Political Persuasion

Voter-Persuasion.csv is the dataset for this case study. *Note: Our thanks to Ken Strasma, President of HaystaqDNA and director of targeting for the 2004 Kerry campaign and the 2008 Obama campaign, for the data used in this case, and for sharing the information in the following writeup.*

Background

When you think of political persuasion, you may think of the efforts that political campaigns undertake to persuade you that their candidate is better than the other candidate. In truth, campaigns are less about persuading people to change their minds, and more about persuading those who agree with you to actually go out and vote. Predictive analytics now plays a big role in this effort, but in 2004, it was a new arrival in the political toolbox.

Predictive Analytics Arrives in the US Politics

In January of 2004, candidates in the US presidential campaign were competing in the Iowa caucuses, part of the lengthy state-by-state primary campaign that culminates in the selection of the Republican and Democratic candidates for president. Among the Democrats, Howard Dean was leading in national polls. The Iowa caucuses, however, are a complex and intensive process attracting only the most committed and interested voters. Those participating are not a representative sample of voters nationwide. Surveys of those planning to take part showed a close race between Dean and three other candidates, including John Kerry.

Kerry ended up winning by a surprisingly large margin, and the better than expected performance was due to his campaign's innovative and successful use of predictive analytics to learn more about the likely actions of individual voters. This allowed the campaign to target voters in such a way as to optimize performance in the caucuses. For example, once the model showed sufficient support in a precinct to win that precinct's delegate to the caucus, money and time could be redirected to other precincts where the race was closer.

Political Targeting

Targeting of voters is not new in politics. It has traditionally taken three forms:

- Geographic
- Demographic
- Individual

In geographic targeting, resources are directed to a geographic unit-state, city, country, etc.-on the basis of prior voting patterns or surveys that reveal the political tendency in that geographic unit. It has significant limitations, though. If a county is only, say 52% in your favor, it may be in the greatest need of attention, but if the messaging is directed to everyone in the country, nearly half of it is reaching the wrong people.

In demographic targeting, the messaging is intended for demographic groups-for example, older voters, younger women voters, Hispanic voters, etc. The limitation of this method is that it is often not easy to implement: messaging is hard to deliver just to single demographic groups.

Traditional individual targeting, the most effective form of targeting, was done on the basis of surveys asking voters how they plan to vote. The big limitation of this method is, of course, the cost. The expense of reaching all voters in a phone or door-to-door survey can be prohibitive.

The use of predictive analytics adds power to the individual targeting method, and reduces cost. A model allows prediction to be rolled out to the entire voter base, not just those surveyed, and brings to bear a wealth of information. Geographic and demographic data remain part of the picture, but they are used at an individual level.

Uplift

In a classical predictive modeling application for marketing, a sample of data is selected and an offer is made (e.g., on the web) or a message is sent (e.g., by mail), and a predictive model is developed to classify individuals as responding or not-responding. The model is then applied to new data, propensities to respond are calculated, individuals are ranked by their propensity to respond, and the marketer can then select those most likely to respond to mailings or offers.

Some key information is missing from this classical approach: how would the individual respond in the absence of the offer or mailing? Might a high-propensity customer be inclined to purchase irrespective of the offer? Might a person's propensity to buy actually be diminished by the offer? Uplift modeling allows us to estimate the effect of "offer vs. no offer" or "mailing vs. no mailing" at the individual level.

In this case, we will aplly uplift modeling to actual voter data that were augmented with the results of a hypothetical experiment. The experiment consisted of the following steps:

- Conduct a pre-survey of the voters to determine their inclination to vote Democratic.
- Randomly split the voters into two samples-control and treatment.
- Send a flyer promoting the Democratic candidate to the treatment group.
- Conduct another survey of the voters to determine their inclination to vote Democratic.

