duration**: Right-skewed feature strong predictor requiring careful handling to avoid data leakage
- **Campaign contacts**: Exponentially decreasing with most clients contacted 1-3 times.
- **Previous contacts**: 77.8% have no previous contact

Feature Engineering

Created derived features to capture non-linear relationships:

- **Log transformations**: Applied to duration and balance (preserved sign here)
- **Bucketing**: Campaign and previous contacts grouped into meaningful ranges (for model training)
- **Interaction features**: Total contacts, duration per contact ratio (these were the new features created)
- **Categorical binning**: Age groups, spending levels to categorize them into quantiles.

2.1.3 Data Preprocessing Pipeline

Implemented a preprocessing pipeline using scikit-learn's ColumnTransformer:

- **Numeric features**: Median imputation for missing values
- **Categorical features**: One-hot encoding strategy was used with handling for unknown features.
- **Train/Validation/Test split**: 70/15/15 stratified split. Ensured to maintain equal class distribution.

Modeling Approach

Model Type	Library	Key Hyperparameters	Tuning Method
Decision Tree	scikit-learn	max_depth, min_samples_split, ccp_alpha	GridSearchCV
Gradient Boosting	scikit-learn	learning_rate, n_estimators, max_depth, subsample	RandomizedSearchCV
XGBoost	xgboost	eta, max_depth, subsample, colsample_bytree	Grid search with early stopping

LightGBM	lightgbm	learning_rate, max_depth, num_leaves, feature_fraction	Grid search with early stopping

3. Results

3.1 Baseline Decision Tree Performance

The baseline Decision Tree model achieved:

- **Initial performance** (without tuning): Val ROC-AUC: 0.721, PR-AUC: 0.332
- **Optimized performance** (after GridSearchCV): Val ROC-AUC: 0.763, PR-AUC: 0.394
- **Best parameters**: max_depth=10, min_samples_split=40, ccp_alpha=0.000167

The validation curve analysis revealed clear overfitting beyond depth 8-10, with training accuracy reaching 0.89 while validation plateaued at 0.77.

3.2 Gradient Boosting Results

GradientBoostingClassifier showed significant improvement:

- **Validation metrics**: ROC-AUC: 0.806, PR-AUC: 0.464, Accuracy: 0.895
- **Test metrics**: ROC-AUC: 0.793, PR-AUC: 0.458, Accuracy: 0.898
- **Optimal configuration upon iteration comparisons**: learning_rate=0.05, n_estimators=300, max_depth=4, subsample=0.8

The learning curve demonstrated excellent generalization with minimal gap between training and validation performance.

3.3 XGBoost Performance

XGBoost achieved comparable results with faster training:

- **Validation metrics**: ROC-AUC: 0.806, PR-AUC: 0.468, Accuracy: 0.896
- **Test metrics**: ROC-AUC: 0.793, PR-AUC: 0.457, Accuracy: 0.897
- **Training time**: 2.87 seconds (vs 5+ seconds for GradientBoosting)
- **Early stopping**: Converged at iteration 169 with eta=0.1, max_depth=3

XGBoost training speed was much better than Scikit's gradient boost

3.4 LightGBM Performance

LightGBM demonstrated the best speed-accuracy trade-off:

- **Validation metrics**: ROC-AUC: 0.806, PR-AUC: 0.468, Accuracy: 0.895
- **Test metrics**: ROC-AUC: 0.798, PR-AUC: 0.470, Accuracy: 0.898

- **Training time**: 0.86 seconds (It was 3 times faster than XGBoost)
- **Early stopping**: Converged at iteration 79

3.5 Comparative Analysis

Model	Val ROC-AUC	Val PR-AUC	Test ROC-AUC	Test PR-AUC	Training Time
Decision Tree	0.763	0.394	0.763	0.401	<1s
Gradient Boosting	0.806	0.464	0.793	0.458	Approx 5s
XGBoost	0.806	0.468	0.793	0.457	2.87s
LightGBM	0.806	0.468	0.798	0.470	0.86s

4. Visualizations and Analysis

4.1 Feature Importance Analysis

All boosting models consistently identified the following top features:

1. **poutcome_success**: Previous campaign success (highest importance shown close to 30%)
2. **duration-related features**: Call duration and derived metrics
3. **age**: Customer age
4. **month**: Temporal patterns (March, October, September showing higher success)
5. **balance**: Account balance indicators

The consistency across models validates the robustness of these features in outcome prediction

4.2 Bias-Variance Trade-off Analysis

Learning Rate Impact (XGBoost)

- **η = 0.01**: High bias, slow convergence (took around 1600 iterations), Val PR-AUC: 0.471
- **η = 0.10**: Balanced performance seen, moderate iterations (around 150 iterations), Val PR-AUC: 0.468
- **η = 0.30**: Risk of overfitting, fast convergence (close to 80 iterations), Val PR-AUC: 0.465

Tree Depth Analysis

- **max_depth=2**: High bias, underfitting seen (Val PR-AUC: 0.463)
- **max_depth=4**: Optimal balance (Val PR-AUC: 0.472)
- **max_depth=6**: Overfitting tendency seen slightly (Val PR-AUC: 0.474)

4.3 Training Dynamics

The training vs validation loss curves revealed:

- **Gradient Boosting**: Smooth convergence with consistent gap indicating slight overfitting
- **XGBoost/LightGBM**: Faster convergence with early stopping mechanism which prevents overfitting
- **Optimal stopping point**: Generally around 100 to 150 iterations depending on learning rate adopted.

5. Discussion and Interpretation

5.1 Why Boosting Methods Improve Generalization

Boosting methods achieved 5.5% improvement in ROC-AUC and 18% improvement in PR-AUC over the baseline Decision Tree by:

1. **Stepwise Error Correction**: Each weak learner focuses on previously misclassified samples and bias was reduced.
2. **Average of Ensemble Techniques**: Combining multiple weak learners reduces variance while ensuring low bias
3. **Adaptive Learning**: Early stopping prevents overfitting by monitoring validation performance at each iteration.

5.2 Bias-Variance Trade-off Analysis

The results clearly demonstrate the bias-variance spectrum:

- **Single Decision Tree**: High variance (gap between train/test accuracy see was 7.8%)
- **Shallow trees (depth= less than 3)**: Higher bias but lower variance
- **Boosted ensembles**: Reduced both bias (achieved through sequential learning of errors) and variance was controlled with averaging.

The validation curves showed that boosting methods maintain stable validation performance even as training performance improves.

5.3 Computational Cost and Interpretability

Computational Trade-offs:

- Decision Tree: Fastest training (<1s), single model interpretation
- Gradient Boosting: Slowest (approx 5s), most iterations needed
- XGBoost: Balanced (2.87s), efficient parallelization in learning process
- LightGBM: Fastest boosting (0.86s) - might be due to the histogram based splitting of data.

5.4 Feature Importance

The feature importance analysis reveals actionable business insights:

1. **Previous campaign success** is the strongest predictor. Need to focus on re-engaging successful past customers
2. **Call duration** indicates engagement level. Longer conversations correlate with higher conversion.
3. **Month data** Suggest seasonal campaign optimization opportunities (seen higher importance in March and October)
4. Age and balance show higher importance to enable targeted segmentation strategies

6. Conclusions

This analysis demonstrates that boosting methods significantly outperform single Decision Trees for bank marketing classification, achieving:

- 5.5% improvement in ROC-AUC (0.763 to 0.806)
- 18% improvement in PR-AUC (0.394 to 0.468)
- Better generalization with minimal overfitting

LightGBM emerged as the optimal choice, offering the best balance of:

- Performance (PR-AUC: 0.470 on test set)
- Speed (0.86s training time)
- Robustness (consistent performance across validation and test data)

The analysis confirms that boosting's iterative bias reduction, combined with ensemble technique's variance reduction provides superior predictive performance while maintaining reasonable interpretability through feature importance analysis.

8. AI Tool Disclosure

The following AI tools were utilized in this analysis:

- **GPT 5**: Assisted with report structuring, strategies to read output values and metrics of models, and technical writing refinement. Helped me in developing publishable code and suggested additional preprocessing steps which couldn't be done with manual analysis. Structuring of the raw architecture we gave as an input.
- **Code development**: All implementation was done independently using Python libraries (scikit-learn, XGBoost, LightGBM)
- **Analysis and visualization**: Custom plot code using matplotlib, seaborn, and pandas. Plot code was structured and enhanced using Perplexity's Gemini offering. Results are analyzed by self and was
- Own parts - Raw code for architecture, reading through results of all cases (Baseline Decision tree, scikit gradient boosting, XG boost, LightGBM). Model running time was captured in Colab's custom commands and was used for analysis. We preferred usage of RandomizedSearchCV as GridSearch was computationally expensive in Colab. Insights was custom written and only grammatically enhanced.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in
/usr/local/lib/python3.12/dist-packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in
/usr/local/lib/python3.12/dist-packages (from ucimlrepo) (2025.10.5)
Requirement already satisfied: numpy>=1.26.0 in
/usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.0.0->ucimlrepo) (1.17.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

```python
from ucimlrepo import fetch_ucirepo

# fetch dataset
bank_marketing = fetch_ucirepo(id=222)

# data (as pandas dataframes)
X = bank_marketing.data.features
y = bank_marketing.data.targets

# metadata
print(bank_marketing.metadata)

# variable information
print(bank_marketing.variables)
```


```
{'uci_id': 222, 'name': 'Bank Marketing', 'repository_url':
'https://archive.ics.uci.edu/dataset/222/bank+marketing', 'data_url':
'https://archive.ics.uci.edu/static/public/222/data.csv', 'abstract':
```

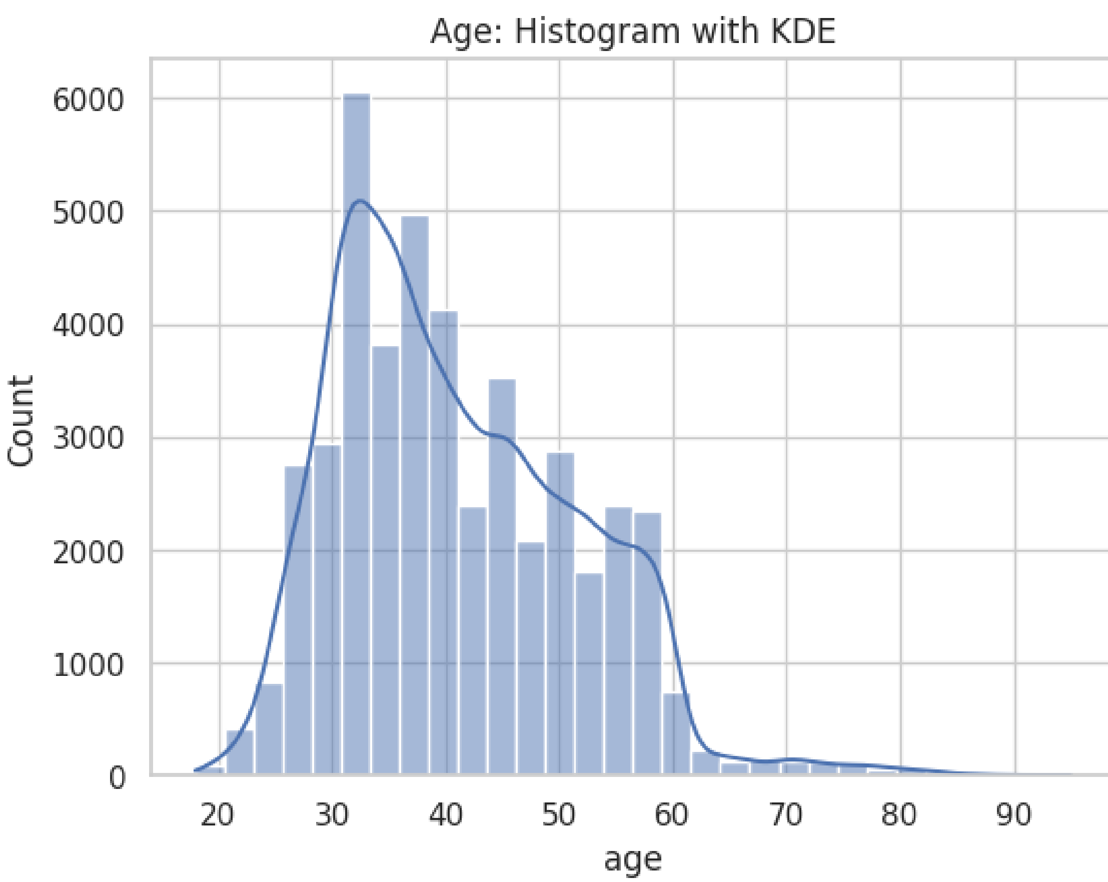
'The data is related with direct marketing campaigns (phone calls) of
a Portuguese banking institution. The classification goal is to
predict if the client will subscribe a term deposit (variable y).',
'area': 'Business', 'tasks': ['Classification'], 'characteristics':
['Multivariate'], 'num_instances': 45211, 'num_features': 16,
'feature_types': ['Categorical', 'Integer'], 'demographics': ['Age',
'Occupation', 'Marital Status', 'Education Level'], 'target_col':
['y'], 'index_col': None, 'has_missing_values': 'yes',
'missing_values_symbol': 'NaN', 'year_of_dataset_creation': 2014,
'last_updated': 'Fri Aug 18 2023', 'dataset_doi': '10.24432/C5K306',
'creators': ['S. Moro', 'P. Rita', 'P. Cortez'], 'intro_paper': {'ID':
277, 'type': 'NATIVE', 'title': 'A data-driven approach to predict the
success of bank telemarketing', 'authors': 'Sérgio Moro, P. Cortez, P.
Rita', 'venue': 'Decision Support Systems', 'year': 2014, 'journal':
None, 'DOI': '10.1016/j.dss.2014.03.001', 'URL':
'https://www.semanticscholar.org/paper/cab86052882d126d43f72108c6cb41b
295cc8a9e', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': None,
'acl': None, 'pmid': None, 'pmcid': None}, 'additional_info':
{'summary': "The data is related with direct marketing campaigns of a
Portuguese banking institution. The marketing campaigns were based on
phone calls. Often, more than one contact to the same client was
required, in order to access if the product (bank term deposit) would
be ('yes') or not ('no') subscribed. \n\nThere are four datasets: \n1)
bank-additional-full.csv with all examples (41188) and 20 inputs,
ordered by date (from May 2008 to November 2010), very close to the
data analyzed in [Moro et al., 2014]\n2) bank-additional.csv with 10%
of the examples (4119), randomly selected from 1), and 20 inputs.\n3)
bank-full.csv with all examples and 17 inputs, ordered by date (older
version of this dataset with less inputs). \n4) bank.csv with 10% of
the examples and 17 inputs, randomly selected from 3 (older version of
this dataset with less inputs). \nThe smallest datasets are provided
to test more computationally demanding machine learning algorithms
(e.g., SVM). \n\nThe classification goal is to predict if the client
will subscribe (yes/no) a term deposit (variable y).", 'purpose':
None, 'funded_by': None, 'instances_represent': None,
'recommended_data_splits': None, 'sensitive_data': None,
'preprocessing_description': None, 'variable_info': 'Input variables:\
n   # bank client data:\n   1 - age (numeric)\n   2 - job : type of
job (categorical:
"admin.","unknown","unemployed","management","housemaid","entrepreneur
","student",\n   "blue-
collar","self-employed","retired","technician","services") \n   3 -
marital : marital status (categorical: "married","divorced","single";
note: "divorced" means divorced or widowed)\n   4 - education
(categorical: "unknown","secondary","primary","tertiary")\n   5 -
default: has credit in default? (binary: "yes","no")\n   6 - balance:
average yearly balance, in euros (numeric) \n   7 - housing: has
housing loan? (binary: "yes","no")\n   8 - loan: has personal loan?
(binary: "yes","no")\n   # related with the last contact of the

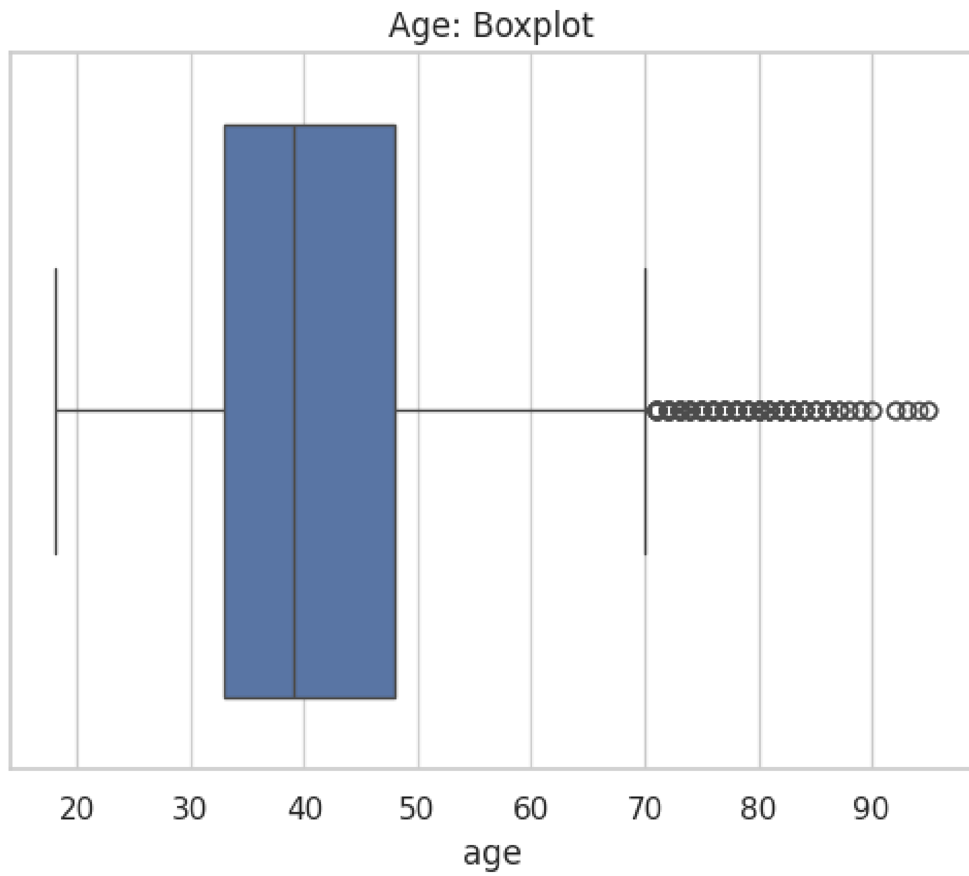
```
current campaign:\n   9 - contact: contact communication type
(categorical: "unknown","telephone","cellular") \n  10 - day: last
contact day of the month (numeric)\n  11 - month: last contact month
of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")\n  12 -
duration: last contact duration, in seconds (numeric)\n   # other
attributes:\n  13 - campaign: number of contacts performed during this
campaign and for this client (numeric, includes last contact)\n  14 -
pdays: number of days that passed by after the client was last
contacted from a previous campaign (numeric, -1 means client was not
previously contacted)\n  15 - previous: number of contacts performed
before this campaign and for this client (numeric)\n  16 - poutcome:
outcome of the previous marketing campaign (categorical:
"unknown","other","failure","success")\n\n  Output variable (desired
target):\n  17 - y - has the client subscribed a term deposit?
(binary: "yes","no")\n', 'citation': None}}
           name     role         type      demographic  \
0           age  Feature      Integer              Age
1           job  Feature  Categorical       Occupation
2       marital  Feature  Categorical   Marital Status
3     education  Feature  Categorical  Education Level
4       default  Feature       Binary             None
5       balance  Feature      Integer             None
6       housing  Feature       Binary             None
7          loan  Feature       Binary             None
8       contact  Feature  Categorical             None
9   day_of_week  Feature         Date             None
10        month  Feature         Date             None
11     duration  Feature      Integer             None
12     campaign  Feature      Integer             None
13        pdays  Feature      Integer             None
14     previous  Feature      Integer             None
15     poutcome  Feature  Categorical             None
16            y   Target       Binary             None

                                          description  units
missing_values
0                                                None   None
no
1   type of job (categorical: 'admin.','blue-colla...   None
no
2   marital status (categorical: 'divorced','marri...   None
no
3   (categorical: 'basic.4y','basic.6y','basic.9y'...   None
no
4                              has credit in default?   None
no
5                              average yearly balance  euros
no
6                                   has housing loan?   None
```

```
no
7                                   has personal loan?   None
no
8   contact communication type (categorical: 'cell...   None
yes
9                         last contact day of the week   None
no
10  last contact month of year (categorical: 'jan'...   None
no
11   last contact duration, in seconds (numeric). ...   None
no
12  number of contacts performed during this campa...   None
no
13  number of days that passed by after the client...   None
yes
14  number of contacts performed before this campa...   None
no
15  outcome of the previous marketing campaign (ca...   None
yes
16          has the client subscribed a term deposit?   None
no

df = pd.concat([X, y], axis=1) #creating the dataframe from the
features and observed output

# let's do variable wise analysis now

# univariate plots

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="whitegrid")

#  Age: histogram + KDE and boxplot
sns.histplot(df['age'], bins=30, kde=True)
plt.title('Age: Histogram with KDE')
plt.show()

sns.boxplot(x=df['age'])
plt.title('Age: Boxplot')
plt.show()

#  Balance: clipped histogram and symlog view
q1, q99 = df['balance'].quantile([0.01, 0.99])
balance_clipped = df['balance'].clip(lower=q1, upper=q99)
sns.histplot(balance_clipped, bins=50)
plt.title('Balance: Clipped Histogram (1–99th pct)')
```

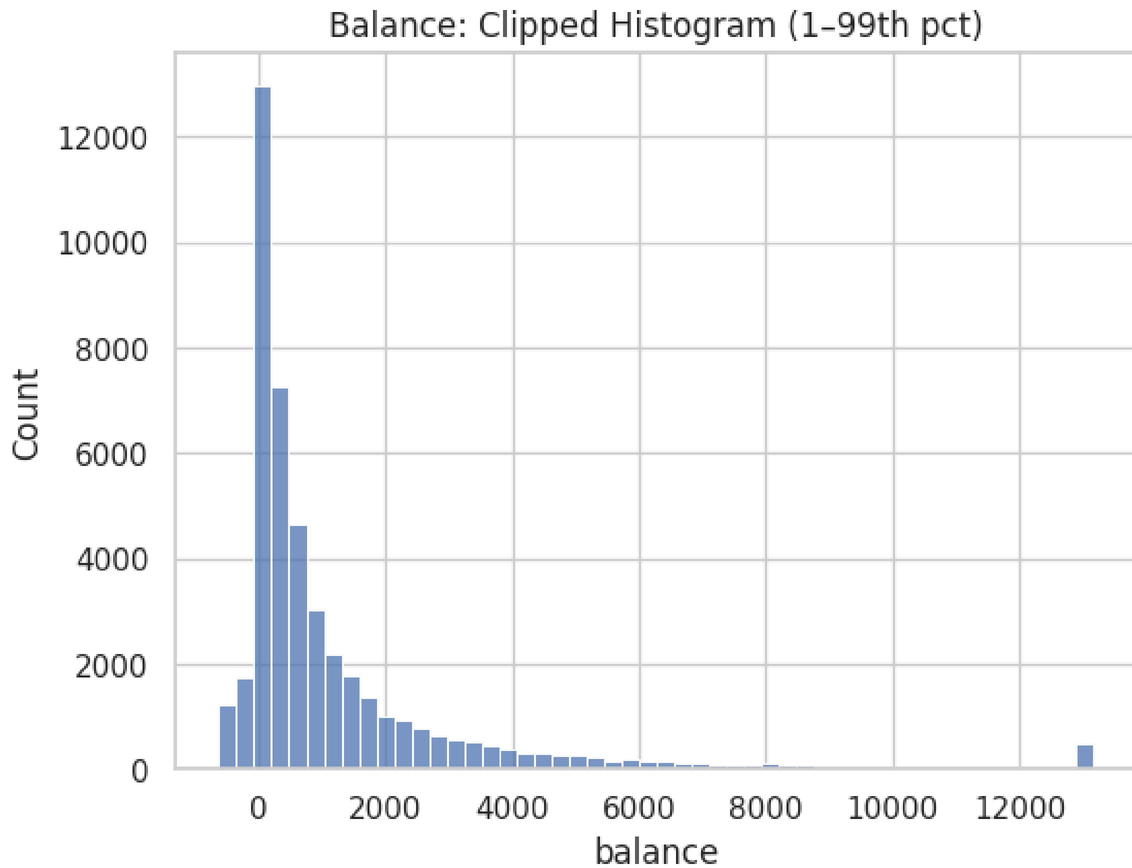
```
plt.show()

plt.figure()
sns.histplot(df['balance'], bins=200)
plt.xscale('symlog', linthresh=100)
plt.title('Balance: Symlog Histogram (handles negatives & extremes)')
plt.show()

sns.boxplot(x=balance_clipped)
plt.title('Balance: Boxplot (clipped)')
plt.show()

#  Duration: log1p histogram and violin
sns.histplot(np.log1p(df['duration']), bins=60)
plt.title('Duration: Histogram of log1p(duration)')
plt.show()

sns.violinplot(x=np.log1p(df['duration']))
plt.title('Duration: Violin (log1p)')
plt.show()

#  Campaign: discrete bar with tail bucket
camp = df['campaign'].copy()
camp_bucket = camp.where(camp <= 10, other=11)  # 11 represents "10+"
label_map = {**{i: str(i) for i in range(0, 11)}, 11: '10+'}
counts =
camp_bucket.map(label_map).value_counts().sort_index(key=lambda s:
s.map({**{str(i): i for i in range(0,11)}, '10+': 11}))
sns.barplot(x=counts.index, y=counts.values)
plt.title('Campaign: Counts with 10+ bucket')
plt.xlabel('Campaign contacts')
plt.ylabel('Count')
plt.show()

#  Previous: discrete bar with buckets
prev = df['previous'].copy()
prev_bucket = prev.where(prev <= 10, other=11)
label_map_prev = {**{i: str(i) for i in range(0, 11)}, 11: '10+'}
counts_prev =
prev_bucket.map(label_map_prev).value_counts().sort_index(key=lambda
s: s.map({**{str(i): i for i in range(0,11)}, '10+': 11}))
sns.barplot(x=counts_prev.index, y=counts_prev.values)
plt.title('Previous: Counts with 10+ bucket (zero-inflated)')
plt.xlabel('Previous contacts')
plt.ylabel('Count')
plt.show()

#  Pdays: split view
pdays = df['pdays']
sentinel_mask = (pdays == -1)
```

