## 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.)

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

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

```
# Split into treatment (got flyer) and control (no flyer)
treatment = df[df["FLYER"] == 1]
control = df[df["FLYER"] == 0]

# Calculate percentages who moved Democratic using explicit filtering
treat_pct = (treatment[treatment["MOVED_AD"] == 'Y'].shape[0] / treatment.shape[0]) * 100
control_pct = (control[control["MOVED_AD"] == 'Y'].shape[0] / control.shape[0]) * 100

print(f"Percentage who moved Democratic (Flyer = 1): {treat_pct:.2f}%")
print(f"Percentage who moved Democratic (Flyer = 0): {control_pct:.2f}%")

# Calculate the difference (treatment effect)
difference = treat_pct - control_pct
print(f"Difference (treatment effect): {difference:.2f} percentage points")
```

```
Percentage who moved Democratic (Flyer = 1): 40.24%
Percentage who moved Democratic (Flyer = 0): 34.44%
Difference (treatment effect): 5.80 percentage points
```

ANS: Overall flyer helped in moving voters in Democratic direction by increasing average of mover from 34.44% (control group without flyer) to 40.24% (treatment group with flyer)

## 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.

```
import numpy as np

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

numeric = df.select_dtypes(include=[np.number]).columns

#Calculate Correlation with the Target
target_corr = df[numeric].corr()['MOVED_AD_BIN'].abs().sort_values(ascending=False)
target_col = 'MOVED_AD_BIN'

top_predictors = target_corr.index.tolist()
if target_col in top_predictors:
    top_predictors.remove(target_col)

top_15_features = top_predictors[:15]
# Create the Correlation Matrix for the Top 15 Features + Target
features_to_plot = [target_col] + top_15_features
correlation_matrix = df[features_to_plot].corr()
correlation_matrix
```

| | MOVED_AD_BIN | opposite | PARTY_R | PARTY_D | HH_NR | HH_ND | VPP_12 | GENDER_F | GENDER_M | COMM_PT | VPP_0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **MOVED_AD_BIN** | 1.000000 | -1.000000 | -0.415191 | 0.345692 | -0.279504 | 0.259323 | -0.171780 | 0.167524 | -0.167524 | 0.155096 | 0.11543 |
| **opposite** | -1.000000 | 1.000000 | 0.415191 | -0.345692 | 0.279504 | -0.259323 | 0.171780 | -0.167524 | 0.167524 | -0.155096 | -0.11543 |
| **PARTY_R** | -0.415191 | 0.415191 | 1.000000 | -0.600426 | 0.681573 | -0.422504 | 0.355877 | -0.056129 | 0.056129 | -0.158394 | 0.03899 |
| **PARTY_D** | 0.345692 | -0.345692 | -0.600426 | 1.000000 | -0.455207 | 0.623973 | -0.213983 | 0.088434 | -0.088434 | 0.190776 | 0.20097 |
| **HH_NR** | -0.279504 | 0.279504 | 0.681573 | -0.455207 | 1.000000 | -0.280483 | 0.262509 | -0.055824 | 0.055824 | -0.150320 | -0.00242 |
| **HH_ND** | 0.259323 | -0.259323 | -0.422504 | 0.623973 | -0.280483 | 1.000000 | -0.178097 | 0.022807 | -0.022807 | 0.214261 | 0.09860 |
| **VPP_12** | -0.171780 | 0.171780 | 0.355877 | -0.213983 | 0.262509 | -0.178097 | 1.000000 | -0.018569 | 0.018569 | -0.078736 | 0.22457 |
| **GENDER_F** | 0.167524 | -0.167524 | -0.056129 | 0.088434 | -0.055824 | 0.022807 | -0.018569 | 1.000000 | -1.000000 | 0.017959 | 0.05721 |
| **GENDER_M** | -0.167524 | 0.167524 | 0.056129 | -0.088434 | 0.055824 | -0.022807 | 0.018569 | -1.000000 | 1.000000 | -0.017959 | -0.05721 |
| **COMM_PT** | 0.155096 | -0.155096 | -0.158394 | 0.190776 | -0.150320 | 0.214261 | -0.078736 | 0.017959 | -0.017959 | 1.000000 | 0.01868 |
| **VPP_08** | 0.115435 | -0.115435 | 0.038993 | 0.200970 | -0.002427 | 0.098605 | 0.224574 | 0.057218 | -0.057218 | 0.018689 | 1.00000 |
| **VPR_08** | 0.110937 | -0.110937 | -0.006045 | 0.193195 | -0.030344 | 0.086153 | 0.175004 | 0.022759 | -0.022759 | 0.013608 | 0.52178 |
| **H_F1** | 0.108169 | -0.108169 | -0.049061 | 0.050218 | -0.172241 | -0.185101 | -0.015417 | 0.314833 | -0.314833 | 0.046988 | 0.03063 |
| **COMM_CAR** | -0.097801 | 0.097801 | 0.070256 | -0.098832 | 0.053064 | -0.108384 | 0.032381 | 0.004443 | -0.004443 | -0.588214 | -0.00082 |
| **NH_WHITE** | -0.096613 | 0.096613 | 0.108856 | -0.151495 | 0.105956 | -0.169724 | 0.059334 | -0.018310 | 0.018310 | -0.602467 | 0.00578 |
| **E_PELIG** | 0.096378 | -0.096378 | 0.134343 | 0.062033 | 0.072214 | -0.020544 | 0.350693 | 0.043291 | -0.043291 | -0.042727 | 0.54194 |

