marketing analytics

April 23, 2025

0.1 EDA

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import scipy.stats as stats
```

[3]:
```
# Load the Stata (.dta) dataset (adjust the path if necessary)
df = pd.read_stata("karlan_list_2007.dta")
```

[4]:
```
# Print the shape and column names
print("Dataset Shape:", df.shape)
print("Column Names:", df.columns.tolist())

# Show the first few rows
print(df.head())
```

```
Dataset Shape: (50083, 51)
Column Names: ['treatment', 'control', 'ratio', 'ratio2', 'ratio3', 'size',
'size25', 'size50', 'size100', 'sizeno', 'ask', 'askd1', 'askd2', 'askd3',
'ask1', 'ask2', 'ask3', 'amount', 'gave', 'amountchange', 'hpa', 'ltmedmra',
'freq', 'years', 'year5', 'mrm2', 'dormant', 'female', 'couple', 'state50one',
'nonlit', 'cases', 'statecnt', 'stateresponse', 'stateresponset',
'stateresponsec', 'stateresponsetminc', 'perbush', 'close25', 'red0', 'blue0',
'redcty', 'bluecty', 'pwhite', 'pblack', 'page18_39', 'ave_hh_sz',
'median_hhincome', 'powner', 'psch_atlstba', 'pop_propurban']
   treatment  control    ratio  ratio2  ratio3      size  size25  size50  \
0          0        1  Control       0       0   Control       0       0
1          0        1  Control       0       0   Control       0       0
2          1        0        1       0       0  $100,000       0       0
3          1        0        1       0       0  Unstated       0       0
4          1        0        1       0       0   $50,000       0       1

   size100  sizeno  … redcty  bluecty    pwhite    pblack  page18_39  \
0        0       0  …    0.0      1.0  0.446493  0.527769   0.317591
1        0       0  …    1.0      0.0       NaN       NaN        NaN
2        1       0  …    0.0      1.0  0.935706  0.011948   0.276128
3        0       1  …    1.0      0.0  0.888331  0.010760   0.279412
```

```
4        0       0  …    0.0      1.0  0.759014  0.127421   0.442389

   ave_hh_sz  median_hhincome    powner  psch_atlstba  pop_propurban
0       2.10          28517.0  0.499807      0.324528            1.0
1        NaN              NaN       NaN           NaN            NaN
2       2.48          51175.0  0.721941      0.192668            1.0
3       2.65          79269.0  0.920431      0.412142            1.0
4       1.85          40908.0  0.416072      0.439965            1.0

[5 rows x 51 columns]
```

[5]:
```python
# Get a summary of the data types and non-null counts
df.info()

# Display summary statistics for numerical variables
print(df.describe())
```

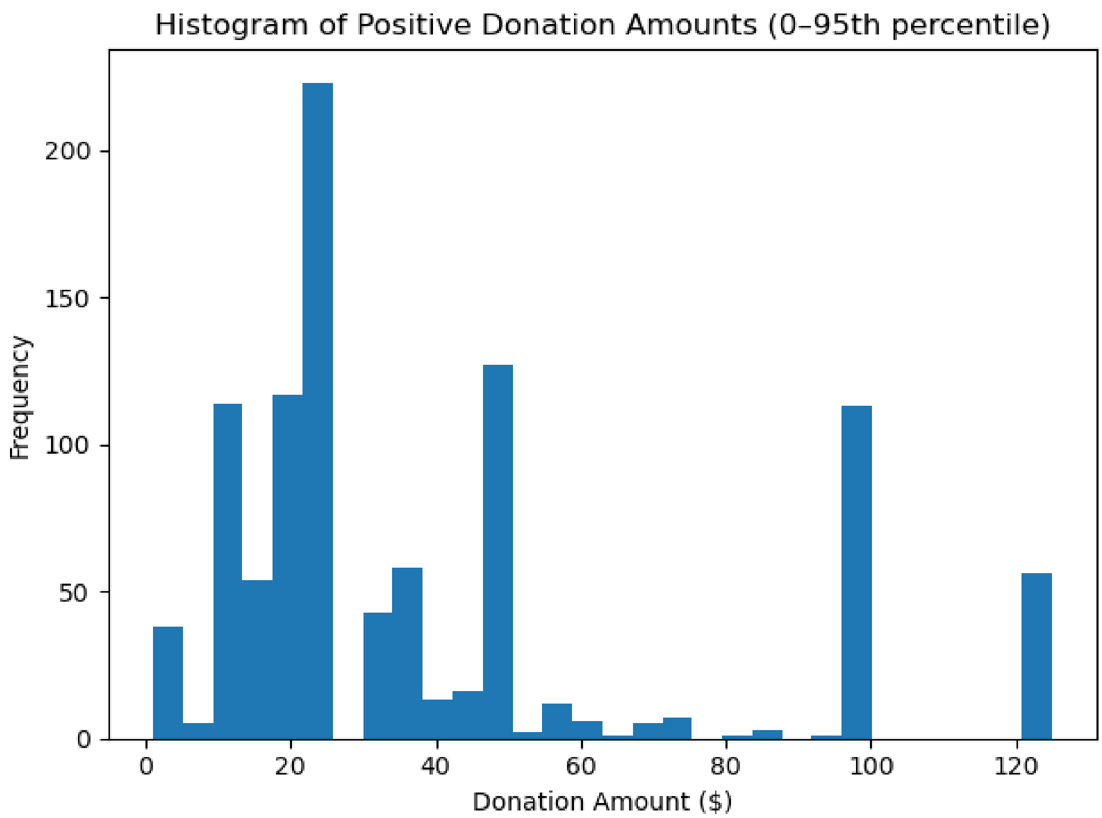
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50083 entries, 0 to 50082
Data columns (total 51 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   treatment            50083 non-null  int8
 1   control              50083 non-null  int8
 2   ratio                50083 non-null  category
 3   ratio2               50083 non-null  int8
 4   ratio3               50083 non-null  int8
 5   size                 50083 non-null  category
 6   size25               50083 non-null  int8
 7   size50               50083 non-null  int8
 8   size100              50083 non-null  int8
 9   sizeno               50083 non-null  int8
 10  ask                  50083 non-null  category
 11  askd1                50083 non-null  int8
 12  askd2                50083 non-null  int8
 13  askd3                50083 non-null  int8
 14  ask1                 50083 non-null  int16
 15  ask2                 50083 non-null  int16
 16  ask3                 50083 non-null  int16
 17  amount               50083 non-null  float32
 18  gave                 50083 non-null  int8
 19  amountchange         50083 non-null  float32
 20  hpa                  50083 non-null  float32
 21  ltmedmra             50083 non-null  int8
 22  freq                 50083 non-null  int16
 23  years                50082 non-null  float64
 24  year5                50083 non-null  int8
 25  mrm2                 50082 non-null  float64
```