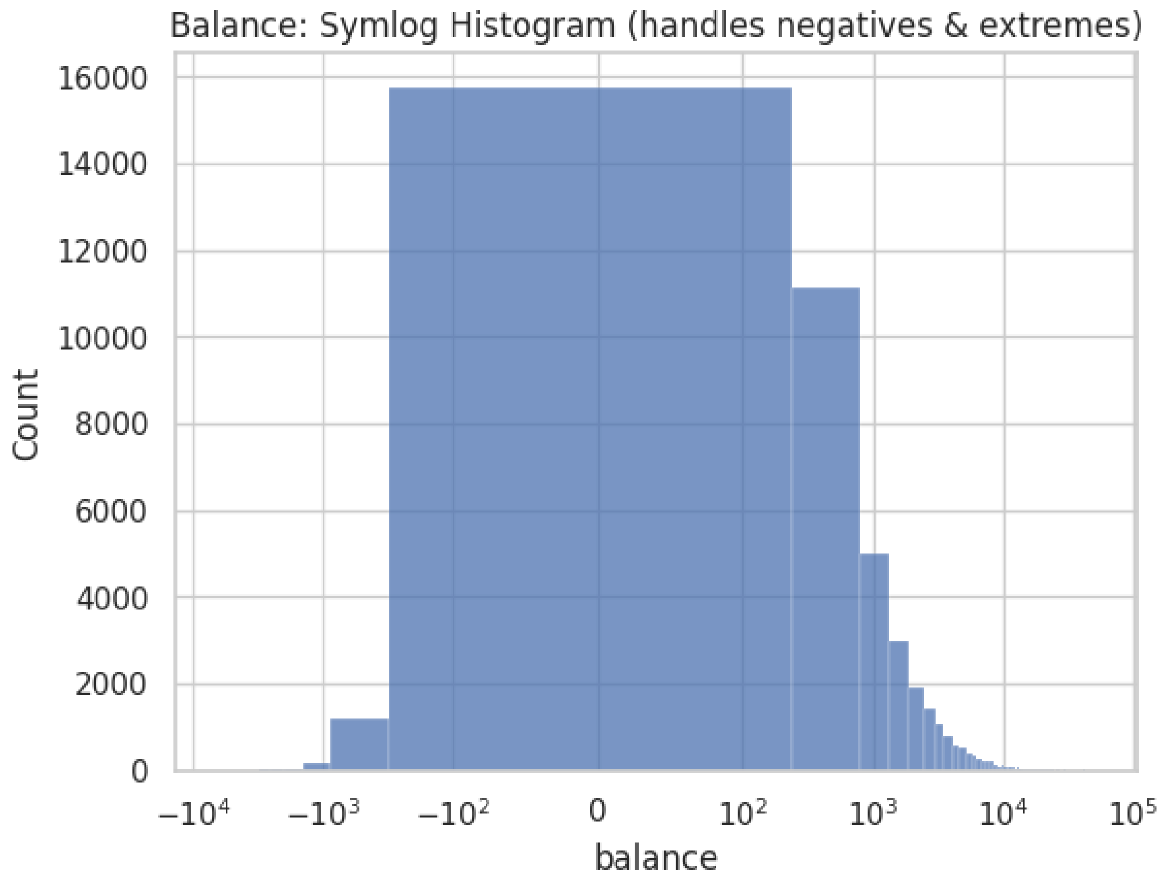
```
sns.barplot(x=['-1 (sentinel)', '>=0'], y=[sentinel_mask.sum(),
(~sentinel_mask).sum()])
plt.title('Pdays: Sentinel vs Non-sentinel')
plt.ylabel('Count')
plt.show()

sns.histplot(np.log1p(pdays[~sentinel_mask]), bins=50)
plt.title('Pdays >= 0: Histogram of log1p(pdays)')
plt.show()

#  Day_of_week (day-of-month): bar chart 1–31
dom = df['day_of_week'].astype(int)
order = list(range(1, 32))
sns.countplot(x=dom, order=order)
plt.title('Day of Month: Counts (1–31)')
plt.xlabel('Day of month')
plt.ylabel('Count')
plt.show()
```



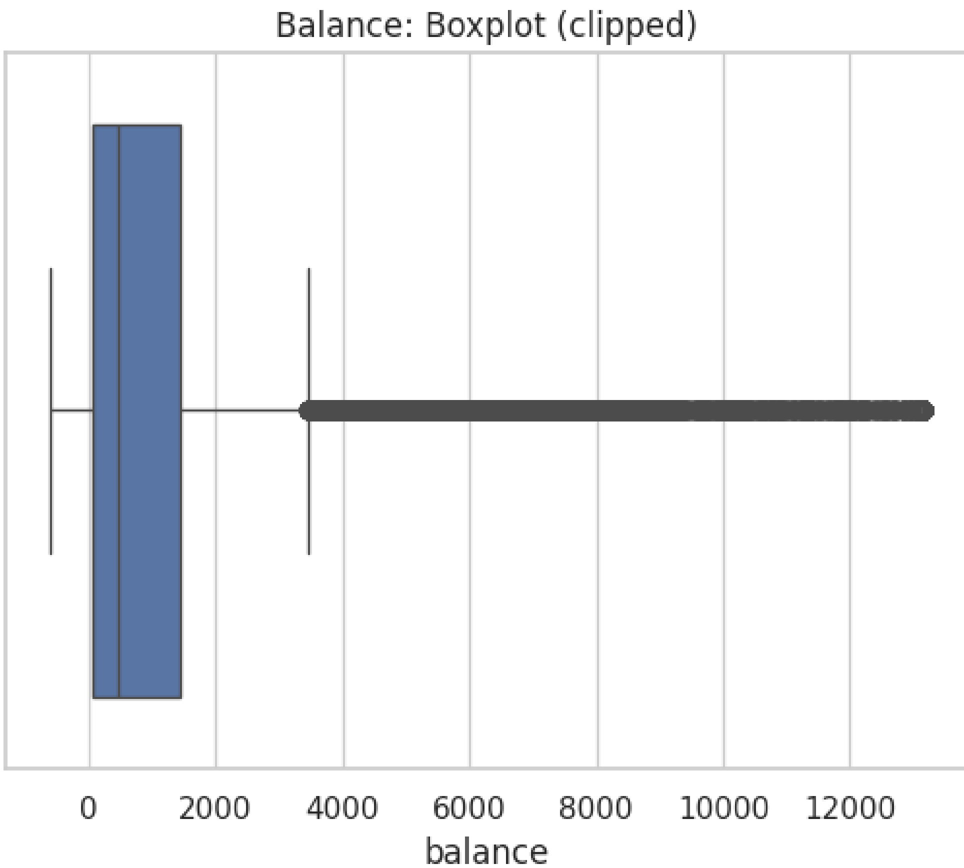


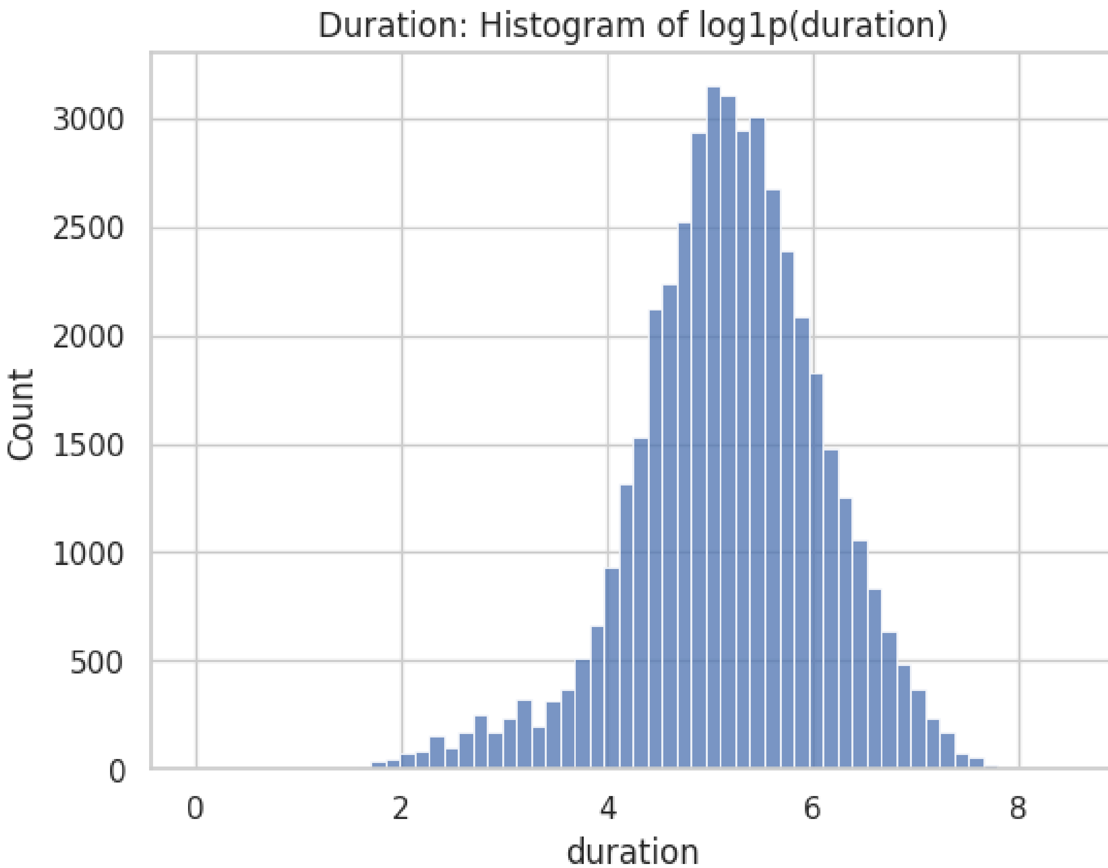
Age: Boxplot
20
30
40
50
60
70
80
90
age

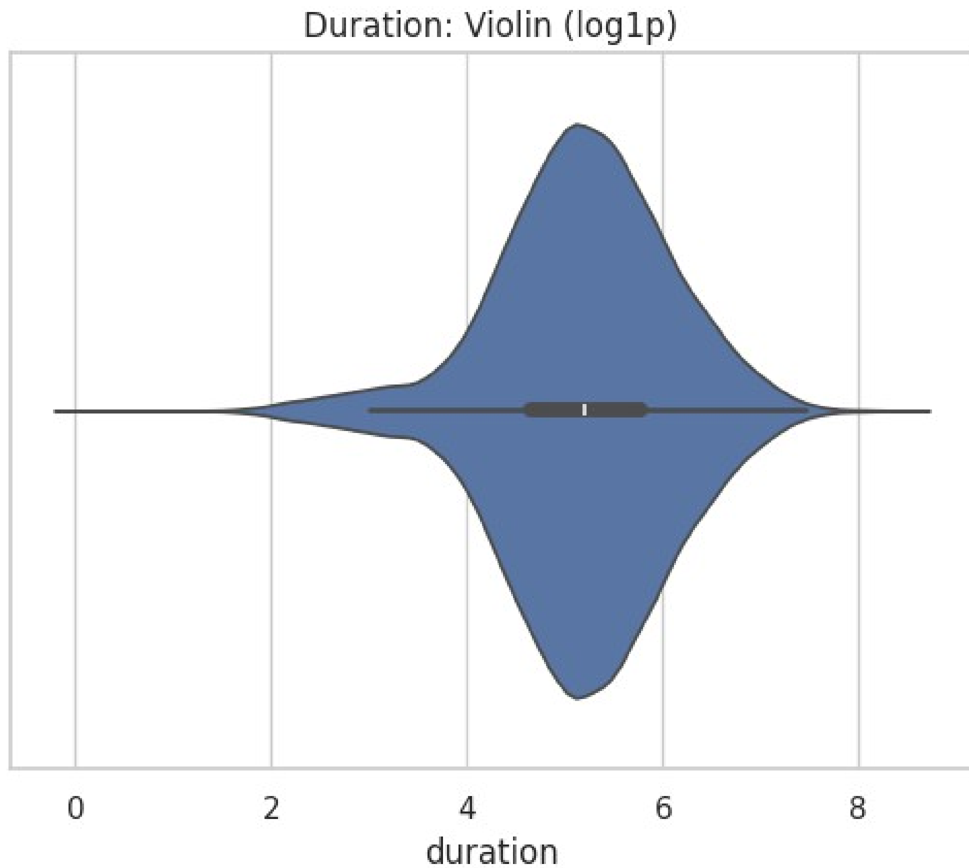

Balance: Clipped Histogram (1–99th pct)
Count
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000
balance



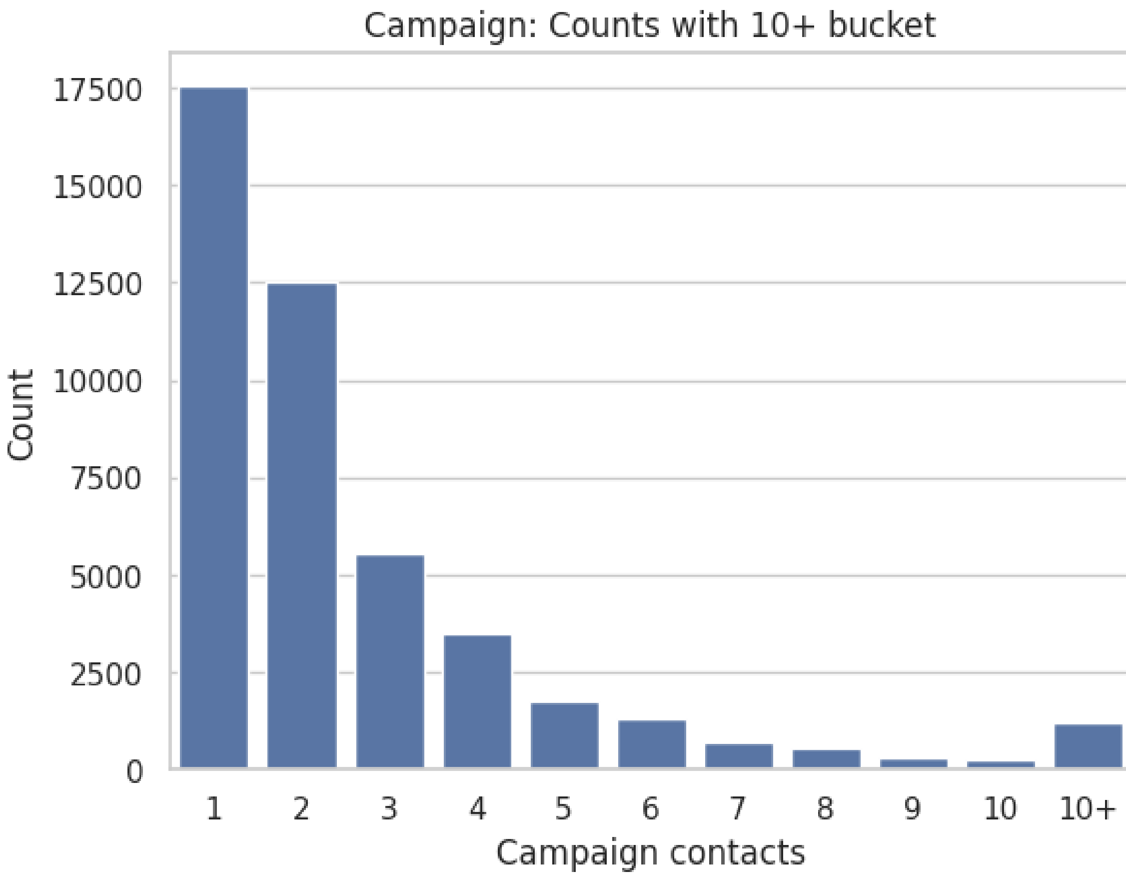
Balance: Symlog Histogram (handles negatives & extremes)
Count
balance
16000
14000
12000
10000
8000
6000
4000
2000
0
-10^4
-10^3
-10^2
0
10^2
10^3
10^4
10^5

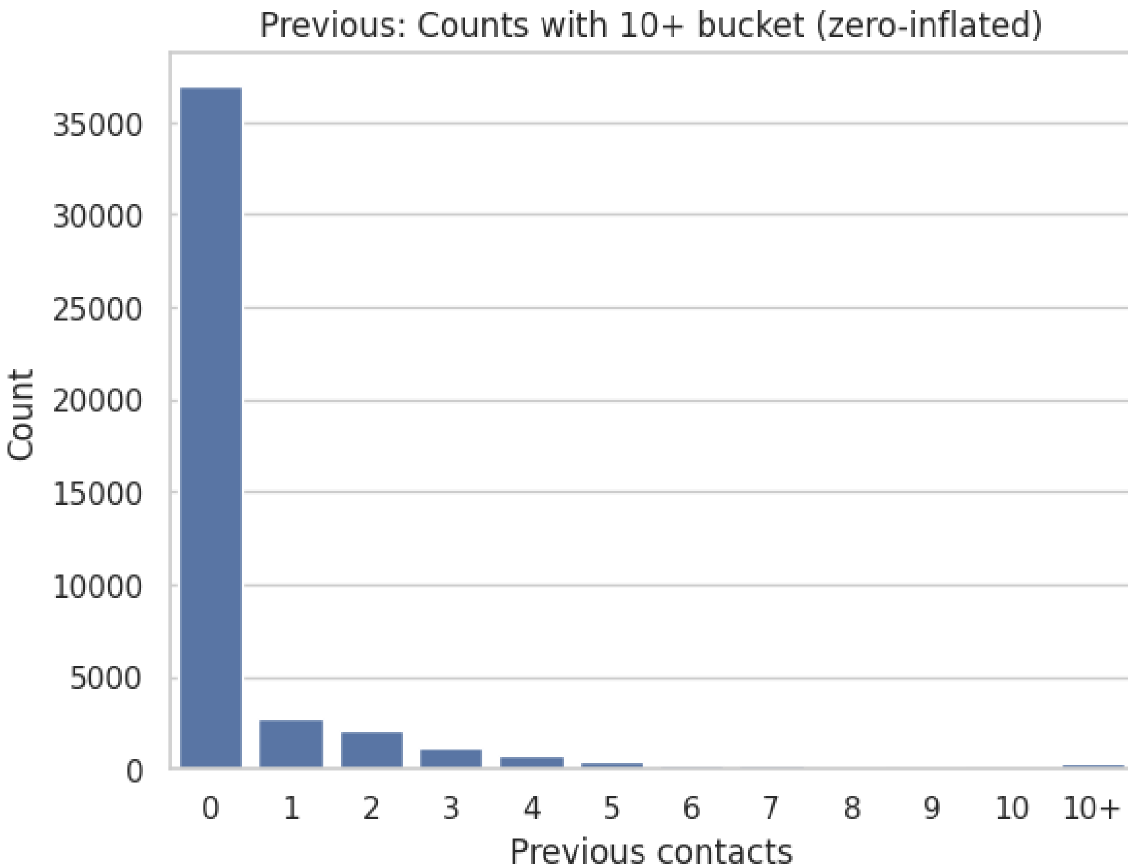

Balance: Boxplot (clipped)
0
2000
4000
6000
8000
10000
12000
balance

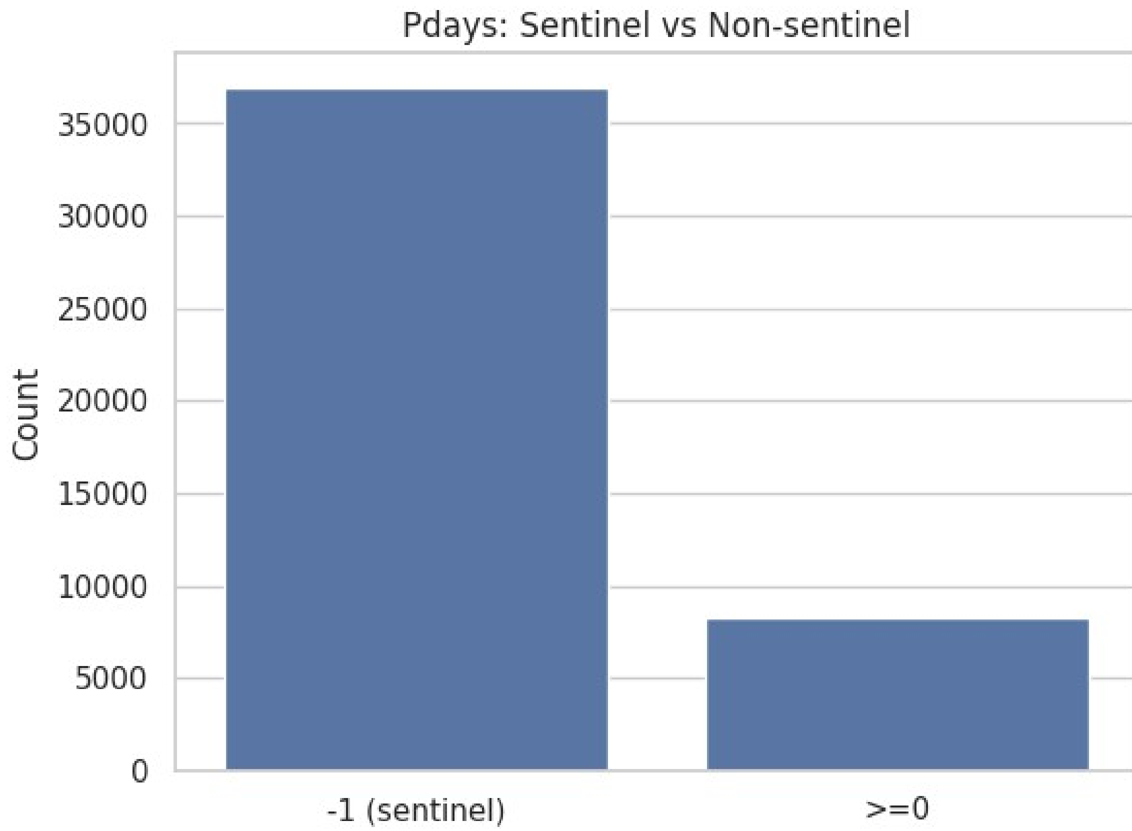

Duration: Histogram of log1p(duration)
Count
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
duration



