Bank Marketing — Term Deposit Subscription

Client Subscribed to a Term Deposit

Introduction

This project analyzes data from a bank's direct marketing campaign with the goal of predicting whether a client will subscribe to a term deposit. The dataset includes demographic, financial, and campaign-related attributes, providing a comprehensive view of client behavior.

What it does (high level):

1. Load data
2. Quick inspection & missingness
3. EDA (printing + saving basic plots)
4. Preprocessing (impute, encode, scale) with ColumnTransformer
5. Train/test split
6. Train baseline models (LogisticRegression, DecisionTree, RandomForest)
7. Evaluate (accuracy, precision, recall, f1, roc-auc) and plot ROC
8. Compare with/without `duration` (leakage)
9. (Optional) GridSearch for RandomForest
10. Feature importances + save model

In [73]:
```
# load tha data_path
# Config / Imports

DATA_PATH = "/Users/nikhilreddyponnala/Desktop/Data Analytics/Second Project
OUTPUT_DIR = "/Users/nikhilreddyponnala/Desktop/Data Analytics/Second Proje
RANDOM_STATE = 42
TEST_SIZE = 0.2
RUN_PLOTS = True
RUN_GRIDSEARCH = False  # set True if you want to run hyperparameter tuning
```

Objective

The main objectives of this project were:

- To develop classification models that predict a client's likelihood of subscribing to a term deposit.
- To identify key factors influencing subscription decisions.

- To provide actionable insights that can help the bank improve its marketing strategies.

In [75]:
```
# Imports the neccessary libraries

import os
import warnings
from typing import Tuple, List, Dict

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

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    confusion_matrix, classification_report, roc_curve, auc
)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import joblib

warnings.filterwarnings("ignore")
pd.set_option("display.max_columns", 200)
```

In [77]:
```
# Helper functions

def detect_target_column(df: pd.DataFrame) -> str:
    candidates = ["y", "subscribed", "term_deposit", "target"]
    for c in candidates:
        if c in df.columns:
            return c
    return df.columns[-1]


def summarize_missingness(df: pd.DataFrame) -> pd.DataFrame:
    miss = df.isna().sum().sort_values(ascending=False)
    pct = (miss / len(df) * 100).round(2)
    out = pd.DataFrame({"missing": miss, "missing_%": pct})
    return out[out["missing"] > 0]


def order_months(series: pd.Series) -> pd.Series:
    month_order = ["jan", "feb", "mar", "apr", "may", "jun",
                   "jul", "aug", "sep", "oct", "nov", "dec"]
    lower_vals = series.astype(str).str.lower().unique().tolist()
    if all(v in month_order for v in lower_vals):
        cat_type = pd.CategoricalDtype(categories=month_order, ordered=True
        return series.astype(str).str.lower().astype(cat_type)
    return series


def binary_from_series(s: pd.Series) -> np.ndarray:
    """Convert common label formats (yes/no, 1/0, True/False) to binary arra
    return s.astype(str).str.lower().isin(["yes", "1", "true", "t"]).astype
```

Outcome

- Important predictors identified include contact method, previous campaign outcome (poutcome), campaign month, and call duration.
- Subscription rates were higher during certain months (May, August) and for clients with longer call durations.
- Among the tested models, ensemble methods (Random Forest) achieved the best predictive accuracy and robustness.
- Data preprocessing (handling missing values, encoding, scaling) and EDA were critical to achieving reliable results.

