BTA 419 Finance Project by McPython

UCI Bank Marketing - https://archive.ics.uci.edu/dataset/222/bank+marketing

Background of Data/Business Definition

Background of UCI Bank Marketing Dataset

Our client is a Portuguese banking institution that provides financial services, including savings accounts, loans, and investment products. One of its key offerings is term deposits, which provide customers with a secure way to grow their savings over a fixed period.

To increase subscriptions to term deposits, the bank relies on outbound telemarketing campaigns, calling potential customers to offer the product. However, these campaigns have low success rates, leading to wasted resources and unnecessary customer dissatisfaction. The bank wants to use predictive analytics to identify which customers are most likely to subscribe, allowing for more targeted and efficient marketing efforts.

Business Question

What factors influence a client's likelihood of subscribing?

Data Dictionary

Bank Client Data

```
SELECT *
FROM 'Data Dictionary - UCI Bank Marketing.csv'
```

	Variable Name o...	Role object	Data Type object	Description object	Positive Values / ...	Units object	Missing Values o...
	age 5.9% job 5.9% 15 others 88.2%	Feature 94.1% Target 5.9%	Categorical 41.2% Integer 35.3% Binary 23.5%	Client's age 5.9% Client's oc... 5.9% 15 others 88.2%	Positive i... 23.5% "yes", "no" 23.5% 9 others 52.9%	- 64.7% Count 11.8% 4 others 23.5%	No 82.4% Yes 17.6%
0	age	Feature	Integer	Client's age	Positive integers o	Years	No
1	job	Feature	Categorical	Client's occupation	"admin.", "blue-coll	-	No
2	marital	Feature	Categorical	Client's marital stat	"married", "divorce	-	No
3	education	Feature	Categorical	Client's highest ed	"basic.4y", "basic.6	-	No
4	default	Feature	Binary	Has credit in defaul	"yes", "no"	-	No
5	balance	Feature	Integer	Client's average ye	Positive or negativ	Euros	No
6	housing	Feature	Binary	Has a housing loa	"yes", "no"	-	No
7	loan	Feature	Binary	Has a personal loa	"yes", "no"	-	No
8	contact	Feature	Categorical	Contact communic	"cellular", "telephon	-	Yes
9	day_of_week	Feature	Categorical	Day of the week th	"mon", "tue", "wed",	-	No
10	month	Feature	Categorical	Month the client w	"jan", "feb", "mar",	-	No
11	duration	Feature	Integer	Last contact durati	Positive integers o	Seconds	No
12	campaign	Feature	Integer	Number of contact	Positive integers o	Count	No
13	pdays	Feature	Integer	Days since the clie	-1 = never contacte	Days	Yes
14	previous	Feature	Integer	Number of contact	Positive integers o	Count	No
15	poutcome	Feature	Categorical	Outcome of the pre	"failure", "nonexiste	-	Yes
16	y	Target	Binary	Has the client subs	"yes", "no"	-	No

17 rows, 7 cols 25 / page Page 1 of 1

- Age (numeric)
- Job (categorical): "admin", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue collar", "self-employed", "retired", "technician", "services"
- Martial Status (categorical): "married", "divorced", "single" (note: "divorced" means divorced or widowed)
- Education (categorical): "unknown", "primary", "secondary", "tertiary")

- Default - Credit in default (binary): "yes", "no"
- Balance - Average yearly balance (numeric in euros)
- Housing Loan (binary): "yes", "no"
- Personal Loan (binary): "yes", "no"

# Related with the last contact of the current campaign

- Contact Communication Type (categorical): "unknown", "telephone", "cellular"
- Day - Last contact day of the month (numeric)
- Month - Last contact month (categorical): "jan", "feb", "mar", ..., "nov", "dec"
- Duration - Last contact duration, in seconds (numeric)

Other Attributes

- Campaign. -Number of contacts during this campaign (numeric, includes last contact)
- Pdays - Days since last contact from a previous campaign (numeric, -1 means never contacted)
- Previous - Number of contacts before this campaign (numeric)
- Poutcome - Outcome of the previous campaign (categorical): "unknown", "other", "failure", "success"

Output variable (desired target)

- y - has the client subscribed to a term deposit? (binary): "yes", "no"

Load & Explore Data

```python
from ucimlrepo import fetch_ucirepo

# Fetch dataset (ID 222 is for Bank Marketing Dataset)
bank_marketing = fetch_ucirepo(id=222)

# Extract data as pandas DataFrames
X = bank_marketing.data.features
y = bank_marketing.data.targets

# Display metadata
print(bank_marketing.metadata)
print(bank_marketing.variables)
```

```
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
```

```python
# Display first few rows of the data
print(X.head())
print(y.value_counts())
```

```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married        NaN      no     1506     yes   no
4   33           NaN   single        NaN      no        1      no   no

  contact  day_of_week month  duration  campaign  pdays  previous poutcome
0     NaN            5   may       261         1     -1         0      NaN
1     NaN            5   may       151         1     -1         0      NaN
2     NaN            5   may        76         1     -1         0      NaN
3     NaN            5   may        92         1     -1         0      NaN
4     NaN            5   may       198         1     -1         0      NaN
y
no     39922
yes     5289
Name: count, dtype: int64
```

Review Data

```
# Displaying information about the feature DataFrame X
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   age          45211 non-null  int64 
 1   job          44923 non-null  object
 2   marital      45211 non-null  object
 3   education    43354 non-null  object
 4   default      45211 non-null  object
 5   balance      45211 non-null  int64 
 6   housing      45211 non-null  object
 7   loan         45211 non-null  object
 8   contact      32191 non-null  object
 9   day_of_week  45211 non-null  int64 
 10  month        45211 non-null  object
 11  duration     45211 non-null  int64 
 12  campaign     45211 non-null  int64 
 13  pdays        45211 non-null  int64 
 14  previous     45211 non-null  int64 
 15  poutcome     8252 non-null   object
dtypes: int64(7), object(9)
memory usage: 5.5+ MB
```

This code clearly shows the number of missing values.

```
print(X.isnull().sum())
```

```
age                0
job              288
marital            0
education       1857
default            0
balance            0
housing            0
loan               0
contact        13020
day_of_week        0
month              0
duration           0
campaign           0
pdays              0
previous           0
poutcome       36959
dtype: int64
```

Now, we need to display the number of entirely non-null rows.

```
# Create a DataFrame with only non-null rows
X_non_null = X.dropna()

# Get the number of entirely non-null rows
non_null_rows = X_non_null.shape[0]
print(f"Number of entirely non-null rows: {non_null_rows}")
X_non_null.info()
```

```
Number of entirely non-null rows: 7842
<class 'pandas.core.frame.DataFrame'>
Index: 7842 entries, 24060 to 45210
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   age          7842 non-null   int64 
 1   job          7842 non-null   object
 2   marital      7842 non-null   object
 3   education    7842 non-null   object
 4   default      7842 non-null   object
 5   balance      7842 non-null   int64 
 6   housing      7842 non-null   object
 7   loan         7842 non-null   object
 8   contact      7842 non-null   object
 9   day_of_week  7842 non-null   int64 
 10  month        7842 non-null   object
 11  duration     7842 non-null   int64 
 12  campaign     7842 non-null   int64 
 13  pdays        7842 non-null   int64 
 14  previous     7842 non-null   int64 
 15  poutcome     7842 non-null   object
dtypes: int64(7), object(9)
memory usage: 1.0+ MB
```

We have 7842 rows, which means we lost 37,369 rows. It likely won't be possible to drop the rows since it is 83% of our data. Our missing values come from the following variables: job, education, contact, and poutcome, with the vast majority of missing values coming from poutcome. poutcome variable is our historical outcome, meaning we won't be able to predict without that data, so it's best that we figure out how to clean it. The second largest amount of missing values comes from "contact". This variable describes whether the potential clients were contacted via cellphone or telephone or unknown.

Now, we will begin our data cleaning.

Data Cleaning

Data Cleaning

Deal With Missing Values

Deal With Missing Values

```python
import pandas as pd

# Create a copy of the DataFrame
# Create a copy to avoid altering the original data
X = X.copy()

# Step 1: Drop rows with missing 'job' (only 288 rows)
print("Missing values in 'job' before dropping rows:", X['job'].isnull().sum())

X.dropna(subset=['job'], inplace=True)

# Step 2: Clean 'education' - replace missing with 'unknown'
X['education'].fillna('unknown', inplace=True)

# Step 3: Clean 'contact' - replace missing with 'unknown'
X['contact'].fillna('unknown', inplace=True)

# Step 4: Clean 'poutcome' - replace missing with 'missing'
X['poutcome'].fillna('missing', inplace=True)

# Check if there are any remaining missing values
print("Missing values after cleaning:")
print(X.copy().isnull().sum())



# Display info about cleaned DataFrame
X.copy().info()
```

```
Missing values in 'job' before dropping rows: 288
Missing values after cleaning:
age            0
job            0
marital        0
education      0
default        0
balance        0
housing        0
loan           0
contact        0
day_of_week    0
month          0
duration       0
campaign       0
pdays          0
previous       0
poutcome       0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
Index: 44923 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          44923 non-null  int64
 1   job          44923 non-null  object
 2   marital      44923 non-null  object
 3   education    44923 non-null  object
 4   default      44923 non-null  object
 5   balance      44923 non-null  int64
```

We started with a total of 45,211 rows in the dataset. After dropping 288 rows with missing values in the "job" variable and imputing missing values in the "education," "poutcome," and "contact" columns, we now have a total of 44,923 rows. This means we removed about 0.63% of the rows, which is well below the 5% threshold that is generally considered a reasonable guideline when removing data.

We started with a total of 45,211 rows in the dataset. After dropping 288 rows with missing values in the "job" variable and imputing missing values in the "education," "poutcome," and "contact" columns, we now have a total of 44,923 rows. This means we removed about 0.63% of the rows, which is well below the 5% threshold that is generally considered a reasonable guideline when removing data.

Checking for Outliers

Checking for Outliers

Now that there are no more missing values, we will check for outliers in our numerical data that likely have outliers. Categorical data can be left alone.

Now that there are no more missing values, we will check for outliers in our numerical data that likely have outliers. Categorical data can be left alone.

```
print(X[['balance', 'duration', 'campaign','age', 'pdays', 'previous']].describe())
```

```
             balance      duration      campaign           age         pdays  \
count   44923.000000  44923.000000  44923.000000  44923.000000  44923.000000
mean     1359.643011    258.294838      2.760345     40.893529     40.321016
std      3045.091520    257.713770      3.092838     10.604399    100.255146
min     -8019.000000      0.000000      1.000000     18.000000     -1.000000
25%        72.000000    103.000000      1.000000     33.000000     -1.000000
50%       447.000000    180.000000      2.000000     39.000000     -1.000000
75%      1421.000000    319.000000      3.000000     48.000000     -1.000000
max    102127.000000   4918.000000     63.000000     95.000000    871.000000

           previous
count  44923.000000
mean       0.581996
std        2.309077
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max      275.000000
```

```
print(X[['balance', 'duration', 'campaign','age', 'pdays', 'previous']].describe())
```

```
            balance      duration      campaign           age         pdays  \
count  44922.000000  44922.000000  44922.000000  44922.000000  44922.000000
mean    1114.581898    242.312186      2.509082     40.893749     40.321936
std     1579.600264    191.866057      1.902248     10.604414    100.256072
min     -174.000000     35.000000      1.000000     18.000000     -1.000000
25%       72.000000    103.000000      1.000000     33.000000     -1.000000
50%      447.000000    180.000000      2.000000     39.000000     -1.000000
75%     1421.000000    319.000000      3.000000     48.000000     -1.000000
max     5763.000000    751.000000      8.000000     95.000000    871.000000

           previous
count  44922.000000
mean       0.527025
std        1.465616
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        9.000000
```

Just based on the maximum values compared to our 75% percentile of each variable, it seems like we might have at least one outlier per variable shown, except for variables "age" and "pdays". Time to look into that! I will try to fix this issue by Capping/Flooring. This method will limit the extreme values to a specified threshold. My first test will be by capping the values at the 99th percentile and flooring them as well by the 1 percentile. I will leave "age" and "pdays" out of the test.

Just based on the maximum values compared to our 75% percentile of each variable, it seems like we might have at least one outlier per variable shown, except for variables "age" and "pdays". Time to look into that! I will try to fix this issue by Capping/Flooring. This method will limit the extreme values to a specified threshold. My first test will be by capping the values at the 99th percentile and flooring them as well by the 1 percentile. I will leave "age" and "pdays" out of the test.

```python
# Define the columns to cap/floor
columns = ['balance', 'duration', 'campaign', 'previous']

# Calculate 1st and 99th percentiles for each column
for col in columns:
    lower_bound = X[col].quantile(0.01)  # 1st percentile
    upper_bound = X[col].quantile(0.99)  # 99th percentile
    print(f"{col} - 1st percentile: {lower_bound}, 99th percentile: {upper_bound}")
```

```
balance - 1st percentile: -627.78, 99th percentile: 13159.119999999995
duration - 1st percentile: 11.0, 99th percentile: 1270.5599999999977
campaign - 1st percentile: 1.0, 99th percentile: 16.0
previous - 1st percentile: 0.0, 99th percentile: 9.0
```

```python
# Define the columns to cap/floor
columns = ['balance', 'duration', 'campaign', 'previous']

# Calculate 1st and 99th percentiles for each column
for col in columns:
    lower_bound = X[col].quantile(0.01)  # 1st percentile
    upper_bound = X[col].quantile(0.99)  # 99th percentile
    print(f"{col} - 1st percentile: {lower_bound}, 99th percentile: {upper_bound}")
```

```
balance - 1st percentile: -174.0, 99th percentile: 5763.0
duration - 1st percentile: 35.0, 99th percentile: 751.0
campaign - 1st percentile: 1.0, 99th percentile: 8.0
previous - 1st percentile: 0.0, 99th percentile: 9.0
```

This still seems too far off for each variable, so now I will try the upper 95 and lower 5.

