In [143…
```
import pandas as pd

df=pd.read_csv("tnelectionsformlmodels.csv")
```

In [144…
```
df.columns
```

Out[144…
```
Index(['Unnamed: 0', 'Constituency_No', 'Position', 'Candidate', 'Sex',
       'Party', 'Votes', 'Age', 'Valid_Votes', 'Electors', 'Constituency_Name',
       'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
       'Turnout_Percentage', 'Vote_Share_Percentage', 'Deposit_Lost', 'Margin',
       'Margin_Percentage', 'ENOP', 'pid', 'Party_ID', 'Contested',
       'Last_Party', 'Last_Party_ID', 'Last_Constituency_Name',
       'Same_Constituency', 'Same_Party', 'No_Terms', 'Turncoat', 'Incumbent',
       'Recontest', 'Education_Qualification', 'Main_Profession',
       'Main_Profession_Desc', 'Second_Profession', 'Second_Profession_Desc',
       'Result', 'Non_Voters', 'Non_Voters_Percentage',
       'Non_Voters_Percentage_1', 'Invalid_Votes_percentage', 'No_Terms_lost',
       'Alliance', 'Current_Status'],
      dtype='object')
```

In [145…
```
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Result', 'Alliance','No_Terms_lost']]


data = data[data['Party'] != 'NOTA']
data
```

Out[145…

	Sex	Party	Age	Electors	Constituency_Type	District_Name	Sub_Region	No_of_Candidates	ENOP	Contested	Turncoat	I
0	M	DMK	60.0	284412	GEN	TIRUVALLUR	CHENNAI CITY REGION	13	2.27	1.0	False	
1	M	PMK	50.0	284412	GEN	TIRUVALLUR	CHENNAI CITY REGION	13	2.27	1.0	False	
2	F	NTK	31.0	284412	GEN	TIRUVALLUR	CHENNAI CITY REGION	13	2.27	1.0	False	
3	M	DMDK	45.0	284412	GEN	TIRUVALLUR	CHENNAI CITY REGION	13	2.27	1.0	False	
5	M	BSP	28.0	284412	GEN	TIRUVALLUR	CHENNAI CITY REGION	13	2.27	1.0	False	
...	...	...	...	...	...	...	...	...	...	...	...	
4227	M	IND	42.0	257959	GEN	KANNIYAKUMARI	SOUTHERN REGION	15	2.27	1.0	False	
4228	M	IND	61.0	257959	GEN	KANNIYAKUMARI	SOUTHERN REGION	15	2.27	1.0	False	
4229	M	IND	45.0	257959	GEN	KANNIYAKUMARI	SOUTHERN REGION	15	2.27	1.0	False	
4230	M	National Democratic Party of South India	34.0	257959	GEN	KANNIYAKUMARI	SOUTHERN REGION	15	2.27	1.0	False	
4231	M	Tamilnadu Mahatma Gandhi	72.0	257959	GEN	KANNIYAKUMARI	SOUTHERN REGION	15	2.27	1.0	False	

Sex	Party	Age	Electors	Constituency_Type	District_Name	Sub_Region	No_of_Candidates	ENOP	Contested	Turncoat	In
	Makkal Katchi										

3998 rows × 18 columns

In [146…

```
####XGB MODEL WITH FEATURE SELECTION####

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn import metrics
from xgboost import XGBClassifier, plot_importance
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths
data = data.dropna()

# Verify the lengths before proceeding
print(f"Length of data: {len(data)}")

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
```

```
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Verify the lengths after encoding
print(f"Length of X_encoded: {len(X_encoded)}")
print(f"Length of y: {len(y)}")

# Check the result
print(X_encoded.head())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y2 = est.predict_proba(X_test)
    y1 = est.predict(X_test)
    print("Accuracy: ", metrics.accuracy_score(y_test, y1))
    print("Area under the ROC curve: ", metrics.roc_auc_score(y_test, y2[:, 1]))
    print("F-metric: ", metrics.f1_score(y_test, y1))
    print(" ")
    print("Classification report:")
    print(metrics.classification_report(y_test, y1))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y1, y2[:, 1]

# Train the model and make predictions
xgb0, y_pred_b, y_pred2_b = classify(XGBClassifier(), X_train, y_train, X_test, y_test)

# Display the feature importances
print("Feature Importances:", xgb0.feature_importances_)
plot_importance(xgb0)
plt.show()

conf_matrix = confusion_matrix(y_test, y_pred_b)
print("Confusion Matrix:")
```

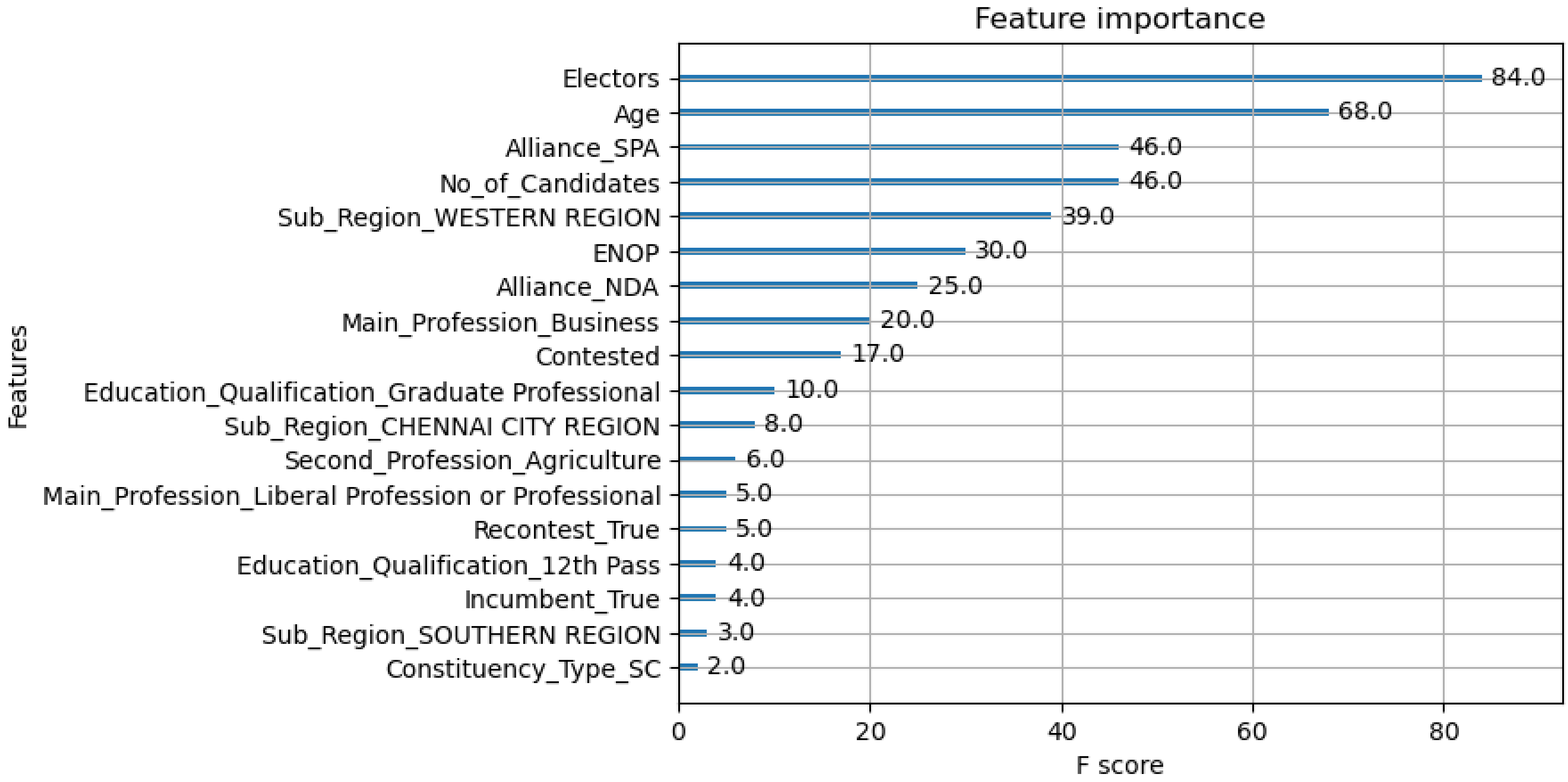
```
print(conf_matrix)

# Function to format feature importances
def feat_importance(model):
    feature_importance = model.feature_importances_
    feature_names = X_encoded.columns
    importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance})
    return importance_df.sort_values(by='Importance', ascending=False)

# Get and display feature importances in a dataframe
feat1 = feat_importance(xgb0)
print(feat1)
```

```
C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2493927970.py:19: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
```

```
Length of data: 213
Length of X_encoded: 213
Length of y: 213
      Age  Electors  No_of_Candidates  ENOP  Contested  Sex_M  Sex_O  \
20   42.0    268994                11  2.70        1.0   True  False
88   51.0    454179                21  3.03        2.0   True  False
90   35.0    454179                21  3.03        1.0   True  False
132  62.0    456079                21  2.78        2.0   True  False
133  62.0    456079                21  2.78        3.0   True  False

     Party_AMMK  Party_Akhila India Jananayaka Makkal Katchi (Dr. Isaac)  \
20        False                                              False
88        False                                              False
90        False                                              False
132       False                                              False
133       False                                              False

     Party_Anaithindia Samudaya Munnetra Kazhagam  ...  \
20                                          False  ...
88                                          False  ...
90                                          False  ...
132                                         False  ...
133                                         False  ...

     Second_Profession_Small Business or Self-employed  \
20                                               False
88                                               False
90                                                True
132                                              False
133                                              False

     Second_Profession_Social Work  Second_Profession_Student  \
20                            True                      False
88                           False                      False
90                           False                      False
132                          False                      False
133                          False                      False

     Second_Profession_Traditional Occupation  Alliance_IND  Alliance_NDA  \
20                                      False          True         False
88                                      False         False          True
```

```
90                                     False         False         False
132                                    False         False         False
133                                    False         False          True

     Alliance_NTK  Alliance_PF  Alliance_PFA  Alliance_SPA
20          False        False         False         False
88          False        False         False         False
90           True        False         False         False
132         False        False         False          True
133         False        False         False         False

[5 rows x 114 columns]
Accuracy:  0.9767441860465116
Area under the ROC curve:  0.9811827956989247
F-metric:  0.96

Classification report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        31
           1       0.92      1.00      0.96        12

    accuracy                           0.98        43
   macro avg       0.96      0.98      0.97        43
weighted avg       0.98      0.98      0.98        43


Evaluation by cross-validation:
[0.91176471 0.85294118 0.97058824 0.73529412 0.85294118]
Feature Importances: [0.03026949 0.02091528 0.02733238 0.02228206 0.05221242 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.00833624 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
 0.         0.03042844 0.04163916 0.07671909 0.         0.04734556
 0.06085131 0.02819965 0.         0.         0.         0.
 0.05591263 0.         0.         0.         0.         0.02904444
 0.         0.         0.         0.01676937 0.         0.
 0.         0.         0.01516336 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.1696109  0.         0.         0.         0.26696828]
```



```
Confusion Matrix:
[[30  1]
 [ 0 12]]
                                       Feature  Importance
113                               Alliance_SPA    0.266968
109                               Alliance_NDA    0.169611
75                    Sub_Region_WESTERN REGION    0.076719
78                              Recontest_True    0.060851
84   Education_Qualification_Graduate Professional    0.055913
..                                          ...         ...
42                        District_Name_CHENNAI    0.000000
41                        Constituency_Type_ST    0.000000
39                  Party_Vidial Valarchi Perani    0.000000
38                                   Party_VCK    0.000000
57                    District_Name_PUDUKKOTTAI    0.000000

[114 rows x 2 columns]
```

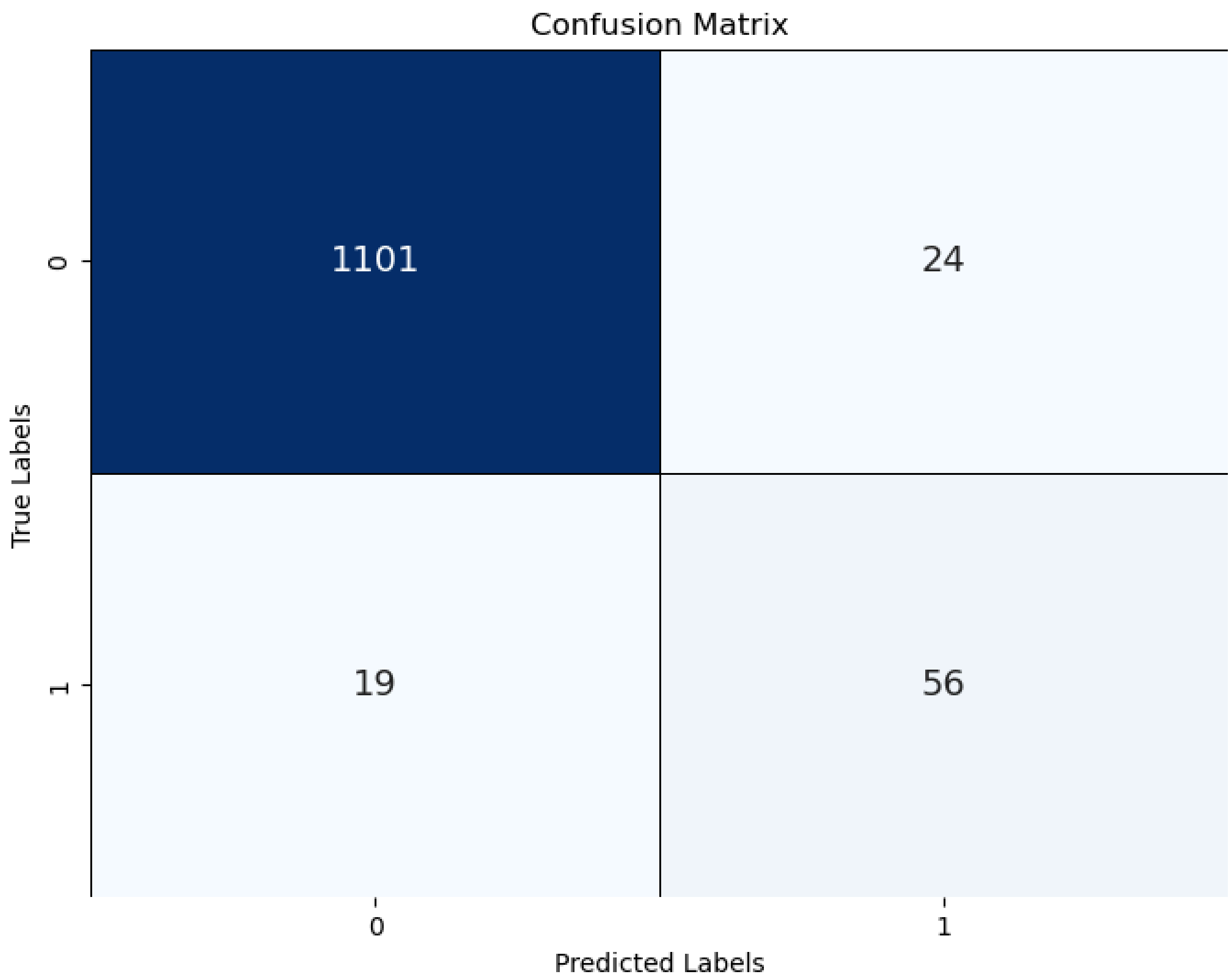
In [147…

```
#####XGB CLASSIFIER ###############


import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score, f1_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age','Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths
```

```
# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
    print("Area under the ROC curve: ", roc_auc_score(y_test, est.predict_proba(X_test)[:, 1]))
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred, est.predict_proba(X_test)[:, 1]

# Train the model and make predictions
xgb_balanced, y_pred_b_balanced, y_pred2_b_balanced = classify(XGBClassifier(scale_pos_weight=1), X_train, y_train, X_test, y_

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_b_balanced)
print("Confusion Matrix:")
print(conf_matrix)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
```

```
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy:  0.9641666666666666
Area under the ROC curve:  0.9820207407407407
F-metric:  0.7225806451612903

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1125
           1       0.70      0.75      0.72        75

    accuracy                           0.96      1200
   macro avg       0.84      0.86      0.85      1200
weighted avg       0.97      0.96      0.96      1200


Evaluation by cross-validation:
[0.96785714 0.96785714 0.96071429 0.96243292 0.96779964]
Confusion Matrix:
[[1101   24]
 [  19   56]]
```



In [ ]:

```

```

In [148…

```
#######K NEAREST NEIGHBOURS ######

import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelBinarizer
```

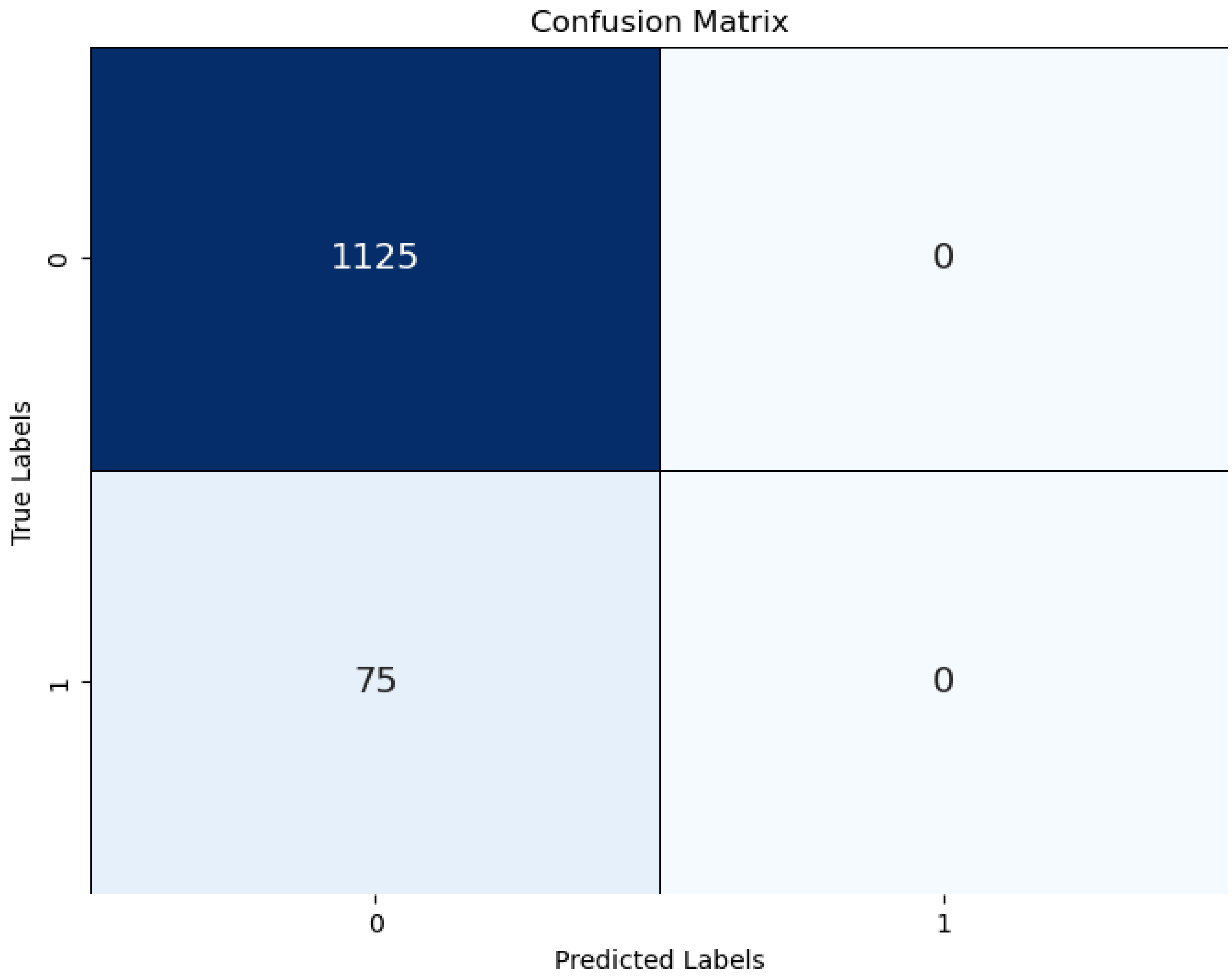
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure data is in the correct format
X_encoded = X_encoded.values
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
    try:
        y_pred_proba = est.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_pred_proba)
    except AttributeError:
        roc_auc = "N/A (predict_proba not available for this classifier)"

    print("Area under the ROC curve: ", roc_auc)
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred

# Train the model and make predictions with KNeighborsClassifier
knc, y_p = classify(KNeighborsClassifier(), X_train, y_train, X_test, y_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_p)
print("Confusion Matrix:")
print(conf_matrix)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy:  0.9375
Area under the ROC curve:  0.40309925925925927
F-metric:  0.0

Classification report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1125
           1       0.00      0.00      0.00        75

    accuracy                           0.94      1200
   macro avg       0.47      0.50      0.48      1200
weighted avg       0.88      0.94      0.91      1200


Evaluation by cross-validation:
```

```
C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.

C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.

C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
```

```
[0.94285714 0.94285714 0.94285714 0.94454383 0.94275492]
Confusion Matrix:
[[1125    0]
 [  75    0]]
```



In [ ]:

In [149…

```
####LOGISTIC REGRESSION####
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, roc_auc_score
```

```
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure data is in the correct format
X_encoded = X_encoded.values
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
    try:
        y_pred_proba = est.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_pred_proba)
    except AttributeError:
        roc_auc = "N/A (predict_proba not available for this classifier)"

    print("Area under the ROC curve: ", roc_auc)
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred

# Train the model and make predictions with LogisticRegression
logit, y_p = classify(LogisticRegression(), X_train, y_train, X_test, y_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_p)
print("Confusion Matrix:")
print(conf_matrix)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

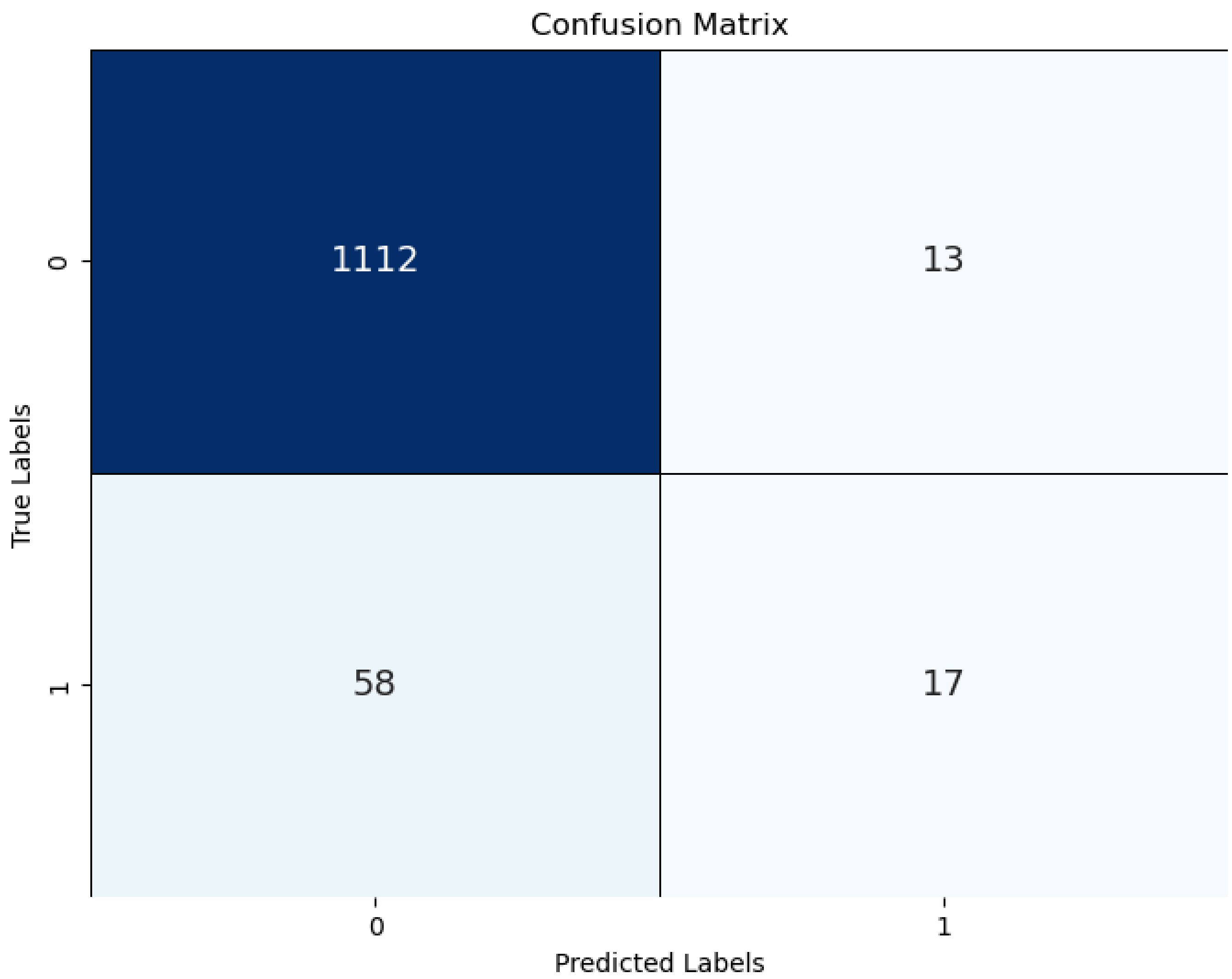
```
C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
Accuracy:  0.9408333333333333
Area under the ROC curve:  0.9161362962962963
F-metric:  0.3238095238095238

Classification report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1125
           1       0.57      0.23      0.32        75

    accuracy                           0.94      1200
   macro avg       0.76      0.61      0.65      1200
weighted avg       0.93      0.94      0.93      1200


Evaluation by cross-validation:
```

```
C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
[0.95535714 0.95       0.94285714 0.94275492 0.94454383]
Confusion Matrix:
[[1112   13]
 [  58   17]]
```



In [ ]:

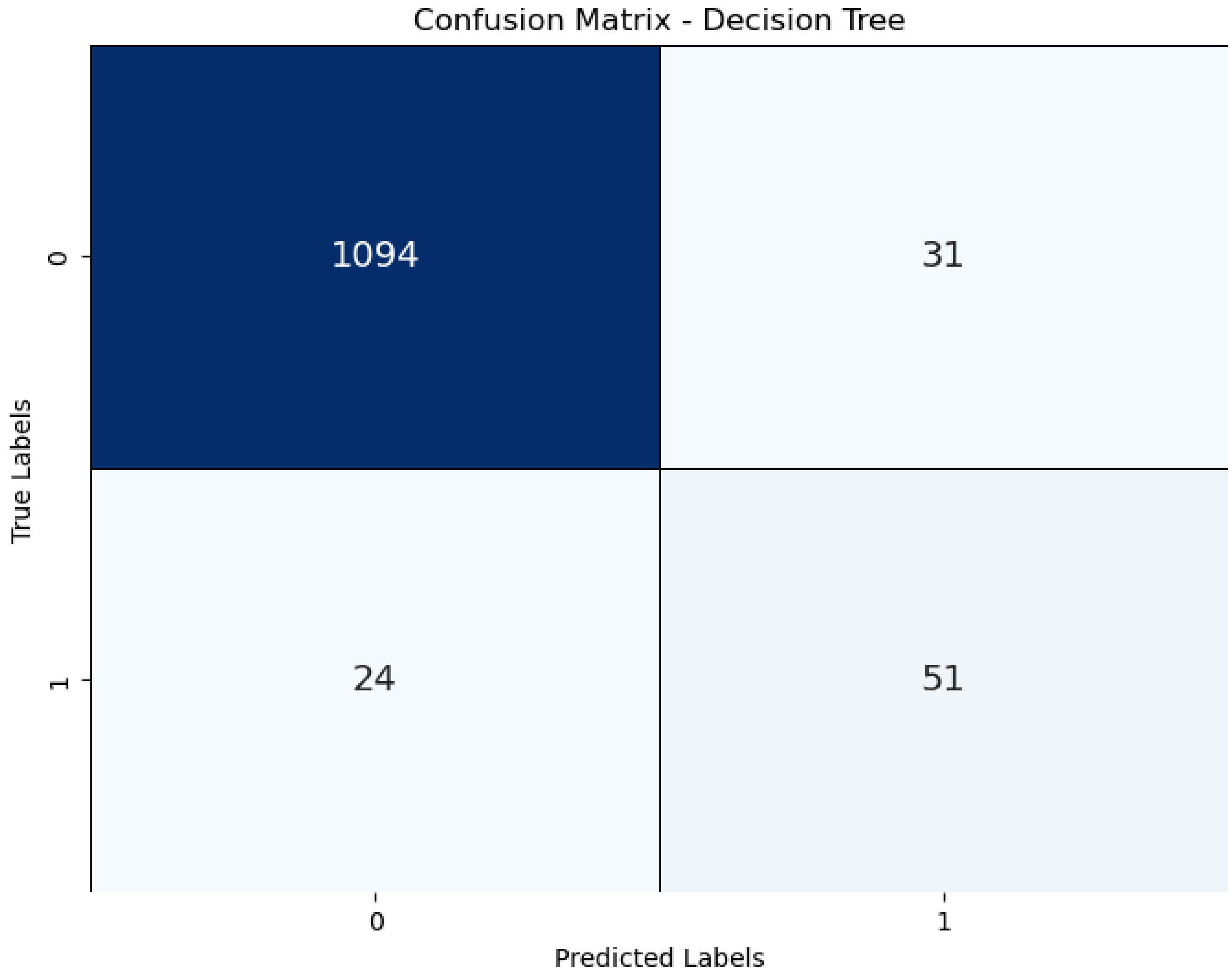
In [150…

```
###DECISION TREE CLASSIFIER#####

import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelBinarizer
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure data is in the correct format
X_encoded = X_encoded.values
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
    try:
        y_pred_proba = est.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_pred_proba)
    except AttributeError:
        roc_auc = "N/A (predict_proba not available for this classifier)"

    print("Area under the ROC curve: ", roc_auc)
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred

# Train the Decision Tree classifier
dt = DecisionTreeClassifier(random_state=42)
dt, y_pred_dt = classify(dt, X_train, y_train, X_test, y_test)

# Confusion matrix for Decision Tree
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
print("Confusion Matrix - Decision Tree:")
print(conf_matrix_dt)

# Plotting the confusion matrix for Decision Tree
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Decision Tree')
plt.show()
```

```
Accuracy:  0.9541666666666667
Area under the ROC curve:  0.8262222222222223
F-metric:  0.6496815286624203

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1125
           1       0.62      0.68      0.65        75

    accuracy                           0.95      1200
   macro avg       0.80      0.83      0.81      1200
weighted avg       0.96      0.95      0.96      1200


Evaluation by cross-validation:
[0.97321429 0.95714286 0.95714286 0.97316637 0.96779964]
Confusion Matrix - Decision Tree:
[[1094   31]
 [  24   51]]
```



In [ ]:

In [151…

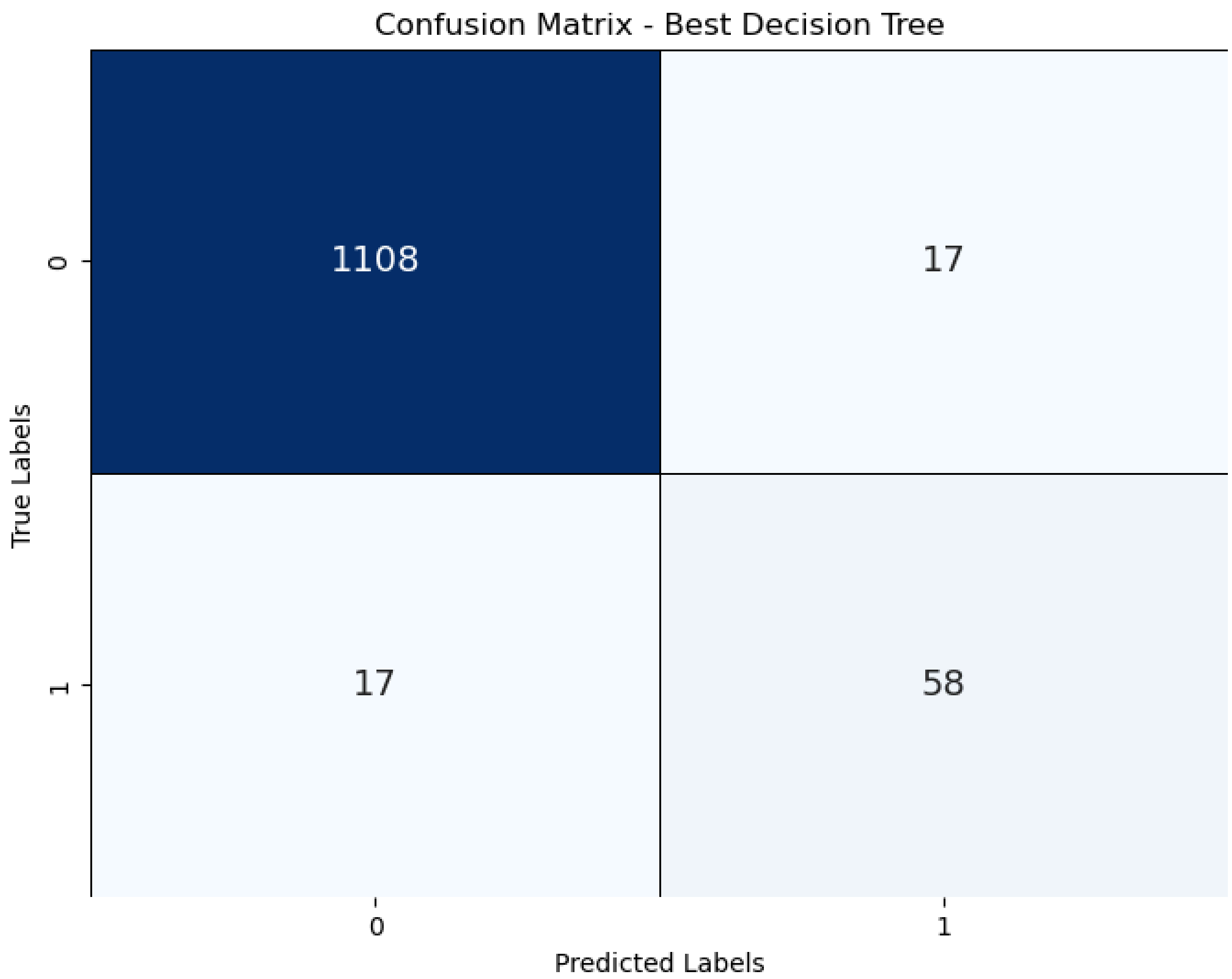
```
####GRIDSEARCH CV###########

import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelBinarizer
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure data is in the correct format
X_encoded = X_encoded.values
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
    try:
        y_pred_proba = est.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_pred_proba)
    except AttributeError:
        roc_auc = "N/A (predict_proba not available for this classifier)"

    print("Area under the ROC curve: ", roc_auc)
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred

# Define the parameter grid
param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 10, 20, 30],
    'min_samples_leaf': [1, 5, 10, 20],
    'max_features': [None, 'sqrt', 'log2']
}

# Instantiate the GridSearchCV object
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42), param_grid=param_grid, cv=5, n_jobs=-1, scoring=

# Fit the model
grid_search.fit(X_train, y_train)

# Print the best parameters and the best score
print("Best parameters found: ", grid_search.best_params_)
print("Best accuracy: ", grid_search.best_score_)

# Use the best estimator for predictions
best_dt = grid_search.best_estimator_
best_dt, y_pred_best_dt = classify(best_dt, X_train, y_train, X_test, y_test)

# Confusion matrix for the best Decision Tree
conf_matrix_best_dt = confusion_matrix(y_test, y_pred_best_dt)
```

```
print("Confusion Matrix - Best Decision Tree:")
print(conf_matrix_best_dt)

# Plotting the confusion matrix for the best Decision Tree
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_best_dt, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Best Decision Tree')
plt.show()
```

```
Best parameters found:  {'max_depth': None, 'max_features': None, 'min_samples_leaf': 20, 'min_samples_split': 2}
Best accuracy:  0.9764119601328904
Accuracy:  0.9716666666666667
Area under the ROC curve:  0.9788207407407409
F-metric:  0.7733333333333333

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1125
           1       0.77      0.77      0.77        75

    accuracy                           0.97      1200
   macro avg       0.88      0.88      0.88      1200
weighted avg       0.97      0.97      0.97      1200


Evaluation by cross-validation:
[0.97678571 0.97321429 0.97857143 0.97674419 0.97674419]
Confusion Matrix - Best Decision Tree:
[[1108   17]
 [  17   58]]
```



In [ ]:

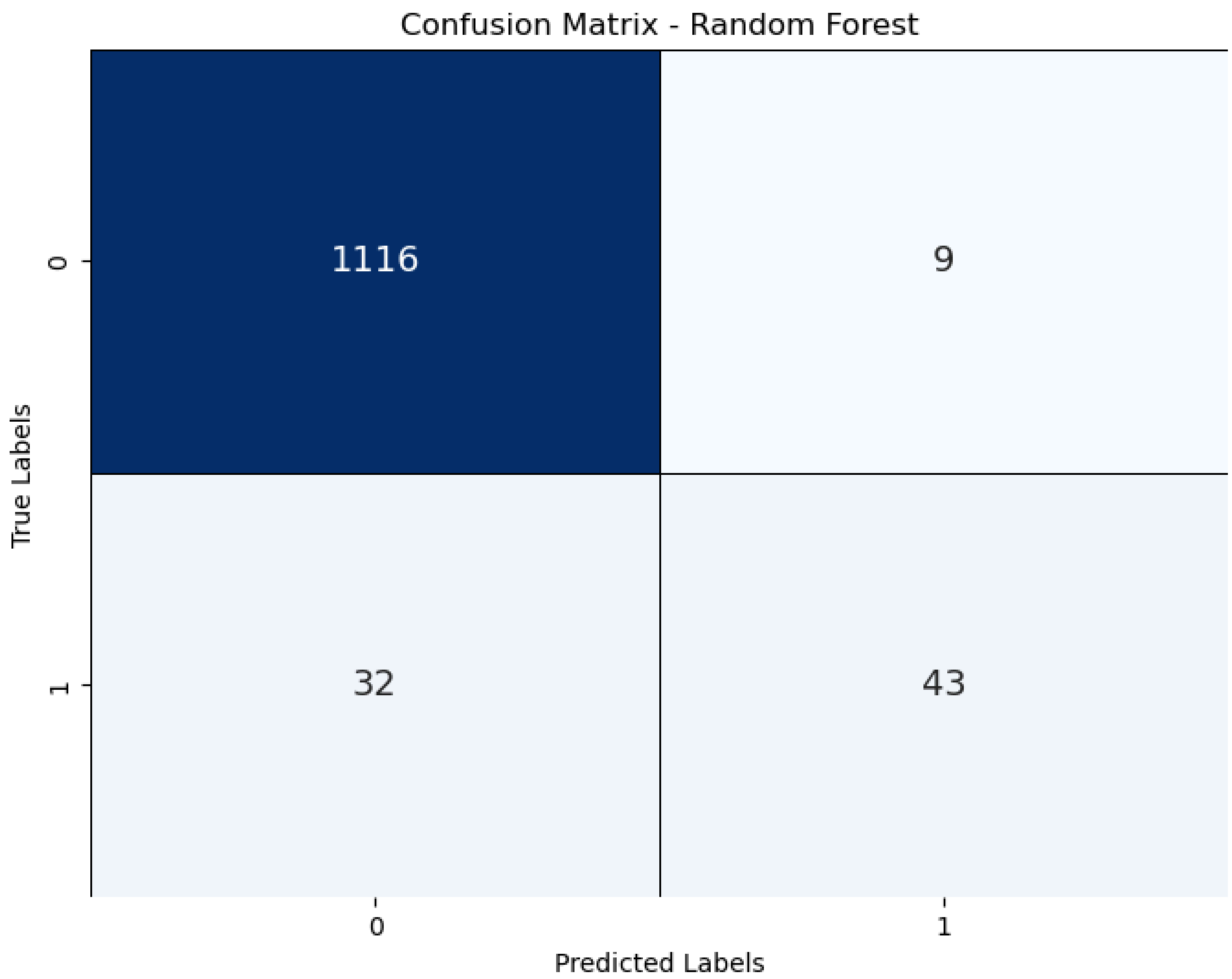
In [152…

```
#######RANDOM FOREST CLASSIFIER ##########

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure data is in the correct format
X_encoded = X_encoded.values
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
    try:
        print("Area under the ROC curve: ", roc_auc_score(y_test, est.predict_proba(X_test)[:, 1]))
```

```
    except AttributeError:
        print("Area under the ROC curve cannot be calculated for this model.")
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred

# Train the Random Forest classifier
rf = RandomForestClassifier(random_state=42)
rf, y_pred_rf = classify(rf, X_train, y_train, X_test, y_test)

# Confusion matrix for Random Forest
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix - Random Forest:")
print(conf_matrix_rf)

# Plotting the confusion matrix for Random Forest
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

```
Accuracy:  0.9658333333333333
Area under the ROC curve:  0.9845807407407408
F-metric:  0.6771653543307087

Classification report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1125
           1       0.83      0.57      0.68        75

    accuracy                           0.97      1200
   macro avg       0.90      0.78      0.83      1200
weighted avg       0.96      0.97      0.96      1200


Evaluation by cross-validation:
[0.975      0.97321429 0.96964286 0.96958855 0.96779964]
Confusion Matrix - Random Forest:
[[1116    9]
 [  32   43]]
```



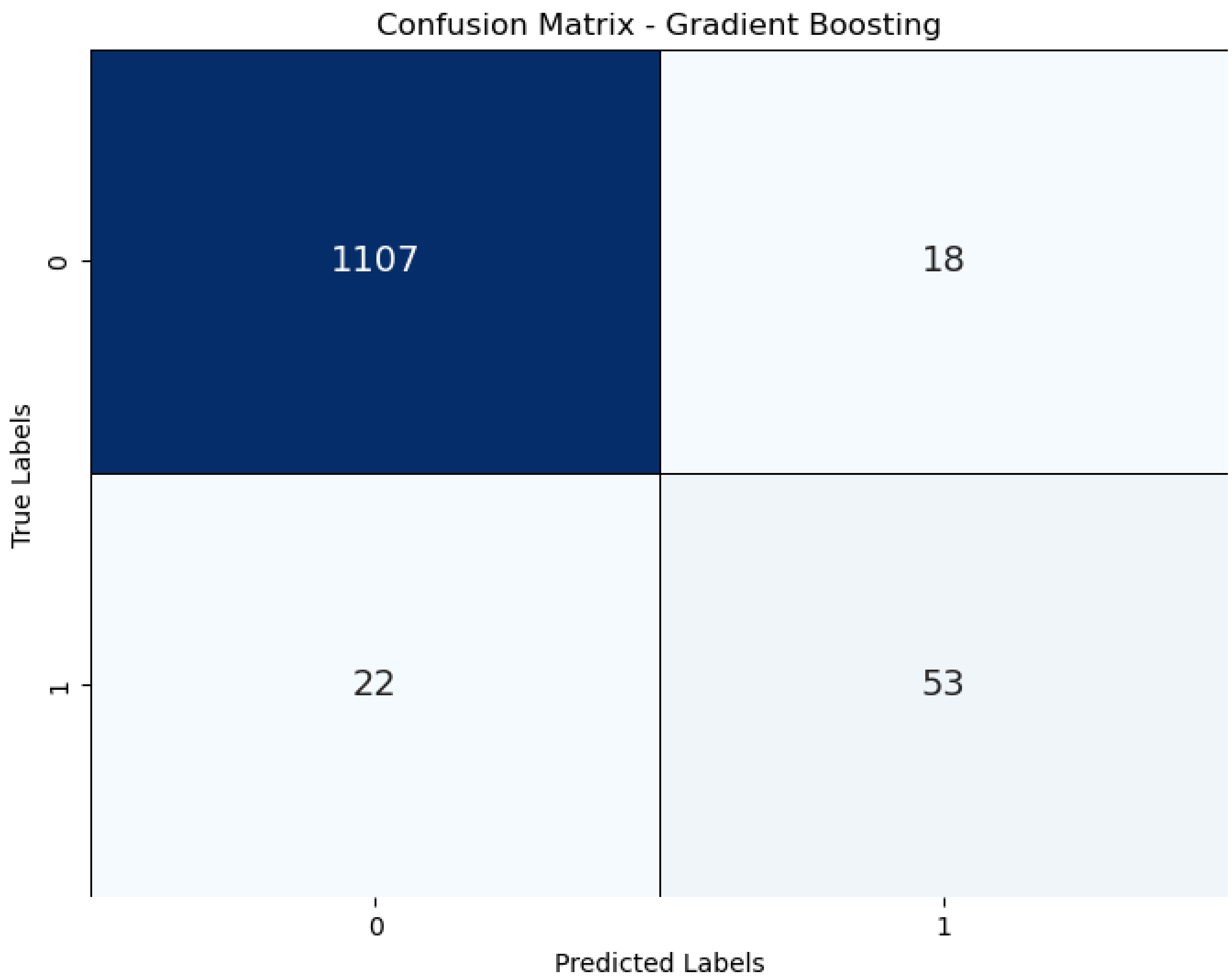
In [153…

```
######3GRADIENT BOOSTING CLASSIFIER############
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Drop rows with missing values to ensure consistent lengths

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Ensure data is in the correct format
X_encoded = X_encoded.values
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_state=42)

# Define the classify function
def classify(est, x, y, X_test, y_test):
    est.fit(x, y)
    y_pred = est.predict(X_test)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
    try:
        print("Area under the ROC curve: ", roc_auc_score(y_test, est.predict_proba(X_test)[:, 1]))
    except AttributeError:
```

```
        print("Area under the ROC curve cannot be calculated for this model.")
    print("F-metric: ", f1_score(y_test, y_pred))
    print(" ")
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    print(" ")
    print("Evaluation by cross-validation:")
    print(cross_val_score(est, x, y, cv=5))
    return est, y_pred

# Train the Gradient Boosting classifier
gb = GradientBoostingClassifier(random_state=42)
gb, y_pred_gb = classify(gb, X_train, y_train, X_test, y_test)

# Confusion matrix for Gradient Boosting
conf_matrix_gb = confusion_matrix(y_test, y_pred_gb)
print("Confusion Matrix - Gradient Boosting:")
print(conf_matrix_gb)

# Plotting the confusion matrix for Gradient Boosting
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_gb, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Gradient Boosting')
plt.show()
```

```
Accuracy:  0.9666666666666667
Area under the ROC curve:  0.9850370370370369
F-metric:  0.726027397260274

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1125
           1       0.75      0.71      0.73        75

    accuracy                           0.97      1200
   macro avg       0.86      0.85      0.85      1200
weighted avg       0.97      0.97      0.97      1200


Evaluation by cross-validation:
[0.975      0.9625     0.97321429 0.97316637 0.980322  ]
Confusion Matrix - Gradient Boosting:
[[1107   18]
 [  22   53]]
```



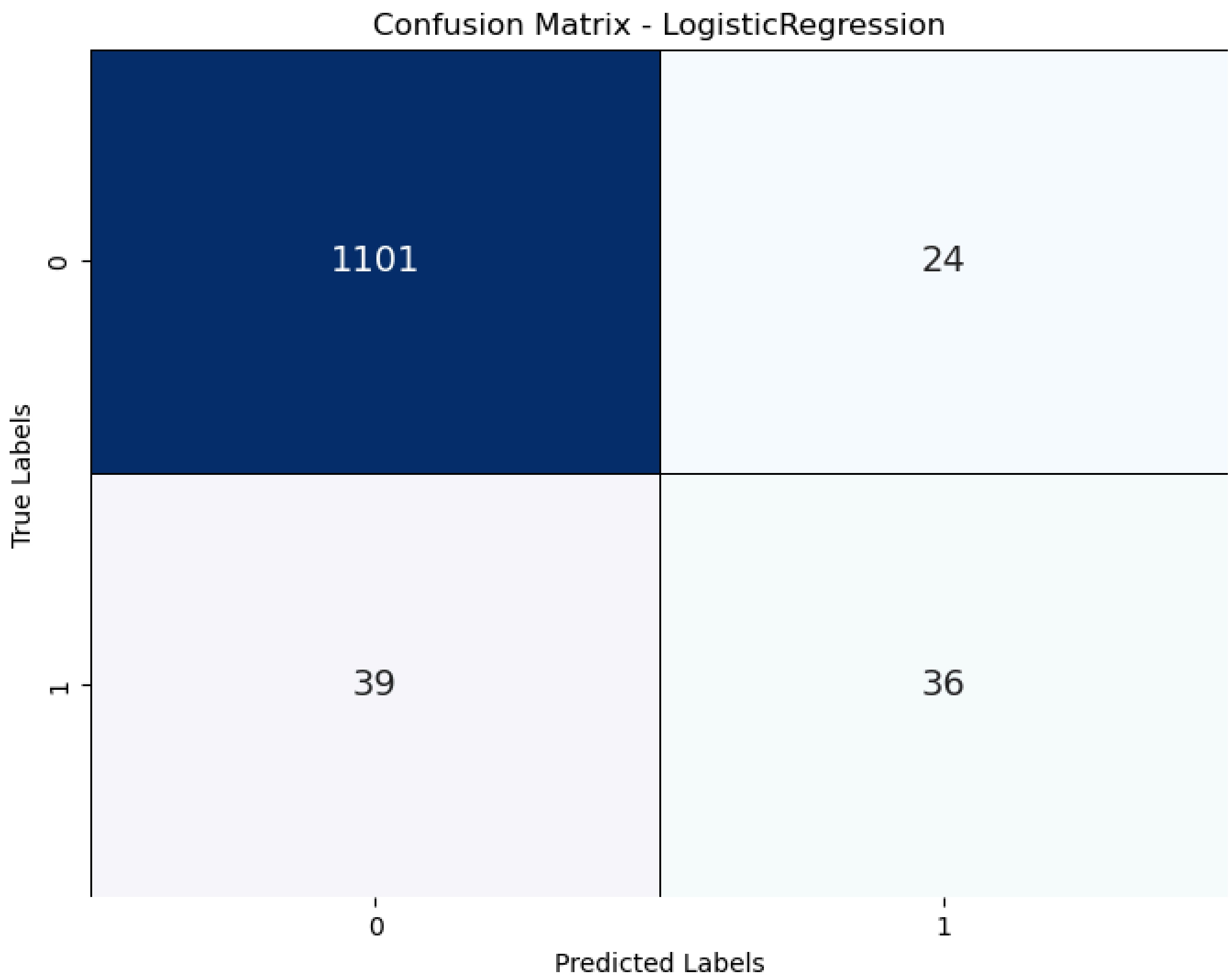
In [194…

```
import os
import joblib
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer, StandardScaler
from sklearn.metrics import f1_score, accuracy_score, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
```

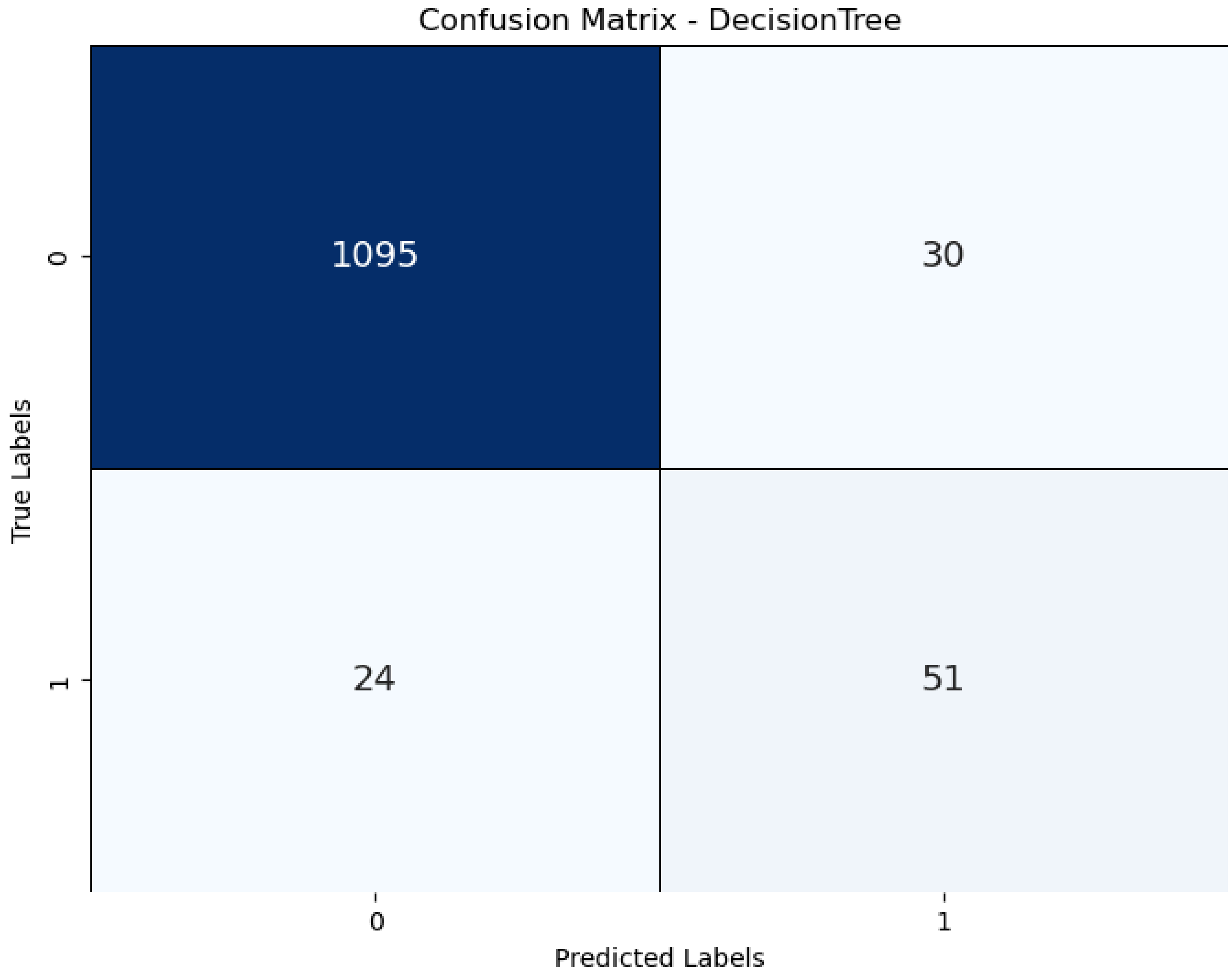
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure the model folder exists
model_folder = "election_models"
os.makedirs(model_folder, exist_ok=True)

# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Binarize the target variable
lb_style = LabelBinarizer()
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Encode categorical variables
X_encoded = pd.get_dummies(X, drop_first=True)

# Standardize numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Define classifiers
```

```python
classifiers = {
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "DecisionTree": DecisionTreeClassifier(random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42),
    "GradientBoosting": GradientBoostingClassifier(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    "KNN": KNeighborsClassifier()
}

# Train, evaluate, and save models
for name, model in classifiers.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    f1 = f1_score(y_test, y_pred)

    # Save model with F1 score in filename
    model_filename = f"{name}_model_f1_{f1:.4f}.pkl"
    model_path = os.path.join(model_folder, model_filename)
    joblib.dump(model, model_path)
    print(f"{name} model saved as {model_path}")

    # Print evaluation metrics
    print(f"\n{name} Model Performance:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("F1 Score:", f1)
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Plot Confusion Matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
                annot_kws={'size': 14}, linewidths=0.5, linecolor='black')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()
```

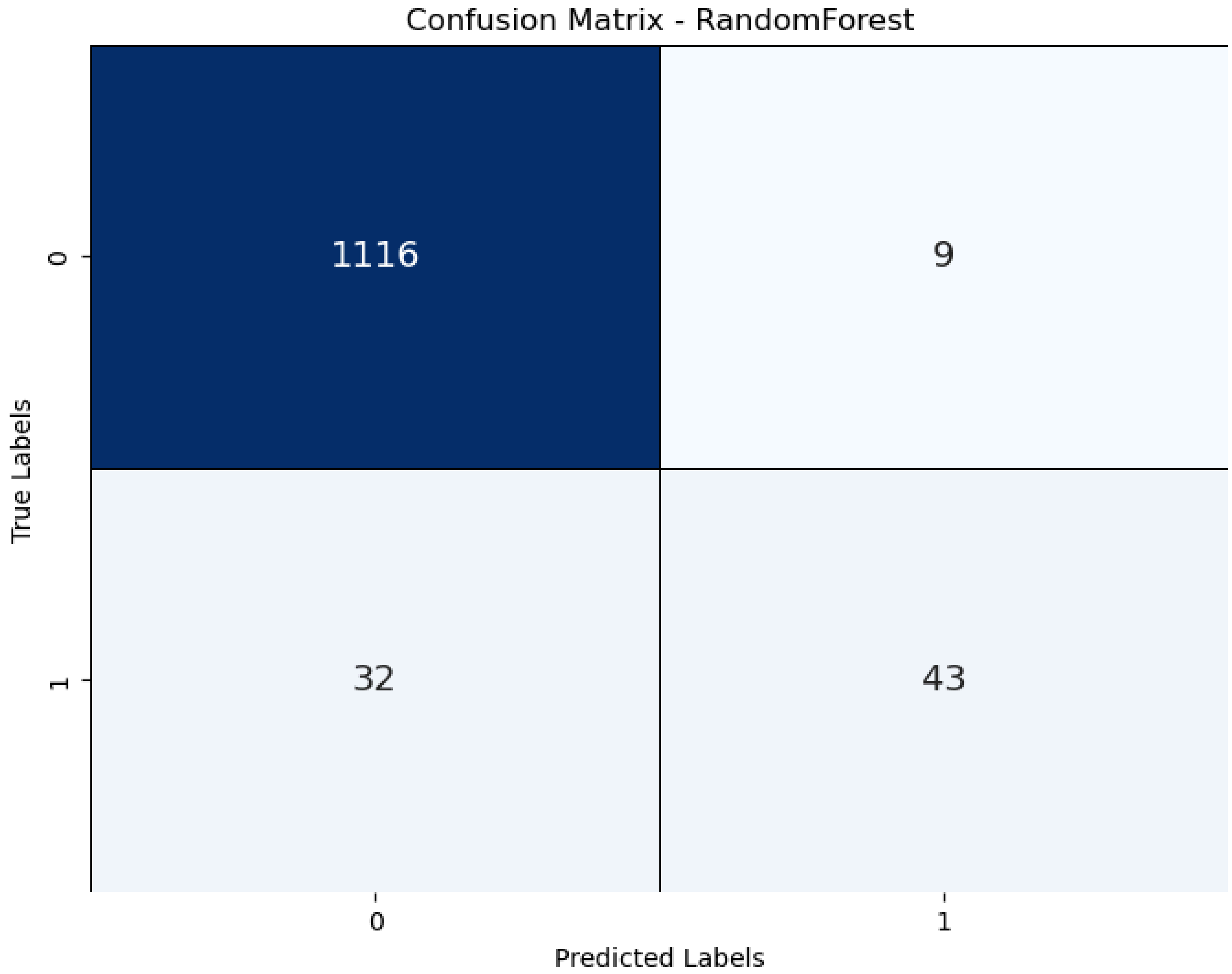
```
LogisticRegression model saved as election_models\LogisticRegression_model_f1_0.5333.pkl

LogisticRegression Model Performance:
Accuracy: 0.9475
F1 Score: 0.5333333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      1125
           1       0.60      0.48      0.53        75

    accuracy                           0.95      1200
   macro avg       0.78      0.73      0.75      1200
weighted avg       0.94      0.95      0.94      1200
```


Confusion Matrix - LogisticRegression
1101
24
39
36
0
1
True Labels
0
1
Predicted Labels

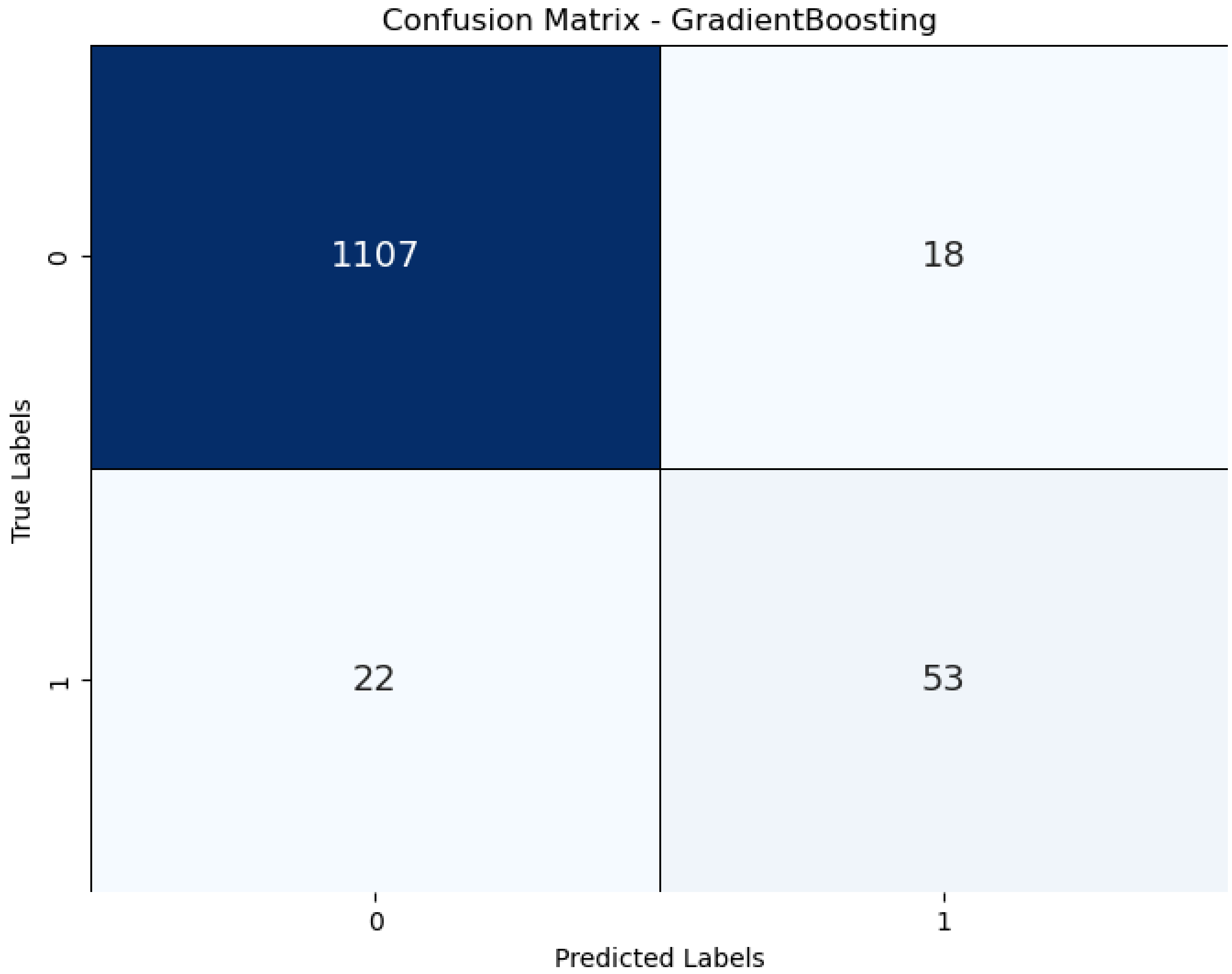
```
DecisionTree model saved as election_models\DecisionTree_model_f1_0.6538.pkl

DecisionTree Model Performance:
Accuracy: 0.955
F1 Score: 0.6538461538461539
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.97      0.98      1125
           1       0.63      0.68      0.65        75

    accuracy                           0.95      1200
   macro avg       0.80      0.83      0.81      1200
weighted avg       0.96      0.95      0.96      1200
```


Confusion Matrix - DecisionTree
1095
30
24
51
True Labels
0
1
Predicted Labels
0
1

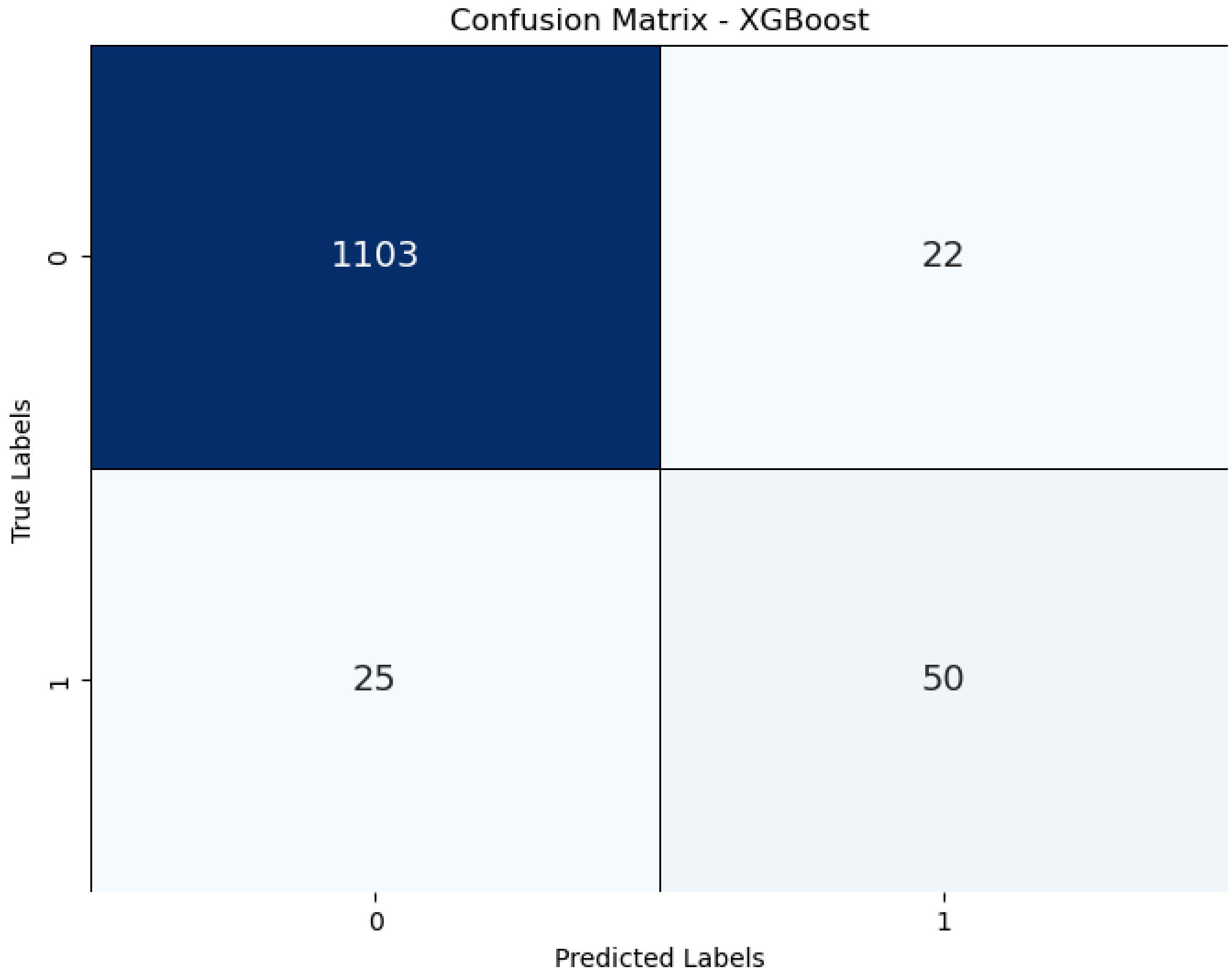
```
RandomForest model saved as election_models\RandomForest_model_f1_0.6772.pkl

RandomForest Model Performance:
Accuracy: 0.9658333333333333
F1 Score: 0.6771653543307087
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98      1125
           1       0.83      0.57      0.68        75

    accuracy                           0.97      1200
   macro avg       0.90      0.78      0.83      1200
weighted avg       0.96      0.97      0.96      1200
```


Confusion Matrix - RandomForest
1116
9
32
43
0
1
True Labels
0
1
Predicted Labels

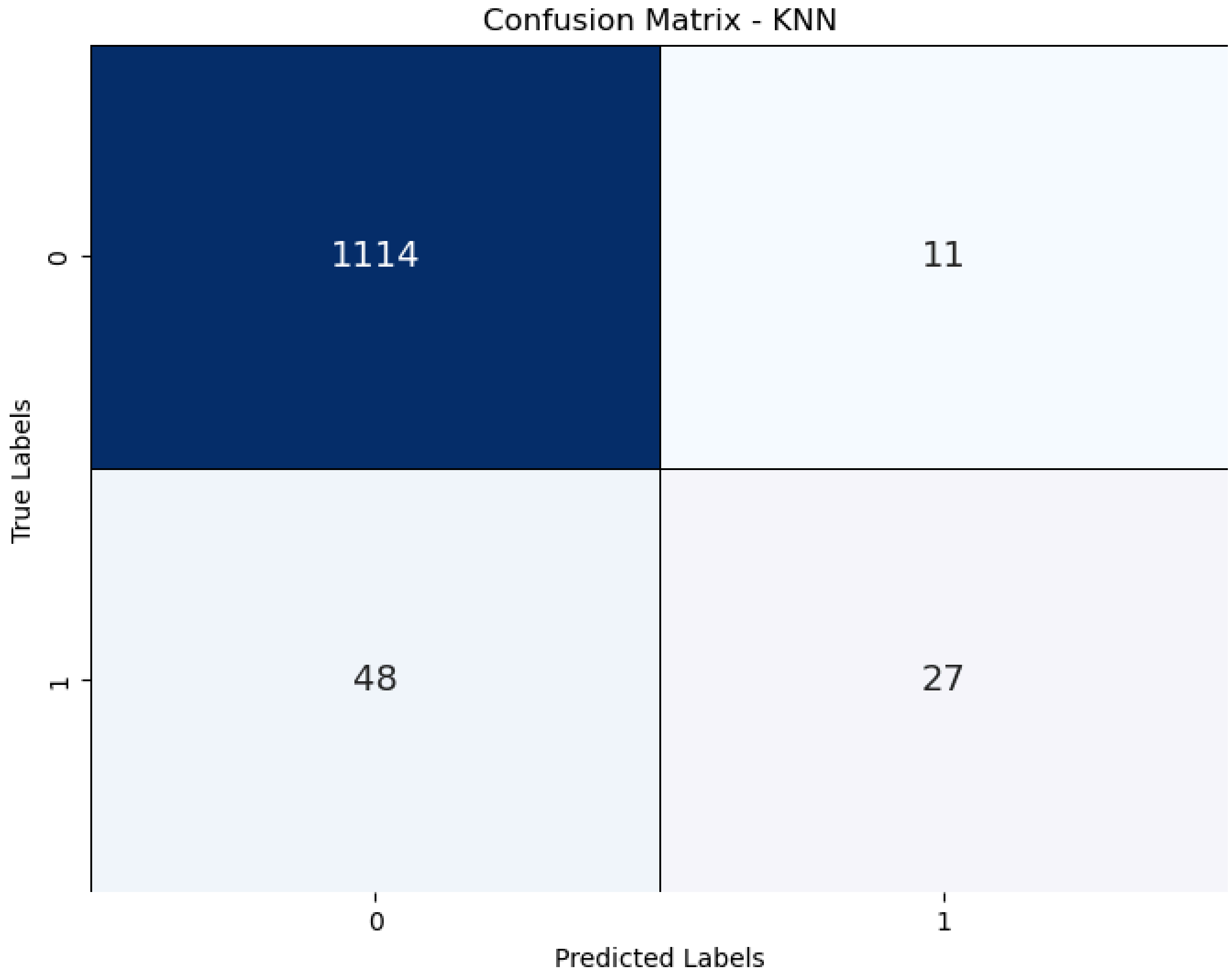
```
GradientBoosting model saved as election_models\GradientBoosting_model_f1_0.7260.pkl

GradientBoosting Model Performance:
Accuracy: 0.9666666666666667
F1 Score: 0.726027397260274
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1125
           1       0.75      0.71      0.73        75

    accuracy                           0.97      1200
   macro avg       0.86      0.85      0.85      1200
weighted avg       0.97      0.97      0.97      1200
```


Confusion Matrix - GradientBoosting
1107
18
22
53
True Labels
Predicted Labels
0
1

```
XGBoost model saved as election_models\XGBoost_model_f1_0.6803.pkl

XGBoost Model Performance:
Accuracy: 0.9608333333333333
F1 Score: 0.6802721088435374
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1125
           1       0.69      0.67      0.68        75

    accuracy                           0.96      1200
   macro avg       0.84      0.82      0.83      1200
weighted avg       0.96      0.96      0.96      1200
```


Confusion Matrix - XGBoost
1103
22
25
50
True Labels
0
1
0
1
Predicted Labels

```
KNN model saved as election_models\KNN_model_f1_0.4779.pkl

KNN Model Performance:
Accuracy: 0.9508333333333333
F1 Score: 0.4778761061946903
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      1125
           1       0.71      0.36      0.48        75

    accuracy                           0.95      1200
   macro avg       0.83      0.68      0.73      1200
weighted avg       0.94      0.95      0.94      1200
```



In [ ]:

In [ ]:

In [192…

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer, StandardScaler
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
import plotly.graph_objects as go

# Define a function to plot interactive bar charts
def plot_metrics(model_names, accuracy, precision, recall, f1_score):
    fig = go.Figure()

    # Add bars for each model and each metric
    for i, model_name in enumerate(model_names):
        fig.add_trace(go.Bar(name=model_name, x=['Accuracy', 'Precision', 'Recall', 'F1 Score'],
                             y=[accuracy[i], precision[i], recall[i], f1_score[i]]))

    # Update layout
    fig.update_layout(barmode='group',
                      title='Model Metrics Comparison',
                      xaxis_title='Metrics',
                      yaxis_title='Score')

    # Show plot
    fig.show()


# Assuming 'df' is your DataFrame
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Ensure consistent data types in 'Result' column
data['Result'] = data['Result'].astype(str)

# Binarize the target variable
lb_style = LabelBinarizer()
```

```
y = lb_style.fit_transform(data['Result']).ravel()

# Separate the 'Result' column
X = data.drop(columns=['Result'])

# Apply pd.get_dummies to encode categorical variables, dropping the first category to avoid multicollinearity
X_encoded = pd.get_dummies(X, drop_first=True)

# Standardize numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Train the models and calculate metrics

# Logistic Regression
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)

# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

# Gradient Boosting
gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
precision_gb = precision_score(y_test, y_pred_gb)
```

```
recall_gb = recall_score(y_test, y_pred_gb)
f1_gb = f1_score(y_test, y_pred_gb)

# XGBoost
xgb_model = xgb.XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
precision_xgb = precision_score(y_test, y_pred_xgb)
recall_xgb = recall_score(y_test, y_pred_xgb)
f1_xgb = f1_score(y_test, y_pred_xgb)

# K-Nearest Neighbors
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)

# Append Decision Tree model to lists
model_names = ['Logistic Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost', 'K-Nearest Neighbors', 'Decision Tree']
accuracy_scores = [accuracy_lr, accuracy_rf, accuracy_gb, accuracy_xgb, accuracy_knn, accuracy_dt]
precision_scores = [precision_lr, precision_rf, precision_gb, precision_xgb, precision_knn, precision_dt]
recall_scores = [recall_lr, recall_rf, recall_gb, recall_xgb, recall_knn, recall_dt]
f1_scores = [f1_lr, f1_rf, f1_gb, f1_xgb, f1_knn, f1_dt]

# Plotting interactive bar charts
plot_metrics(model_names, accuracy_scores, precision_scores, recall_scores, f1_scores)
```

MODEL PREDICTION

In [ ]:

```
```

In [180…

```
import os
import joblib
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, RandomForestClassifier, GradientBoostingClassif
from xgboost import XGBRegressor, XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import pickle

# Load dataset for reference (to get column names and unique values)
df=pd.read_csv("tnelectionsformlmodels.csv")

# Load models from the folder
def load_models(model_folder="election_models"):
    models = {}
    for file in os.listdir(model_folder):
        if file.endswith(".pkl"):
            model_name = file.split("_model")[0]
            models[model_name] = joblib.load(os.path.join(model_folder, file))
    return models

# Prompt user for input
def get_user_input(df):
    user_input = {}
    for column in df.columns:
        if df[column].dtype == 'object' or df[column].dtype == 'bool':
            unique_values = df[column].dropna().unique()
            print(f"Available values for {column}: {list(unique_values)}")
        value = input(f"Enter value for {column}: ")
        user_input[column] = value
    return user_input

# Convert user input into DataFrame
def preprocess_input(user_input, df):
    input_df = pd.DataFrame([user_input])
    input_df = pd.get_dummies(input_df, drop_first=True)
    all_columns = pd.get_dummies(df, drop_first=True).columns
    for col in all_columns:
        if col not in input_df.columns:
```

```
            input_df[col] = 0
    input_df = input_df[all_columns]
    return input_df.values

# Select only relevant features
data = df[['Sex', 'Party', 'Age', 'Electors',
           'Constituency_Type', 'District_Name', 'Sub_Region', 'No_of_Candidates',
           'ENOP', 'Contested', 'Turncoat', 'Incumbent', 'Recontest', 'Education_Qualification',
           'Main_Profession', 'Second_Profession', 'Result', 'Alliance', 'Votes']]

# Remove rows where Party is "NOTA"
data = data[data['Party'] != 'NOTA']

# Convert 'Result' column to string type
data['Result'] = data['Result'].astype(str)

# Binarize the target variable
lb_style = LabelBinarizer()
y_result = lb_style.fit_transform(data['Result']).ravel()
y_votes = data['Votes']

# Separate features and target variable
X = data.drop(columns=['Result', 'Votes'])

# Encode categorical features
X_encoded = pd.get_dummies(X, drop_first=True)

# Split the dataset
X_train, X_test, y_train_result, y_test_result, y_train_votes, y_test_votes = train_test_split(X_encoded, y_result, y_votes, t

# Define classifiers
classifiers = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier()
}

# Define regressors
```

```
regressors = {
    "Random Forest Regressor": RandomForestRegressor(),
    "Gradient Boosting Regressor": GradientBoostingRegressor(),
    "XGBoost Regressor": XGBRegressor()
}

# Train and save classifiers
for name, model in classifiers.items():
    model.fit(X_train, y_train_result)
    joblib.dump(model, f"{name.replace(' ', '_')}_classifier.pkl")

# Train and save regressors
for name, model in regressors.items():
    model.fit(X_train, y_train_votes)
    joblib.dump(model, f"{name.replace(' ', '_')}_regressor.pkl")

print("✅ All models trained and saved successfully!")

# ------------------- USER INPUT & PREDICTION -------------------

# Get categorical and boolean columns
categorical_cols = X.select_dtypes(include=['object', 'bool']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Prompt user for input
user_input = {}
print("\n🔹 Enter the values for the following features:")

for col in X.columns:
    if col in categorical_cols:
        unique_values = df[col].dropna().unique()
        print(f"\n👉 {col} (Choose from: {unique_values})")
        user_input[col] = input(f"Enter {col}: ")
    else:
        user_input[col] = float(input(f"\n👉 {col}: "))

# Convert input to DataFrame
user_df = pd.DataFrame([user_input])

# Encode user input using the same encoding as training data
user_encoded = pd.get_dummies(user_df)
```

```
missing_cols = set(X_encoded.columns) - set(user_encoded.columns)
for col in missing_cols:
    user_encoded[col] = 0  # Add missing columns with default value

user_encoded = user_encoded[X_encoded.columns]  # Ensure same column order

# Load models and predict
print("\n🔍 Predictions from each model:\n")
for name, model in classifiers.items():
    model = joblib.load(f"{name.replace(' ', '_')}_classifier.pkl")
    prediction = model.predict(user_encoded)[0]
    predicted_class = lb_style.classes_[prediction]  # Convert to original label
    print(f"✅ {name} (Result): {predicted_class}")

for name, model in regressors.items():
    model = joblib.load(f"{name.replace(' ', '_')}_regressor.pkl")
    prediction = model.predict(user_encoded)[0]
    print(f"✅ {name} (Votes): {prediction}")
```

```
C:\Users\ariva\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
✅ All models trained and saved successfully!

🔹 Enter the values for the following features:

👉 Sex (Choose from: ['M' 'F' 'O'])
```

```
👉 Party (Choose from: ['DMK' 'PMK' 'NTK' 'DMDK' 'NOTA' 'BSP' 'IJK' 'IND'
 'Anaithu Makkal Arasiyal Katchi' 'INC' 'ADMK' 'MNM' 'AMMK'
 'Anna MGR Dravida Makkal Kalgam' 'All India Jananayaka Makkal Kazhagam'
 'My India Party' 'Tamil Nadu Ilangyar Katchi'
 'Tamilnadu Makkal Nalvazhvu Periyakkam' 'MGR Makkal Katchi' 'NCP'
 'Bahujan Dravida Party' 'Desiya Makkal Sakthi Katchi' 'SHS'
 'Naadaalum Makkal Katchi' 'Samata Party' 'SUCI' 'RPI(A)'
 'Makkal Nalvaazhvuk Katchi' 'United States of India Party'
 'New Generation People’s Party' 'Thamizhaga Munnetra Congress'
 'Veerath Thiyagi Viswanathadoss Thozhilalarkal Katchi'
 'National Democratic Party of South India' 'Makkalatchi Katchi'
 'Valamaana  Tamizhagam   Katchi' 'Nam India Naam Indiyar Katchi'
 'Republican Party of India  (Sivaraj)' 'LJP' 'Tipu Sultan Party' 'BJP'
 'RJD' 'Tamizhaga Murpokku Makkal Katchi'
 'Mahathma Makkal Munnetra Kazhakam' 'Jebamani Janata'
 'Anna MGR Dravida Munnetra Kazhagam' 'SDPI'
 'Bhartiya Manavadhikaar Federal Party' 'Puthiya Tamilagam'
 'Dravida Murpokku Makkal Katchi' 'CPI(ML)(L)' 'VCK'
 'Desiya Sirupanmayinar Makkal Iyakkam' 'Makkal Munnetra Peravai'
 'Kamarajar Deseeya Congress' 'All Pensioner’s Party' 'IUML' 'AIMIM'
 'Akhil Bharat Hindu Mahasabha' 'All India Youth Development Party'
 'Anaithu Makkal Puratchi Katchi' 'Rashtriya Ulama Council'
 'All India Uzhavargal Uzhaippalargal Katchi' 'Makkal Nala Kazhagam'
 'Samaniya Makkal Nala Katchi'
 'Anna Puratchi Thalaivar Amma Dravida Munnetra Kazhagam' 'NPP' 'CPI' 'SP'
 'Dhesiya Makkal Kazhagam' 'CPM' 'Anna Dravidar Kazhagam'
 'Ganasangam Party of India' 'Anaithindia Samudaya Munnetra Kazhagam'
 'All  India  Pattali  Munnetra Katchi' 'MAKKAL SAKTHI KATCHI'
 'Indhia Kudiarasu Katchi' 'Ambedkarite Party of India'
 'Tamilaga Makkal Thannurimai Katchi' 'Ahimsa Socialist Party'
 'Manitha Urimaigal Kalaagam' 'Namathu Kongu Munnetra Kalagam'
 'Makkal Thilagam Munnetra Kazhagam,'
 'India Dravida Makkal Munnetra Katchi'
 'Kongu  Desa  Marumalarchi  Makkal  Katchi' 'AITC'
 'Ezhuchi Tamilargal Munnetra Kazhagam' 'Hindustan Janta Party'
 'Makkal Sananayaga Kudiyarasu Katchi,' 'Anaithu Makkal Munnetra Kazhagam'
 'RSPS' 'Vidial Valarchi Perani' 'Akila India Vallalar Peravai' 'JD(S)'
 'Ilantamilar Munnani Kazhagam' 'AISMK' "People's Party of India(secular)"
 'Namadhu Makkal Katchi'
 'Akhila India Jananayaka Makkal Katchi (Dr. Isaac)'
 'Tamil Telugu National Party' 'Ambedkar Political Party'
```

```
'All India MGR Makkal Munnetra Kazhagam' 'Naam Indiar Party'
'Desa Makkal Munnetrak Kazhgam' 'South India Forward Bloc'
'Aanaithinthiya Jananayaka Pathukappu Kazhagam'
'Universal Brotherhood Movement'
'Anaithu Ulaga Tamilargal Munnetra Kalagam' 'Ulaga Makkal Katchi'
'Tamilnadu Mahatma Gandhi Makkal Katchi'])
👉 Constituency_Type (Choose from: ['GEN' 'SC' 'ST'])
👉 District_Name (Choose from: ['TIRUVALLUR' 'CHENNAI' 'KANCHIPURAM' 'VELLORE' 'KRISHNAGIRI' 'DHARMAPURI'
'TIRUVANNAMALAI' 'VILLUPURAM' 'SALEM' 'NAMAKKAL' 'ERODE' 'TIRUPUR'
'NILGIRIS' 'COIMBATORE' 'DINDIGUL' 'KARUR' 'TIRUCHIRAPPALLI' 'PERAMBALUR'
'ARIYALUR' 'CUDDALORE' 'NAGAPATTINAM' 'TIRUVARUR' 'THANJAVUR'
'PUDUKKOTTAI' 'SIVAGANGA' 'MADURAI' 'THENI' 'VIRUDHUNAGAR'
'RAMANATHAPURAM' 'THOOTHUKUDI' 'TIRUNELVELI' 'KANNIYAKUMARI'])
👉 Sub_Region (Choose from: ['CHENNAI CITY REGION' 'WESTERN REGION' 'SOUTHERN REGION' 'CENTRAL REGION'])
👉 Turncoat (Choose from: [False True])
👉 Incumbent (Choose from: [False True])
👉 Recontest (Choose from: [False True])
👉 Education_Qualification (Choose from: ['10th Pass' '8th Pass' 'Graduate Professional' 'Others' 'Graduate'
'Post Graduate' '12th Pass' '5th Pass' 'Illiterate' 'Doctorate'
'Literate'])
👉 Main_Profession (Choose from: ['Business' 'Salaried Work or Employed' 'Agriculture' 'Other'
'Labourer or Daily Wage' 'Liberal Profession or Professional' 'Education'
'Small Business or Self-employed' 'Social Work' 'Unemployed' 'Politics'
'Retired or Pension' 'Agricultural Labour' 'Former Government'
'Traditional Occupation' 'Student' 'Religious Occupation'])
👉 Second_Profession (Choose from: ['Social Work' 'Agriculture' 'Small Business or Self-employed' 'Politics'
'Student' 'Liberal Profession or Professional' 'Education'
'Salaried Work or Employed' 'Labourer or Daily Wage'
'Agricultural Labour' 'Retired or Pension' 'Traditional Occupation'])
👉 Alliance (Choose from: ['SPA' 'NDA' 'NTK' 'PF' 'IND' 'BSP' 'PFA'])
```

```
C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
```

```
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
```

```
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:
```

```
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:
```

```
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`
```

```
C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`
```

```
C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
```

```
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
```

```
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:
```

```
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:
```

```
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`
```

```
C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`
```

```
C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
```

```
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`

C:\Users\ariva\AppData\Local\Temp\ipykernel_30420\2290769046.py:128: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy
()`
```

```
🔍 Predictions from each model:

✅ Logistic Regression (Result): Lost
✅ Random Forest (Result): Won
✅ Gradient Boosting (Result): Won
✅ XGBoost (Result): Won
✅ K-Nearest Neighbors (Result): Lost
✅ Decision Tree (Result): Won
✅ Random Forest Regressor (Votes): 95401.75
✅ Gradient Boosting Regressor (Votes): 94863.11547135469
✅ XGBoost Regressor (Votes): 92575.859375
```

In [ ]:

In [ ]:

Importing packages

In [64]:

```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from scipy.stats import linregress
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
```

Linear Regression Line for Electors vs Valid votes

In [66]:

```
df = df[df['Position'] == 1]
electors = df['Electors']
valid_votes = df['Valid_Votes']
fig = px.scatter(x=electors, y=valid_votes, trendline="ols", labels={'x': 'Electors', 'y': 'Valid Votes'},
```

```
                 title='Scatter Plot of Electors vs Valid Votes with Regression Line')
fig.show()
```

Checking for r squared and r adjusted squared value

In [68]:
```
slope, intercept, r_value, p_value, std_err = linregress(valid_votes, electors)
r_squared = r_value ** 2
```

```
n = len(electors)
k = 1
r_adj_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

a = r_squared
b = r_adj_squared

print(f"R-squared:", a)
print(f"Adjusted R-squared:", b)

X_train, X_test, y_train, y_test = train_test_split(valid_votes, electors, test_size=0.2, random_state=25)
```

```
R-squared: 0.7126528845157296
Adjusted R-squared: 0.7114143193627802
```

HYPERTUNING AND PARAMETER TESTING

In [70]:

```
X = df[['Electors']]
y = df['Valid_Votes']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)

linear_reg = LinearRegression()
param_grid = {
    'fit_intercept': [True, False],
    'positive': [True, False]
}
grid_search = GridSearchCV(estimator=linear_reg, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X, y)

c=grid_search.best_estimator_.score(X, y)

print(grid_search.best_params_)
print('R2 Score of Test is :', c)

best_slope = grid_search.best_estimator_.coef_[0]
best_intercept = grid_search.best_estimator_.intercept_
```

```
{'fit_intercept': True, 'positive': True}
R2 Score of Test is : 0.7126528845157284
```

fit_intercept: When set to True (default), the model calculates the intercept. If set to False, no intercept will be calculated, and the regression line will pass through the origin (0,0). If the data is not centered, setting this parameter to False may result in biased estimates.

positive: When set to True, forces the coefficients of the linear regression model to be positive. This option is useful when you want to enforce positivity constraints on the coefficients, such as in cases where negative coefficients don't make sense (e.g., predicting quantities that cannot be negative). It is only supported for dense arrays.

R2 score similar after and before :

There might not be any Negative Coefficients: The coefficients of the linear regression model might naturally tend to be positive. In this case, enforcing the positive constraint wouldn't change the coefficients much because they were already positive or close to zero.

Bar visualization for each score

In [73]:
```
labels = ['R-squared', 'Adjusted R-squared', 'R2 Score of Test']
values = [a, b, c]

fig = go.Figure([go.Bar(x=labels, y=values)])

fig.update_layout(title='Comparison of R-squared, Adjusted R-squared, and R2 Score',
                  xaxis_title='Metrics',
                  yaxis_title='Value')

fig.show()
```

Comparing similar models to check for best model

DECISION TREE REGRESSOR R2 SCORE

In [76]:
```
X = df[['Electors']]
y = df['Valid_Votes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)

decision_tree_reg = DecisionTreeRegressor()
decision_tree_reg.fit(X_train, y_train)

c = decision_tree_reg.score(X_test, y_test)
slope, intercept, r_value, p_value, std_err = linregress(valid_votes, electors)

r_squared = r_value ** 2

n = len(electors)
k = 1
r_adj_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

labels = ['R-squared', 'Adjusted R-squared', 'R2 Score of Test', 'Decision Tree R2 Score']
values = [r_squared, r_adj_squared, c, c]

fig = go.Figure([go.Bar(x=labels, y=values)])

fig.update_layout(title='Comparison of R-squared, Adjusted R-squared, and R2 Score between Linear Regression and Decision Tree
                  xaxis_title='Metrics',
                  yaxis_title='Value')

fig.show()
```

POLYNOMIAL REGRESSOR R2 SCORE

In [78]:
```
X = df[['Electors']]
y = df['Valid_Votes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)
```

```
poly = PolynomialFeatures(degree=2)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

poly_reg = LinearRegression()
poly_reg.fit(X_poly_train, y_train)


c = poly_reg.score(X_poly_test, y_test)

slope, intercept, r_value, p_value, std_err = linregress(valid_votes, electors)


r_squared = r_value ** 2

n = len(electors)
k = 1

r_adj_squared = 1 - ((1 - r_squared) * (n - 1) / (n - k - 1))

labels = ['R-squared', 'Adjusted R-squared', 'R2 Score of Test', 'Polynomial Regression R2 Score']
values = [r_squared, r_adj_squared, c, c]

fig = go.Figure([go.Bar(x=labels, y=values)])

fig.update_layout(title='Comparison of R-squared, Adjusted R-squared, and R2 Score between Linear Regression and Polynomial Re
                  xaxis_title='Metrics',
                  yaxis_title='Value')

fig.show()
```

DECISION TREE REGRESSOR

In [80]:
```
df = df[df['Position'] == 1]
electors = df['Electors'].values.reshape(-1, 1)
valid_votes = df['Valid_Votes']
model = DecisionTreeRegressor()
```

```
model.fit(electors, valid_votes)
electors_range = np.linspace(electors.min(), electors.max(), 100).reshape(-1, 1)
predicted_valid_votes = model.predict(electors_range)
fig = px.scatter(x=df['Electors'], y=valid_votes, labels={'x': 'Electors', 'y': 'Valid Votes'},
                 title='Scatter Plot of Electors vs Valid Votes with Decision Tree Regressor')
fig.add_trace(go.Scatter(x=electors_range.flatten(), y=predicted_valid_votes, mode='lines',
                         name='Decision Tree Regressor', line=dict(color='orange')))

fig.show()
```

In [81]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Assuming you already have df, electors, and valid_votes from your previous code
```

```python
# Filter the dataframe for Position == 1
df = df[df['Position'] == 1]
electors = df['Electors'].values.reshape(-1, 1)  # Reshape to 2D array for sklearn
valid_votes = df['Valid_Votes']

# Train the Polynomial Regressor
poly_features = PolynomialFeatures(degree=2)  # You can change the degree as needed
electors_poly = poly_features.fit_transform(electors)
model = LinearRegression()
model.fit(electors_poly, valid_votes)

# Predict Valid Votes for the entire range of Electors
electors_range = np.linspace(electors.min(), electors.max(), 100).reshape(-1, 1)
electors_range_poly = poly_features.transform(electors_range)
predicted_valid_votes = model.predict(electors_range_poly)

# Plot the scatter plot with the regression line
fig = px.scatter(x=df['Electors'], y=valid_votes, labels={'x': 'Electors', 'y': 'Valid Votes'},
                 title='Scatter Plot of Electors vs Valid Votes with Polynomial Regressor')

# Add the regression line to the plot
fig.add_trace(go.Scatter(x=electors_range.flatten(), y=predicted_valid_votes, mode='lines',
                         name='Polynomial Regressor', line=dict(color='green')))

# Show the plot
fig.show()
```

PREDICTING FOR THE BEST FIT MODEL

In [83]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
```

```
from scipy.stats import linregress

df = df[df['Position'] == 1]
electors = df['Electors']
valid_votes = df['Valid_Votes']

fig = px.scatter(x=electors, y=valid_votes, trendline="ols", labels={'x': 'Electors', 'y': 'Valid Votes'},
                 title='Scatter Plot of Electors vs Valid Votes with Regression Line')

x_value = float(input("Enter the value of Electors (x): "))
slope, intercept, _, _, _ = linregress(electors, valid_votes)
y_value = slope * x_value + intercept
print("The predicted value of Valid voters is:",y_value)
fig.add_trace(go.Scatter(x=[x_value], y=[y_value], mode='markers', marker=dict(color='red'), name='User Input'))
fig.show()
```

```
The predicted value of Valid voters is: 58257.160249348366
```

In [ ]:

```
```