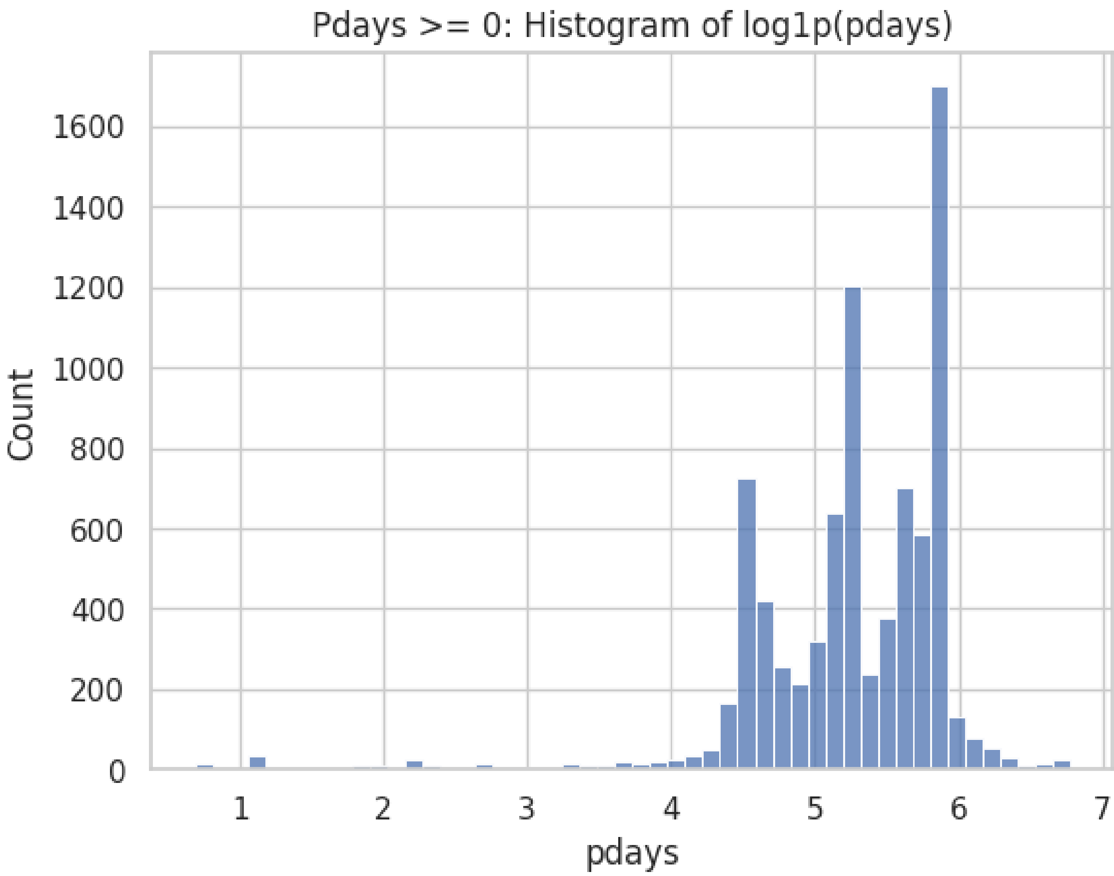
Duration: Violin (log1p)
0
2
4
6
8
duration


Campaign: Counts with 10+ bucket
17500
15000
12500
10000
7500
5000
2500
0
Count
1
2
3
4
5
6
7
8
9
10
10+
Campaign contacts

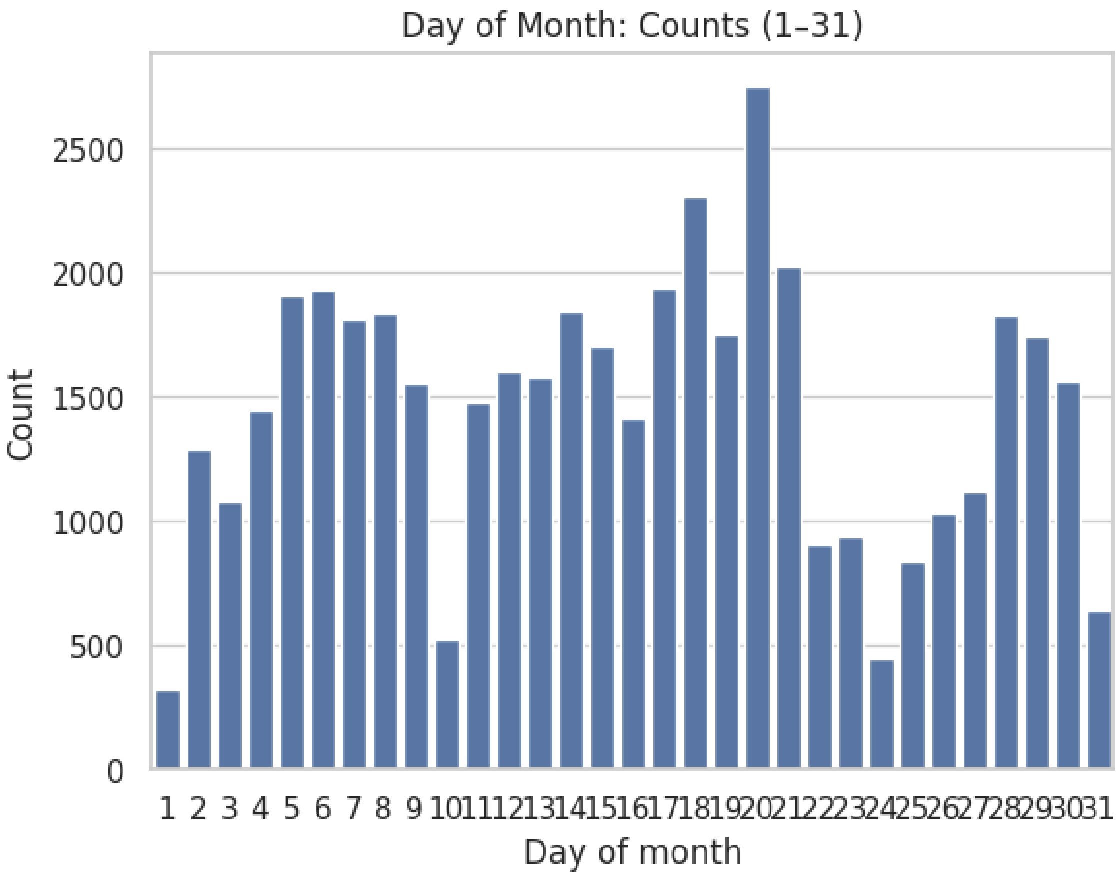

Previous: Counts with 10+ bucket (zero-inflated)
Count
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8
9
10
10+
Previous contacts



Pdays: Sentinel vs Non-sentinel
35000
30000
25000
20000
15000
10000
5000
0
Count
-1 (sentinel)
>=0



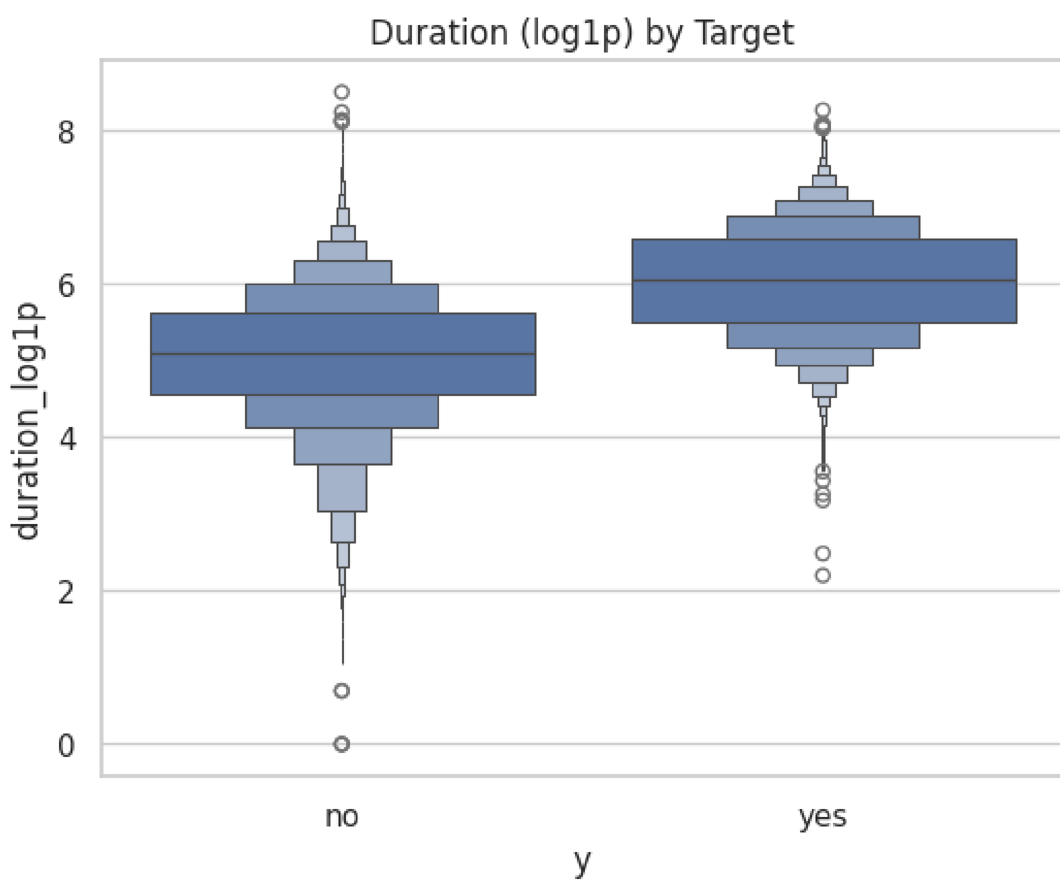
Pdays >= 0: Histogram of log1p(pdays)
Count
0
200
400
600
800
1000
1200
1400
1600
1
2
3
4
5
6
7
pdays



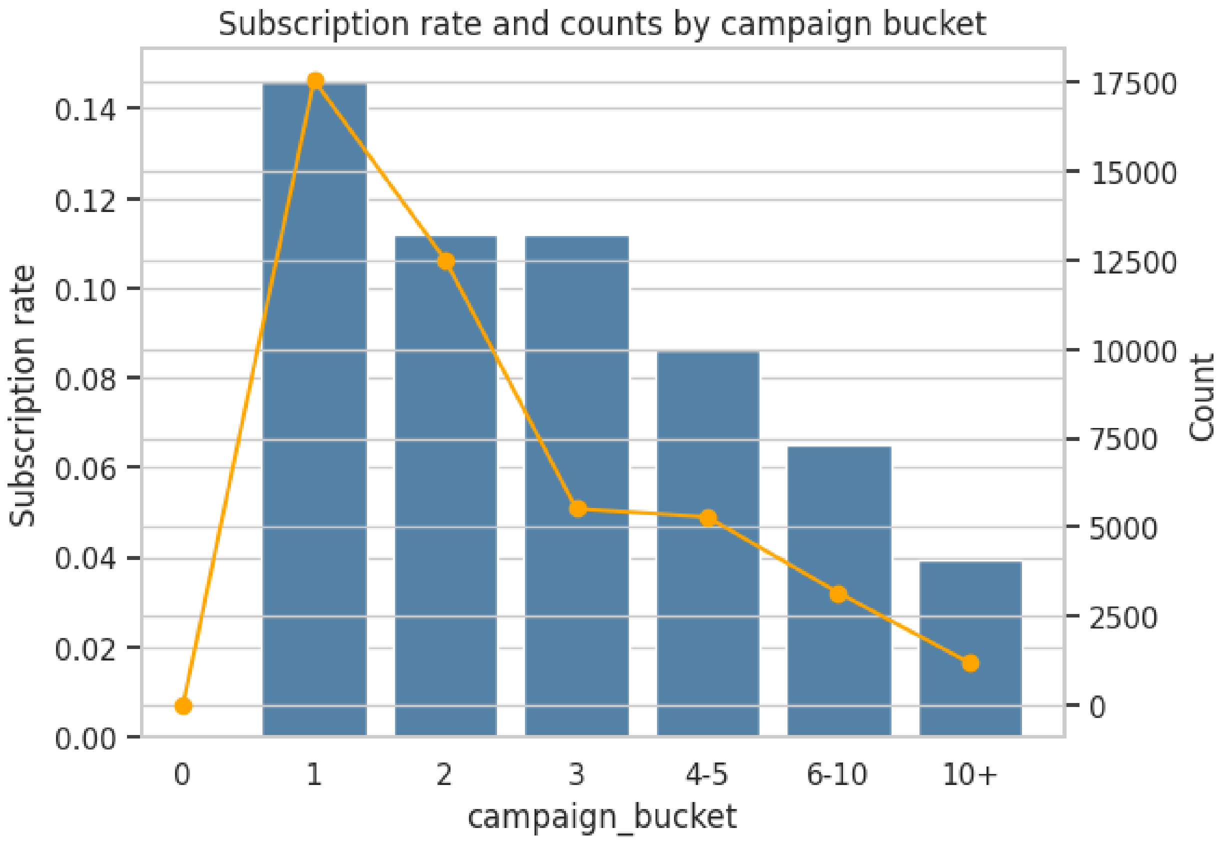

```
import numpy as np, pandas as pd, seaborn as sns, matplotlib.pyplot as
plt
sns.set(style="whitegrid")


df['y_bin'] = (df['y'].str.lower() == 'yes').astype(int)
df['pdays_sentinel'] = (df['pdays'] == -1).astype(int)
df['pdays_pos'] = df['pdays'].clip(lower=0)
df['duration_log1p'] = np.log1p(df['duration'])
df['balance_log1p_signed'] = np.sign(df['balance']) *
np.log1p(np.abs(df['balance']))
df['campaign_bucket'] = pd.cut(df['campaign'], bins=[-
0.5,0.5,1.5,2.5,3.5,5.5,10.5, 1e9],
                               labels=['0','1','2','3','4-5','6-
10','10+'])
df['previous_bucket'] = pd.cut(df['previous'], bins=[-
0.5,0.5,1.5,5.5,10.5, 1e9],
                               labels=['0','1','2-5','6-10','10+'])
df['day_of_month'] = df['day_of_week']

#  Target rate by duration (log1p) with counts
sns.boxenplot(x='y', y='duration_log1p', data=df, order=['no','yes'])
plt.title('Duration (log1p) by Target')
```

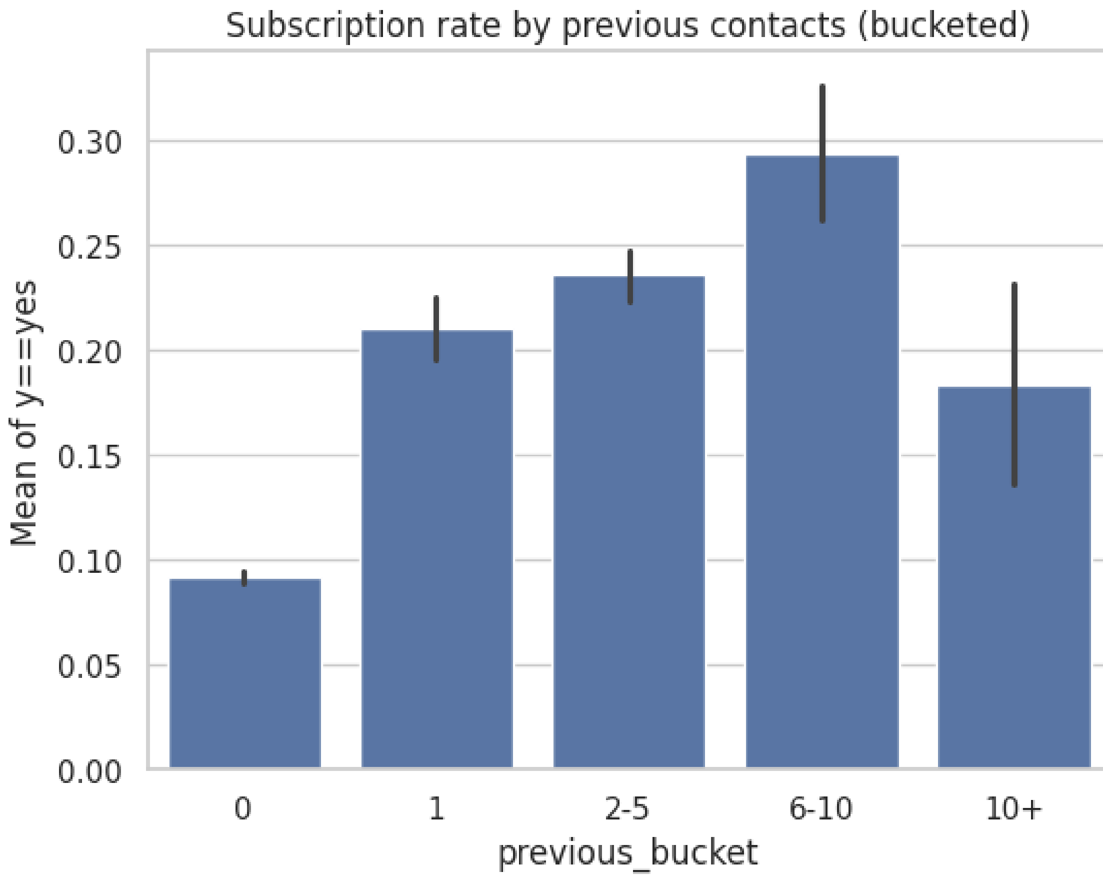
```
plt.show()

#  Target rate by campaign bucket with count overlay
rate = df.groupby('campaign_bucket')['y_bin'].mean().reset_index()
cnt =
df['campaign_bucket'].value_counts().reindex(rate['campaign_bucket']).
reset_index()
fig, ax1 = plt.subplots()
sns.barplot(x='campaign_bucket', y='y_bin', data=rate,
color='steelblue', ax=ax1)
ax1.set_ylabel('Subscription rate')
ax2 = ax1.twinx()
ax2.plot(cnt['campaign_bucket'], cnt['count'], color='orange',
marker='o')
ax2.set_ylabel('Count')
ax1.set_title('Subscription rate and counts by campaign bucket')
plt.show()

#  Target rate by previous bucket
sns.barplot(x='previous_bucket', y='y_bin', data=df,
estimator=np.mean, order=['0','1','2-5','6-10','10+'])
plt.title('Subscription rate by previous contacts (bucketed)')
plt.ylabel('Mean of y==yes')
plt.show()

#  Pdays sentinel vs non-sentinel: rate comparison
sns.barplot(x=df['pdays_sentinel'].map({1:'-1 sentinel',0:'>=0'}),
y='y_bin', data=df, estimator=np.mean)
plt.title('Subscription rate: pdays sentinel vs non-sentinel')
plt.ylabel('Mean of y==yes')
plt.xlabel('')
plt.show()

#  For non-sentinel pdays: rate across log1p(pdays_pos) bins
df_non = df[df['pdays'] >= 0].copy()
df_non['pdays_bin'] = pd.qcut(np.log1p(df_non['pdays_pos']), q=6,
duplicates='drop')
sns.barplot(x='pdays_bin', y='y_bin', data=df_non, estimator=np.mean)
plt.title('Subscription rate across pdays (log1p) quantile bins')
plt.ylabel('Mean of y==yes')
plt.xlabel('log1p(pdays) quantiles')
plt.show()

#  Duration vs campaign (joint): hexbin by target to spot interactions
g = sns.jointplot(
    data=df, x='duration_log1p', y='campaign', kind='hex',
gridsize=30, cmap='Blues'
)
g.fig.suptitle('Duration (log1p) vs Campaign: density')
plt.show()
```

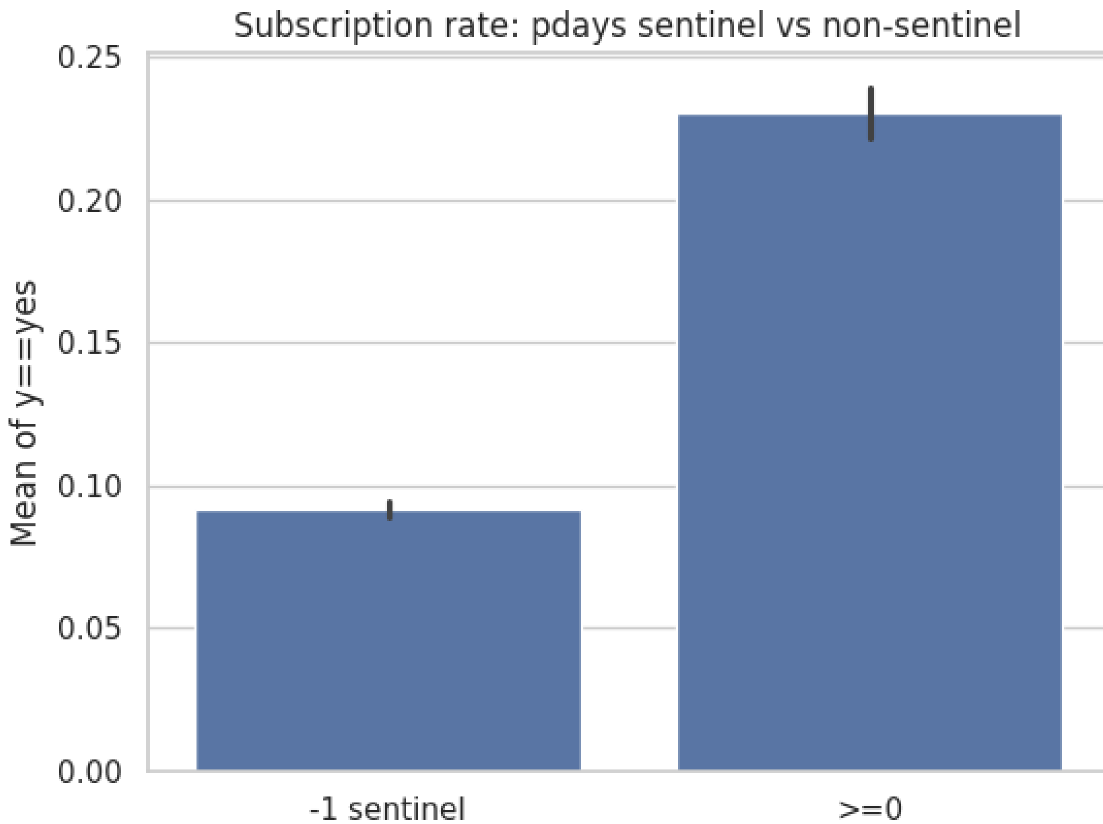
```
#  Categorical drivers: target rate by contact, month, poutcome, job
for col in
['contact','month','poutcome','job','education','marital','housing','l
oan','default']:
    tmp = df.copy()
    tmp[col] = tmp[col].fillna('Missing')
    plt.figure(figsize=(10,4))
    sns.barplot(x=col, y='y_bin', data=tmp, estimator=np.mean,
order=sorted(tmp[col].unique()))
    plt.xticks(rotation=45, ha='right')
    plt.title(f'Subscription rate by {col}')
    plt.ylabel('Mean of y==yes')
    plt.tight_layout(); plt.show()

#  Correlations (Spearman) among numeric features
num_cols =
['age','balance','duration','campaign','pdays_pos','previous']
corr = df[num_cols + ['y_bin']].corr(method='spearman')
print(corr)
```