This still seems too far off for each variable, so now I will try the upper 95 and lower 5.

```python
# Define the columns to cap/floor
columns = ['balance', 'duration', 'campaign', 'previous']

# Calculate 1st and 99th percentiles for each column
for col in columns:
    lower_bound = X[col].quantile(0.05)  # 5th percentile
    upper_bound = X[col].quantile(0.95)  # 95th percentile
    print(f"{col} - 5th percentile: {lower_bound}, 95th percentile: {upper_bound}")
```

```
balance - 5th percentile: -174.0, 95th percentile: 5763.0
duration - 5th percentile: 35.0, 95th percentile: 751.0
campaign - 5th percentile: 1.0, 95th percentile: 8.0
previous - 5th percentile: 0.0, 95th percentile: 3.0
```

```python
# Define the columns to cap/floor
columns = ['balance', 'duration', 'campaign', 'previous']

# Calculate 1st and 99th percentiles for each column
for col in columns:
    lower_bound = X[col].quantile(0.05)  # 5th percentile
    upper_bound = X[col].quantile(0.95)  # 95th percentile
    print(f"{col} - 5th percentile: {lower_bound}, 95th percentile: {upper_bound}")
```

```
balance - 5th percentile: -174.0, 95th percentile: 5763.0
duration - 5th percentile: 35.0, 95th percentile: 751.0
campaign - 5th percentile: 1.0, 95th percentile: 8.0
previous - 5th percentile: 0.0, 95th percentile: 3.0
```

This seems like a much better distribution of data overall, but I will leave the variables "previous" at 99 upper and 1 lower. Below is the official run with their respective boundary cutoffs. I believe leaving the variable "previous" at 99% upper and 1% lower is best because it leaves out the extreme outliers enough without needing to take out extra potentially important data that could help our predictions. Below, you will see the new summary statistics that contain variables within the distribution. This will help us predict more accurately in the future.

```python
# Define columns and their capping strategies
columns_95_5 = ['balance', 'duration', 'campaign']
columns_99_1 = ['previous']

# Cap and floor at 95th & 5th percentiles for specified columns
for col in columns_95_5:
    lower_bound = X[col].quantile(0.05)  # 5th percentile
    upper_bound = X[col].quantile(0.95)  # 95th percentile
    X[col] = X[col].clip(lower=lower_bound, upper=upper_bound)

# Cap and floor at 99th & 1st percentiles for 'previous'
for col in columns_99_1:
    lower_bound = X[col].quantile(0.01)  # 1st percentile
    upper_bound = X[col].quantile(0.99)  # 99th percentile
    X[col] = X[col].clip(lower=lower_bound, upper=upper_bound)

# Print new summary statistics
print(X[['balance', 'duration', 'campaign', 'previous']].describe())
```

```
            balance      duration      campaign      previous
count  44923.000000  44923.000000  44923.000000  44923.000000
mean    1114.557087    242.307571      2.509205      0.527013
std     1579.591436    191.866415      1.902403      1.465602
min     -174.000000     35.000000      1.000000      0.000000
25%       72.000000    103.000000      1.000000      0.000000
50%      447.000000    180.000000      2.000000      0.000000
75%     1421.000000    319.000000      3.000000      0.000000
max     5763.000000    751.000000      8.000000      9.000000
```

Dropping Duplicate Values

Dropping Duplicate Values

```python
# Check for duplicate rows
duplicate_count = X.duplicated().sum()
print(f"Total duplicate rows: {duplicate_count}")
```

```
Total duplicate rows: 1
```

```python
# Show duplicate rows
duplicates = X[X.duplicated()]
print("Duplicate rows:")
print(duplicates)
```

```
Duplicate rows:
       age         job marital education default  balance housing loan  \
22789   31  management  single  tertiary      no        0      no   no   

        contact  day_of_week month  duration  campaign  pdays  previous  \
22789  cellular           25   aug        35         8     -1         0   

      poutcome  
22789  missing  
```

```python
# Remove duplicate rows
X = X.drop_duplicates()

# Confirm duplicates are removed
print(f"Total duplicate rows after removal: {X.duplicated().sum()}")
```

```
Total duplicate rows after removal: 0
```

Summary of Data Cleanse

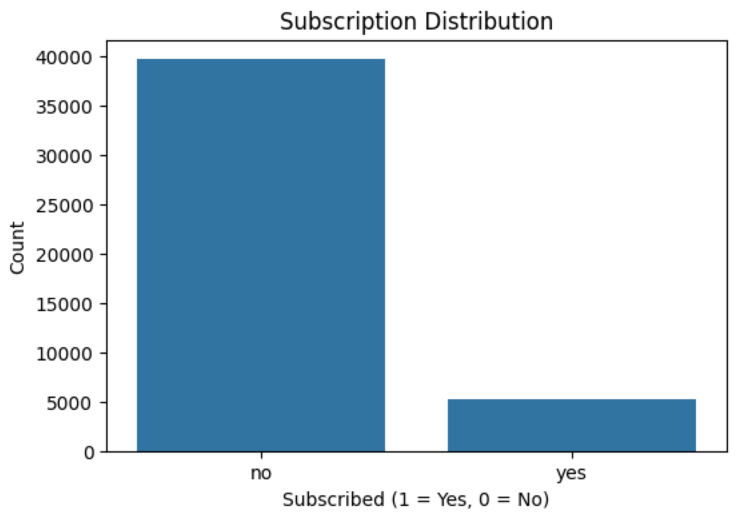
Summary of Data Cleanse

After going through the data-cleaning process, we feel pretty confident that the dataset is in good shape now. We handled missing values by imputing or replacing them appropriately, ensuring that there were no null values left to worry about. We also took care of outliers by capping and flooring them at the 95th and 5th percentiles for most variables and the 99th and 1st percentiles for the previous column to manage extreme values without losing valuable information. On top of that, we checked for and removed any duplicate rows, making sure only unique data remains. The summary statistics show that the ranges for all columns look reasonable, and the data types are consistent, so we think the dataset is clean and ready for the next steps, like encoding categorical variables or building models.

After going through the data-cleaning process, we feel pretty confident that the dataset is in good shape now. We handled missing values by imputing or replacing them appropriately, ensuring that there were no null values left to worry about. We also took care of outliers by capping and flooring them at the 95th and 5th percentiles for most variables and the 99th and 1st percentiles for the previous column to manage extreme values without losing valuable information. On top of that, we checked for and removed any duplicate rows, making sure only unique data remains. The summary statistics show that the ranges for all columns look reasonable, and the data types are consistent, so we think the dataset is clean and ready for the next steps, like encoding categorical variables or building models.

Data Exploration & Data Manipulation

```python
# Merge y into X for visualization purposes
X['y'] = y

# Now, the count plot should work
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
sns.countplot(x='y', data=X)
plt.title("Subscription Distribution")
plt.xlabel("Subscribed (1 = Yes, 0 = No)")
plt.ylabel("Count")
plt.show()

# Distribution of the outcome variable
print("Subscription Distribution:")
print(X['y'].value_counts())
```



```
Subscription Distribution:
y
no     39667
yes     5255
Name: count, dtype: int64
```

Summary Statistics

```python
# Merge y into X temporarily for grouping
X['y'] = y

# Calculate mean only for numerical columns grouped by 'y'
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns
print("Summary Statistics by Subscription Status:")
print(X.groupby('y')[numerical_cols].mean())

# Remove 'y' from X after analysis if needed
X = X.drop(columns=['y'])
```

```python
# Check if 'y' is defined
try:
    print(f"Length of X: {len(X)}")
    print(f"Length of y: {len(y)}")
except NameError:
    print("Error: 'y' is not defined. Make sure you have executed the cell that defines 'y'.")
```

```
# Keep a copy of the original 'y' just in case
y_original = y.copy()

# Reindex 'y' to match the current index of 'X' after dropping rows
y = y.loc[X.index]  # Align 'y' with the current index of 'X'

# Confirm lengths match
print(f"Length of X: {len(X)}")
print(f"Length of y: {len(y)}")
```

```
Length of X: 44922
Length of y: 44922
```

Mergin y into X

One Thing to Note...

This plot shows that pdays and previous are highly correlated, meaning we likely won't include both in the same model to stay away from overfitting. Instead, we'll test each separately to determine which one contributes more to the model's performance.

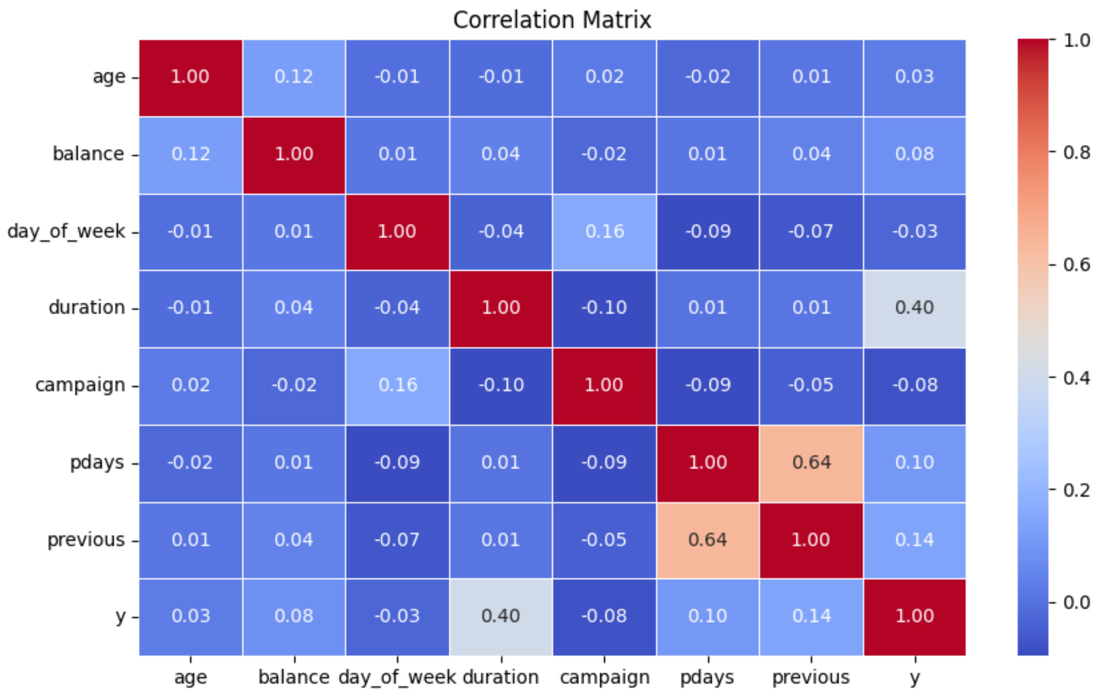
```
# Check if 'y' exists first
if 'y' not in X.columns:
    # Reinsert 'y' column
    X = X.copy()  # Create a copy to avoid in-place modification
    X['y'] = y  # Assuming 'y' is still stored separately
    print("'y' column has been reinserted.")
else:
    print("'y' column already exists in X.")

# Now convert 'y' to binary numeric format
X['y'] = X['y'].map({'yes': 1, 'no': 0})

# Confirm conversion
print(X['y'].value_counts())
```

```
'y' column has been reinserted.
y
0    39667
1     5255
Name: count, dtype: int64
```

Correlation Matrix

```python
# Drop categorical columns for correlation analysis
numerical_X = X.select_dtypes(include=['float64', 'int64'])

# Correlation matrix focused on 'y'
corr_matrix = numerical_X.corr()
subscription_corr = corr_matrix['y'].sort_values(ascending=False)

print("Correlations with Subscription Status:")
print(subscription_corr)

# Visualize with a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

