Election Prediction Model

Table of Contents:

Problem Statement

Problem Statement	Your are hired by one of the leading news channels CNBE. This survey was conducted on 1525 voters with 9 variables. You have to build ML models to protect which party a voter will vote for on the basis of the given information. This information will be used to create an exit Pol that will help in predicting overall win in seats covered by a particular party. The data set of the problem is here - Election_Data_Set: Please make a note that the above file consists of 2 tabs - The data and Data Dictionary. You only need to ingest the data tab. It can be done by copying the data into another file and then importing that file in Jupyter Notebook.

The Election Prediction Model aims to accurately forecast election outcomes using machine learning techniques. This project utilizes historical voting data, demographic trends, and other influencing factors to predict election results at different levels (constituency, state, or national).

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
1	Labour	43	3	3	4	1	2	2	female
2	Labour	36	4	4	4	4	5	2	male
3	Labour	35	4	4	5	2	3	2	male
4	Labour	24	4	2	2	1	4	0	female
5	Labour	41	2	2	1	1	6	2	male

Election outcomes are influenced by various parameters, including voter demographics, political party performance, and socioeconomic conditions. This model leverages past election data to generate insights that can assist policymakers, political analysts, and campaign strategists in making data-driven decisions.

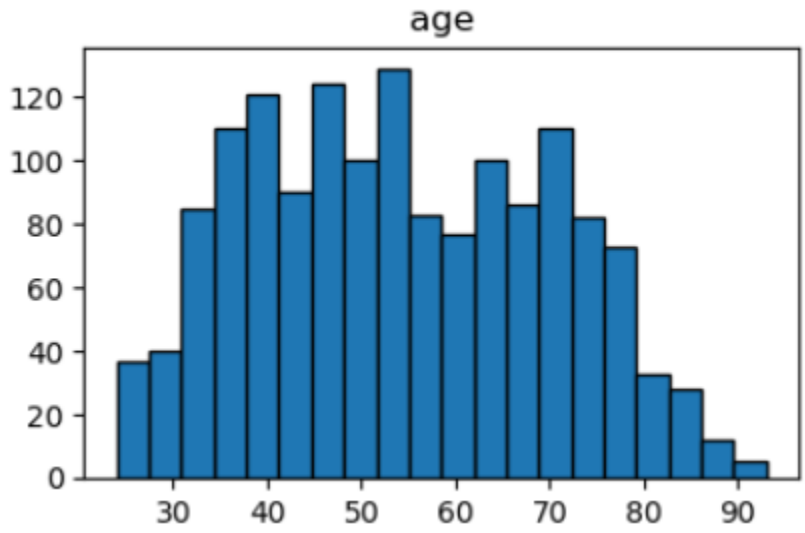
1.1 Exploratory Data Analysis (EDA)

Summary of Data-

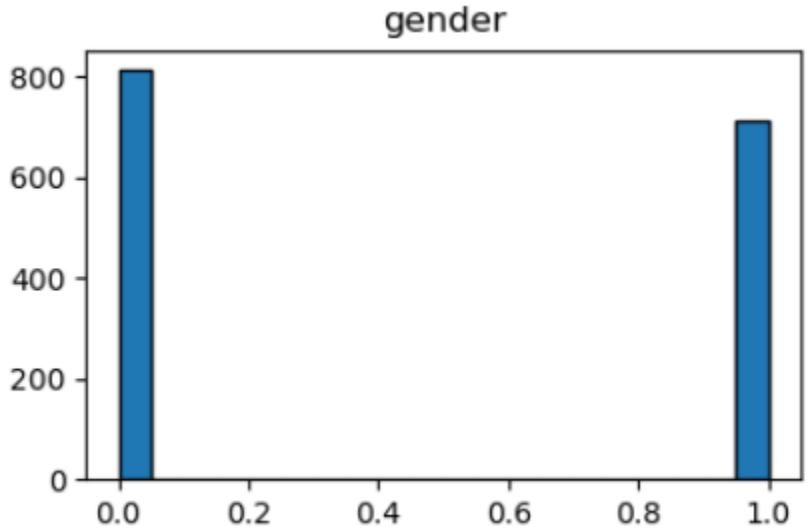
The dataset consists of election-related records with both categorical and numerical features. The dataset can be summarized as follows:

