Comparing Gradient Boosting Methods: XGBoost vs LightGBM vs CatBoost

RSU — Machine Learning / AI Course — Nataliia Kinash

Objective

In this practice notebook you will:

1. Load the **Bank Marketing** dataset (UCI) — predict whether a client subscribes to a term deposit
2. Prepare the data for all three gradient boosting frameworks
3. Train **XGBoost**, **LightGBM**, and **CatBoost** classifiers
4. Compare their **accuracy**, **ROC AUC**, **training time**, and **feature importance**
5. Visualise the results

About the Dataset

The Bank Marketing dataset contains data from direct marketing campaigns (phone calls) of a Portuguese bank. The task is binary classification: predict if a client will subscribe to a term deposit (`y`: yes/no).

- **~41,000 samples**, 20 features
- Mix of **numerical** (age, balance, duration, etc.) and **categorical** (job, marital, education, contact, month, etc.) features
- **Imbalanced** classes (~11% positive)

Instructions

Look for `# TODO:` comments — these mark the gaps you need to fill in. Each gap has a hint to guide you.

Step 0: Install and Import Libraries

In [1]:
```
# pip install xgboost lightgbm catboost scikit-learn pandas matplotlib seabo
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

```
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import (
    accuracy_score, roc_auc_score, classification_report,
    confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay
)

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

print('All libraries imported successfully!')
```

```
All libraries imported successfully!
```

Step 1: Load and Explore the Dataset

In [3]:
```
from pathlib import Path

df = pd.read_csv(Path("bank.csv"))
if "deposit" in df.columns and "y" not in df.columns:
    df = df.rename(columns={"deposit": "y"})
```

In [4]:
```
print('Column types:')
print(df.dtypes)

numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = df.select_dtypes(include=['object']).columns.drop('y').to

print(f'\nNumerical columns: {numerical_cols}')
print(f'Categorical columns: {categorical_cols}')
```

```
Column types:
age          int64
job            str
marital        str
education      str
default        str
balance      int64
housing        str
loan           str
contact        str
day          int64
month          str
duration     int64
campaign     int64
pdays        int64
previous     int64
poutcome       str
y              str
dtype: object

Numerical columns: ['age', 'balance', 'day', 'duration', 'campaign', 'pday
s', 'previous']
Categorical columns: ['job', 'marital', 'education', 'default', 'housing',
'loan', 'contact', 'month', 'poutcome']
```

In [5]:
```
print('Target distribution:')
print(df['y'].value_counts())
print(f'\nPositive class ratio: {(df["y"] == "yes").mean():.1%}')
```

```
Target distribution:
y
no     5873
yes    5289
Name: count, dtype: int64

Positive class ratio: 47.4%
```

In [6]:
```
print('Missing values per column:')
print(df.isnull().sum())
```

```
Missing values per column:
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

Step 2: Data Preparation

We need **three versions** of the data because each method handles categorical features differently:

Method	Categorical Handling
XGBoost	Requires label encoding (or one-hot encoding)
LightGBM	Accepts integer-encoded categoricals with `categorical_feature` parameter
CatBoost	Handles raw categorical features natively

In [7]:
```
# Identify feature types
categorical_cols = df.select_dtypes(include=['object']).columns.drop('y').to
numerical_cols = df.select_dtypes(include=['number']).columns.tolist()

print(f'Categorical features ({len(categorical_cols)}): {categorical_cols}')
print(f'Numerical features ({len(numerical_cols)}): {numerical_cols}')
```

```
Categorical features (9): ['job', 'marital', 'education', 'default', 'housin
g', 'loan', 'contact', 'month', 'poutcome']
Numerical features (7): ['age', 'balance', 'day', 'duration', 'campaign', 'p
days', 'previous']
```

In [8]:
```
df['target'] = (df['y'] == 'yes').astype(int)

X = df.drop(columns=['y', 'target'])
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
print(f'Training set: {X_train.shape[0]} samples')
print(f'Test set:     {X_test.shape[0]} samples')
print(f'Positive class in train: {y_train.mean():.1%}')
print(f'Positive class in test:  {y_test.mean():.1%}')
```

```
Training set: 8929 samples
Test set:     2233 samples
Positive class in train: 47.4%
Positive class in test:  47.4%
```