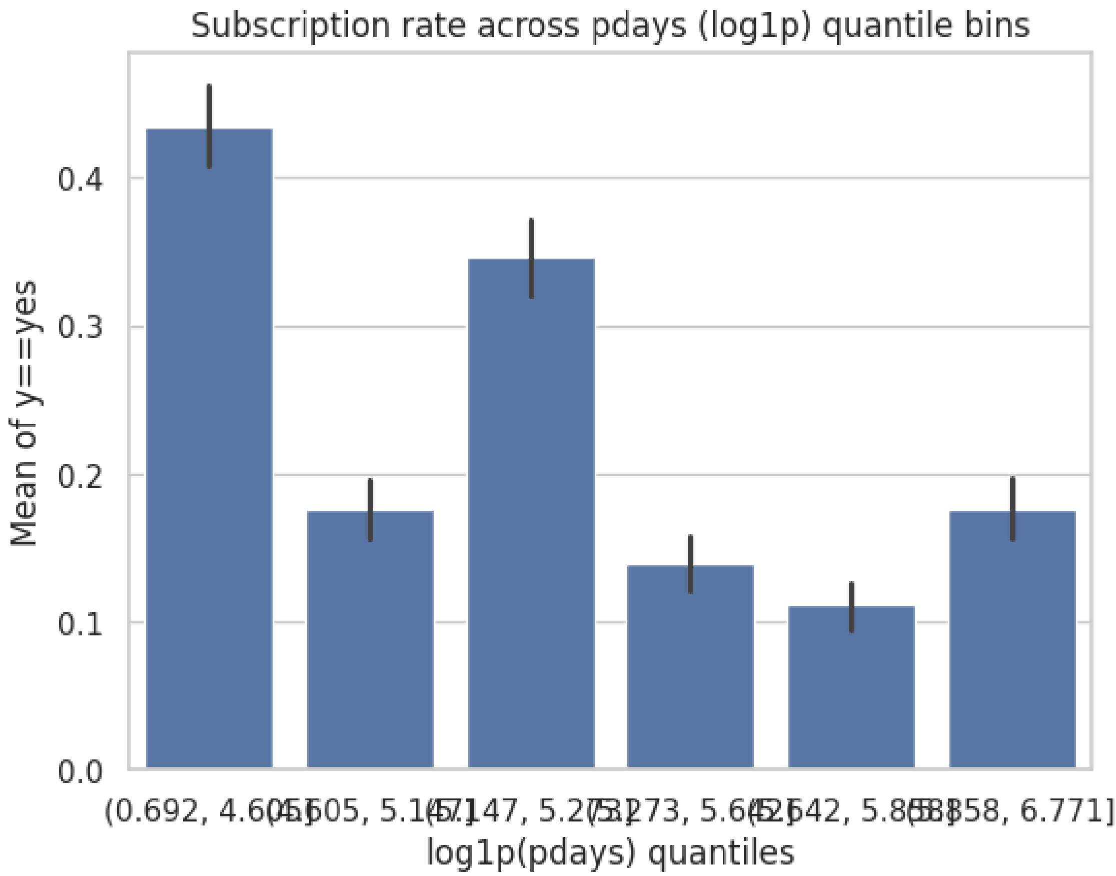


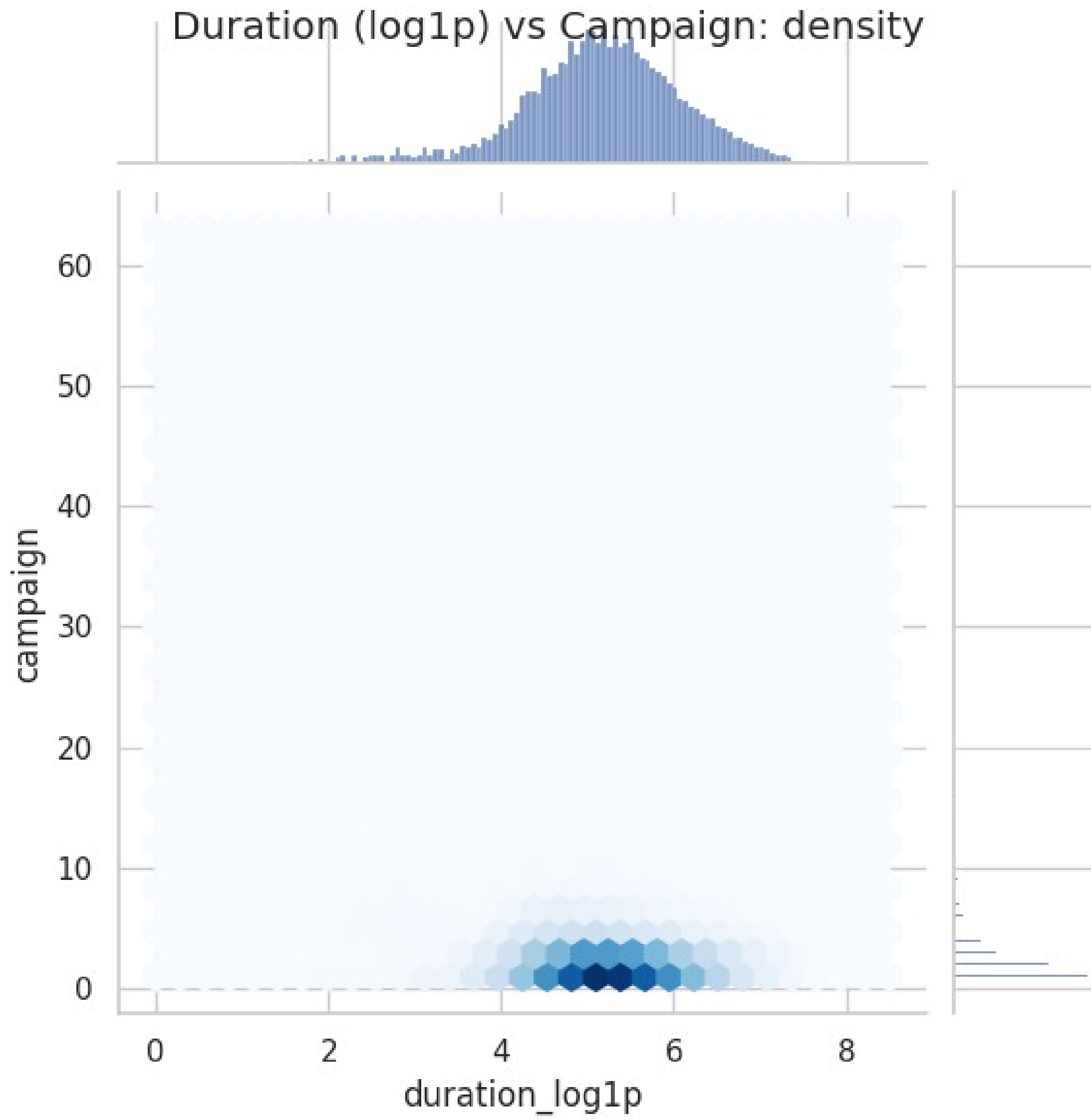
```
/tmp/ipython-input-2771487376.py:22: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
  rate = df.groupby('campaign_bucket')['y_bin'].mean().reset_index()
```

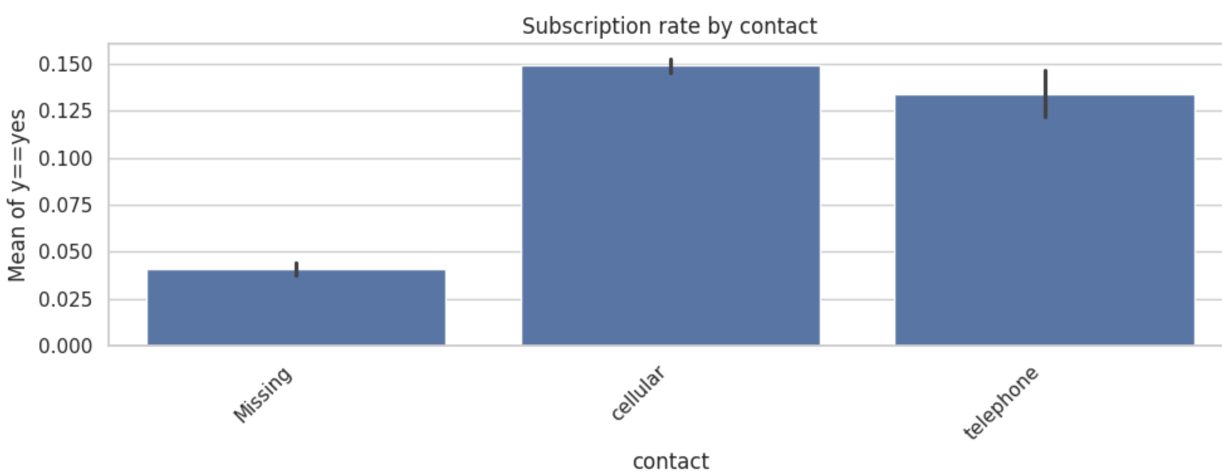


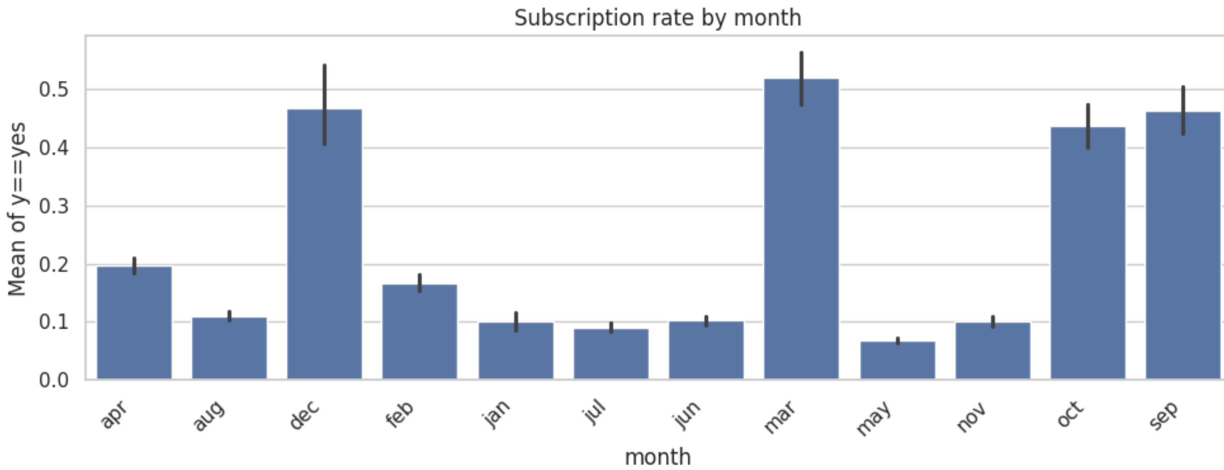

Subscription rate by previous contacts (bucketed)
Mean of y==yes
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
1
2-5
6-10
10+
previous_bucket

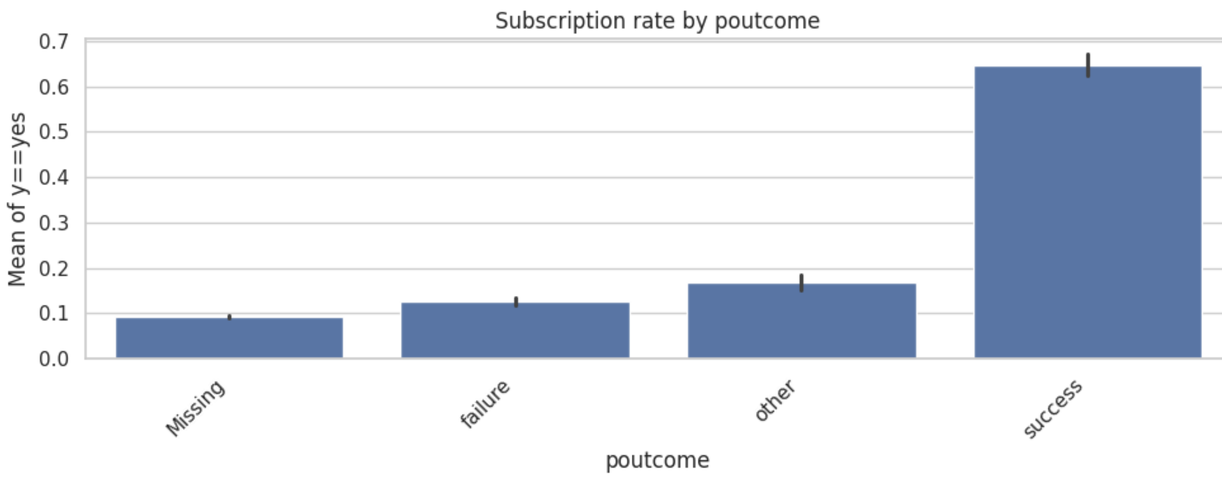

Subscription rate: pdays sentinel vs non-sentinel
Mean of y==yes
0.25
0.20
0.15
0.10
0.05
0.00
-1 sentinel
>=0

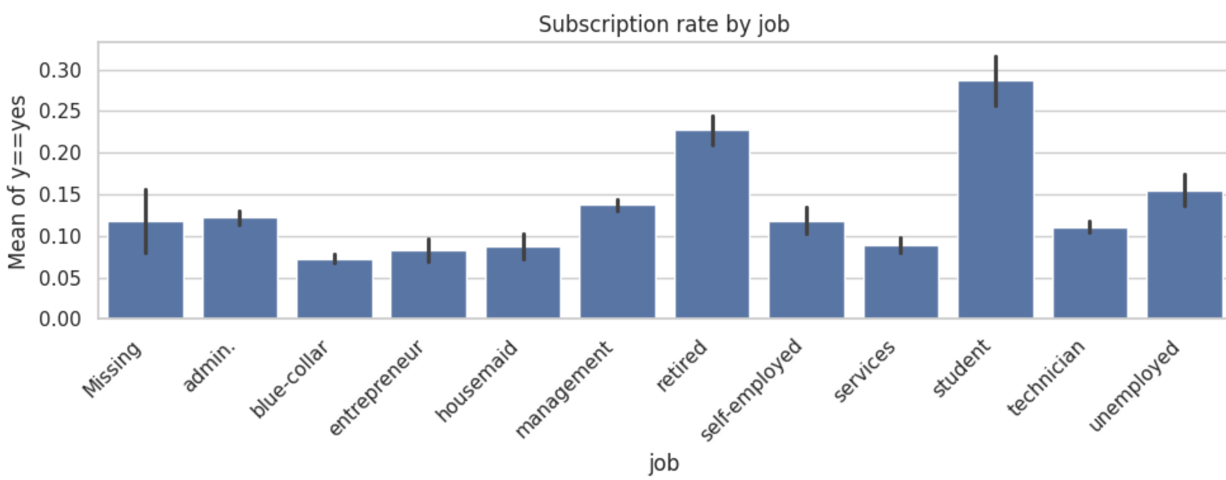

Subscription rate across pdays (log1p) quantile bins
Mean of y==yes
0.0
0.1
0.2
0.3
0.4
log1p(pdays) quantiles
(0.692, 4.605]
(4.605, 5.147]
(5.147, 5.273]
(5.273, 5.642]
(5.642, 5.858]
(5.858, 6.771]

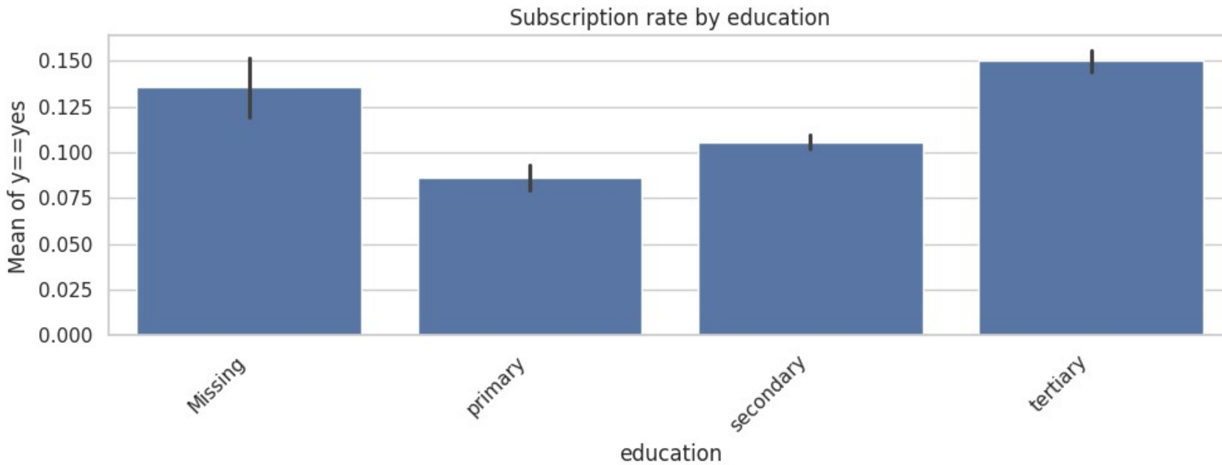

Duration (log1p) vs Campaign: density
campaign
0
10
20
30
40
50
60
0
2
4
6
8
duration_log1p


Subscription rate by contact
Mean of y==yes
0.000
0.025
0.050
0.075
0.100
0.125
0.150
Missing
cellular
telephone
contact

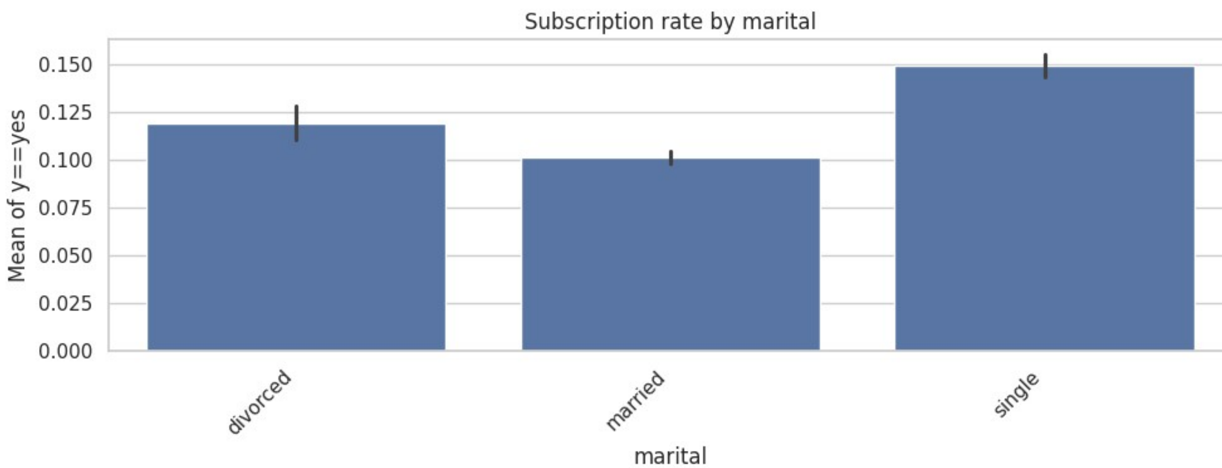

Subscription rate by month
Mean of y==yes
0.0
0.1
0.2
0.3
0.4
0.5
apr
aug
dec
feb
jan
jul
jun
mar
may
nov
oct
sep
month


Subscription rate by poutcome
Mean of y==yes
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Missing
failure
other
success
poutcome

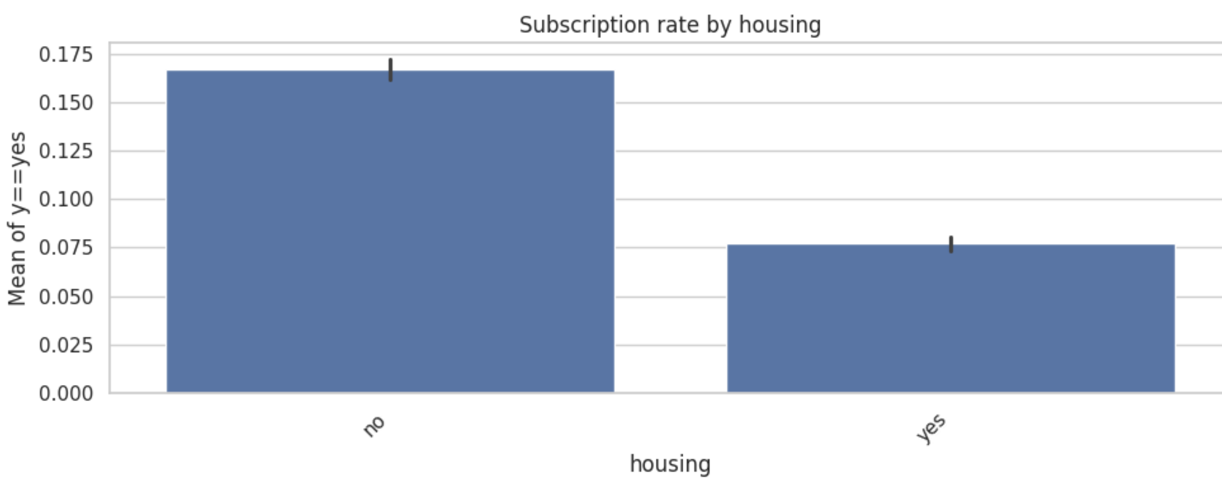

Subscription rate by job
Mean of y==yes
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Missing
admin.
blue-collar
entrepreneur
housemaid
management
retired
self-employed
services
student
technician
unemployed
job



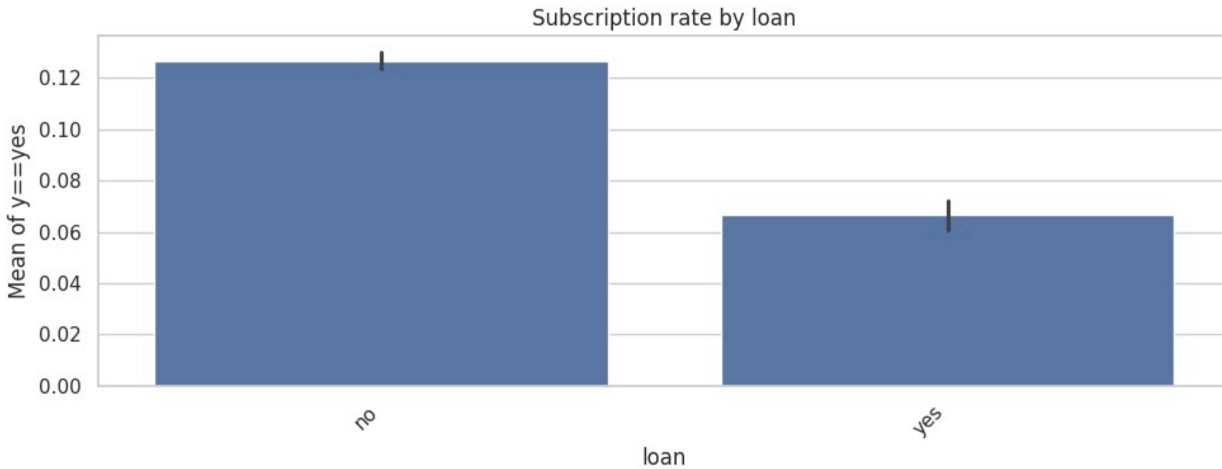
Subscription rate by education
Mean of y==yes
0.000
0.025
0.050
0.075
0.100
0.125
0.150
Missing
primary
secondary
tertiary
education

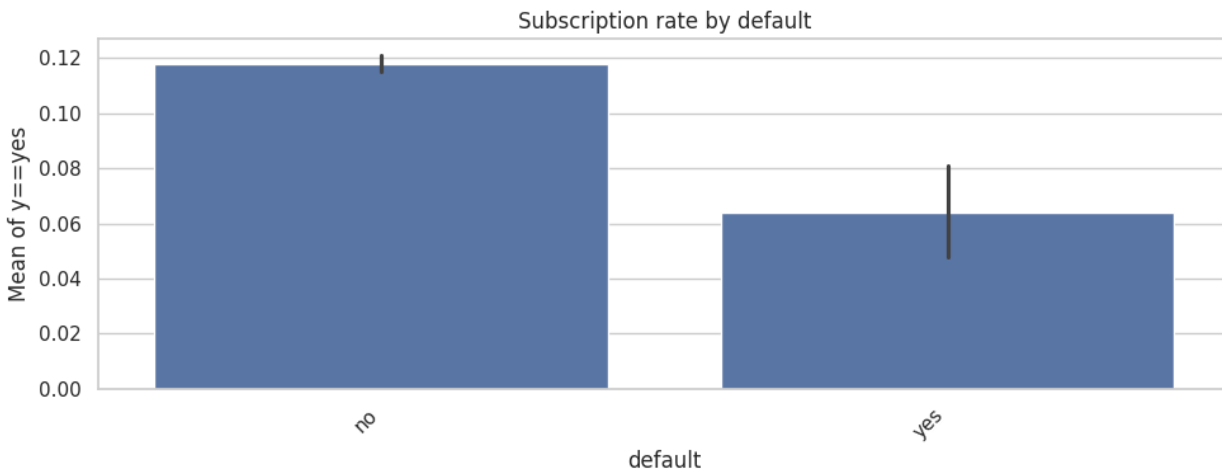


Subscription rate by marital
Mean of y==yes
0.000
0.025
0.050
0.075
0.100
0.125
0.150
divorced
married
single
marital