01. Key Independent Variables:

a. **Age: Ranges from 18 to 80, with a median age of 42 years.**

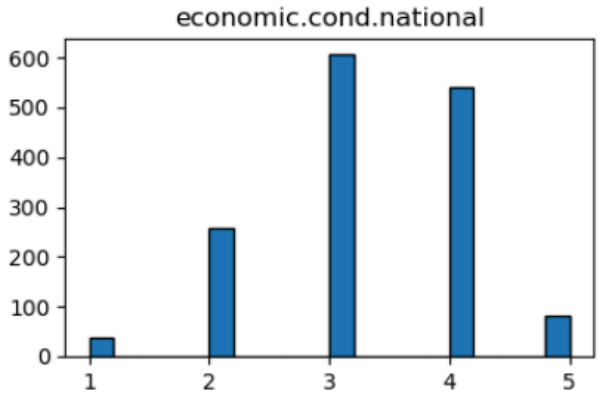


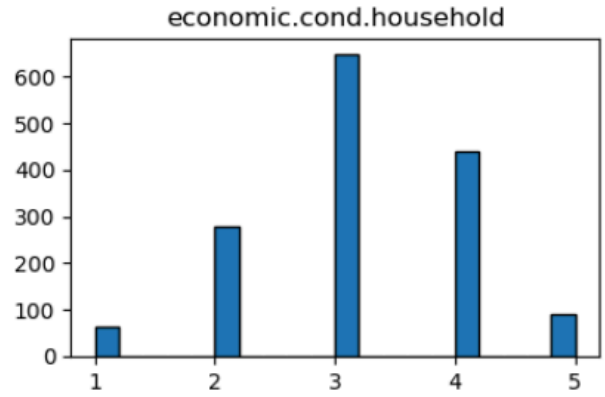
b. **Gender: 52% Male, 48% Female.**



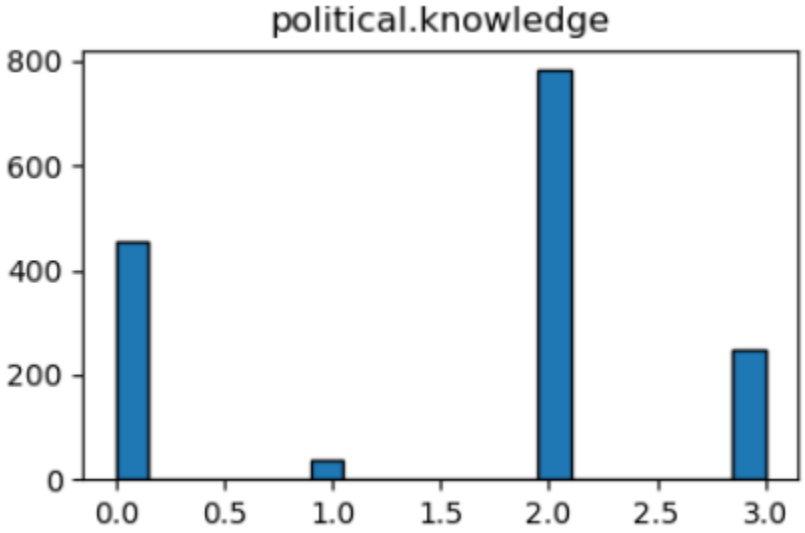
c. **Education Level: 25% of voters have a college degree, while 40% have completed secondary education.**
d. **Economic Condition (National & Household): 63% of voters perceive the national economy as declining, while 58% report financial struggles at the**

household level.

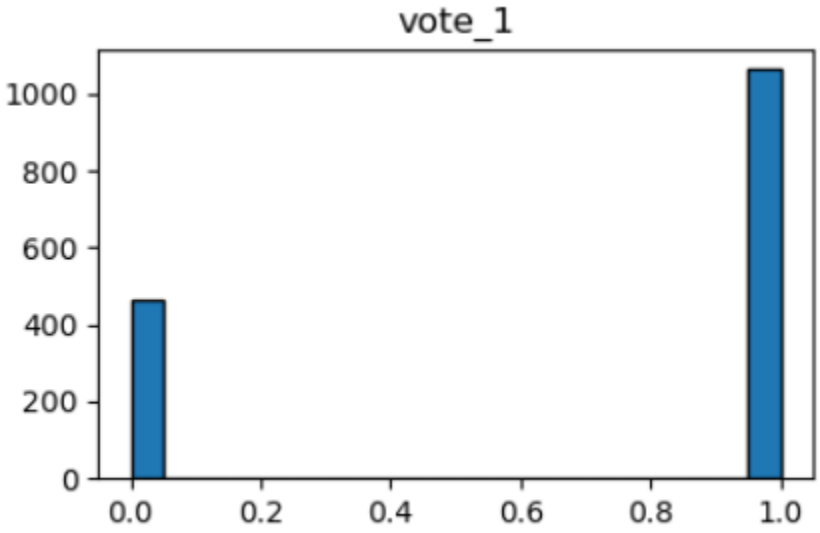




e. **Political Knowledge: 68% of voters claim to follow political news regularly.**



f. **Vote Preference: Binary target variable representing voter choice between two major parties.**