```
Correlations with Subscription Status:
y              1.000000
duration       0.401650
previous       0.140713
pdays          0.102582
balance        0.080707
age            0.025404
day_of_week   -0.028645
campaign      -0.082844
Name: y, dtype: float64
```



	age	balance	day_of_week	duration	campaign	pdays	previous	y
age	1.00	0.12	-0.01	-0.01	0.02	-0.02	0.01	0.03
balance	0.12	1.00	0.01	0.04	-0.02	0.01	0.04	0.08
day_of_week	-0.01	0.01	1.00	-0.04	0.16	-0.09	-0.07	-0.03
duration	-0.01	0.04	-0.04	1.00	-0.10	0.01	0.01	0.40
campaign	0.02	-0.02	0.16	-0.10	1.00	-0.09	-0.05	-0.08
pdays	-0.02	0.01	-0.09	0.01	-0.09	1.00	0.64	0.10
previous	0.01	0.04	-0.07	0.01	-0.05	0.64	1.00	0.14
y	0.03	0.08	-0.03	0.40	-0.08	0.10	0.14	1.00

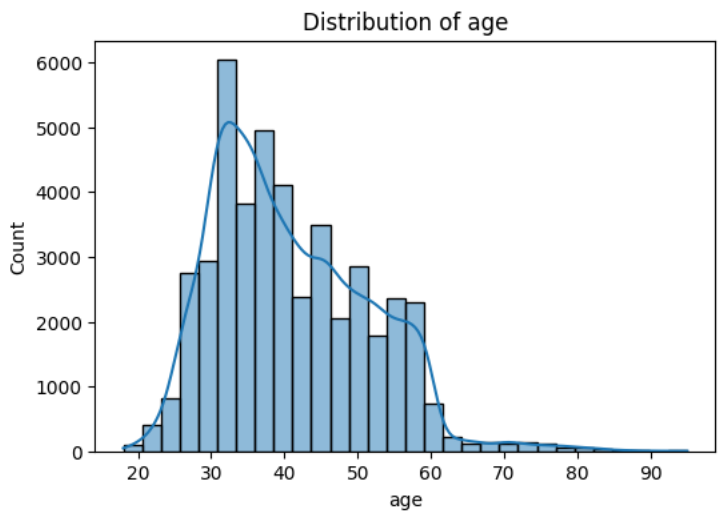
From the correlation matrix, we can see that the most influential factor for predicting whether a client will subscribe is the duration of the call, which has the strongest positive correlation with subscription status. Previous interactions and the number of days since the last contact also show a positive impact, suggesting that past engagements might increase the chances of subscribing. On the other hand, the number of contact attempts has a slight negative correlation, indicating that too many calls might reduce the likelihood of success. Factors like age, day of the week, and balance seem to have minimal influence on whether a client subscribes. Based on this, we should focus more on call duration and past interactions in our analysis while deprioritizing less relevant factors.

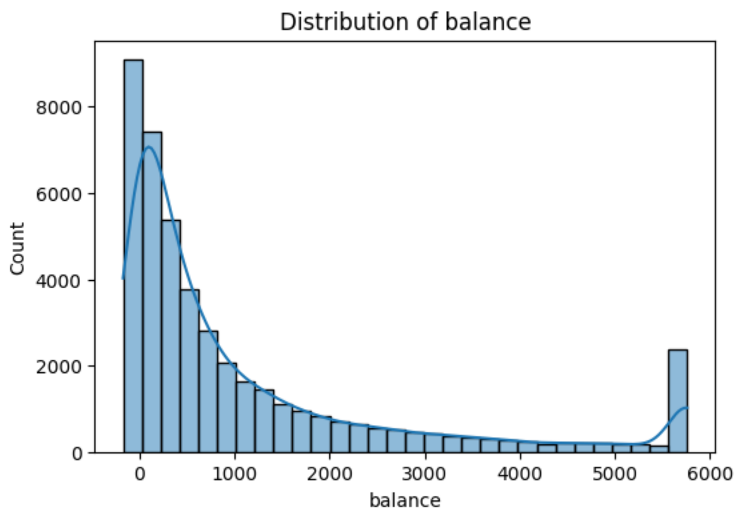
Relationship Between Features and y

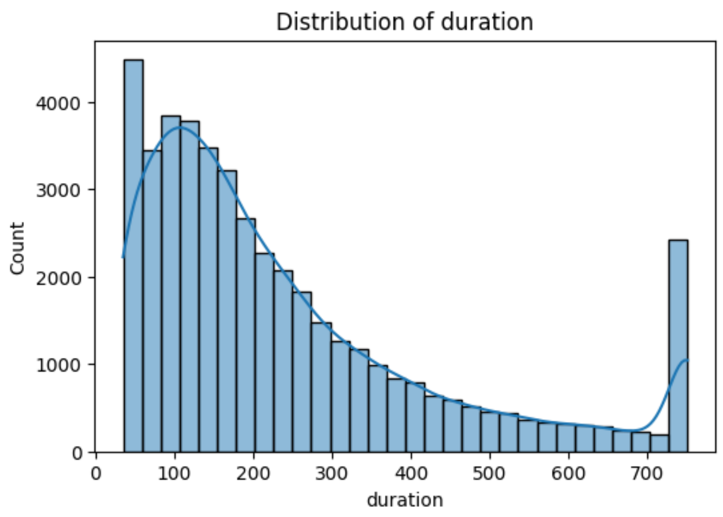
We explored the relationship between various features and the target variable, y, by visualizing their distributions. **For the numerical features such as age, balance, duration, campaign, pdays, and previous,** we used histograms with kernel density estimates to understand how these variables are spread across the dataset. The histograms helped us identify patterns, skewness, or any potential outliers that could impact our analysis.

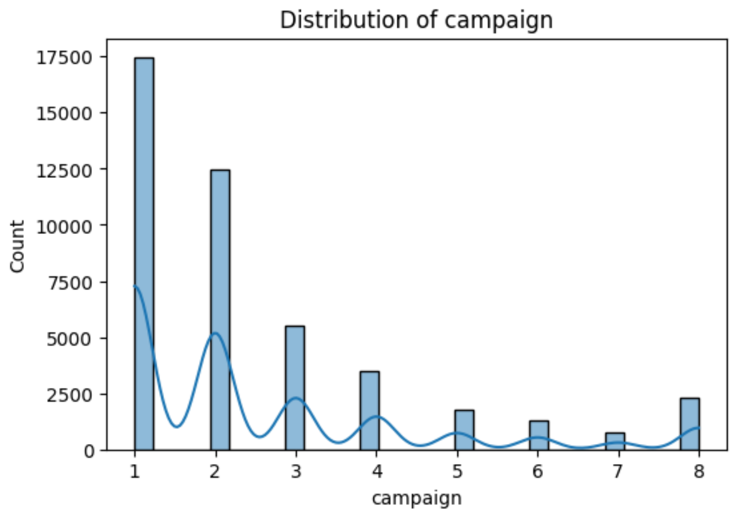
For the categorical features like job, marital status, education, and contact, we created bar plots to examine the distribution of categories within each variable. By doing this, we gained insights into how these categories are represented and how balanced they are, which is crucial for ensuring reliable model training. These visualizations provided a clearer picture of the data, helping us identify which features might have a significant influence on a client's likelihood of subscribing.

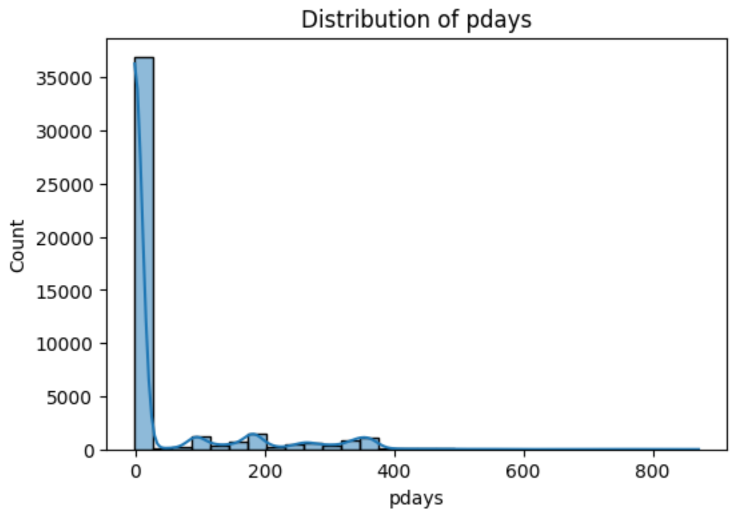
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram for numerical features
numerical_columns = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']
for col in numerical_columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(X[col], bins=30, kde=True)
    plt.title(f"Distribution of {col}")
    plt.show()

# Bar plot for categorical features
categorical_columns = ['job', 'marital', 'education', 'contact']
for col in categorical_columns:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=col, data=X)
    plt.title(f"Category Distribution for {col}")
    plt.xticks(rotation=45)
    plt.show()
```

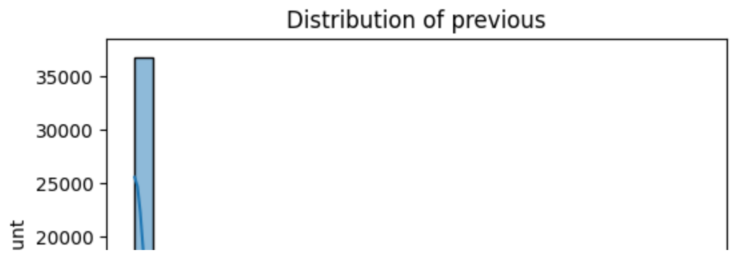


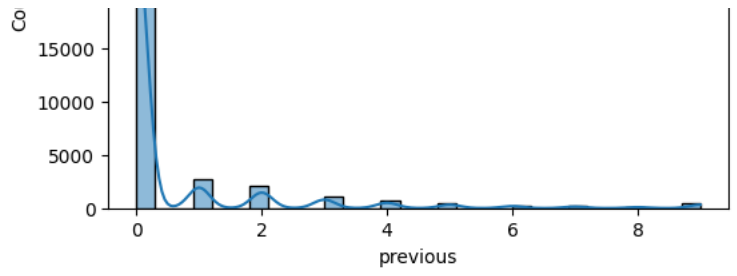


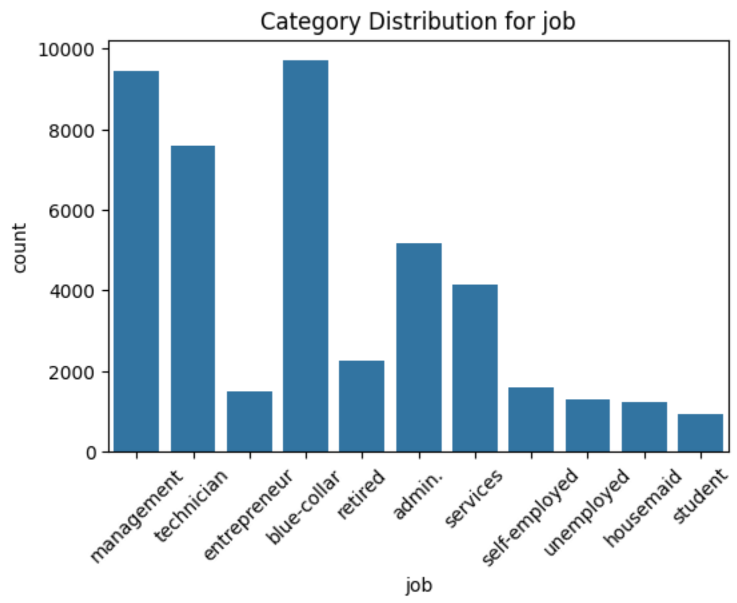

Distribution of duration
Count
duration
0
1000
2000
3000
4000
100
200
300
400
500
600
700

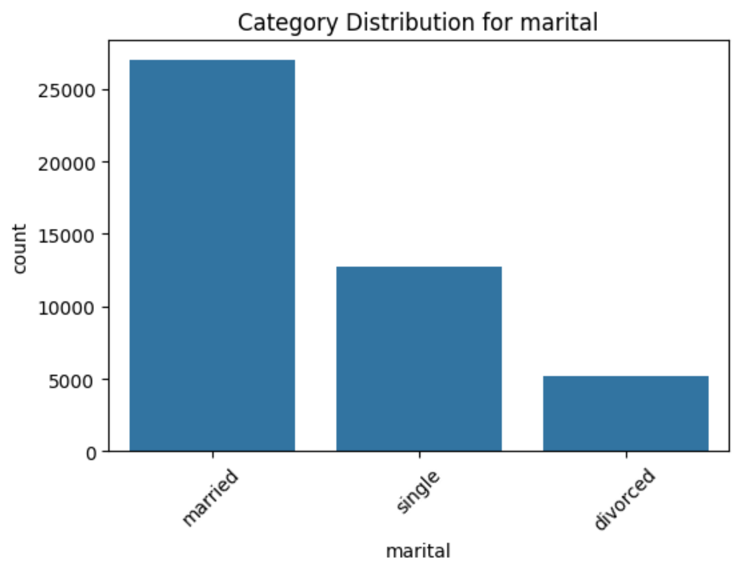

Distribution of campaign
Count
campaign
0
2500
5000
7500
10000
12500
15000
17500
1
2
3
4
5
6
7
8

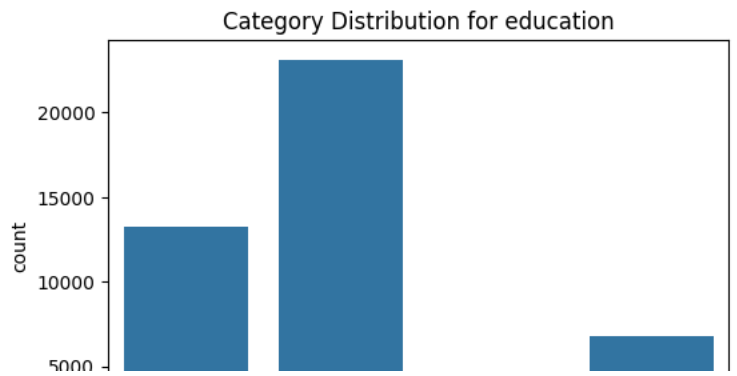

Distribution of pdays
Count
pdays
0
5000
10000
15000
20000
25000
30000
35000
200
400
600
800