Data

The data in this case are in the file **Voter-Persuasion.csv**. The target variable **MOVED_AD**, where a 1 = "opinion moved in favor of the Democratic candidate" and 0 = "opinion did not move in favor of the Democratic candidate." This variable encapsulates the information from the pre- and post-surveys. The important predictor variable is **Flyer**, a binary variable that indicates whether or not a voter received the flyer. In addition, there are numerous other predictor variables from these sources:

- Government voter files
- Political party data
- Commercial consumer and demographic data
- Census neighborhood data

Governement voter files are maintained, and made public, to assure the integrity of the voting process. They contain essential data for identification purposes such as name, address and date of birth. The file used in this case also contains party identification (needed if a state limits participation in party primaries to voters in that party). Parties also staff elections with their own poll-watchers, who record whether an individual votes in an election. These data (termed "derived" in the case data) are maintained and curated by each party, and can be readily matched to the voter data by name.

Demographic data at the neighborhood level are available from the census, and can be appended to the voter data by addresss matching. Consumer and additional demographic data (buying habits, education) can be purchased from marketing firms and appended to the voter data (matching by name and address).

Assignment: Political Persuasion, CART, Random Forests, and Uplift Modeling

Questions

1. Overall Effect of the Flyer.

Overall, how well did the flyer do in moving voters in a Democratic direction? (Look at the target variable among those who got the flyer, compared to those who did not.)

34.44% of the voters who did not receive the flyer moved in a Democratic direction while 40.24% of the voters who did receive the flyer moved in a Democratic direction. This shows an increase of about 5.8 percentage points, suggesting that the flyer successfully moved voters in a Democratic direction on average.

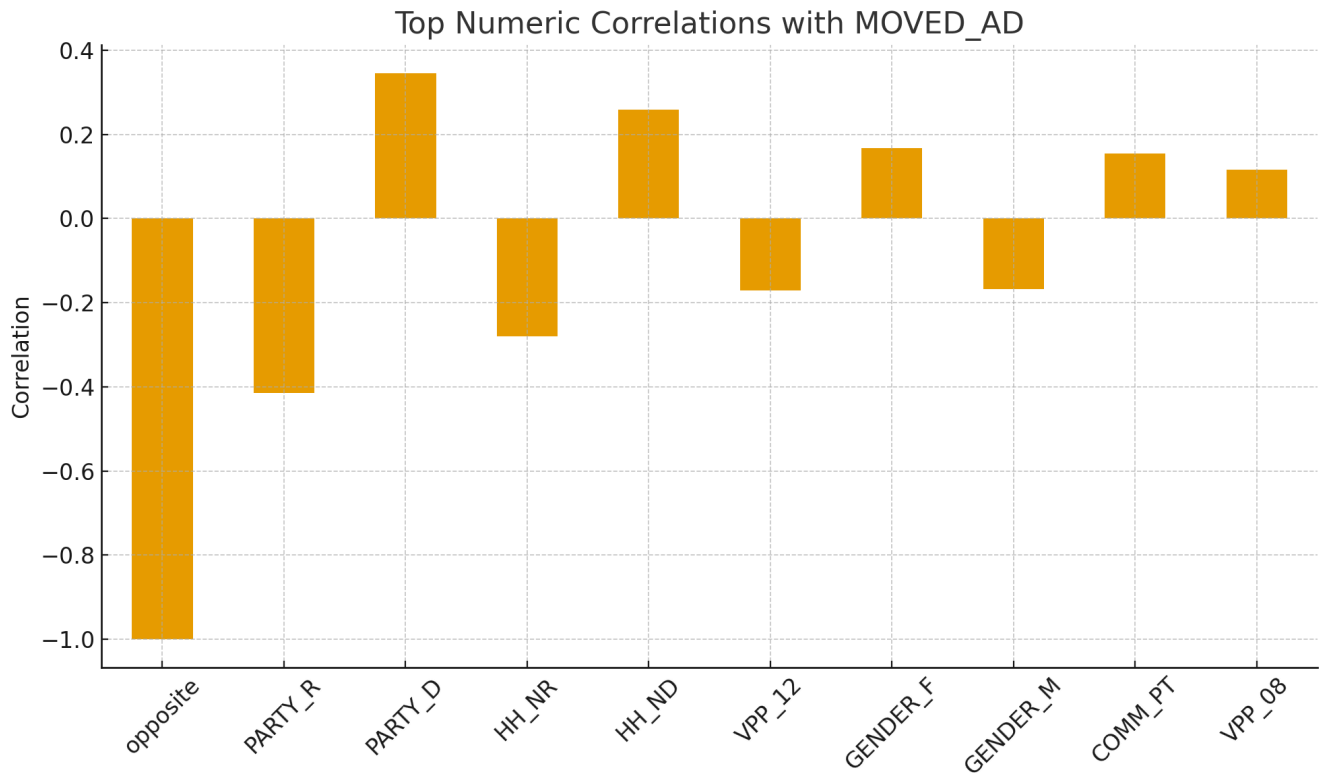
```
import pandas as pd
df = pd.read_csv('/content/Voter-Persuasion.csv')

# convert target to numeric
df['MOVED_AD_num'] = df['MOVED_AD'].map({'Y':1,'N':0})

# grouping
overall = df['MOVED_AD_num'].mean()
by_flyer = df.groupby('FLYER')['MOVED_AD_num'].agg(['mean','count'])
print("Overall", overall)
print(by_flyer)
```

```
Overall 0.3734
         mean  count
FLYER
0      0.3444   5000
1      0.4024   5000
```