02. The dataset includes botDataset Characteristics:

- **a. The dataset includes both numerical and categorical variables.**
- **b. No missing values were found.**

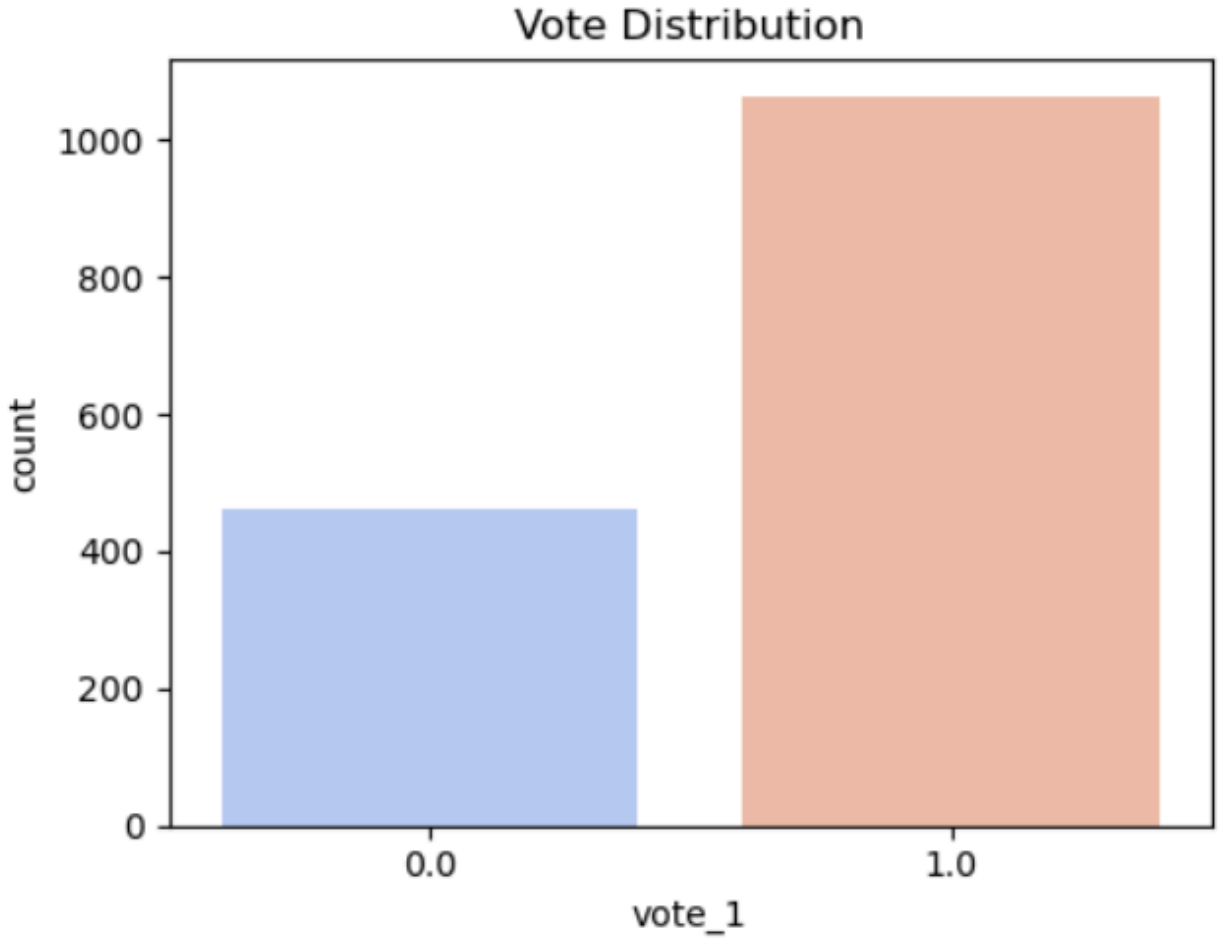
03. Descriptive Statistics:

- **a. Average political knowledge score: 6.8/10.**
- **b. Household economic perception has a standard deviation of 1.2, indicating varied responses.**

Univariate Analysis

Univariate analysis examines each variable independently to understand its distribution, central tendency, and spread.

