Predictive Modeling: the importance of data in decision making for NPOs.

Introduction

Non-Profit Organizations (NPOs) play a vital role in social, educational, and economic development, but some, especially in underdeveloped countries like Haiti, rely heavily on gut instinct and traditional methods to make decisions, which leads to poor donor retention, ineffective communication, weak donor interaction, and unsuccessful campaigns. Since NPOs rely heavily on fundraising, understanding donor behavior is a must, as one of the major challenges NPOs face is maintaining donor engagement and loyalty over time.

The primary objective of this project is to showcase the importance of using data to make informed decisions in NPOs. For that purpose, the project aims to analyze donor behavior and categorize donors for a random and fictitious organization. The insights gained throughout this analysis will foster the use of data and machine learning in decision-making, which can therefore help NPOs design more effective fundraising and communication strategies, ultimately enhancing their ability to sustain and grow their initiatives.

To achieve this, we use the Donor Data from Kaggle, which comprises six tables detailing donors, donations, campaigns, project results, and engagement activities. These data provide a comprehensive view of donor interactions and contributions, forming the basis for our analysis.

Methodology

Exploratory data Analysis

- data preparation
- business understanding
- data understanding

Major questions

These questions deepen our grasp of the business and data context, steering the analysis toward the NPOs' core concerns and enabling a thorough exploration of donor dynamics.

This part is crucial, as it serves as a compass that guides our analysis and helps address stakeholders' concerns or uncover relevant information.

Modeling/ Model Evaluation

Recommendations

Contact information

the contact information of the two data scientist who work on this project is provided.

EDA – Exploratory Data Analysis

Data preparation

Libraries and files importation

```
# libraries importation
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.metrics import roc_auc_score, f1_score, recall_score, accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# files importation
donors= pd.read_csv('donors.csv')
campaigns= pd.read_csv('campaigns.csv')
```

```
donations= pd.read_csv('donations_linked.csv')
engagement_history= pd.read_csv('engagement_history.csv')
engagement_outcomes =pd.read_csv('engagement_outcomes.csv')
impact= pd.read_csv('impact.csv')
```

```
print(f"Number of Donors" ,donors.shape[0] ,"\nNumber donations", donations.shape[0],
     "\nNumber of Campaigns", campaigns.shape[0], "\nEngagement_history",engagement_history.shape[0],"\nImpact",impact.shap
```

```
Number of Donors 4891
Number donations 17840
Number of Campaigns 10
Engagement_history 24238
Impact 50
```

Merging all Tables

```
Donors_donations= pd.merge(donors,donations , on='DonorID', how='left')
Donors_donations_Campaigns= pd.merge(Donors_donations, campaigns, on='CampaignID', how='left')
Donors_donations_Campaigns_engagement_history =pd.merge(Donors_donations_Campaigns, engagement_history, on='DonorID',  how=
Donors_donations_Campaigns_engagement_history_impacts =pd.merge(Donors_donations_Campaigns_engagement_history,impact, on ='
```

Comment: there are 7 tables in the dataset, we merge 6 of them because donors.csv and donations.csv are the same tables.

```
# all tables are merged into one
df= Donors_donations_Campaigns_engagement_history_impacts

```

```
# the merged tables
df.head()

```

	DonorID	Age	Gender	Location	JoinDate	DonationID	DonationDate	Amount	CampaignID	CampaignName	StartDate	EndDat
0	DNR00001	56	Male	QLD	2020-10-03 22:59:27.552825	DNT002656	2015-06-16	124.99	CAMP005	Youth Initiative 005	59:27.6	59:27
1	DNR00001	56	Male	QLD	2020-10-03 22:59:27.552825	DNT002656	2015-06-16	124.99	CAMP005	Youth Initiative 005	59:27.6	59:27
2	DNR00001	56	Male	QLD	2020-10-03 22:59:27.552825	DNT002656	2015-06-16	124.99	CAMP005	Youth Initiative 005	59:27.6	59:27
3	DNR00001	56	Male	QLD	2020-10-03 22:59:27.552825	DNT002656	2015-06-16	124.99	CAMP005	Youth Initiative 005	59:27.6	59:27
4	DNR00001	56	Male	QLD	2020-10-03 22:59:27.552825	DNT002656	2015-06-16	124.99	CAMP005	Youth Initiative 005	59:27.6	59:27

```
print( 'Number of Records',df.shape[0])
```

```
Number of Records 440083
```

Check for null or missing values

```
df.isnull().sum().sort_values(ascending=False)
```

	0
CampaignID	638
Amount	638
DonationDate	638
DonationID	638
Cost	638
ActualAmount	638
Value	638
ImpactType	638
StartDate	638
CampaignName	638
EndDate	638
TargetAmount	638
Location	0
Gender	0
Age	0
DonorID	0
JoinDate	0
Channel	0
Date	0
EngagementOutcome	0

dtype: int64

```
# drop rows with null values
df[df.isnull().any(axis=1)]
df = df.dropna(subset=['DonationID'])
```

Dealing with Duplicates values

```
# drop rows with duplicates
duplicates = df[df.duplicated()]
print("Number of duplicate rows before dropping:", duplicates.shape[0])
df = df.drop_duplicates()
print("Number of duplicate rows after dropping:", df.duplicated().sum())
# Dropping rows duplicate donation ID
df = df.drop_duplicates(subset=['DonationID'])
```

```
Number of duplicate rows before dropping: 205
Number of duplicate rows after dropping: 0
```

Comment: Dropping duplicated DonationID rows is a must since we don't want to consider the same donation many time, which would biased our analysis.

```
# to verify if all null values were dropped run the code in this cell:
# df.isnull().sum().sort_values()
```

Fix the dates Format

```
df['JoinDate'] = pd.to_datetime(df['JoinDate'])
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
df['DonationDate'] = pd.to_datetime(df['DonationDate'], dayfirst=True)
```

Remove unneccesary columns

```
df = df.drop(['StartDate', 'EndDate'], axis=1)
```

Table's shape

```
1 # Run the code in this cell to see table's shape after cleaning
2 df.shape
```

```
(17840, 18)
```

```
1 # after cleaning we save the  merged dataset
2 df.to_csv('Data.csv',index=0)
```

Explore basic statistics for numerical features (mean, median, min, max)

```
1 df.describe().round(2)
```