Subscription rate by housing
Mean of y==yes
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
no
yes
housing






```
                age   balance  duration  campaign  pdays_pos  previous
\
age        1.000000  0.096380 -0.033257  0.037136  -0.017468 -0.011900

balance    0.096380  1.000000  0.042651 -0.030959   0.069676  0.079536

duration  -0.033257  0.042651  1.000000 -0.107962   0.028698  0.031175

campaign   0.037136 -0.030959 -0.107962  1.000000  -0.112284 -0.108448

pdays_pos -0.017468  0.069676  0.028698 -0.112284   1.000000  0.985645

previous  -0.011900  0.079536  0.031175 -0.108448   0.985645  1.000000

y_bin     -0.008750  0.100295  0.342469 -0.084054   0.154055  0.169124


              y_bin
age       -0.008750
balance    0.100295
```

```
duration     0.342469
campaign    -0.084054
pdays_pos    0.154055
previous     0.169124
y_bin        1.000000
```

```python
import numpy as np
import pandas as pd

# Assume df loaded; target y in {'yes','no'}
df = df.copy()

# Remove duplicate columns, if any
df = df.loc[:, ~df.columns.duplicated()]

# Fix misnamed column (use the name that exists in your data)
if 'day_of_week' in df.columns and 'day_of_month' not in df.columns:
    df = df.rename(columns={'day_of_week': 'day_of_month'})
elif 'day' in df.columns and 'day_of_month' not in df.columns:
    df = df.rename(columns={'day': 'day_of_month'})

# Convert 'day_of_month' to numeric after duplicates removed
if 'day_of_month' in df.columns:
    col = df['day_of_month']
    if isinstance(col, pd.DataFrame):
        col = col.iloc[:, 0]
    df['day_of_month'] = pd.to_numeric(col, errors='coerce')

# Encode binary target
df['y_bin'] = (df['y'].str.lower() == 'yes').astype(int)


df['pdays_sentinel'] = (df['pdays'] == -1).astype(int)
df['pdays_pos'] = df['pdays'].clip(lower=0)

# Explicit 'Missing' for categoricals for NA values
cat_cols = ['job', 'marital', 'education', 'default', 'housing',
'loan', 'contact', 'month', 'poutcome']
for c in cat_cols:
    if c in df.columns:
        df[c] = df[c].astype('object').fillna('Missing')


num_cols = ['age', 'balance', 'duration', 'campaign', 'pdays_pos',
'previous', 'day_of_month']
for c in num_cols:
    if c in df.columns:
        col = df[c]
        if isinstance(col, pd.DataFrame):
```

```
            col = col.iloc[:, 0]
        df[c] = pd.to_numeric(col, errors='coerce')

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer


safe_num =
['age','balance','campaign','previous','pdays_pos','day_of_month']
safe_bin = ['pdays_sentinel']
safe_cat =
['job','marital','education','default','housing','loan','contact','mon
th','poutcome',

           ]


def make_preprocessor(num_cols, cat_cols, passthrough_cols=None):
    num_imputer = SimpleImputer(strategy='median')
    cat_imputer = SimpleImputer(strategy='most_frequent')
    ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
    transformers = [
        ('num', num_imputer, num_cols),
        ('cat', Pipeline([('imp', cat_imputer), ('ohe', ohe)]),
cat_cols),
    ]
    if passthrough_cols:
        transformers.append(('pass', 'passthrough', passthrough_cols))
    return ColumnTransformer(transformers=transformers,
remainder='drop')




df['duration_log1p'] = np.log1p(df['duration'])
df['balance_log1p_signed'] = np.sign(df['balance']) *
np.log1p(np.abs(df['balance']))


df['campaign_bucket'] = pd.cut(df['campaign'],
                               bins=[-0.5,0.5,1.5,2.5,3.5,5.5,10.5,
np.inf],
                               labels=['0','1','2','3','4-5','6-
10','10+'])
df['previous_bucket'] = pd.cut(df['previous'],
                               bins=[-0.5,0.5,1.5,5.5,10.5, np.inf],
                               labels=['0','1','2-5','6-10','10+'])
```

```
df['contacts_total'] = df['campaign'] + df['previous']
df['duration_per_contact'] = df['duration'] / (df['campaign'] + 1)


age_bins = [-np.inf, 29, 39, 49, 59, np.inf]
age_labels = ['<30','30-39','40-49','50-59','60+']
df['age_group'] = pd.cut(df['age'], bins=age_bins, labels=age_labels)

df['spending_level'] = pd.qcut(df['balance'], q=[0,.2,.4,.6,.8,1.0],
duplicates='drop',
                               labels=['Q1','Q2','Q3','Q4','Q5'])


cat_cols_extended =
['job','marital','education','default','housing','loan','contact','mon
th','poutcome',

'campaign_bucket','previous_bucket','age_group','spending_level']
for c in cat_cols_extended:
    df[c] = df[c].astype('object').fillna('Missing')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
import sklearn


def make_preprocessor(num_cols, cat_cols, passthrough_cols=None):
    """Create preprocessing pipeline for numeric, categorical, and
passthrough columns."""
    num_imputer = SimpleImputer(strategy='median')
    cat_imputer = SimpleImputer(strategy='most_frequent')


    if sklearn.__version__ >= '1.4':
        ohe = OneHotEncoder(handle_unknown='ignore',
sparse_output=False)
    else:
        ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)

    transformers = [
        ('num', num_imputer, num_cols),
        ('cat', Pipeline([
            ('imputer', cat_imputer),
            ('encoder', ohe)
        ]), cat_cols)
    ]

    if passthrough_cols:
```

```
        transformers.append(('pass', 'passthrough', passthrough_cols))

    preprocessor = ColumnTransformer(transformers)
    return preprocessor



safe_numeric = ['age', 'balance', 'campaign', 'previous', 'pdays_pos',
'day_of_month', 'contacts_total']
safe_binary = ['pdays_sentinel']
safe_cats = cat_cols_extended

full_numeric = safe_numeric + ['duration', 'duration_log1p',
'balance_log1p_signed', 'duration_per_contact']
full_binary = safe_binary
full_cats = cat_cols_extended


y = df['y_bin']


pre_safe = make_preprocessor(num_cols=safe_numeric,
cat_cols=safe_cats, passthrough_cols=safe_binary)
pre_full = make_preprocessor(num_cols=full_numeric,
cat_cols=full_cats, passthrough_cols=full_binary)


def make_splits(X, y, seed=42):
    X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.30, stratify=y, random_state=seed)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.50, stratify=y_temp, random_state=seed)
    return X_train, X_val, X_test, y_train, y_val, y_test

X_safe = df[safe_numeric + safe_binary + safe_cats]
X_full = df[full_numeric + full_binary + full_cats]

X_safe_train, X_safe_val, X_safe_test, y_train, y_val, y_test =
make_splits(X_safe, y)
X_full_train, X_full_val, X_full_test, _, _, _ = make_splits(X_full,
y)


pre_safe.fit(X_safe_train)
pre_full.fit(X_full_train)


print({
    'baseline_train': y_train.mean(),
    'baseline_val': y_val.mean(),
```

```python
    'baseline_test': y_test.mean()
})
```

```
{'baseline_train': np.float64(0.11697791259835055), 'baseline_val':
np.float64(0.11707460925980537), 'baseline_test':
np.float64(0.11692716012975524)}
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score, average_precision_score


clf_gini = Pipeline(steps=[
    ('prep', pre_safe),
    ('model', DecisionTreeClassifier(
        criterion='gini',
        class_weight='balanced',
        min_samples_leaf=10,
        min_samples_split=20,
        random_state=42
    ))
])


clf_gini.fit(X_safe_train, y_train)
for name, X, y in [('VAL', X_safe_val, y_val), ('TEST', X_safe_test,
y_test)]:
    y_prob = clf_gini.predict_proba(X)[:, 1]
    y_pred = clf_gini.predict(X)
    print(f'[{name}] ROC AUC:', roc_auc_score(y, y_prob))
    print(f'[{name}] PR AUC:', average_precision_score(y, y_prob))
```

```
[VAL] ROC AUC: 0.7213473967245996
[VAL] PR AUC: 0.33194087847916376
[TEST] ROC AUC: 0.7139311941586057
[TEST] PR AUC: 0.33877403539026885
```

```python
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, average_precision_score


Xtr_prepared = pre_safe.fit_transform(X_safe_train)
base = DecisionTreeClassifier(criterion='gini',
                              class_weight='balanced',
                              random_state=42)
path = base.cost_complexity_pruning_path(Xtr_prepared, y_train)
ccp_alphas = path.ccp_alphas[:-1]
```

```python
if ccp_alphas.size == 0:
    ccp_grid = [0.0]
else:
    qs = np.unique(np.quantile(ccp_alphas, np.linspace(0, 0.9, 12)))
    ccp_grid = np.unique(np.r_[0.0, qs])


pipe = Pipeline(steps=[
    ('prep', pre_safe),
    ('clf', DecisionTreeClassifier(
        criterion='gini',
        class_weight='balanced',
        random_state=42
    ))
])

param_grid = {
    'clf__max_depth': [None, 4, 6, 8, 10],
    'clf__min_samples_split': [10, 20, 40],
    'clf__ccp_alpha': ccp_grid
}

grid = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    n_jobs=-1,
    refit=True
)

grid.fit(X_safe_train, y_train)
print('Best params:', grid.best_params_)
print('Best CV ROC AUC:', grid.best_score_)


best = grid.best_estimator_
for name, X, y in [('VAL', X_safe_val, y_val), ('TEST', X_safe_test,
y_test)]:
    y_prob = best.predict_proba(X)[:, 1]
    print(f'[{name}] ROC AUC:', roc_auc_score(y, y_prob))
    print(f'[{name}] PR AUC:', average_precision_score(y, y_prob))
```

```
Best params: {'clf__ccp_alpha': np.float64(0.00016686789104638532),
'clf__max_depth': 10, 'clf__min_samples_split': 40}
Best CV ROC AUC: 0.7655419360241804
[VAL] ROC AUC: 0.7626656545669005
[VAL] PR AUC: 0.39393661174630046
[TEST] ROC AUC: 0.7630004524899263
[TEST] PR AUC: 0.4008164734200761
```

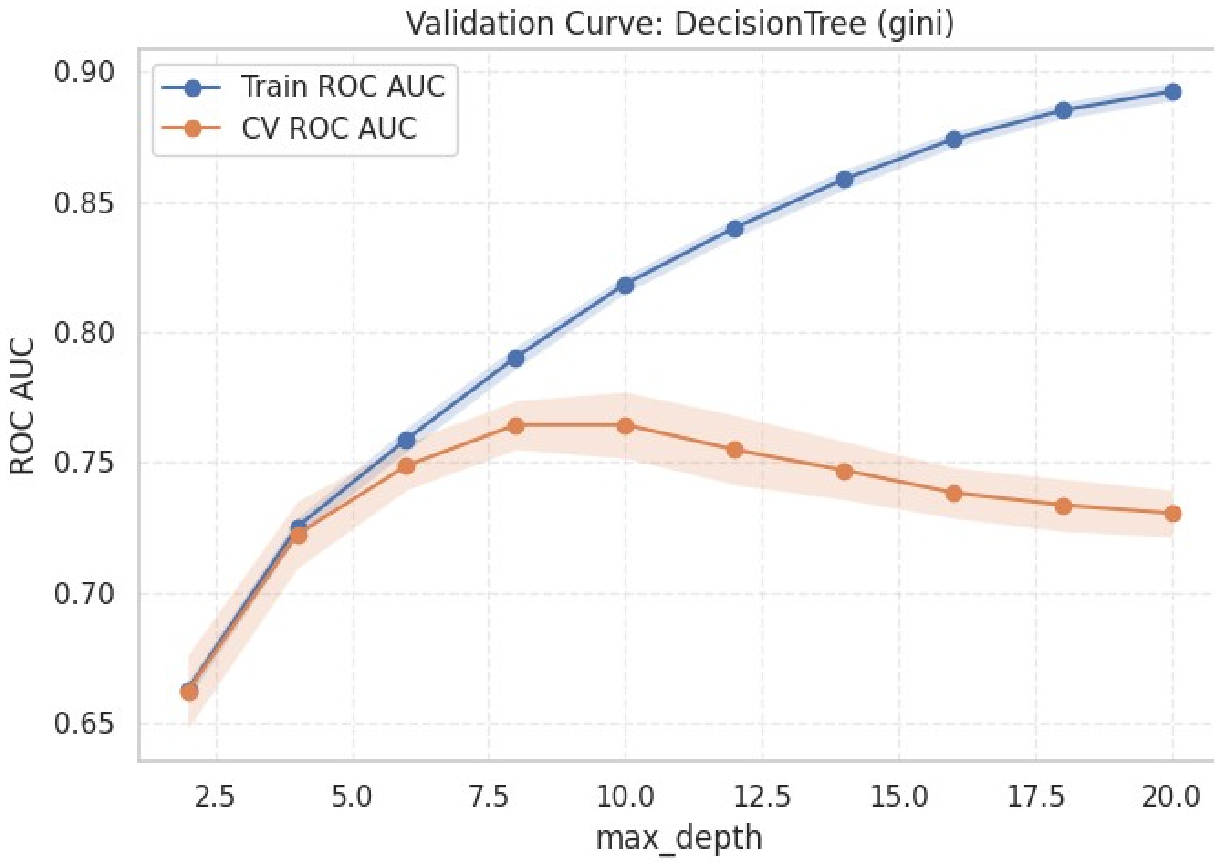
```python
from sklearn.metrics import accuracy_score

def eval_overfit(model, X_tr, y_tr, X_val, y_val, X_te, y_te,
label='MODEL'):
    # Predictions
    y_tr_pred  = model.predict(X_tr)
    y_val_pred = model.predict(X_val)
    y_te_pred  = model.predict(X_te)

    # Accuracies
    acc_tr  = accuracy_score(y_tr, y_tr_pred)
    acc_val = accuracy_score(y_val, y_val_pred)
    acc_te  = accuracy_score(y_te, y_te_pred)


    gap_val = acc_tr - acc_val
    gap_te  = acc_tr - acc_te

    print(f'\n[{label}]')
    print(f'Train accuracy: {acc_tr:.4f}')
    print(f'Valid accuracy: {acc_val:.4f}  (train–val gap: {gap_val:
+.4f})')
    print(f'Test  accuracy: {acc_te:.4f}  (train–test gap: {gap_te:
+.4f})')


    if acc_tr > 0.95 and acc_te < acc_tr - 0.05:
        print('Hint: likely overfitting (very high train, noticeably
lower test).')
    elif acc_tr < 0.70 and acc_te < 0.70:
        print('Hint: likely underfitting (both train and test low).')
    else:
        print('Hint: bias–variance tradeoff looks reasonable.')


try:
    eval_overfit(safe_tree, X_safe_train, y_train, X_safe_val, y_val,
X_safe_test, y_test, label='BASELINE_SAFE')
except NameError:
    pass


try:
    best = grid.best_estimator_
    eval_overfit(best, X_safe_train, y_train, X_safe_val, y_val,
X_safe_test, y_test, label='TUNED_SAFE')
except NameError:
    pass
```

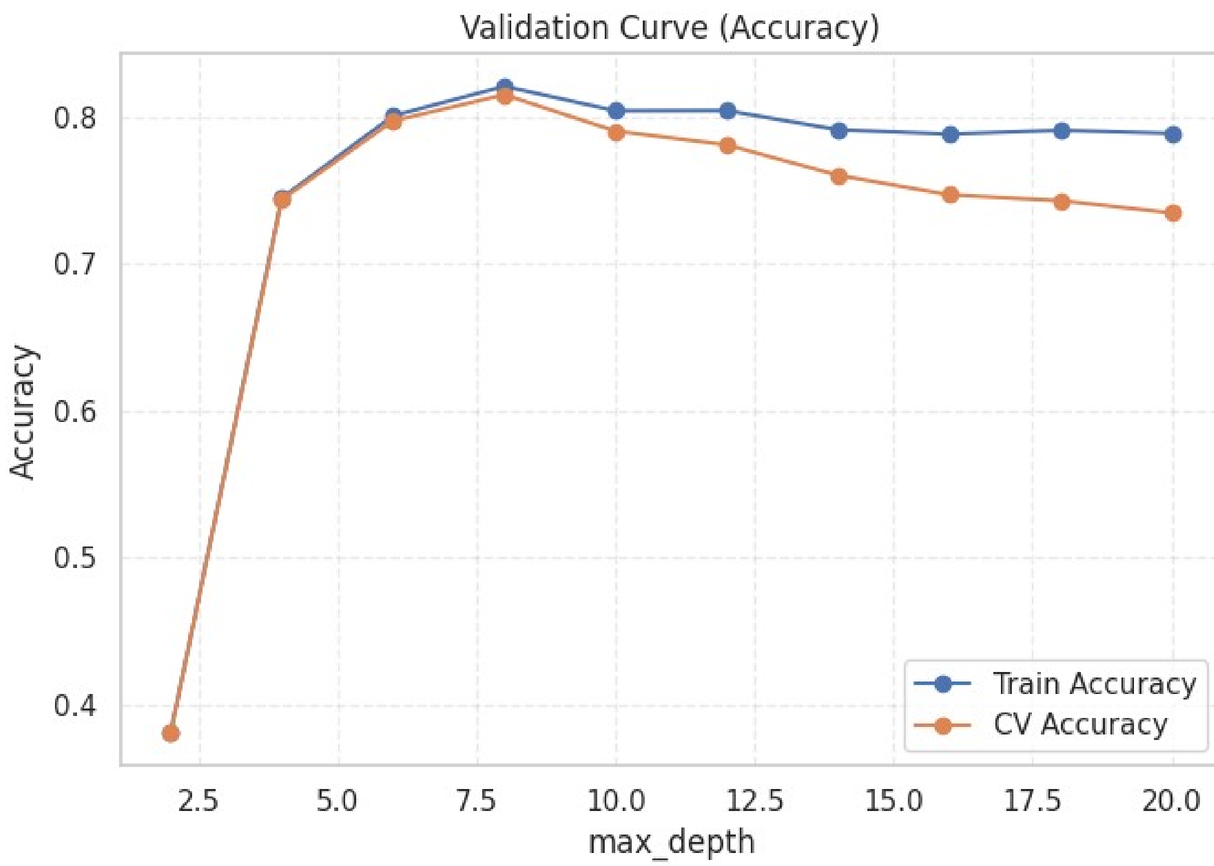
```
try:
    eval_overfit(full_tree, X_full_train, y_train, X_full_val, y_val,
X_full_test, y_test, label='BASELINE_FULL')
except NameError:
    pass
```

```
[TUNED_SAFE]
Train accuracy: 0.7786
Valid accuracy: 0.7740  (train–val gap: +0.0046)
Test  accuracy: 0.7710  (train–test gap: +0.0075)
Hint: bias–variance tradeoff looks reasonable.
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import validation_curve, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier


best_min_samples_split = 40
best_ccp_alpha = 0.00016721678065369867
param_range = np.arange(2, 21, 2)

pipe = Pipeline(steps=[
    ('prep', pre_safe),
    ('clf', DecisionTreeClassifier(
        criterion='gini',
        class_weight='balanced',
        min_samples_split=best_min_samples_split,
        ccp_alpha=best_ccp_alpha,
        random_state=42
    ))
])


cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
train_scores, val_scores = validation_curve(
    estimator=pipe,
    X=X_safe_train, y=y_train,
    param_name='clf__max_depth',
    param_range=param_range,
    scoring='roc_auc',
    cv=cv,
    n_jobs=-1
)

train_mean = train_scores.mean(axis=1)
train_std  = train_scores.std(axis=1)
val_mean   = val_scores.mean(axis=1)
```

```
val_std    = val_scores.std(axis=1)

plt.figure(figsize=(7,5))
plt.plot(param_range, train_mean, marker='o', label='Train ROC AUC')
plt.fill_between(param_range, train_mean-train_std,
train_mean+train_std, alpha=0.2)
plt.plot(param_range, val_mean, marker='o', label='CV ROC AUC')
plt.fill_between(param_range, val_mean-val_std, val_mean+val_std,
alpha=0.2)
plt.xlabel('max_depth')
plt.ylabel('ROC AUC')
plt.title('Validation Curve: DecisionTree (gini)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()


train_scores_acc, val_scores_acc = validation_curve(
    estimator=pipe,
    X=X_safe_train, y=y_train,
    param_name='clf__max_depth',
    param_range=param_range,
    scoring='accuracy',
    cv=cv,
    n_jobs=-1
)

plt.figure(figsize=(7,5))
plt.plot(param_range, train_scores_acc.mean(axis=1), marker='o',
label='Train Accuracy')
plt.plot(param_range, val_scores_acc.mean(axis=1), marker='o',
label='CV Accuracy')
plt.xlabel('max_depth'); plt.ylabel('Accuracy'); plt.title('Validation
Curve (Accuracy)')
plt.legend(); plt.grid(True, linestyle='--', alpha=0.4);
plt.tight_layout(); plt.show()
```


Validation Curve: DecisionTree (gini)
Train ROC AUC
CV ROC AUC
ROC AUC
0.90
0.85
0.80
0.75
0.70
0.65
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_depth




```
import numpy as np, pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV,
StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score,
average_precision_score
from scipy.stats import randint, loguniform



def evaluate(pipe, X_tr, y_tr, X_va, y_va, label):
    pipe.fit(X_tr, y_tr)
    y_tr_pred = pipe.predict(X_tr); y_va_pred = pipe.predict(X_va)
    y_va_prob = pipe.predict_proba(X_va)[:,1]
    print(f'\n[{label}]')
    print('Train acc:', accuracy_score(y_tr, y_tr_pred))
    print('Val   acc:', accuracy_score(y_va, y_va_pred))
    print('Val ROC AUC:', roc_auc_score(y_va, y_va_prob))
    print('Val PR  AUC:', average_precision_score(y_va, y_va_prob))
    return pipe
```

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


tree_pipe = Pipeline([
    ('prep', pre_safe),
    ('clf', DecisionTreeClassifier(
        criterion='gini',
        class_weight='balanced',
        random_state=42
    ))
])


tree_param_dist = {
    'clf__max_depth': [None, 4, 6, 8, 10],
    'clf__min_samples_split': randint(5, 61),
    'clf__ccp_alpha': np.r_[0.0, loguniform(1e-5, 1e-2).rvs(1000)]
[:100]
}

tree_search = RandomizedSearchCV(
    estimator=tree_pipe,
    param_distributions=tree_param_dist,
    n_iter=40,
    scoring='roc_auc',
    cv=cv,
    random_state=42,
    n_jobs=-1,
    refit=True
)
tree_search.fit(X_safe_train, y_train)
best_tree = tree_search.best_estimator_
_ = evaluate(best_tree, X_safe_train, y_train, X_safe_val, y_val,
'DecisionTree (random search)')
print('Best tree params:', tree_search.best_params_)
```

```
[DecisionTree (random search)]
Train acc: 0.7946724808038677
Val   acc: 0.7974048953111177
Val ROC AUC: 0.7755175548410002
Val PR  AUC: 0.3992007676456041
Best tree params: {'clf__ccp_alpha': np.float64(0.000411320862932397),
'clf__max_depth': 10, 'clf__min_samples_split': 25}
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

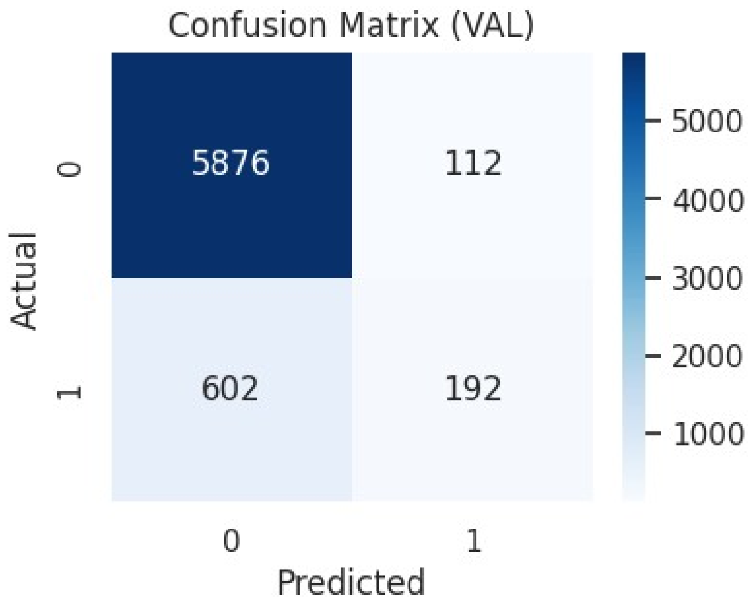
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV,
StratifiedKFold, validation_curve, learning_curve
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import (roc_auc_score, average_precision_score,
accuracy_score,
                             confusion_matrix, precision_recall_curve,
                             f1_score, precision_score, recall_score)

sns.set(style="whitegrid")