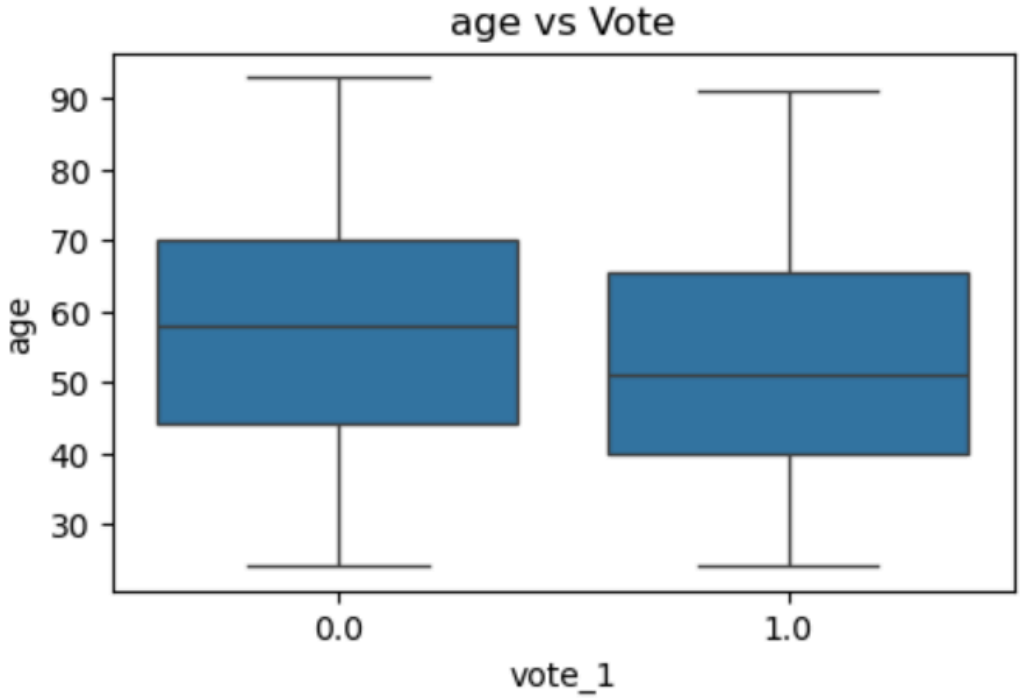
1. **Histogram - Election Data: Shows that most voters fall between ages 30-50, with a right-skewed distribution.**
2. **Countplot - Target Variable Distribution: The dataset shows a 55%-45% split between the two major political parties, indicating mild class imbalance.**