In [9]:
```
X_train_xgb = X_train.copy()
X_test_xgb = X_test.copy()

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    X_train_xgb[col] = le.fit_transform(X_train_xgb[col])
    X_test_xgb[col] = le.transform(X_test_xgb[col])
    label_encoders[col] = le

print('XGBoost data ready (label encoded)')
X_train_xgb.head()
```

```
XGBoost data ready (label encoded)
```

Out[9]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
8644	46	0	1	1	0	526	0	1	0	31	
2632	46	0	1	2	0	2552	0	0	0	3	
3056	25	9	2	1	0	139	0	1	0	11	
1080	43	9	1	1	0	3288	1	0	0	21	
4006	37	1	2	3	0	217	0	0	0	23	

In [10]:
```
X_train_lgb = X_train_xgb.copy()
X_test_lgb = X_test_xgb.copy()

for col in categorical_cols:
    X_train_lgb[col] = X_train_lgb[col].astype('category')
    X_test_lgb[col] = X_test_lgb[col].astype('category')

print('LightGBM data ready (category dtype)')
print(X_train_lgb.dtypes)
```

```
LightGBM data ready (category dtype)
age             int64
job          category
marital      category
education    category
default      category
balance         int64
housing      category
loan         category
contact      category
day             int64
month        category
duration        int64
campaign        int64
pdays           int64
previous        int64
poutcome     category
dtype: object
```

In [11]:
```
X_train_cat = X_train.copy()
X_test_cat = X_test.copy()

cat_feature_indices = [X_train_cat.columns.get_loc(col) for col in categoric

print('CatBoost data ready (raw categorical features)')
print(f'Categorical feature indices: {cat_feature_indices}')
X_train_cat.head()
```

```
CatBoost data ready (raw categorical features)
Categorical feature indices: [1, 2, 3, 4, 6, 7, 8, 10, 15]
```

Out[11]:

	age	job	marital	education	default	balance	housing	loan	contact	c
8644	46	admin.	married	secondary	no	526	no	yes	cellular	
2632	46	admin.	married	tertiary	no	2552	no	no	cellular	
3056	25	technician	single	secondary	no	139	no	yes	cellular	
1080	43	technician	married	secondary	no	3288	yes	no	cellular	
4006	37	blue-collar	single	unknown	no	217	no	no	cellular	

Step 3: Train the Three Models

We use comparable hyperparameters across all three to ensure a fair comparison.

In [12]:
```
# Dictionary to store results
results = {}

# Common settings
N_ESTIMATORS = 500
LEARNING_RATE = 0.05
```

```
MAX_DEPTH = 6
RANDOM_STATE = 42
```

3.1 XGBoost

In [13]:
```
xgb_model = XGBClassifier(
    n_estimators=N_ESTIMATORS,
    learning_rate=LEARNING_RATE,
    max_depth=MAX_DEPTH,
    random_state=RANDOM_STATE,
    eval_metric='logloss',
    use_label_encoder=False,
    verbosity=0
)

start_time = time.time()
xgb_model.fit(X_train_xgb, y_train)
xgb_train_time = time.time() - start_time

xgb_pred = xgb_model.predict(X_test_xgb)
xgb_pred_proba = xgb_model.predict_proba(X_test_xgb)[:, 1]

results['XGBoost'] = {
    'accuracy': accuracy_score(y_test, xgb_pred),
    'roc_auc': roc_auc_score(y_test, xgb_pred_proba),
    'train_time': xgb_train_time,
    'model': xgb_model
}

print(f'XGBoost trained in {xgb_train_time:.2f} seconds')
print(f'Accuracy: {results["XGBoost"]["accuracy"]:.4f}')
print(f'ROC AUC:  {results["XGBoost"]["roc_auc"]:.4f}')
```

```
XGBoost trained in 0.89 seconds
Accuracy: 0.8594
ROC AUC:  0.9261
```