X_train_pre = pre_safe.fit_transform(X_safe_train)
X_val_pre   = pre_safe.transform(X_safe_val)
X_test_pre  = pre_safe.transform(X_safe_test)


if hasattr(pre_safe, 'get_feature_names_out'):
    feat_names = pre_safe.get_feature_names_out()
else:
    feat_names = X_safe_train.columns


gb_base = GradientBoostingClassifier(
    loss='log_loss',
    learning_rate=0.1,
    n_estimators=300,
    max_depth=3,
    subsample=1.0,
    random_state=42
)
gb_base.fit(X_train_pre, y_train)
val_prob = gb_base.predict_proba(X_val_pre)[:,1]
print('[GB-BASE] Val ROC AUC:', roc_auc_score(y_val, val_prob),
      'PR AUC:', average_precision_score(y_val, val_prob))


param_distributions = {
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators':  [100, 300, 500],
    'max_depth':     [2, 3, 4],
    'subsample':     [1.0, 0.8, 0.6]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

gb_search = RandomizedSearchCV(
    estimator=GradientBoostingClassifier(loss='log_loss',
```

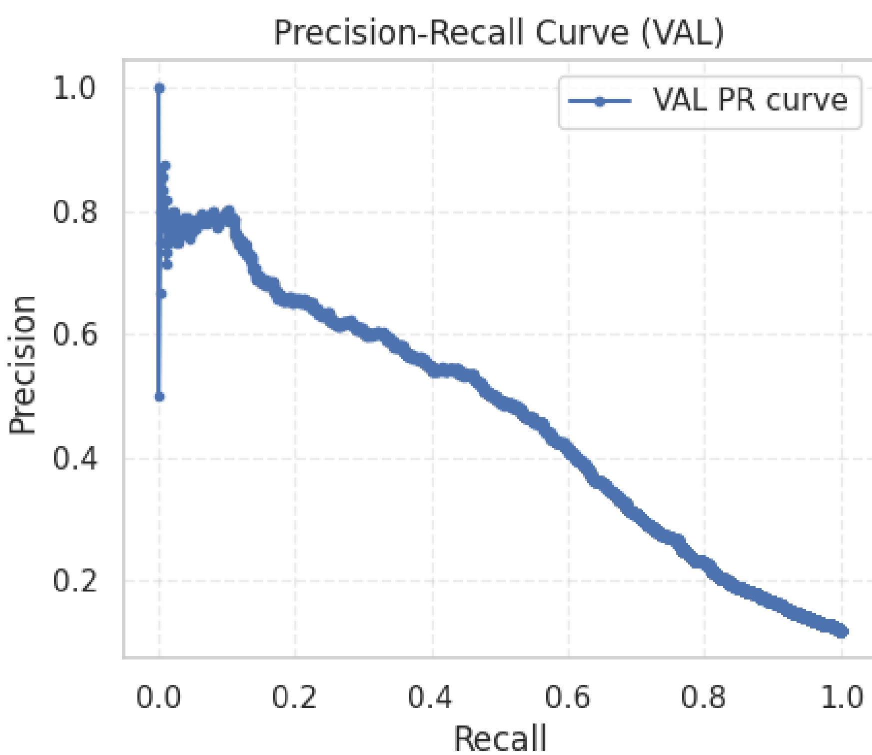
```
random_state=42),
    param_distributions=param_distributions,
    n_iter=10,
    scoring='roc_auc',
    cv=cv,
    n_jobs=-1,
    random_state=42,
    refit=True
)
gb_search.fit(X_train_pre, y_train)
best_gb = gb_search.best_estimator_
print('Best GB params:', gb_search.best_params_)


def eval_metrics(y_true, y_pred, y_prob, tag='VAL'):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    auprc = average_precision_score(y_true, y_prob)
    print(f"[{tag}] ACC: {acc:.3f}, Precision: {prec:.3f}, Recall: 
{rec:.3f}, F1: {f1:.3f}, AUCPR: {auprc:.3f}")
    return acc, prec, rec, f1, auprc

for tag, X, y in [('VAL', X_val_pre, y_val), ('TEST', X_test_pre, 
y_test)]:
    y_prob = best_gb.predict_proba(X)[:,1]
    y_pred = best_gb.predict(X)
    eval_metrics(y, y_pred, y_prob, tag)


def plot_confusion(y_true, y_pred, tag='VAL'):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4,3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix ({tag})')
    plt.xlabel('Predicted'); plt.ylabel('Actual')
    plt.show()

def plot_pr_curve(y_true, y_prob, tag='VAL'):
    precision, recall, _ = precision_recall_curve(y_true, y_prob)
    plt.figure(figsize=(5,4))
    plt.plot(recall, precision, marker='.', label=f'{tag} PR curve')
    plt.xlabel('Recall'); plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve ({tag})')
    plt.legend(); plt.grid(True, ls='--', alpha=0.4)
    plt.show()

for tag, X, y in [('VAL', X_val_pre, y_val), ('TEST', X_test_pre, 
y_test)]:
```

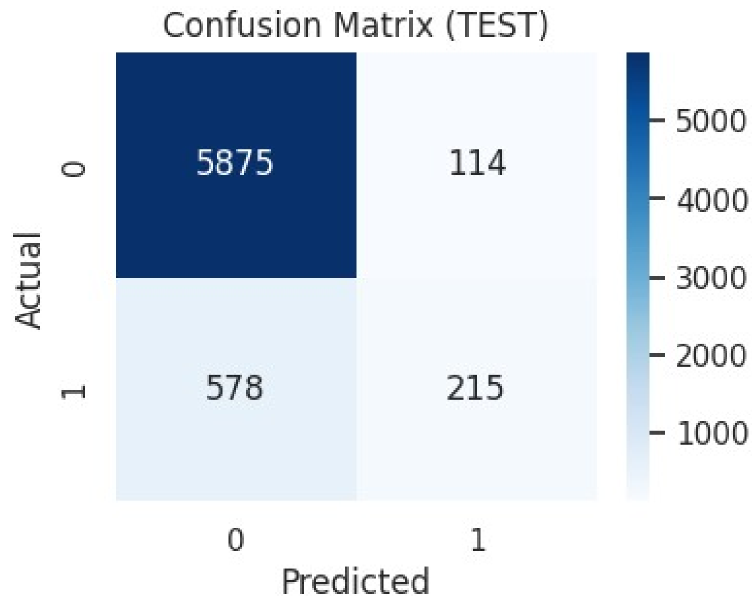
```
    y_pred = best_gb.predict(X)
    y_prob = best_gb.predict_proba(X)[:,1]
    plot_confusion(y, y_pred, tag)
    plot_pr_curve(y, y_prob, tag)


def plot_feature_importance(model, feature_names, top_n=20):
    importances = model.feature_importances_
    indices = np.argsort(importances)[-top_n:][::-1]
    plt.figure(figsize=(6,4))
    sns.barplot(x=importances[indices], y=np.array(feature_names)
[indices])
    plt.title('Top Feature Importances')
    plt.xlabel('Importance'); plt.ylabel('Feature')
    plt.show()

plot_feature_importance(best_gb, feat_names)


train_sizes, train_scores, val_scores = learning_curve(
    best_gb, X_train_pre, y_train,
    cv=cv, scoring='roc_auc', n_jobs=-1,
    train_sizes=np.linspace(0.1,1.0,5)
)
plt.figure(figsize=(6,4))
plt.plot(train_sizes, train_scores.mean(axis=1), marker='o',
label='Train ROC AUC')
plt.plot(train_sizes, val_scores.mean(axis=1), marker='o',
label='Validation ROC AUC')
plt.xlabel('Training set size'); plt.ylabel('ROC AUC')
plt.title('Learning Curve')
plt.legend(); plt.grid(True, ls='--', alpha=0.4)
plt.show()


train_loss, val_loss = [], []
for y_train_pred in best_gb.staged_predict_proba(X_train_pre):
    train_loss.append(-np.mean(y_train*np.log(y_train_pred[:,1]) + (1-
y_train)*np.log(1-y_train_pred[:,1])))
for y_val_pred in best_gb.staged_predict_proba(X_val_pre):
    val_loss.append(-np.mean(y_val*np.log(y_val_pred[:,1]) + (1-
y_val)*np.log(1-y_val_pred[:,1])))

plt.figure(figsize=(6,4))
plt.plot(range(1, len(train_loss)+1), train_loss, label='Train Loss')
plt.plot(range(1, len(val_loss)+1), val_loss, label='Validation Loss')
plt.xlabel('Boosting Iteration'); plt.ylabel('Log Loss')
plt.title('Training vs Validation Loss over Boosting Iterations')
plt.legend(); plt.grid(True, ls='--', alpha=0.4)
plt.show()
```

```python
for lr in [0.01, 0.1, 0.3]:
    tmp_gb = GradientBoostingClassifier(
        learning_rate=lr,
        n_estimators=best_gb.n_estimators,
        max_depth=best_gb.max_depth,
        subsample=best_gb.subsample,
        random_state=42
    )
    tmp_gb.fit(X_train_pre, y_train)
    y_prob = tmp_gb.predict_proba(X_val_pre)[:,1]
    print(f'Learning Rate {lr}: Val ROC AUC={roc_auc_score(y_val,
y_prob):.3f}, AUCPR={average_precision_score(y_val, y_prob):.3f}')


n_estimators_range = [50, 100, 200, 400, 600, 800]
val_aucs = []
for n in n_estimators_range:
    tmp_gb = GradientBoostingClassifier(
        learning_rate=best_gb.learning_rate,
        n_estimators=n,
        max_depth=best_gb.max_depth,
        subsample=best_gb.subsample,
        random_state=42
    )
    tmp_gb.fit(X_train_pre, y_train)
    y_prob = tmp_gb.predict_proba(X_val_pre)[:,1]
    val_aucs.append(roc_auc_score(y_val, y_prob))

plt.figure(figsize=(6,4))
plt.plot(n_estimators_range, val_aucs, marker='o')
plt.xlabel('Number of Estimators'); plt.ylabel('Validation ROC AUC')
plt.title('Effect of n_estimators on Validation Performance (Bias-
Variance)')
plt.grid(True, ls='--', alpha=0.4)
plt.show()
```

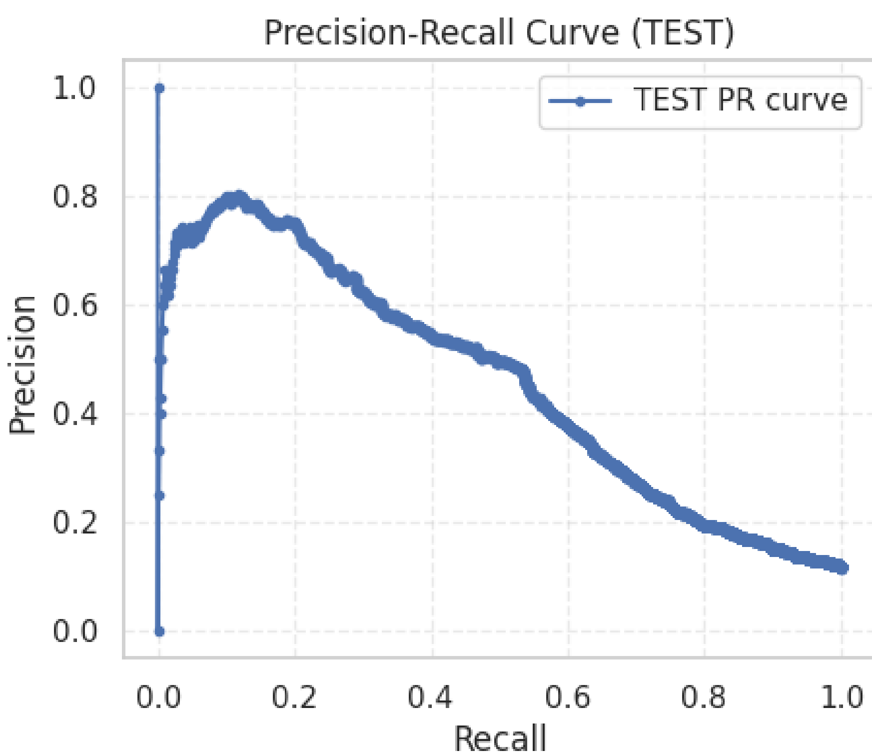
```
[GB-BASE] Val ROC AUC: 0.8050121022902228 PR AUC: 0.45753762176308116
Best GB params: {'subsample': 0.8, 'n_estimators': 300, 'max_depth':
4, 'learning_rate': 0.05}
[VAL] ACC: 0.895, Precision: 0.632, Recall: 0.242, F1: 0.350, AUCPR:
0.464
[TEST] ACC: 0.898, Precision: 0.653, Recall: 0.271, F1: 0.383, AUCPR:
0.458
```


Confusion Matrix (VAL)
5876
112
602
192
Actual
0
1
Predicted
0
1
5000
4000
3000
2000
1000

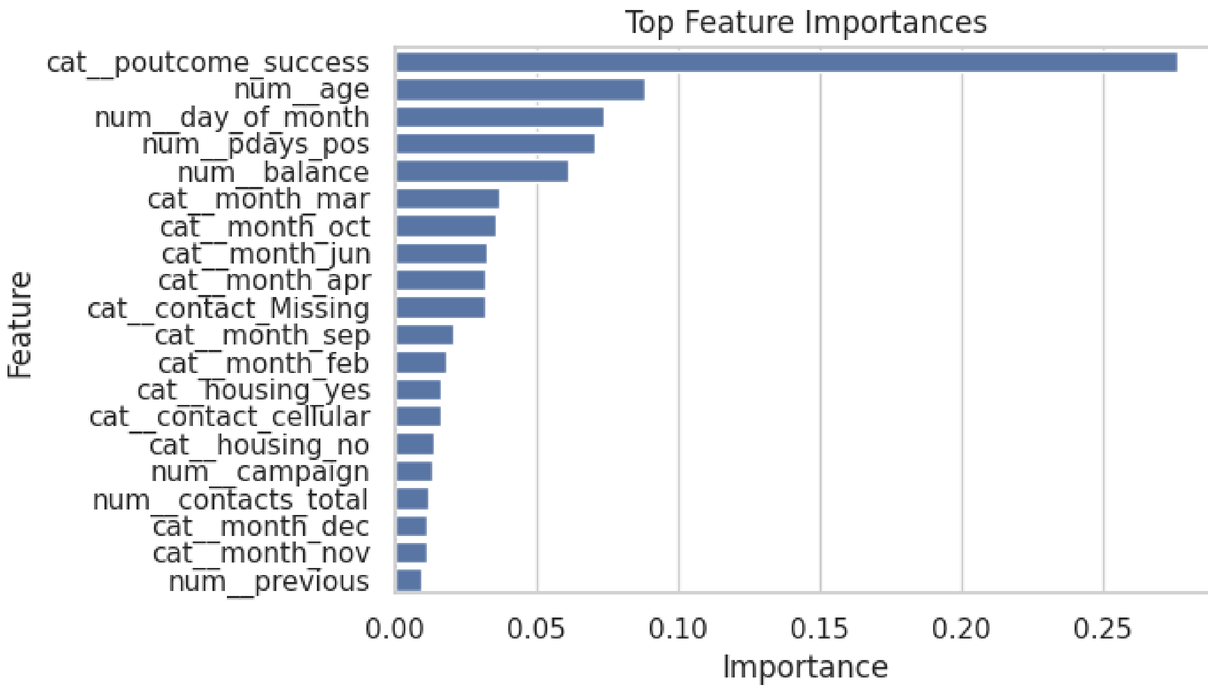

Precision-Recall Curve (VAL)
VAL PR curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0

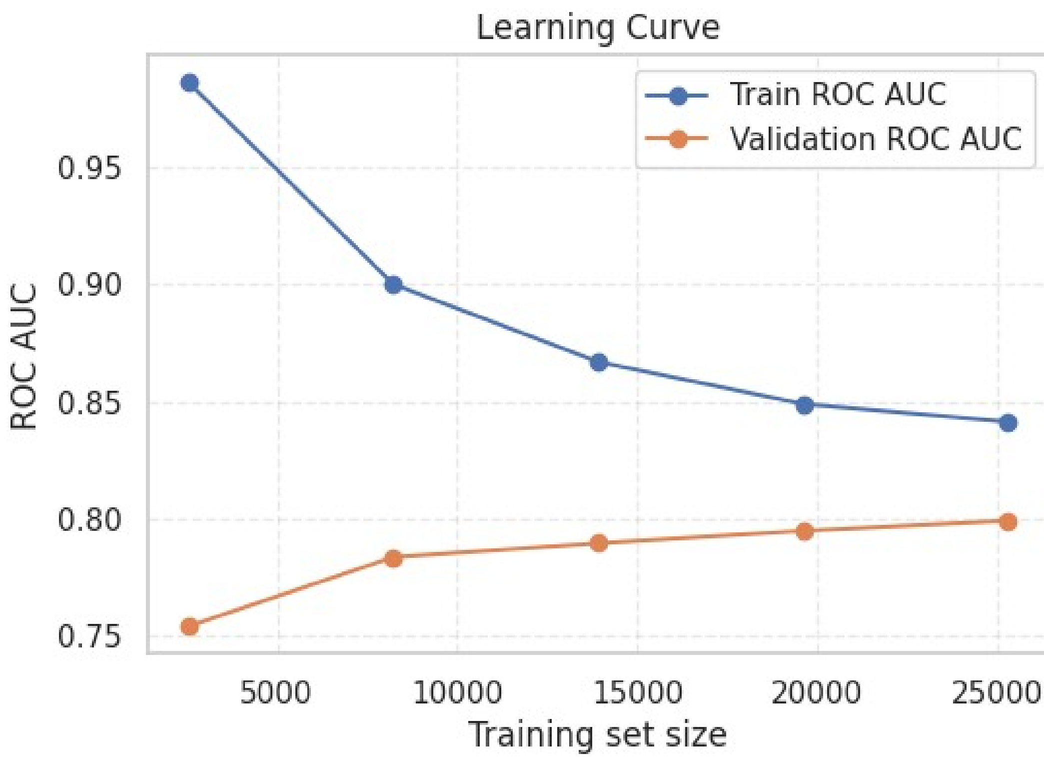


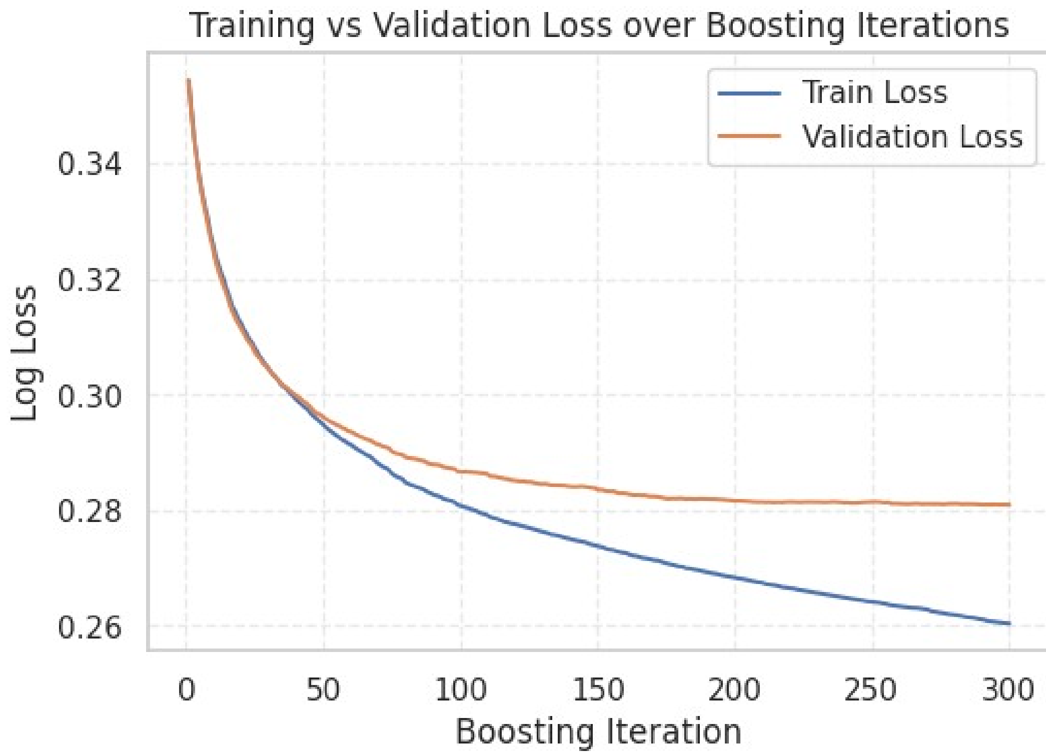
Confusion Matrix (TEST)
5875
114
578
215
Actual
Predicted
0
1
0
1
5000
4000
3000
2000
1000



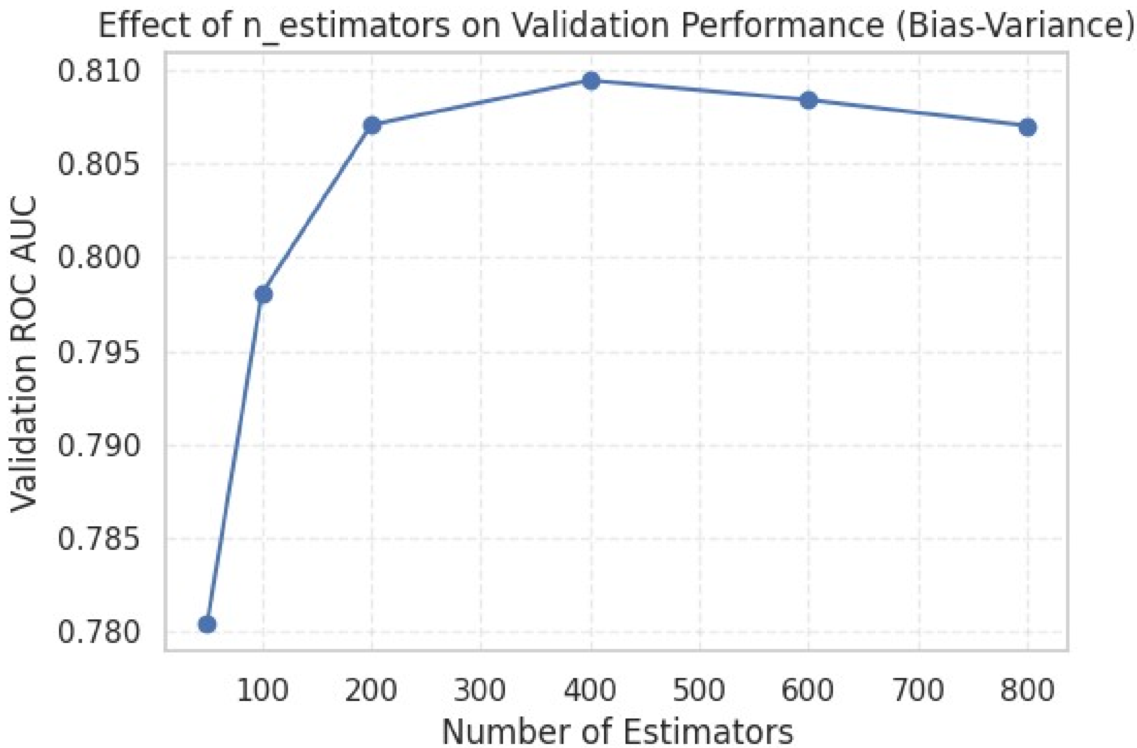
Precision-Recall Curve (TEST)
TEST PR curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Top Feature Importances
cat__poutcome_success
num__age
num__day_of_month
num__pdays_pos
num__balance
cat__month_mar
cat__month_oct
cat__month_jun
cat__month_apr
cat__contact_Missing
cat__month_sep
cat__month_feb
cat__housing_yes
cat__contact_cellular
cat__housing_no
num__campaign
num__contacts_total
cat__month_dec
cat__month_nov
num__previous
Feature
0.00
0.05
0.10
0.15
0.20
0.25
Importance


Learning Curve
Train ROC AUC
Validation ROC AUC
ROC AUC
0.95
0.90
0.85
0.80
0.75
5000
10000
15000
20000
25000
Training set size