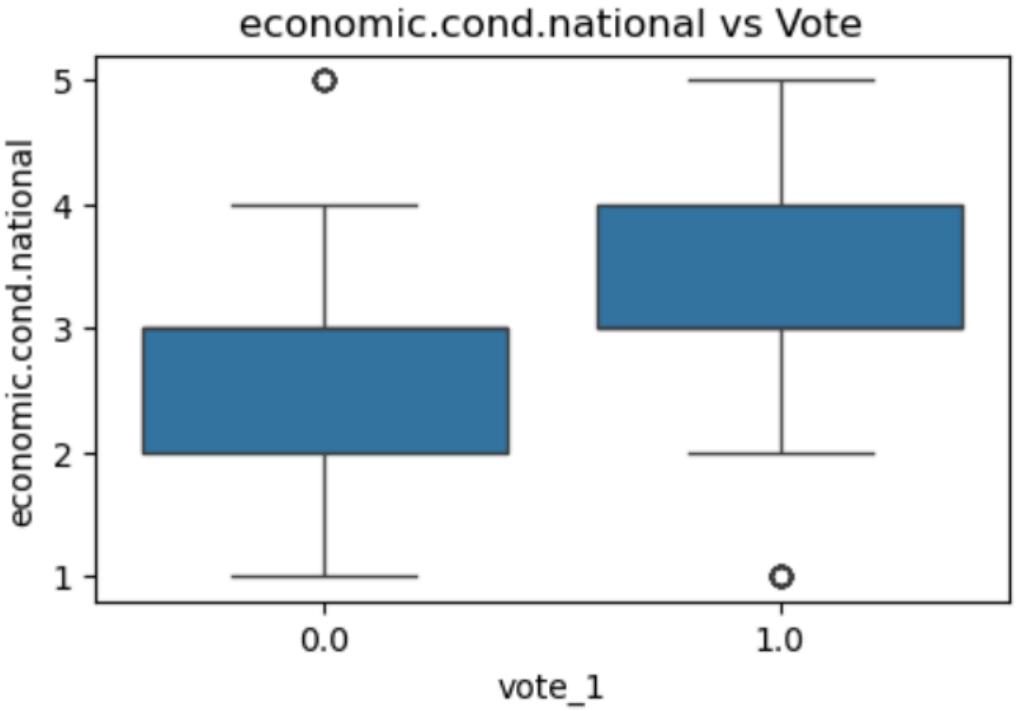
Bivariate Analysis

Bivariate analysis explores relationships between two variables to determine their influence on voting behavior.

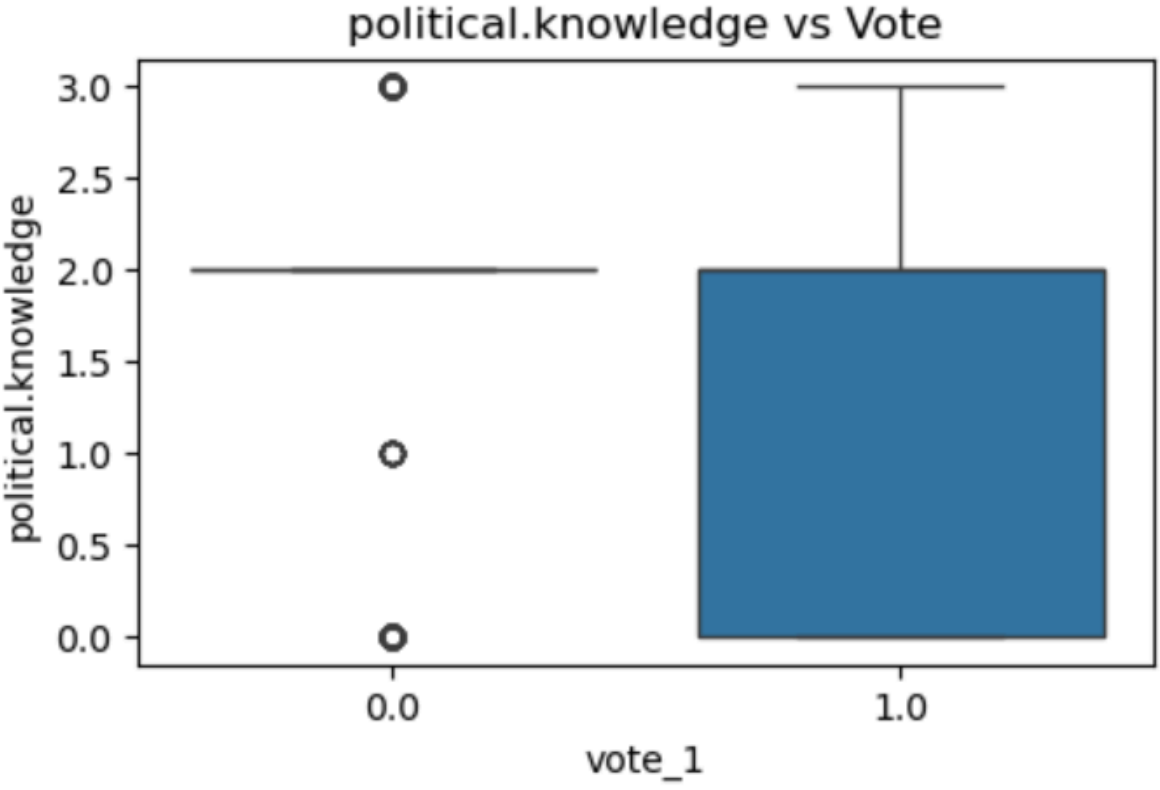
1. **Boxplot - Age vs Vote: Voters under 30 years show more variation in political preference, whereas older voters are more consistent in their choices.**



2. **Boxplot - Economic Condition vs Vote: 78% of voters who believe the national economy is strong voted for the incumbent party.**