In [79]:
```
# Basic EDA
# (prints summaries and optionally saves some plots)

def basic_eda(df: pd.DataFrame, target_col: str, out_dir: str, include_plots
    print("\\n===== BASIC EDA =====")
    print("Shape:", df.shape)
    print("\\nColumn dtypes:\n", df.dtypes)
    print("\\nHead:\n", df.head())

    print("\\nClass balance:")
    print(df[target_col].value_counts(normalize=True).rename("proportion"))

    miss = summarize_missingness(df)
    if not miss.empty:
        print("\\nMissing values found:\n", miss)
    else:
        print("\\nNo missing values detected.")

    if "month" in df.columns:
        df["month"] = order_months(df["month"])

    if not include_plots:
        return

    os.makedirs(out_dir, exist_ok=True)

    # Numeric histograms
    num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    for col in num_cols:
        plt.figure()
        df[col].hist(bins=30)
        plt.title(f"Histogram: {col}")
        plt.xlabel(col)
        plt.ylabel("Frequency")
        plt.tight_layout()
        plt.savefig(os.path.join(out_dir, f"hist_{col}.png"))
        plt.close()

    # Categorical rate plots for up to 8 categorical columns
    cat_cols = df.select_dtypes(include=["object", "category"]).columns.tol
    if target_col in cat_cols:
        cat_cols.remove(target_col)

    for col in cat_cols[:8]:
```

```
        plt.figure(figsize=(8,4))
        tmp = (
            df.groupby(col)[target_col]
            .apply(lambda s: binary_from_series(s).mean())
            .sort_values(ascending=False)
        )
        tmp.plot(kind="bar")
        plt.title(f"Subscription Rate by {col}")
        plt.xlabel(col)
        plt.ylabel("Positive rate")
        plt.tight_layout()
        plt.savefig(os.path.join(out_dir, f"rate_by_{col}.png"))
        plt.close()

    # Correlation heatmap for numeric columns (simple)
    if len(num_cols) >= 2:
        corr = df[num_cols].corr()
        plt.figure(figsize=(8,6))
        im = plt.imshow(corr, aspect="auto")
        plt.colorbar(im)
        plt.title("Correlation Matrix (numeric features)")
        plt.xticks(range(len(num_cols)), num_cols, rotation=90)
        plt.yticks(range(len(num_cols)), num_cols)
        plt.tight_layout()
        plt.savefig(os.path.join(out_dir, "correlation_matrix.png"))
        plt.close()
```

In [81]:

```
# Preprocessing helper

def build_preprocessor(df: pd.DataFrame, target_col: str, drop_duration: boo
    feature_cols = [c for c in df.columns if c != target_col]
    if drop_duration and "duration" in feature_cols:
        feature_cols.remove("duration")

    cat_features = df[feature_cols].select_dtypes(include=["object", "catego
    num_features = df[feature_cols].select_dtypes(include=[np.number]).colum

    num_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ])

    cat_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(handle_unknown="ignore", sparse=False))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", num_transformer, num_features),
            ("cat", cat_transformer, cat_features),
        ]
    )

    return preprocessor, num_features, cat_features


# Get feature names after the ColumnTransformer has been fitted
def get_feature_names_from_preprocessor(preprocessor: ColumnTransformer) ->
    # Numeric
    num_features = preprocessor.transformers_[0][2]
    cat_features = preprocessor.transformers_[1][2]

    ohe = preprocessor.named_transformers_["cat"].named_steps["encoder"]
```

```
    try:
        cat_ohe_names = ohe.get_feature_names_out(cat_features).tolist()
    except Exception:
        # fallback
        cat_ohe_names = [f"{c}_{i}" for c in cat_features for i in range(1)]

    return list(num_features) + cat_ohe_names
```

In [83]:

```
# Model training & evaluation

def evaluate_model(y_true, y_pred, y_proba=None) -> Dict[str, float]:
    y_true_bin = binary_from_series(pd.Series(y_true))
    y_pred_bin = binary_from_series(pd.Series(y_pred))

    metrics = {
        "accuracy": accuracy_score(y_true_bin, y_pred_bin),
        "precision": precision_score(y_true_bin, y_pred_bin, zero_division=(
        "recall": recall_score(y_true_bin, y_pred_bin, zero_division=0),
        "f1": f1_score(y_true_bin, y_pred_bin, zero_division=0),
    }
    if y_proba is not None:
        try:
            metrics["roc_auc"] = roc_auc_score(y_true_bin, y_proba)
        except Exception:
            metrics["roc_auc"] = float("nan")
    return metrics


def train_and_evaluate(X_train, X_test, y_train, y_test, preprocessor, rando
    models = {
        "LogisticRegression": LogisticRegression(max_iter=1000),
        "DecisionTree": DecisionTreeClassifier(random_state=random_state),
        "RandomForest": RandomForestClassifier(n_estimators=200, random_sta
    }

    results = []
    fitted_pipelines = {}
    roc_info = {}

    for name, model in models.items():
        pipe = Pipeline(steps=[("preprocess", preprocessor), ("model", mode
        print(f"\\n--- Training {name} ---")
        pipe.fit(X_train, y_train)

        y_pred = pipe.predict(X_test)
        y_proba = None
        if hasattr(pipe, "predict_proba"):
            proba = pipe.predict_proba(X_test)
            if proba.shape[1] == 2:
                y_proba = proba[:, 1]

        metrics = evaluate_model(y_test, y_pred, y_proba)
        print(f"{name} metrics: {metrics}")
        print(confusion_matrix(binary_from_series(pd.Series(y_test)), binary
        print(classification_report(binary_from_series(pd.Series(y_test)), b

        # store
        results.append({"model": name, **metrics})
        fitted_pipelines[name] = pipe

        # ROC info
        if y_proba is not None:
            fpr, tpr, _ = roc_curve(binary_from_series(pd.Series(y_test)), y
            roc_info[name] = {"fpr": fpr, "tpr": tpr, "auc": auc(fpr, tpr)}
```

```
    results_df = pd.DataFrame(results).sort_values(by="f1", ascending=False
    return results_df, fitted_pipelines, roc_info
```