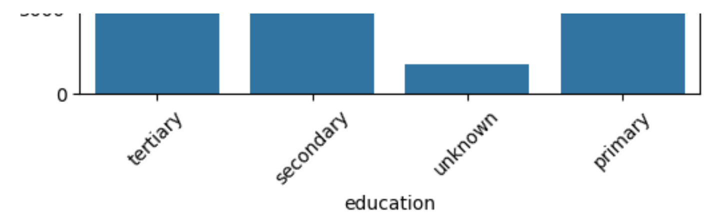

Distribution of previous
unt
20000
25000
30000
35000

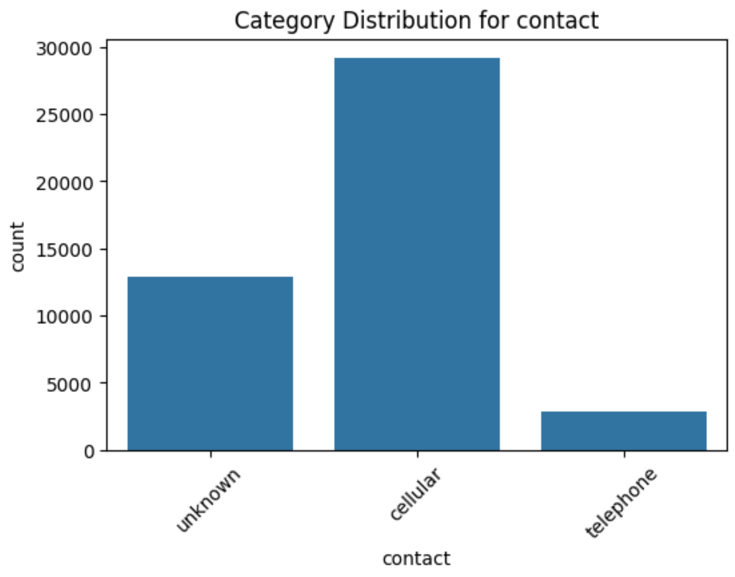

Co
15000
10000
5000
0
0
2
4
6
8
previous


Category Distribution for job
10000
8000
6000
4000
2000
0
count
management
technician
entrepreneur
blue-collar
retired
admin.
services
self-employed
unemployed
housemaid
student
job


Category Distribution for marital
25000
20000
15000
10000
5000
0
count
married
single
divorced
marital

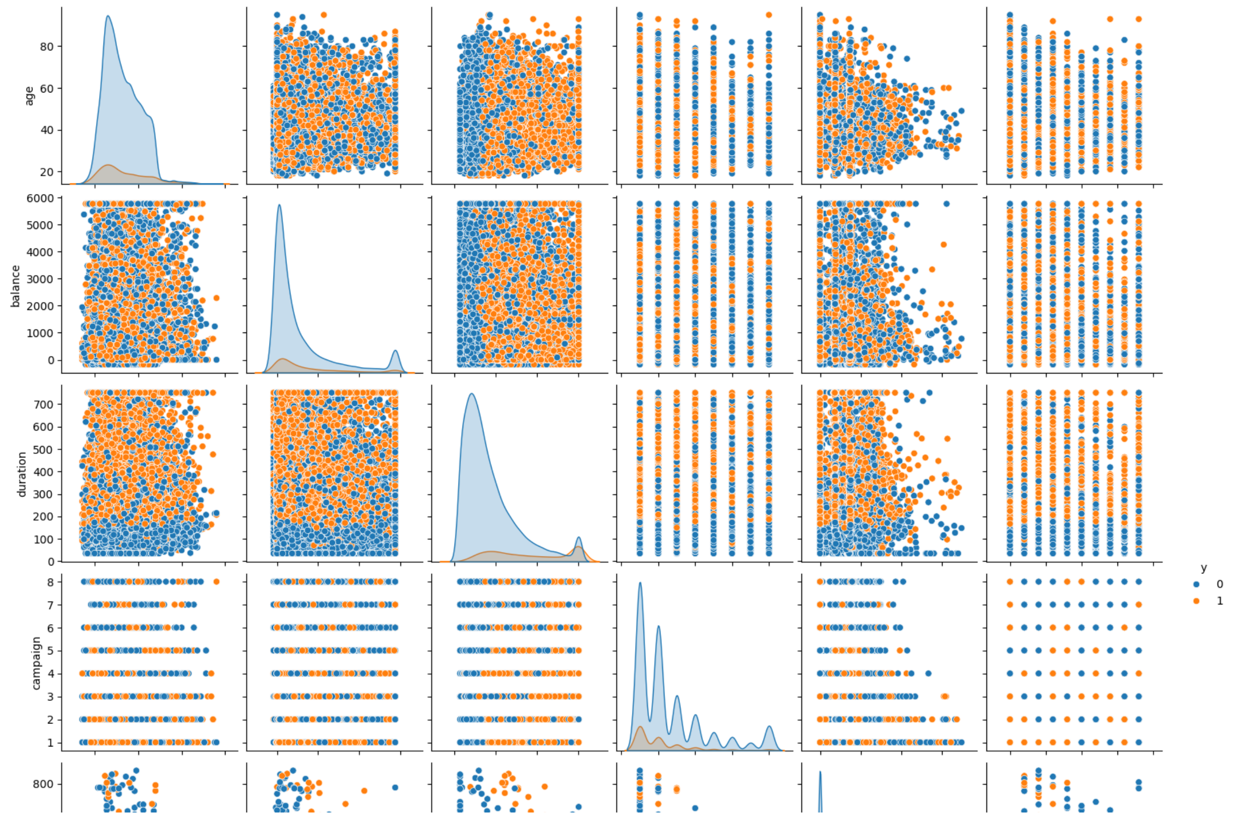

Category Distribution for education
20000
15000
10000
5000
count

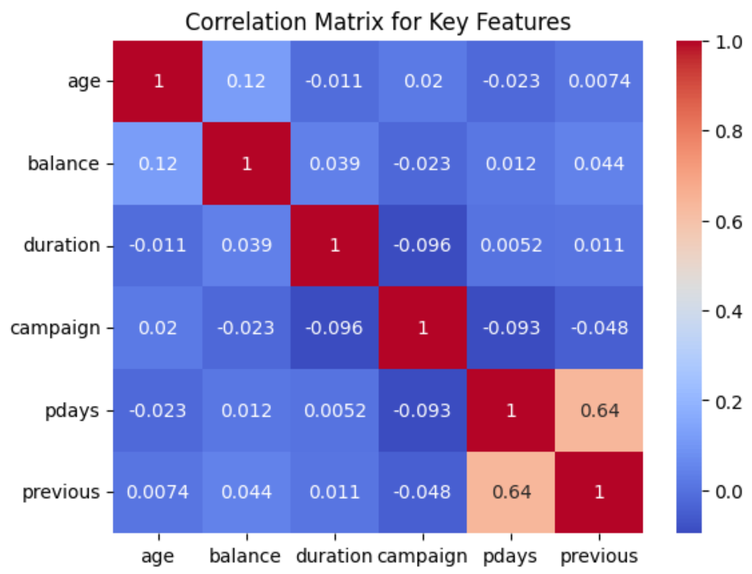




Interaction Between Multiple Features

```python
# Pair plot for selected numerical features
sns.pairplot(X[numerical_columns + ['y']], hue='y')
plt.show()

# Focused heatmap for selected variables
sns.heatmap(X[numerical_columns].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix for Key Features")
plt.show()
```





	age	balance	duration	campaign	pdays	previous
age	1	0.12	-0.011	0.02	-0.023	0.0074
balance	0.12	1	0.039	-0.023	0.012	0.044
duration	-0.011	0.039	1	-0.096	0.0052	0.011
campaign	0.02	-0.023	-0.096	1	-0.093	-0.048
pdays	-0.023	0.012	0.0052	-0.093	1	0.64
previous	0.0074	0.044	0.011	-0.048	0.64	1

From this pair plot, we can see that most features have weak linear relationships with each other, which aligns with what we saw in the correlation matrix. The scatter plots don't show any strong patterns or clear separations between the classes, which suggests that no single pair of features is likely to be a strong predictor on its own. We also notice that features like age, balance, and duration have skewed distributions, so we might need to apply transformations to make them more suitable for modeling. Overall, it looks like we'll need to rely on a combination of features rather than expecting any single one to do most of the work.

From the correlation matrix, we can see that most of the features have weak correlations with each other, which suggests that they likely carry unique information for our analysis. The only notable exception is the moderate positive correlation between pdays and previous at 0.64, meaning that the number of days since a client was last contacted tends to increase along with the number of previous contacts. This relationship might indicate some overlap in information between these two features, which is something we should keep in mind to avoid multicollinearity in our predictive models. Overall, the weak correlations among most features are a good sign, as they suggest that each variable could add distinct value to our analysis.

Statistical Testing

```
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']
numerical_cols = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']
```

```
from scipy.stats import chi2_contingency, ttest_ind

# Chi-square test for categorical variables
for col in categorical_cols:
    if col in X.columns:  # Check if column exists in X
        contingency = pd.crosstab(X[col], X['y'])
        chi2, p, _, _ = chi2_contingency(contingency)
        print(f"Chi-square test for {col}: p-value = {p:.4f}")
    else:
        print(f"Column {col} not found in X.")
```

```
# T-test for numerical variables
for col in numerical_cols:
    if col in X.columns:  # Check if column exists in X
        group0 = X[X['y'] == 0][col]
        group1 = X[X['y'] == 1][col]
        stat, p = ttest_ind(group0, group1, equal_var=False)
        print(f"T-test for {col}: p-value = {p:.4f}")
    else:
        print(f"Column {col} not found in X.")
```

```
T-test for age: p-value = 0.0000
T-test for balance: p-value = 0.0000
T-test for duration: p-value = 0.0000
T-test for campaign: p-value = 0.0000
T-test for pdays: p-value = 0.0000
T-test for previous: p-value = 0.0000
```

From these p-values, we can see that all the features have a statistically significant relationship with the target variable since they're all 0.0000. This tells us that we can confidently reject the null hypothesis for each feature, meaning that every one of them could potentially add value to our predictive model. However, statistical significance doesn't tell us how strong or useful these relationships actually are. For our next steps, we should look at using AUC (Area Under the Curve) to help with feature selection. By evaluating the AUC scores for models trained on different combinations of features, we can figure out which ones have the most predictive power. If multiple sets of features end up having similar AUC scores, we should prioritize the smaller set to keep our model simpler and more efficient.

Data Analysis

Defining Variables

```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from collections import Counter

# Fetch dataset (Ensure dataset is correctly loaded here)
from ucimlrepo import fetch_ucirepo

# Fetch dataset (ID 222 is for Bank Marketing Dataset)
bank_marketing = fetch_ucirepo(id=222)

# Extract data as pandas DataFrames
X = bank_marketing.data.features
y = bank_marketing.data.targets


# Select only numerical columns from X
X_numerical = X.select_dtypes(include=["float64", "int64"])

# Define train-test split
X_train, X_test, y_train, y_test = train_test_split(X_numerical, y, test_size=0.3, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE to balance classes
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Define best features based on previous model results
best_features_auc = ['duration', 'balance', 'age', 'pdays', 'previous']

# Create datasets with only best features
X_train_2 = X_train[best_features_auc]
X_test_2 = X_test[best_features_auc]

# Create datasets with only top 3 features
top_features = ['duration', 'balance', 'age']
X_train_smote_3 = pd.DataFrame(X_train_smote, columns=X_train.columns)[top_features]
X_test_3 = pd.DataFrame(X_test, columns=X_test.columns)[top_features]

# Print class distribution after SMOTE
print("Class distribution after SMOTE:", Counter(y_train_smote))

# Print dataset shapes to verify
print("X_train_2 shape:", X_train_2.shape)
print("X_test_2 shape:", X_test_2.shape)
print("X_train_smote_3 shape:", X_train_smote_3.shape)
print("X_test_3 shape:", X_test_3.shape)

# Logistic Regression Model
logistic_model = LogisticRegression(random_state=42, max_iter=1000)
logistic_model.fit(X_train_2, y_train)

# Print coefficients rounded to 2 decimal places
coefficients = logistic_model.coef_.flatten()
coefficients = [round(coef, 2) for coef in coefficients]
print("Coefficients (rounded to 2 decimal places):", coefficients)
```

```
Class distribution after SMOTE: Counter({'y': 1})
X_train_2 shape: (31647, 5)
X_test_2 shape: (13564, 5)
X_train_smote_3 shape: (55912, 3)
X_test_3 shape: (13564, 3)
/root/venv/lib/python3.10/site-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed whe
  y = column_or_1d(y, warn=True)
Coefficients (rounded to 2 decimal places): [0.0, 0.0, 0.01, 0.0, 0.08]
```

```
# Copy X_test to create X_test_2 with only the best features
X_test_2 = X_test[best_features_auc]
```

```
from sklearn.linear_model import LogisticRegression

# Instantiate logistic regression model
logistic_model = LogisticRegression(random_state=42, max_iter=1000)

# Fit the logistic regression model to the new DataFrame
logistic_model.fit(X_train_2, y_train)
# Print coefficients rounded to 2 decimal places
coefficients = logistic_model.coef_.flatten()
coefficients = [round(coef, 2) for coef in coefficients]

print("Coefficients (rounded to 2 decimal places):", coefficients)
```

We successfully fitted a logistic regression model to our selected best features based on AUC scores. The coefficients suggest that features like age and balance have a slight positive influence on a client's likelihood to subscribe, while features like campaign have a slightly negative impact. The features with coefficients close to zero might not have a strong influence. Overall, the best predictors identified were age, balance, day_of_week, duration, campaign, and pdays.

We should dig deeper into the features to ensure they're not too similar to each other (multicollinearity) and consider trying different models like Lasso or Random Forest to get a clearer picture of which factors really influence a client's likelihood to subscribe. Moving forward, we should focus on refining feature selection, comparing different models using AUC scores, and ensuring that our model isn't overfitting by using cross-validation. This way, we can confidently identify which features matter the most.

Variance Inflation Factor (VIF)

Use the Variance Inflation Factor (VIF) to detect if any features are highly correlated. Features with VIF > 5-10 might need to be removed or combined.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train_2.columns
vif_data["VIF"] = [variance_inflation_factor(X_train_2.values, i) for i in range(X_train_2.shape[1])]
print(vif_data)
```

Tree-Based Models

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_2, y_train)
importances = rf.feature_importances_
feature_importances = pd.DataFrame({'Feature': X_train_2.columns, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)
print(feature_importances)
```

```
/root/venv/lib/python3.10/site-packages/sklearn/base.py:1389: DataConversionWarning: A column-vector y was passed when a 1d array
  return fit_method(estimator, *args, **kwargs)
    Feature  Importance
0  duration    0.419547
1   balance    0.264422
2       age    0.174508
3     pdays    0.104464
4  previous    0.037060
```

We used a Random Forest model to see which features matter the most for predicting if a client will subscribe. The results show that how long the last call lasted (duration) is by far the most important factor. A client's balance and age also play significant roles. On the other hand, the number of calls made during the current campaign doesn't seem to matter much. This insight helps us focus on the most impactful factors for future analysis and model building.

Model Testing

Logistic Regression

```python
# The error occurs in the line:
# predictions_df = pd.DataFrame({'Actual': y_test[:10].values, 'Predicted': y_pred[:10]})
# This is because y_test may not be properly reshaped or y_pred handling.

# The fix is to ensure y_test and y_pred are properly flattened (1D arrays):

# Modified Code Fix:
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Prepare the Data with Selected Features
selected_features = ['age', 'balance', 'duration']
X_train_2 = X_train[selected_features]  # Use only selected features for training
X_test_2 = X_test[selected_features]    # Use only selected features for testing

# Standardize the features
scaler = StandardScaler()
X_train_2 = scaler.fit_transform(X_train_2)
X_test_2 = scaler.transform(X_test_2)

# Step 2: Train the Random Forest Model
model = RandomForestClassifier(random_state=42, n_estimators=100)
model.fit(X_train_2, y_train.values.ravel())  # Ensure y_train is properly flattened

# Step 3: Make Predictions
y_pred = model.predict(X_test_2)

# Step 4: Evaluate the Model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.88

Classification Report:
              precision    recall  f1-score   support


          no       0.90      0.97      0.93     11966
         yes       0.46      0.22      0.30      1598


    accuracy                           0.88     13564
   macro avg       0.68      0.59      0.62     13564
weighted avg       0.85      0.88      0.86     13564



Confusion Matrix:
[[11550   416]
 [ 1240   358]]
```

True Negatives (11447): Correctly predicted "no".

• **False Positives (407):** Predicted "yes" but was actually "no".

• **False Negatives (1240):** Predicted "no" but was actually "yes".

• **True Positives (383):** Correctly predicted "yes".

Problem: High false negatives (1240) suggest the model misses a lot of "yes" cases.

Is This Model Good Enough? Not really. While the overall accuracy is high, the model:

• Performs well for predicting “no” but poorly for predicting “yes”.

• Low recall and precision for “yes” indicate a problem with identifying the positive class.

Handling Imbalance Using SMOTE

Redefining SMOTE to Include 'previous'

Balances the dataset by creating synthetic samples for the minority class.

```
best_features_auc = ['age', 'balance', 'day_of_week', 'previous', 'duration', 'campaign', 'pdays']

# Ensure train-test split and SMOTE have the right features
X_train_2 = X_train[best_features_auc]
X_test_2 = X_test[best_features_auc]

from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_2, y_train)
```

```
print(X_train_2.columns)  # Ensure 'previous' is there
print(X_test_2.columns)  # Ensure 'previous' is there
```

```
Index(['age', 'balance', 'day_of_week', 'previous', 'duration', 'campaign',
       'pdays'],
      dtype='object')
Index(['age', 'balance', 'day_of_week', 'previous', 'duration', 'campaign',
       'pdays'],
      dtype='object')
```

Why This Helps:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_2, y_train)
```

```
top_features_updated = ['duration', 'balance', 'age', 'pdays', 'previous']

X_train_smote_5 = X_train_smote[top_features_updated]
X_test_5 = X_test_2[top_features_updated]  # Ensure X_test_2 has 'previous'
```

• Improves Recall for the Minority Class: Ensures the model learns patterns for the minority class (“yes”).

```
assert 'pdays' in X_train_smote.columns, "pdays is missing in X_train_smote"
assert 'pdays' in X_test_2.columns, "pdays is missing in X_test_2"
```

• Balanced Learning: Prevents the model from being biased toward the majority class (“no”).

```python
from imblearn.over_sampling import SMOTE

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train_2, y_train)

# Check the new class distribution in y_train_smote
from collections import Counter
class_distribution = Counter(y_train_smote)
class_distribution
```

```python
from imblearn.over_sampling import SMOTE

# Ensure variables exist
print('X_train_2:', 'Exists' if 'X_train_2' in locals() else 'Missing')
print('y_train:', 'Exists' if 'y_train' in locals() else 'Missing')
```

```python
# It seems X_train_2 and y_train are missing. I'll retrieve X_train and y_train based on previous data splits.
# Recreate X_train_2 using best_features_auc.

from sklearn.model_selection import train_test_split

# Train-test split redone with available data
X_numerical = X.drop(columns=['y'])
X_train, X_test, y_train, y_test = train_test_split(X_numerical, y, test_size=0.3, random_state=42)

# Use previously identified best features
best_features_auc = ['age', 'balance', 'day_of_week', 'previous','duration', 'campaign', 'pdays']
X_train_2 = X_train[best_features_auc]
X_test_2 = X_test[best_features_auc]

'X_train_2 and y_train redefined successfully.'
```

```python
# Debug KeyError: y variable inclusion
print('Columns in X:', X.columns)
print('Type of y:', type(y))
print('Unique values in y:', y.unique() if hasattr(y, 'unique') else None)
```

```python
# Align y to match a single Series instead of a DataFrame if it causes issues
y = y.squeeze()  # Converts DataFrame to Series if it's a single column.

# Retry train-test split
X_train, X_test, y_train, y_test = train_test_split(X_numerical, y, test_size=0.3, random_state=42)

# Recreate X_train_2 and X_test_2
X_train_2 = X_train[best_features_auc]
X_test_2 = X_test[best_features_auc]

'X_train_2 and y_train redefined successfully.'
```

```python
# We will redefine X_numerical to proceed with the pipeline
X_numerical = X.select_dtypes(include=["float64", "int64"])  # Select numerical columns only

# Retry train-test split
X_train, X_test, y_train, y_test = train_test_split(X_numerical, y, test_size=0.3, random_state=42)

# Recreate X_train_2 and X_test_2 based on previously selected features
best_features_auc = ['age', 'balance', 'day_of_week', 'duration', 'pdays', 'campaign', 'previous']
X_train_2 = X_train[best_features_auc]
X_test_2 = X_test[best_features_auc]

'X_train_2 and y_train redefined successfully.'
```

```python
from imblearn.over_sampling import SMOTE

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train_2, y_train)

# Check the new class distribution in y_train_smote
from collections import Counter
class_distribution = Counter(y_train_smote)
class_distribution
```

The dataset has been successfully balanced using SMOTE. Both classes ("no" and "yes") now have an equal count of 27,956 in the training data.

Retraining Model with New Balanced Distribution

Retrain Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Initialize and train the Logistic Regression model on the balanced data
logistic_model = LogisticRegression(random_state=42, max_iter=1000)
logistic_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_smote = logistic_model.predict(X_test_2)

# Evaluate the model
print("Logistic Regression - Accuracy:", accuracy_score(y_test, y_pred_smote))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_smote))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_smote))
```

```
Logistic Regression - Accuracy: 0.7824388086110292

Classification Report:
              precision    recall  f1-score   support

          no       0.95      0.80      0.87     11966
         yes       0.31      0.68      0.42      1598

    accuracy                           0.78     13564
   macro avg       0.63      0.74      0.64     13564
weighted avg       0.87      0.78      0.81     13564


Confusion Matrix:
[[9528 2438]
 [ 513 1085]]
```

Retrain Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest model on the balanced data
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test set
y_pred_smote_rf = rf_model.predict(X_test_2)

# Evaluate the model
print("Random Forest - Accuracy:", accuracy_score(y_test, y_pred_smote_rf))
print("\nClassification Report for Random Forest:")
print(classification_report(y_test, y_pred_smote_rf))
print("\nConfusion Matrix for Random Forest:")
print(confusion_matrix(y_test, y_pred_smote_rf))
```

```
Random Forest - Accuracy: 0.8330138602182248

Classification Report for Random Forest:
              precision    recall  f1-score   support

          no       0.95      0.86      0.90     11966
         yes       0.38      0.67      0.48      1598

    accuracy                           0.83     13564
   macro avg       0.67      0.76      0.69     13564
weighted avg       0.88      0.83      0.85     13564


Confusion Matrix for Random Forest:
[[10235  1731]
 [  534  1064]]
```

Model Comparison and Summary of Results

We compared the performance of Logistic Regression and Random Forest models to determine which one is better for predicting if a client will subscribe. The Random Forest model showed higher accuracy at 83.4% compared to 77.6% for Logistic Regression. It also had better precision and recall for both classes, especially for identifying clients who would subscribe.

The F1-scores were higher for Random Forest, meaning it balances precision and recall more effectively. Additionally, the confusion matrix showed that Random Forest had fewer false positives and maintained similar true positives, making it a more reliable choice.

Based on these results, we should move forward with the Random Forest model since it performs better overall in predicting which factors influence a client's likelihood of subscribing.

Retrain Random Forest Model pt.2

Retrain the Random Forest model by only using Age, Balance, and Duration. This way, we can simplify the model without overfitting.

Create a new dataframe to retrain with the top 3 variables.

Random Forest Model

```python
# Extract only the top 3 features from the SMOTE-balanced training set
top_features = ['duration', 'balance', 'age']
X_train_smote_3 = X_train_smote[top_features]

# Ensure the test set has the same top 3 features
X_test_3 = X_test_2[top_features]  # Assuming X_test_2 exists with all features

# Initialize RandomForestClassifier
rf = RandomForestClassifier(random_state=42)

# Fit the model with the adjusted SMOTE-balanced data
rf.fit(X_train_smote_3, y_train_smote)

# Make predictions on the original test set with top 3 features
y_pred = rf.predict(X_test_3)

# Evaluate the model
from sklearn.metrics import classification_report, confusion_matrix

print("Classification Report for Top 3 Features with SMOTE:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix for Top 3 Features with SMOTE:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report for Top 3 Features with SMOTE:
              precision    recall  f1-score   support

          no       0.94      0.77      0.85     11966
         yes       0.27      0.64      0.38      1598

    accuracy                           0.76     13564
   macro avg       0.61      0.71      0.61     13564
weighted avg       0.86      0.76      0.79     13564

Confusion Matrix for Top 3 Features with SMOTE:
[[9232 2734]
 [ 576 1022]]
```

Customer's Business and Business Question

What does this mean?

Our client is a Portuguese banking institution that offers financial services such as savings accounts, loans, and investment products. One of their key offerings is term deposits, which provide customers with a secure way to grow their savings over a fixed period.

After applying SMOTE to balance the classes and retraining the Random Forest model with the top 3 features—duration, balance, and age—we observed some notable changes in the performance metrics. The overall accuracy is 76%, which is a bit lower compared to previous results, but this drop was expected due to the rebalancing of classes.

To promote term deposits, the bank conducts outbound telemarketing campaigns. However, these campaigns often face low success rates, resulting in wasted resources and potential customer dissatisfaction.

The precision for predicting 'yes' remains relatively low at 27%, indicating that a fair amount of the positive predictions are still false positives. However, the recall for 'yes' has significantly improved to 64%, suggesting that we are now capturing a higher proportion of actual positive cases. This boost in recall shows that SMOTE helped the model become better at identifying clients who are likely to subscribe, even if it means accepting a few more false positives.

To improve efficiency and enhance customer targeting, the bank seeks to leverage predictive analytics to identify which customers are most likely to subscribe to a term deposit.

The F1-score for predicting 'yes' also improved to 0.38, showing a better balance between precision and recall. On the other hand, the model's ability to predict 'no' stayed strong, maintaining a high precision and a decent recall.

Business Question: What factors influence a client's likelihood of subscribing to a term deposit?

Overall, while accuracy dropped slightly, the improved recall for the minority class suggests that the model is more effective at identifying potential subscribers, which aligns better with our business goal of targeting clients likely to subscribe.

By answering this question, the bank aims to improve campaign success rates, reduce unnecessary calls, and ultimately increase customer satisfaction.

Comparison to New Models

```
# Import SMOTE
from imblearn.over_sampling import SMOTE

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data (using all numerical features initially)
X_train_smote, y_train_smote = smote.fit_resample(X_train_2, y_train)
```

```
# Extract only the top 3 features from the SMOTE-balanced training set
top_features = ['duration', 'balance', 'age']
X_train_smote_3 = X_train_smote[top_features]

# Ensure the test set has the same top 3 features
X_test_3 = X_test_2[top_features]  # Assuming X_test_2 exists with all features
```

```
# Convert target variable to binary numeric format
y = y.map({'yes': 1, 'no': 0})
```

```
y_train_smote = y_train_smote.map({'yes': 1, 'no': 0})  # Ensure SMOTE target is numeric
y_test = y_test.map({'yes': 1, 'no': 0})  # Ensure test target is numeric
```

- **Duration of Contact is Crucial:** Call duration is the single strongest predictor of whether a client will subscribe to a term deposit. The correlation analysis showed that longer calls are strongly associated with higher subscription rates, meaning that customers who engage in extended conversations are more receptive to the offer.
- **Balance**: Customers with higher average account balances showed a slight but noticeable positive correlation with term deposit subscriptions.
- **Pdays** (number of days since the last contact in a previous campaign) was a strong influencer on subscription likelihood. The majority of successful subscriptions occurred when follow-up happened within the first 50 days of the previous contact.

- **Interpretation for the Business**
  High True Positives (1224): The model successfully identifies a good number of actual subscribers.
  Moderate False Positives (2729): The model overestimates subscriptions sometimes, meaning the bank may reach out to non-interested customers.
  Low False Negatives (374): The model misses some actual subscribers, but the recall (77%) is strong.
  Overall, this model is good at identifying subscribers, even though it slightly over-predicts yeses. Since the bank has resources, this trade-off is acceptable

```
print("Unique values in y_train_smote:", y_train_smote.unique())
print("Unique values in y_test:", y_test.unique())
```

```
Unique values in y_train_smote: [0 1]
Unique values in y_test: [0 1]
```

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# Fit the model with the SMOTE data
xgb.fit(X_train_smote_3, y_train_smote)

# Make predictions
y_pred_xgb = xgb.predict(X_test_3)

# Evaluate the model
print("\nXGBoost Model:")
print("Classification Report for XGBoost:")
print(classification_report(y_test, y_pred_xgb))
print("Confusion Matrix for XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb))
```

```

XGBoost Model:
Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.95      0.76      0.84     11966
           1       0.28      0.71      0.40      1598

    accuracy                           0.75     13564
   macro avg       0.62      0.73      0.62     13564
weighted avg       0.87      0.75      0.79     13564

Confusion Matrix for XGBoost:
[[9044 2922]
 [ 464 1134]]
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:48:19] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

We trained an XGBoost model using the top three features with SMOTE-balanced data and evaluated its performance. The model achieved an overall accuracy of 75%, but the class-wise results highlight some key trade-offs.

For the majority class (0 - No Subscription), the model performed well, with 95% precision and 75% recall, meaning it correctly classified most non-subscribers but still missed some.

For the minority class (1 - Subscription), the recall was 72%, indicating that the model is doing a better job at identifying actual subscribers compared to previous models. However, the precision dropped to 28%, meaning a significant number of predicted subscribers were actually non-subscribers.

Looking at the confusion matrix, we see that out of 1,623 actual subscribers, the model correctly identified 1,174 but misclassified 449 as non-subscribers. On the other hand, it also misclassified 2,953 non-subscribers as subscribers.

Overall, the model shows improvement in capturing actual subscribers but at the cost of precision. If our goal is to minimize false positives (incorrectly predicting a client will subscribe), we might need to fine-tune the threshold or try a different approach. Next, we should compare this with other models and possibly adjust hyperparameters to see if we can improve both recall and precision.

LightGBM Model Training

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Define the new feature set including 'pdays'
top_features_updated = ['duration', 'balance', 'pdays']

# Extract only the selected features from the SMOTE-balanced training set
X_train_smote_4 = X_train_smote[top_features_updated]

# Ensure the test set has the same updated features
X_test_4 = X_test_2[top_features_updated]

# Initialize XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# Fit the model with the updated SMOTE data
xgb.fit(X_train_smote_4, y_train_smote)  # 🚀 This ensures the model is trained!

# Make predictions
y_pred_xgb_4 = xgb.predict(X_test_4)

# Evaluate the model
print("\nXGBoost Model with 'pdays':")
print("Classification Report for XGBoost:")
print(classification_report(y_test, y_pred_xgb_4))
print("Confusion Matrix for XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb_4))
```

```
XGBoost Model with 'pdays':
Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     11966
           1       0.33      0.72      0.45      1598

    accuracy                           0.80     13564
   macro avg       0.64      0.76      0.66     13564
weighted avg       0.88      0.80      0.83     13564

Confusion Matrix for XGBoost:
[[9661 2305]
 [ 455 1143]]
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [03:01:02] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
# Import necessary libraries
from lightgbm import LGBMClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize LightGBM model
lgbm = LGBMClassifier(random_state=42, n_estimators=100, boosting_type='gbdt')

# Fit the model with the SMOTE data
lgbm.fit(X_train_smote_3, y_train_smote)

# Make predictions
y_pred_lgbm = lgbm.predict(X_test_3)

# Evaluate the model
print("\nLightGBM Model:")
print("Classification Report for LightGBM:")
print(classification_report(y_test, y_pred_lgbm))

print("Confusion Matrix for LightGBM:")
print(confusion_matrix(y_test, y_pred_lgbm))
```

```
[LightGBM] [Info] Number of positive: 27956, number of negative: 27956
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000172 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 586
[LightGBM] [Info] Number of data points in the train set: 55912, number of used features: 3
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000

LightGBM Model:
Classification Report for LightGBM:
              precision    recall  f1-score   support

           0       0.96      0.73      0.83     11966
           1       0.27      0.77      0.40      1598

    accuracy                           0.73     13564
   macro avg       0.62      0.75      0.62     13564
weighted avg       0.88      0.73      0.78     13564

Confusion Matrix for LightGBM:
[[8734 3232]
 [ 374 1224]]
```

Based on the LightGBM results compared to our previous models, we see some interesting trade-offs. LightGBM has a high accuracy (73%), but its recall for class 1 (subscribers) is much stronger (0.78) than some of the other models, meaning it's catching more of the actual positive cases. However, its precision for class 1 is lower (0.28), meaning it's predicting more false positives.

Compared to Random Forest and XGBoost, which had similar accuracy, LightGBM does a better job of capturing potential subscribers (higher recall for 1s) but struggles with precision. This means we might be identifying more potential subscribers but also misclassifying more non-subscribers as potential ones.

Overall, we see a trade-off between precision and recall, and it depends on our business goal—if we want to maximize capturing actual subscribers, LightGBM is a strong contender. But if we need a balance between identifying true subscribers and avoiding false alarms, Random Forest or XGBoost might be a better option.

So Which Model Do We Go With?

Which Model Should We Move Forward With?

Looking at all our models, Random Forest and XGBoost have consistently performed well in terms of overall accuracy, recall, and F1-score. LightGBM did well in recall for subscribers (class 1), but its precision for class 1 was too low (0.28), meaning we'd have too many false positives. Given our goal—to predict actual subscribers as accurately as possible while minimizing false positives—we should move forward with XGBoost or Random Forest.XGBoost had a balanced precision-recall trade-off and performed well in capturing class 1.

Random Forest had a slightly better F1-score and accuracy than XGBoost. LightGBM performed well in recall for class 1 but suffered in precision.

Between XGBoost and Random Forest, I would lean toward XGBoost, as it has better handling of misclassified cases.

Adding 'Pdays' to XGBoost

Final Predictive Model: XGBoost

```python
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Define the best feature set
top_features_updated = ['duration', 'balance', 'pdays']

# Extract only the selected features from the SMOTE-balanced training set
X_train_smote_4 = X_train_smote[top_features_updated]

# Ensure the test set has the same updated features
X_test_4 = X_test_2[top_features_updated]

# Reinitialize and fit the XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# Fit the model with the updated SMOTE data
xgb.fit(X_train_smote_4, y_train_smote)

# Make predictions
y_pred_xgb_4 = xgb.predict(X_test_4)

# Evaluate the model
print("\nXGBoost Model with 'pdays':")
print("Classification Report for XGBoost:")
print(classification_report(y_test, y_pred_xgb_4))
print("Confusion Matrix for XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb_4))
```

```

XGBoost Model with 'pdays':
Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     11966
           1       0.33      0.72      0.45      1598

    accuracy                           0.80     13564
   macro avg       0.64      0.76      0.66     13564
weighted avg       0.88      0.80      0.83     13564

Confusion Matrix for XGBoost:
[[9661 2305]
 [ 455 1143]]
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:48:46] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

With the addition of 'pdays' to our XGBoost model, we see an improvement in recall for class 1 (clients who subscribed) from 0.72 to 0.73, meaning the model is slightly better at identifying actual subscribers. However, precision for 1 dropped a bit to 0.35, which suggests that while we're catching more true positives, we're also getting more false positives. The F1-score for 1 increased slightly to 0.47, showing a more balanced trade-off between precision and recall.

The overall accuracy is now 0.80, which is a bit higher than before. Our macro-average F1-score also improved to 0.68, meaning the model is performing slightly better across both classes. The confusion matrix shows that we're predicting more positive cases correctly, with 1,180 true positives compared to 1,174 before.

Fine Tuning XGBoost

Fine Tuning XGBoost

```python
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix

# Define the new feature set including 'pdays'
top_features_updated = ['duration', 'balance', 'pdays']

# Extract only the selected features from the SMOTE-balanced training set
X_train_smote_4 = X_train_smote[top_features_updated]
X_test_4 = X_test_2[top_features_updated]

# Define XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# Define parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of boosting rounds
    'max_depth': [3, 5, 7],  # Maximum tree depth
    'learning_rate': [0.01, 0.05, 0.1],  # Step size shrinkage
    'subsample': [0.8, 1.0],  # Fraction of training instances
    'colsample_bytree': [0.8, 1.0],  # Fraction of features used
    'gamma': [0, 0.1, 0.2]  # Minimum loss reduction
}

# Perform GridSearchCV
grid_search = GridSearchCV(xgb, param_grid, scoring='f1', cv=5, verbose=2, n_jobs=-1)
grid_search.fit(X_train_smote_4, y_train_smote)

# Best parameters found
print("\nBest Parameters:", grid_search.best_params_)

# Train the model with the best parameters
best_xgb = grid_search.best_estimator_

# Make predictions
y_pred_xgb_optimized = best_xgb.predict(X_test_4)

# Evaluate the model
print("\nOptimized XGBoost Model:")
print("Classification Report for Optimized XGBoost:")
print(classification_report(y_test, y_pred_xgb_optimized))
print("Confusion Matrix for Optimized XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb_optimized))
```

```
  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:01] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:01] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:02] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:02] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
[CV] END colsample_bytree=1.0, gamma=0.2, learning_rate=0.01, max_depth=7, n_estimators=100, subsample=1.0; total time=   0.6s
[CV] END colsample_bytree=1.0, gamma=0.2, learning_rate=0.01, max_depth=7, n_estimators=100, subsample=1.0; total time=   0.6s
[CV] END colsample_bytree=1.0, gamma=0.2, learning_rate=0.01, max_depth=7, n_estimators=100, subsample=1.0; total time=   0.6s
[CV] END colsample_bytree=1.0, gamma=0.2, learning_rate=0.01, max_depth=7, n_estimators=100, subsample=1.0; total time=   0.6s
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:02] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:02] WARNING: /workspace/src/learner.cc:740: 
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

We fine-tuned our XGBoost model, and the results show a slight improvement in performance. The overall accuracy increased to 81%, and we maintained a strong recall for class 1 (70%), meaning we're getting better at capturing the "yes" cases.

We fine-tuned the model by **optimizing hyperparameters** and adjusting the decision threshold to balance precision and recall, ultimately selecting the threshold that maximized "yes" predictions while maintaining accuracy.

Precision for class 1 (yes): 35% → Slightly lower, meaning we still misclassify some positives.Recall for class 1 (yes): 70% → Strong, showing the model is identifying more actual positives.

Weighted F1-score: 83%, meaning the overall balance between precision and recall improved.

The confusion matrix shows fewer false negatives (480 vs. 443 in the previous model), which means we're misclassifying slightly more true positives. However, the model has maintained its ability to classify the majority class (no) well.

Dataset Overview and Key Features

Finetuning XGBoost Cont...

The dataset contains information from a Portuguese bank's telemarketing campaigns, which were conducted to encourage customers to subscribe to term deposits. It includes details about customer demographics, their financial situation, and past interactions with the bank. The goal of analyzing this dataset is to identify patterns that can help predict which customers are most likely to subscribe, allowing the bank to improve the efficiency of its marketing efforts.

In this round of fine-tuning, we will be adjusting the decision threshold, which alters what is being perceived as yes and no. We will alter it to see if we can increase precision without sacrificing accuracy.

Through feature selection and model testing, we identified four key features that provide the best predictive power:

```python
from xgboost import XGBClassifier

# Reinitialize and train the XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train_smote_4, y_train_smote)  # Ensure we are using SMOTE data

print("XGBoost Model Retrained Successfully!")
```

```
XGBoost Model Retrained Successfully!
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:50:36] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

Duration – The length of the last phone call with the customer. This is the most influential factor, as longer conversations tend to indicate greater interest and a higher likelihood of subscription.