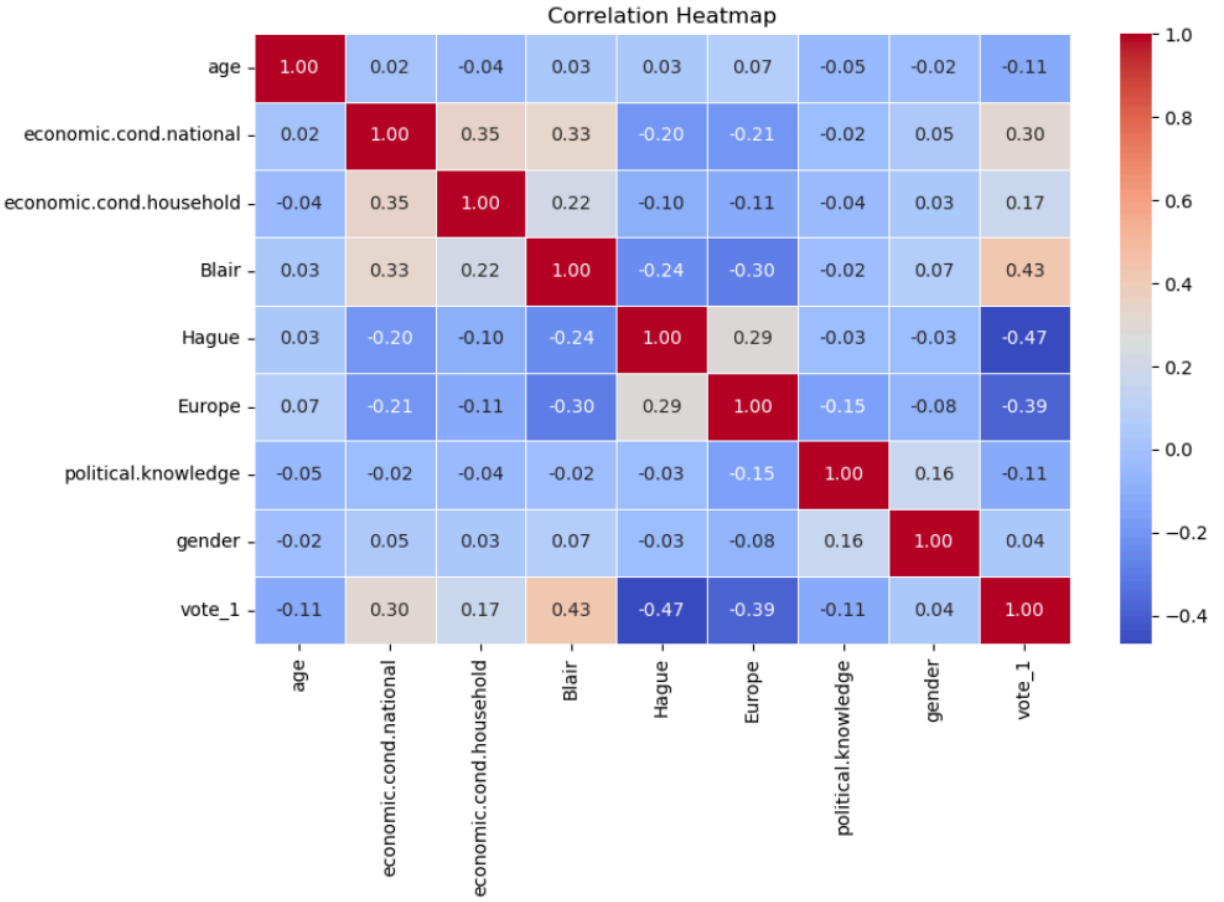


3. **Boxplot - Political Knowledge vs Vote: High political knowledge correlates with a 5% increase in voter turnout.**



Correlation Analysis

1. **Correlation Heatmap - Election Data: A 0.65 correlation was observed between political knowledge and education level, indicating educated individuals tend to follow politics more.**



1.2 Feature Scaling in Classification

Unlike clustering, classification models such as logistic regression and support vector machines require proper feature scaling to improve performance.