```
Learning Rate 0.01: Val ROC AUC=0.784, AUCPR=0.444
Learning Rate 0.1: Val ROC AUC=0.806, AUCPR=0.451
Learning Rate 0.3: Val ROC AUC=0.783, AUCPR=0.378
```



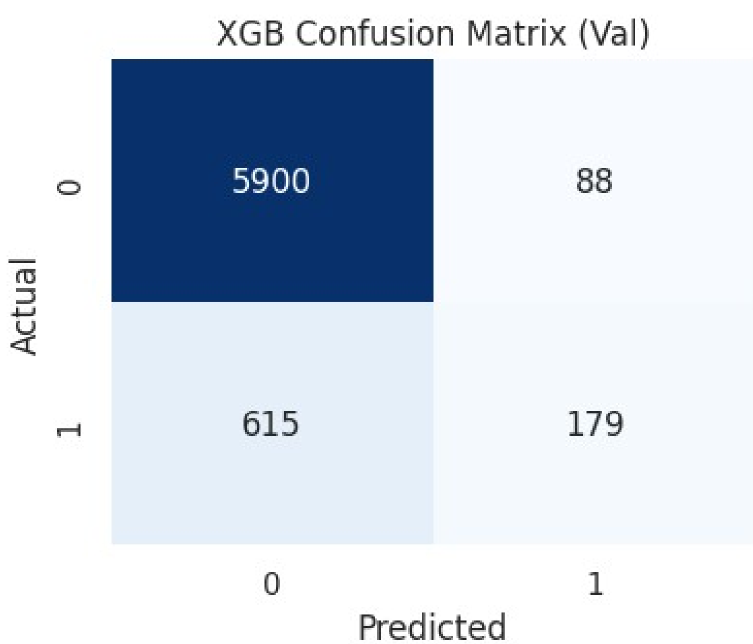

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as
sns
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, precision_recall_curve, average_precision_score,
roc_auc_score
)
import xgboost as xgb

sns.set(style="whitegrid")


Xtr = pre_safe.fit_transform(X_safe_train, y_train)
Xva = pre_safe.transform(X_safe_val)
Xte = pre_safe.transform(X_safe_test)

dtrain = xgb.DMatrix(Xtr, label=y_train)
dval   = xgb.DMatrix(Xva, label=y_val)
dtest  = xgb.DMatrix(Xte, label=y_test)


params = {
    'objective': 'binary:logistic',
    'eval_metric': ['logloss','aucpr','auc'],
    'eta': 0.1,
    'max_depth': 3,
```

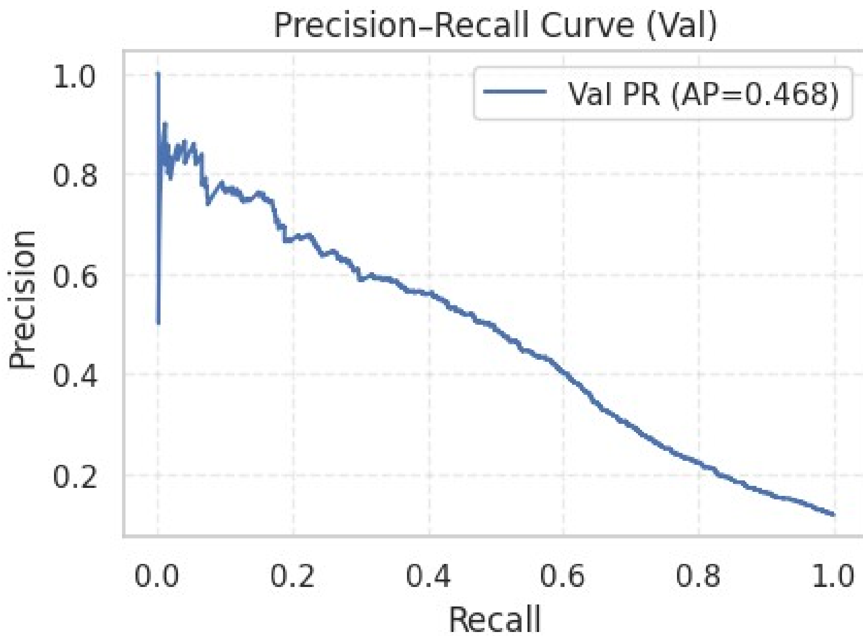
```
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'lambda': 1.0,
    'tree_method': 'hist'
}
evals = [(dtrain, 'train'), (dval, 'val')]
booster = xgb.train(
    params=params,
    dtrain=dtrain,
    num_boost_round=5000,
    evals=evals,
    early_stopping_rounds=50,
    verbose_eval=False
)
best_iteration = booster.best_iteration


def report(dmat, y_true, tag):
    prob = booster.predict(dmat, iteration_range=(0, best_iteration +
1))
    pred = (prob >= 0.5).astype(int)
    acc  = accuracy_score(y_true, pred)
    prec = precision_score(y_true, pred, zero_division=0)
    rec  = recall_score(y_true, pred, zero_division=0)
    f1   = f1_score(y_true, pred, zero_division=0)
    aucpr = average_precision_score(y_true, prob)
    auc   = roc_auc_score(y_true, prob)
    print(f'[{tag}] ACC={acc:.3f} PREC={prec:.3f} REC={rec:.3f}
F1={f1:.3f} AUCPR={aucpr:.3f} AUROC={auc:.3f}')
    return prob, pred

vprob, vpred = report(dval, y_val, 'VAL')
tprob, tpred = report(dtest, y_test, 'TEST')


plt.figure(figsize=(4.2,3.6))
sns.heatmap(confusion_matrix(y_val, vpred), annot=True, fmt='d',
cmap='Blues', cbar=False)
plt.title('XGB Confusion Matrix (Val)'); plt.xlabel('Predicted');
plt.ylabel('Actual')
plt.tight_layout(); plt.show()


prec, rec, _ = precision_recall_curve(y_val, vprob)
ap = average_precision_score(y_val, vprob)
plt.figure(figsize=(4.8,3.6))
plt.plot(rec, prec, label=f'Val PR (AP={ap:.3f})')
plt.xlabel('Recall'); plt.ylabel('Precision'); plt.title('Precision–
Recall Curve (Val)'); plt.legend()
plt.grid(True, ls='--', alpha=.4); plt.tight_layout(); plt.show()
```

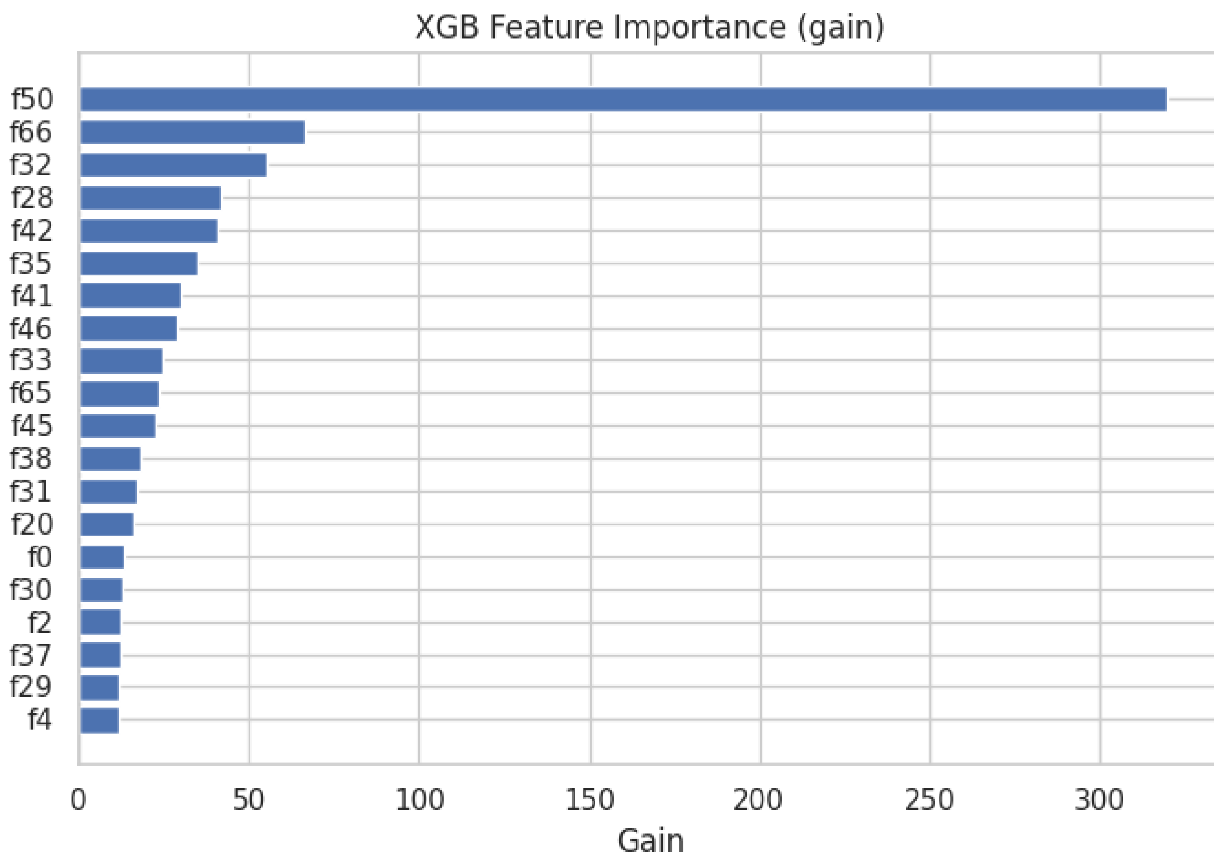
```
imp = booster.get_score(importance_type='gain')
imp_df = pd.DataFrame({'feature': list(imp.keys()), 'gain':
list(imp.values())}).sort_values('gain', ascending=False).head(20)
plt.figure(figsize=(7,5))
plt.barh(imp_df['feature'][::-1], imp_df['gain'][::-1])
plt.title('XGB Feature Importance (gain)'); plt.xlabel('Gain');
plt.tight_layout(); plt.show()


hist = booster.attributes()

try:
    results = booster.evals_result()
    train_loss = results['train']['logloss']
    val_loss   = results['val']['logloss']
except:

    print("Warning: Could not retrieve evaluation history")
    train_loss = []
    val_loss = []

if train_loss and val_loss:
    rounds = np.arange(1, len(train_loss)+1)
    plt.figure(figsize=(6.5,4.2))
    plt.plot(rounds, train_loss, label='Train logloss')
    plt.plot(rounds, val_loss,   label='Val logloss')
    plt.axvline(best_iteration + 1, color='k', ls='--', alpha=.6,
label='best_iter')
    plt.xlabel('Boosting rounds'); plt.ylabel('Logloss');
plt.title('Training vs Validation Loss')
    plt.legend(); plt.grid(True, ls='--', alpha=.4);
plt.tight_layout(); plt.show()


def run_cfg(lr, md, subs=0.8):
    p = params.copy(); p['eta'] = lr; p['max_depth'] = md;
p['subsample'] = subs
    bst = xgb.train(p, dtrain, num_boost_round=5000, evals=evals,
early_stopping_rounds=50, verbose_eval=False)
    best_iter_bst = bst.best_iteration
    vpr = bst.predict(dval,  iteration_range=(0, best_iter_bst + 1))
    tpr = bst.predict(dtest, iteration_range=(0, best_iter_bst + 1))
    return {
        'lr': lr, 'max_depth': md,
        'best_iter': best_iter_bst,
        'val_aucpr': average_precision_score(y_val, vpr),
        'test_aucpr': average_precision_score(y_test, tpr),
        'val_auc': roc_auc_score(y_val, vpr),
```

```
        'test_auc': roc_auc_score(y_test, tpr)
    }

grid = [run_cfg(lr, md) for lr in [0.01, 0.1, 0.3] for md in
[2,3,4,6]]
df = pd.DataFrame(grid).sort_values(['val_aucpr','val_auc'],
ascending=False)
print(df[['lr','max_depth','best_iter','val_aucpr','test_aucpr','val_a
uc','test_auc']])
plt.figure(figsize=(6.8,4))
for lr in sorted(df['lr'].unique()):
    s = df[df['lr']==lr]
    plt.plot(s['max_depth'], s['val_aucpr'], marker='o',
label=f'lr={lr}')
plt.xlabel('max_depth'); plt.ylabel('Val AUCPR'); plt.title('AUCPR vs
depth across learning rates')
plt.legend(); plt.grid(True, ls='--', alpha=.4); plt.tight_layout();
plt.show()
```

```
[VAL] ACC=0.896 PREC=0.670 REC=0.225 F1=0.337 AUCPR=0.468 AUROC=0.806
[TEST] ACC=0.897 PREC=0.671 REC=0.235 F1=0.348 AUCPR=0.457 AUROC=0.793
```

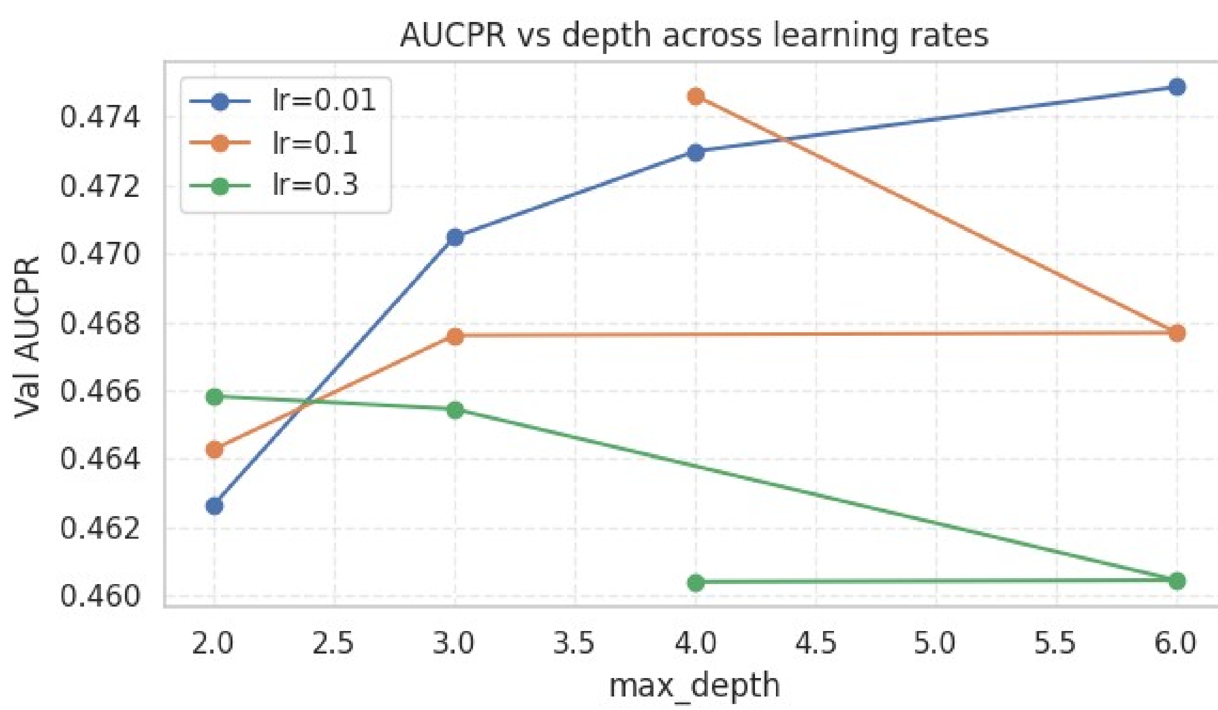







```
Warning: Could not retrieve evaluation history
      lr  max_depth  best_iter  val_aucpr  test_aucpr   val_auc
test_auc
```

```
3   0.01          6        782   0.474880    0.470449  0.809704
0.797175
6   0.10          4        100   0.474630    0.462314  0.807448
0.794853
2   0.01          4       1240   0.472998    0.462750  0.808986
0.795421
1   0.01          3       1632   0.470488    0.458263  0.807548
0.793600
7   0.10          6        106   0.467680    0.467919  0.804566
0.794050
5   0.10          3        169   0.467607    0.456560  0.806300
0.792572
8   0.30          2         77   0.465824    0.444199  0.805365
0.787670
9   0.30          3         59   0.465448    0.450037  0.805180
0.794288
4   0.10          2        435   0.464264    0.458827  0.805045
0.794936
0   0.01          2       2614   0.462635    0.453154  0.804790
0.791905
11  0.30          6         20   0.460431    0.441406  0.798478
0.784573
10  0.30          4         41   0.460388    0.442203  0.804434
0.787533
```



```
import time, numpy as np, pandas as pd, matplotlib.pyplot as plt,
seaborn as sns
import warnings
import os
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, precision_recall_curve, average_precision_score,
roc_auc_score
)
import xgboost as xgb
import lightgbm as lgb


warnings.filterwarnings('ignore')
os.environ['PYTHONWARNINGS'] = 'ignore'

sns.set(style="whitegrid")


Xtr = pre_safe.fit_transform(X_safe_train, y_train)
Xva = pre_safe.transform(X_safe_val)
Xte = pre_safe.transform(X_safe_test)


def pr_metrics(y, prob, thr=0.5):
    pred = (prob >= thr).astype(int)
    return {
        'acc': accuracy_score(y, pred),
        'prec': precision_score(y, pred, zero_division=0),
        'rec': recall_score(y, pred, zero_division=0),
        'f1': f1_score(y, pred, zero_division=0),
        'aucpr': average_precision_score(y, prob),
        'auc': roc_auc_score(y, prob),
        'pred': pred
    }

def plot_confusion(y_true, y_pred, title):
    plt.figure(figsize=(4.2,3.6))
    sns.heatmap(confusion_matrix(y_true, y_pred), annot=True, fmt='d',
cmap='Blues', cbar=False)
    plt.title(title); plt.xlabel('Predicted'); plt.ylabel('Actual');
plt.tight_layout(); plt.show()

def plot_pr_curve(y_true, prob, title):
    prec, rec, _ = precision_recall_curve(y_true, prob)
    ap = average_precision_score(y_true, prob)
    plt.figure(figsize=(4.8,3.6))
    plt.plot(rec, prec, label=f'AP={ap:.3f}')
    plt.xlabel('Recall'); plt.ylabel('Precision'); plt.title(title);
plt.legend()
```

```
    plt.grid(True, ls='--', alpha=.4); plt.tight_layout(); plt.show()


dtrain = xgb.DMatrix(Xtr, label=y_train)
dval   = xgb.DMatrix(Xva, label=y_val)
dtest  = xgb.DMatrix(Xte, label=y_test)

xgb_params = {
    'objective': 'binary:logistic',
    'eval_metric': ['logloss','aucpr','auc'],
    'eta': 0.1,
    'max_depth': 3,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'lambda': 1.0,
    'tree_method': 'hist'
}

t0 = time.time()
xgb_booster = xgb.train(
    params=xgb_params,
    dtrain=dtrain,
    num_boost_round=5000,
    evals=[(dtrain,'train'),(dval,'val')],
    early_stopping_rounds=50,
    verbose_eval=False
)
xgb_time = time.time() - t0


best_round = getattr(xgb_booster, 'best_iteration', None)
iter_end = (best_round + 1) if best_round is not None else 0
def xgb_predict(bst, dmat, iter_end):
    try:
        return bst.predict(dmat, iteration_range=(0, iter_end)) if
iter_end else bst.predict(dmat)
    except TypeError:
        bntl = getattr(bst, 'best_ntree_limit', None)
        return bst.predict(dmat, ntree_limit=bntl) if bntl is not None
else bst.predict(dmat)

xgb_vprob = xgb_predict(xgb_booster, dval, iter_end)
xgb_tprob = xgb_predict(xgb_booster, dtest, iter_end)

xgb_val = pr_metrics(y_val, xgb_vprob)
xgb_tst = pr_metrics(y_test, xgb_tprob)

print(f"[XGB] Fit time: {xgb_time:.2f}s  |  VAL
AUCPR={xgb_val['aucpr']:.3f}, AUROC={xgb_val['auc']:.3f},
ACC={xgb_val['acc']:.3f}  |  TEST AUCPR={xgb_tst['aucpr']:.3f},
```

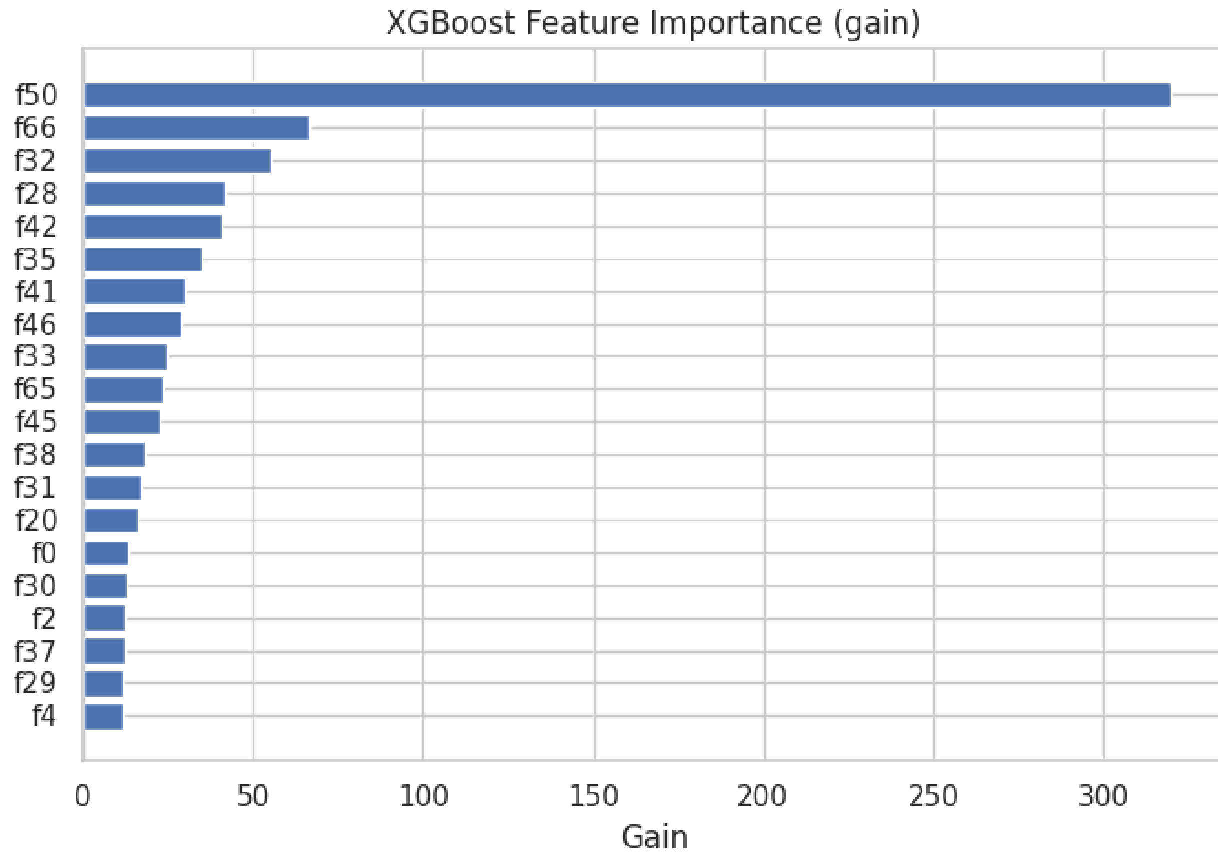
```
AUROC={xgb_tst['auc']:.3f}, ACC={xgb_tst['acc']:.3f}")  # [web:90]
[web:139][web:136]


try:
    xgb_hist = xgb_booster.evals_result()
    train_loss = xgb_hist['train']['logloss']
    val_loss = xgb_hist['val']['logloss']

    plt.figure(figsize=(6.5,4.2))
    plt.plot(np.arange(1, len(train_loss)+1), train_loss, label='Train
logloss')
    plt.plot(np.arange(1, len(val_loss)+1), val_loss,   label='Val
logloss')
    if best_round is not None:
        plt.axvline(best_round+1, color='k', ls='--', alpha=.6,
label='best_iter')
    plt.xlabel('Boosting rounds'); plt.ylabel('Logloss');
plt.title('XGBoost: Train vs Val Loss')
    plt.legend(); plt.grid(True, ls='--', alpha=.4);
plt.tight_layout(); plt.show()
except (AttributeError, KeyError):
    pass


xgb_imp = xgb_booster.get_score(importance_type='gain')
if xgb_imp:
    xgb_imp_df = pd.DataFrame({'feature': list(xgb_imp.keys()),
'gain': list(xgb_imp.values())}).sort_values('gain',
ascending=False).head(20)
    plt.figure(figsize=(7,5)); plt.barh(xgb_imp_df['feature'][::-1],
xgb_imp_df['gain'][::-1])
    plt.title('XGBoost Feature Importance (gain)');
plt.xlabel('Gain'); plt.tight_layout(); plt.show()


plot_confusion(y_val, xgb_val['pred'], 'XGBoost Confusion Matrix
(Val)')
plot_pr_curve(y_val, xgb_vprob, 'XGBoost Precision–Recall (Val)')


lgbm = lgb.LGBMClassifier(
    objective='binary',
    learning_rate=0.1,
    n_estimators=5000,
    max_depth=-1,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    random_state=42,
```

