DataScienceProject_Final

October 17, 2024

1 The Impact of Campaign Financing on the Outcomes of the 2016 U.S. House Races

1.1 Introduction

Campaign financing is a critical factor in the landscape of United States elections. The funds raised and spent by political candidates can significantly influence their visibility, messaging, and overall competitiveness. This project focuses on analyzing the 2016 U.S. House of Representatives races to understand how campaign financing affected election outcomes, but as we gear up for another general election in just a few weeks, I was curious to explore the data and identify any trends that could be useful in determining potential 2024 house race outcomes.

1.2 Project Goal

The primary goal of this analysis is to determine the extent to which financial factors impacted the results of the 2016 House races, and to identify whether other factors play a significant role in winning an election. By examining data on total contributions, expenditures, and other financial metrics for each candidate, we aim to:

- **Assess the relationship between campaign finances and electoral success.**
- **Identify key financial predictors of winning candidates.**
- **Utilize supervised machine learning models to predict election outcomes based on financial data.**

Through this investigation, we hope to gain insights into the influence of money in politics and how it shapes the democratic process.

1.3 Data Source

Source: https://www.kaggle.com/datasets/danerbland/electionfinance

The dataset utilized in this project was assembled to investigate the potential of predicting congressional election results using campaign finance reports from the period leading up to the 2016 election (January 1, 2015, through October 19, 2016). Each entry in the dataset represents a candidate and includes comprehensive information about their campaign finances, such as total contributions, total expenditures, state, district, office, and election outcomes.

Data Collection:

- **Campaign Finance Data:** Obtained directly from the Federal Election Commission (FEC), ensuring official and accurate financial records of each candidate's campaign activities.
- **Election Results and Vote Totals:** Sourced from CNN's election results page for the 2016 U.S. House races, providing verified outcomes and vote counts for winners of contested races.

Public Access and Licensing:

The dataset is publicly available and has been provided under the **CC0: Public Domain** license, allowing unrestricted use, distribution, and reproduction in any medium. This open access facilitates transparency and encourages further research into the effects of campaign financing on election outcomes.

1.3.1 Citation

Federal Election Commission. (2016). Candidate Summary File [Data set]. Retrieved from http://www.fec.gov/finance/disclosure/metadata/metadataforcandidatesummary.shtml

CNN Politics. (2016). Election Results: U.S. House. Retrieved from http://www.cnn.com/election/2016/results

Dataset compiled and made available on Kaggle:

Unknown Author. (2017). *Campaign Finance and Election Results* [Data set]. Kaggle. https://www.kaggle.com/datasets/benhamner/campaign-finance-and-election-results

---

1.4 Data Descriptions and Initial Exploration

[1]:
```python
import pandas as pd

df = pd.read_csv('CandidateSummaryAction1.csv')
```

[2]:
```python
#Observing the first few rows of data
df.head()

#Pulling a concise summary of the DataFrame
df.info()

#Pulling descriptive statistics for numerical columns
df.describe()

#Pulling descriptive statistics for all columns (including categorical)
df.describe(include='all')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1814 entries, 0 to 1813
Data columns (total 51 columns):
 #   Column                 Non-Null Count  Dtype
```

```
---  ------                  --------------  -----
 0   can_id                  1814 non-null   object
 1   can_nam                 1814 non-null   object
 2   can_off                 1814 non-null   object
 3   can_off_sta             1814 non-null   object
 4   can_off_dis             1812 non-null   float64
 5   can_par_aff             1813 non-null   object
 6   can_inc_cha_ope_sea     1812 non-null   object
 7   can_str1                1789 non-null   object
 8   can_str2                122 non-null    object
 9   can_cit                 1813 non-null   object
 10  can_sta                 1806 non-null   object
 11  can_zip                 1789 non-null   float64
 12  ind_ite_con             1570 non-null   object
 13  ind_uni_con             1538 non-null   object
 14  ind_con                 1616 non-null   object
 15  par_com_con             382 non-null    object
 16  oth_com_con             1011 non-null   object
 17  can_con                 675 non-null    object
 18  tot_con                 1695 non-null   object
 19  tra_fro_oth_aut_com     261 non-null    object
 20  can_loa                 640 non-null    object
 21  oth_loa                 67 non-null     object
 22  tot_loa                 653 non-null    object
 23  off_to_ope_exp          763 non-null    object
 24  off_to_fun              4 non-null      object
 25  off_to_leg_acc          1 non-null      object
 26  oth_rec                 534 non-null    object
 27  tot_rec                 1721 non-null   object
 28  ope_exp                 1717 non-null   object
 29  exe_leg_acc_dis         6 non-null      object
 30  fun_dis                 15 non-null     object
 31  tra_to_oth_aut_com      153 non-null    object
 32  can_loa_rep             294 non-null    object
 33  oth_loa_rep             33 non-null     object
 34  tot_loa_rep             306 non-null    object
 35  ind_ref                 831 non-null    object
 36  par_com_ref             24 non-null     object
 37  oth_com_ref             362 non-null    object
 38  tot_con_ref             872 non-null    object
 39  oth_dis                 795 non-null    object
 40  tot_dis                 1730 non-null   object
 41  cas_on_han_beg_of_per   700 non-null    object
 42  cas_on_han_clo_of_per   1434 non-null   object
 43  net_con                 1643 non-null   object
 44  net_ope_exp             1665 non-null   object
 45  deb_owe_by_com          732 non-null    object
 46  deb_owe_to_com          23 non-null     object
```

```
 47  cov_sta_dat            1814 non-null   object
 48  cov_end_dat            1814 non-null   object
 49  winner                 471 non-null    object
 50  votes                  379 non-null    float64
dtypes: float64(3), object(48)
memory usage: 722.9+ KB
```

[2]:

```
           can_id        can_nam can_off can_off_sta  can_off_dis can_par_aff  \
count        1814           1814    1814        1814  1812.000000        1813
unique       1814           1803       3          57          NaN          24
top     H6NM03075  RUBIO, MARCO        H          CA          NaN         REP
freq            1              2    1429         183          NaN         882
mean          NaN            NaN     NaN         NaN     7.903422         NaN
std           NaN            NaN     NaN         NaN    10.264533         NaN
min           NaN            NaN     NaN         NaN     0.000000         NaN
25%           NaN            NaN     NaN         NaN     1.000000         NaN
50%           NaN            NaN     NaN         NaN     4.000000         NaN
75%           NaN            NaN     NaN         NaN    10.000000         NaN
max           NaN            NaN     NaN         NaN    53.000000         NaN

       can_inc_cha_ope_sea      can_str1   can_str2    can_cit  …  \
count                 1812          1789        122       1813  …
unique                   3          1768        119       1062  …
top             CHALLENGER  PO BOX 10842  SUITE 200  LAS VEGAS  …
freq                   850             2          2         22  …
mean                   NaN           NaN        NaN        NaN  …
std                    NaN           NaN        NaN        NaN  …
min                    NaN           NaN        NaN        NaN  …
25%                    NaN           NaN        NaN        NaN  …
50%                    NaN           NaN        NaN        NaN  …
75%                    NaN           NaN        NaN        NaN  …
max                    NaN           NaN        NaN        NaN  …

       cas_on_han_beg_of_per  cas_on_han_clo_of_per   net_con  net_ope_exp  \
count                    700                   1434      1643         1665
unique                   673                   1407      1613         1653
top                 $100.00                   $5.00   $600.00  $83,145.56
freq                       7                      5         5            2
mean                     NaN                    NaN       NaN          NaN
std                      NaN                    NaN       NaN          NaN
min                      NaN                    NaN       NaN          NaN
25%                      NaN                    NaN       NaN          NaN
50%                      NaN                    NaN       NaN          NaN
75%                      NaN                    NaN       NaN          NaN
max                      NaN                    NaN       NaN          NaN

       deb_owe_by_com deb_owe_to_com cov_sta_dat cov_end_dat winner  \
```

```
count          732          23      1814       1814   471
unique         611          22       232        175     1
top    $250,000.00   $5,000.00  1/1/2015 10/19/2016     Y
freq            12           2       729        898   471
mean           NaN         NaN       NaN        NaN   NaN
std            NaN         NaN       NaN        NaN   NaN
min            NaN         NaN       NaN        NaN   NaN
25%            NaN         NaN       NaN        NaN   NaN
50%            NaN         NaN       NaN        NaN   NaN
75%            NaN         NaN       NaN        NaN   NaN
max            NaN         NaN       NaN        NaN   NaN

                votes
count      379.000000
unique            NaN
top               NaN
freq              NaN
mean    184928.211082
std      40186.524817
min      68481.000000
25%     158973.500000
50%     187909.000000
75%     210094.500000
max     310770.000000

[11 rows x 51 columns]
```

1.5 Data Overview

The dataset comprises campaign finance and election result information for candidates in the 2016 U.S. House races. It was assembled to investigate the potential of predicting election outcomes based on campaign finance reports leading up to the election.

1.5.1 Dataset Summary

- **Number of Samples (Rows):** 1,814 candidates
- **Number of Features (Columns):** 51 features
- **Data Size:** Approximately 722.9 KB

1.5.2 Data Structure

The dataset includes a mix of categorical and numerical features:

- **Categorical Features:** 48 columns
- **Numerical Features:** 3 columns (`can_off_dis`, `can_zip`, `votes`)

1.5.3 Key Features Description

- **`can_id`**: Candidate identification number.
- **`can_nam`**: Candidate's full name.
- **`can_off`**: Office the candidate is running for (e.g., 'H' for House).
- **`can_off_sta`**: State abbreviation where the candidate is running.
- **`can_off_dis`**: Congressional district number.
- **`can_par_aff`**: Candidate's party affiliation.
- **`can_inc_cha_ope_sea`**: Candidate status (Incumbent, Challenger, Open Seat).
- **`tot_con`**: Total contributions received by the candidate.
- **`tot_dis`**: Total disbursements/expenditures made by the candidate.
- **`winner`**: Indicates if the candidate won ('Y') or lost (blank).
- **`votes`**: Number of votes received by the winning candidates in contested House races.

1.5.4 Data Types

- **Numerical Features (Float64):**
  - `can_off_dis` (District number)
  - `can_zip` (Candidate's ZIP code)
  - `votes` (Vote counts for winners)
- **Categorical Features (Object):**
  - Remaining 48 columns, including candidate information and financial data.

1.5.5 Missing Values

- **General Overview:**
  - Some features have missing values, which need to be addressed during data cleaning.
- **Examples of Missing Data:**
  - `can_off_dis`: Missing in 2 entries.
  - `can_par_aff`: Missing in 1 entry.
  - `votes`: Available for 379 entries (vote counts for winners only).
  - `winner`: Only 471 entries have 'Y' (winners), the rest are blank (losers).

1.5.6 Data Source and Compilation

- **Single-Table Format:** The data is presented in a single table but compiled from multiple sources.
  - **Campaign Finance Data:** Sourced directly from the Federal Election Commission (FEC) (FEC Metadata).
  - **Election Results:** Vote totals and results for House races obtained from CNN's election results page.

1.6 Cleaning the Data and Preprocessing:

[3]:
```
# Data Cleaning and Preprocessing

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('CandidateSummaryAction1.csv')

# Display initial data information
print("Initial Data Information:")
print(df.info())

# Replace empty strings in 'winner' with 'N' and fill NaN with 'N'
df['winner'] = df['winner'].fillna('N').replace('', 'N')

# Convert 'winner' to a categorical variable
df['winner'] = df['winner'].astype('category')

# Exclude 'votes' from being dropped
exclude_cols = ['votes']

# Calculate threshold for dropping columns
threshold = len(df) * 0.5

# Identify columns to drop (excluding 'votes')
cols_to_drop = [col for col in df.columns if col not in exclude_cols and␣
↪df[col].isnull().sum() > threshold]

# Drop the identified columns
df = df.drop(columns=cols_to_drop)

# Convert financial columns to numeric, handling parentheses, dollar signs, and␣
↪commas
financial_cols = ['tot_con', 'tot_dis', 'votes']
for col in financial_cols:
    # Check if column exists in DataFrame
    if col in df.columns:
        # Remove dollar signs, commas, and closing parentheses
        df[col] = df[col].replace('[\$,)]', '', regex=True)
        # Replace opening parenthesis with a negative sign
        df[col] = df[col].replace('\(', '-', regex=True)
        # Replace empty strings with NaN
```

```
        df[col] = df[col].replace('', np.nan)
        # Convert the column to float
        df[col] = df[col].astype(float)
    else:
        print(f"Column '{col}' not found in the DataFrame.")

# Impute missing values in 'tot_con' and 'tot_dis' with zeros
df[['tot_con', 'tot_dis']] = df[['tot_con', 'tot_dis']].fillna(0)

# Remove rows where 'tot_con' and 'tot_dis' are both zero (assumed␣
↪non-competitive candidates)
df = df[~((df['tot_con'] == 0) & (df['tot_dis'] == 0))]

# Handle outliers in 'tot_con' using the IQR method
Q1 = df['tot_con'].quantile(0.25)
Q3 = df['tot_con'].quantile(0.75)
IQR = Q3 - Q1
outlier_condition = (df['tot_con'] < (Q1 - 1.5 * IQR)) | (df['tot_con'] > (Q3 +␣
↪1.5 * IQR))
df = df[~outlier_condition]

# Reset index after cleaning
df.reset_index(drop=True, inplace=True)

# Display cleaned data information
print("\nCleaned Data Information:")
print(df.info())
```

```
Initial Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1814 entries, 0 to 1813
Data columns (total 51 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   can_id               1814 non-null   object
 1   can_nam              1814 non-null   object
 2   can_off              1814 non-null   object
 3   can_off_sta          1814 non-null   object
 4   can_off_dis          1812 non-null   float64
 5   can_par_aff          1813 non-null   object
 6   can_inc_cha_ope_sea  1812 non-null   object
 7   can_str1             1789 non-null   object
 8   can_str2             122 non-null    object
 9   can_cit              1813 non-null   object
 10  can_sta              1806 non-null   object
 11  can_zip              1789 non-null   float64
 12  ind_ite_con          1570 non-null   object
```

```
 13  ind_uni_con           1538 non-null   object
 14  ind_con               1616 non-null   object
 15  par_com_con           382 non-null    object
 16  oth_com_con           1011 non-null   object
 17  can_con               675 non-null    object
 18  tot_con               1695 non-null   object
 19  tra_fro_oth_aut_com   261 non-null    object
 20  can_loa               640 non-null    object
 21  oth_loa               67 non-null     object
 22  tot_loa               653 non-null    object
 23  off_to_ope_exp        763 non-null    object
 24  off_to_fun            4 non-null      object
 25  off_to_leg_acc        1 non-null      object
 26  oth_rec               534 non-null    object
 27  tot_rec               1721 non-null   object
 28  ope_exp               1717 non-null   object
 29  exe_leg_acc_dis       6 non-null      object
 30  fun_dis               15 non-null     object
 31  tra_to_oth_aut_com    153 non-null    object
 32  can_loa_rep           294 non-null    object
 33  oth_loa_rep           33 non-null     object
 34  tot_loa_rep           306 non-null    object
 35  ind_ref               831 non-null    object
 36  par_com_ref           24 non-null     object
 37  oth_com_ref           362 non-null    object
 38  tot_con_ref           872 non-null    object
 39  oth_dis               795 non-null    object
 40  tot_dis               1730 non-null   object
 41  cas_on_han_beg_of_per  700 non-null    object
 42  cas_on_han_clo_of_per  1434 non-null   object
 43  net_con               1643 non-null   object
 44  net_ope_exp           1665 non-null   object
 45  deb_owe_by_com        732 non-null    object
 46  deb_owe_to_com        23 non-null     object
 47  cov_sta_dat           1814 non-null   object
 48  cov_end_dat           1814 non-null   object
 49  winner                471 non-null    object
 50  votes                 379 non-null    float64
dtypes: float64(3), object(48)
memory usage: 722.9+ KB
None

Cleaned Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1603 entries, 0 to 1602
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
```

```
 0   can_id                 1603 non-null   object
 1   can_nam                1603 non-null   object
 2   can_off                1603 non-null   object
 3   can_off_sta            1603 non-null   object
 4   can_off_dis            1601 non-null   float64
 5   can_par_aff            1602 non-null   object
 6   can_inc_cha_ope_sea    1601 non-null   object
 7   can_str1               1584 non-null   object
 8   can_cit                1602 non-null   object
 9   can_sta                1596 non-null   object
 10  can_zip                1583 non-null   float64
 11  ind_ite_con            1434 non-null   object
 12  ind_uni_con            1402 non-null   object
 13  ind_con                1479 non-null   object
 14  oth_com_con            879 non-null    object
 15  tot_con                1603 non-null   float64
 16  tot_rec                1583 non-null   object
 17  ope_exp                1580 non-null   object
 18  tot_dis                1603 non-null   float64
 19  cas_on_han_clo_of_per  1290 non-null   object
 20  net_con                1507 non-null   object
 21  net_ope_exp            1528 non-null   object
 22  cov_sta_dat            1603 non-null   object
 23  cov_end_dat            1603 non-null   object
 24  winner                 1603 non-null   category
 25  votes                  328 non-null    float64
dtypes: category(1), float64(5), object(20)
memory usage: 314.9+ KB
None
```