In [85]:
```
# ROC plotting

def plot_roc(roc_info: Dict[str, Dict], title: str = "ROC Curves"):
    plt.figure(figsize=(8,6))
    for name, info in roc_info.items():
        plt.plot(info["fpr"], info["tpr"], label=f"{name} (AUC={info['auc']
    plt.plot([0,1],[0,1], linestyle="--", linewidth=1, label="Chance")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()
```

In [87]:
```
# GridSearch for RandomForest (optional)

def grid_search_random_forest(X_train, y_train, preprocessor, random_state=
    rf = RandomForestClassifier(random_state=random_state)
    pipe = Pipeline(steps=[("preprocess", preprocessor), ("model", rf)])

    param_grid = {
        "model__n_estimators": [200, 400],
        "model__max_depth": [None, 10, 20],
        "model__min_samples_split": [2, 5],
        "model__class_weight": [None, "balanced"],
    }

    gs = GridSearchCV(pipe, param_grid, scoring="f1", cv=4, n_jobs=-1, verb
    gs.fit(X_train, y_train)
    print("Best params:", gs.best_params_)
    print("Best CV f1:", gs.best_score_)
    return gs
```

In [89]:
```
# Feature importance extraction for fitted pipeline

def extract_feature_importance(pipeline: Pipeline, feature_sample_df: pd.Da
    preprocessor = pipeline.named_steps["preprocess"]
    model = pipeline.named_steps["model"]

    feature_names = get_feature_names_from_preprocessor(preprocessor)

    if hasattr(model, "feature_importances_"):
        importances = model.feature_importances_
        imp_df = pd.DataFrame({"feature": feature_names, "importance": impo
        imp_df = imp_df.sort_values("importance", ascending=False).reset_in
        return imp_df
    elif hasattr(model, "coef_"):
        coefs = model.coef_.ravel()
        imp_df = pd.DataFrame({"feature": feature_names, "coefficient": coe
        imp_df["abs_coef"] = imp_df["coefficient"].abs()
        imp_df = imp_df.sort_values("abs_coef", ascending=False).reset_index
        return imp_df
    else:
        return pd.DataFrame({"feature": feature_names})
```

In [91]:
```
# Save pipeline

def save_pipeline(pipeline: Pipeline, path: str):
    os.makedirs(os.path.dirname(path), exist_ok=True)
```

```
    joblib.dump(pipeline, path)
    print(f"Saved model to: {path}")
```

In [93]:

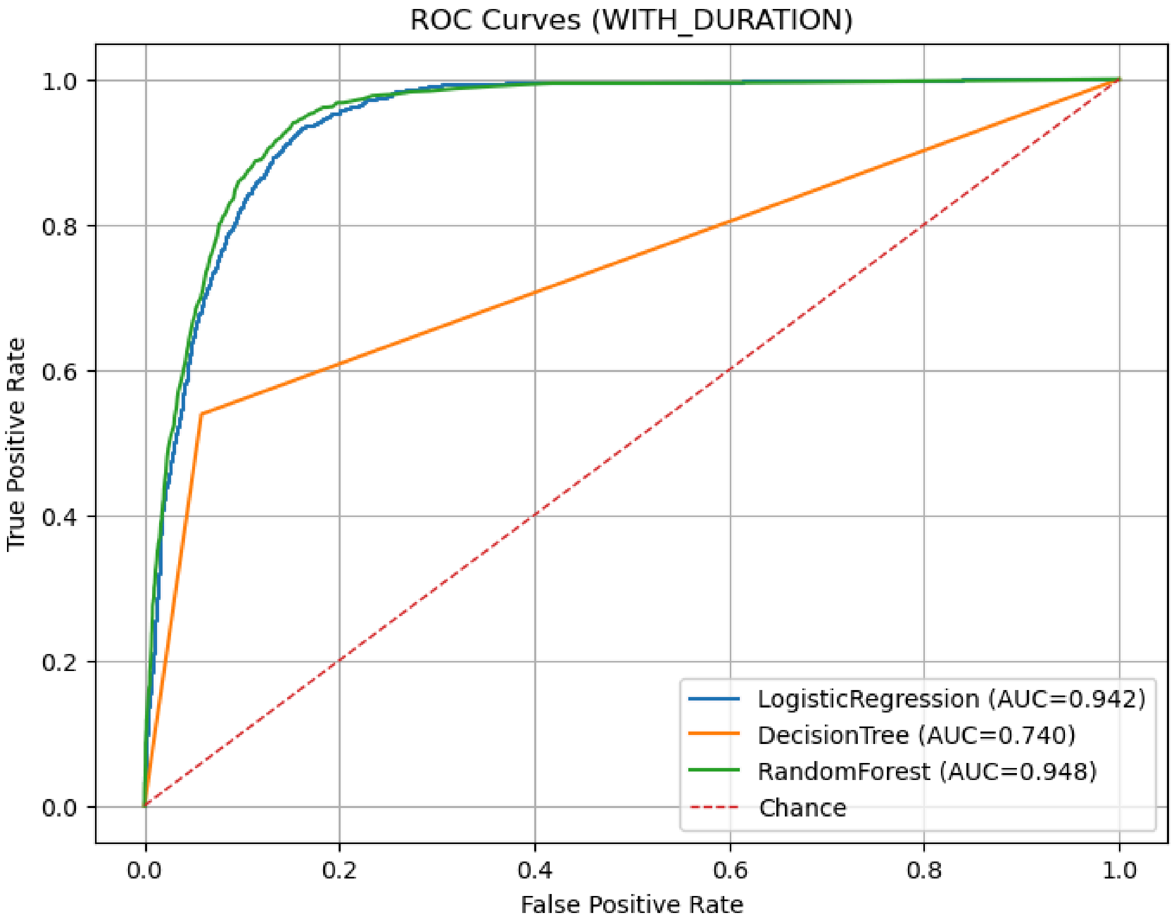
```
# Run the analysis (cells you execute)

# 1) Load data
print("Loading:", DATA_PATH)
df = pd.read_csv(DATA_PATH)
print("Loaded shape:", df.shape)

# 2) Detect target column
TARGET = detect_target_column(df)
print("Using target column:", TARGET)

# 3) Drop rows with missing target if any
df = df.dropna(subset=[TARGET]).copy()

# 4) EDA
basic_eda(df, TARGET, out_dir=os.path.join(OUTPUT_DIR, "eda_plots"), include

# 5) Prepare features and target
X = df.drop(columns=[TARGET])
y = df[TARGET]

# We'll run two configurations: with and without 'duration'
configurations = [("WITH_DURATION", False), ("WITHOUT_DURATION", True)]

for label, drop_duration in configurations:
    print("\\n==============================")
    print("Configuration:", label)
    print("==============================")

    preprocessor, num_features, cat_features = build_preprocessor(df, TARGE

    # 6) Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TES

    # 7) Train & evaluate baseline models
    results_df, pipelines, roc_info = train_and_evaluate(X_train, X_test, y_
    print("\\nModel comparison for", label)
    display(results_df)

    # Save results
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    results_df.to_csv(os.path.join(OUTPUT_DIR, f"model_results_{label}.csv"

    # 8) Plot ROC curves (if available)
    if roc_info:
        plot_roc(roc_info, title=f"ROC Curves ({label})")

    # 9) Optionally tune RandomForest
    if RUN_GRIDSEARCH:
        gs = grid_search_random_forest(X_train, y_train, preprocessor, rand
        best_rf = gs.best_estimator_
        # Evaluate tuned
        y_pred = best_rf.predict(X_test)
        y_proba = None
        if hasattr(best_rf, "predict_proba"):
            proba = best_rf.predict_proba(X_test)
            if proba.shape[1] == 2:
                y_proba = proba[:,1]
        tuned_metrics = evaluate_model(y_test, y_pred, y_proba)
        print("Tuned RF metrics:", tuned_metrics)
```

```
    # 10) Feature importance for best pipeline (by f1 in results_df)
    best_model_name = results_df.loc[0, "model"]
    best_pipe = pipelines[best_model_name]
    try:
        imp_df = extract_feature_importance(best_pipe, X_test)
        if not imp_df.empty:
            print("Top 20 features for", best_model_name)
            display(imp_df.head(20))
            imp_df.to_csv(os.path.join(OUTPUT_DIR, f"feature_importance_{lab
    except Exception as e:
        print("Could not extract feature importance:", e)

    # 11) Save best pipeline
    save_pipeline(best_pipe, os.path.join(OUTPUT_DIR, f"best_pipeline_{label

print("\\nAll configurations finished. Check the outputs folder for artifac
```

```
Loading: /Users/nikhilreddyponnala/Desktop/Data Analytics/Second Project/Cl
ient subscribed to a term deposit/Dataset/bankmarketing.csv
Loaded shape: (41188, 21)
Using target column: y
\n===== BASIC EDA =====
Shape: (41188, 21)
\nColumn dtypes:
 age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
\nHead:
    age        job  marital    education  default housing loan    contact  
\
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  month day_of_week  duration  campaign  pdays  previous     poutcome  \
0   may         mon       261         1    999         0  nonexistent
1   may         mon       149         1    999         0  nonexistent
2   may         mon       226         1    999         0  nonexistent
3   may         mon       151         1    999         0  nonexistent
4   may         mon       307         1    999         0  nonexistent

   emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0           1.1          93.994          -36.4      4.857       5191.0  no
1           1.1          93.994          -36.4      4.857       5191.0  no
2           1.1          93.994          -36.4      4.857       5191.0  no
3           1.1          93.994          -36.4      4.857       5191.0  no
4           1.1          93.994          -36.4      4.857       5191.0  no
\nClass balance:
y
no     0.887346
yes    0.112654
Name: proportion, dtype: float64
\nNo missing values detected.
\n=============================
Configuration: WITH_DURATION
=============================
\n--- Training LogisticRegression ---
LogisticRegression metrics: {'accuracy': 0.9163631949502307, 'precision':
0.7100175746924429, 'recall': 0.4353448275862069, 'f1': 0.539746158984636,
'roc_auc': 0.9424181270342941}
```

```
[[7145  165]
 [ 524  404]]
              precision    recall  f1-score   support

           0      0.932     0.977     0.954      7310
           1      0.710     0.435     0.540       928

    accuracy                          0.916      8238
   macro avg      0.821     0.706     0.747      8238
weighted avg      0.907     0.916     0.907      8238

\n--- Training DecisionTree ---
DecisionTree metrics: {'accuracy': 0.896091284292304, 'precision': 0.538793
1034482759, 'recall': 0.5387931034482759, 'f1': 0.5387931034482759, 'roc_au
c': 0.7401215859238643}
[[6882  428]
 [ 428  500]]
              precision    recall  f1-score   support

           0      0.941     0.941     0.941      7310
           1      0.539     0.539     0.539       928

    accuracy                          0.896      8238
   macro avg      0.740     0.740     0.740      8238
weighted avg      0.896     0.896     0.896      8238

\n--- Training RandomForest ---
RandomForest metrics: {'accuracy': 0.9208545763534839, 'precision': 0.72402
5974025974, 'recall': 0.48060344827586204, 'f1': 0.577720207253886, 'roc_au
c': 0.948476416930044}
[[7140  170]
 [ 482  446]]
              precision    recall  f1-score   support

           0      0.937     0.977     0.956      7310
           1      0.724     0.481     0.578       928

    accuracy                          0.921      8238
   macro avg      0.830     0.729     0.767      8238
weighted avg      0.913     0.921     0.914      8238

\nModel comparison for WITH_DURATION
```