3.2 LightGBM

In [14]:
```
lgb_model = LGBMClassifier(
    n_estimators=N_ESTIMATORS,
    learning_rate=LEARNING_RATE,
    max_depth=MAX_DEPTH,
    random_state=RANDOM_STATE,
    verbose=-1
)

start_time = time.time()
lgb_model.fit(X_train_lgb, y_train)
lgb_train_time = time.time() - start_time

lgb_pred = lgb_model.predict(X_test_lgb)
lgb_pred_proba = lgb_model.predict_proba(X_test_lgb)[:, 1]
```

```
results['LightGBM'] = {
    'accuracy': accuracy_score(y_test, lgb_pred),
    'roc_auc': roc_auc_score(y_test, lgb_pred_proba),
    'train_time': lgb_train_time,
    'model': lgb_model
}

print(f'LightGBM trained in {lgb_train_time:.2f} seconds')
print(f'Accuracy: {results["LightGBM"]["accuracy"]:.4f}')
print(f'ROC AUC:  {results["LightGBM"]["roc_auc"]:.4f}')
```

```
LightGBM trained in 2.46 seconds
Accuracy: 0.8630
ROC AUC:  0.9270
```

3.3 CatBoost

In [15]:
```
cat_model = CatBoostClassifier(
    iterations=N_ESTIMATORS,
    learning_rate=LEARNING_RATE,
    depth=MAX_DEPTH,
    random_state=RANDOM_STATE,
    cat_features=cat_feature_indices,
    verbose=0
)

start_time = time.time()
cat_model.fit(X_train_cat, y_train)
cat_train_time = time.time() - start_time

cat_pred = cat_model.predict(X_test_cat).astype(int)
cat_pred_proba = cat_model.predict_proba(X_test_cat)[:, 1]

results['CatBoost'] = {
    'accuracy': accuracy_score(y_test, cat_pred),
    'roc_auc': roc_auc_score(y_test, cat_pred_proba),
    'train_time': cat_train_time,
    'model': cat_model
}

print(f'CatBoost trained in {cat_train_time:.2f} seconds')
print(f'Accuracy: {results["CatBoost"]["accuracy"]:.4f}')
print(f'ROC AUC:  {results["CatBoost"]["roc_auc"]:.4f}')
```

```
CatBoost trained in 1.94 seconds
Accuracy: 0.8661
ROC AUC:  0.9319
```

Step 4: Compare Results

In [16]:
```
comparison_df = pd.DataFrame({
    'Model': ['XGBoost', 'LightGBM', 'CatBoost'],
    'Accuracy': [results[m]['accuracy'] for m in ['XGBoost', 'LightGBM', 'Ca
    'ROC AUC': [results[m]['roc_auc'] for m in ['XGBoost', 'LightGBM', 'CatB
```

```
    'Training Time (s)': [results[m]['train_time'] for m in ['XGBoost', 'Lig
})

comparison_df = comparison_df.set_index('Model')
comparison_df.style.highlight_max(axis=0, subset=['Accuracy', 'ROC AUC'], co
                   .highlight_min(axis=0, subset=['Training Time (s)'], col
                   .format('{:.4f}', subset=['Accuracy', 'ROC AUC']) \
                   .format('{:.2f}', subset=['Training Time (s)'])
```

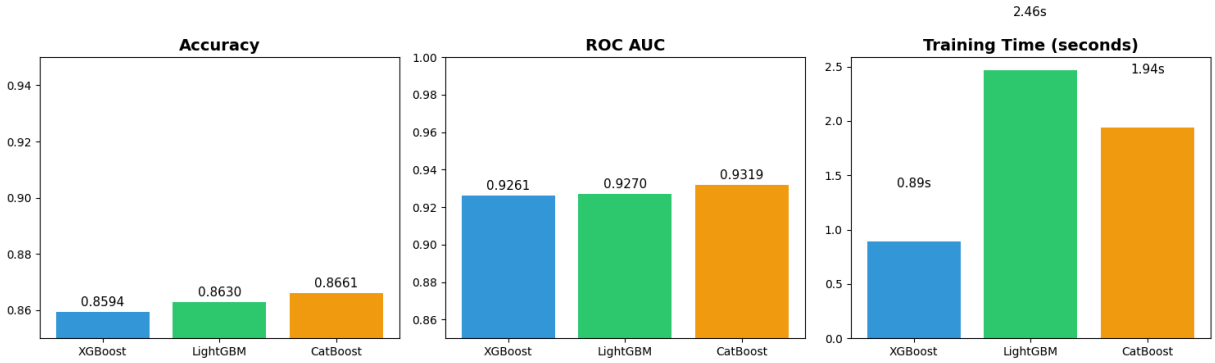
Out[16]:

Model	Accuracy	ROC AUC	Training Time (s)
XGBoost	0.8594	0.9261	0.89
LightGBM	0.8630	0.9270	2.46
CatBoost	0.8661	0.9319	1.94