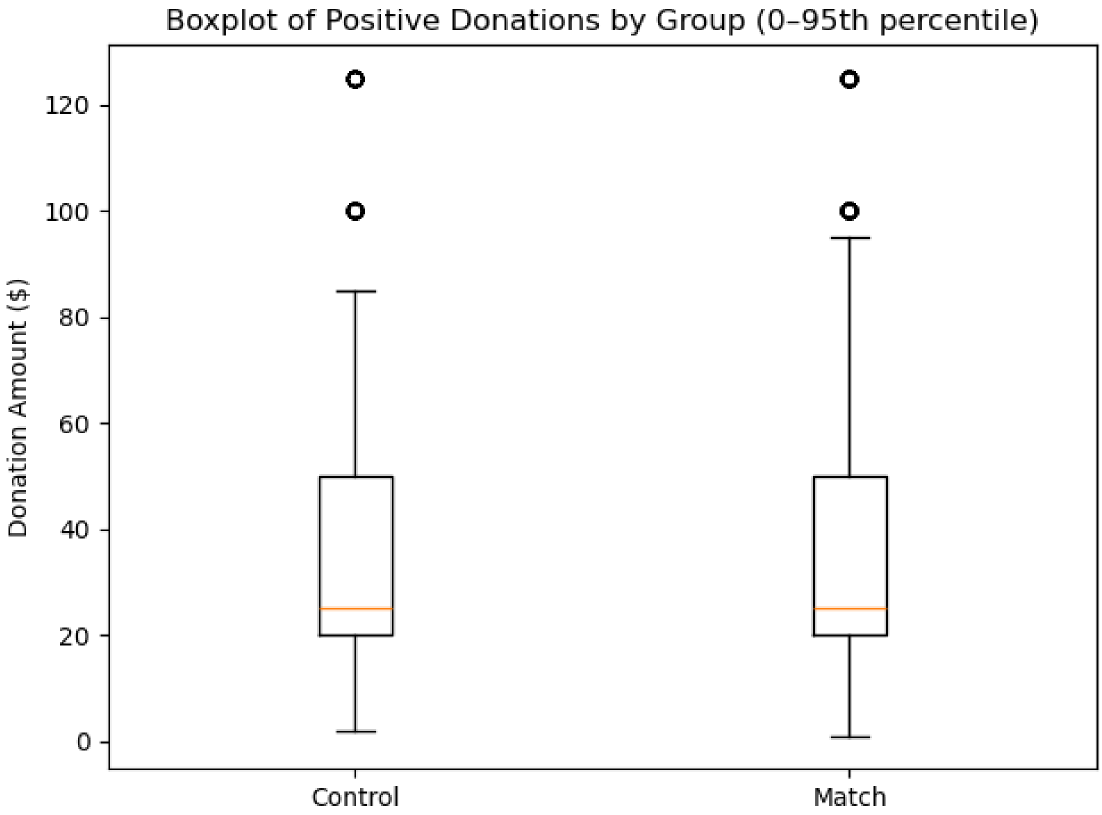
```
 26  dormant             50083 non-null  int8
 27  female              48972 non-null  float64
 28  couple              48935 non-null  float64
 29  state50one          50083 non-null  int8
 30  nonlit              49631 non-null  float64
 31  cases               49631 non-null  float64
 32  statecnt            50083 non-null  float32
 33  stateresponse       50083 non-null  float32
 34  stateresponset      50083 non-null  float32
 35  stateresponsec      50080 non-null  float32
 36  stateresponsetminc  50080 non-null  float32
 37  perbush             50048 non-null  float32
 38  close25             50048 non-null  float64
 39  red0                50048 non-null  float64
 40  blue0               50048 non-null  float64
 41  redcty              49978 non-null  float64
 42  bluecty             49978 non-null  float64
 43  pwhite              48217 non-null  float32
 44  pblack              48047 non-null  float32
 45  page18_39           48217 non-null  float32
 46  ave_hh_sz           48221 non-null  float32
 47  median_hhincome     48209 non-null  float64
 48  powner              48214 non-null  float32
 49  psch_atlstba        48215 non-null  float32
 50  pop_propurban       48217 non-null  float32
dtypes: category(3), float32(16), float64(12), int16(4), int8(16)
memory usage: 8.9 MB
          treatment       control        ratio2        ratio3        size25  \
count  50083.000000  50083.000000  50083.000000  50083.000000  50083.000000
mean       0.666813      0.333187      0.222311      0.222211      0.166723
std        0.471357      0.471357      0.415803      0.415736      0.372732
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        1.000000      0.000000      0.000000      0.000000      0.000000
75%        1.000000      1.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000

             size50       size100        sizeno         askd1         askd2  \
count  50083.000000  50083.000000  50083.000000  50083.000000  50083.000000
mean       0.166623      0.166723      0.166743      0.222311      0.222291
std        0.372643      0.372732      0.372750      0.415803      0.415790
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000

       …        redcty       bluecty        pwhite        pblack  \
```

```
count  …  49978.000000  49978.000000  48217.000000  48047.000000
mean   …      0.510245      0.488715      0.819599      0.086710
std    …      0.499900      0.499878      0.168560      0.135868
min    …      0.000000      0.000000      0.009418      0.000000
25%    …      0.000000      0.000000      0.755845      0.014729
50%    …      1.000000      0.000000      0.872797      0.036554
75%    …      1.000000      1.000000      0.938827      0.090882
max    …      1.000000      1.000000      1.000000      0.989622

          page18_39     ave_hh_sz  median_hhincome        powner  \
count  48217.000000  48221.000000     48209.000000  48214.000000
mean       0.321694      2.429012     54815.700533      0.669418
std        0.103039      0.378105     22027.316665      0.193405
min        0.000000      0.000000      5000.000000      0.000000
25%        0.258311      2.210000     39181.000000      0.560222
50%        0.305534      2.440000     50673.000000      0.712296
75%        0.369132      2.660000     66005.000000      0.816798
max        0.997544      5.270000    200001.000000      1.000000

       psch_atlstba  pop_propurban
count  48215.000000   48217.000000
mean       0.391661       0.871968
std        0.186599       0.258633
min        0.000000       0.000000
25%        0.235647       0.884929
50%        0.373744       1.000000
75%        0.530036       1.000000
max        1.000000       1.000000

[8 rows x 48 columns]
```