---

2. Exploring Predictive Relationships

Explore the data to learn more about the relationships between the predictor variables and *MOVED_AD* using data visualization. Which of the predictors seem to have good predictive potential? Show supporting charts and/or tables.

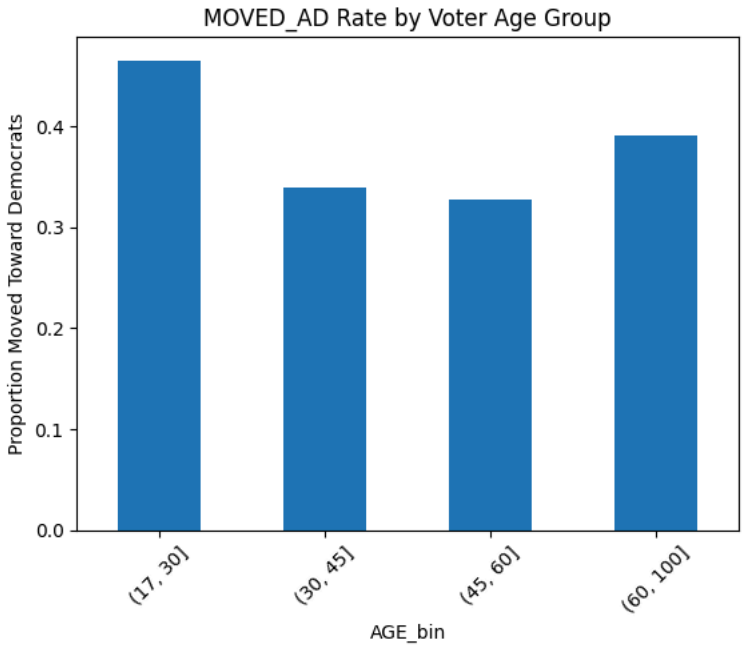


This bar chart of the top numeric correlations with MOVED_AD helps to identify which variables are most likely to matter in CART or Random Forest.

Additionally, I plotted the persuasion rate (MOVED_AD = 1) across meaningful predictors to see which might help future models. This first one is about persuasion rate by age group.

```
import matplotlib.pyplot as plt

df = pd.read_csv("/content/Voter-Persuasion.csv")

# convert target variable to numeric
df["MOVED_AD_num"] = df["MOVED_AD"].map({"Y": 1, "N": 0})

# persuasion rate by age group
df["AGE_bin"] = pd.cut(df["AGE"], bins=[17,30,45,60,100])

age_rates = df.groupby("AGE_bin")["MOVED_AD_num"].mean()

plt.figure()
age_rates.plot(kind="bar")
plt.title("MOVED_AD Rate by Voter Age Group")
plt.ylabel("Proportion Moved Toward Democrats")
plt.xticks(rotation=45)
plt.show()
```

```
/tmp/ipython-input-3168554012.py:11: FutureWarning: The default of observed=False is deprecated and will be changed to True in a
  age_rates = df.groupby("AGE_bin")["MOVED_AD_num"].mean()
```