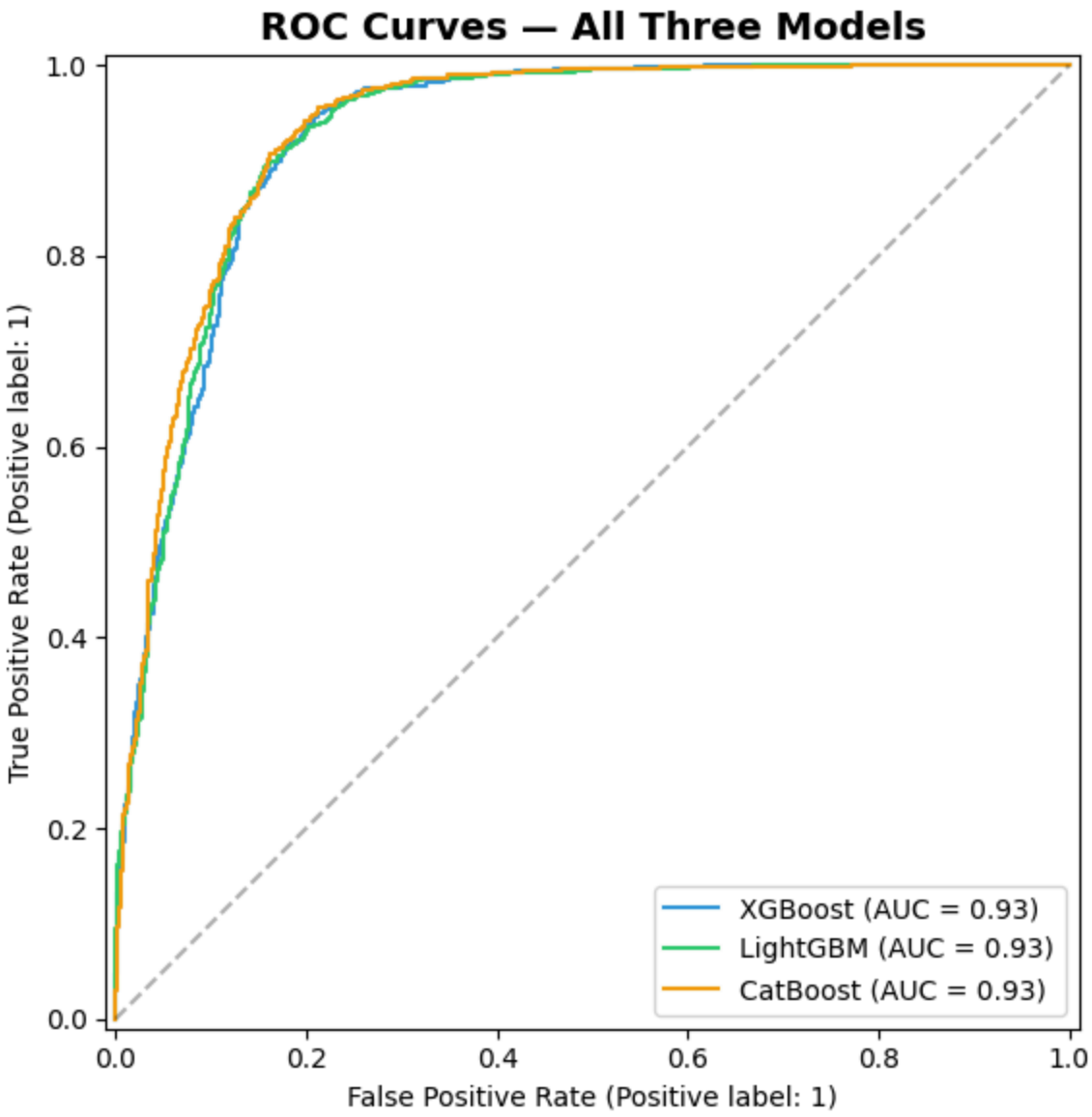
Step 5: Visualise the Comparison

In [17]:

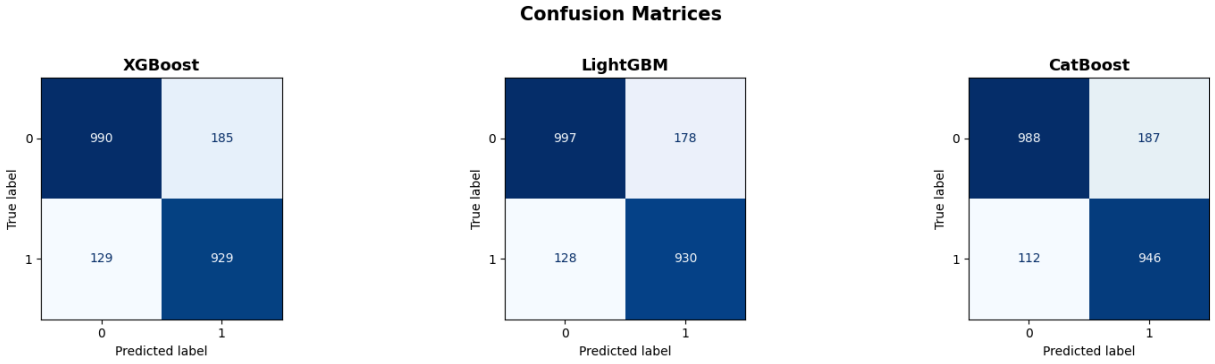
```
# Bar chart comparison (provided — just run this cell)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
colors = ['#3498db', '#2ecc71', '#f39c12']
models = ['XGBoost', 'LightGBM', 'CatBoost']

# Accuracy
axes[0].bar(models, [results[m]['accuracy'] for m in models], color=colors)
axes[0].set_title('Accuracy', fontsize=14, fontweight='bold')
axes[0].set_ylim(0.85, 0.95)
for i, m in enumerate(models):
    axes[0].text(i, results[m]['accuracy'] + 0.002, f'{results[m]["accuracy"
                ha='center', fontsize=11)

# ROC AUC
axes[1].bar(models, [results[m]['roc_auc'] for m in models], color=colors)
axes[1].set_title('ROC AUC', fontsize=14, fontweight='bold')
axes[1].set_ylim(0.85, 1.0)
for i, m in enumerate(models):
    axes[1].text(i, results[m]['roc_auc'] + 0.003, f'{results[m]["roc_auc"]:
                ha='center', fontsize=11)

# Training Time
axes[2].bar(models, [results[m]['train_time'] for m in models], color=colors
axes[2].set_title('Training Time (seconds)', fontsize=14, fontweight='bold')
for i, m in enumerate(models):
    axes[2].text(i, results[m]['train_time'] + 0.5, f'{results[m]["train_tim
                ha='center', fontsize=11)

plt.suptitle('XGBoost vs LightGBM vs CatBoost — Bank Marketing Dataset',
             fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```



In [18]:
```
fig, ax = plt.subplots(figsize=(8, 6))

RocCurveDisplay.from_predictions(y_test, xgb_pred_proba, name='XGBoost', ax=
RocCurveDisplay.from_predictions(y_test, lgb_pred_proba, name='LightGBM', ax
RocCurveDisplay.from_predictions(y_test, cat_pred_proba, name='CatBoost', ax

ax.set_title('ROC Curves — All Three Models', fontsize=14, fontweight='bold'
ax.plot([0, 1], [0, 1], 'k--', alpha=0.3)
plt.tight_layout()
plt.show()
```

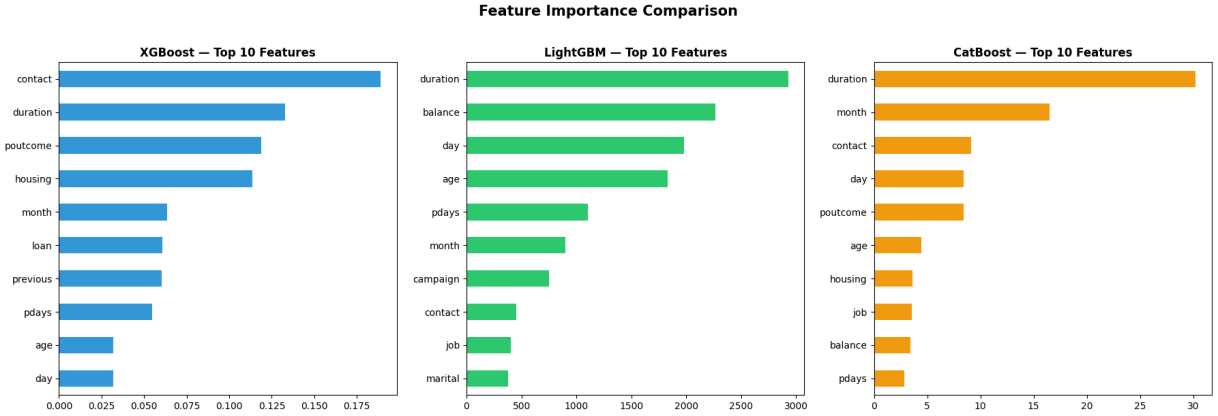


In [19]:
```
fig, axes = plt.subplots(1, 3, figsize=(16, 4))