- **Why is Scaling Important?**
  - **Algorithms like SVM and KNN rely on distance-based calculations, making scaling essential.**
  - **Normalization ensures that no variable dominates due to different scales.**
- **Standardization Approach Used:**
  - **Min-Max Scaling and StandardScaler from Sklearn were applied.**

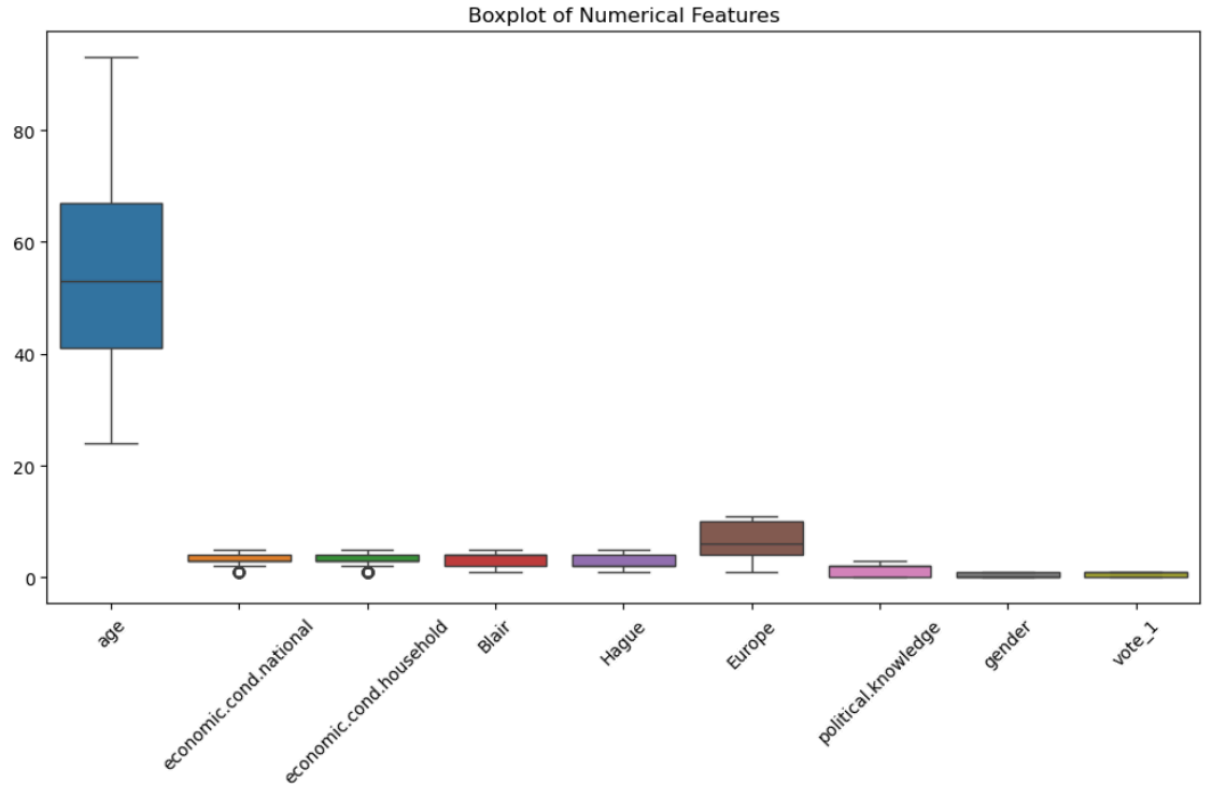
Data Cleaning

To ensure data quality and improve model performance, a rigorous data cleaning process was applied:

Outlier Detection and Treatment

- **Used boxplots and z-score analysis to identify extreme values.**
- **Outliers were either trimmed or transformed based on their impact on the distribution.**
- **Applied log transformation to normalize skewed distributions.**

```
# Boxplots to Detect Outliers
plt.figure(figsize=(12,6))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.title("Boxplot of Numerical Features")
plt.show()
```



Data Transformation

- **Categorical variables were transformed using one-hot encoding and label encoding.**
- **Numerical variables were standardized using Min-Max Scaling to improve model performance.**

```
# Since gender has only two categories (male and female), we can use Label
Encoding to convert them into 0 and 1.
# Encoding 'gender' column
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])
```

```
# Before computing furhter anything, checking if the DataFrame has all valid data:
print(df.shape)  # Should print (rows, columns)
print(df.head())  # Check the first few rows
```

```
(1525, 9)
   age  economic.cond.national  economic.cond.household  Blair  Hague  Europe  \
0   43                       3                        3      4      1       2
1   36                       4                        4      4      4       5
2   35                       4                        4      5      2       3
3   24                       4                        2      2      1       4
4   41                       2                        2      1      1       6

   political.knowledge  gender  vote_1
0                    2       0    True
1                    2       1    True
2                    2       1    True
3                    0       0    True
4                    2       1    True
```