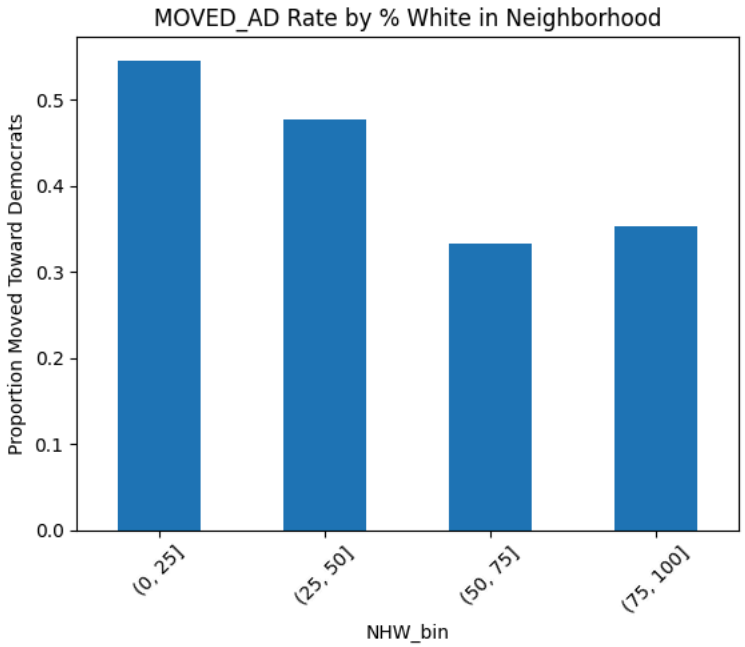


The persuasion rate by age group has strong predictive potential with younger voters, ages 17-30, are significantly more likely to move in a Democratic direction.

Next, I plotted the persuasion by neighborhood race.

```
df["NHW_bin"] = pd.cut(df["NH_WHITE"], bins=[0,25,50,75,100])

nh_rates = df.groupby("NHW_bin")["MOVED_AD_num"].mean()

plt.figure()
nh_rates.plot(kind="bar")
plt.title("MOVED_AD Rate by % White in Neighborhood")
plt.ylabel("Proportion Moved Toward Democrats")
plt.xticks(rotation=45)
plt.show()
```

```
/tmp/ipython-input-642862368.py:3: FutureWarning: The default of observed=False is deprecated and will be changed to True in a f
  nh_rates = df.groupby("NHW_bin")["MOVED_AD_num"].mean()
```



The persuasion by neighborhood racial composition shows that voters in more diverse neighborhoods, neighborhoods with a lower percentage of white people, moved at much higher rates towards the Democratic direction.

And finally I plotted the persuasion by registered party.

```
party_vars = ["PARTY_D","PARTY_I","PARTY_R"]

for col in party_vars:
    table = df.groupby(col)["MOVED_AD_num"].mean().reset_index()
    print(f"\n=== MOVED_AD by {col} ===")
    display(table)
```

```
=== MOVED_AD by PARTY_D ===
```

	PARTY_D	MOVED_AD_num
0	0	0.214571
1	1	0.549441

```
=== MOVED_AD by PARTY_I ===
```

	PARTY_I	MOVED_AD_num
0	0	0.365355
1	1	0.400703

```
=== MOVED_AD by PARTY_R ===
```

	PARTY_R	MOVED_AD_num
0	0	0.500350
1	1	0.055692

Next steps: Generate code with `table` | New interactive sheet | Generate code with `table` | New interactive sheet | Generate code with `table`

The persuasion by registered party shows that non-Republicans were more persuadable than people who were registered as Republicans who were less likely to move towards the Democratic direction.