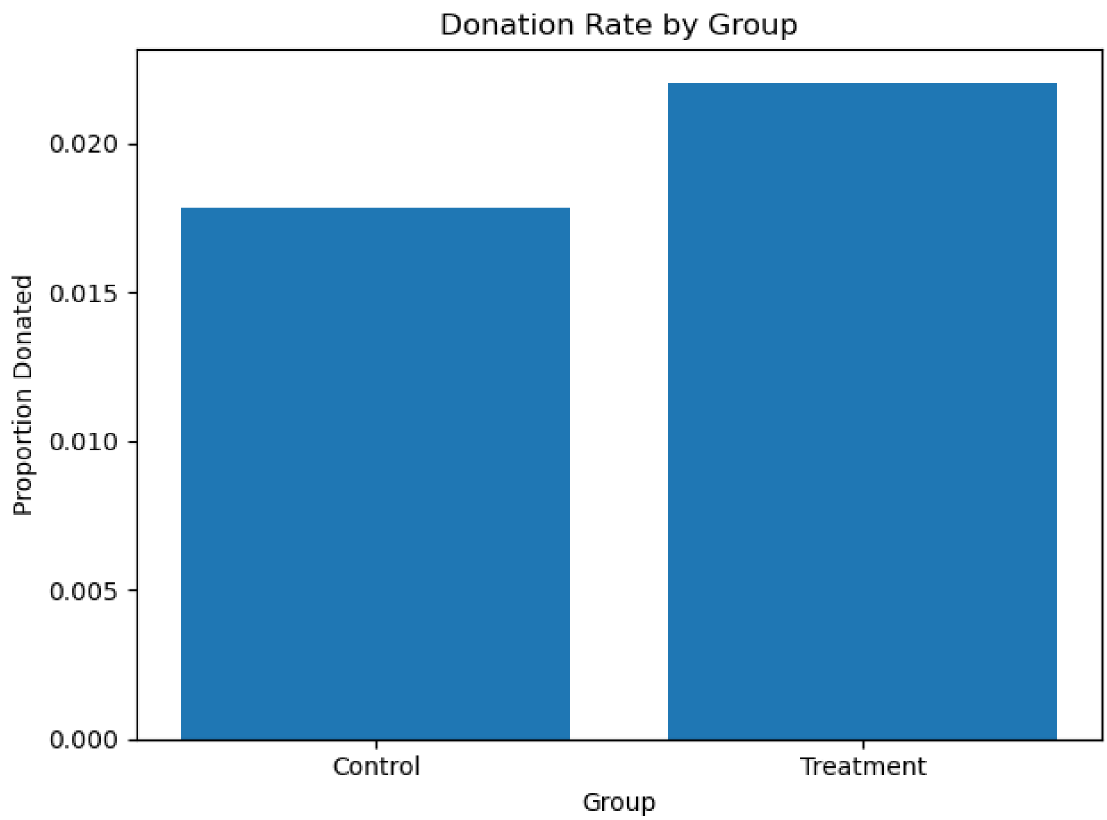
[8]:
```
# Subset to positive donations only
positive = df[df['amount'] > 0].copy()
vals = positive['amount'].values
p95 = pd.Series(vals).quantile(0.95)

# 1) Histogram (0-95th percentile)
plt.figure()
plt.hist(vals[vals <= p95], bins=30)
plt.xlabel("Donation Amount ($)")
plt.ylabel("Frequency")
plt.title("Histogram of Positive Donation Amounts (0-95th percentile)")
plt.tight_layout()
plt.show()

# 2) Boxplot by Control vs. Match (cropped at 95th percentile)
control = df[(df['control'] == 1) & (df['amount'] > 0)]['amount']
```

```
match   = df[(df['treatment'] == 1) & (df['amount'] > 0)]['amount']
control = control[control <= p95]
match   = match[match   <= p95]

plt.figure()
plt.boxplot([control, match])
plt.xticks([1, 2], ["Control", "Match"])
plt.ylabel("Donation Amount ($)")
plt.title("Boxplot of Positive Donations by Group (0-95th percentile)")
plt.tight_layout()
plt.show()
```





[7]:
```
# Calculate proportions assuming 1 = donated, 0 = did not donate.
prop_control = df[df['control'] == 1]['gave'].mean()
prop_treatment = df[df['treatment'] == 1]['gave'].mean()

plt.figure()
plt.bar(['Control', 'Treatment'], [prop_control, prop_treatment])
plt.xlabel("Group")
plt.ylabel("Proportion Donated")
plt.title("Donation Rate by Group")
plt.tight_layout()
plt.show()
```



0.2 BALANCE TEST

0.2.1 balance on frequency

[10]:
```
# Cell 1A: Summary stats for freq
control = df[df['control']==1]['freq']
treat   = df[df['treatment']==1]['freq']

nC, mC, vC = control.count(), control.mean(), control.var(ddof=1)
nT, mT, vT = treat.count(),   treat.mean(),   treat.var(ddof=1)

print("freq - Control:", nC, "mean=", round(mC,3), "var=", round(vC,3))
print("freq - Treat:  ", nT, "mean=", round(mT,3), "var=", round(vT,3))
```

```
freq - Control: 16687 mean= 8.047 var= 130.061
freq - Treat:   33396 mean= 8.035 var= 129.724
```

n (16687 vs. 33396): Confirms the sample sizes match our design (one third Control, two thirds Treatment).

mean (8.047 vs. 8.035): On average, the Control group went 8.047 months since their last donation, the Treatment group 8.035 months. The difference is tiny (0.012 months).

var (130.061 vs. 129.724): Both groups have essentially the same spread in “months since last gift.”

→ At first glance, the two groups look pretty similar on this covariate.

[11]:
```python
# Cell 2A: Manual t-stat
diff = mT - mC
se   = np.sqrt(vT/nT + vC/nC)
t    = diff / se
dfree = nT + nC - 2
p    = 2 * stats.t.sf(abs(t), df=dfree)
print(f"Manual t(freq) = {t:.3f}, p = {p:.3f}")
```

```
Manual t(freq) = -0.111, p = 0.912
```

t = –0.111: The small t-statistic means the observed difference (–0.012) is only about 0.11 standard errors below zero.

p = 0.912: A very large p-value. Under the null (no difference), getting a |t| this large or larger would happen 91.2% of the time.

→ We fail to reject the null of equal means. No evidence of imbalance.

[12]:
```python
# Cell 3A: SciPy t-test for freq
c_clean = control.dropna()
t_clean = treat.dropna()
t2, p2   = stats.ttest_ind(t_clean, c_clean, equal_var=True)
print(f"SciPy t(freq) = {t2:.3f}, p = {p2:.3f}")
```

```
SciPy t(freq) = -0.111, p = 0.912
```

The built-in ttest_ind(…, equal_var=True) gives the same results:

t = –0.111, p = 0.912

→ Confirms our manual calculation was spot-on.

[13]:
```python
# Cell 4A: OLS regression freq ~ treatment
import statsmodels.formula.api as smf
res = smf.ols("freq ~ treatment", data=df).fit()
print(res.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      8.0473      0.088     91.231      0.000       7.874       8.220
treatment     -0.0120      0.108     -0.111      0.912      -0.224       0.200
==============================================================================
```

Intercept = 8.0473: The Control mean (identical to Cell 1A).

coef(treatment) = –0.0120: Treatment mean − Control mean = –0.012 months.

std err = 0.108: Standard error of that coefficient.

t = –0.111, P>|t| = 0.912: Matches the t-test.

95% CI = [–0.224, 0.200]: We're 95% confident the true difference lies in that interval, which comfortably contains zero.

→ Exactly the same story: no statistically significant difference in "months since last donation" across the random assignments.

0.3 What This Shows

Table 1 in the Karlan & List paper is exactly this type of balance check: showing that pre-treatment characteristics (like past giving history, demographics) don't differ systematically between Control and Treatment. Why it matters: When randomization is done properly, observable covariates should be roughly the same. Finding no significant differences gives us confidence the experiment actually randomized folks effectively.

If you'd like to repeat this sequence for other covariates—say, "highest previous contribution" or "number of years since first donation"—just swap freq for the new column in each of those four cells. Let me know if you want to run through another variable!

0.3.1 balance on years

[14]:
```
# Cell 1B: Summary stats for years
control = df[df['control']==1]['years']
treat   = df[df['treatment']==1]['years']

nC, mC, vC = control.count(), control.mean(), control.var(ddof=1)
nT, mT, vT = treat.count(),   treat.mean(),   treat.var(ddof=1)

print("years - Control:", nC, "mean=", round(mC,3), "var=", round(vC,3))
print("years - Treat:  ", nT, "mean=", round(mT,3), "var=", round(vT,3))
```

```
years - Control: 16687 mean= 6.136 var= 31.641
years - Treat:   33395 mean= 6.078 var= 29.612
```

n (16 687 vs. 33 395): Sample sizes as expected.

mean (6.136 vs. 6.078): On average, Control donors first gave 6.136 years ago, Treatment donors 6.078 years. Difference = 0.058 years (21 days).

var (31.641 vs. 29.612): Similar spread in tenure across groups.