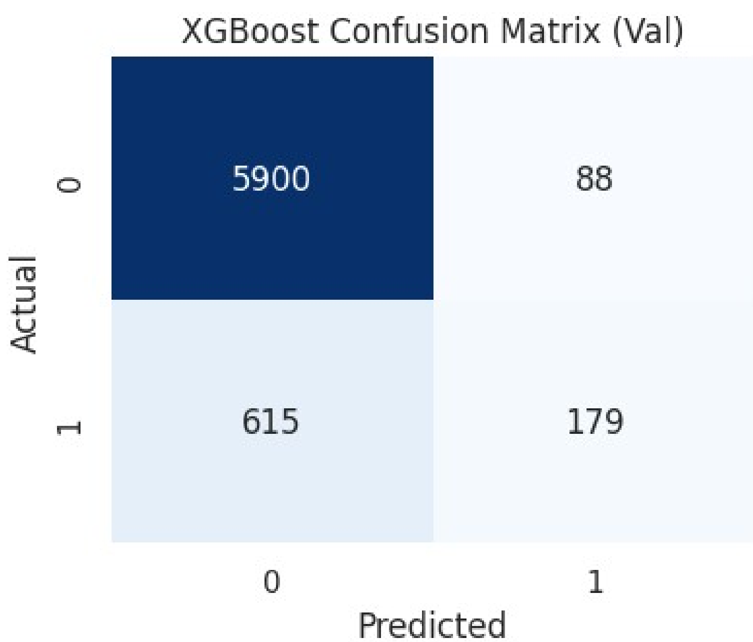
```
    verbosity=-1,
    force_row_wise=True
)

callbacks = [
    lgb.early_stopping(stopping_rounds=50, verbose=False),
    lgb.log_evaluation(period=0)
]

t0 = time.time()
lgbm.fit(
    Xtr, y_train,
    eval_set=[(Xtr, y_train), (Xva, y_val)],
    callbacks=callbacks
)
lgb_time = time.time() - t0
best_iter_lgb = getattr(lgbm, 'best_iteration_', None)


def lgb_predict(model, X):
    try:
        return model.predict_proba(X, num_iteration=best_iter_lgb)
[:,1] if best_iter_lgb is not None else model.predict_proba(X)[:,1]
    except TypeError:
        return model.predict_proba(X)[:,1]

lgb_vprob = lgb_predict(lgbm, Xva)
lgb_tprob = lgb_predict(lgbm, Xte)

lgb_val = pr_metrics(y_val, lgb_vprob)
lgb_tst = pr_metrics(y_test, lgb_tprob)

print(f"[LGBM] Fit time: {lgb_time:.2f}s  |  VAL
AUCPR={lgb_val['aucpr']:.3f}, AUROC={lgb_val['auc']:.3f},
ACC={lgb_val['acc']:.3f}  |  TEST AUCPR={lgb_tst['aucpr']:.3f},
AUROC={lgb_tst['auc']:.3f}, ACC={lgb_tst['acc']:.3f}")


try:
    lgb_hist = lgbm.evals_result_
    lgb_tr_loss = lgb_hist['training']['binary_logloss'] if
'binary_logloss' in lgb_hist.get('training', {}) else
lgb_hist['training'].get('multi_logloss', [])
    lgb_va_loss = lgb_hist['valid_1']['binary_logloss'] if
'binary_logloss' in lgb_hist.get('valid_1', {}) else
lgb_hist['valid_1'].get('multi_logloss', [])

    plt.figure(figsize=(6.5,4.2))
    plt.plot(np.arange(1, len(lgb_tr_loss)+1), lgb_tr_loss,
label='Train logloss')
```

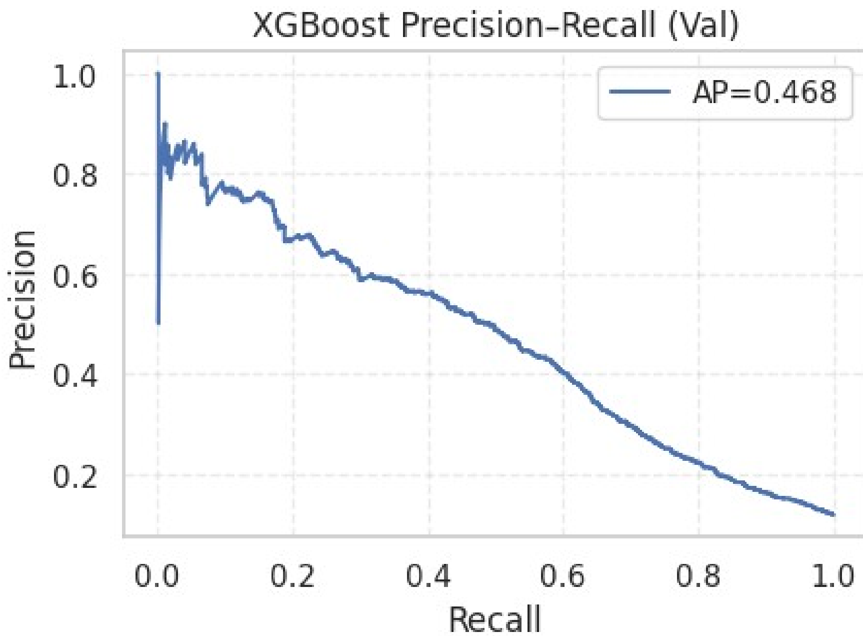
```
    plt.plot(np.arange(1, len(lgb_va_loss)+1), lgb_va_loss, label='Val
logloss')
    if best_iter_lgb is not None:
        plt.axvline(best_iter_lgb, color='k', ls='--', alpha=.6,
label='best_iter')
    plt.xlabel('Boosting rounds'); plt.ylabel('Logloss');
plt.title('LightGBM: Train vs Val Loss')
    plt.legend(); plt.grid(True, ls='--', alpha=.4);
plt.tight_layout(); plt.show()
except (AttributeError, KeyError):
    pass


try:
    lgb_imp = lgbm.booster_.feature_importance(importance_type='gain')
    lgb_names = lgbm.booster_.feature_name()
    imp_df = pd.DataFrame({'feature': lgb_names, 'gain':
lgb_imp}).sort_values('gain', ascending=False).head(20)
    plt.figure(figsize=(7,5)); plt.barh(imp_df['feature'][::-1],
imp_df['gain'][::-1])
    plt.title('LightGBM Feature Importance (gain)');
plt.xlabel('Gain'); plt.tight_layout(); plt.show()
except Exception:
    pass


plot_confusion(y_val, lgb_val['pred'], 'LightGBM Confusion Matrix
(Val)')
plot_pr_curve(y_val, lgb_vprob, 'LightGBM Precision–Recall (Val)')


def xgb_sweep(lr, md):
    p = xgb_params.copy(); p['eta'] = lr; p['max_depth'] = md
    bst = xgb.train(p, dtrain, num_boost_round=5000,
evals=[(dtrain,'train'),(dval,'val')], early_stopping_rounds=50,
verbose_eval=False)
    br = getattr(bst,'best_iteration', None)
    ie = br+1 if br is not None else 0
    vpr = xgb_predict(bst, dval, ie); tpr = xgb_predict(bst, dtest,
ie)
    return {'lib':'xgb','lr':lr,'depth':md,'best_iter':br,
'val_aucpr':average_precision_score(y_val, vpr),
'test_aucpr':average_precision_score(y_test, tpr)}

def lgb_sweep(lr, md):
    mdl = lgb.LGBMClassifier(
        objective='binary',
        learning_rate=lr,
        n_estimators=5000,
        max_depth=md,
```

```
        subsample=0.8,
        colsample_bytree=0.8,
        reg_lambda=1.0,
        random_state=42,
        verbosity=-1,
        force_row_wise=True
    )
    mdl.fit(Xtr, y_train, eval_set=[(Xtr, y_train),(Xva, y_val)],
callbacks=[lgb.early_stopping(stopping_rounds=50, verbose=False),
lgb.log_evaluation(period=0)])
    bi = getattr(mdl,'best_iteration_', None)
    try:
        vpr = mdl.predict_proba(Xva, num_iteration=bi)[:,1] if bi is
not None else mdl.predict_proba(Xva)[:,1]
        tpr = mdl.predict_proba(Xte, num_iteration=bi)[:,1] if bi is
not None else mdl.predict_proba(Xte)[:,1]
    except TypeError:
        vpr = mdl.predict_proba(Xva)[:,1]
        tpr = mdl.predict_proba(Xte)[:,1]
    return {'lib':'lgb','lr':lr,'depth':md,'best_iter':bi,
'val_aucpr':average_precision_score(y_val, vpr),
'test_aucpr':average_precision_score(y_test, tpr)}

grid = []
for lr in [0.01, 0.1, 0.3]:
    for md in [2, 3, 4, 6]:
        grid.append(xgb_sweep(lr, md))
        grid.append(lgb_sweep(lr, md))
df = pd.DataFrame(grid).sort_values(['lib','val_aucpr'],
ascending=[True, False])
print("\nHyperparameter Grid Search Results (Top configurations by
AUCPR):")
print(df.head(12))

plt.figure(figsize=(7.2,4))
for lib in ['xgb','lgb']:
    subset = df[df['lib']==lib]
    for lr in sorted(subset['lr'].unique()):
        s = subset[subset['lr']==lr]
        plt.plot(s['depth'], s['val_aucpr'], marker='o',
label=f'{lib}-lr={lr}')
plt.xlabel('max_depth'); plt.ylabel('Val AUCPR'); plt.title('AUCPR vs
depth across learning rates (XGB vs LGB)')
plt.legend(ncol=2); plt.grid(True, ls='--', alpha=.4);
plt.tight_layout(); plt.show()
```

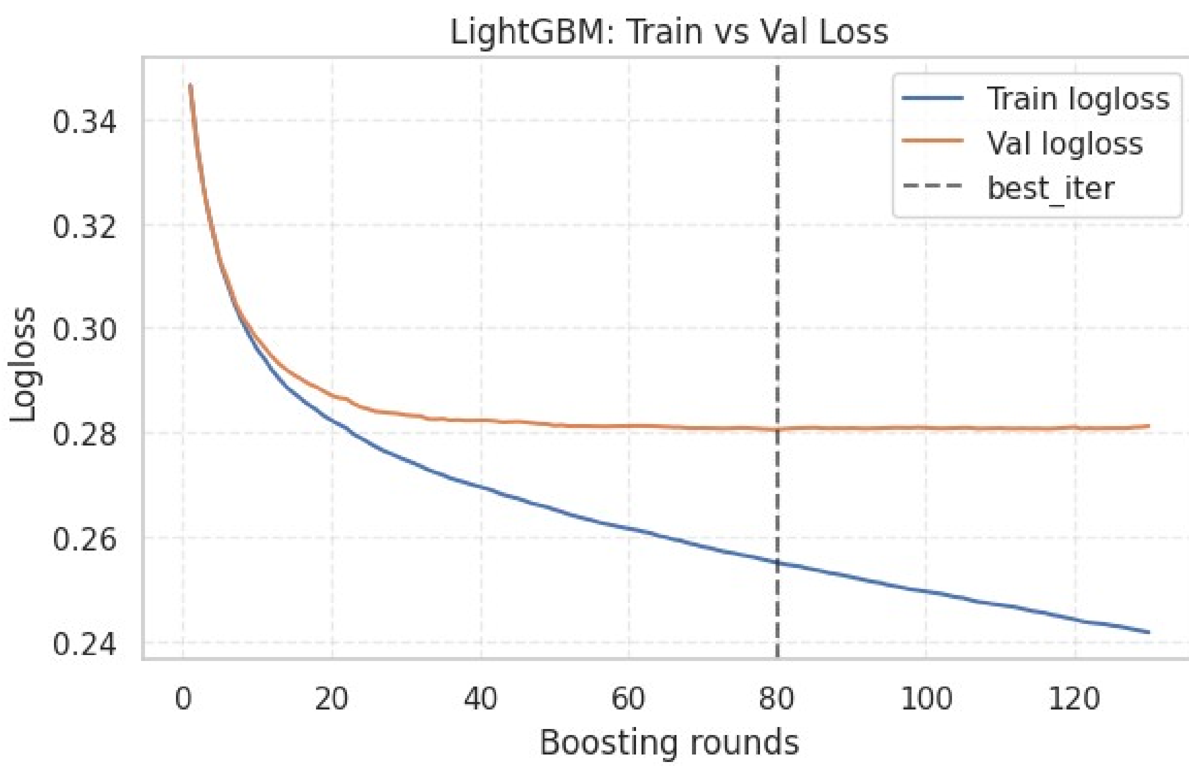
```
[XGB] Fit time: 2.87s  |  VAL AUCPR=0.468, AUROC=0.806, ACC=0.896  |
TEST AUCPR=0.457, AUROC=0.793, ACC=0.897
```

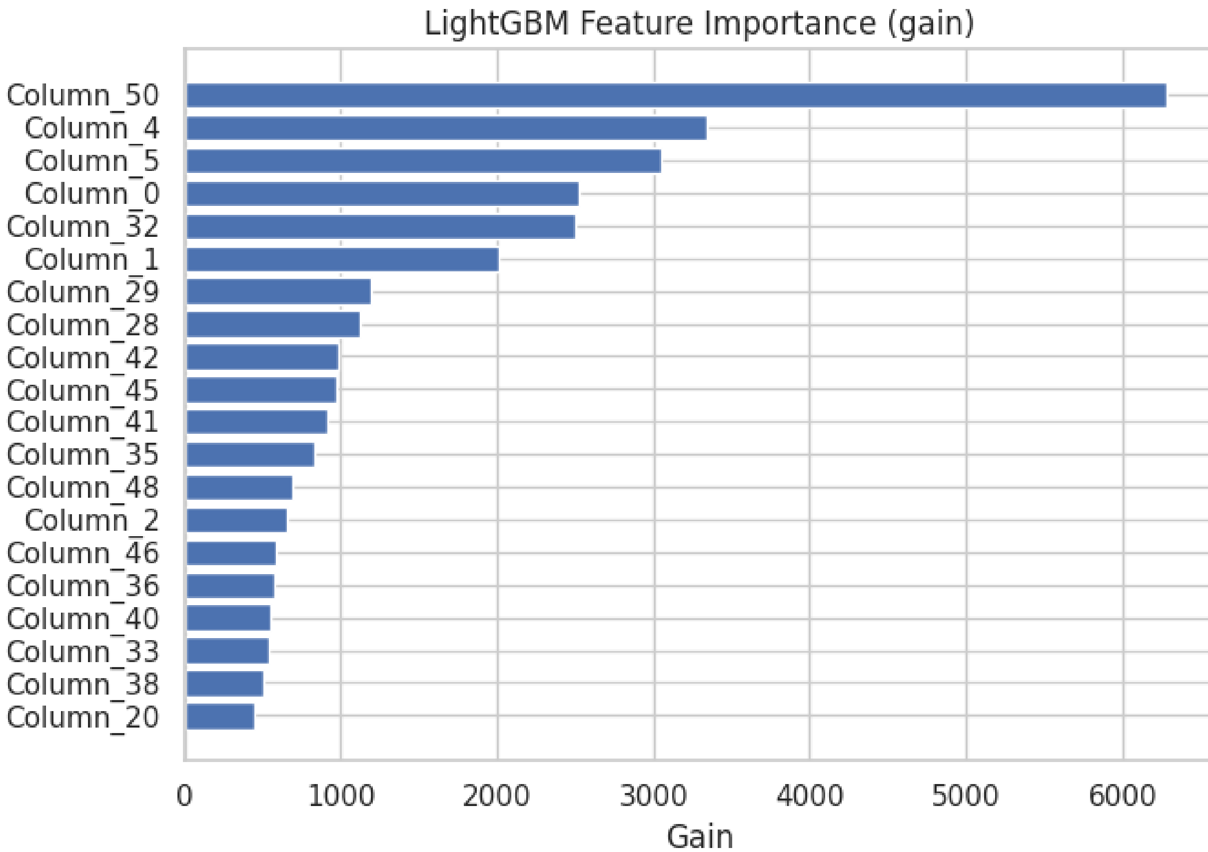

XGBoost Feature Importance (gain)
f50
f66
f32
f28
f42
f35
f41
f46
f33
f65
f45
f38
f31
f20
f0
f30
f2
f37
f29
f4
0
50
100
150
200
250
300
Gain

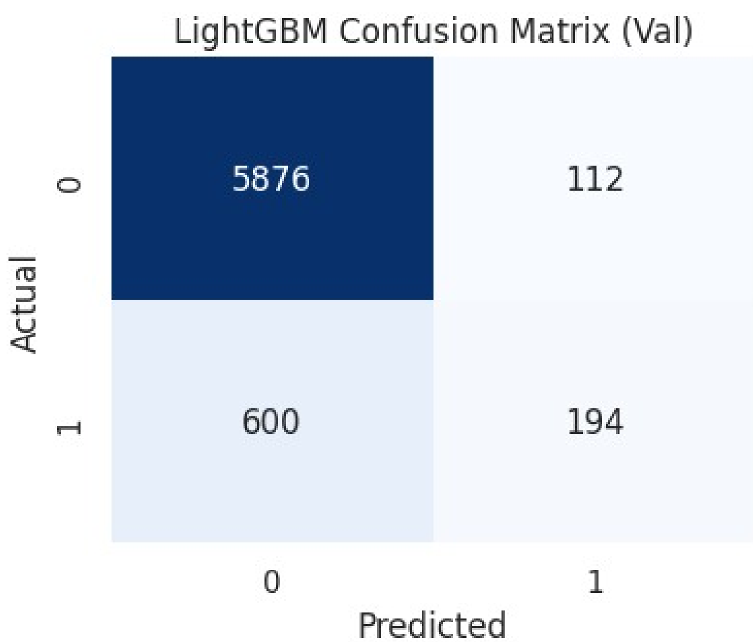

XGBoost Confusion Matrix (Val)
5900
88
615
179
Actual
0
1
Predicted

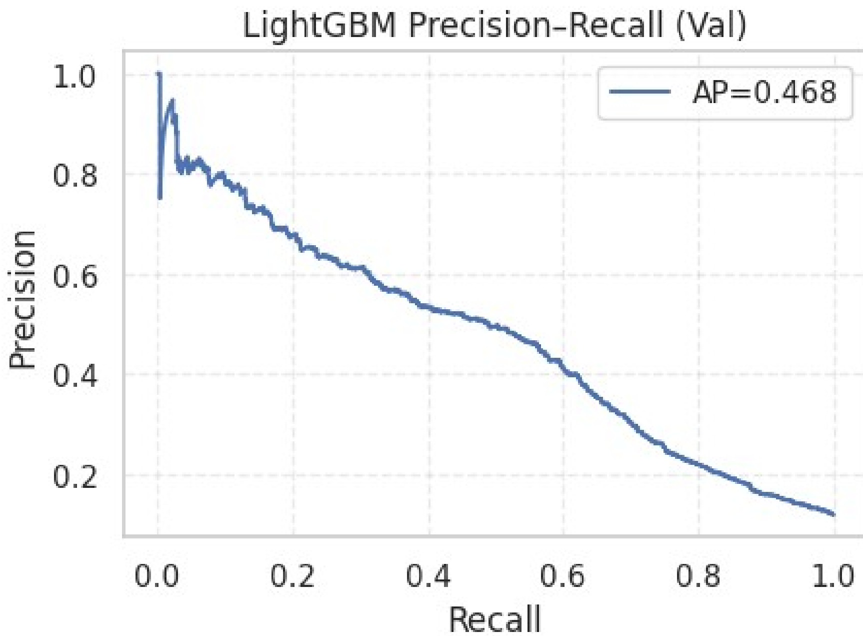



```
[LGBM] Fit time: 0.86s  |  VAL AUCPR=0.468, AUROC=0.806, ACC=0.895  |
TEST AUCPR=0.470, AUROC=0.798, ACC=0.898
```

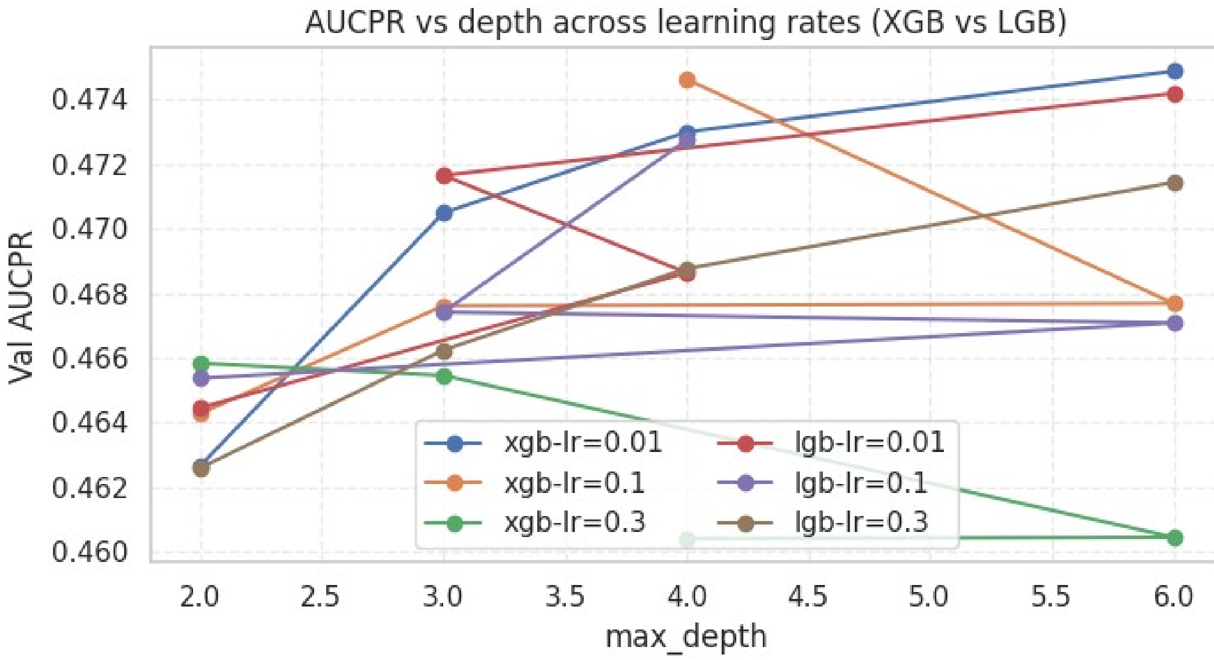



LightGBM Feature Importance (gain)
Column_50
Column_4
Column_5
Column_0
Column_32
Column_1
Column_29
Column_28
Column_42
Column_45
Column_41
Column_35
Column_48
Column_2
Column_46
Column_36
Column_40
Column_33
Column_38
Column_20
0
1000
2000
3000
4000
5000
6000
Gain


LightGBM Confusion Matrix (Val)
Actual
0
1
5876
112
600
194
0
1
Predicted




```
Hyperparameter Grid Search Results (Top configurations by AUCPR):
    lib    lr  depth  best_iter  val_aucpr  test_aucpr
7   lgb  0.01      6        986   0.474191    0.471731
13  lgb  0.10      4        172   0.472748    0.466267
3   lgb  0.01      3       2360   0.471650    0.463317
23  lgb  0.30      6         27   0.471430    0.455875
21  lgb  0.30      4         55   0.468761    0.452501
5   lgb  0.01      4       1631   0.468631    0.467714
11  lgb  0.10      3        317   0.467412    0.462164
15  lgb  0.10      6         76   0.467078    0.464983
19  lgb  0.30      3         64   0.466232    0.456841
9   lgb  0.10      2        554   0.465360    0.458534
1   lgb  0.01      2       4056   0.464440    0.459129
17  lgb  0.30      2        181   0.462576    0.454157
```



AUCPR vs depth across learning rates (XGB vs LGB)
Val AUCPR
max_depth
0.474
0.472
0.470
0.468
0.466
0.464
0.462
0.460
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
xgb-lr=0.01
xgb-lr=0.1
xgb-lr=0.3
lgb-lr=0.01
lgb-lr=0.1
lgb-lr=0.3