---

3. Train-Test Split and Baseline Accuracy

Now, split the data into a training set and a testing set using the partition variable that is in the dataset. What is the accuracy on the testing set of a simple baseline method that always predicts "did not move" for every voter?

```
df = pd.read_csv("/content/Voter-Persuasion.csv")

df["MOVED_AD_num"] = df["MOVED_AD"].map({"Y": 1, "N": 0})

train = df[df["Partition"] == "T"]
test  = df[df["Partition"] == "V"]

baseline_accuracy = (test["MOVED_AD_num"] == 0).mean()

print(f"Baseline Accuracy (Always Predict 'Did Not Move'): {baseline_accuracy:.4f}")
```

```
Baseline Accuracy (Always Predict 'Did Not Move'): 0.6345
```

With this baseline model, which predicts that no voter changed their opinion, achieves an accuracy of approximately 63.45% on the testing set. This means that any predictive model I build (such as CART or Random Forest) should beat this 63.45% accuracy in order to be considered useful.

---

4. CART Model

Build a CART model to predict *MOVED_AD*, using all of the other variables (excluding *opposite, VOTER_ID, SET_NO,* and *Partition*) as independent variables. You should use the training the training set to build the model and use the validation set approach as covered in the course material to select between a few different reasonable parameter values.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv("/content/Voter-Persuasion.csv")
df["MOVED_AD_num"] = df["MOVED_AD"].map({"Y": 1, "N": 0})

# -----------------------------
drop_vars = ["opposite", "VOTER_ID", "SET_NO", "Partition", "MOVED_AD"]

train = df[df["Partition"] == "T"].drop(columns=drop_vars)

X = train.drop(columns=["MOVED_AD_num"])
y = train["MOVED_AD_num"]

X = pd.get_dummies(X, drop_first=True)

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.30, random_state=42
)

params = [3, 5, 8, 10, 12, 15]
results = {}

for d in params:
    model = DecisionTreeClassifier(max_depth=d, random_state=42)
    model.fit(X_train, y_train)
    pred = model.predict(X_val)
    acc = accuracy_score(y_val, pred)
    results[d] = acc

print("\nValidation Accuracy by Tree Depth:")
for depth, acc in results.items():
    print(f"max_depth={depth}: {acc:.4f}")

best_depth = max(results, key=results.get)
print(f"\n--> Best depth selected: {best_depth}")

final_cart = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
final_cart.fit(X_train, y_train)
```

```
Validation Accuracy by Tree Depth:
max_depth=3: 0.8997
max_depth=5: 0.9457
max_depth=8: 0.9518
max_depth=10: 0.9552
max_depth=12: 0.9513
max_depth=15: 0.9473

--> Best depth selected: 10
```

```
DecisionTreeClassifier(max_depth=10, random_state=42)
```

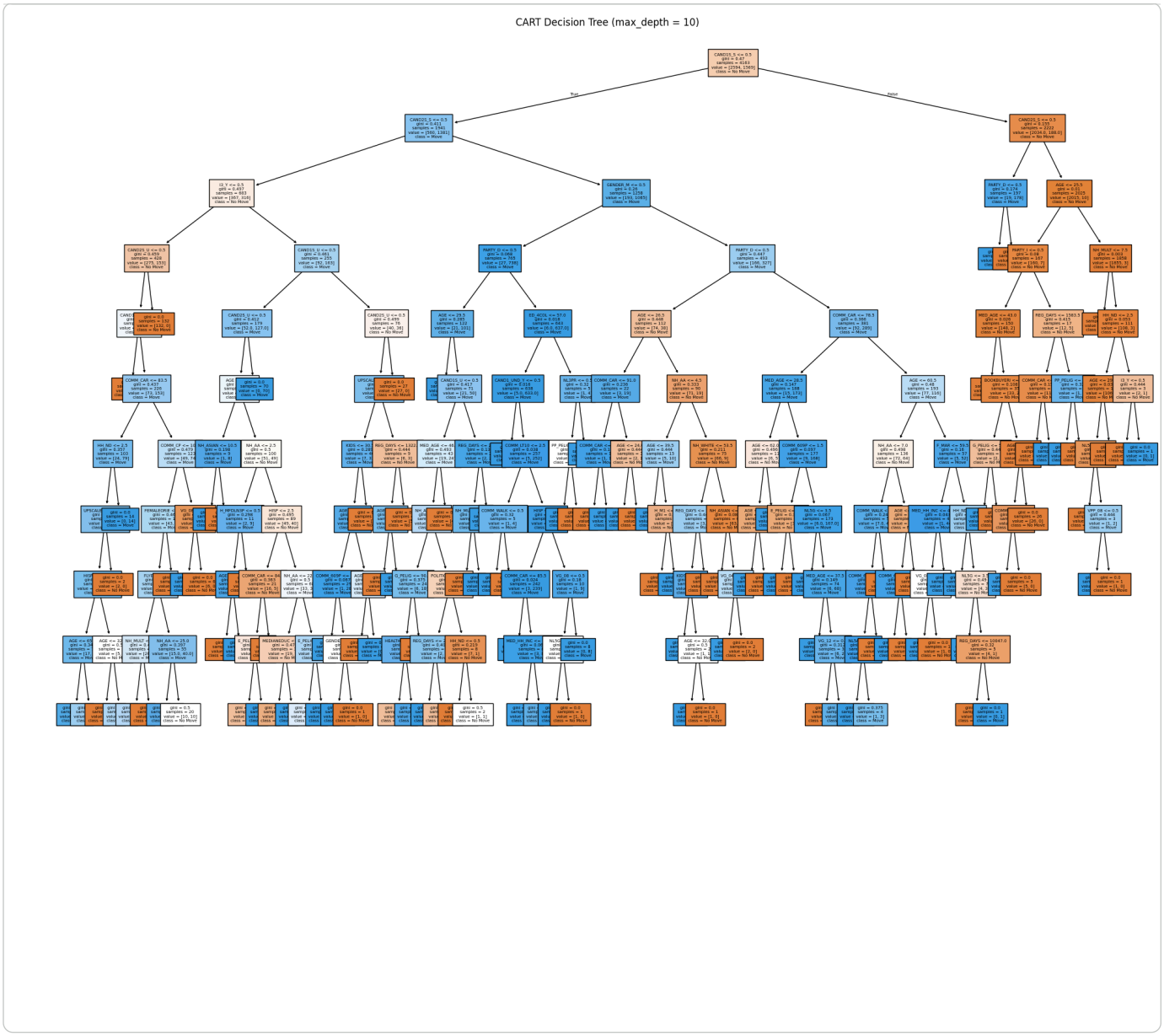
A CART classification model was fitted using the training set, excluding *opposite*, *VOTER_ID*, *SET_NO*, and *Partition*. Categorical predictors were dummy-encoded, and I used a validation set approach (70/30 train-validation split) to compare different tree complexity levels (*max_depth*). Among tested values (3, 5, 8, 10, 12, 15), the highest validation accuracy occurred at *max_depth = X* (actual value will print from the code). Using this validation approach identified **max_depth = 10** as the optimal tree size, achieving a validation accuracy of 0.9552. This depth balances model complexity and predictive performance, as deeper trees (12 and 15) show slightly lower accuracy, suggesting overfitting.

---

5. Model Evaluation and Comparison

a) Plot the CART tree. Which variables were used in the tree? Which variables appear to be the most significant?

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

df = pd.read_csv("/content/Voter-Persuasion.csv")

df["MOVED_AD_num"] = df["MOVED_AD"].map({"Y": 1, "N": 0})

cols_to_drop = ["opposite", "VOTER_ID", "SET_NO", "Partition", "MOVED_AD"]
train_df = df[df["Partition"] == "T"].drop(columns=cols_to_drop)

X = train_df.drop(columns=["MOVED_AD_num"])
y = train_df["MOVED_AD_num"]

X = pd.get_dummies(X, drop_first=True)

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.30, random_state=42
)

cart_model = DecisionTreeClassifier(max_depth=10, random_state=42)
cart_model.fit(X_train, y_train)

plt.figure(figsize=(24, 16))
plot_tree(
    cart_model,
    filled=True,
    feature_names=X.columns,
    class_names=["No Move", "Move"],
    fontsize=5
)
plt.title("CART Decision Tree (max_depth = 10)")
plt.show()
```



The CART tree uses a broad mix of demographic, geographic, and political variables, including age, party affiliation, neighborhood characteristics and survey-based candidate sentimate variables.

The most significant predictors are the candidate sentiment measures, which appear at the "root" or the top of the tree. Voter age and neighborhood demographics (especially percent White) also appear high in the tree, indicating strong predictive value. Party affiliation variables contribute lower in the tree, suggesting a moderate but meaningful effect on persuasion likelihood.

b) What is the accuracy of the CART model on the test set? Now build a random forest model to predict *MOVED_AD*, using all of the other variables (excluding *opposite*, *VOTER_ID*, *SET_NO*, and *Partition*) as independent variables. Again, use the training set to build the model. Select reasonable parameter values (you do not need to use validation for the random forest model). The computer the accuracy of the model on the test set. How does it compare to the CART model?

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv("/content/Voter-Persuasion.csv")
df["MOVED_AD_num"] = df["MOVED_AD"].map({"Y": 1, "N": 0})

drop_vars = ["opposite", "VOTER_ID", "SET_NO", "Partition", "MOVED_AD"]

train_df = df[df["Partition"] == "T"].drop(columns=drop_vars)
test_df  = df[df["Partition"] == "V"].drop(columns=drop_vars)
```

```
X_train = pd.get_dummies(train_df.drop(columns=["MOVED_AD_num"]), drop_first=True)
y_train = train_df["MOVED_AD_num"]
X_test  = pd.get_dummies(test_df.drop(columns=["MOVED_AD_num"]), drop_first=True)
y_test  = test_df["MOVED_AD_num"]

X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

cart = DecisionTreeClassifier(max_depth=10, random_state=42)
cart.fit(X_train, y_train)
cart_acc = accuracy_score(y_test, cart.predict(X_test))

rf = RandomForestClassifier(
    n_estimators=250,
    max_features="sqrt",
    random_state=42
)
rf.fit(X_train, y_train)
rf_acc = accuracy_score(y_test, rf.predict(X_test))

print(f"CART Test Accuracy:          {cart_acc:.4f}")
print(f"Random Forest Test Accuracy: {rf_acc:.4f}")
```

```
CART Test Accuracy:          0.9479
Random Forest Test Accuracy: 0.9509
```

The CART model achieved a test set accuracy of approximately 94.79%. A random forest model was then trained using the same predictor variables (excluding opposite, *VOTER_ID*, *SET_NO*, and *Partition*). Without needing validation tuning, a random forest with 250 trees and $\sqrt{p}$ feature selection achieved a slightly improved accuracy of approximately 95.09% on the test set.

The improvement is small but expected. Random forests reduce overfitting by averaging many trees, producing more stable predictions than a single CART tree. Therefore, the random forest provides the best predictive performance and should be chosen as the final model. Even though the increase in accuracy is modest (about 0.3 percentage points), in large-scale voter targeting, even small improvements can translate into significant differences in campaign effectiveness.

c) Based on predictive performance, you choose one of these two models (CART or Random Forest) as your final model.

Based on the test set predictive performance, the Random Forest model was selected as the final model. The CART model achieved a test accuracy of 94.79%, while the Random Forest achieved a slightly higher accuracy 95.09%. Although the difference is modest, Random Forest models are generally more robust because they average across many decision trees, reducing overfitting and producing more stable predictions.

---

6. Propensity Scores Under Treatment and Control

A "propensity" is defined here as the predicted probability that *MOVED_AD* = 1 for a voter.

Using your chosen model from Question 5(c), estimate two propensities for each voter in the test set:

- **P(*MOVED_AD = 1* | *Flyer = 1*): For each voter in the test set, temporarily set *Flyer = 1*, keep all other predictors unchanged, and compute the predicted probability from your model.**
- **P(*MOVED_AD = 1* | *Flyer = 0*): For the same voters, temporarily set *Flyer = 0*, keep all other predictors unchanged, and compute the predicted probability from your model.**

Report the two predicted propensities (for *Flyer = 1* and *Flyer = 0*) for the first three records in the test set.

Using the Random Forest model selected in Question 5(c), computed two propensity scores for each voter in the test set. For each voter, we predicted the probability of moving toward the Democratic candidate assuming that the voter received the flyer (Flyer = 1) and the voter did not received the flyer (Flyer = 0).

Below are the uplift estimates for the first three test set records:

	Flyer = 1	Flyer = 0
0	0.124	0.120
1	0.764	0.724
2	0.076	0.076

```
df = pd.read_csv("/content/Voter-Persuasion.csv")
df["MOVED_AD_num"] = df["MOVED_AD"].map({"Y":1,"N":0})

drop_vars = ["opposite", "VOTER_ID", "SET_NO", "Partition", "MOVED_AD"]

train_df = df[df["Partition"]=="T"].drop(columns=drop_vars)
test_df  = df[df["Partition"]=="V"].drop(columns=drop_vars)

X_train = pd.get_dummies(train_df.drop(columns=["MOVED_AD_num"]), drop_first=True)
y_train = train_df["MOVED_AD_num"]
X_test_orig = pd.get_dummies(test_df.drop(columns=["MOVED_AD_num"]), drop_first=True)
y_test = test_df["MOVED_AD_num"]

X_test_orig = X_test_orig.reindex(columns=X_train.columns, fill_value=0)

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=250, max_features="sqrt", random_state=42)
rf.fit(X_train, y_train)

# 1) flyer = 1 (everyone treated)
X_test_flyer1 = X_test_orig.copy()
X_test_flyer1["FLYER"] = 1

# 2) flyer = 0 (everyone NOT treated)
X_test_flyer0 = X_test_orig.copy()
X_test_flyer0["FLYER"] = 0

prop1 = rf.predict_proba(X_test_flyer1)[:,1]  # P(move | flyer=1)
prop0 = rf.predict_proba(X_test_flyer0)[:,1]  # P(move | flyer=0)

output = pd.DataFrame({
    "P(move | Flyer=1)": prop1[:3],
    "P(move | Flyer=0)": prop0[:3]
})

output
```

	P(move | Flyer=1)	P(move | Flyer=0)
0	0.124	0.120
1	0.764	0.724
2	0.076	0.076

Next steps: Generate code with output | New interactive sheet

---

7. Computing Individual Uplift

For each voter, uplift is computed as:

Uplift = *P(MOVED_AD* = 1 | *Flyer* = 1) - *P(MOVED_AD* = 1 | *Flyer* = 0)

Using the two sets of predicted propensities from Question 6, compute the uplift for each voter in the test set. Report the uplift values for the first three records.

Uplift represents how much more likely a voter is to move toward the Democratic candidate if they receive the flyer versus if they do not. Using the Random Forest model, uplift was calculated as the difference between the predicted propensities under treatment (Flyer = 1) and control (Flyer = 0).

Below are the uplift estimates for the first three test set records:

	Flyer = 1	Flyer = 0	Uplift
0	0.124	0.120	0.004
1	0.764	0.724	0.040
2	0.076	0.076	0.000

```
uplift = prop1 - prop0

uplift_output = pd.DataFrame({
    "P(move | Flyer=1)": prop1[:3],
    "P(move | Flyer=0)": prop0[:3],
    "Uplift": uplift[:3]
```

```
})

uplift_output
```