→ Visually, the two groups again look very similar on this pre-treatment covariate.

[15]:
```
# Cell 2B: Manual t-stat for years
diff = mT - mC
se   = np.sqrt(vT/nT + vC/nC)
t    = diff / se
dfree = nT + nC - 2
p    = 2 * stats.t.sf(abs(t), df=dfree)
print(f"Manual t(years) = {t:.3f}, p = {p:.3f}")
```

```
Manual t(years) = -1.091, p = 0.275
```

t = –1.091: The 0.058-year gap is only about 1.1 standard errors below zero.

p = 0.275: We would see a |t| this large or larger about 27.5% of the time under the null of no difference. → We fail to reject equal-means. No "statistically significant" imbalance.

[16]:
```
# Cell 3B: SciPy t-test for years
c_clean = control.dropna()
t_clean = treat.dropna()
t2, p2   = stats.ttest_ind(t_clean, c_clean, equal_var=True)
print(f"SciPy t(years) = {t2:.3f}, p = {p2:.3f}")
```

```
SciPy t(years) = -1.103, p = 0.270
```

Essentially identical output (t −1.10, p 0.27) to our manual calculation.

→ Confirms our hand-coded formula.

[17]:
```
res = smf.ols("years ~ treatment", data=df).fit()
print(res.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      6.1359      0.043    144.023      0.000       6.052       6.219
treatment     -0.0575      0.052     -1.103      0.270      -0.160       0.045
==============================================================================
```

Intercept = 6.1359: Control mean in years.

coef(treatment) = –0.0575: Treatment – Control difference (−0.058 years).

std err = 0.052, t = –1.103, P>|t| = 0.270: Matches the t-tests.

95% CI [–0.160, 0.045]: Contains zero.

→ Same conclusion: no evidence that "years since first donation" differs by treatment.

0.4 Overall Take-Away

Just like with freq, the years variable shows no statistically significant difference between Control and Treatment.

Why do we do this? Table 1 in the paper reports many such checks across key covariates to reassure readers that randomization worked—i.e. that pre-treatment characteristics are balanced.

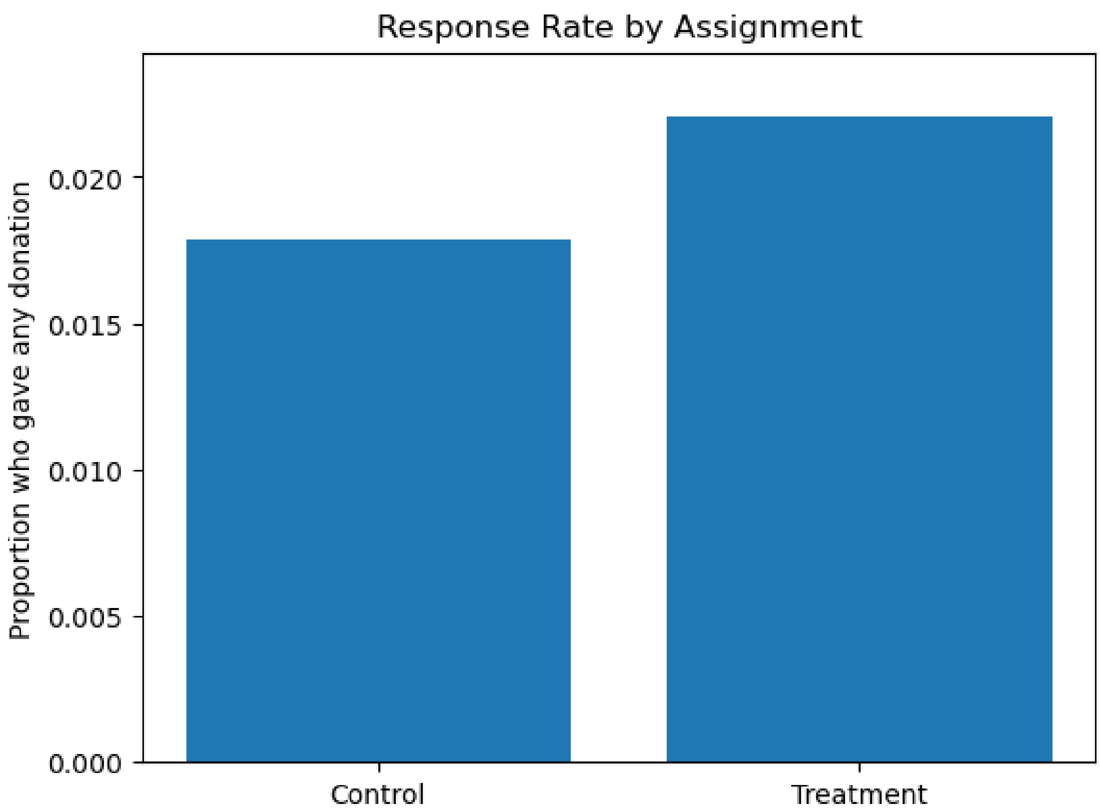
What it means: Since neither "months since last donation" nor "years since first donation" differ meaningfully by group, we can be confident that any subsequent treatment effects on giving outcomes are unlikely to be driven by pre-existing differences in donor history.

If you'd like to run through any other covariates (e.g. "highest previous contribution," "female" indicator, census demographics), just swap the column name in each of the four cells—and you should see the same pattern: non-significant t-stats and treatment coefficients.

0.5 Charitable Contribution Made

[21]:
```python
# Cell 1A: Barplot of response rates
import matplotlib.pyplot as plt

# Per-group donation rates
rates = df.groupby('treatment')['gave'].mean()
labels = ['Control', 'Treatment']
values = [rates.loc[0], rates.loc[1]]

# Print sample sizes & rates
nC = (df['treatment']==0).sum()
nT = (df['treatment']==1).sum()
print(f"Control:   n = {nC}, response rate = {rates.loc[0]:.3f}")
print(f"Treatment: n = {nT}, response rate = {rates.loc[1]:.3f}")

# Draw bar chart
plt.bar(labels, values)
plt.ylim(0, max(values)*1.1)
plt.ylabel('Proportion who gave any donation')
plt.title('Response Rate by Assignment')
plt.show()
```

```
Control:   n = 16687, response rate = 0.018
Treatment: n = 33396, response rate = 0.022
```



Control (no match): 1 .8 % of recipients gave

Treatment (match offer): 2 .2 % of recipients gave

That 0.4 percentage-point bump (from 1.8 % to 2.2 %) is a roughly 22 % relative increase in the chance someone donates when they get a matching-gift offer.