	Age	JoinDate	DonationDate	Amount	TargetAmount	ActualAmount	Date	Value	Cost
count	17840.00	17840	17840	17840.00	17840.00	17840.00	17840	17840.00	17840.0
mean	48.44	2019-10-26 01:02:47.732196864	2020-02-09 03:17:45.470851840	110.24	62092.24	59866.38	2023-02-25 02:59:26.098654720	33.52	100.0
min	18.00	2015-04-23 22:59:27.552825	2015-01-01 00:00:00	10.00	33193.34	33179.39	2022-07-25 00:00:00	19.00	100.0
25%	33.00	2017-08-20 22:59:27.552825088	2017-07-07 00:00:00	38.90	49248.12	44689.69	2022-09-22 00:00:00	24.00	100.0
50%	48.00	2019-10-27 22:59:27.552825088	2020-02-11 00:00:00	79.15	58107.99	57113.83	2022-12-12 00:00:00	25.00	100.0
75%	64.00	2022-01-08	2022-09-11	147.52	69723.66	68170.70	2023-05-22	27.00	100.0

Business understanding

In today's fast-evolving fundraising landscape, nonprofit organisations face growing pressure to engage supporters meaningfully, demonstrate results, and build trust through transparency. Traditional, instinct-driven approaches no longer meet modern donor expectations

Adopting a data-informed approach is not just a technical upgrade—it's a cultural shift that aligns mission, message, and measurement. With the right mindset and existing tools, even small organisations can use data to boost engagement and impact.

Delivering measurable change increasingly depends on how well nonprofits engage and retain supporters. As donor expectations rise and attention spans shrink, sustaining long-term relationships becomes a major challenge.McKinsey Global Institute says that data-driven organisations are 23 times more likely to acquire customers, 6 times as likely to retain customers and 19 times as likely to be profitable, showing the importance of including data in decision making.

Our objectives focus on helping NPOs reduce donor churn, strengthen their relationships with donors, and improve alignment between fundraising and impact, with data driven analysis. We aim to help them engage donors and inspire new strategies for attraction and retention.

Data Understanding

A synthetic dataset was created to reflect the operational characteristics of a nonprofit organisation, focused on youth and community welfare. It includes relational tables for donors, donations, campaigns, impact outcomes, and engagement activities. These tables are structured to support a complete engagement analysis from start to finish.

Although the data is fictitious, it reflects realistic patterns based on sector / research. These patterns include donor attrition, variations in donation behavior, and inconsistent campaign performance . The goal is to encourage the use of data in nonprofit environments and to provide the NPO community with a clear example of what a data-informed approach can accomplish. This simulation allows for open analysis while maintaining privacy and ethical standards.

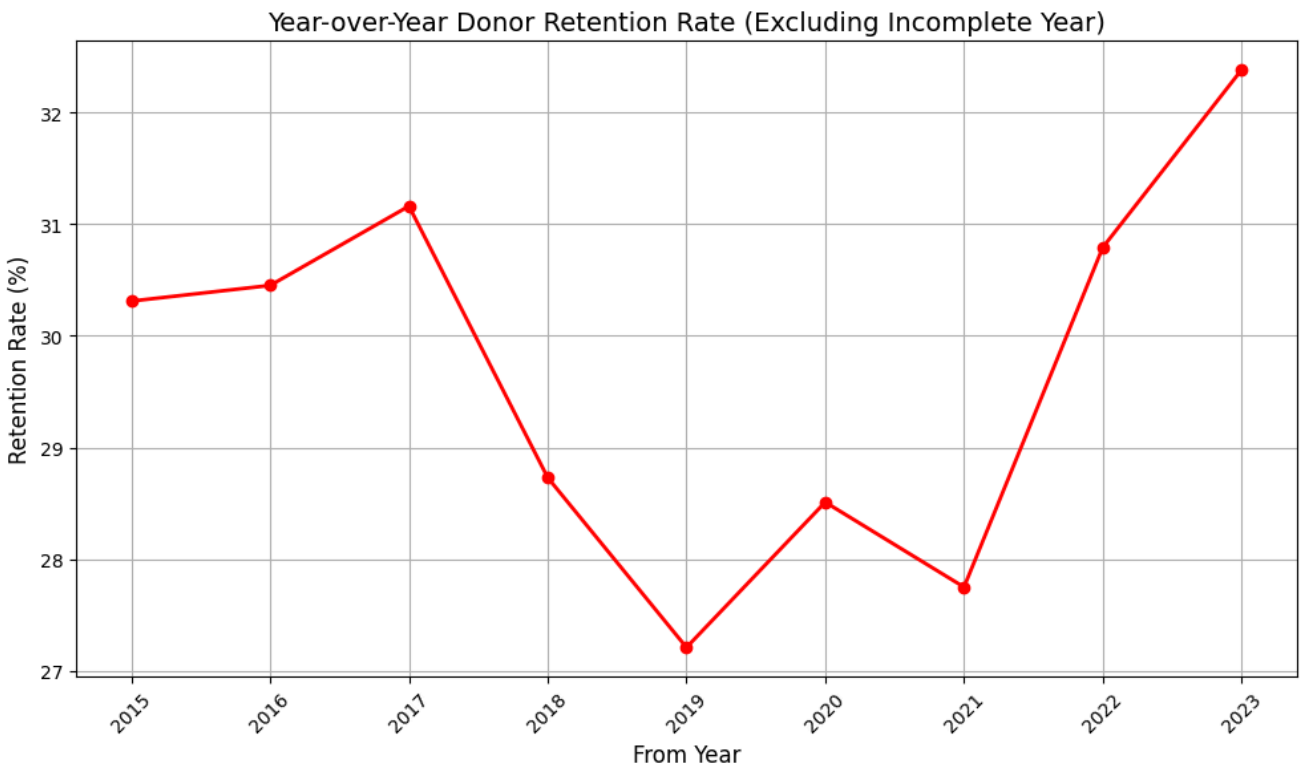
The structure of the dataset makes it possible to demonstrate how nonprofit engagement can be improved through analytics without relying on sensitive real-world data.

Major questions

this section is a follow up of the two previous sections. it builds our business and data understanding and allow us to leverage the data to get meaningful and pertinent information. We will also tell the purpose of each Newfound information, and by the same occasion foster a data informed approached in NPOs.

How has donor retention evolved over time ?

```
# conversion
df['DonationDate'] = pd.to_datetime(df['DonationDate'], dayfirst=True)

# Extract donation year
df['DonationYear'] = df['DonationDate'].dt.year
years = sorted(df['DonationYear'].unique())
retention_stats = []

for i in range(len(years) -1 ):
    year_current = years[i]
    year_next = years[i + 1]

    donors_current = set(df[df['DonationYear'] == year_current]['DonorID'])
    donors_next = set(df[df['DonationYear'] == year_next]['DonorID'])

    retained_donors = donors_current & donors_next
    retention_rate = len(retained_donors) / len(donors_current) * 100 if donors_current else 0

    retention_stats.append({
        "From Year": year_current,
        "To Year": year_next,
        "Donors in Year": len(donors_current),
        "Retained Donors": len(retained_donors),
        "Retention Rate (%)": round(retention_rate, 2)
    })

retention_df = pd.DataFrame(retention_stats)

# Exclude the last year from the plot as it has incomplete data
retention_df_plot = retention_df[:-1]

plt.figure(figsize=(10, 6))
plt.plot(retention_df_plot["From Year"].astype(str), retention_df_plot["Retention Rate (%)"], marker='o',  color = "red", l
plt.title("Year-over-Year Donor Retention Rate (Excluding Incomplete Year)", fontsize=14)
plt.xlabel("From Year", fontsize=12)
plt.ylabel("Retention Rate (%)", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
# to save the image
plt.savefig('1st_image.png')
# to see the image
plt.show()

```



Comment: This line graph illustrates the percentage of donors retained from one year to the next over a ten-year period. While retention fluctuates around 30%, occasional dips highlight the importance of targeted re-engagement strategies and sustained communication with donor segments.

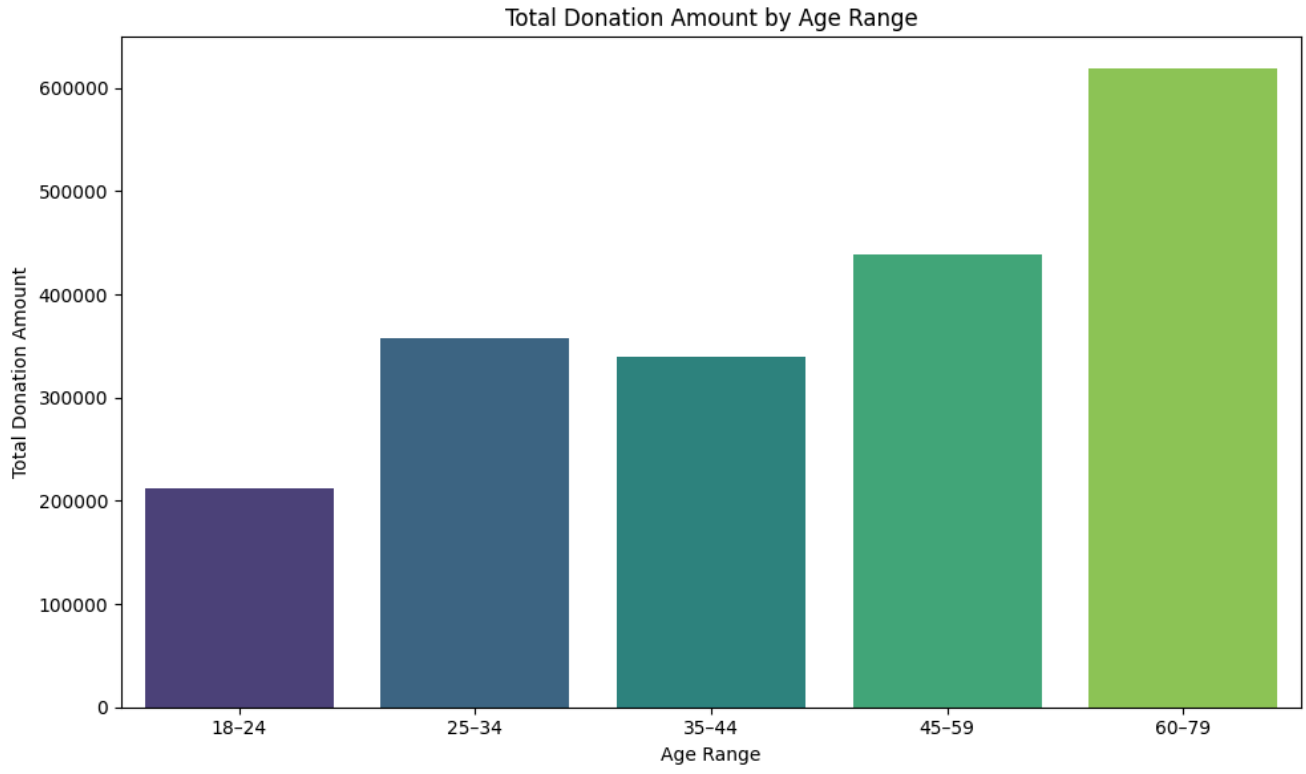
Any NPO will want this information to position themselves regarding donor retention. By knowing how they retain donors, they can determine whether or not to foster interaction with donors, build more campaigns or any other inspired-data decision that could improve donor retention.

What is the donation frequency per age range ?

```
# Find out Maximum and Minimum age
print("the oldest person are ", df.Age.min())
print("the youngest person are ", df.Age.max())
```

```
the oldest person are  18
the youngest person are  79
```

```
# Define age ranges

# Define bins and matching labels
bins = [18, 24, 35, 45, 59, 79]
labels = ['18-24', '25-34', '35-44', '45-59', '60-79']  # 5 labels for 5 intervals

# Create a new column with age ranges
df['AgeRange'] = pd.cut(df['Age'], bins=bins, labels=labels, right=True, include_lowest=True)

# Calculate total donation amount per age range
donation_by_age_range = df.groupby('AgeRange', observed=True)['Amount'].sum().reset_index()

# Plot the bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='AgeRange', y='Amount', data=donation_by_age_range, palette='viridis')

plt.title('Total Donation Amount by Age Range')
plt.xlabel('Age Range')
plt.ylabel('Total Donation Amount')
plt.tight_layout()
# to save the image
plt.savefig('2nd_image.png')
# to see the image
plt.show()

```

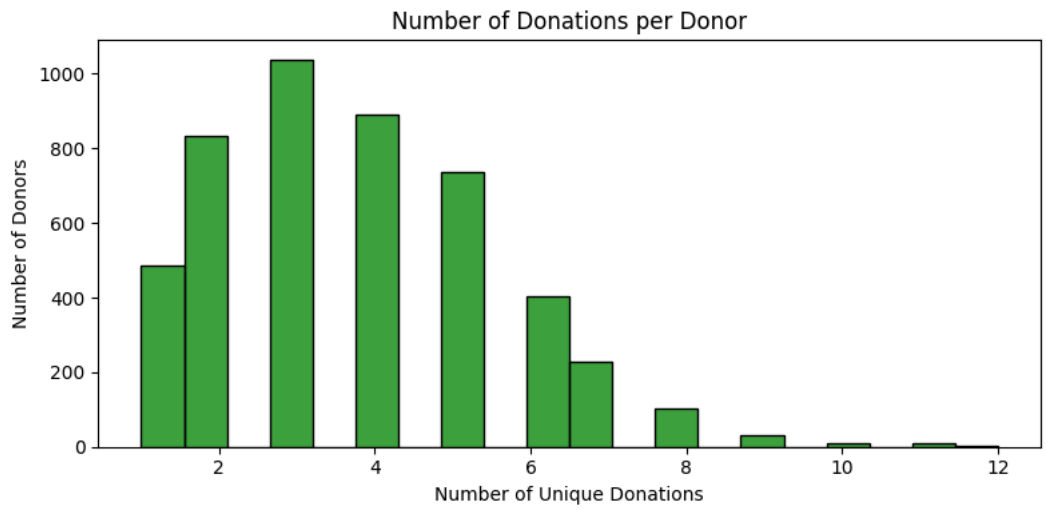


Comment: Donors who are older are likely to donate more than young individuals in the range of 25 to 34 years old. The results match real world data.

Any NPO would want this information to know which age range to target the most and which one to sensitize to donate more, whether it's through campaign messages or any other means.

How many time donors are likely to donate ?

```
# import necessary library
import matplotlib.pyplot as plt
import seaborn as sns

# Group by DonorID to get donation count
donation_counts = donations.groupby('DonorID')['DonationID'].count()

# Plot
plt.figure(figsize=(8, 4))
sns.histplot(donation_counts, bins=20, kde=False, color='green')
plt.title('Number of Donations per Donor')
plt.xlabel('Number of Unique Donations')
plt.ylabel('Number of Donors')
plt.tight_layout()
# to save the image
plt.savefig('3rd_image.png')
# to see the image
plt.show()
```



Comment: Most donors donates around 2 to 4 times. We can observe that the Number of Donors decreases For higher number of donations, showing moderate donor retention.

This type of information is very useful because some NPOs might want to set their new objectives to higher number of donations per donor.

What is the donation amount repartition per gender?

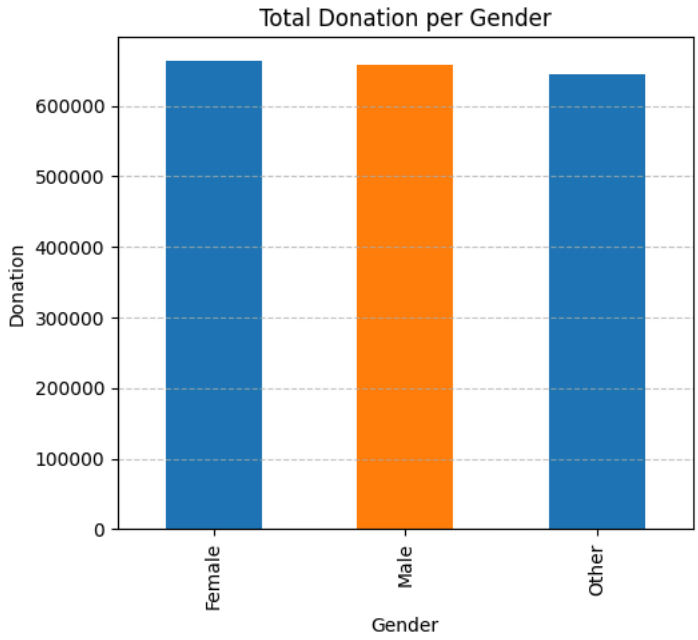
```
df.Age.value_counts()

```

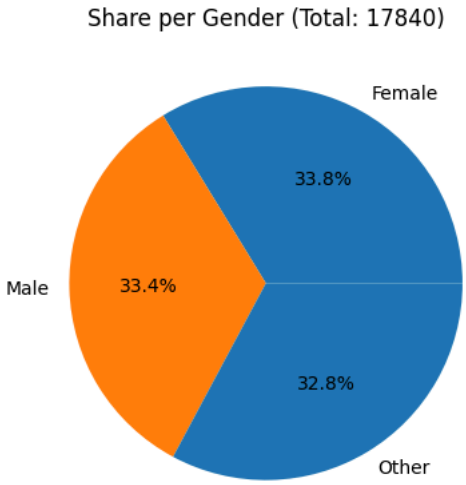
Age	count
45	370
40	351
53	349
33	349
43	342
...	...
70	236
51	231
28	225
67	212
60	210

62 rows × 1 columns

dtype: int64

```
donation_by_gender = df.groupby('Gender')['Amount'].sum()

# Kalkile total donation tout ansanm
total_donation = df['Amount'].count()

# subplots (2 graf)
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# 1 Bar chart  donation by gender
donation_by_gender.plot(kind='bar', ax=axes[0], color=['#1f77b4', '#ff7f0e'])
axes[0].set_title('Total Donation per Gender')
axes[0].set_xlabel('Gender')
axes[0].set_ylabel('Donation')
axes[0].grid(axis='y', linestyle='--', alpha=0.7)

# 2 Pie chart
donation_by_gender.plot(kind='pie', autopct='%1.1f%%', ax=axes[1], ylabel='', colors=['#1f77b4', '#ff7f0e'])
axes[1].set_title(f'Share per Gender (Total: {total_donation})')


plt.tight_layout()
# to save the image
plt.savefig('4rth_image.png')
# to see the image
plt.show()
donation_by_gender
```





Gender	Amount
Female	663950.26
Male	657135.77
Other	645596.74

dtype: float64

Comment: For this fictitious dataset, gender is almost equally distributed, but in some regions, NPOs might observe some imbalance and, therefore, make it their objective to understand why and do something about it.

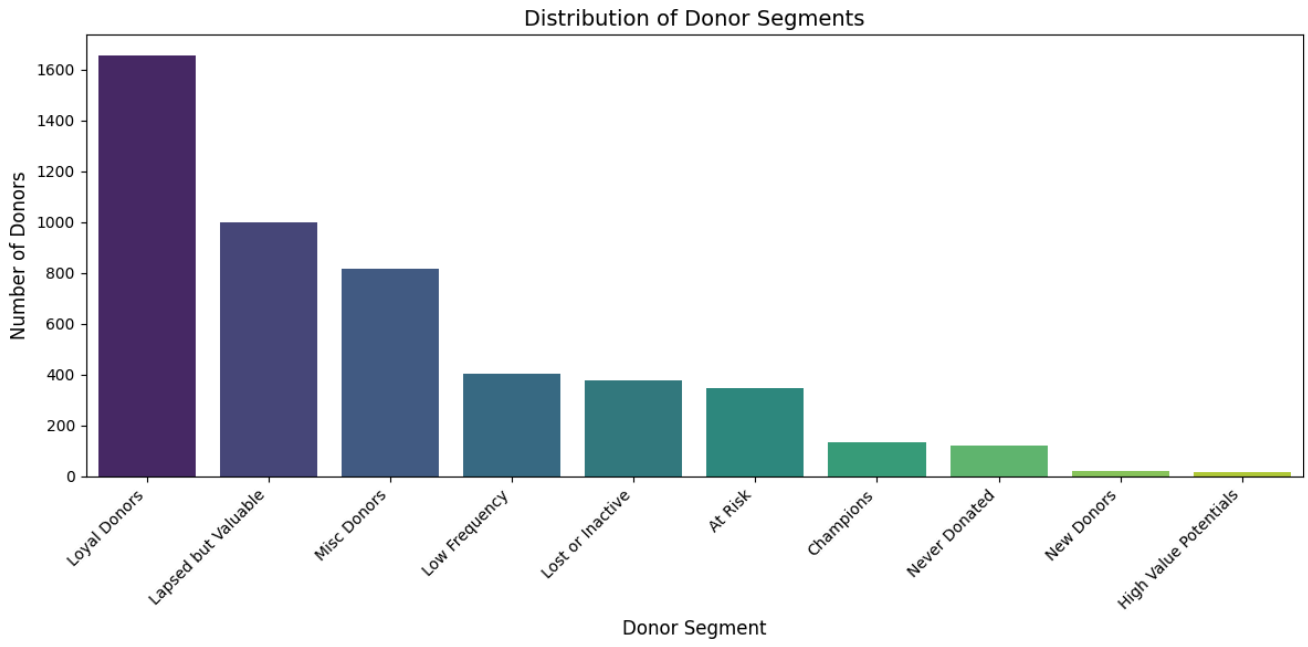
How can the different type of donors be classified ?

```
import numpy as np

# Convert donation date to datetime
df['DonationDate'] = pd.to_datetime(df['DonationDate'])

# Create donor-level summary
today = pd.to_datetime('today')
donor_summary = df.groupby('DonorID').agg(
    FirstDonationDate=('DonationDate', 'min'),
    LastDonationDate=('DonationDate', 'max'),
    Frequency=('DonationID', 'count'),
    Monetary=('Amount', 'sum')
).reset_index()
donor_summary['Recency'] = (today - donor_summary['LastDonationDate']).dt.days

# Merge with full donor list
donor_full_summary = pd.merge(donors[['DonorID']], donor_summary, on='DonorID', how='left')
donor_full_summary['Recency'] = donor_full_summary['Recency'].fillna(np.inf)
donor_full_summary['Frequency'] = donor_full_summary['Frequency'].fillna(0)
donor_full_summary['Monetary'] = donor_full_summary['Monetary'].fillna(0)
# Assign segments
def assign_segment(row):
    r = row['Recency']
    f = row['Frequency']
    m = row['Monetary']

    if np.isinf(r) or pd.isna(r):
        return 'Never Donated'
    elif r <= 365 and f == 1:
        return 'New Donors'
    elif r <= 365 and f >= 4 and m >= 750:
        return 'Champions'
    elif r <= 1095 and f >= 4:
        return 'Loyal Donors'
    elif r <= 1095 and 2 <= f <= 4 and m >= 750:
        return 'High Value Potentials'
    elif r > 1095 and f >= 2 and m >= 250:
```

```
        return 'Lapsed but Valuable'
    elif 365 < r <= 1095 and f <= 2:
        return 'At Risk'
    elif r > 1095 and f == 1:
        return 'Lost or Inactive'
    elif f <= 2 and m < 250:
        return 'Low Frequency'
    else:
        return 'Misc Donors'

donor_full_summary['Segment'] = donor_full_summary.apply(assign_segment, axis=1)

# Return segment summary for confirmation
segment_summary = donor_full_summary['Segment'].value_counts().reset_index()
segment_summary.columns = ['Segment', 'Count']
segment_summary
```

	Segment	Count
0	Loyal Donors	1653
1	Lapsed but Valuable	998
2	Misc Donors	818
3	Low Frequency	405
4	Lost or Inactive	376
5	At Risk	347
6	Champions	133
7	Never Donated	122
8	New Donors	20
9	High Value Potentials	19

```
plt.figure(figsize=(12, 6))
sns.barplot(x='Segment', y='Count', data=segment_summary, palette='viridis')
plt.title('Distribution of Donor Segments', fontsize=14)
plt.xlabel('Donor Segment', fontsize=12)
plt.ylabel('Number of Donors', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
# to save the image
plt.savefig('7th_image.png')
# to see the image
```



Comments: Not every donor contributes the same. Therefore, donors have to be interacted with differently. Some donors may receive messages of recognition and heartfelt remerciements, while others may need to be re-engaged toward donating again. Knowing the different donor segments is crucial information for NPOs.

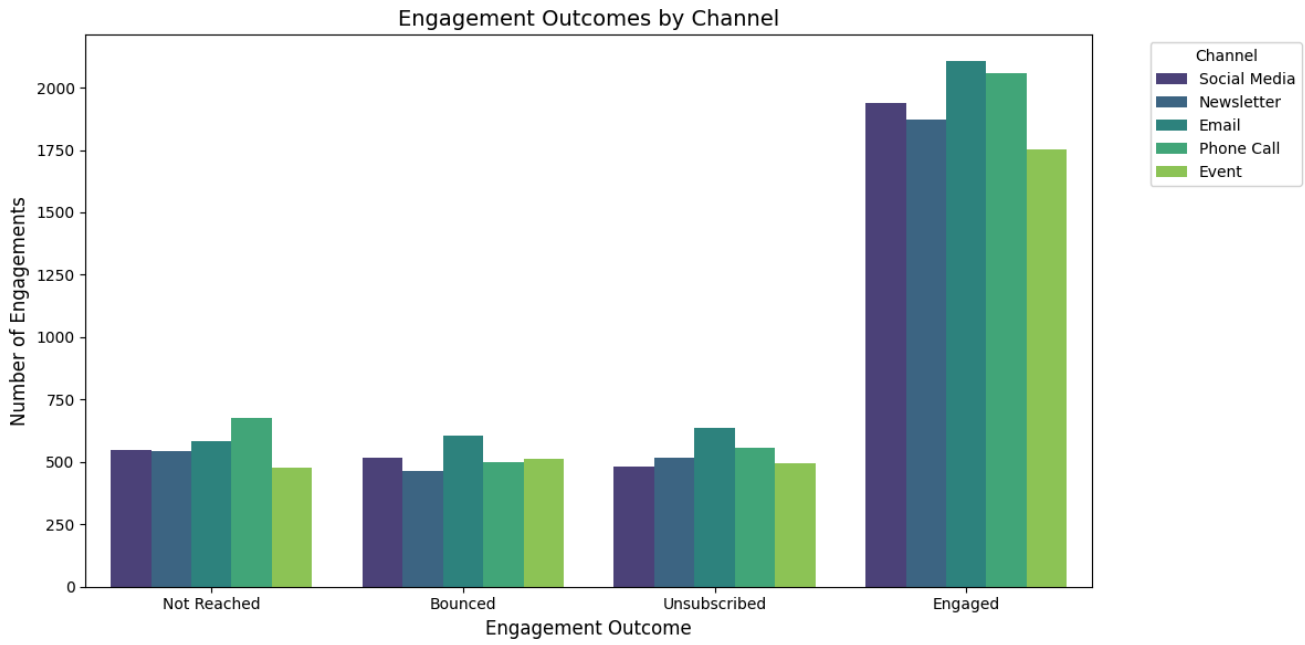
what channel results to the best engagement outcome?

```
1 df.EngagementOutcome.value_counts()
```