	model	accuracy	precision	recall	f1	roc_auc
0	RandomForest	0.920855	0.724026	0.480603	0.577720	0.948476
1	LogisticRegression	0.916363	0.710018	0.435345	0.539746	0.942418
2	DecisionTree	0.896091	0.538793	0.538793	0.538793	0.740122

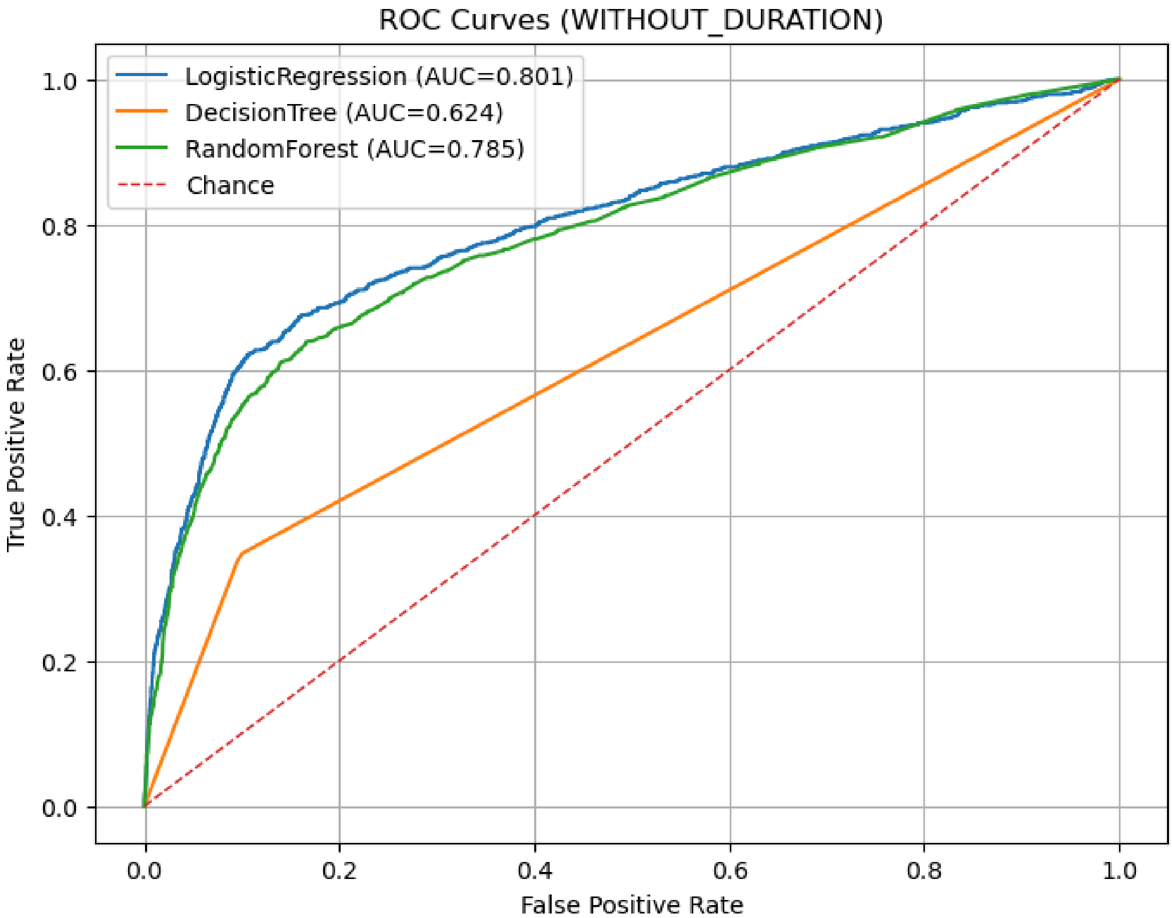



```
Top 20 features for RandomForest
```