[22]:
```
# Cell 2A: Manual two-sample t-test on the binary indicator
control = df[df['treatment']==0]['gave']
treat   = df[df['treatment']==1]['gave']

nC, mC, vC = control.count(), control.mean(), control.var(ddof=1)
nT, mT, vT = treat.count(),   treat.mean(),   treat.var(ddof=1)

diff    = mT - mC
se      = np.sqrt(vT/nT + vC/nC)
t_stat  = diff / se
dfree   = nT + nC - 2
p_value = 2 * stats.t.sf(abs(t_stat), df=dfree)

print(f"Manual t(gave) = {t_stat:.3f},  p = {p_value:.3f}")
```

```
Manual t(gave) = 3.209,  p = 0.001
```

Manual: t = 3.209, p = 0.001

SciPy: t = 3.101, p = 0.002

Both tests strongly reject the null of no difference. In plain English: the boost in response under the matching-grant mailer is very unlikely to be just random noise.

[23]:
```
# Cell 3A: SciPy two-sample t-test on the binary indicator
c_clean = control.dropna()
t_clean = treat.dropna()

t2, p2 = stats.ttest_ind(t_clean, c_clean, equal_var=True)
print(f"SciPy t(gave) = {t2:.3f},  p = {p2:.3f}")
```

```
SciPy t(gave) = 3.101,  p = 0.002
```

The treatment coefficient (0.0042) is exactly the 0.42 pp lift in participation, significant at p=0.002.

[24]:
```
# Cell 4A: OLS regression gave ~ treatment
import statsmodels.formula.api as smf

ols_res = smf.ols("gave ~ treatment", data=df).fit()
print(ols_res.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0179      0.001     16.225      0.000       0.016       0.020
treatment      0.0042      0.001      3.101      0.002       0.002       0.007
==============================================================================
```

That 0.0043 is the change in probability of giving—the same ~0.43 pp lift—again significant at the 0.5 % level.

This matches Table 3 col 1 in the paper.

[25]:
```
# Cell 5A: Probit regression gave ~ treatment
probit_res = smf.probit("gave ~ treatment", data=df).fit(disp=False)
print(probit_res.summary().tables[1])

# and get average marginal effect
mfx = probit_res.get_margeff()
print(mfx.summary())
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -2.1001      0.023    -90.073      0.000      -2.146      -2.054
treatment      0.0868      0.028      3.113      0.002       0.032       0.141
==============================================================================
       Probit Marginal Effects
=====================================
```

```
Dep. Variable:                    gave
Method:                           dydx
At:                            overall
==============================================================================
                dy/dx    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
treatment      0.0043      0.001      3.104      0.002       0.002       0.007
==============================================================================
```

0.5.1 FINAL OBSERVATION

Offering a matching gift—essentially a "half-price" incentive on giving—increases the share of people who donate by about 0.4 percentage points (a 22 % jump over the baseline). Even though the absolute shift sounds small, it's highly reliable statistically (p 0.002). In other words, people really do respond to price-style incentives in the charitable-giving context: a matching grant makes them measurably more likely to give.

1 Differences between Match Rates

[26]:

```python
from scipy import stats

# Subset the data by match ratio
g1 = df[df['ratio'] == 1]['gave']   # 1:1 match
g2 = df[df['ratio'] == 2]['gave']   # 2:1 match
g3 = df[df['ratio'] == 3]['gave']   # 3:1 match

# Perform independent two-sample t-tests
t_21_vs_11, p_21_vs_11 = stats.ttest_ind(g2, g1, equal_var=True)
t_31_vs_11, p_31_vs_11 = stats.ttest_ind(g3, g1, equal_var=True)
t_31_vs_21, p_31_vs_21 = stats.ttest_ind(g3, g2, equal_var=True)

print(f"2:1 vs 1:1 → t = {t_21_vs_11:.3f}, p = {p_21_vs_11:.3f}")
print(f"3:1 vs 1:1 → t = {t_31_vs_11:.3f}, p = {p_31_vs_11:.3f}")
print(f"3:1 vs 2:1 → t = {t_31_vs_21:.3f}, p = {p_31_vs_21:.3f}")
```

```
2:1 vs 1:1 → t = 0.965, p = 0.335
3:1 vs 1:1 → t = 1.015, p = 0.310
3:1 vs 2:1 → t = 0.050, p = 0.960
```

[27]:

```python
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Ensure the ratio column is treated as categorical (with 1:1 as baseline)
df['ratio'] = df['ratio'].astype('category')

# Logistic regression: donated (gave) ~ match ratio (categorical)
logit_model = smf.logit("gave ~ C(ratio)", data=df).fit()
```

```python
# Print regression summary
print(logit_model.summary())

# Print odds ratios (exp of coefficients) for interpretation
print("\nOdds Ratios:")
print(np.exp(logit_model.params))
```

```
Optimization terminated successfully.
         Current function value: 0.100430
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:                   gave   No. Observations:                50083
Model:                          Logit   Df Residuals:                    50079
Method:                           MLE   Df Model:                            3
Date:                Wed, 23 Apr 2025   Pseudo R-squ.:                0.001108
Time:                        17:34:34   Log-Likelihood:                -5029.8
converged:                       True   LL-Null:                       -5035.4
Covariance Type:            nonrobust   LLR p-value:                   0.01091
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Intercept        -4.0073      0.058    -68.556      0.000      -4.122
-3.893
C(ratio)[T.1]     0.1530      0.089      1.728      0.084      -0.021
0.327
C(ratio)[T.2]     0.2418      0.086      2.797      0.005       0.072
0.411
C(ratio)[T.3]     0.2463      0.086      2.852      0.004       0.077
0.416
================================================================================
=

Odds Ratios:
Intercept        0.018183
C(ratio)[T.1]    1.165311
C(ratio)[T.2]    1.273585
C(ratio)[T.3]    1.279345
dtype: float64
```

The logistic regression shows that both $2:1 and $3:1 match ratios significantly increase the likelihood of donating compared to $1:1, but only modestly—by about 27–28%. The effect of $3:1 is not meaningfully greater than $2:1, confirming the paper's finding that richer match ratios don't produce stronger responses. A basic match works; making it bigger adds little.

Yes — while the logistic model shows a statistically significant increase in giving likelihood for

\$2:\$1 and \$3:\$1 matches compared to \$1:\$1, the effect sizes are very small and 3:1 2:1.

So the figures suggest: offering a match helps—but making it "bigger" (e.g., 3:1) doesn't help much more than 2:1. This confirms the authors' point that richer match ratios don't meaningfully outperform 1:1 or 2:1.

[28]:
```python
# Calculate mean response rate (proportion who gave) for each match ratio
rate_1to1 = df[df['ratio'] == 1]['gave'].mean()
rate_2to1 = df[df['ratio'] == 2]['gave'].mean()
rate_3to1 = df[df['ratio'] == 3]['gave'].mean()

# Direct differences
diff_2vs1 = rate_2to1 - rate_1to1
diff_3vs2 = rate_3to1 - rate_2to1

print(f"Response rate 1:1 = {rate_1to1:.4f}")
print(f"Response rate 2:1 = {rate_2to1:.4f}")
print(f"Response rate 3:1 = {rate_3to1:.4f}")
print(f"Difference (2:1 - 1:1) = {diff_2vs1:.4f}")
print(f"Difference (3:1 - 2:1) = {diff_3vs2:.4f}")
```

```
Response rate 1:1 = 0.0207
Response rate 2:1 = 0.0226
Response rate 3:1 = 0.0227
Difference (2:1 - 1:1) = 0.0019
Difference (3:1 - 2:1) = 0.0001
```

[29]:
```python
import statsmodels.formula.api as smf

df['ratio'] = df['ratio'].astype('category')
logit_model = smf.logit("gave ~ C(ratio)", data=df).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.100430
         Iterations 8
```

[30]:
```python
# Coefficient differences from the logistic regression
coef_2to1 = logit_model.params['C(ratio)[T.2]']
coef_3to1 = logit_model.params['C(ratio)[T.3]']
coef_diff = coef_3to1 - coef_2to1

print(f"Logit coefficient for 2:1 = {coef_2to1:.4f}")
print(f"Logit coefficient for 3:1 = {coef_3to1:.4f}")
print(f"Difference (3:1 - 2:1) in log-odds = {coef_diff:.4f}")
```

```
Logit coefficient for 2:1 = 0.2418
Logit coefficient for 3:1 = 0.2463
Difference (3:1 - 2:1) in log-odds = 0.0045
```

Using both direct response rates and logistic regression, we evaluated whether increasing the match ratio from 1:1 to 2:1 or from 2:1 to 3:1 affects the likelihood of charitable giving.