EngagementOutcome	count
Engaged	9723
Not Reached	2831
Unsubscribed	2688
Bounced	2598

dtype: int64

```
 1 # Create a countplot showing outcomes across channels
 2 plt.figure(figsize=(12, 6))
 3 sns.countplot(data=df, x='EngagementOutcome', hue='Channel', palette='viridis')
 4 plt.title('Engagement Outcomes by Channel', fontsize=14)
 5 plt.xlabel('Engagement Outcome', fontsize=12)
 6 plt.ylabel('Number of Engagements', fontsize=12)
 7 plt.xticks(rotation=0)
 8 plt.legend(title='Channel', bbox_to_anchor=(1.05, 1), loc='upper left')
 9 plt.tight_layout()
10 # to save the image
11 plt.savefig('6xth_image.png')
12 # to see the image
13 plt.show()
```




```
1 df.Channel.value_counts()
```

Channel	count
Email	3931
Phone Call	3790
Social Media	3486
Newsletter	3398
Event	3235

dtype: int64

Comment: Channels seem to have approximately the same engagement outcome in the context of this analysis. However, some NPOs might have a certain channel that leads to better results. For example, in the context of Haiti, some NPOs may want to use social media instead of email, since it is more widely used.

Modeling

==========

We will develop a concrete and practical plan, along with an ML pipeline, that aligns with the objective of reducing churn and unsubscriptions, strengthening donor relationships, and improving alignment between collection and impact — with clear, actionable ML deliverables.

```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classification_report
from xgboost import XGBClassifier
#import shap
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Model 1

bellow are the many steps taken to build the model.

Feature Engineering

In this part we will generate new features that will help us create our model. these features will be the criterion that will help us predict donor loyalty.

Splitting the Data The model will be based on historical data to predict the loyalty of a donor.

we will split the historical data into two parts: one for the train set and one for the test set. the train set wil contain all historical data that are three years before 2025.

```
cutoff_date = df['DonationDate'].max() - pd.Timedelta(days=365)
# Historical data (before cutoff)
history = df[df['DonationDate'] <= cutoff_date]
# Future data (after cutoff)
future = df[df['DonationDate'] > cutoff_date]
```

Basic RFM Calculation RFM stands for : Recency, Frequency, Monetary

we will also create the target variable Loyalty indicating if the donor is still active.

```
#  RFM calculation
rfm = history.groupby('DonorID').agg(
    Recency=('DonationDate', lambda x: (cutoff_date - x.max()).days),
    Frequency=('DonationID', 'count'),
    Monetary=('Amount', 'sum'),
    AvgAmount=('Amount', 'mean'),

).reset_index()


```

```
# Loyalty = 1 if the donor donated after cutoff_date
rfm['Loyalty'] = (rfm['DonorID'].isin(future['DonorID'])).astype(int)
```

Additional Behavioral Variables

```
# Variance of donation amounts
var_df = history.groupby('DonorID')['Amount'].std().fillna(0).reset_index()
var_df.rename(columns={'Amount': 'DonationVariance'}, inplace=True)

# Average interval between donations (days)
interval_df = history.groupby('DonorID')['DonationDate'].apply(
    lambda x: x.sort_values().diff().dt.days.mean()
).fillna(0).reset_index(name='AvgInterval')

# Donor tenure
tenure_df = history.groupby('DonorID')['DonationDate'].min().reset_index()
tenure_df['Tenure'] = (cutoff_date - tenure_df['DonationDate']).dt.days
tenure_df.drop(columns='DonationDate', inplace=True)

# Merge
rfm = rfm.merge(var_df, on='DonorID', how='left')
rfm = rfm.merge(interval_df, on='DonorID', how='left')
rfm = rfm.merge(tenure_df, on='DonorID', how='left')

```

Finds the channel that the donor uses most often.

that will help us Capture the donor's preferred communication/interaction channel, which often correlates with loyalty.

```
# Most frequent channel
fav_channel_df = history.groupby('DonorID')['Channel'].agg(
    lambda x: x.mode()[0] if len(x.mode()) > 0 else 'unknown'
).reset_index(name='FavChannel')

# Number of different channels used
diversity_df = history.groupby('DonorID')['Channel'].nunique().reset_index(name='ChannelDiversity')

# Counts per channel type (integers, e.g., Online, Event, Phone)
channel_counts = history.pivot_table(
    index='DonorID',
    columns='Channel',
    values='DonationID',
    aggfunc='count',
    fill_value=0
).reset_index()

# Merge all channel features into the main RFM dataframe
rfm = rfm.merge(fav_channel_df, on='DonorID', how='left')
rfm = rfm.merge(diversity_df, on='DonorID', how='left')
rfm = rfm.merge(channel_counts, on='DonorID', how='left')

# Group by DonorID to get one AgeRange and Gender per donor (most frequent value)
age_df = df.groupby('DonorID')['AgeRange'].agg(lambda x: x.mode()[0]).reset_index()
##gender = df.groupby('DonorID')['Gender'].agg(lambda x: x.mode()[0]).reset_index()

# Merge into the main RFM dataset
rfm = rfm.merge(age_df, on='DonorID', how='left')
##rfm = rfm.merge(gender, on='DonorID', how='left')

# Display updated RFM
rfm.head(1)

```

	DonorID	Recency	Frequency	Monetary	AvgAmount	Loyalty	DonationVariance	AvgInterval	Tenure	FavChannel	ChannelDiversi
0	DNR00001	880	3	377.67	125.89	0	2.035166	1174.5	3229	Social Media	

Prepare Features We select the numerical and categorical features for the model and Separates features (X) from the target (y).