	feature	importance
0	duration	0.275604
1	euribor3m	0.089149
2	age	0.081391
3	nr.employed	0.050342
4	campaign	0.039708
5	pdays	0.031904
6	cons.conf.idx	0.026639
7	emp.var.rate	0.024893
8	cons.price.idx	0.021355
9	poutcome_success	0.017230
10	housing_yes	0.013425
11	housing_no	0.013279
12	job_admin.	0.012212
13	education_university.degree	0.011921
14	previous	0.011825
15	day_of_week_mon	0.011656
16	marital_married	0.011625
17	education_high.school	0.011163
18	day_of_week_thu	0.010898
19	marital_single	0.010629

```
Saved model to: /Users/nikhilreddyponnala/Desktop/Data Analytics/Second Pro
ject/Client subscribed to a term deposit/Dataset/outputs/best_pipeline_WITH
_DURATION.joblib
\n==============================
Configuration: WITHOUT_DURATION
=============================
\n--- Training LogisticRegression ---
LogisticRegression metrics: {'accuracy': 0.9011896091284293, 'precision':
0.69, 'recall': 0.22306034482758622, 'f1': 0.3371335504885994, 'roc_auc':
0.8008559513420445}
[[7217   93]
 [ 721  207]]
              precision    recall  f1-score   support

           0      0.909     0.987     0.947      7310
           1      0.690     0.223     0.337       928

    accuracy                          0.901      8238
   macro avg      0.800     0.605     0.642      8238
weighted avg      0.884     0.901     0.878      8238

\n--- Training DecisionTree ---
DecisionTree metrics: {'accuracy': 0.84122359796067, 'precision': 0.31, 're
call': 0.33405172413793105, 'f1': 0.32157676348547715, 'roc_auc': 0.6239625
837303646}
[[6620  690]
 [ 618  310]]
              precision    recall  f1-score   support

           0      0.915     0.906     0.910      7310
           1      0.310     0.334     0.322       928

    accuracy                          0.841      8238
   macro avg      0.612     0.620     0.616      8238
weighted avg      0.847     0.841     0.844      8238

\n--- Training RandomForest ---
RandomForest metrics: {'accuracy': 0.8964554503520272, 'precision': 0.57731
95876288659, 'recall': 0.3017241379310345, 'f1': 0.39631988676574664, 'roc_
auc': 0.7850626798433888}
[[7105  205]
 [ 648  280]]
              precision    recall  f1-score   support

           0      0.916     0.972     0.943      7310
           1      0.577     0.302     0.396       928

    accuracy                          0.896      8238
   macro avg      0.747     0.637     0.670      8238
weighted avg      0.878     0.896     0.882      8238

\nModel comparison for WITHOUT_DURATION
```

	model	accuracy	precision	recall	f1	roc_auc
0	RandomForest	0.896455	0.57732	0.301724	0.396320	0.785063
1	LogisticRegression	0.901190	0.69000	0.223060	0.337134	0.800856
2	DecisionTree	0.841224	0.31000	0.334052	0.321577	0.623963




```
Top 20 features for RandomForest
```

	feature	importance
0	age	0.157801
1	euribor3m	0.118479
2	campaign	0.081813
3	nr.employed	0.044044
4	pdays	0.035113
5	cons.conf.idx	0.026467
6	emp.var.rate	0.023362
7	poutcome_success	0.023222
8	cons.price.idx	0.021792
9	housing_yes	0.020780
10	housing_no	0.020500
11	job_admin.	0.017844
12	marital_married	0.017009
13	education_university.degree	0.016889
14	education_high.school	0.016406
15	day_of_week_mon	0.015714
16	previous	0.015374
17	marital_single	0.015287
18	day_of_week_wed	0.015167
19	day_of_week_thu	0.014946

```
Saved model to: /Users/nikhilreddyponnala/Desktop/Data Analytics/Second Pro
ject/Client subscribed to a term deposit/Dataset/outputs/best_pipeline_WITH
OUT_DURATION.joblib
\nAll configurations finished. Check the outputs folder for artifacts.
```

Conclusion

The analysis highlights that timing, communication channel, and client engagement quality play significant roles in determining whether a client subscribes to a term deposit. By focusing on these factors and leveraging predictive models, banks can optimize campaign strategies, improve customer targeting, and enhance conversion rates.

In [ ]: