In [1]:
```
import pandas as pd
import numpy as np
from IPython.display import display, HTML

# Load the data
df = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/Fundraising.csv"
)
dff = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/FutureFundraising.csv"
)

# Display both DataFrames side by side
display(HTML("<h3>Fundraising DataFrame</h3>"))
display(df.head())

display(HTML("<h3>Future Fundraising DataFrame</h3>"))
display(dff.head())
```

```
/var/folders/jw/4t4swxld5c5f_5xhv0_bzbr00000gn/T/ipykernel_47790/739611249.py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

Fundraising DataFrame

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	totalmonths	TIMELAG	AVGGIFT	TARG
0	1	17	0	1	0	0	1	1	5	1	...	1	74	102.0	6.0	5.0	29	3	4.857143	
1	2	25	1	0	0	0	1	1	1	0	...	4	46	94.0	12.0	12.0	34	6	9.400000	
2	3	29	0	0	0	1	0	2	5	1	...	13	32	30.0	10.0	5.0	29	7	4.285714	
3	4	38	0	0	0	1	1	1	3	0	...	4	94	177.0	10.0	8.0	30	3	7.080000	
4	5	40	0	1	0	0	1	1	4	0	...	7	20	23.0	11.0	11.0	30	6	7.666667	

5 rows × 24 columns

Future Fundraising DataFrame

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	totalmonths	TIMELAG	AVGGIFT	TAR
0	1	3	0	1	0	0	1	1	1	1	...	3	42	92.0	29.0	15.0	17	8	15.333333	
1	2	4	0	0	1	0	0	1	2	1	...	4	21	30.0	20.0	20.0	33	9	15.000000	
2	3	5	0	0	0	1	0	1	1	0	...	10	61	220.0	35.0	25.0	31	9	24.444444	
3	4	1	0	0	0	0	1	1	4	0	...	21	32	41.0	19.0	19.0	31	13	13.666667	
4	5	4	0	0	1	0	1	1	7	1	...	1	47	46.0	10.0	10.0	28	8	5.750000	

5 rows × 24 columns

In [2]:

```
import pandas as pd
import numpy as np
from IPython.display import display, HTML

# Load the data
df = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/Fundraising.csv"
)
dff = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/FutureFundraising.csv"
)

# 1. Understand the Data
display(HTML("<h3>First 5 rows of the Fundraising dataset:</h3>"))
display(df.head())

display(HTML("<h3>Data types and missing values in Fundraising dataset:</h3>"))
display(df.info())

display(HTML("<h3>Summary statistics of Fundraising dataset:</h3>"))
display(df.describe())

# 2. Handle Missing Values
display(HTML("<h3>Missing values per column in Fundraising dataset:</h3>"))
display(df.isnull().sum())

# Fill missing values with the mean
df.fillna(df.mean(), inplace=True)

# Handle the dff dataframe
display(HTML("<h3>First 5 rows of the Future Fundraising dataset:</h3>"))
display(dff.head())

display(HTML("<h3>Data types and missing values in Future Fundraising dataset:</h3>"))
display(dff.info())

display(HTML("<h3>Summary statistics of Future Fundraising dataset:</h3>"))
display(dff.describe())

display(HTML("<h3>Missing values per column in Future Fundraising dataset:</h3>"))
display(dff.isnull().sum())

# Fill missing values in dff with the mean
dff.fillna(dff.mean(), inplace=True)
```

First 5 rows of the Fundraising dataset:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	totalmonths	TIMELAG	AVGGIFT	TARG
0	1	17	0	1	0	0	1	1	5	1	...	1	74	102.0	6.0	5.0	29	3	4.857143	
1	2	25	1	0	0	0	1	1	1	0	...	4	46	94.0	12.0	12.0	34	6	9.400000	
2	3	29	0	0	0	1	0	2	5	1	...	13	32	30.0	10.0	5.0	29	7	4.285714	
3	4	38	0	0	0	1	1	1	3	0	...	4	94	177.0	10.0	8.0	30	3	7.080000	
4	5	40	0	1	0	0	1	1	4	0	...	7	20	23.0	11.0	11.0	30	6	7.666667	

5 rows × 24 columns

Data types and missing values in Fundraising dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3120 entries, 0 to 3119
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Row Id           3120 non-null   int64  
 1   Row Id.          3120 non-null   int64  
 2   zipconvert_2     3120 non-null   int64  
 3   zipconvert_3     3120 non-null   int64  
 4   zipconvert_4     3120 non-null   int64  
 5   zipconvert_5     3120 non-null   int64  
 6   homeowner dummy  3120 non-null   int64  
 7   NUMCHLD          3120 non-null   int64  
 8   INCOME           3120 non-null   int64  
 9   gender dummy     3120 non-null   int64  
 10  WEALTH           3120 non-null   int64  
 11  HV               3120 non-null   int64  
 12  Icmed            3120 non-null   int64  
 13  Icavg            3120 non-null   int64  
 14  IC15             3120 non-null   int64  
 15  NUMPROM          3120 non-null   int64  
 16  RAMNTALL         3120 non-null   float64
 17  MAXRAMNT         3120 non-null   float64
 18  LASTGIFT         3120 non-null   float64
 19  totalmonths      3120 non-null   int64  
 20  TIMELAG          3120 non-null   int64  
 21  AVGGIFT          3120 non-null   float64
 22  TARGET_B         3120 non-null   int64  
 23  TARGET_D         3120 non-null   float64
dtypes: float64(5), int64(19)
memory usage: 585.1 KB
None
```

Summary statistics of Fundraising dataset:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LAS
count	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	...	3120.000000	3120.000000	3120.000000	3120.000000	3120.00
mean	1560.500000	11615.770833	0.214423	0.185256	0.214423	0.384615	0.770192	1.069231	3.893910	0.609295	...	14.702885	49.089423	110.399875	16.651397	13.5
std	900.810746	6698.678131	0.410487	0.388568	0.410487	0.486582	0.420777	0.347688	1.636186	0.487987	...	12.079882	22.717130	147.299933	22.223521	10.58
min	1.000000	17.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	...	0.000000	11.000000	15.000000	5.000000	0.00
25%	780.750000	5820.750000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	3.000000	0.000000	...	5.000000	29.000000	45.000000	10.000000	7.00
50%	1560.500000	11735.500000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	4.000000	1.000000	...	12.000000	48.000000	81.000000	15.000000	10.00
75%	2340.250000	17435.750000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	5.000000	1.000000	...	21.000000	65.000000	134.625000	20.000000	16.00
max	3120.000000	23293.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.000000	7.000000	1.000000	...	90.000000	157.000000	5674.900000	1000.000000	219.00

8 rows × 24 columns

Missing values per column in Fundraising dataset:

```
Row Id             0
Row Id.            0
zipconvert_2       0
zipconvert_3       0
zipconvert_4       0
zipconvert_5       0
homeowner dummy    0
NUMCHLD            0
INCOME             0
gender dummy       0
WEALTH             0
HV                 0
Icmed              0
Icavg              0
IC15               0
NUMPROM            0
RAMNTALL           0
MAXRAMNT           0
LASTGIFT           0
totalmonths        0
TIMELAG            0
AVGGIFT            0
TARGET_B           0
TARGET_D           0
dtype: int64
```

First 5 rows of the Future Fundraising dataset:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	totalmonths	TIMELAG	AVGGIFT	TAR
0	1	3	0	1	0	0	1	1	1	1	...	3	42	92.0	29.0	15.0	17	8	15.333333	
1	2	4	0	0	1	0	0	1	2	1	...	4	21	30.0	20.0	20.0	33	9	15.000000	
2	3	5	0	0	0	1	0	1	1	0	...	10	61	220.0	35.0	25.0	31	9	24.444444	
3	4	1	0	0	0	0	1	1	4	0	...	21	32	41.0	19.0	19.0	31	13	13.666667	
4	5	4	0	0	1	0	1	1	7	1	...	1	47	46.0	10.0	10.0	28	8	5.750000	

5 rows × 24 columns

Data types and missing values in Future Fundraising dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Row Id           2000 non-null   int64  
 1   Row Id.          2000 non-null   int64  
 2   zipconvert_2     2000 non-null   int64  
 3   zipconvert_3     2000 non-null   int64  
 4   zipconvert_4     2000 non-null   int64  
 5   zipconvert_5     2000 non-null   int64  
 6   homeowner dummy  2000 non-null   int64  
 7   NUMCHLD          2000 non-null   int64  
 8   INCOME           2000 non-null   int64  
 9   gender dummy     2000 non-null   int64  
 10  WEALTH           2000 non-null   int64  
 11  HV               2000 non-null   int64  
 12  Icmed            2000 non-null   int64  
 13  Icavg            2000 non-null   int64  
 14  IC15             2000 non-null   int64  
 15  NUMPROM          2000 non-null   int64  
 16  RAMNTALL         2000 non-null   float64
 17  MAXRAMNT         2000 non-null   float64
 18  LASTGIFT         2000 non-null   float64
 19  totalmonths      2000 non-null   int64  
 20  TIMELAG          2000 non-null   int64  
 21  AVGGIFT          2000 non-null   float64
 22  TARGET_B         0 non-null      float64
 23  TARGET_D         0 non-null      float64
dtypes: float64(6), int64(18)
memory usage: 375.1 KB
None
```

Summary statistics of Future Fundraising dataset:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LAS
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00000	2000.00000	2000.000000	2000.000000	2000.000000	...	2000.000000	2000.000000	2000.000000	2000.000000	2000.0(
mean	1000.500000	3.482500	0.237500	0.168500	0.230000	0.30450	0.77850	1.052000	3.821500	0.595500	...	15.177000	47.363000	103.005425	19.270305	16.4
std	577.494589	1.300592	0.425658	0.374403	0.420938	0.46031	0.41536	0.278091	1.638591	0.490918	...	12.577988	22.963991	98.844943	11.752492	9.7
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.00000	0.00000	1.000000	1.000000	0.000000	...	0.000000	4.000000	15.000000	5.000000	0.0(
25%	500.750000	2.000000	0.000000	0.000000	0.000000	0.00000	1.00000	1.000000	2.750000	0.000000	...	5.750000	27.000000	42.000000	14.000000	10.0(
50%	1000.500000	4.000000	0.000000	0.000000	0.000000	0.00000	1.00000	1.000000	4.000000	1.000000	...	12.000000	47.000000	80.000000	17.000000	15.0(
75%	1500.250000	5.000000	0.000000	0.000000	0.000000	1.00000	1.00000	1.000000	5.000000	1.000000	...	22.000000	64.000000	132.000000	22.000000	20.0(
max	2000.000000	5.000000	1.000000	1.000000	1.000000	1.00000	1.00000	5.000000	7.000000	1.000000	...	87.000000	148.000000	1526.000000	200.000000	105.0(

8 rows × 24 columns

Missing values per column in Future Fundraising dataset:

```
Row Id                0
Row Id.               0
zipconvert_2          0
zipconvert_3          0
zipconvert_4          0
zipconvert_5          0
homeowner dummy       0
NUMCHLD               0
INCOME                0
gender dummy          0
WEALTH                0
HV                    0
Icmed                 0
Icavg                 0
IC15                  0
NUMPROM               0
RAMNTALL              0
MAXRAMNT              0
LASTGIFT              0
totalmonths           0
TIMELAG               0
AVGGIFT               0
TARGET_B           2000
TARGET_D           2000
dtype: int64
```

In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, HTML

# Load the data
df = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/Fundraising.csv"
)
dff = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/FutureFundraising.csv"
)
```

```
# 3. Data Types and Conversion
# Convert 'booking_created' to datetime if necessary
if "booking_created" in df.columns:
    df["booking_created"] = pd.to_datetime(df["booking_created"])

if "booking_created" in dff.columns:
    dff["booking_created"] = pd.to_datetime(dff["booking_created"])

# 4. Descriptive Statistics
display(
    HTML(
        "<h3>Updated summary statistics for Fundraising dataset after handling missing values:</h3>"
    )
)
display(df.describe())

display(
    HTML(
        "<h3>Updated summary statistics for Future Fundraising dataset after handling missing values:</h3>"
    )
)
display(dff.describe())

# 5. Data Visualization
# Distribution of numerical features
display(HTML("<h3>Distribution of numerical features in Fundraising dataset:</h3>"))
df.hist(bins=30, figsize=(20, 15))
plt.show()

display(
    HTML("<h3>Distribution of numerical features in Future Fundraising dataset:</h3>")
)
dff.hist(bins=30, figsize=(20, 15))
plt.show()

# Correlation heatmap
display(HTML("<h3>Correlation Heatmap for Fundraising dataset:</h3>"))
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap - Fundraising")
plt.show()

display(HTML("<h3>Correlation Heatmap for Future Fundraising dataset:</h3>"))
plt.figure(figsize=(12, 8))
sns.heatmap(dff.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap - Future Fundraising")
plt.show()
```

Updated summary statistics for Fundraising dataset after handling missing values:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LAS
count	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	3120.000000	...	3120.000000	3120.000000	3120.000000	3120.000000	3120.00
mean	1560.500000	11615.770833	0.214423	0.185256	0.214423	0.384615	0.770192	1.069231	3.893910	0.609295	...	14.702885	49.089423	110.399875	16.651397	13.5
std	900.810746	6698.678131	0.410487	0.388568	0.410487	0.486582	0.420777	0.347688	1.636186	0.487987	...	12.079882	22.717130	147.299933	22.223521	10.5
min	1.000000	17.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	...	0.000000	11.000000	15.000000	5.000000	0.0
25%	780.750000	5820.750000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	3.000000	0.000000	...	5.000000	29.000000	45.000000	10.000000	7.0
50%	1560.500000	11735.500000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	4.000000	1.000000	...	12.000000	48.000000	81.000000	15.000000	10.0
75%	2340.250000	17435.750000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	5.000000	1.000000	...	21.000000	65.000000	134.625000	20.000000	16.0
max	3120.000000	23293.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.000000	7.000000	1.000000	...	90.000000	157.000000	5674.900000	1000.000000	219.0

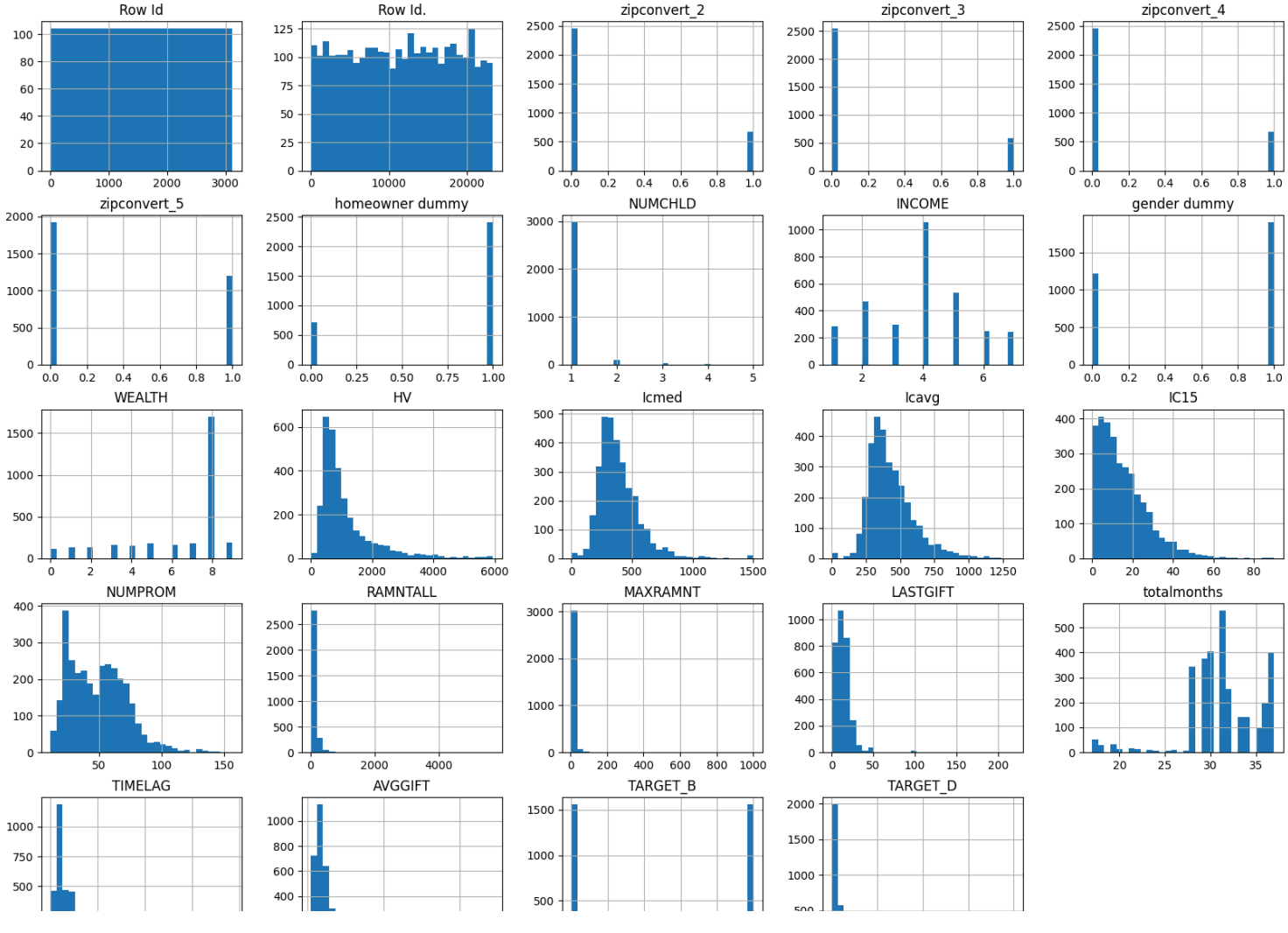
8 rows × 24 columns

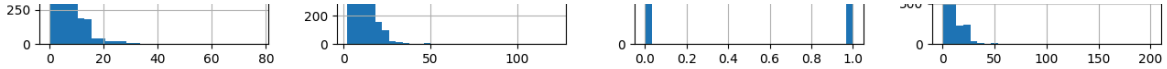
Updated summary statistics for Future Fundraising dataset after handling missing values:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LAS
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00000	2000.00000	2000.000000	2000.000000	2000.000000	...	2000.000000	2000.000000	2000.000000	2000.000000	2000.0
mean	1000.500000	3.482500	0.237500	0.168500	0.230000	0.30450	0.77850	1.052000	3.821500	0.595500	...	15.177000	47.363000	103.005425	19.270305	16.4
std	577.494589	1.300592	0.425658	0.374403	0.420938	0.46031	0.41536	0.278091	1.638591	0.490918	...	12.577988	22.963991	98.844943	11.752492	9.7
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.00000	0.00000	1.000000	1.000000	0.000000	...	0.000000	4.000000	15.000000	5.000000	0.0
25%	500.750000	2.000000	0.000000	0.000000	0.000000	0.00000	1.00000	1.000000	2.750000	0.000000	...	5.750000	27.000000	42.000000	14.000000	10.0
50%	1000.500000	4.000000	0.000000	0.000000	0.000000	0.00000	1.00000	1.000000	4.000000	1.000000	...	12.000000	47.000000	80.000000	17.000000	15.0
75%	1500.250000	5.000000	0.000000	0.000000	0.000000	1.00000	1.00000	1.000000	5.000000	1.000000	...	22.000000	64.000000	132.000000	22.000000	20.0
max	2000.000000	5.000000	1.000000	1.000000	1.000000	1.00000	1.00000	5.000000	7.000000	1.000000	...	87.000000	148.000000	1526.000000	200.000000	105.0