```
channel_cols = [c for c in channel_counts.columns if c != 'DonorID']

features = [
    'Recency', 'Frequency', 'Monetary', 'AvgAmount',
    'DonationVariance', 'AvgInterval', 'Tenure',
    'ChannelDiversity', 'FavChannel','AgeRange'
```

```
] + channel_cols

X = rfm[features]
y = rfm['Loyalty']


numeric_features = [
    'Recency', 'Frequency', 'Monetary', 'AvgAmount',
    'DonationVariance', 'AvgInterval', 'Tenure',
    'ChannelDiversity'
] + channel_cols

categorical_features = ['FavChannel','AgeRange']
```

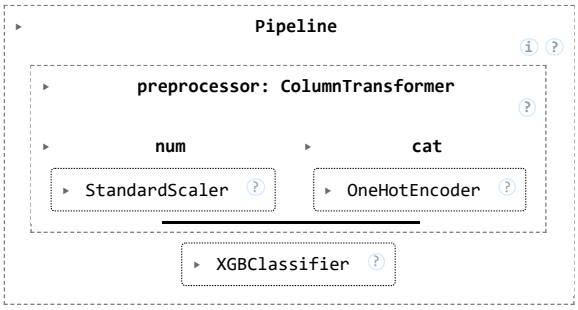
Split train / test

```
##Split train / test

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


```

Chosen model:

The model used is XGBoost Classifier, selected for its ability to handle non-linear relationships and imbalanced datasets.

```
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
])

ratio = y_train.value_counts()[0] / y_train.value_counts()[1]
clf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(
        n_estimators=300,
        learning_rate=0.05,
        max_depth=5,
        subsample=0.8,
        colsample_bytree=0.8,
        scale_pos_weight=ratio,
        eval_metric='logloss',
        random_state=42
    ))
])

clf_pipeline.fit(X_train, y_train)
```

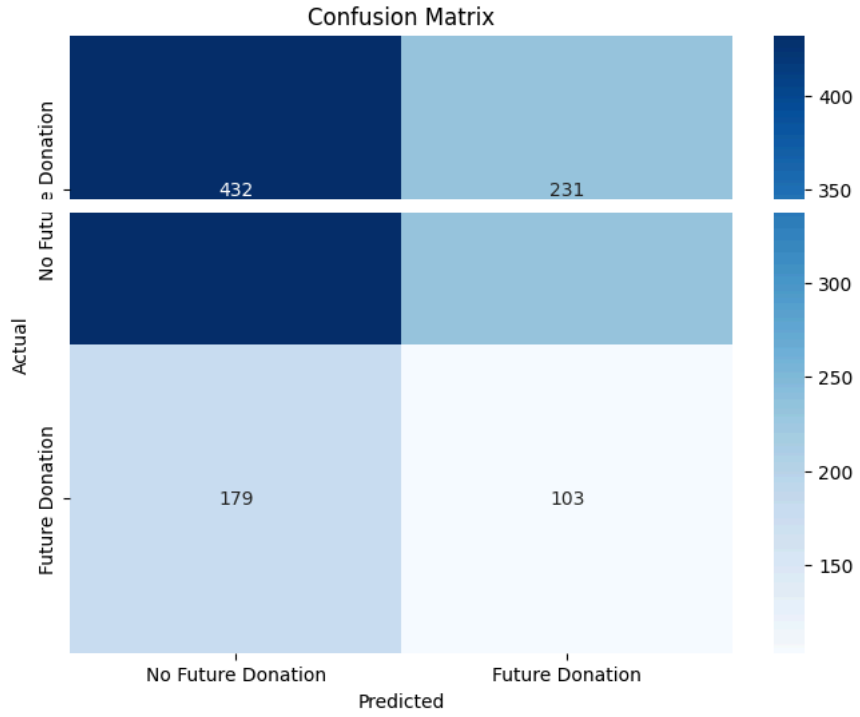


Model Evaluation

```
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix
# Prédictions sur le test set
y_pred = clf_pipeline.predict(X_test)
y_proba = clf_pipeline.predict_proba(X_test)[:, 1]

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# ROC-AUC
roc_auc = roc_auc_score(y_test, y_proba)
print("ROC-AUC:", roc_auc)

```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Future Donation', 'Future Donation'], yticklabels=['No
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
# to save plot
plt.savefig('8th_image.png')
# to see plot
plt.show()
```

```
Accuracy: 0.5661375661375662
ROC-AUC: 0.5091915107559662
Confusion Matrix:
 [[432 231]
 [179 103]]
```



Interpretation

The model correctly predicts donor loyalty about 57% of the time, slightly better than random (50%), but not yet reliable for decision-making. it has limited ability to distinguish between loyal and non-loyal donors (0.5 = random). It needs improvement. The model correctly identifies many non-loyal donors (432), but struggles with loyal donors (103 correctly, 179 missed).

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.65      0.68       663
           1       0.31      0.37      0.33       282

    accuracy                           0.57       945
   macro avg       0.51      0.51      0.51       945
weighted avg       0.59      0.57      0.58       945
```

Overview of Model's results

```
rfm['Fidelite_Prob'] = clf_pipeline.predict_proba(rfm[features])[:, 1]

rfm['Fidelite_Segment'] = pd.cut(
    rfm['Fidelite_Prob'],
    bins=[0, 0.33, 0.66, 1],
    labels=['Weak Loyalty ', 'Average Loyalty', 'Strong Loyalty']

```

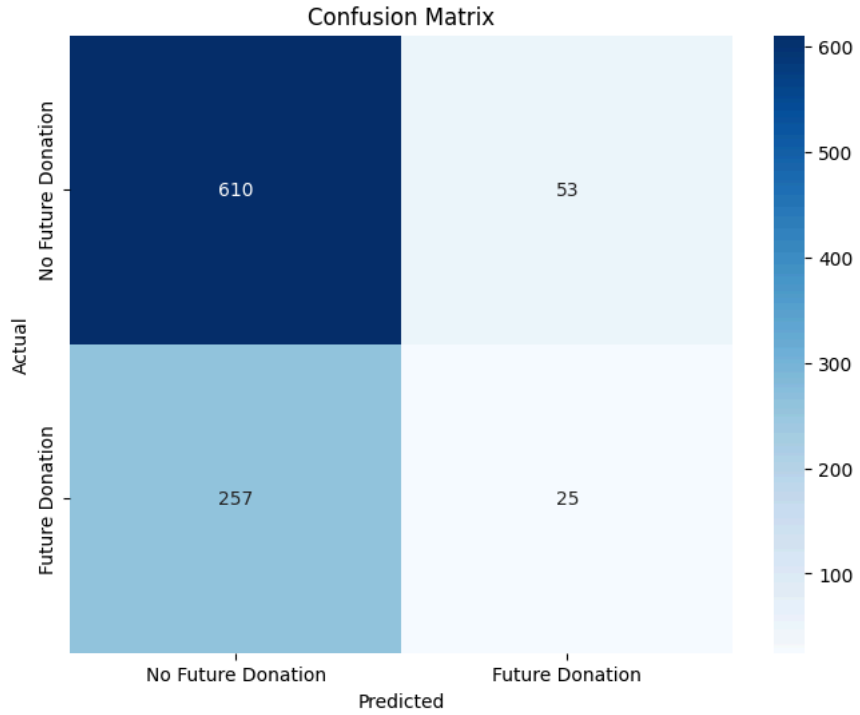
```
)

print("\n 🔹 Quick overview on the result upon all the Datateset :\n ")
print(rfm[['DonorID', 'Fidelite_Prob', 'Fidelite_Segment']].head())

```

```
🔹 Quick overview on the result upon all the Datateset :

    DonorID  Fidelite_Prob Fidelite_Segment
0  DNR00001       0.341485  Average Loyalty
1  DNR00002       0.504820  Average Loyalty
2  DNR00003       0.149082    Weak Loyalty
3  DNR00004       0.508528  Average Loyalty
4  DNR00005       0.583386  Average Loyalty
```

Model 2 : Random Forest

```
rf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(
        n_estimators=300,
        max_depth=None,
        min_samples_split=5,
        min_samples_leaf=2,
        class_weight='balanced',
        random_state=42,
        n_jobs=-1
    ))
])

# Convert target variables to integer type
y_train = y_train.astype(int)
y_test = y_test.astype(int)

# Train the model
rf_pipeline.fit(X_train, y_train)

# Predictions
y_pred = rf_pipeline.predict(X_test)
y_proba = rf_pipeline.predict_proba(X_test)[:, 1]

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_proba))
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Future Donation', 'Future Donation'], yticklabels=['No
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
# to save plot
plt.savefig('9th_image.png')
# to see plot
plt.show()
```

```
Accuracy: 0.671957671957672
ROC-AUC: 0.5066482675994566
Confusion Matrix:
 [[610  53]
 [257  25]]

Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.92      0.80       663
           1       0.32      0.09      0.14       282

    accuracy                           0.67       945
   macro avg       0.51      0.50      0.47       945
weighted avg       0.59      0.67      0.60       945
```



Interpretation

The model correctly predicts about 67% of loyal donors. That's higher than the XGBoost model (≈0.57). We can therefore say that this one does better overall at predicting loyal donors

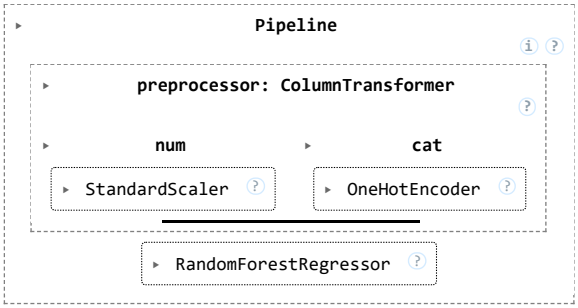
However, accuracy alone is misleading when classes are imbalanced (as is the case here.

Model 3 : regression task

this model is build to try to predict future donation amount.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Aggregate donor-level data
dataset = df.groupby('DonorID').agg({
    'Age':'first',
    'Gender':'first',
    'JoinDate':'first',
    'Location':'first',
    'Amount':['sum','mean','count'],
    'Channel': lambda x: x.mode()[0],
    'EngagementOutcome': lambda x: x.mode()[0],
    'Value':'first'
}).reset_index()

dataset.columns = ['DonorID','Age','Gender','JoinDate','Location','TotalAmount','AvgAmount','DonationCount','Channel','Enga

# 2. Create the Target
# target 2: Future average donation amount (regression)
dataset['FutureDonation'] = np.random.randint(0,2,size=len(dataset))  # Randomly assign whether donor will donate (0/1)
```

```
dataset['FutureAmount'] = np.where(dataset['FutureDonation']==1,        # Future donation amount
                                   np.random.uniform(10,500,size=len(dataset)),
                                   0)

# Target variable: 'Value'
y = dataset['FutureAmount']

# Select relevant features
features = ['Age','Gender','Location','JoinDate','TotalAmount','AvgAmount','DonationCount','Channel','EngagementOutcome','V
X = dataset[features]

# Define numeric and categorical columns
numeric_features = ['Age', 'TotalAmount','DonationCount']
categorical_features = ['Gender', 'Channel', 'EngagementOutcome']

# Create preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),        # Standardize numeric features
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)  # One-hot encode categorical features
    ])

from sklearn.model_selection import train_test_split

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.ensemble import RandomForestRegressor

# Create the model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),         # Apply preprocessing
    ('regressor', RandomForestRegressor(random_state=42))  # Train Random Forest Regressor
])

# Train the model
model.fit(X_train, y_train)

```



This pipeline predicts the expected future donation amount for each donor, enabling nonprofits to plan fundraising strategies and target high-value donors more effectively.

```
from sklearn.metrics import mean_squared_error, r2_score

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error (RMSE)

# Calculate the coefficient of determination R²
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse}")
print(f"R²: {r2}")

```

```
RMSE: 171.94733378480254
R²: -0.13073355442002632
```

These metrics allow you to quantify the performance of the regression model in predicting donors' future donation amounts. The MSE measures the average squared difference between predicted and actual values. The RMSE is the square root of the MSE, providing an error expressed in the same units as the target variable (FutureAmount). A lower RMSE indicates better predictive performance.

Model Performance

RMSE ≈ 165.5 On average, the model's predictions for future donation amounts deviate by about 165 units from the actual values. Since donations vary widely, this represents a relatively large error, indicating that the predictions are not very precise.

R^2 ≈ -0.10 An R^2 value below 0 indicates that the model performs worse than a simple baseline that predicts the mean donation for all donors. This suggests that the Random Forest model fails to capture meaningful patterns from the current features for predicting future donation amounts.

Testing model With a new data.

```
new_data = {
    'Age': [45],
    'Gender': ['Male'],
    'Location': ['Ouest, Haiti'],
    'TotalAmount': [1700.00],
    'Channel': ['Email'],
    'ImpactType': ['Education'],
    'DonationCount':[4],
    'EngagementOutcome':['none']
}
new_df = pd.DataFrame(new_data)
```

```

# future value
predicted_value = model.predict(new_df)
print(f" Future value expected : {predicted_value[0]}")
```

```
Future value expected : 225.47299810821997
```

Recommendations

The use of data plays a crucial role in decision-making. Data can be used for inferrential anysis, predictive anlysis, etc... data allows organisation to make better and more informed decision overall.

Contact information 1

Bellow: The Contact Information Of the Two Data scientist in charge of the project.

- First Name: Haender Michael
- Last Name : Jean Louis
- Email : michaelhaenderjeanlouis@gmail.com
- Phone : +509 31742772
- whatsapp : +509 33905060
- LinkedIn : https://www.linkedin.com/in/michael-haender-jean-louis-4b7320316?utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=ios_app

Contact information 2

- First Name : John Widno
- Last Name : DORCY
- Email : dorcyjohnwidno97@gmail.com
- Phone : +509 38 05 8388
- LinkedIn : https://www.linkedin.com/in/john-widno-dorcy-19399a216/