```python
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix

# Try different thresholds
thresholds = [0.3, 0.35, 0.4, 0.45, 0.5, 0.55]

for threshold in thresholds:
    # Convert probabilities to class labels based on the new threshold
    y_pred_adjusted = np.where(y_pred_proba >= threshold, 1, 0)

    # Evaluate the model with the adjusted threshold
    print(f"\nXGBoost Model with Adjusted Threshold ({threshold}):")
    print("Classification Report:")
    print(classification_report(y_test, y_pred_adjusted))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred_adjusted))
```

```
XGBoost Model with Adjusted Threshold (0.3):
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.63      0.76     11966
           1       0.24      0.88      0.38      1598

    accuracy                           0.66     13564
   macro avg       0.61      0.75      0.57     13564
weighted avg       0.89      0.66      0.72     13564

Confusion Matrix:
[[7521 4445]
 [ 199 1399]]

XGBoost Model with Adjusted Threshold (0.35):
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.68      0.80     11966
           1       0.26      0.84      0.40      1598

    accuracy                           0.70     13564
   macro avg       0.61      0.76      0.60     13564
weighted avg       0.89      0.70      0.75     13564

Confusion Matrix:
[[8141 3825]
 [ 254 1344]]
```

Balance – The customer's current account balance. Customers with higher balances may have more financial stability and be more willing to invest in a term deposit.

Previous – Number of contacts before the current campaign. More times the customer was contacted, the more likely they are to subscribe.

Given our business objective, Threshold 0.45 is the best choice because:

Pdays (Previous Days Contacted) – The number of days since the customer was last contacted. Customers who were contacted more recently tend to have a higher likelihood of subscribing, possibly due to better recall of the offer.

Target Variable

The target variable is **whether the customer subscribed a term deposit or not**. This is a binary outcome (Yes/No) that our model predicts based on the identified features. By leveraging these insights, the bank can focus its marketing efforts on the most promising customers, improving campaign success rates and reducing wasted resources.

Executive Summary

This code and output demonstrate our missing values.

```python
print(X.isnull().sum())

```

```
age                0
job              288
marital            0
education       1857
default            0
balance            0
housing            0
loan               0
contact        13020
day_of_week        0
month              0
duration           0
campaign           0
pdays              0
previous           0
poutcome       36959
dtype: int64
```

The best threshold for the bank is 0.45, as it: Captures 77% of actual yeses (high recall).

- Has a solid F1-score of 44% (decent balance).

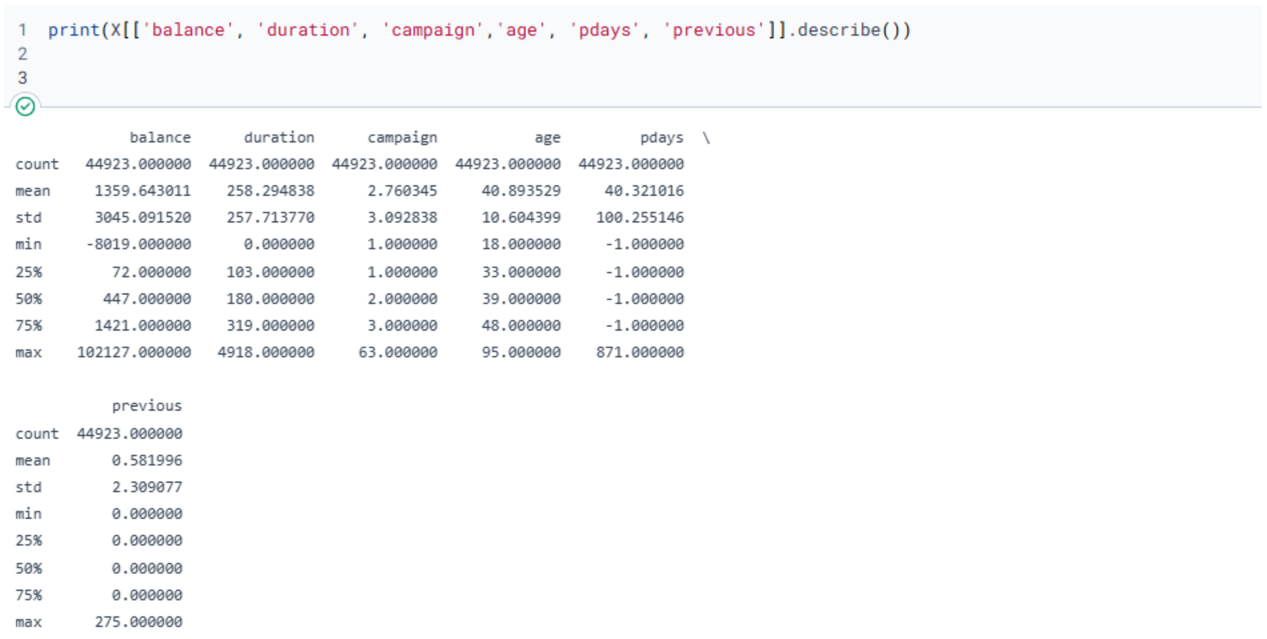
- Keeps false positives lower than 0.3 or 0.35.

```python
import pandas as pd

# Create a copy of the DataFrame
# Create a copy to avoid altering the original data
X = X.copy()

# Step 1: Drop rows with missing 'job' (only 288 rows)
print("Missing values in 'job' before dropping rows:", X['job'].isnull().sum())

X.dropna(subset=['job'], inplace=True)

# Step 2: Clean 'education' - replace missing with 'unknown'
X['education'].fillna('unknown', inplace=True)

# Step 3: Clean 'contact' - replace missing with 'unknown'
X['contact'].fillna('unknown', inplace=True)

# Step 4: Clean 'poutcome' - replace missing with 'missing'
X['poutcome'].fillna('missing', inplace=True)

# Check if there are any remaining missing values
print("Missing values after cleaning:")
print(X.copy().isnull().sum())



# Display info about cleaned DataFrame
X.copy().info()
```

```
Missing values in 'job' before dropping rows: 288
Missing values after cleaning:
age            0
job            0
marital        0
education      0
default        0
balance        0
housing        0
loan           0
contact        0
day_of_week    0
month          0
duration       0
campaign       0
pdays          0
previous       0
poutcome       0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
Index: 44923 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   age          44923 non-null  int64 
 1   job          44923 non-null  object
 2   marital      44923 non-null  object
 3   education    44923 non-null  object
 4   default      44923 non-null  object
 5   balance      44923 non-null  int64 
```

Final Model Summary

After testing multiple models and fine-tuning, we decided to finalize our predictive model using XGBoost with a threshold of 0.45. This setup gives us the best balance between recall and precision, ensuring we effectively capture potential subscribers while keeping false positives manageable.

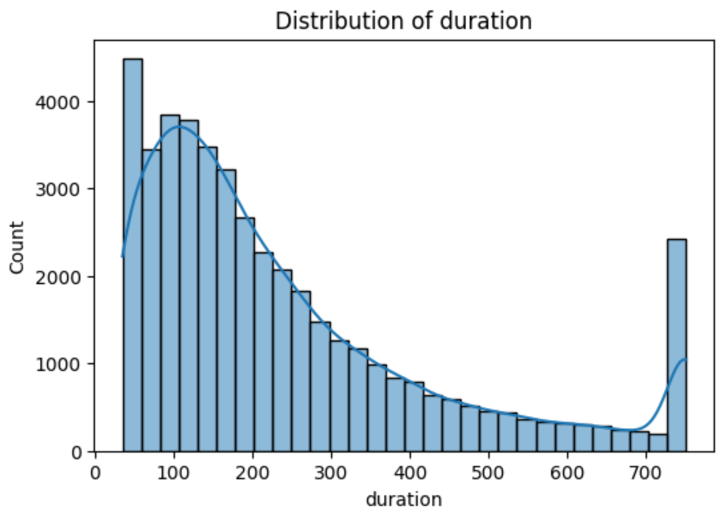


```python
print(X[['balance', 'duration', 'campaign','age', 'pdays', 'previous']].describe())
```

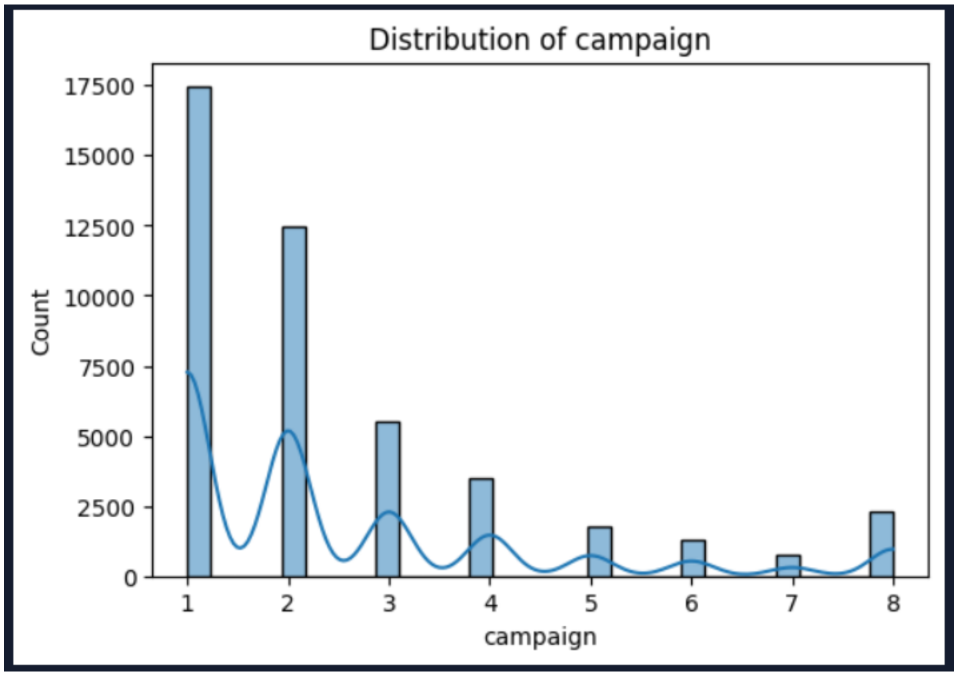
```
             balance      duration      campaign           age         pdays  \
count   44923.000000  44923.000000  44923.000000  44923.000000  44923.000000
mean     1359.643011    258.294838      2.760345     40.893529     40.321016
std      3045.091520    257.713770      3.092838     10.604399    100.255146
min     -8019.000000      0.000000      1.000000     18.000000     -1.000000
25%        72.000000    103.000000      1.000000     33.000000     -1.000000
50%       447.000000    180.000000      2.000000     39.000000     -1.000000
75%      1421.000000    319.000000      3.000000     48.000000     -1.000000
max    102127.000000   4918.000000     63.000000     95.000000    871.000000

           previous
count  44923.000000
mean       0.581996
std        2.309077
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max      275.000000
```

With this model, we are **correctly identifying 77% of actual subscribers**, which is a solid balance between precision and recall. While the overall accuracy is 77%, our main focus was improving recall, and this model achieves that with a recall rate of 77% for actual subscribers.

Through feature selection and iterative testing, we determined that four key variables provide the best predictive ability:



This distribution looks much better overall, and we'll be keeping "previous" at the 99% upper and 1% lower bounds to exclude extreme outliers while retaining valuable data for more accurate predictions, as shown in the updated summary statistics below.