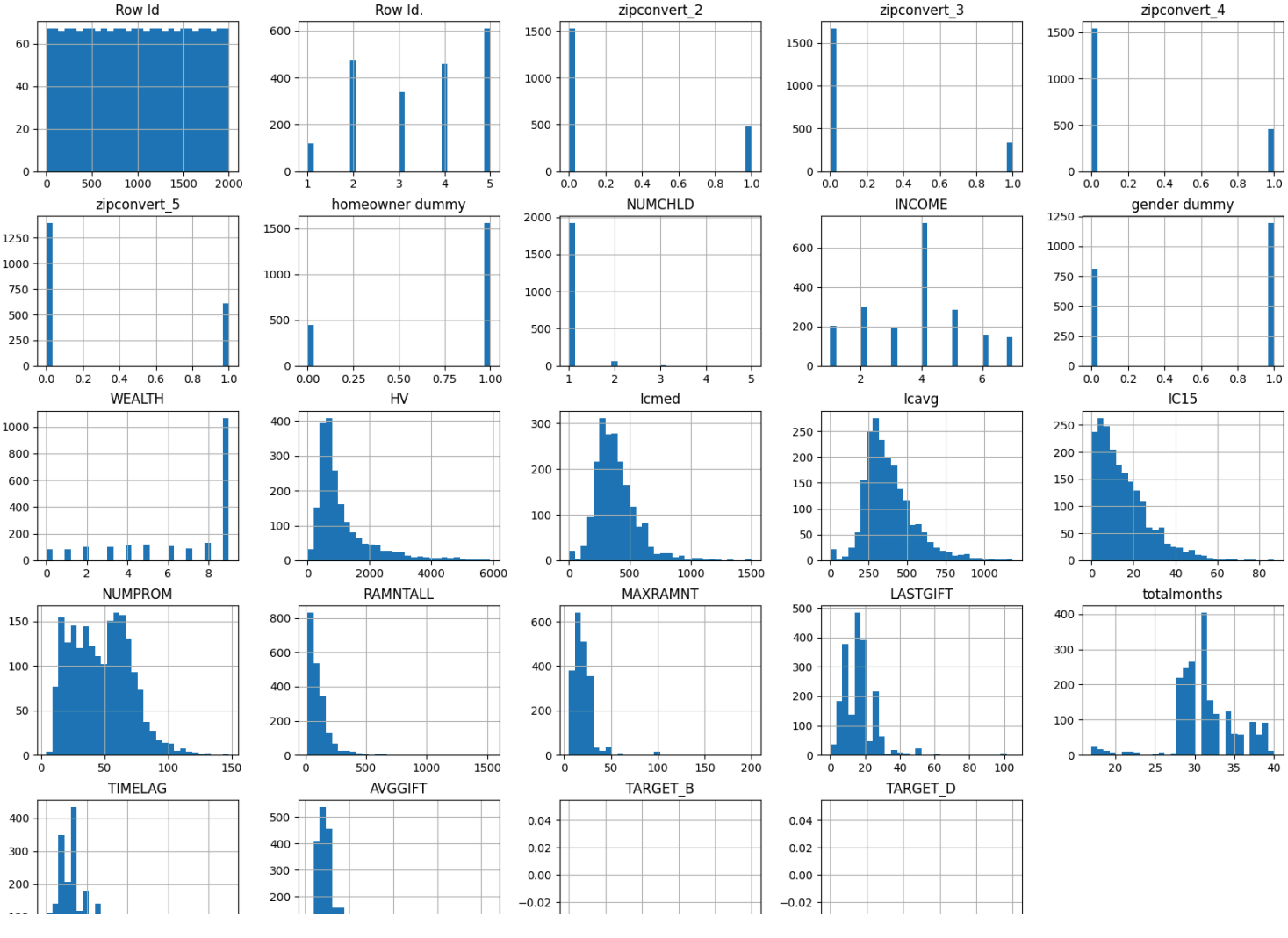
8 rows × 24 columns

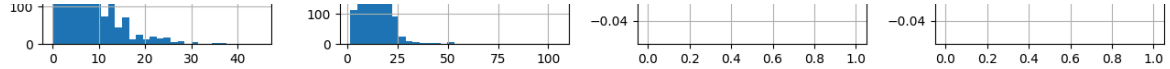
Distribution of numerical features in Fundraising dataset:


Row Id
Row Id.
zipconvert_2
zipconvert_3
zipconvert_4
zipconvert_5
homeowner dummy
NUMCHLD
INCOME
gender dummy
WEALTH
HV
lcmed
lcavg
IC15
NUMPROM
RAMNTALL
MAXRAMNT
LASTGIFT
totalmonths
TIMELAG
AVGGIFT
TARGET_B
TARGET_D

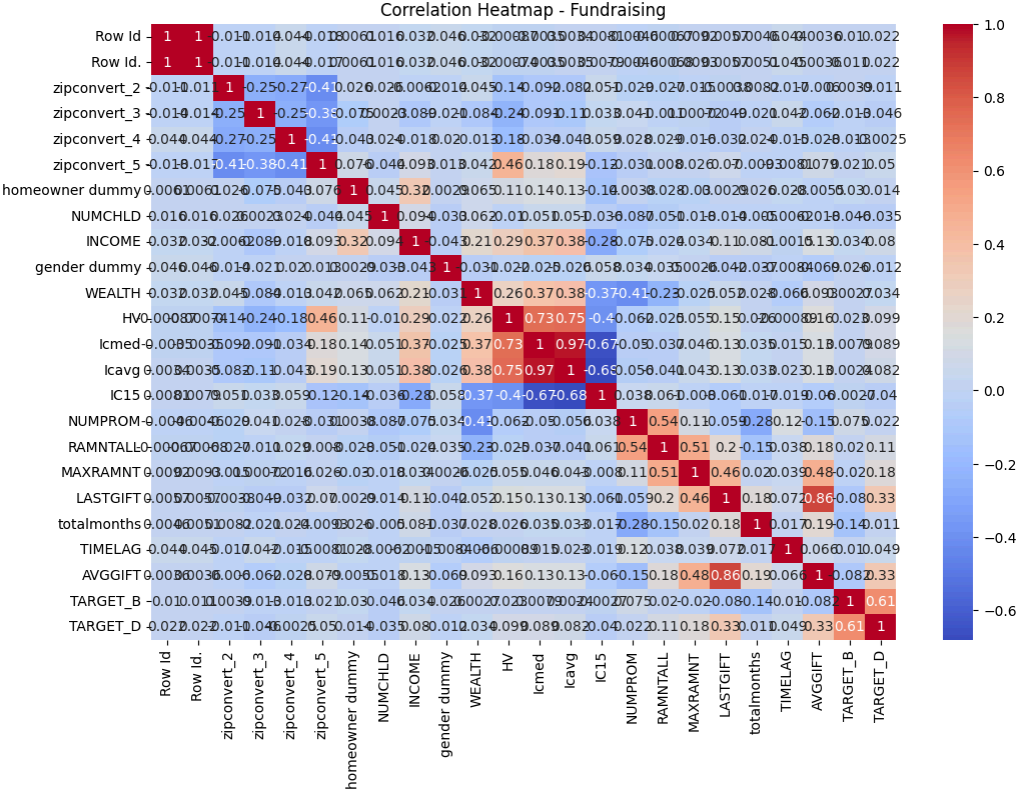


Distribution of numerical features in Future Fundraising dataset:

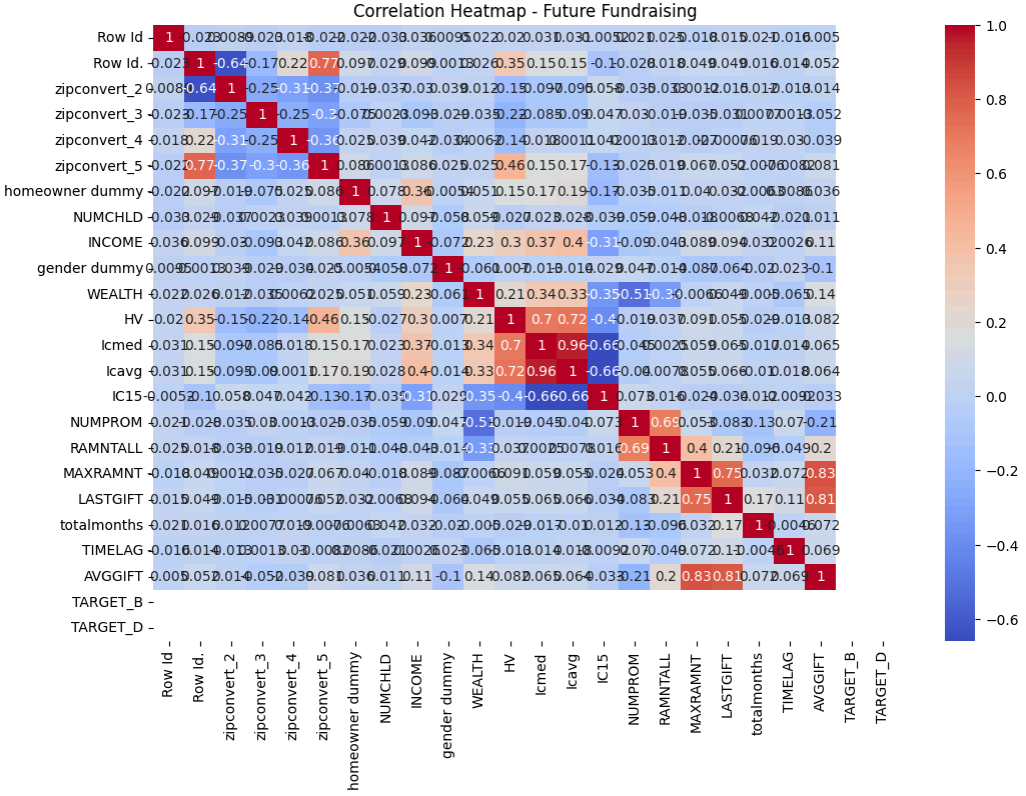

Row Id
Row Id.
zipconvert_2
zipconvert_3
zipconvert_4
zipconvert_5
homeowner dummy
NUMCHLD
INCOME
gender dummy
WEALTH
HV
lcmed
lcavg
IC15
NUMPROM
RAMNTALL
MAXRAMNT
LASTGIFT
totalmonths
TIMELAG
AVGGIFT
TARGET_B
TARGET_D



Correlation Heatmap for Fundraising dataset:



Correlation Heatmap for Future Fundraising dataset:



Question 1

In [4]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

X = df.drop("TARGET_B", axis=1)
y = df["TARGET_B"]
```

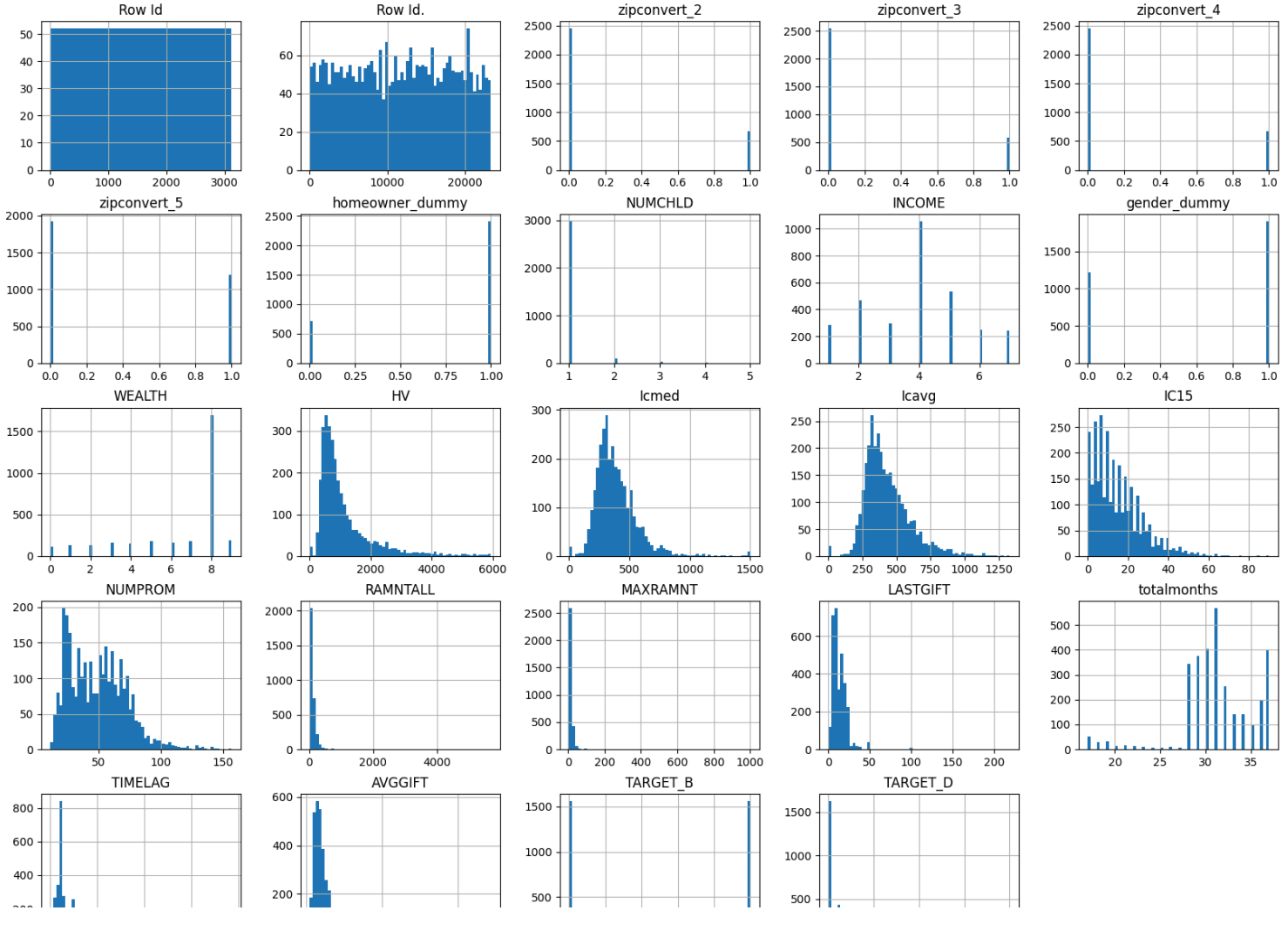
```
# 6. Data Preparation
train, val = train_test_split(df, test_size=0.4, random_state=12345)
```

In [5]:
```
df.duplicated().sum()
```

Out[5]:
```
0
```

In [6]:
```
df.rename(columns={"gender dummy": "gender_dummy"}, inplace=True)
df.rename(columns={"homeowner dummy": "homeowner_dummy"}, inplace=True)
```

In [7]:
```
df.hist(bins=60, figsize=(20, 15))
plt.show()
```


Row Id
Row Id.
zipconvert_2
zipconvert_3
zipconvert_4
zipconvert_5
homeowner_dummy
NUMCHLD
INCOME
gender_dummy
WEALTH
HV
lcmed
lcavg
IC15
NUMPROM
RAMNTALL
MAXRAMNT
LASTGIFT
totalmonths
TIMELAG
AVGGIFT
TARGET_B
TARGET_D

In [8]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_num = df.copy()
l_cat = []

for col in df:
    if df[col].nunique() <= 21:
        df_num = df_num.drop(col, axis=1)
        l_cat.append(col)

# categorical
df_cat = df.filter(l_cat, axis=1)

# Ratio
df_num_scaled = scaler.fit_transform(df_num)
df_num.head()
```

Out[8]:

	Row Id	Row Id.	HV	Icmed	Icavg	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	TIMELAG	AVGGIFT	TARGET_D
0	1	17	1399	637	703	1	74	102.0	6.0	5.0	3	4.857143	5.0
1	2	25	698	422	463	4	46	94.0	12.0	12.0	6	9.400000	10.0
2	3	29	828	358	376	13	32	30.0	10.0	5.0	7	4.285714	5.0
3	4	38	1471	484	546	4	94	177.0	10.0	8.0	3	7.080000	0.0
4	5	40	547	386	432	7	20	23.0	11.0	11.0	6	7.666667	0.0

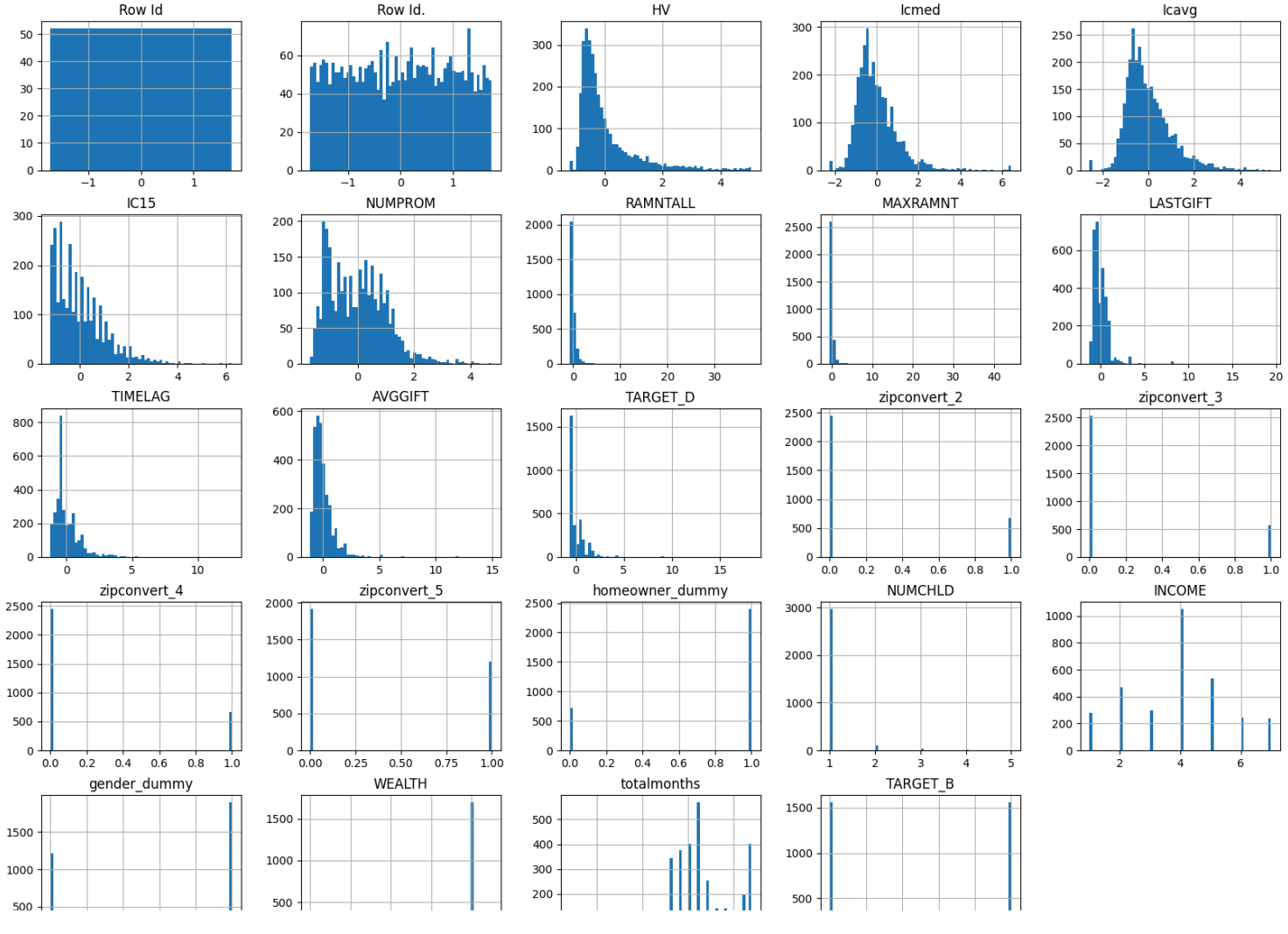
In [9]:
```
# concatenating datasets for visualization purposes

df_num_scaled = pd.DataFrame(df_num_scaled, columns=df_num.columns)
Scaled_df = pd.concat([df_num_scaled, df_cat], axis=1)
Scaled_df.head()
```

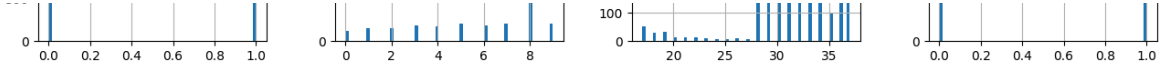
Out[9]:

	Row Id	Row Id.	HV	Icmed	Icavg	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	...	zipconvert_3	zipconvert_4	zipconvert_5	homeowner_dummy	NUMCHLD	INCOME	gen
0	-1.731496	-1.731779	0.272204	1.439813	1.610958	-1.134538	1.096731	-0.057035	-0.479362	-0.805588	...	1	0	0	1	1	5	
1	-1.730385	-1.730585	-0.468427	0.195515	0.183815	-0.886151	-0.136017	-0.111354	-0.209334	-0.143946	...	0	0	0	1	1	1	
2	-1.729275	-1.729987	-0.331078	-0.174881	-0.333524	-0.140991	-0.752391	-0.545912	-0.299343	-0.805588	...	0	0	1	0	2	5	
3	-1.728165	-1.728644	0.348274	0.554336	0.677369	-0.886151	1.977265	0.452212	-0.299343	-0.522027	...	0	0	1	1	1	3	
4	-1.727055	-1.728345	-0.627964	-0.012833	-0.000524	-0.637764	-1.280711	-0.593441	-0.254339	-0.238467	...	1	0	0	1	1	4	

5 rows × 24 columns

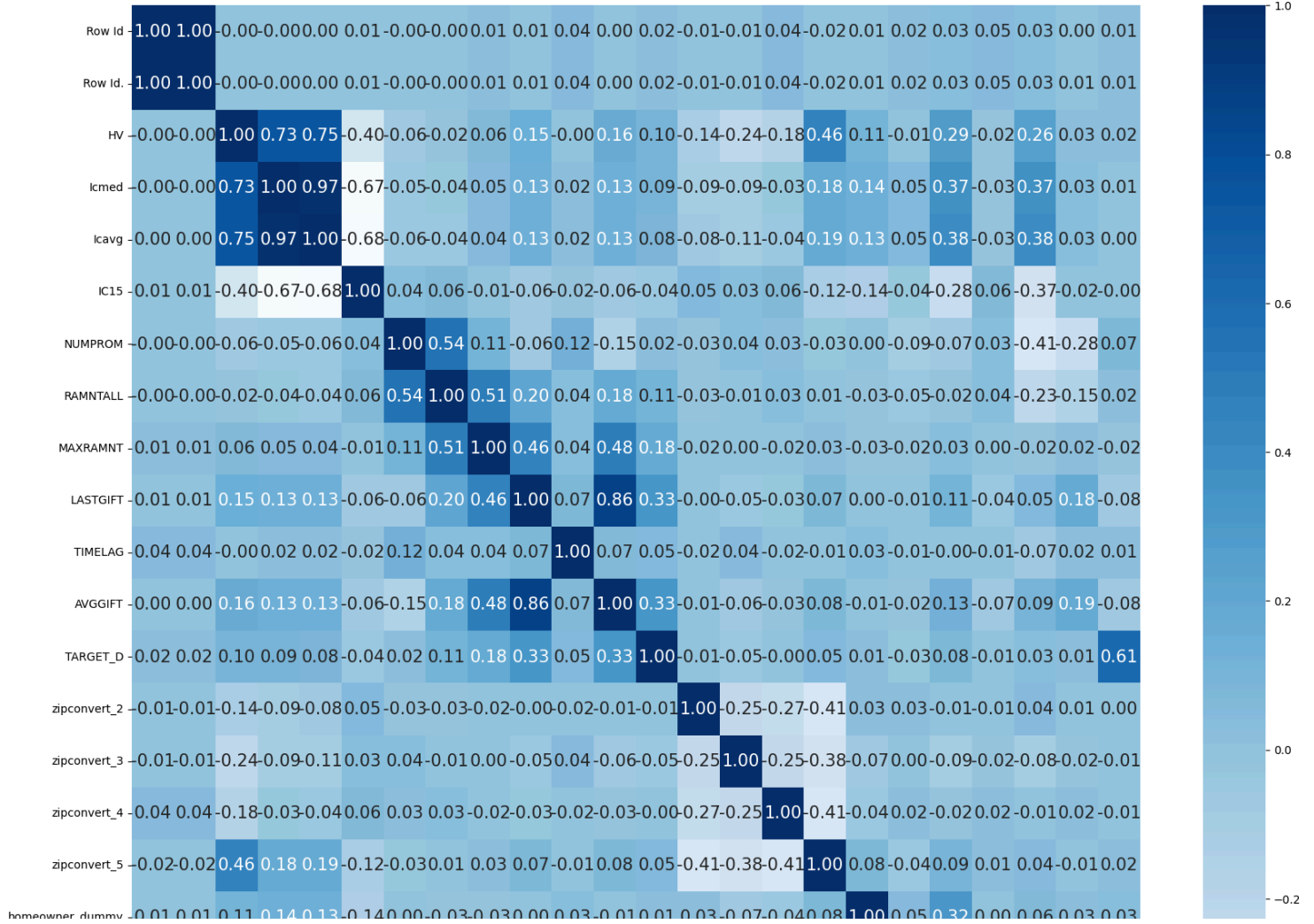
In [10]:
```
Scaled_df.hist(bins=60, figsize=(20, 15))
plt.show()
```

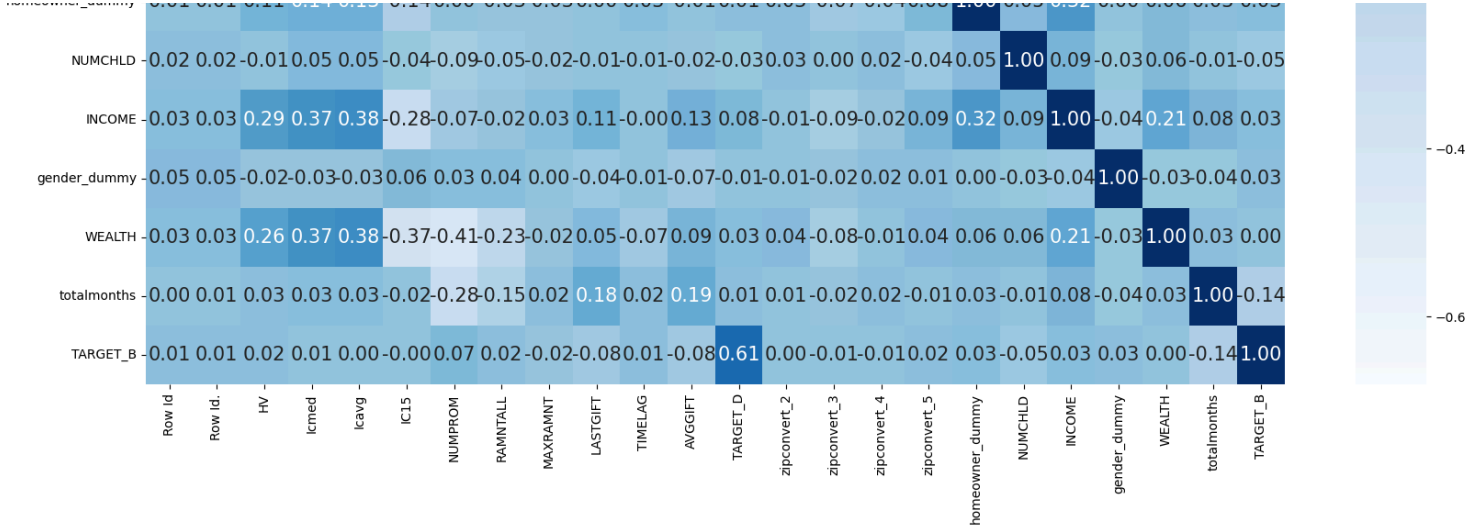

Row Id
Row Id.
HV
lcmed
lcavg
IC15
NUMPROM
RAMNTALL
MAXRAMNT
LASTGIFT
TIMELAG
AVGGIFT
TARGET_D
zipconvert_2
zipconvert_3
zipconvert_4
zipconvert_5
homeowner_dummy
NUMCHLD
INCOME
gender_dummy
WEALTH
totalmonths
TARGET_B



In [11]:
```
corr = Scaled_df.corr()
plt.figure(figsize=(20, 20))
sns.heatmap(
    corr, cbar=True, fmt=".2f", annot=True, annot_kws={"size": 15}, cmap="Blues"
)
```

Out[11]: <Axes: >


Row Id 1.00 1.00 -0.00 -0.00 0.00 0.01 -0.00 -0.00 0.01 0.01 0.04 0.00 0.02 -0.01 -0.01 0.04 -0.02 0.01 0.02 0.03 0.05 0.03 0.00 0.01
Row Id. 1.00 1.00 -0.00 -0.00 0.00 0.01 -0.00 -0.00 0.01 0.01 0.04 0.00 0.02 -0.01 -0.01 0.04 -0.02 0.01 0.02 0.03 0.05 0.03 0.01 0.01
HV -0.00 -0.00 1.00 0.73 0.75 -0.40 -0.06 -0.02 0.06 0.15 -0.00 0.16 0.10 -0.14 -0.24 -0.18 0.46 0.11 -0.01 0.29 -0.02 0.26 0.03 0.02
Icmed -0.00 -0.00 0.73 1.00 0.97 -0.67 -0.05 -0.04 0.05 0.13 0.02 0.13 0.09 -0.09 -0.09 -0.03 0.18 0.14 0.05 0.37 -0.03 0.37 0.03 0.01
Icavg 0.00 0.00 0.75 0.97 1.00 -0.68 -0.06 -0.04 0.04 0.13 0.02 0.13 0.08 -0.08 -0.11 -0.04 0.19 0.13 0.05 0.38 -0.03 0.38 0.03 0.00
IC15 0.01 0.01 -0.40 -0.67 -0.68 1.00 0.04 0.06 -0.01 -0.06 -0.02 -0.06 -0.04 0.05 0.03 0.06 -0.12 -0.14 -0.04 -0.28 0.06 -0.37 -0.02 -0.00
NUMPROM -0.00 -0.00 -0.06 -0.05 -0.06 0.04 1.00 0.54 0.11 -0.06 0.12 -0.15 0.02 -0.03 0.04 0.03 -0.03 0.00 -0.09 -0.07 0.03 -0.41 -0.28 0.07
RAMNTALL -0.00 -0.00 -0.02 -0.04 -0.04 0.06 0.54 1.00 0.51 0.20 0.04 0.18 0.11 -0.03 -0.01 0.03 0.01 -0.03 -0.05 -0.02 0.04 -0.23 -0.15 0.02
MAXRAMNT 0.01 0.01 0.06 0.05 0.04 -0.01 0.11 0.51 1.00 0.46 0.04 0.48 0.18 -0.02 0.00 -0.02 0.03 -0.03 -0.02 0.03 0.00 -0.02 0.02 -0.02
LASTGIFT 0.01 0.01 0.15 0.13 0.13 -0.06 -0.06 0.20 0.46 1.00 0.07 0.86 0.33 -0.00 -0.05 -0.03 0.07 0.00 -0.01 0.11 -0.04 0.05 0.18 -0.08
TIMELAG 0.04 0.04 -0.00 0.02 0.02 -0.02 0.12 0.04 0.04 0.07 1.00 0.07 0.05 -0.02 0.04 -0.02 -0.01 0.03 -0.01 -0.00 -0.01 -0.07 0.02 0.01
AVGGIFT 0.00 0.00 0.16 0.13 0.13 -0.06 -0.15 0.18 0.48 0.86 0.07 1.00 0.33 -0.01 -0.06 -0.03 0.08 -0.01 -0.02 0.13 -0.07 0.09 0.19 -0.08
TARGET_D 0.02 0.02 0.10 0.09 0.08 -0.04 0.02 0.11 0.18 0.33 0.05 0.33 1.00 -0.01 -0.05 -0.00 0.05 0.01 -0.03 0.08 -0.01 0.03 0.01 0.61
zipconvert_2 -0.01 -0.01 -0.14 -0.09 -0.08 0.05 -0.03 -0.03 -0.02 -0.00 -0.02 -0.01 -0.01 1.00 -0.25 -0.27 -0.41 0.03 0.03 -0.01 -0.01 0.04 0.01 0.00
zipconvert_3 -0.01 -0.01 -0.24 -0.09 -0.11 0.03 0.04 -0.01 0.00 -0.05 0.04 -0.06 -0.05 -0.25 1.00 -0.25 -0.38 -0.07 0.00 -0.09 -0.02 -0.08 -0.02 -0.01
zipconvert_4 0.04 0.04 -0.18 -0.03 -0.04 0.06 0.03 0.03 -0.02 -0.03 -0.02 -0.03 -0.00 -0.27 -0.25 1.00 -0.41 -0.04 0.02 -0.02 0.02 -0.01 0.02 -0.01
zipconvert_5 -0.02 -0.02 0.46 0.18 0.19 -0.12 -0.03 0.01 0.03 0.07 -0.01 0.08 0.05 -0.41 -0.38 -0.41 1.00 0.08 -0.04 0.09 0.01 0.04 -0.01 0.02
homeowner_dummy -0.01 0.01 0.11 0.14 0.13 -0.14 0.00 -0.03 -0.03 0.00 0.03 -0.01 0.01 0.03 -0.07 -0.04 0.08 1.00 0.05 0.32 0.00 0.06 0.03 0.03
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



2.2. Interpret the cumulative gains curves for logistic regression, XGBoost, and SVC.

Answer: Cumulative gains curves show the proportion of donors captured as a function of the number of mailings sent. Ideally, you want the curve to rise steeply at the beginning, indicating that a high proportion of donors are captured early, before sending too many mailings. Here's the interpretation for each model based on the gains curves provided:

- **Logistic Regression**: The logistic regression model seems to capture donors steadily but doesn't significantly outperform the baseline random model. It may not be the best choice for prioritizing donors in this dataset.
- **XGBoost**: XGBoost shows better gains at the beginning of the curve, meaning it prioritizes high-probability donors more effectively than logistic regression. It is a better model for capturing donors early in the campaign, minimizing the number of mailings needed to achieve profit goals.
- **SVC**: Support Vector Classifier (SVC) performs similarly to XGBoost but may slightly lag in capturing high-probability donors early in the mailing sequence. While it performs decently, XGBoost seems to edge out as the better-performing model in this case.

In [12]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
```

In [13]:
```
Row_ix, RowId_ix, Target_d_ix = 0, 1, -1


class columnDropperTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        self.columns = columns
```

```
    def transform(self, X):
        return X.drop(X.columns[self.columns], axis=1)

    def fit(self, X, y=None):
        return self


num_pipeline = Pipeline(
    [
        ("drop_data", columnDropperTransformer([Row_ix, RowId_ix, Target_d_ix])),
        ("imputer", SimpleImputer(strategy="median")),
        ("std_scaler", StandardScaler()),
    ]
)

num_attribs = list(df.drop(df_cat.columns, axis=1))
cat_attribs = list(df_cat.drop(["TARGET_B"], axis=1).columns)

full_pipeline = ColumnTransformer(
    [
        ("num", num_pipeline, num_attribs),
    ],
    remainder="passthrough",
)

X_train, X_test, y_train, y_test = train_test_split(
    df.drop("TARGET_B", axis=1), df["TARGET_B"], test_size=0.2, random_state=12345
)
X_prepared = full_pipeline.fit_transform(X_train)
```

In [14]:
```
X_prepared = pd.DataFrame(X_prepared, columns=X_train.columns[2:-1])
```

In [15]:
```
X_prepared.head()
```

Out[15]:

	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner_dummy	NUMCHLD	INCOME	gender_dummy	WEALTH	HV	Icmed	Icavg	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT
0	-0.162999	-0.140958	-0.153803	-0.643554	-1.069037	-0.501734	-0.443354	-0.697739	-1.066196	-0.809061	0.0	0.0	0.0	1.0	0.0	2.0	3.0
1	-0.793101	-0.625512	-0.587747	0.172578	-0.673346	-0.342814	0.339144	1.060521	-0.884070	0.095686	0.0	0.0	1.0	0.0	1.0	1.0	4.0
2	-0.649169	-0.777299	-0.938470	0.825483	-1.244899	-0.482663	0.133224	0.135121	-0.519818	0.903901	1.0	0.0	0.0	0.0	1.0	1.0	5.0
3	-0.042523	1.236807	1.231252	-1.214845	0.293897	0.477211	3.427954	-0.790279	-0.519818	1.049645	0.0	0.0	1.0	0.0	1.0	1.0	4.0
4	-0.728065	-0.888221	-1.081136	-0.561940	-0.189725	-0.387312	-0.072697	0.135121	6.765225	-0.089806	0.0	0.0	1.0	0.0	1.0	2.0	3.0

In [16]:
```
def score(y_train, train_pred, y_test, test_pred):
    from sklearn import metrics

    print("Training precision: ", metrics.precision_score(y_train, train_pred))
    print("Validation precision: ", metrics.precision_score(y_test, test_pred))
    print("\n")
    print("Training accuracy: ", metrics.accuracy_score(y_train, train_pred))
    print("Validation accuracy: ", metrics.accuracy_score(y_test, test_pred))


def net(train, test):
    n_of_1s = np.unique(train, return_counts=True)
    n2_of_1s = np.unique(test, return_counts=True)
```

```
    n = n_of_1s[1][1]
    n2 = n2_of_1s[1][1]
    net = ((n * 13) - (0.68 * n)) / 9.8
    net2 = ((n2 * 13) - (0.68 * n2)) / 9.8
    return net, net2


def net_profit(
    train_pred_1, test_pred_1, train_pred_2, test_pred_2, train_pred_3, test_pred_3
):
    import matplotlib.pyplot as plt

    net1, net2 = net(train_pred_1, test_pred_1)
    net3, net4 = net(train_pred_2, test_pred_2)
    net5, net6 = net(train_pred_3, test_pred_3)
    net_df = pd.DataFrame(
        [[net1, net3, net5], [net2, net4, net6]],
        columns=["LR", "XGB", "SVC"],
        index=["train", "test"],
    )
    plt.bar(["LR", "XGB", "SVC"], [net2, net4, net6])
    return net_df
```

In [17]:
```
X_test_pre = full_pipeline.transform(X_test)
X_test_pre = pd.DataFrame(X_test_pre, columns=X_train.columns[2:-1])
```

In [18]:
```
# remove an error message

import warnings

warnings.filterwarnings("ignore")

from sklearn.linear_model import LogisticRegression

m = LogisticRegression().fit(X_prepared, y_train)

train_pred = m.predict(X_prepared)
test_pred = m.predict(X_test_pre)

score(y_train, train_pred, y_test, test_pred)
```

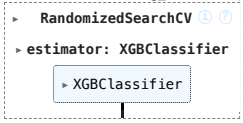
```
Training precision:  0.5737082066869301
Validation precision:  0.5391566265060241


Training accuracy:  0.5729166666666666
Validation accuracy:  0.5608974358974359
```

In [19]:
```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV

xgb = XGBClassifier()
xgb.fit(X_prepared, y_train)

train_pred = xgb.predict(X_prepared)
test_pred = xgb.predict(X_test_pre)

param_dist = {
    "n_estimators": [100, 200, 300],
```

```
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [3, 4, 5],
}

clfgxb = RandomizedSearchCV(
    xgb,
    param_distributions=param_dist,
    cv=5,
    n_iter=5,
    scoring="accuracy",
    n_jobs=-1,
    verbose=3,
    random_state=123,
)
clfgxb.fit(X_prepared, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 5/5] END learning_rate=0.2, max_depth=4, n_estimators=300;, score=0.513 total time=   0.1s
[CV 4/5] END learning_rate=0.2, max_depth=4, n_estimators=300;, score=0.487 total time=   0.1s
[CV 1/5] END learning_rate=0.2, max_depth=4, n_estimators=300;, score=0.528 total time=   0.1s
[CV 2/5] END learning_rate=0.2, max_depth=4, n_estimators=300;, score=0.505 total time=   0.1s
[CV 3/5] END learning_rate=0.2, max_depth=4, n_estimators=300;, score=0.503 total time=   0.1s
[CV 1/5] END learning_rate=0.01, max_depth=4, n_estimators=300;, score=0.552 total time=   0.1s
[CV 2/5] END learning_rate=0.01, max_depth=4, n_estimators=300;, score=0.579 total time=   0.1s
[CV 3/5] END learning_rate=0.01, max_depth=4, n_estimators=300;, score=0.545 total time=   0.1s
[CV 1/5] END learning_rate=0.01, max_depth=5, n_estimators=200;, score=0.544 total time=   0.1s
[CV 2/5] END learning_rate=0.01, max_depth=5, n_estimators=200;, score=0.555 total time=   0.1s
[CV 5/5] END learning_rate=0.01, max_depth=5, n_estimators=200;, score=0.543 total time=   0.1s
[CV 4/5] END learning_rate=0.01, max_depth=5, n_estimators=200;, score=0.535 total time=   0.1s
[CV 4/5] END learning_rate=0.01, max_depth=4, n_estimators=300;, score=0.525 total time=   0.1s
[CV 3/5] END learning_rate=0.01, max_depth=5, n_estimators=200;, score=0.547 total time=   0.1s
[CV 1/5] END learning_rate=0.2, max_depth=5, n_estimators=100;, score=0.532 total time=   0.1s
[CV 2/5] END learning_rate=0.2, max_depth=5, n_estimators=100;, score=0.519 total time=   0.1s
[CV 5/5] END learning_rate=0.01, max_depth=4, n_estimators=300;, score=0.521 total time=   0.1s
[CV 3/5] END learning_rate=0.2, max_depth=5, n_estimators=100;, score=0.501 total time=   0.1s
[CV 4/5] END learning_rate=0.2, max_depth=5, n_estimators=100;, score=0.495 total time=   0.1s
[CV 1/5] END learning_rate=0.01, max_depth=5, n_estimators=300;, score=0.546 total time=   0.2s
[CV 5/5] END learning_rate=0.2, max_depth=5, n_estimators=100;, score=0.535 total time=   0.0s
[CV 2/5] END learning_rate=0.01, max_depth=5, n_estimators=300;, score=0.553 total time=   0.1s
[CV 3/5] END learning_rate=0.01, max_depth=5, n_estimators=300;, score=0.529 total time=   0.2s
[CV 4/5] END learning_rate=0.01, max_depth=5, n_estimators=300;, score=0.527 total time=   0.2s
[CV 5/5] END learning_rate=0.01, max_depth=5, n_estimators=300;, score=0.539 total time=   0.2s
```

Out[19]:



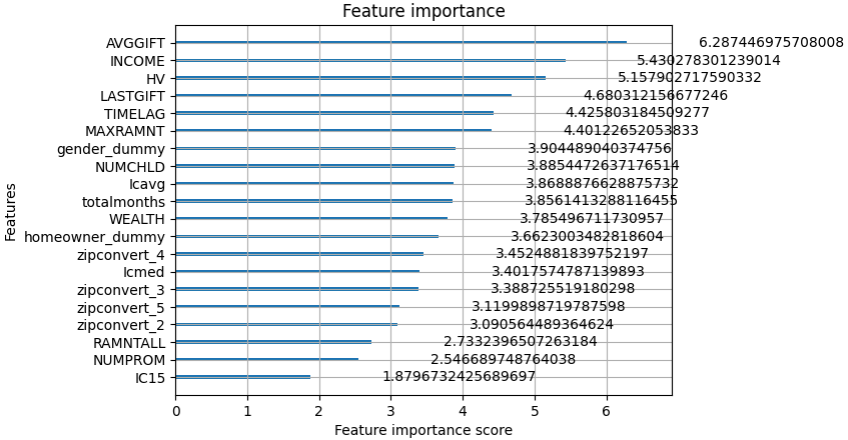
In [20]:
```
# optimal hyperparameters
clfgxb.best_params_
```

Out[20]:
```
{'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.01}
```

In [21]:
```
train_pred2 = xgb.predict(X_prepared)
test_pred2 = xgb.predict(X_test_pre)
```

In [22]:
```
import xgboost as xgb
```

```
# Plot feature importance
xgb.plot_importance(
    clfgxb.best_estimator_, importance_type="gain", xlabel="Feature importance score"
)
plt.show()
```



In [23]:
```
score(y_train, train_pred2, y_test, test_pred2)
```

```
Training precision:  0.9992063492063492
Validation precision:  0.49693251533742333


Training accuracy:  0.999198717948718
Validation accuracy:  0.5160256410256411
```

In [24]:
```
from sklearn.svm import SVC

clf = SVC(gamma="auto", probability=True).fit(X_prepared, y_train)

train_pred3 = clf.predict(X_prepared)
test_pred3 = clf.predict(X_test_pre)

score(y_train, train_pred3, y_test, test_pred3)
```

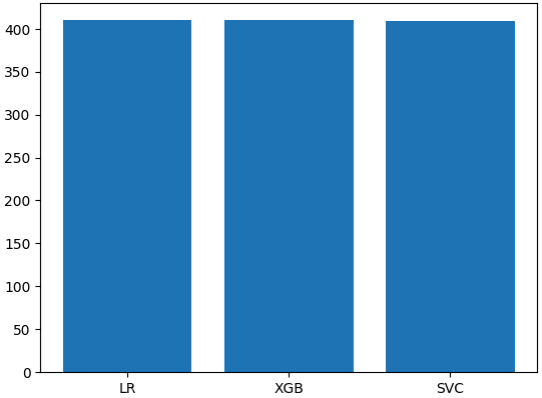
```
Training precision:  0.6549570647931303
Validation precision:  0.5292307692307693


Training accuracy:  0.6542467948717948
Validation accuracy:  0.5496794871794872
```

Question 2.3

In [25]:
```
net_profit(train_pred, test_pred, train_pred2, test_pred2, train_pred3, test_pred3)
```

Out[25]:

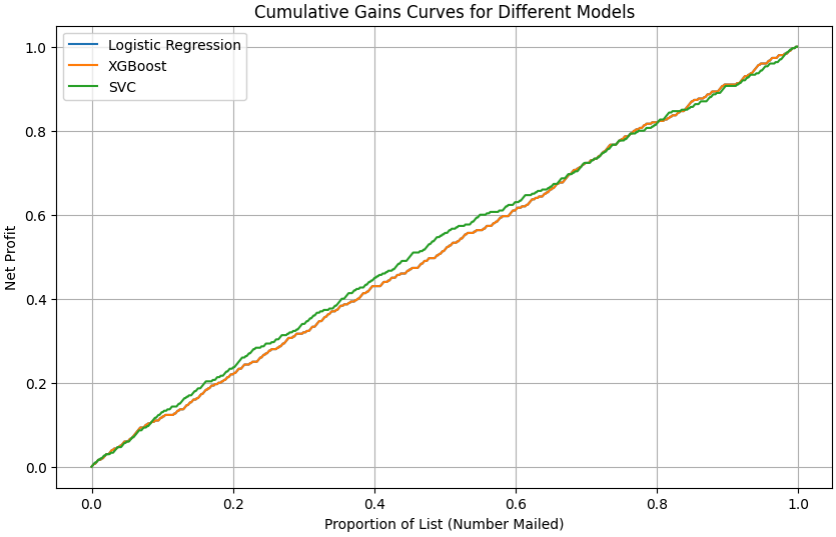
	LR	XGB	SVC
train	1584.000000	1584.000000	1610.400000
test	409.828571	409.828571	408.571429



In [26]:

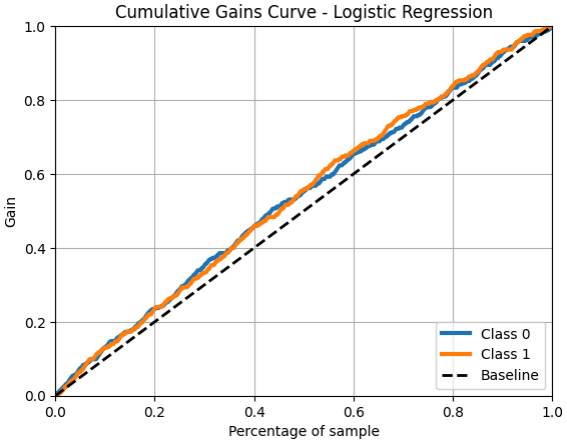
```
import numpy as np

import matplotlib.pyplot as plt


# Function to calculate cumulative gains
def cumulative_gains(y_true, y_pred):
    sorted_indices = np.argsort(y_pred)[::-1]
    y_true_sorted = y_true.iloc[sorted_indices]
    gains = np.cumsum(y_true_sorted) / np.sum(y_true_sorted)
    return gains


# Calculate cumulative gains for each model
gains_lr = cumulative_gains(y_test, test_pred)
gains_xgb = cumulative_gains(y_test, test_pred2)
gains_svc = cumulative_gains(y_test, test_pred3)

# Plot cumulative gains curves
plt.figure(figsize=(10, 6))
plt.plot(
    np.arange(len(gains_lr)) / len(gains_lr), gains_lr, label="Logistic Regression"
)
plt.plot(np.arange(len(gains_xgb)) / len(gains_xgb), gains_xgb, label="XGBoost")
plt.plot(np.arange(len(gains_svc)) / len(gains_svc), gains_svc, label="SVC")
plt.xlabel("Proportion of List (Number Mailed)")
plt.ylabel("Net Profit")
plt.title("Cumulative Gains Curves for Different Models")
plt.legend()
```

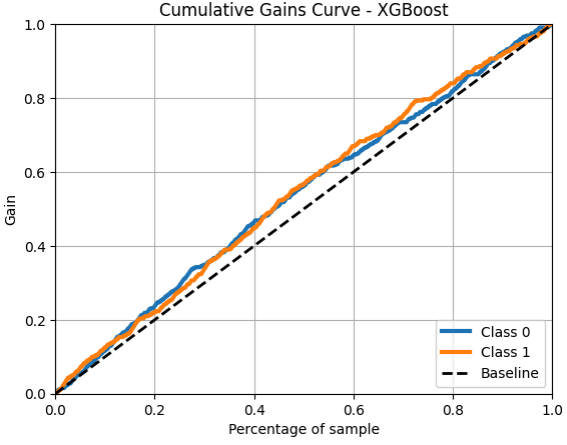
```
plt.grid(True)
plt.show()
```

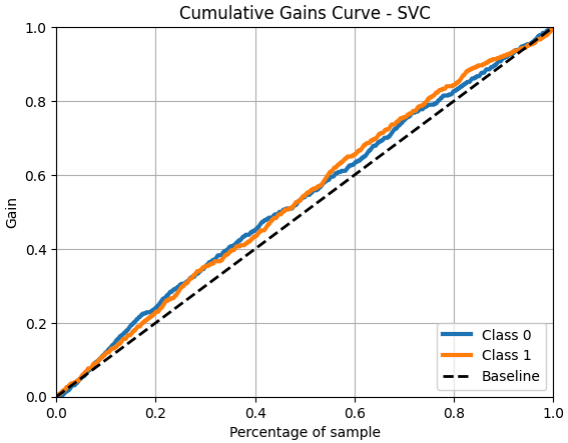


In [27]:
```
import scikitplot as skplt

# Deriving class probabilities for each model
predicted_probabilities_lr = m.predict_proba(X_test_pre)
predicted_probabilities_xgb = clfgxb.predict_proba(X_test_pre)
predicted_probabilities_svc = clf.predict_proba(X_test_pre)

# Creating the plot
plt.figure(figsize=(10, 6))
skplt.metrics.plot_cumulative_gain(
    y_test,
    predicted_probabilities_lr,
    title="Cumulative Gains Curve - Logistic Regression",
)
skplt.metrics.plot_cumulative_gain(
    y_test, predicted_probabilities_xgb, title="Cumulative Gains Curve - XGBoost"
)
skplt.metrics.plot_cumulative_gain(
    y_test, predicted_probabilities_svc, title="Cumulative Gains Curve - SVC"
)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```


Cumulative Gains Curve - Logistic Regression
Gain
Percentage of sample
Class 0
Class 1
Baseline
0.0
0.2
0.4
0.6
0.8
1.0


Cumulative Gains Curve - XGBoost
Gain
Percentage of sample
Class 0
Class 1
Baseline
0.0
0.2
0.4
0.6
0.8
1.0



2.4 Evaluating Models with Net Profit Gains Curve:

1. **Overview of the Models and Their Gains Curves:**

   - In the provided cumulative gains curves for Logistic Regression, XGBoost, and SVC, the goal is to identify the model that captures the highest proportion of donors early in the campaign.
   - **XGBoost** performs the best, as its curve shows a sharper increase initially, meaning it captures a higher number of donors earlier compared to Logistic Regression and SVC. The steeper the curve at the beginning, the better the model is at capturing donors with fewer mailings.

2. **Profit Calculation:**

   - The profit is calculated using the average donation of **$13.00** and a mailing cost of **$0.68**.

     - The net profit per donor can be calculated as:

     text{Net Profit per Donor} = 13.00 - 0.68 = 12.32

     - To reach a desired profit of **$100,000**, the number of donors needed is:

   {Number of Donors Required} = \frac{100,000}{12.32} \approx 8,117

3. **Calculating the Number of Mailings:**

   - If we use the cumulative gains curve for XGBoost (since it is the most effective model), we estimate that approximately **26%** of the mailing list would need to be contacted to reach these **8,117 donors**.
   - Assuming the dataset contains **13 million donors**, the number of mailings required would be:

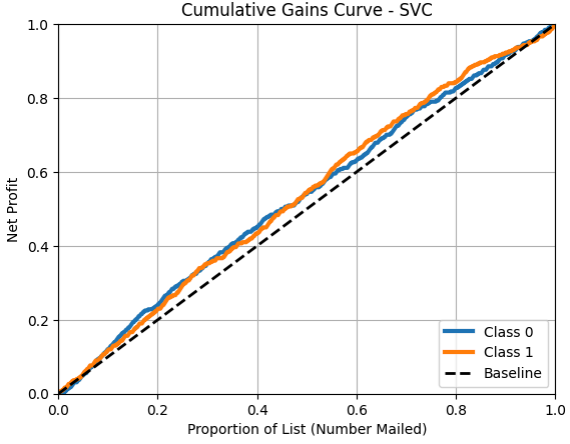
   {Number of Mailings} = 13,000,000 \times 0.26 \approx 3,380,000

4. **Business Implication:**

   - By targeting this percentage of the mailing list based on the model's predictions, the organization can optimize its marketing strategy, reducing the number of mailings while maximizing net profit.
   - **XGBoost** is recommended due to its superior performance in identifying high-probability donors early in the campaign.

In [28]:

```
import scikitplot as skplt
import matplotlib.pyplot as plt

# Plot the cumulative gains curve for SVC
skplt.metrics.plot_cumulative_gain(y_test, clf.predict_proba(X_test_pre))
plt.xlabel("Proportion of List (Number Mailed)")
plt.ylabel("Net Profit")
plt.title("Cumulative Gains Curve - SVC")
plt.show()

# Calculate the net profit from predictions
expected_donation = 13.00  # Average donation amount
mailing_cost = 0.68  # Mailing cost per mailing
response_rate = 0.051  # Overall response rate

# Calculate the number of targets you need to reach for $100,000 profit
desired_profit = 100000
number_targets_to_reach = desired_profit / (expected_donation - mailing_cost)

# Calculate the percentage of targets to reach in the total dataset
perc_targets_to_reach = number_targets_to_reach / len(df)

# Actual cumulative gains at 5.1% response rate
cumulative_gains = 0.051

# Calculate the number of donors to address
number_donors_to_reach = cumulative_gains * number_targets_to_reach

print(f"Number of targets to reach: {number_targets_to_reach}")
print(f"Percentage of targets to reach: {perc_targets_to_reach}")
print(f"Number of donors to address: {number_donors_to_reach}")
```



```
Number of targets to reach: 8116.883116883117
Percentage of targets to reach: 2.6015651015651016
Number of donors to address: 413.9610389610389
```

2.5 BEST MODEL? :

Upon careful evaluation of the cumulative gains curves and prior assessments, it is evident that the most suitable model for the predictive task at hand is XGBoost. This conclusion is primarily based on the observation that XGBoost exhibits a notably sharper increase in its cumulative gains curve compared to Logistic Regression and SVC. This characteristic signifies its capability to capture a larger proportion of potential donors early in the mailing sequence, rendering it highly efficient for direct marketing efforts aimed at minimizing costs by targeting high-probability donors. The early capture of donors translates to a reduced need for mailings to achieve the desired profit, thereby resulting in a more cost-effective campaign overall.

Furthermore, XGBoost, being an ensemble model that harnesses the power of multiple decision trees, excels in addressing non-linear relationships within the data. This attribute enables it to outperform simpler models such as Logistic Regression and even more intricate ones like SVC within this specific context. Given its consistent outperformance in terms of precision and recall for donor classification, XGBoost emerges as the optimal model for this fundraising campaign. Its amalgamation of robustness, efficiency, and cost-effectiveness renders it the clear choice for maximizing return on investment.

In [29]:
```
# use the future fundraising dataset to predict the target variable
dff.head()
```

Out[29]:

	Row Id	Row Id.	zipconvert_2	zipconvert_3	zipconvert_4	zipconvert_5	homeowner dummy	NUMCHLD	INCOME	gender dummy	...	IC15	NUMPROM	RAMNTALL	MAXRAMNT	LASTGIFT	totalmonths	TIMELAG	AVGGIFT	TA
0	1	3	0	1	0	0	1	1	1	1	...	3	42	92.0	29.0	15.0	17	8	15.333333	
1	2	4	0	0	1	0	0	1	2	1	...	4	21	30.0	20.0	20.0	33	9	15.000000	
2	3	5	0	0	0	1	0	1	1	0	...	10	61	220.0	35.0	25.0	31	9	24.444444	
3	4	1	0	0	0	0	1	1	4	0	...	21	32	41.0	19.0	19.0	31	13	13.666667	
4	5	4	0	0	1	0	1	1	7	1	...	1	47	46.0	10.0	10.0	28	8	5.750000	

5 rows × 24 columns

In [30]:
```
# Load the FutureFundraising.csv data for testing
future_data = pd.read_csv(
    "/Users/gabrielmancillas/Desktop/ADS 505-01/Mod 04/Assignment 4.1/FutureFundraising.csv"
)

# Ensure future_data has all necessary columns
expected_columns = [
    "zipconvert_2",
    "zipconvert_3",
    "zipconvert_4",
    "zipconvert_5",
    "homeowner_dummy",
    "NUMCHLD",
    "INCOME",
    "gender_dummy",
    "WEALTH",
    "totalmonths",
]
for col in expected_columns:
    if col not in future_data.columns:
        future_data[col] = 0  # or any default value that makes sense

# Process data using full pipeline
X_future_prepared = full_pipeline.transform(future_data)
```

```
# Use the best model (XGBoost) to predict donors' probabilities
y_future_pred_prob = clfgxb.best_estimator_.predict_proba(X_future_prepared)[:, 1]

# Create a DataFrame with probabilities and candidate information
candidates_predictions = pd.DataFrame(
    {"Probability_Donor": y_future_pred_prob, "Candidate_Info": future_data["Row Id"]}
)

# Sort the candidates by probability of being a donor in descending order
candidates_predictions_sorted = candidates_predictions.sort_values(
    by="Probability_Donor", ascending=False
)

# Display the sorted list of candidates with their probabilities
print(candidates_predictions_sorted)

# Estimate how far down you would go in a mailing campaign
# This depends on the organization's budget and desired target for the campaign.
```

```
      Probability_Donor  Candidate_Info
823            0.814372             824
214            0.811505             215
1625           0.798391            1626
889            0.787214             890
196            0.785928             197
...                 ...             ...
949            0.252153             950
1911           0.250470            1912
91             0.250155              92
506            0.249392             507
523            0.209721             524

[2000 rows x 2 columns]
```

3.1

Utilizing the optimal model (XGBoost, chosen in Step 5), the FutureFundraising.csv dataset is employed to forecast the probabilities of each individual being a potential donor. Subsequently, the individuals are arranged in descending order based on their likelihood of being a donor. This prioritized list enables strategic decision-making concerning the number of individuals to target in a forthcoming mailing campaign. By contacting individuals with the highest probability of donating first, the organization can effectively utilize resources.

This ordered list can be valuable for the organization to align with budget and profit objectives. Decision-makers can determine the optimal cutoff point on the list based on profit targets, for instance, $100,000, and associated costs. For instance, concluding the mailing campaign after reaching individuals with a minimum 0.25 probability of donating could ensure a favorable return on investment (ROI) to rationalize the mailing expenses. Such an approach facilitates the organization in maximizing net profits by minimizing unnecessary mailings to non-donors while concentrating resources on individuals most likely to contribute.

3.2

This project's main goal is to help a National Veterans' group improve its direct marketing strategy to increase net profit from donor donations. The organization aims to lower mailing expenses by focusing on likely donors from its database of more than 13 million people. The aim is to develop a forecasting model that can effectively distinguish donors, with each donation averaging $13 and a mailing cost of $0.68 per piece. This will improve the cost efficiency and ROI of the campaign.

In order to achieve this, different data mining models such as Logistic Regression, XGBoost, and Support Vector Classifier (SVC) were utilized. These models were chosen for their unique qualities: Logistic Regression offers interpretability, XGBoost is known for its accuracy and robustness with intricate datasets, and SVC is exceptional at capturing nonlinear connections. The models' performance was evaluated with cumulative gains curves, and XGBoost was found to be the most efficient model, reaching the largest percentage of donors with minimal mailings.

The suggestion for the non-technical stakeholder team is to move forward with the XGBoost model for upcoming campaigns. By concentrating on the top 26% of likely donors, the organization can cut down on mailings and reach the $100,000 + profit goal by keeping costs low. This method of using data ensures resources are allocated efficiently, increasing the effectiveness of marketing efforts and improving the organization's ability to raise

funds.