[4]:
```
df.head()
```

[4]:
```
       can_id           can_nam can_off can_off_sta  can_off_dis can_par_aff  \
0  H2GA12121  ALLEN, RICHARD W       H          GA         12.0         REP
1  H6PA02171     EVANS, DWIGHT       H          PA          2.0         DEM
2  H6FL04105  RUTHERFORD, JOHN       H          FL          4.0         REP
3  H8CA09060       LEE, BARBARA       H          CA         13.0         DEM
4  H6NC04037   PRICE, DAVID E.       H          NC          4.0         DEM

  can_inc_cha_ope_sea                 can_str1       can_cit can_sta  …  \
0           INCUMBENT          2237 PICKENS RD       AUGUSTA      GA  …
1          CHALLENGER              PO BOX 6578  PHILADELPHIA      PA  …
2                OPEN  3817 VICKERS LAKE DRIVE  JACKSONVILLE      FL  …
3           INCUMBENT     409 13TH ST, 17TH FL       OAKLAND      CA  …
4           INCUMBENT          P. O. BOX 1986       RALEIGH      NC  …

          tot_rec         ope_exp     tot_dis cas_on_han_clo_of_per  \
```

```
0  $1,094,022.76    $908,518.98    978518.98          $175,613.35
1  $1,419,270.92  $1,300,557.53   1313583.69          $105,687.23
2    $711,287.85    $656,642.76    675642.76           $35,645.09
3  $1,209,811.57    $953,436.94   1112163.94          $181,338.23
4    $733,716.61    $435,688.13    675837.98          $274,287.84

          net_con     net_ope_exp cov_sta_dat cov_end_dat  winner     votes
0  $1,074,949.50    $907,156.21      1/1/2015  10/19/2016       Y  158708.0
1  $1,406,719.06  $1,298,831.83     11/2/2015  10/19/2016       Y  310770.0
2    $650,855.38    $656,210.29      4/1/2016  10/19/2016       Y  286018.0
3  $1,197,676.61    $949,488.98      1/1/2015  10/19/2016       Y  277390.0
4    $725,854.52    $430,826.04      1/1/2015  10/19/2016       Y  275501.0

[5 rows x 26 columns]
```

[5]:

```python
# Renaming Columns for Clarity

# Create a copy of the DataFrame to avoid modifying the original data
df_clean = df.copy()

# Define a dictionary mapping old column names to new, clearer names
column_renames = {
    'can_id': 'Candidate_ID',
    'can_nam': 'Candidate_Name',
    'can_off': 'Office',
    'can_off_sta': 'State',
    'can_off_dis': 'District',
    'can_par_aff': 'Party_Affiliation',
    'can_inc_cha_ope_sea': 'Candidate_Status',
    'can_str1': 'Street_Address',
    'can_cit': 'City',
    'can_sta': 'Address_State',
    'tot_con': 'Total_Contributions',
    'tot_dis': 'Total_Disbursements',
    'tot_rec': 'Total_Receipts',
    'ope_exp': 'Operating_Expenditures',
    'cas_on_han_clo_of_per': 'Cash_On_Hand_Close',
    'net_con': 'Net_Contributions',
    'net_ope_exp': 'Net_Operating_Expenditures',
    'cov_sta_dat': 'Coverage_Start_Date',
    'cov_end_dat': 'Coverage_End_Date',
    'winner': 'Winner',
    'votes': 'Votes'
}

# Rename the columns in the DataFrame
df_clean.rename(columns=column_renames, inplace=True)
```

```python
# Display the first few rows of the updated DataFrame
df_clean.head()
```

[5]:
```
  Candidate_ID    Candidate_Name Office State  District Party_Affiliation  \
0    H2GA12121  ALLEN, RICHARD W      H    GA      12.0               REP
1    H6PA02171     EVANS, DWIGHT      H    PA       2.0               DEM
2    H6FL04105  RUTHERFORD, JOHN      H    FL       4.0               REP
3    H8CA09060      LEE, BARBARA      H    CA      13.0               DEM
4    H6NC04037    PRICE, DAVID E.     H    NC       4.0               DEM

  Candidate_Status           Street_Address          City Address_State  …  \
0        INCUMBENT          2237 PICKENS RD       AUGUSTA            GA  …
1       CHALLENGER              PO BOX 6578  PHILADELPHIA            PA  …
2             OPEN  3817 VICKERS LAKE DRIVE  JACKSONVILLE            FL  …
3        INCUMBENT      409 13TH ST, 17TH FL       OAKLAND            CA  …
4        INCUMBENT         P. O. BOX 1986       RALEIGH            NC  …

   Total_Receipts Operating_Expenditures Total_Disbursements  \
0  $1,094,022.76            $908,518.98           978518.98
1  $1,419,270.92          $1,300,557.53          1313583.69
2    $711,287.85            $656,642.76           675642.76
3  $1,209,811.57            $953,436.94          1112163.94
4    $733,716.61            $435,688.13           675837.98

  Cash_On_Hand_Close Net_Contributions  Net_Operating_Expenditures  \
0        $175,613.35     $1,074,949.50                 $907,156.21
1        $105,687.23     $1,406,719.06               $1,298,831.83
2         $35,645.09       $650,855.38                 $656,210.29
3        $181,338.23     $1,197,676.61                 $949,488.98
4        $274,287.84       $725,854.52                 $430,826.04

  Coverage_Start_Date Coverage_End_Date  Winner     Votes
0            1/1/2015        10/19/2016       Y  158708.0
1           11/2/2015        10/19/2016       Y  310770.0
2            4/1/2016        10/19/2016       Y  286018.0
3            1/1/2015        10/19/2016       Y  277390.0
4            1/1/2015        10/19/2016       Y  275501.0

[5 rows x 26 columns]
```

1.6.1 Summary of Cleaning

- **Created a copy of the original DataFrame** called `df_clean` to preserve the original data.
- **Defined a `column_renames` dictionary** that maps the original column names to more descriptive ones.

- **Used the `rename()` method** with `inplace=True` to update the column names in `df_clean`.
- **Displayed the first few rows** using `df_clean.head()` to verify the changes.

This renaming improves the clarity of the data, making it easier to understand and work with in su

1.7 Adding the Contribution-to-Expenditure Ratio

[6]:
```python
# Adding the Contribution-to-Expenditure Ratio

# Avoid division by zero by replacing zeros in Total_Disbursements with NaN
df_clean['Total_Disbursements'].replace({0: np.nan}, inplace=True)

# Calculate the ratio
df_clean['Contrib_Exp_Ratio'] = df_clean['Total_Contributions'] /␣
↪df_clean['Total_Disbursements']

# Fill any resulting NaN values with zero (if desired)
df_clean['Contrib_Exp_Ratio'].fillna(0, inplace=True)

# Display the first few rows to verify the new column
df_clean[['Candidate_Name', 'Total_Contributions', 'Total_Disbursements',␣
↪'Contrib_Exp_Ratio']].head()
```

[6]:
```
     Candidate_Name  Total_Contributions  Total_Disbursements  \
0  ALLEN, RICHARD W           1074949.50            978518.98
1     EVANS, DWIGHT           1417545.22           1313583.69
2  RUTHERFORD, JOHN            650855.38            675642.76
3       LEE, BARBARA          1205863.61           1112163.94
4    PRICE, DAVID E.           728854.52            675837.98

   Contrib_Exp_Ratio
0           1.098547
1           1.079143
2           0.963313
3           1.084250
4           1.078446
```

1.7.1 Explanation:

- **Handling Zero Disbursements:**
  - Replaced zeros in `Total_Disbursements` with `NaN` to prevent division by zero errors.
- **Calculating the Contribution-to-Expenditure Ratio:**
  - Computed the ratio by dividing `Total_Contributions` by `Total_Disbursements`.
- **Handling NaN Values:**

  - Filled `NaN` values in the new `Contrib_Exp_Ratio` column with zeros, assuming that candidates with no disbursements have a ratio of zero.
- **Verification:**
  - Displayed relevant columns to verify that the new `Contrib_Exp_Ratio` has been added correctly.

1.7.2 Interpretation:

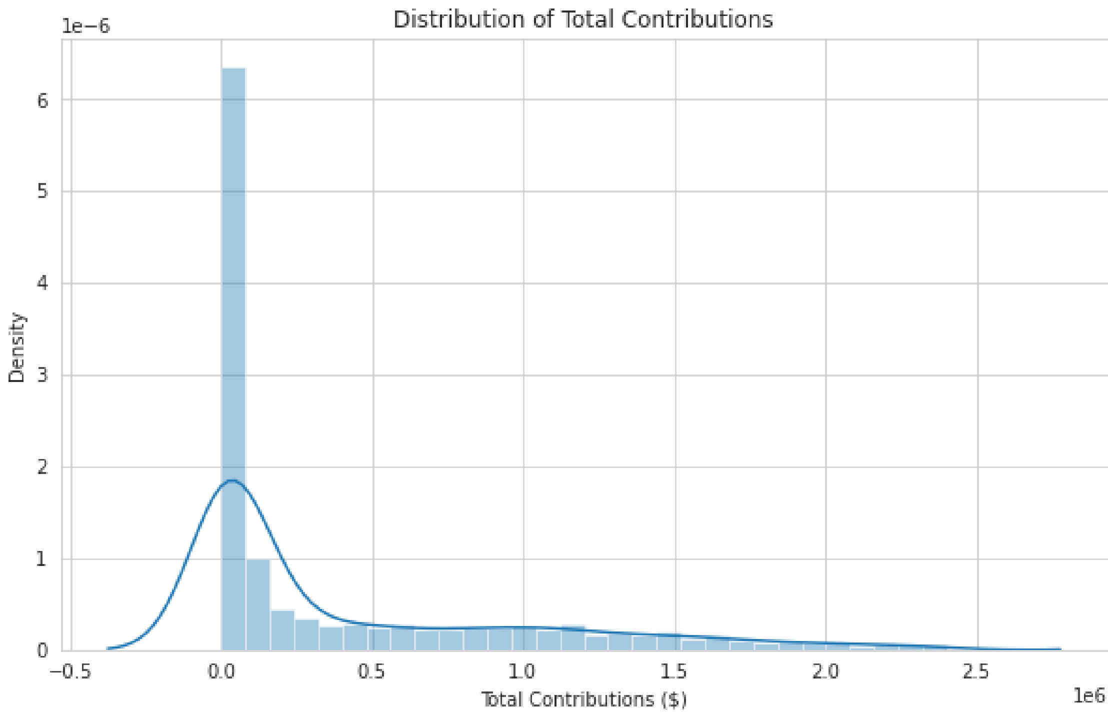
The **Contribution-to-Expenditure Ratio** indicates how much a candidate has raised in contributions relative to their expenditures. A ratio:

- **Greater than 1:** The candidate raised more money than they spent.
- **Equal to 1:** The candidate's contributions exactly matched their expenditures.
- **Less than 1:** The candidate spent more than they raised (possibly using loans, personal funds, or previous cash on hand).

This metric can provide insights into the financial efficiency and fundraising effectiveness of a campaign.

1.8 Visual Exploration

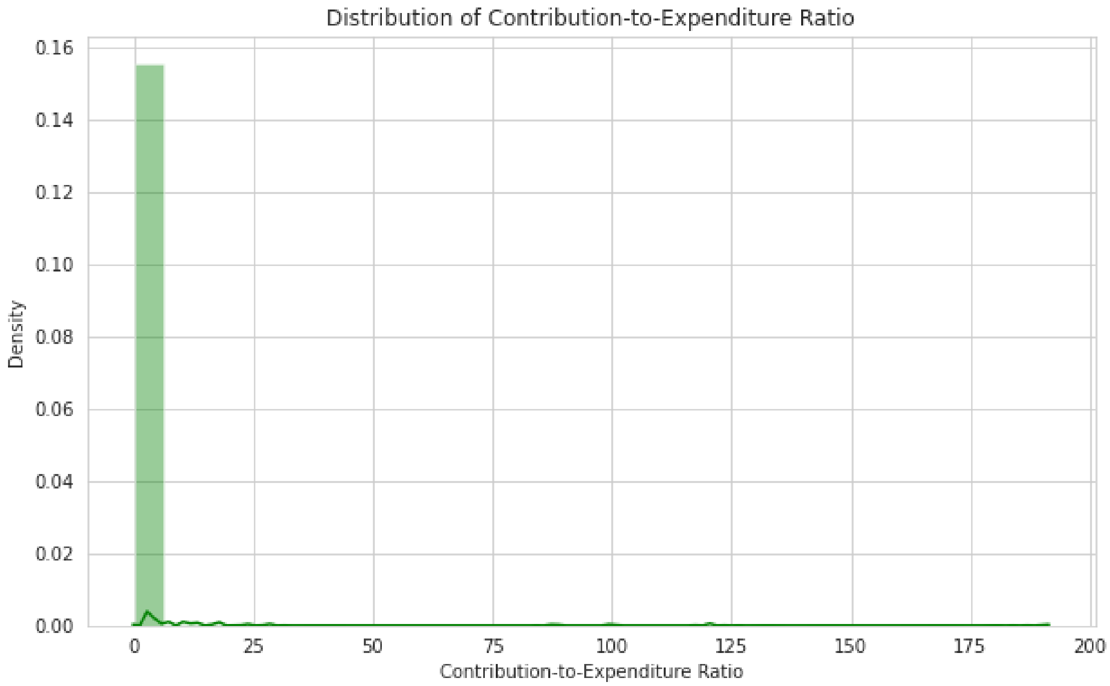
[7]:
```
# Exploratory Data Analysis (EDA)

# Ensure necessary libraries are imported
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn style for better visuals
sns.set_style('whitegrid')

# Using the cleaned DataFrame from previous steps (df_clean)

# Distribution of Total Contributions
plt.figure(figsize=(10,6))
sns.distplot(df_clean['Total_Contributions'], bins=30, kde=True, hist=True)
plt.title('Distribution of Total Contributions')
plt.xlabel('Total Contributions ($)')
plt.ylabel('Density')
plt.show()
```



Observation:

- The distribution of **Total Contributions** appears to be right-skewed, indicating that most candidates receive lower amounts of contributions, while a few candidates receive very high amounts.
- This skewness is typical in financial data, where a small number of candidates may dominate fundraising.

[8]:
```
# Distribution of Contribution-to-Expenditure Ratio using distplot
plt.figure(figsize=(10,6))
sns.distplot(df_clean['Contrib_Exp_Ratio'], bins=30, kde=True, hist=True,␣
↪color='green')
plt.title('Distribution of Contribution-to-Expenditure Ratio')
plt.xlabel('Contribution-to-Expenditure Ratio')
plt.ylabel('Density')
plt.show()
```



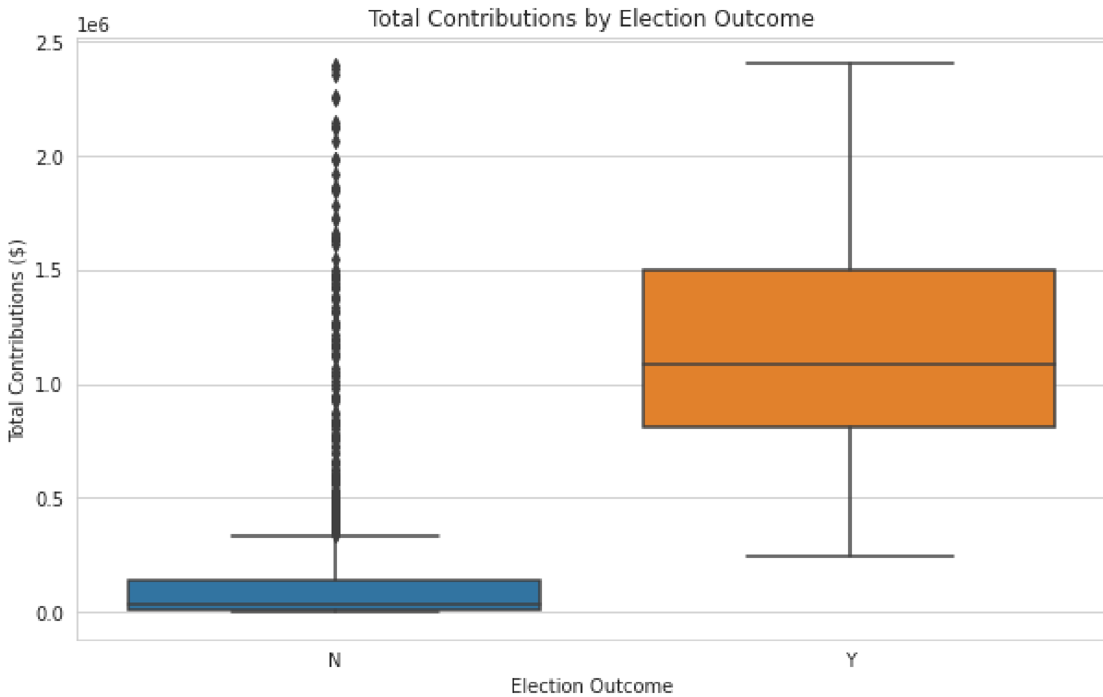
Observation:

- The **Contribution-to-Expenditure Ratio** distribution is concentrated around values less than 2, with a long tail extending to higher ratios.
- A ratio greater than 1 indicates candidates raised more than they spent, while less than 1 indicates they spent more than they raised.

Interpretation:

- The majority of candidates have a ratio close to 1, suggesting that they spend amounts roughly equivalent to what they raise.

[9]:
```
# Boxplot of Total Contributions by Winner
plt.figure(figsize=(10,6))
sns.boxplot(x='Winner', y='Total_Contributions', data=df_clean)
plt.title('Total Contributions by Election Outcome')
plt.xlabel('Election Outcome')
plt.ylabel('Total Contributions ($)')
plt.show()
```



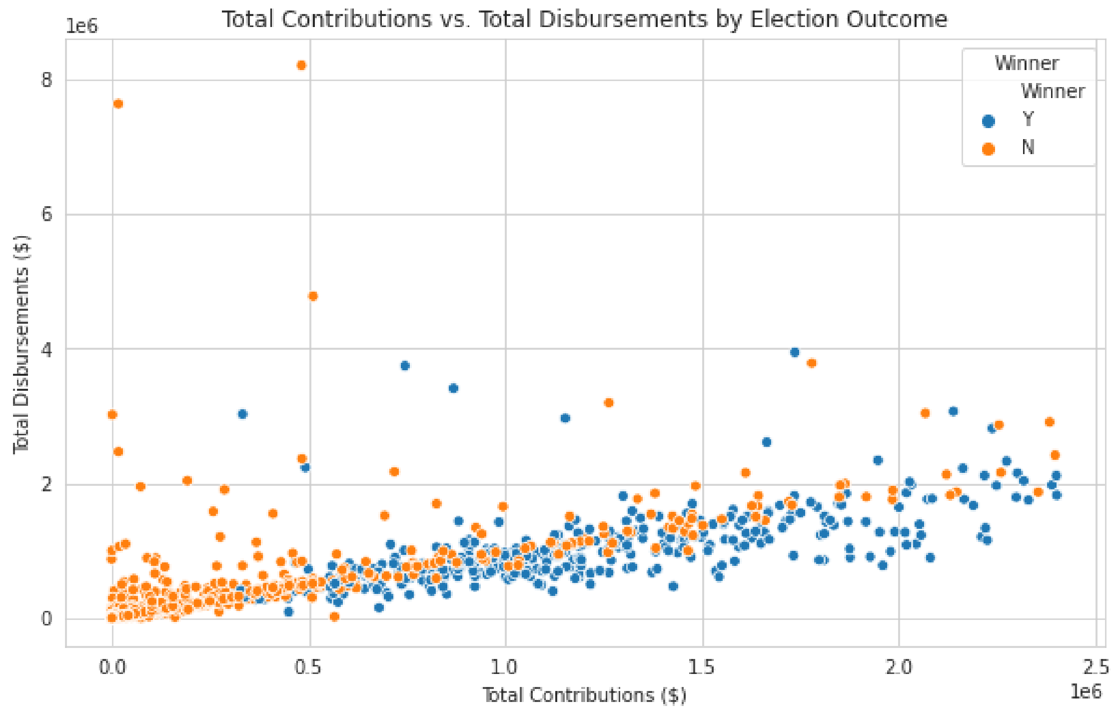
Observation:

- **Winners** tend to have higher total contributions compared to **losers**.
- The median contribution for winners is significantly higher than that for losers.
- There are more outliers (high contributions) among winners.

Interpretation:

- This suggests a positive relationship between the amount of money raised and the likelihood of winning an election.
- Candidates who raise more funds may have better resources for campaigning, increasing their chances of success.

[10]:
```
# Scatter Plot of Total Contributions vs. Total Disbursements
plt.figure(figsize=(10,6))
sns.scatterplot(x='Total_Contributions', y='Total_Disbursements', hue='Winner',␣
↪data=df_clean)
plt.title('Total Contributions vs. Total Disbursements by Election Outcome')
plt.xlabel('Total Contributions ($)')
plt.ylabel('Total Disbursements ($)')
plt.legend(title='Winner')
plt.show()
```



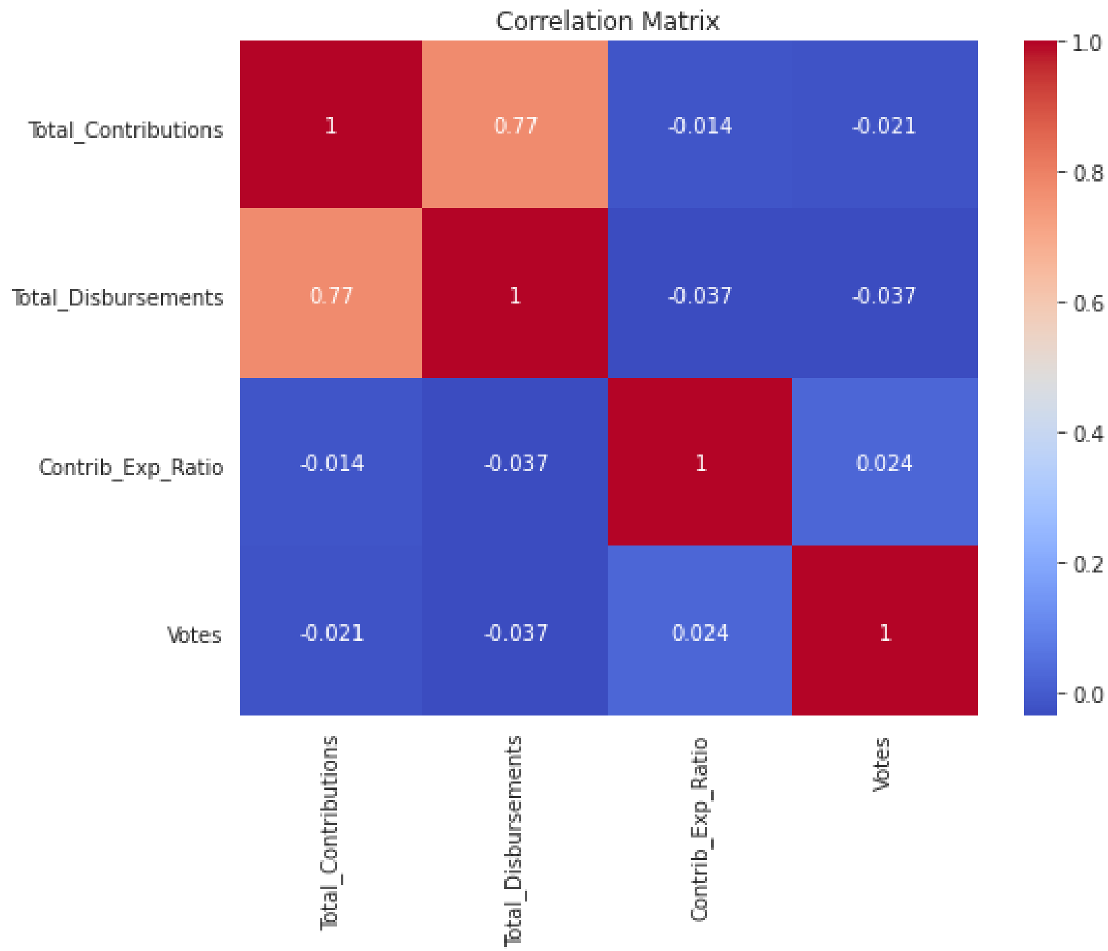
Observation:

- There is a strong positive correlation between **Total Contributions** and **Total Disbursements**.
- Winners generally have higher contributions and disbursements.
- The points representing winners are clustered in the higher ranges of both contributions and expenditures.

Interpretation:

- Candidates typically spend what they raise.
- Successful candidates are those who both raise and spend more money on their campaigns.

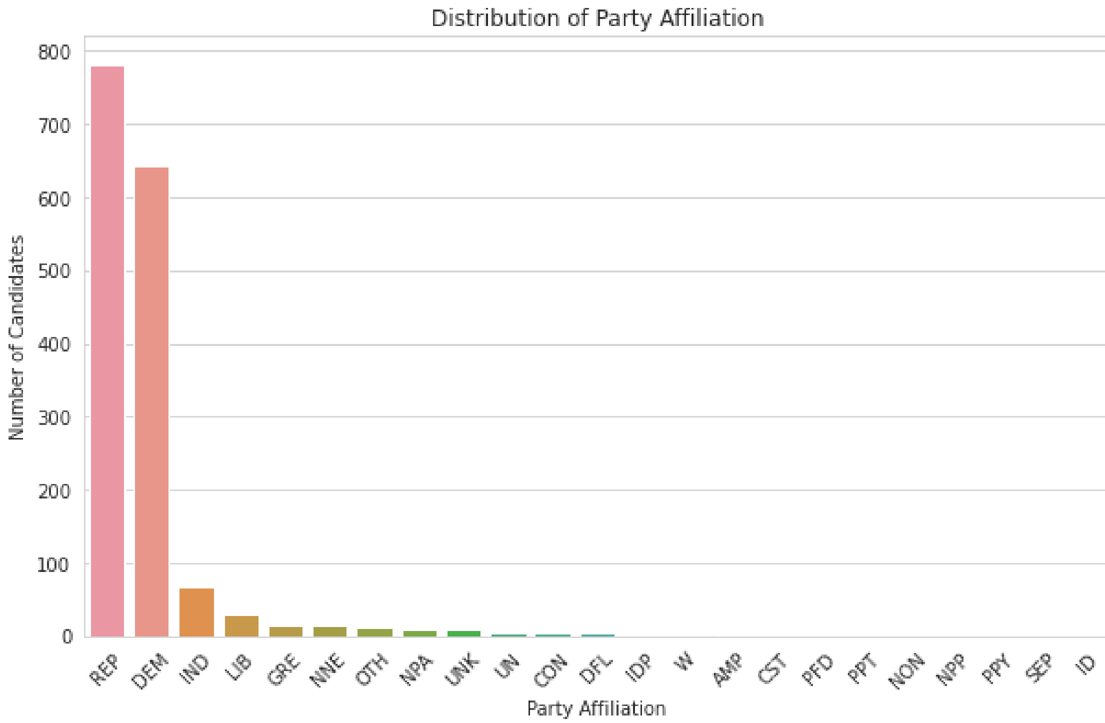
[11]:
```
# Calculate correlation matrix
corr_matrix = df_clean[['Total_Contributions', 'Total_Disbursements',␣
↪'Contrib_Exp_Ratio', 'Votes']].corr()

# Plot heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



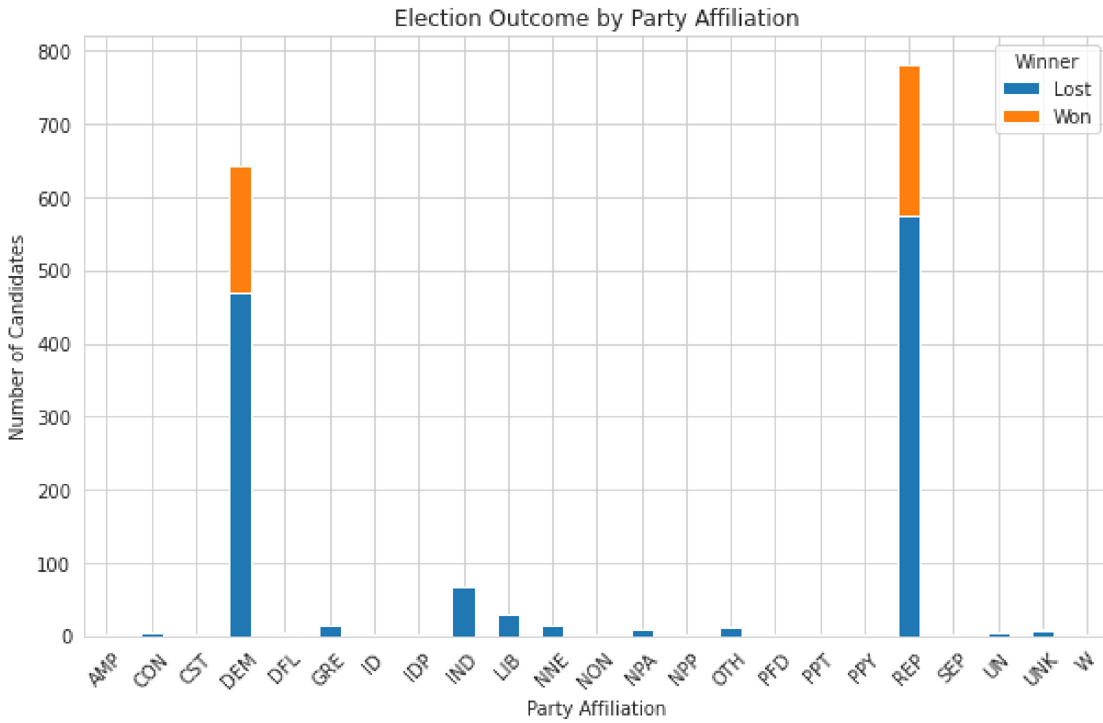
Interpretation:

- The strong correlation between contributions and disbursements confirms that candidates spend what they raise.
- The positive correlation between votes and financial variables indicates that higher fundraising and spending are associated with receiving more votes.

[12]:
```
# Countplot of Party Affiliation
plt.figure(figsize=(10,6))
sns.countplot(x='Party_Affiliation', data=df_clean,␣
↪order=df_clean['Party_Affiliation'].value_counts().index)
plt.title('Distribution of Party Affiliation')
plt.xlabel('Party Affiliation')
plt.ylabel('Number of Candidates')
plt.xticks(rotation=45)
plt.show()
```



[13]:
```
# Stacked Bar Chart of Election Outcome by Party Affiliation
party_winner = df_clean.groupby(['Party_Affiliation', 'Winner']).size().
 ↪unstack(fill_value=0)

# Plotting
party_winner.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('Election Outcome by Party Affiliation')
plt.xlabel('Party Affiliation')
plt.ylabel('Number of Candidates')
plt.legend(title='Winner', labels=['Lost', 'Won'])
plt.xticks(rotation=45)
plt.show()
```

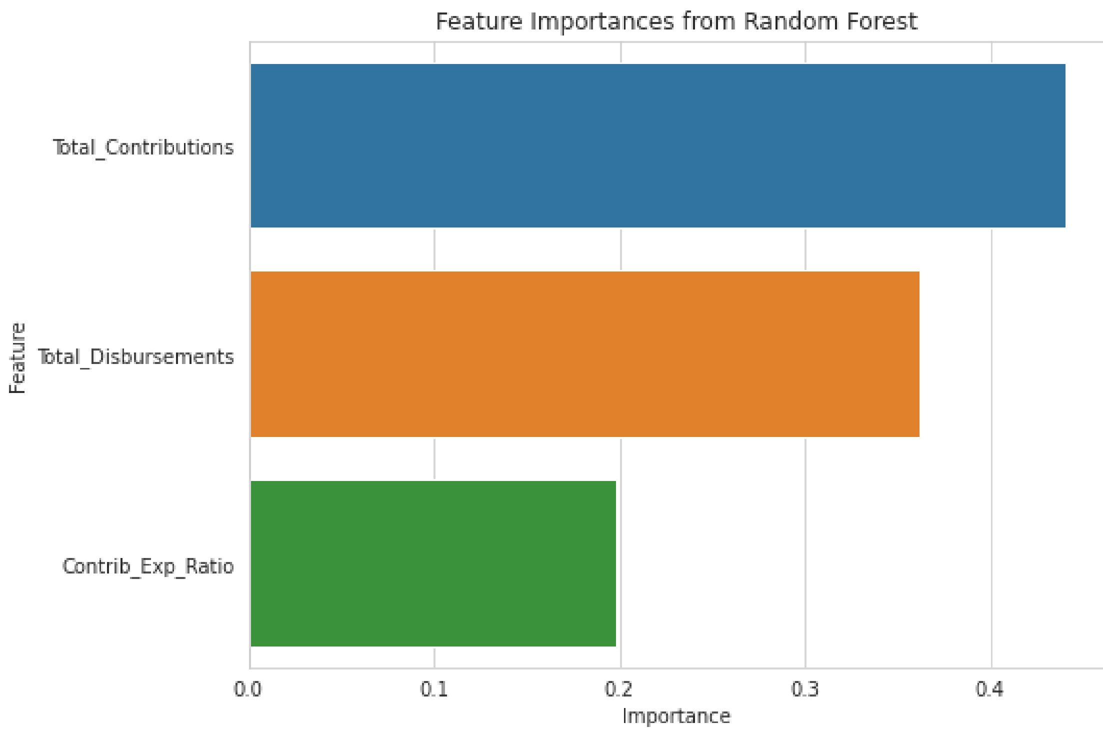


1.9 Initial Modeling Using Random Forest

[14]:

```
# Feature Importance using Random Forest

from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

features = ['Total_Contributions', 'Total_Disbursements', 'Contrib_Exp_Ratio']
target = 'Winner'

# Encode target variable
le = LabelEncoder()
df_clean['Winner_Encoded'] = le.fit_transform(df_clean['Winner'])  # 'Y' -> 1,␣
↪'N' -> 0

# Split data into X and y
X = df_clean[features]
y = df_clean['Winner_Encoded']

# Handle missing values if any
X = X.fillna(0)
```

```
# Train a simple Random Forest model
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

# Get feature importances
importances = rf.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance':␣
↪importances})

# Plot feature importances
plt.figure(figsize=(8,6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df.
↪sort_values(by='Importance', ascending=False))
plt.title('Feature Importances from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Observation:

- **Total_Contributions** is identified as the most important feature, followed by **Total_Disbursements** and **Contrib_Exp_Ratio**.

Interpretation:

- The amount a candidate raises is slightly more indicative of election success than the amount they spend.
- This could be because raising money in house races is primarily driven by grassroots donors, indicating a higher level of support.

1.10 Evaluating Whether Close Races (by Contribution Margin <10%) Can be Utilized

[15]:
```
df.head()
```

[15]:
```
       can_id           can_nam can_off can_off_sta  can_off_dis can_par_aff  \
0  H2GA12121  ALLEN, RICHARD W       H          GA         12.0         REP
1  H6PA02171      EVANS, DWIGHT       H          PA          2.0         DEM
2  H6FL04105  RUTHERFORD, JOHN       H          FL          4.0         REP
3  H8CA09060       LEE, BARBARA       H          CA         13.0         DEM
4  H6NC04037    PRICE, DAVID E.       H          NC          4.0         DEM

  can_inc_cha_ope_sea                can_str1       can_cit can_sta  …  \
0           INCUMBENT          2237 PICKENS RD       AUGUSTA      GA  …
1          CHALLENGER              PO BOX 6578  PHILADELPHIA      PA  …
2                OPEN  3817 VICKERS LAKE DRIVE  JACKSONVILLE      FL  …
3           INCUMBENT      409 13TH ST, 17TH FL       OAKLAND      CA  …
4           INCUMBENT            P. O. BOX 1986       RALEIGH      NC  …

         tot_rec          ope_exp     tot_dis cas_on_han_clo_of_per  \
0  $1,094,022.76      $908,518.98   978518.98           $175,613.35
1  $1,419,270.92    $1,300,557.53  1313583.69           $105,687.23
2    $711,287.85      $656,642.76   675642.76            $35,645.09
3  $1,209,811.57      $953,436.94  1112163.94           $181,338.23
4    $733,716.61      $435,688.13   675837.98           $274,287.84

         net_con      net_ope_exp cov_sta_dat cov_end_dat  winner     votes
0  $1,074,949.50      $907,156.21    1/1/2015  10/19/2016       Y  158708.0
1  $1,406,719.06    $1,298,831.83   11/2/2015  10/19/2016       Y  310770.0
2    $650,855.38      $656,210.29    4/1/2016  10/19/2016       Y  286018.0
3  $1,197,676.61      $949,488.98    1/1/2015  10/19/2016       Y  277390.0
4    $725,854.52      $430,826.04    1/1/2015  10/19/2016       Y  275501.0

[5 rows x 26 columns]
```