for ax, (name, pred) in zip(axes, [('XGBoost', xgb_pred), ('LightGBM', lgb_p
    ConfusionMatrixDisplay.from_predictions(y_test, pred, ax=ax, cmap='Blues
    ax.set_title(name, fontsize=13, fontweight='bold')

plt.suptitle('Confusion Matrices', fontsize=15, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```



Step 6: Feature Importance Comparison

In [20]:
```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

xgb_imp = pd.Series(xgb_model.feature_importances_, index=X_train_xgb.columr
xgb_imp.plot(kind='barh', ax=axes[0], color='#3498db')
axes[0].set_title('XGBoost — Top 10 Features', fontweight='bold')
axes[0].invert_yaxis()

lgb_imp = pd.Series(lgb_model.feature_importances_, index=X_train_lgb.columr
lgb_imp.plot(kind='barh', ax=axes[1], color='#2ecc71')
axes[1].set_title('LightGBM — Top 10 Features', fontweight='bold')
axes[1].invert_yaxis()

cat_imp = pd.Series(cat_model.feature_importances_, index=X_train_cat.columr
cat_imp.plot(kind='barh', ax=axes[2], color='#f39c12')
axes[2].set_title('CatBoost — Top 10 Features', fontweight='bold')
axes[2].invert_yaxis()

plt.suptitle('Feature Importance Comparison', fontsize=15, fontweight='bold'
plt.tight_layout()
plt.show()
```



Step 7: Classification Reports

In [21]:
```
for name, pred in [('XGBoost', xgb_pred), ('LightGBM', lgb_pred), ('CatBoost
    print(f'\n{"=" * 50}')
    print(f'{name} Classification Report')
```

```
    print('=' * 50)
    print(classification_report(y_test, pred, target_names=['No Deposit', 'D
```

```
==================================================
XGBoost Classification Report
==================================================
              precision    recall  f1-score   support

  No Deposit       0.88      0.84      0.86      1175
     Deposit       0.83      0.88      0.86      1058

    accuracy                           0.86      2233
   macro avg       0.86      0.86      0.86      2233
weighted avg       0.86      0.86      0.86      2233


==================================================
LightGBM Classification Report
==================================================
              precision    recall  f1-score   support

  No Deposit       0.89      0.85      0.87      1175
     Deposit       0.84      0.88      0.86      1058

    accuracy                           0.86      2233
   macro avg       0.86      0.86      0.86      2233
weighted avg       0.86      0.86      0.86      2233


==================================================
CatBoost Classification Report
==================================================
              precision    recall  f1-score   support

  No Deposit       0.90      0.84      0.87      1175
     Deposit       0.83      0.89      0.86      1058

    accuracy                           0.87      2233
   macro avg       0.87      0.87      0.87      2233
weighted avg       0.87      0.87      0.87      2233
```

Step 8: Summary and Discussion Questions

Key Observations

Fill in based on your results (use the comparison table and plots above):

1. **Fastest model:** Compare **Training Time (s)** — often **LightGBM** is fastest among the three.
2. **Best ROC AUC:** Compare **ROC AUC** column — typically all three are close on this dataset.

3. **Least preprocessing needed: CatBoost** (raw categoricals + `cat_features` only).
4. **Most important features (across all models):** Often **duration**, **balance**, and campaign-related fields rank highly — check the importance bar charts.

Discussion Questions

1. Which model would you deploy in a bank's production system? Why?
2. How did each model handle the categorical features differently? What was the impact on preprocessing effort?
3. Why might the feature importance rankings differ between models?
4. The dataset is imbalanced (~11% positive). How could you improve recall for the minority class?
5. If the dataset had 500 categorical features with high cardinality, which model would you choose?