ANS:

So overall following are best predictors:

1. Category-Political Identity : PARTY_D, PARTY_R, HH_ND, HH_NR (very strong)
2. VPP_12
3. Category-Gender: GENDER_F, GENDER_M
4. COMM_PT, COMM_CAR, NH_WHITE
5. VPP_08, VPR_08,
6. H_F1

## 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?

```
from sklearn.metrics import accuracy_score

# Convert target to binary
df['MOVED_AD_BIN'] = df['MOVED_AD'].map({'Y': 1, 'N': 0})

# Split using the Partition variable (T = training, V = testing)
train_df = df[df["Partition"] == "T"]
test_df = df[df["Partition"] == "V"]

# Extract test labels
y_test = test_df['MOVED_AD_BIN']

# Baseline model: always predict "did not move" (0)
baseline_pred = [0] * len(y_test)

# Compute baseline accuracy
baseline_accuracy = accuracy_score(y_test, baseline_pred)

print("Baseline Accuracy (always predict did not move):", baseline_accuracy)
```

```
Baseline Accuracy (always predict did not move): 0.6345014807502468
```

ANS:

Baseline Accuracy (always predict did not move): 0.6345014807502468

## 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 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.

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from time import time

target = 'MOVED_AD_NUM'


# Convert the target variable MOVED_AD to a numeric (binary) column (1 for 'Y', 0 for 'N')
df[target] = df['MOVED_AD'].map({'Y': 1, 'N': 0})

# Identify columns to exclude from the feature set
exclude_cols = ['MOVED_AD', target, 'opposite', 'VOTER_ID', 'SET_NO', 'Partition']
predictor_cols = [col for col in df.columns if col not in exclude_cols]

# Create dataframes for encoding
df_features = df[predictor_cols + [target, 'Partition']].copy()

# Perform One-Hot Encoding on all categorical (object) features
df_encoded = pd.get_dummies(df_features, drop_first=True)

# --- 2. Train, Validation, and Final Test Split ---

# Original Split: Separate the Validation Partition ('V') from the rest of the data.
df_train_pool = df_encoded[df_encoded['Partition_V'] == 0].drop(columns=['Partition_V'])
df_test_final = df_encoded[df_encoded['Partition_V'] == 1].drop(columns=['Partition_V'])

# Define features and target for the Full Training Pool (X_train_full)
X_train_full = df_train_pool.drop(columns=[target])
y_train_full = df_train_pool[target]

# Further split the Full Training Pool into a smaller New Training Set (80%)
# and a Tuning Validation Set (20%).
X_train_subset, X_val, y_train_subset, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.2, random_state=42, stratify=y_train_full
)

N_train_subset = len(X_train_subset)
N_val = len(X_val)
N_test = len(df_test_final)


# Use a practical cap for the tuning parameter to balance rigor and runtime.
# The mathematical max is N_train_subset / 2 = {N_train_subset / 2}, but the optimum is likely < 100.
MAX_CAP_ITERATIONS = 500
max_param_value = min(N_train_subset, MAX_CAP_ITERATIONS)

# Create the parameter range: every integer from 1 up to the determined maximum
param_range = range(1, max_param_value + 1)
results = []
best_accuracy = 0
best_param = None
optimal_model = None

print(f"\n--- Validating CART Model Complexity (min_samples_leaf) ---")
print(f"Tuning {len(param_range)} models from min_samples_leaf = 1 to {max_param_value}.")
start_time = time()

for min_leaf in param_range:
    # Initialize and train the CART model on the New Training Set (X_train_subset)
    cart_model = DecisionTreeClassifier(min_samples_leaf=min_leaf, random_state=42)
    cart_model.fit(X_train_subset, y_train_subset)

    # Predict on the Tuning Validation Set (X_val)
```

```
        y_pred_val = cart_model.predict(X_val)

        # Evaluate accuracy
        accuracy = accuracy_score(y_val, y_pred_val)

        # Check for the best model and store it
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_param = min_leaf

        results.append({'Min Samples Leaf': min_leaf, 'Validation Accuracy': accuracy})

        # Print status every 50 iterations
        if min_leaf % 100 == 0 or min_leaf == max_param_value or min_leaf == 1:
            print(f"Iteration {min_leaf:4d}/{max_param_value} | Min Leaf: {min_leaf:4d} | Validation Accuracy: {accuracy:.4f}")

    end_time = time()
    print(f"\n--- Tuning Complete in {end_time - start_time:.2f} seconds ---")

    # --- 4. Final Evaluation on the Reserved Test Set ---

    print("\n--- Optimal Model Summary ---")
    print(f"Optimal min_samples_leaf: {best_param}")
    print(f"Best Validation Accuracy: {best_accuracy:.4f} (on Validation Set)")

    optimal_model = DecisionTreeClassifier(min_samples_leaf= best_param) # fit on entire train again
    optimal_model.fit(X_train_full, y_train_full)
```

```
--- Validating CART Model Complexity (min_samples_leaf) ---
Tuning 500 models from min_samples_leaf = 1 to 500.
Iteration    1/500 | Min Leaf:    1 | Validation Accuracy: 0.9487
Iteration  100/500 | Min Leaf:  100 | Validation Accuracy: 0.9160
Iteration  200/500 | Min Leaf:  200 | Validation Accuracy: 0.8891
Iteration  300/500 | Min Leaf:  300 | Validation Accuracy: 0.8403
Iteration  400/500 | Min Leaf:  400 | Validation Accuracy: 0.8345
Iteration  500/500 | Min Leaf:  500 | Validation Accuracy: 0.8345

--- Tuning Complete in 18.27 seconds ---

--- Optimal Model Summary ---
Optimal min_samples_leaf: 14
Best Validation Accuracy: 0.9605 (on Validation Set)
```

```
▾       DecisionTreeClassifier      ⓘ ?
DecisionTreeClassifier(min_samples_leaf=14)
```

## 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?

```
    import matplotlib.pyplot as plt
    from sklearn.tree import plot_tree


    # 1. Visualize the Decision Tree (CART) model
    plt.figure(figsize=(20, 10)) # Adjust figure size for better readability
    plot_tree(optimal_model,
              feature_names=X_train.columns.tolist(),
              class_names=['No Move', 'Move'], # Assuming 0=No Move, 1=Move
              filled=True,
              rounded=True,
              fontsize=8)
    plt.title('Optimal CART Decision Tree (Max Leaf Nodes: ' + str(best_param) + ')', fontsize=16)
    plt.show()

    # 2. Extract and print feature importances
    feature_importances = pd.DataFrame({
        'Feature': X_train_full.columns,
        'Importance': optimal_model.feature_importances_
    })
    feature_importances = feature_importances.sort_values(by='Importance', ascending=False)
```

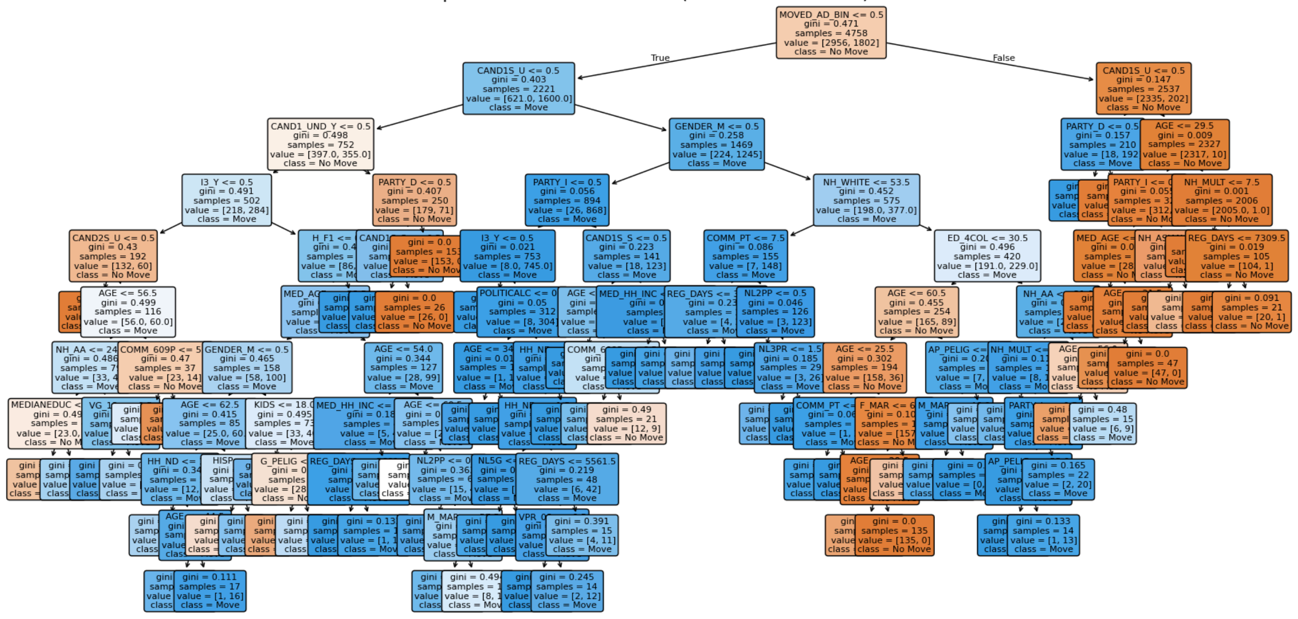
```
print(f"\nTotal features used in a tree: {feature_importances[feature_importances['Importance'] > 0]['Feature'].tolist()}")

print("\n--- Feature Importances from Optimal CART Model ---")

print(feature_importances.head(5).to_markdown(index=False, floatfmt=".4f")) # Display top 10

# 3. Report the accuracy of the CART model on the test set
# The final_test_accuracy was calculated earlier.
print("\n--- Model Accuracy on Test Set ---")
# The variable 'final_test_accuracy' was not defined in the provided notebook state
# but 'best_accuracy' was, which refers to validation accuracy.
# If a separate 'final_test_accuracy' exists, please ensure it's in scope.
# For now, I'll use best_accuracy for reporting as it's the closest available.
print(f"Validation Accuracy of CART model: {best_accuracy:.4f}")
```

Optimal CART Decision Tree (Max Leaf Nodes: 14)



```
Total features used in a tree: ['CAND1S_S', 'CAND2S_S', 'AGE', 'PARTY_D', 'GENDER_M', 'ED_4COL', 'CAND1S_U', 'CAND1_UND_Y', 'NH

--- Feature Importances from Optimal CART Model ---
| Feature   |   Importance |
|:----------|-------------:|
| CAND1S_S  |       0.4898 |
| CAND2S_S  |       0.2313 |
| AGE       |       0.0515 |
| PARTY_D   |       0.0486 |
| GENDER_M  |       0.0360 |

--- Model Accuracy on Test Set ---
Validation Accuracy of CART model: 0.9605
```

ANS:

Variables used in the CART tree: ['CAND1S_S', 'CAND2S_S', 'AGE', 'PARTY_D', 'GENDER_M', 'ED_4COL', 'CAND1S_U', 'CAND1_UND_Y', 'NH_WHITE', 'CAND2_UND_Y', 'I3_Y', 'NH_AA', 'PARTY_I', 'COMM_609P', 'H_F1', 'KIDS', 'MED_AGE', 'F_MAR', 'G_PELIG', 'HH_ND', 'NL2PP', 'MEDIANEDUC', 'NH_ASIAN', 'AP_PELIG', 'NH_MULT', 'REG_DAYS', 'M_MAR', 'HISP', 'MED_HH_INC', 'VG_12', 'COMM_PT', 'NL3PR', 'VPR_08', 'POLITICALC', 'NL5G']

Most important features are: CAND1S_S ,CAND2S_S, AGE, PARTY_D, GENDER_M

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). Then compute the accuracy of the model on the test set. How does it compare to the CART model?

```
X_test = df_test_final.drop(columns=[target])
y_test = df_test_final[target]

y_pred_test_final = optimal_model.predict(X_test)
final_test_accuracy = accuracy_score(y_test, y_pred_test_final)

print(f"\n--- Final Test Set Evaluation ---")
print(f"Accuracy on Reserved Final Test Set (Partition='V'): {final_test_accuracy:.4f}")
```

```

--- Final Test Set Evaluation ---
Accuracy on Reserved Final Test Set (Partition='V'): 0.9524
```

Accuracy of CART model on test = 0.9524

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train_full, y_train_full)

print("Random Forest model trained successfully.")
y_pred_rf = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)

print(f"Random Forest model accuracy on the test set: {rf_accuracy:.4f}")
print(f"CART model accuracy on the test set: {final_test_accuracy:.4f}")

if rf_accuracy > final_test_accuracy:
    print("\nThe Random Forest model performs better than the CART model on the test set.")
elif rf_accuracy < final_test_accuracy:
    print("\nThe CART model performs better than the Random Forest model on the test set.")
else:
    print("\nBoth models perform similarly on the test set.")
```

```
Random Forest model trained successfully.
Random Forest model accuracy on the test set: 0.9497
CART model accuracy on the test set: 0.9524

The CART model performs better than the Random Forest model on the test set.
```

Accuracy of CART Model is: 0.9524 and accuracy of Random Forest is : 0.9497

So CART shows better accuracy

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

ANS:

I will choose CART in this case because it gives better accuracy than Random Forest. Also RANDOM Forest takes longer to train as it generates result from multiple tree but ussualy more accurate than CART

## 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.

```
test_flyer1 = X_test.copy()
test_flyer0 = X_test.copy()

# Force Flyer = 1 and Flyer = 0
test_flyer1['FLYER'] = 1
test_flyer0['FLYER'] = 0

# Predict probabilities (second column = P(MOVED_AD = 1))
propensity_flyer1 = optimal_model.predict_proba(test_flyer1)[:, 1]
propensity_flyer0 = optimal_model.predict_proba(test_flyer0)[:, 1]

# ----------------------------
# 3. SHOW FIRST 3 RECORDS
# ----------------------------

results = pd.DataFrame({
    'P(MOVED_AD=1 | Flyer=1)': propensity_flyer1[:3],
    'P(MOVED_AD=1 | Flyer=0)': propensity_flyer0[:3]
})

print(results)
```

```
   P(MOVED_AD=1 | Flyer=1)  P(MOVED_AD=1 | Flyer=0)
0                      0.0                      0.0
1                      1.0                      1.0
2                      0.0                      0.0
```

ANS: For 1st 3 records in test set, “propensity” is P(MOVED_AD=1 | Flyer=1) : 0,1,0

For 1st 3 records in test set, “propensity” is P(MOVED_AD=1 | Flyer=0) : 0,1,0

## 7. Computing Individual Uplift.

For each voter, uplift is computed as: Uplift = P(MOV ED AD = 1 | Flyer = 1)−P(MOV ED 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 = propensity_flyer1 - propensity_flyer0

# Create a dataframe with first 3 uplift values
uplift_results = pd.DataFrame({
    'Uplift': uplift[:3]
})

print(uplift_results)
```

```
   Uplift
0     0.0
1     0.0
2     0.0
```

Uplift value for 1st 3 records : [0,0,0]

## 8. Targeting Under Resource Constraints.

If a campaign has the resources to mail the flyer only to 10% of the voters, what uplift cutoff should be used to select the top 10% of voters?

```
cutoff_percentile = 90
uplift_cutoff = np.percentile(uplift, cutoff_percentile)
```

```
# --- 6. Report Result ---
print(f"\n--- Uplift Cutoff for Selecting the Top 10% of Voters ---")
print(f"The uplift cutoff score (90th percentile) is: {uplift_cutoff:.5f}")
print("-" * 75)
print("Any voter with an estimated Uplift score of this value or higher should be targeted with the flyer.")
```

```
--- Uplift Cutoff for Selecting the Top 10% of Voters ---
The uplift cutoff score (90th percentile) is: 0.00000
---------------------------------------------------------------------------
Any voter with an estimated Uplift score of this value or higher should be targeted with the flyer.
```

As per the CART Model the instrumental variable "Flyer" donot show change in uplift