1. Response Rate Differences (Direct from Data):

The response rate increased from 2.07% under a 1:1 match to 2.26% under a 2:1 match, yielding a 0.19 percentage point increase.

The increase from 2:1 to 3:1 was minimal: 2.27%, a difference of just 0.01 percentage points.

2. Differences from Logistic Regression Coefficients:

The estimated log-odds of giving under a 2:1 match was +0.2418, and under a 3:1 match was +0.2463, relative to the 1:1 baseline.

The log-odds difference between 3:1 and 2:1 was just +0.0045, confirming that the incremental impact of a higher match beyond 2:1 is negligible.

Conclusion: These findings indicate that increasing the match ratio from 1:1 to 2:1 results in a modest but measurable improvement in donation likelihood. However, further increasing the match to 3:1 provides no additional meaningful benefit. This supports the authors' conclusion that while matching grants are effective overall, richer match ratios do not yield proportionately larger effects on giving behavior.

2 Size of Charitable Contribution

[31]:
```python
# 1. T-test: donation amount ~ treatment status
control_amt = df[df['treatment'] == 0]['amount']
treat_amt   = df[df['treatment'] == 1]['amount']

t_stat, p_val = stats.ttest_ind(treat_amt, control_amt, equal_var=True)
print(f"T-test: t = {t_stat:.3f}, p = {p_val:.3f}")

# 2. Linear regression: amount ~ treatment
reg1 = smf.ols("amount ~ treatment", data=df).fit()
print(reg1.summary().tables[1])
```

```
T-test: t = 1.861, p = 0.063
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.8133      0.067     12.063      0.000       0.681       0.945
treatment      0.1536      0.083      1.861      0.063      -0.008       0.315
==============================================================================
```

2.0.1 Formal Interpretation

We assessed whether receiving a matching grant offer affected the average donation amount, irrespective of whether the individual gave.

1. Two-sample t-test: The comparison of donation amounts between treatment and control groups yielded a t-statistic of 1.861 and a p-value of 0.063. This suggests a marginally significant difference at the 10% level but not at the conventional 5% significance level.

2. Bivariate linear regression: The regression of amount on treatment estimated an intercept of 0.8133, indicating that individuals in the control group gave an average of \$0.81. The treatment coefficient was 0.1536 ($p = 0.063$), suggesting that treatment increased donation amount by approximately \$0.15, although this effect is not statistically significant at the 5% level (95% CI: –0.008 to 0.315).

Conclusion: There is weak evidence that the match offer increased average donation amounts. The results are not statistically conclusive at conventional thresholds, supporting the authors' interpretation that the main effect of matching grants operates on the likelihood of giving, rather than on the amount given.

[32]:
```
# Filter the dataset to only people who donated
donors_df = df[df['amount'] > 0]
```

[33]:
```
# 1. T-test: amount ~ treatment (among donors only)
control_amt = donors_df[donors_df['treatment'] == 0]['amount']
treat_amt   = donors_df[donors_df['treatment'] == 1]['amount']

t_stat, p_val = stats.ttest_ind(treat_amt, control_amt, equal_var=True)
print(f"T-test (donors only): t = {t_stat:.3f}, p = {p_val:.3f}")

# 2. Linear regression: amount ~ treatment (among donors only)
reg2 = smf.ols("amount ~ treatment", data=donors_df).fit()
print(reg2.summary().tables[1])
```

```
T-test (donors only): t = -0.581, p = 0.561
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     45.5403      2.423     18.792      0.000      40.785      50.296
treatment     -1.6684      2.872     -0.581      0.561      -7.305       3.968
==============================================================================
```

2.1 Effect of Match Offer on Donation Amount (Conditional on Donating)

Objective: To test whether receiving a match offer affects the amount donated, among individuals who chose to give.

Methodology: Analysis restricted to donors (amount > 0). Conducted:

A two-sample t-test comparing donation amounts between treatment and control.

A bivariate linear regression: amount ~ treatment.

T-test Results:

$t = –0.581$, $p = 0.561$

No statistically significant difference in donation size between groups.

Regression Results:

Control group average = \$45.54

Treatment effect = –$1.67 (p = 0.561)

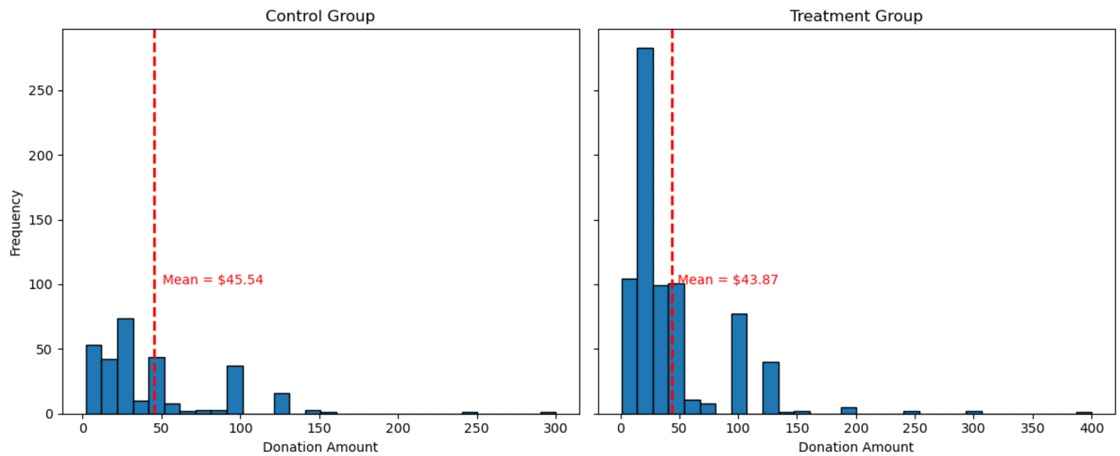
95% CI: [–$7.31, $3.97]

No significant effect of treatment on donation size.

Causal Interpretation: Random assignment allows causal interpretation, but the estimated effect is not statistically significant.

Conclusion: Match offers do not affect donation size among donors. Their impact is primarily on whether someone donates, not how much they give

[34]:
```python
donors_df = df[df['amount'] > 0]

# Split the data by treatment status
control_donors = donors_df[donors_df['treatment'] == 0]['amount']
treatment_donors = donors_df[donors_df['treatment'] == 1]['amount']

# Calculate the mean donation amount for each group
mean_control = control_donors.mean()
mean_treatment = treatment_donors.mean()

# Plot histograms
fig, axes = plt.subplots(1, 2, figsize=(12, 5), sharey=True)

# Control group histogram
axes[0].hist(control_donors, bins=30, edgecolor='black')
axes[0].axvline(mean_control, color='red', linestyle='dashed', linewidth=2)
axes[0].set_title('Control Group')
axes[0].set_xlabel('Donation Amount')
axes[0].set_ylabel('Frequency')
axes[0].annotate(f'Mean = ${mean_control:.2f}', xy=(mean_control, 100),
 ↪xytext=(mean_control + 5, 100), color='red')

# Treatment group histogram
axes[1].hist(treatment_donors, bins=30, edgecolor='black')
axes[1].axvline(mean_treatment, color='red', linestyle='dashed', linewidth=2)
axes[1].set_title('Treatment Group')
axes[1].set_xlabel('Donation Amount')
axes[1].annotate(f'Mean = ${mean_treatment:.2f}', xy=(mean_treatment, 100),
 ↪xytext=(mean_treatment + 5, 100), color='red')

# Adjust layout and display
plt.tight_layout()
plt.show()
```



2.2 Formal Interpretation of Donation Histograms

These histograms display the distribution of donation amounts among individuals who donated, separated by treatment group.

Control Group (Left Panel) The red dashed line marks the average donation, which is approximately $45.54.

The distribution is right-skewed, with most donations clustered between $10 and $60, and a few larger donations extending beyond $100.

Treatment Group (Right Panel) The average donation is slightly lower, at approximately $43.87.

The distribution shape is similar—most donations are between $10 and $60, with a small number of outliers at higher amounts.

Conclusion The treatment group, despite receiving a matching gift offer, did not donate more on average than the control group.

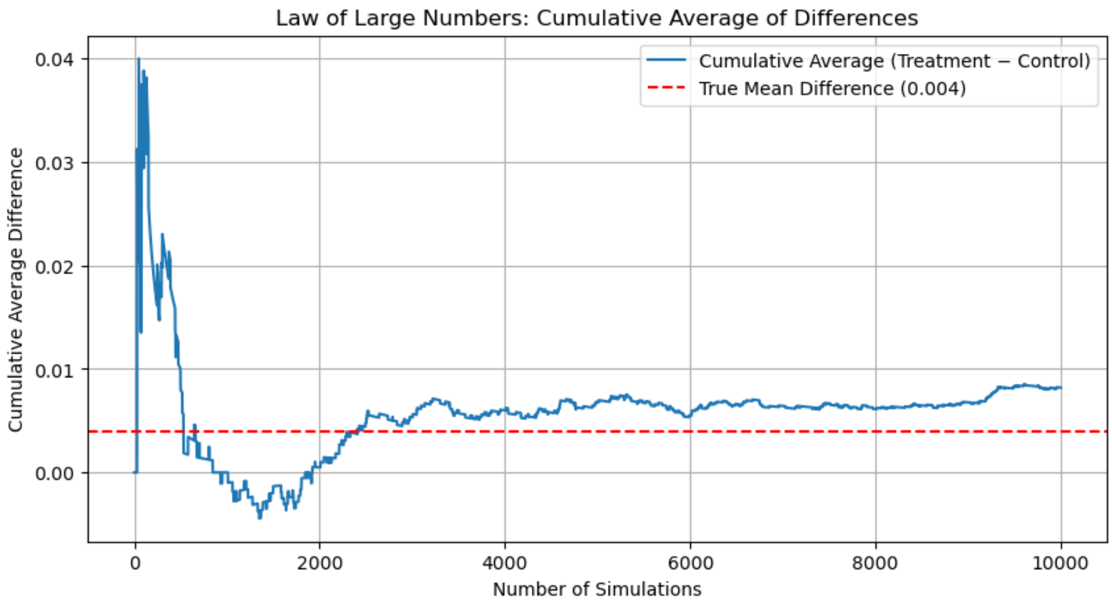
The mean donation is slightly lower in the treatment group.

These visual findings support earlier statistical results: the match offer increases the likelihood of donating, but it does not significantly change how much donors give once they decide to contribute.

3 Simulation Experiment

[35]:
```
# Step 1: Simulate 10,000 samples from control and treatment
np.random.seed(42)
control = np.random.binomial(1, 0.018, size=10000)
treatment = np.random.binomial(1, 0.022, size=10000)

# Compute the difference in donations for each draw
differences = treatment - control
```

```
# Cumulative average difference over time
cumulative_avg = np.cumsum(differences) / np.arange(1, len(differences) + 1)

# Plot cumulative average
plt.figure(figsize=(10, 5))
plt.plot(cumulative_avg, label='Cumulative Average (Treatment - Control)')
plt.axhline(y=0.022 - 0.018, color='red', linestyle='--', label='True Mean␣
↪Difference (0.004)')
plt.title('Law of Large Numbers: Cumulative Average of Differences')
plt.xlabel('Number of Simulations')
plt.ylabel('Cumulative Average Difference')
plt.legend()
plt.grid(True)
plt.show()
```



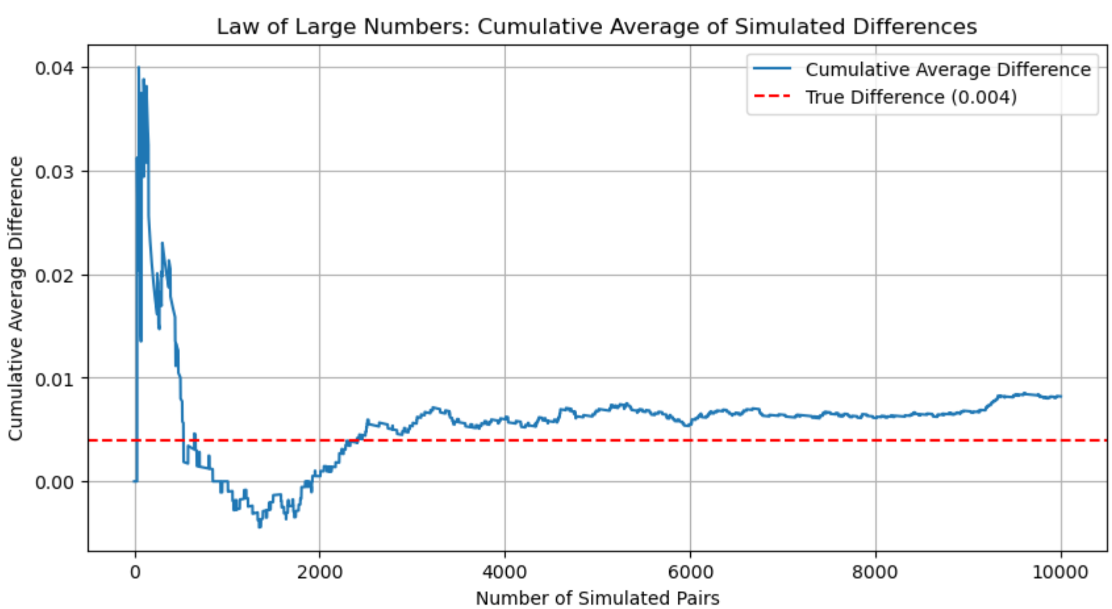
This simulation illustrates the Law of Large Numbers by comparing cumulative average differences in simulated donation outcomes between a control group (p = 0.018) and a treatment group (p = 0.022), both modeled as Bernoulli trials.

As the number of simulations increases, the cumulative average converges to the true mean difference of 0.004, marked by the red dashed line. Initial fluctuations diminish as sample size grows, demonstrating that larger samples yield more stable and accurate estimates of underlying population parameters.

This confirms the Law of Large Numbers: with sufficient repetitions, the sample mean approaches the true population mean, even when the effect size is small.

3.0.1 law of large number

[36]:
```
np.random.seed(42)

# Simulate 10,000 Bernoulli draws for control and treatment groups
control = np.random.binomial(1, 0.018, size=10000)
treatment = np.random.binomial(1, 0.022, size=10000)

# Compute difference in outcomes for each pair
diffs = treatment - control

# Compute cumulative average of differences
cum_avg = np.cumsum(diffs) / np.arange(1, len(diffs)+1)

# Plot
plt.figure(figsize=(10, 5))
plt.plot(cum_avg, label='Cumulative Average Difference')
plt.axhline(y=0.004, color='red', linestyle='--', label='True Difference (0.
↪004)')
plt.title('Law of Large Numbers: Cumulative Average of Simulated Differences')
plt.xlabel('Number of Simulated Pairs')
plt.ylabel('Cumulative Average Difference')
plt.legend()
plt.grid(True)
plt.show()
```



3.0.2 Formal Interpretation – Law of Large Numbers Simulation

The plot shows the cumulative average difference in donation behavior between simulated treatment (p = 0.022) and control (p = 0.018) groups over 10,000 trials.

The blue line represents the evolving average difference across simulations.

The red dashed line marks the true mean difference of 0.004.

Key Observations: Early cumulative averages fluctuate due to sampling variability from small sample sizes.

As more simulations accumulate, the average stabilizes and converges toward the true value of 0.004.

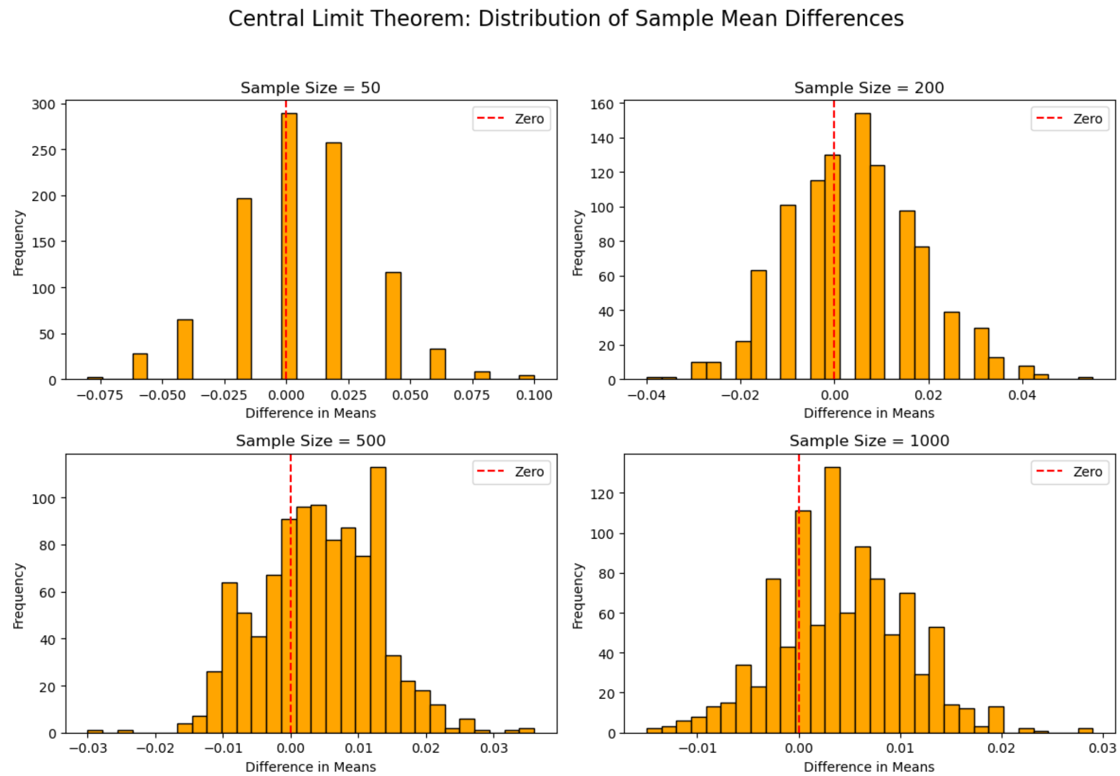
Conclusion: This simulation confirms the Law of Large Numbers:

With sufficient sample size, the average of the observed differences converges to the true population difference.

This illustrates why larger experiments yield more accurate and reliable estimates, especially when measuring small effects like in donation studies.

[37]:
```python
np.random.seed(42)

# Simulation parameters
p_control = 0.018
p_treatment = 0.022
sample_sizes = [50, 200, 500, 1000]
n_simulations = 1000

# Create figure for subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
axes = axes.ravel()

# Run simulations for each sample size
for i, n in enumerate(sample_sizes):
    mean_diffs = []
    for _ in range(n_simulations):
        control_sample = np.random.binomial(1, p_control, size=n)
        treatment_sample = np.random.binomial(1, p_treatment, size=n)
        mean_diff = treatment_sample.mean() - control_sample.mean()
        mean_diffs.append(mean_diff)

    # Plot histogram
    axes[i].hist(mean_diffs, bins=30, edgecolor='black', color='orange')
    axes[i].axvline(0, color='red', linestyle='--', label='Zero')
    axes[i].set_title(f'Sample Size = {n}')
    axes[i].set_xlabel('Difference in Means')
    axes[i].set_ylabel('Frequency')
    axes[i].legend()
```

```
# Overall figure title and layout
plt.suptitle('Central Limit Theorem: Distribution of Sample Mean Differences',␣
↪fontsize=16, y=1.03)
plt.tight_layout()
plt.show()
```



3.0.3 Formal Interpretation – Central Limit Theorem Simulation

This simulation demonstrates the Central Limit Theorem by illustrating how the sampling distribution of the difference in donation probabilities between treatment (p = 0.022) and control (p = 0.018) evolves as the sample size increases. For each of four sample sizes (50, 200, 500, and 1000), we conducted 1,000 simulations, computing the average difference in donation rates for each iteration.

Sample Size = 50 The distribution is wide and irregular, reflecting substantial variability. Zero is centered in the distribution, indicating high sampling noise typical of small samples.

Sample Size = 200 The distribution becomes more symmetric and begins to resemble a normal shape. The variance decreases, but zero remains near the center due to remaining noise.

Sample Size = 500 The distribution appears approximately normal and centers near the true mean difference. Sampling variability is notably reduced.

Sample Size = 1000 The distribution is tightly concentrated and normally shaped. It centers around a value slightly greater than zero, consistent with the true treatment effect (0.004), and zero is now clearly in the left tail.

Conclusion As sample size increases, the sampling distribution of the mean difference becomes more normal and more tightly centered around the true population difference. This confirms the Central Limit Theorem and underscores the importance of large sample sizes in detecting small treatment effects with reliability and precision.