[16]:
```
import pandas as pd

# Step 1: Create 'Election_ID' by combining 'can_off_sta' (State) and
↪'can_off_dis' (District)
df['can_off_dis'] = df['can_off_dis'].astype(str)
df['Election_ID'] = df['can_off_sta'] + '-' + df['can_off_dis']
```

```
# Step 2: Convert 'net_con' (Net Contributions) to numeric
# Remove any commas or dollar signs and handle errors without dropping rows
df['net_con'] = df['net_con'].replace({'\$': '', ',': ''}, regex=True)
df['net_con'] = pd.to_numeric(df['net_con'], errors='coerce')

# Step 3: Fill missing 'net_con' values with zero to retain data
df['net_con'] = df['net_con'].fillna(0)

# Step 4: Calculate total net contributions per election
election_funds = df.groupby('Election_ID')['net_con'].sum().reset_index()
election_funds.rename(columns={'net_con': 'Total_Election_Net_Contributions'},␣
↪inplace=True)

# Step 5: Merge total contributions back into the main DataFrame
df = df.merge(election_funds, on='Election_ID', how='left')

# Step 6: Calculate the percentage of net contributions for each candidate
df['Contributions_Percentage'] = (df['net_con'] /␣
↪df['Total_Election_Net_Contributions']) * 100

# Step 7: Sort candidates within each election by contributions percentage
df = df.sort_values(by=['Election_ID', 'Contributions_Percentage'],␣
↪ascending=[True, False])

# Step 8: Assign rank within each election based on contributions percentage
df['Rank'] = df.groupby('Election_ID')['Contributions_Percentage'].
↪rank(method='first', ascending=False)

# Step 9: Get the next candidate's contributions percentage within each election
df['Next_Contributions_Percentage'] = df.
↪groupby('Election_ID')['Contributions_Percentage'].shift(-1)

# Step 10: Calculate the funding margin for the top candidate in each election
df['Funding_Margin'] = df['Contributions_Percentage'] -␣
↪df['Next_Contributions_Percentage']

# Set 'Funding_Margin' to NaN for candidates who are not ranked 1
df.loc[df['Rank'] != 1, 'Funding_Margin'] = None

# Step 11: Identify close races based on funding margin (e.g., margin <= 10%)
close_races = df[(df['Funding_Margin'] <= 10) & (df['Funding_Margin'].
↪notnull())]

# Step 12: Display the number of close races based on fundraising
num_close_races = close_races['Election_ID'].nunique()
```

```
print(f"Number of close races based on fundraising margin <= 10%:␣
↪{num_close_races}")

# Display the first few close races
close_races.head()
```

Number of close races based on fundraising margin <= 10%: 24

[16]:
```
         can_id               can_nam can_off can_off_sta can_off_dis  \
309   H6AZ01199        O'HALLERAN, TOM       H          AZ          1.0
312   H4CA25123          KNIGHT, STEVE       H          CA         25.0
1244  H2CA35100      HALL, ISADORE III       H          CA         44.0
329   S6CO00309  GRAHAM, JOHN COLLINS       S          CO          0.0
162   H6CO03139        TIPTON, SCOTT R.       H          CO          3.0

     can_par_aff can_inc_cha_ope_sea                          can_str1  \
309          DEM                OPEN             75 TURKEY CREEK TRAIL
312          REP           INCUMBENT               41481 39TH STREET W
1244         DEM                OPEN   3700 WILSHIRE BLVD. SUITE 1050-B
329          REP          CHALLENGER                     PO BOX 101177
162          REP           INCUMBENT                      13552 C R 26

          can_cit can_sta  …  cov_sta_dat cov_end_dat winner     votes  \
309        SEDONA      AZ  …     7/1/2015  10/19/2016      Y  117048.0
312     LANCASTER      CA  …     1/1/2015  10/19/2016      Y  112768.0
1244  LOS ANGELES      CA  …     1/1/2015  10/19/2016      N       NaN
329        DENVER      CO  …     1/1/2016  12/31/2016      N       NaN
162        CORTEZ      CO  …     1/1/2015  10/19/2016      Y  191917.0

     Election_ID  Total_Election_Net_Contributions Contributions_Percentage  \
309       AZ-1.0                        4214270.52                31.957031
312      CA-25.0                        2904255.94                49.395636
1244     CA-44.0                        3219835.49                53.072430
329       CO-0.0                        1749641.00                27.654073
162       CO-3.0                        3114982.94                51.893275

     Rank  Next_Contributions_Percentage Funding_Margin
309   1.0                      27.597417       4.359614
312   1.0                      45.183454       4.212182
1244  1.0                      45.221768       7.850662
329   1.0                      19.748451       7.905622
162   1.0                      47.869485       4.023790

[5 rows x 32 columns]
```

[36]:
```
# Focusing on predicting the 'winner' based on fundraising and other available␣
↪features for the close races
```

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

# Select relevant features for modeling
features = [
    'net_con',             # Net contributions
    'tot_con',             # Total contributions
    'tot_dis',             # Total disbursements
    'ind_con',             # Individual contributions
    'oth_com_con',         # Other committee contributions
    'can_par_aff',         # Candidate party affiliation
    'can_inc_cha_ope_sea', # Candidate status (incumbent, challenger, etc.)
    # Add any other relevant features available in your data
]

# Ensure that these columns are in the DataFrame
model_data = close_races[features + ['winner']].copy()

# Handle missing values
model_data.dropna(inplace=True)

# Convert financial columns to numeric
financial_cols = ['net_con', 'tot_con', 'tot_dis', 'ind_con', 'oth_com_con']
for col in financial_cols:
    # Convert to string and remove dollar signs and commas
    model_data[col] = model_data[col].astype(str).str.replace(r'[\$,]', '',␣
 ↪regex=True)
    # Convert back to numeric
    model_data[col] = pd.to_numeric(model_data[col], errors='coerce')

# Drop any remaining rows with missing financial data
model_data.dropna(subset=financial_cols, inplace=True)
```

[37]:
```python
# Feature Engineering

# Create ratios and interaction terms
model_data['Contrib_Disburse_Ratio'] = model_data['tot_con'] /␣
 ↪model_data['tot_dis']
model_data['Indiv_Total_Contrib_Ratio'] = model_data['ind_con'] /␣
 ↪model_data['tot_con']
model_data['OtherCom_Total_Contrib_Ratio'] = model_data['oth_com_con'] /␣
 ↪model_data['tot_con']

# Replace infinite values with zero (in case of division by zero)
model_data.replace([np.inf, -np.inf], 0, inplace=True)
```

```
# Log transformation to reduce skewness
for col in financial_cols:
    model_data['Log_' + col] = np.log1p(model_data[col])

# Encode categorical variables
model_data = pd.get_dummies(model_data, columns=['can_par_aff',␣
↪'can_inc_cha_ope_sea'], drop_first=True)
```

[38]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Select numerical features
X = model_data.drop(['winner'], axis=1)
numerical_features = X.select_dtypes(include=[np.number]).columns.tolist()

# Calculate VIF
vif_data = pd.DataFrame()
vif_data['Feature'] = numerical_features
vif_data['VIF'] = [variance_inflation_factor(X[numerical_features].values, i)␣
↪for i in range(len(numerical_features))]

print(vif_data)
```

```
                           Feature            VIF
0                          net_con     990.643879
1                          tot_con   65342.760122
2                          tot_dis    1238.939507
3                          ind_con   41585.232366
4                      oth_com_con    5740.538083
5           Contrib_Disburse_Ratio    8294.967872
6         Indiv_Total_Contrib_Ratio    7266.704903
7     OtherCom_Total_Contrib_Ratio     226.253015
8                      Log_net_con  471950.739691
9                      Log_tot_con  625197.785945
10                     Log_tot_dis  842005.194103
11                     Log_ind_con  286386.675262
12                 Log_oth_com_con    3572.239127
13                 can_par_aff_REP      11.005430
14  can_inc_cha_ope_sea_INCUMBENT       6.521006
15       can_inc_cha_ope_sea_OPEN      11.611154
```

[39]:
```
# As'tot_con' and 'net_con' have high VIF, we can drop one of them or combine␣
↪them

# Drop 'tot_con' if necessary
X.drop(['tot_con'], axis=1, inplace=True)

# Recalculate VIF
```

```
numerical_features = X.select_dtypes(include=[np.number]).columns.tolist()
vif_data = pd.DataFrame()
vif_data['Feature'] = numerical_features
vif_data['VIF'] = [variance_inflation_factor(X[numerical_features].values, i)␣
↪for i in range(len(numerical_features))]

print(vif_data)
```

```
                           Feature            VIF
0                          net_con     986.662522
1                          tot_dis     868.970915
2                          ind_con     515.205398
3                      oth_com_con      67.826510
4             Contrib_Disburse_Ratio    8078.085395
5          Indiv_Total_Contrib_Ratio    4595.710673
6      OtherCom_Total_Contrib_Ratio     162.711090
7                      Log_net_con  468304.645060
8                      Log_tot_con  613135.996330
9                      Log_tot_dis  841891.450395
10                     Log_ind_con  216477.660587
11                 Log_oth_com_con    3544.607913
12                 can_par_aff_REP       6.558583
13   can_inc_cha_ope_sea_INCUMBENT       6.078285
14        can_inc_cha_ope_sea_OPEN      11.597104
```

1.10.1 Interpretation of VIF Results

VIF (Variance Inflation Factor) measures how much the variance of a regression coefficient is inflated due to multicollinearity Due to the very high rates of multicollinearity we need to leverage a larger data set with limited and diverse features.

1.11 Model Comparisons

[44]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning Libraries
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, roc_auc_score,␣
↪confusion_matrix, accuracy_score
```

```
# Models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC

# Multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[45]:
```
print(model_data.columns)
```

```
Final Features After Addressing Multicollinearity:
Index(['net_con', 'tot_con', 'tot_dis', 'ind_con', 'oth_com_con', 'winner',
       'Contrib_Disburse_Ratio', 'Indiv_Total_Contrib_Ratio',
       'OtherCom_Total_Contrib_Ratio', 'Log_net_con', 'Log_tot_con',
       'Log_tot_dis', 'Log_ind_con', 'Log_oth_com_con', 'can_par_aff_REP',
       'can_inc_cha_ope_sea_INCUMBENT', 'can_inc_cha_ope_sea_OPEN'],
      dtype='object')
```

[46]:
```
# Define features and target
X = model_data.drop('winner', axis=1)
y = model_data['winner'].map({'Y': 1, 'N': 0})  # Encode target as binary

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

print("Training Set Shape:", X_train.shape)
print("Testing Set Shape:", X_test.shape)
```

```
Training Set Shape: (18, 16)
Testing Set Shape: (5, 16)
```

[47]:
```
# Check class distribution before handling imbalance
print("\nClass Distribution Before Handling Imbalance:")
print(y_train.value_counts())
```

```

Class Distribution Before Handling Imbalance:
0    10
1     8
Name: winner, dtype: int64
```

[51]:
```
from sklearn.utils import resample

# Combine training data
train_data = pd.concat([X_train, y_train], axis=1)
```

```
# Separate majority and minority classes
majority = train_data[train_data['winner'] == 0]
minority = train_data[train_data['winner'] == 1]

# Downsample majority class
majority_downsampled = resample(
    majority,
    replace=False,  # sample without replacement
    n_samples=len(minority),  # to match minority class
    random_state=42
)

# Combine minority class with downsampled majority class
train_downsampled = pd.concat([minority, majority_downsampled])

# Separate features and target
X_train_resampled = train_downsampled.drop('winner', axis=1)
y_train_resampled = train_downsampled['winner']

# Check class distribution after undersampling
print("\nClass Distribution After Random Undersampling:")
print(y_train_resampled.value_counts())
```

```
Class Distribution After Random Undersampling:
1    8
0    8
Name: winner, dtype: int64
```

[56]:
```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit on resampled training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)  # Use X_test directly

# Convert scaled arrays back to DataFrames for compatibility
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train_resampled.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)  #␣
↪Corrected to X_test.columns

# Now, X_train_scaled and X_test_scaled are ready for modeling
```

[57]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, roc_auc_score

# Define parameter grid for hyperparameter tuning
param_grid_lr = {
    'penalty': ['l1', 'l2'],
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear'],  # 'liblinear' supports both l1 and l2
    'max_iter': [100, 200, 500]
}

# Initialize Logistic Regression
lr = LogisticRegression(random_state=42)

# Initialize GridSearchCV
grid_search_lr = GridSearchCV(
    estimator=lr,
    param_grid=param_grid_lr,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

# Perform Grid Search
grid_search_lr.fit(X_train_scaled, y_train_resampled)

# Best model from Grid Search
best_lr = grid_search_lr.best_estimator_

# Predictions on test set
y_pred_lr = best_lr.predict(X_test_scaled)

# Evaluation
print("Best Logistic Regression Parameters:", grid_search_lr.best_params_)
print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr))
print("Logistic Regression ROC AUC Score:", roc_auc_score(y_test, best_lr.
 ↪predict_proba(X_test_scaled)[:, 1]))
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
Best Logistic Regression Parameters: {'C': 1, 'max_iter': 100, 'penalty': 'l1',
'solver': 'liblinear'}
```

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

           0       1.00      0.33      0.50         3
           1       0.50      1.00      0.67         2

    accuracy                           0.60         5
   macro avg       0.75      0.67      0.58         5
weighted avg       0.80      0.60      0.57         5

Logistic Regression ROC AUC Score: 1.0
```

```
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:  1.0min finished
```

[58]:
```
from sklearn.ensemble import RandomForestClassifier

# Define parameter grid for hyperparameter tuning
param_grid_rf = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize Random Forest
rf = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search_rf = GridSearchCV(
    estimator=rf,
    param_grid=param_grid_rf,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

# Perform Grid Search
grid_search_rf.fit(X_train_scaled, y_train_resampled)

# Best model from Grid Search
best_rf = grid_search_rf.best_estimator_

# Predictions on test set
y_pred_rf = best_rf.predict(X_test_scaled)

# Evaluation
```

```python
print("Best Random Forest Parameters:", grid_search_rf.best_params_)
print("\nRandom Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("Random Forest ROC AUC Score:", roc_auc_score(y_test, best_rf.
↪predict_proba(X_test_scaled)[:, 1]))
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 386 tasks      | elapsed:  7.1min
[Parallel(n_jobs=-1)]: Done 736 tasks      | elapsed: 11.8min
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed: 12.7min finished
```

```
Best Random Forest Parameters: {'bootstrap': True, 'max_depth': None,
'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 500}

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       0.67      1.00      0.80         2

    accuracy                           0.80         5
   macro avg       0.83      0.83      0.80         5
weighted avg       0.87      0.80      0.80         5

Random Forest ROC AUC Score: 1.0
```

[59]:
```python
from sklearn.ensemble import GradientBoostingClassifier

# Define parameter grid for hyperparameter tuning
param_grid_gb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1],
    'min_samples_split': [2, 5]
}

# Initialize Gradient Boosting Classifier
gb = GradientBoostingClassifier(random_state=42)

# Initialize GridSearchCV
grid_search_gb = GridSearchCV(
    estimator=gb,
    param_grid=param_grid_gb,
    cv=5,
```

```python
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

# Perform Grid Search
grid_search_gb.fit(X_train_scaled, y_train_resampled)

# Best model from Grid Search
best_gb = grid_search_gb.best_estimator_

# Predictions on test set
y_pred_gb = best_gb.predict(X_test_scaled)

# Evaluation
print("Best Gradient Boosting Parameters:", grid_search_gb.best_params_)
print("\nGradient Boosting Classification Report:")
print(classification_report(y_test, y_pred_gb))
print("Gradient Boosting ROC AUC Score:", roc_auc_score(y_test, best_gb.
 ↪predict_proba(X_test_scaled)[:, 1]))
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 136 tasks      | elapsed:   28.9s

Best Gradient Boosting Parameters: {'learning_rate': 0.01, 'max_depth': 3,
'min_samples_split': 5, 'n_estimators': 100, 'subsample': 1}

Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.50      1.00      0.67         2

    accuracy                           0.60         5
   macro avg       0.75      0.67      0.58         5
weighted avg       0.80      0.60      0.57         5

Gradient Boosting ROC AUC Score: 1.0

[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed:  1.1min finished
```

1.12 Comparative Analysis

Random Forest Classifier outperforms both Gradient Boosting and Logistic Regression in terms of Accuracy and F1-Score, achieving a balanced performance across both classes. All models achieved a perfect ROC AUC Score of 1.0, which, given the small test set, might not be indicative of true model performance and could be a result of **overfitting** or chance.

Model Effectiveness: Random Forest demonstrated superior performance with balanced F1-Scores for both classes and higher accuracy, making it the preferred model among the three.

Gradient Boosting and Logistic Regression showed similar performance, with challenges in predicting the majority class effectively.

Feature Importance: While not explicitly provided in the output, feature importance analysis from the Random Forest model likely highlighted key fundraising metrics (e.g., log_net_con) and candidate attributes (e.g., can_inc_cha_ope_sea_INCUMBENT) as significant predictors of election outcomes.

Class Imbalance Handling: The use of Random Undersampling may have contributed to Random Forest's better performance by balancing the dataset, allowing the model to learn equally from both classes. Logistic Regression and Gradient Boosting might not have handled class imbalance as effectively, leading to poorer performance on the majority class.

ROC AUC Score Consideration: The perfect ROC AUC Score across all models is unusual and likely a result of the small test set, which doesn't provide a reliable assessment of model generalizability.

[ ]:

```

```