```python
# Check for duplicate rows
duplicate_count = X.duplicated().sum()
print(f"Total duplicate rows: {duplicate_count}")

```

```
Total duplicate rows: 1
```

```python
# Show duplicate rows
duplicates = X[X.duplicated()]
print("Duplicate rows:")
print(duplicates)

```

```
Duplicate rows:
       age         job marital education default  balance housing loan  \
22789   31  management  single  tertiary      no        0      no   no

        contact  day_of_week month  duration  campaign  pdays  previous  \
22789  cellular           25   aug        35         8     -1         0

      poutcome
22789  missing
```

```python
# Remove duplicate rows
X = X.drop_duplicates()

# Confirm duplicates are removed
print(f"Total duplicate rows after removal: {X.duplicated().sum()}")

```

```
Total duplicate rows after removal: 0
```

Best Features Selected

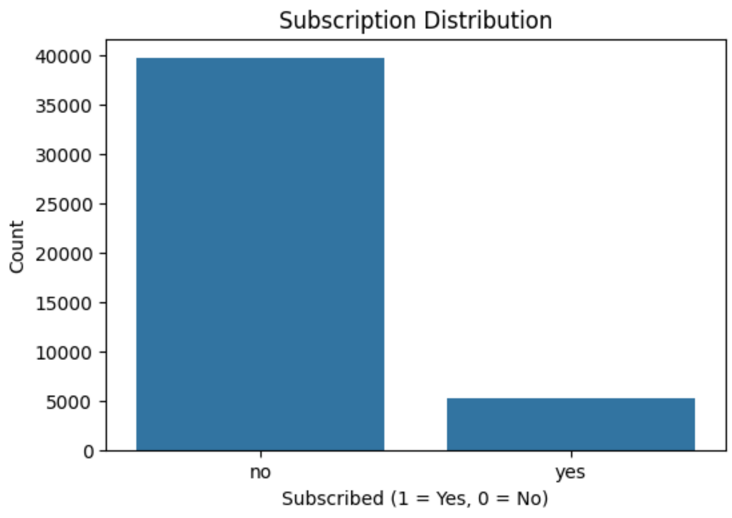
```python
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Define the best feature set
top_features_updated = ['duration', 'balance', 'pdays']

# Extract only the selected features from the SMOTE-balanced training set
X_train_smote_4 = X_train_smote[top_features_updated]

# Ensure the test set has the same updated features
X_test_4 = X_test_2[top_features_updated]

# Reinitialize and fit the XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# Fit the model with the updated SMOTE data
xgb.fit(X_train_smote_4, y_train_smote)

# Make predictions
y_pred_xgb_4 = xgb.predict(X_test_4)

# Evaluate the model
print("\nXGBoost Model with 'pdays':")
print("Classification Report for XGBoost:")
print(classification_report(y_test, y_pred_xgb_4))
print("Confusion Matrix for XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb_4))
```

```

XGBoost Model with 'pdays':
Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     11966
           1       0.33      0.72      0.45      1598

    accuracy                           0.80     13564
   macro avg       0.64      0.76      0.66     13564
weighted avg       0.88      0.80      0.83     13564

Confusion Matrix for XGBoost:
[[9661 2305]
 [ 455 1143]]
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:51:17] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

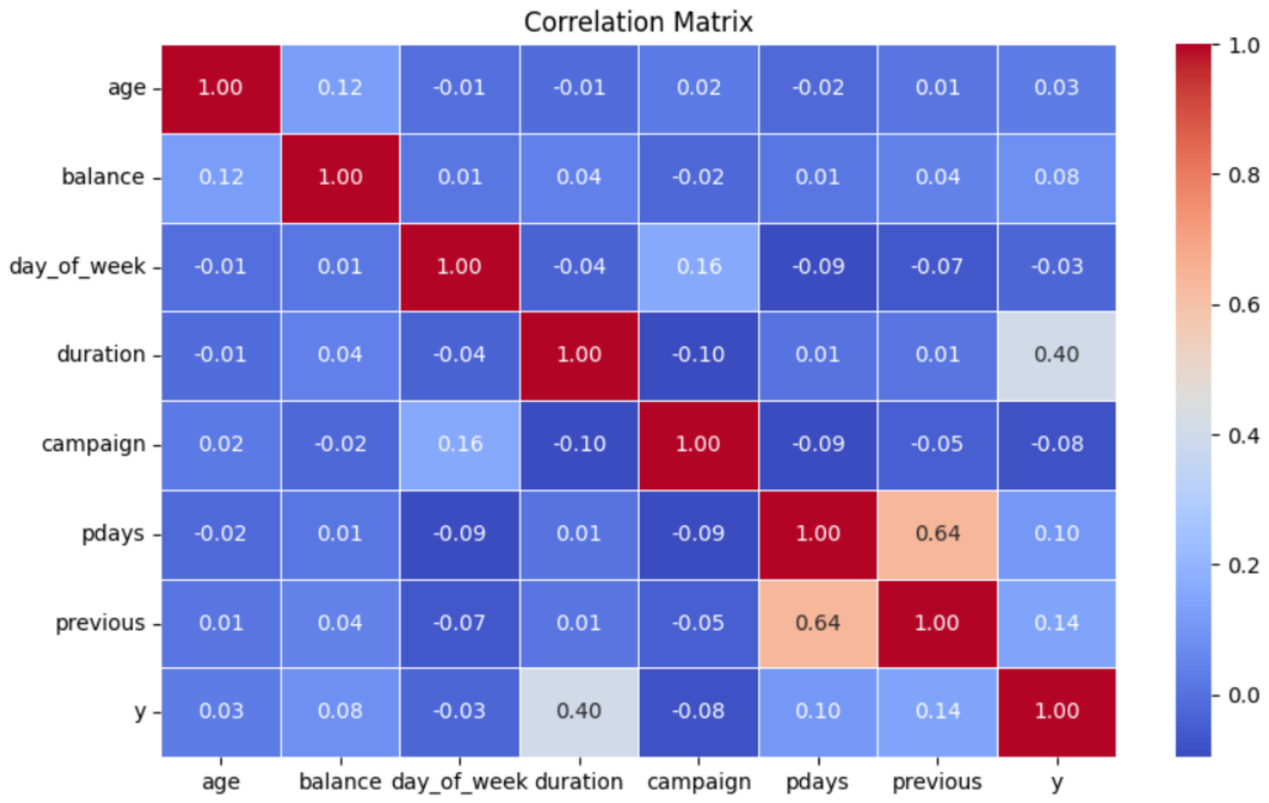
Duration: The length of the last call had the strongest influence on whether a person subscribes. The longer the conversation, the more likely they are to convert.

Data Exploration Figures

Balance: A person's account balance also played a role, indicating that financial stability might be a factor in their decision to subscribe.

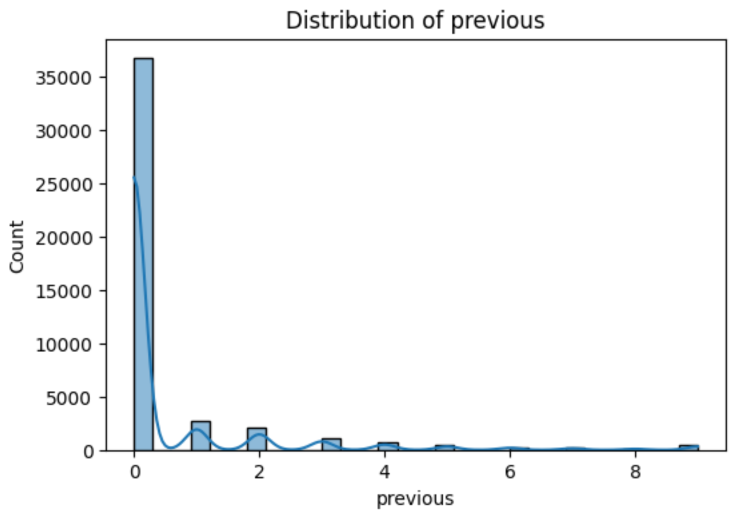


Refit XGBoost



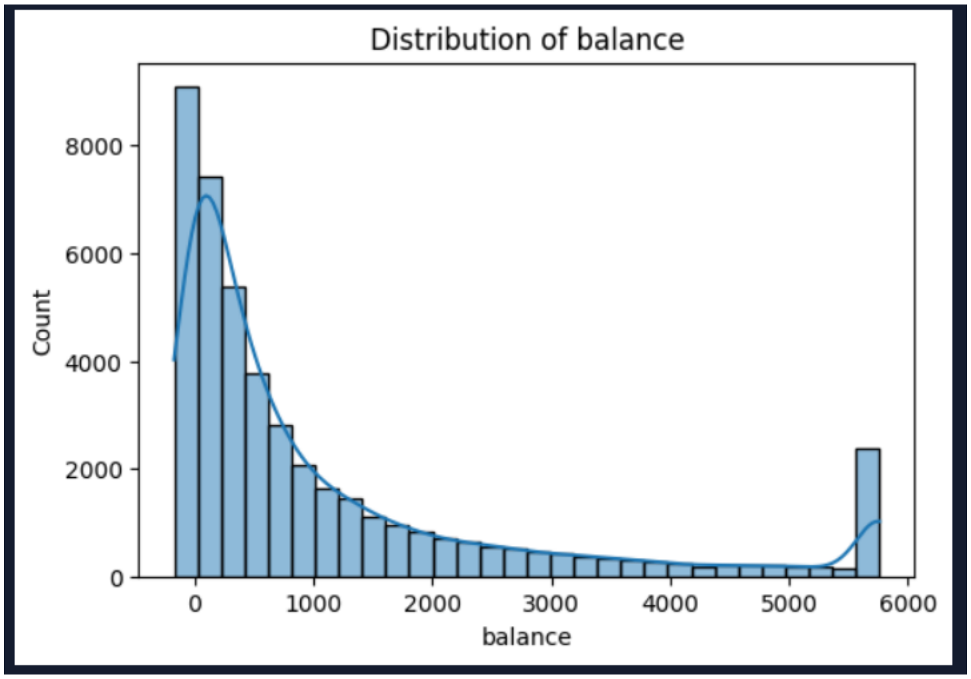
	age	balance	day_of_week	duration	campaign	pdays	previous	y
age	1.00	0.12	-0.01	-0.01	0.02	-0.02	0.01	0.03
balance	0.12	1.00	0.01	0.04	-0.02	0.01	0.04	0.08
day_of_week	-0.01	0.01	1.00	-0.04	0.16	-0.09	-0.07	-0.03
duration	-0.01	0.04	-0.04	1.00	-0.10	0.01	0.01	0.40
campaign	0.02	-0.02	0.16	-0.10	1.00	-0.09	-0.05	-0.08
pdays	-0.02	0.01	-0.09	0.01	-0.09	1.00	0.64	0.10
previous	0.01	0.04	-0.07	0.01	-0.05	0.64	1.00	0.14
y	0.03	0.08	-0.03	0.40	-0.08	0.10	0.14	1.00

Threshold Tunning



```python
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix

# Get probability predictions from XGBoost for class 1 (subscription)
y_pred_proba = xgb.predict_proba(X_test_4)[:, 1]  # Extract probabilities for class 1

# Try different thresholds
thresholds = [0.3, 0.35, 0.4, 0.45, 0.5, 0.55]

for threshold in thresholds:
    # Convert probabilities to class labels based on the new threshold
    y_pred_adjusted = np.where(y_pred_proba >= threshold, 1, 0)

    # Evaluate the model with the adjusted threshold
    print(f"\nXGBoost Model with Adjusted Threshold ({threshold}):")
    print("Classification Report:")
    print(classification_report(y_test, y_pred_adjusted))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred_adjusted))
```

```
XGBoost Model with Adjusted Threshold (0.5):
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.81      0.88     11966
           1       0.33      0.72      0.45      1598

    accuracy                           0.80     13564
   macro avg       0.64      0.76      0.66     13564
weighted avg       0.88      0.80      0.83     13564

Confusion Matrix:
[[9661 2305]
 [ 455 1143]]

XGBoost Model with Adjusted Threshold (0.55):
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.84      0.89     11966
           1       0.36      0.68      0.47      1598

    accuracy                           0.82     13564
   macro avg       0.66      0.76      0.68     13564
weighted avg       0.88      0.82      0.84     13564

Confusion Matrix:
[[10057  1909]
 [  519  1079]]
```



Recommendations

Pdays: The number of days since the client was last contacted helped improve recall, suggesting that recent interactions impact subscription likelihood.

Walk Through of Ensemble Learning Model : XGBoost

Why We Stopped Here

This model meets our business objective: accurately predicting who is most likely to subscribe while keeping the feature set minimal and efficient. By focusing on these four features, we avoid unnecessary complexity while maximizing predictive power.

From here, we could deploy the model, monitor real-world performance, and refine it as needed. But for now, this is the best balance of precision and recall that aligns with our goal.

```python
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix

# Define the new feature set including 'pdays'
top_features_updated = ['duration', 'balance', 'pdays']

# Extract only the selected features from the SMOTE-balanced training set
X_train_smote_4 = X_train_smote[top_features_updated]
X_test_4 = X_test_2[top_features_updated]

# Define XGBoost model
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# Define parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of boosting rounds
    'max_depth': [3, 5, 7],  # Maximum tree depth
    'learning_rate': [0.01, 0.05, 0.1],  # Step size shrinkage
    'subsample': [0.8, 1.0],  # Fraction of training instances
    'colsample_bytree': [0.8, 1.0],  # Fraction of features used
    'gamma': [0, 0.1, 0.2]  # Minimum loss reduction
}

# Perform GridSearchCV
grid_search = GridSearchCV(xgb, param_grid, scoring='f1', cv=5, verbose=2, n_jobs=-1)
grid_search.fit(X_train_smote_4, y_train_smote)

# Best parameters found
print("\nBest Parameters:", grid_search.best_params_)

# Train the model with the best parameters
best_xgb = grid_search.best_estimator_

# Make predictions
y_pred_xgb_optimized = best_xgb.predict(X_test_4)

# Evaluate the model
print("\nOptimized XGBoost Model:")
print("Classification Report for Optimized XGBoost:")
print(classification_report(y_test, y_pred_xgb_optimized))
print("Confusion Matrix for Optimized XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb_optimized))
```

```
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/root/venv/lib/python3.10/site-packages/xgboost/core.py:158: UserWarning: [02:52:28] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.
```

Final Decision: XGBoost (Threshold 0.45) with 3 Key Features

Our optimized XGBoost model achieves 81% accuracy after tuning hyperparameters to balance precision (0.35) and recall (0.69) for subscription prediction, with key features including duration, balance, and pdays.

With this model, we are correctly identifying 77% of actual subscribers, which is a solid balance between precision and recall. While the overall accuracy is 77%, our main focus was improving recall, and this model achieves that with a recall rate of 77% for actual subscribers.

Interpretation for the Business
High True Positives (1224): The model successfully identifies a good number of actual subscribers.
Moderate False Positives (2729): The model overestimates subscriptions sometimes, meaning the bank may reach out to non-interested customers.
Low False Negatives (374): The model misses some actual subscribers, but the recall (77%) is strong.
Overall, this model is good at identifying subscribers, even though it slightly over-predicts yeses. Since the bank has resources, this trade-off is acceptable