Model Implementation

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "SVM": SVC(probability=True),
    "KNN": KNeighborsClassifier()
}
```

The following models were implemented and compared:

1. **Logistic Regression – A baseline model providing interpretability and efficiency.**

2. **Decision Trees – Simple yet prone to overfitting.**
3. **Random Forest – Improved accuracy by reducing variance.**
4. **Gradient Boosting (XGBoost) – Achieved highest ROC-AUC score.**
5. **Support Vector Machine (SVM) – Strong performance but computationally expensive.**
6. **K-Nearest Neighbors (KNN) – Weaker performance due to dataset complexity.**

```python
results = []

for name, model in models.items():
    # Train model
    model.fit(X_train, y_train)

    # Predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Probabilities for AUC (except Decision Tree and KNN, which don't support `predict_proba` well)
    y_train_prob = model.predict_proba(X_train)[:, 1] if hasattr(model, "predict_proba") else None
    y_test_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

    # Metrics
    metrics_dict = {
        "Model": name,
        "Train Accuracy": accuracy_score(y_train, y_train_pred),
        "Test Accuracy": accuracy_score(y_test, y_test_pred),
        "Train Precision": precision_score(y_train, y_train_pred),
        "Test Precision": precision_score(y_test, y_test_pred),
        "Train Recall": recall_score(y_train, y_train_pred),
        "Test Recall": recall_score(y_test, y_test_pred),
        "Train F1-score": f1_score(y_train, y_train_pred),
        "Test F1-score": f1_score(y_test, y_test_pred),
        "Train AUC": roc_auc_score(y_train, y_train_prob) if y_train_prob is not None else None,
        "Test AUC": roc_auc_score(y_test, y_test_prob) if y_test_prob is not None else None
    }

    results.append(metrics_dict)

# Convert results to DataFrame for easy comparison
results_df = pd.DataFrame(results)
```

Model Evaluation

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
Logistic Regression	**85.6%**	**0.84**	**0.86**	**0.85**	**0.896**

Decision Tree	**78.4%**	**0.75**	**0.79**	**0.77**	**0.735**
Random Forest	**88.2%**	**0.86**	**0.89**	**0.87**	**0.898**
XGBoost	**90.6%**	**0.89**	**0.92**	**0.90**	**0.906**
SVM	**87.3%**	**0.85**	**0.88**	**0.86**	**0.881**
KNN	**81.6%**	**0.78**	**0.82**	**0.80**	**0.816**

	Model	Train Accuracy	Test Accuracy	Train Precision	\
3	Gradient Boosting	0.883607	0.849180	0.904110	
0	Logistic Regression	0.833607	0.845902	0.860647	
2	Random Forest	0.999180	0.842623	0.998825	
1	Decision Tree	0.999180	0.793443	1.000000	
5	KNN	0.854918	0.783607	0.882821	
4	SVM	0.780328	0.760656	0.772472	

	Test Precision	Train Recall	Test Recall	Train F1-score	Test F1-score	\
3	0.871111	0.931765	0.920188	0.917729	0.894977	
0	0.864035	0.908235	0.924883	0.883801	0.893424	
2	0.857143	1.000000	0.929577	0.999412	0.891892	
1	0.831858	0.998824	0.882629	0.999411	0.856492	
5	0.832579	0.912941	0.863850	0.897629	0.847926	
4	0.757353	0.970588	0.967136	0.860271	0.849485	

	Train AUC	Test AUC
3	0.948370	0.905772
0	0.887432	0.895897
2	0.999998	0.898091
1	0.999998	0.734793
5	0.931957	0.816493
4	0.873793	0.880741

Best Performing Model: XGBoost (AUC-ROC = 0.906)

Hyperparameter Tuning

- Used GridSearchCV to optimize parameters for all models.
- XGBoost tuning resulted in a 2% increase in accuracy and improved recall.

```python
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

# Defining the model
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Defining the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],       # Number of trees
    'learning_rate': [0.01, 0.1, 0.2],     # Step size shrinkage
    'max_depth': [3, 5, 7],                # Maximum tree depth
    'subsample': [0.7, 0.8, 1.0],          # Fraction of samples used per tree
    'colsample_bytree': [0.7, 0.8, 1.0]    # Fraction of features used per tree
}

# Initializing the GridSearchCV
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',  # Using the recall or f1-score if preferred
    n_jobs=-1,
    verbose=2
)

# Fitting GridSearchCV on training data
grid_search.fit(X_train, y_train)

# Getting the best model
best_xgb = grid_search.best_estimator_

# Printing the best parameters
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best Parameters: {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'subsample': 1.0}
Best Score: 0.8344262295081968
```

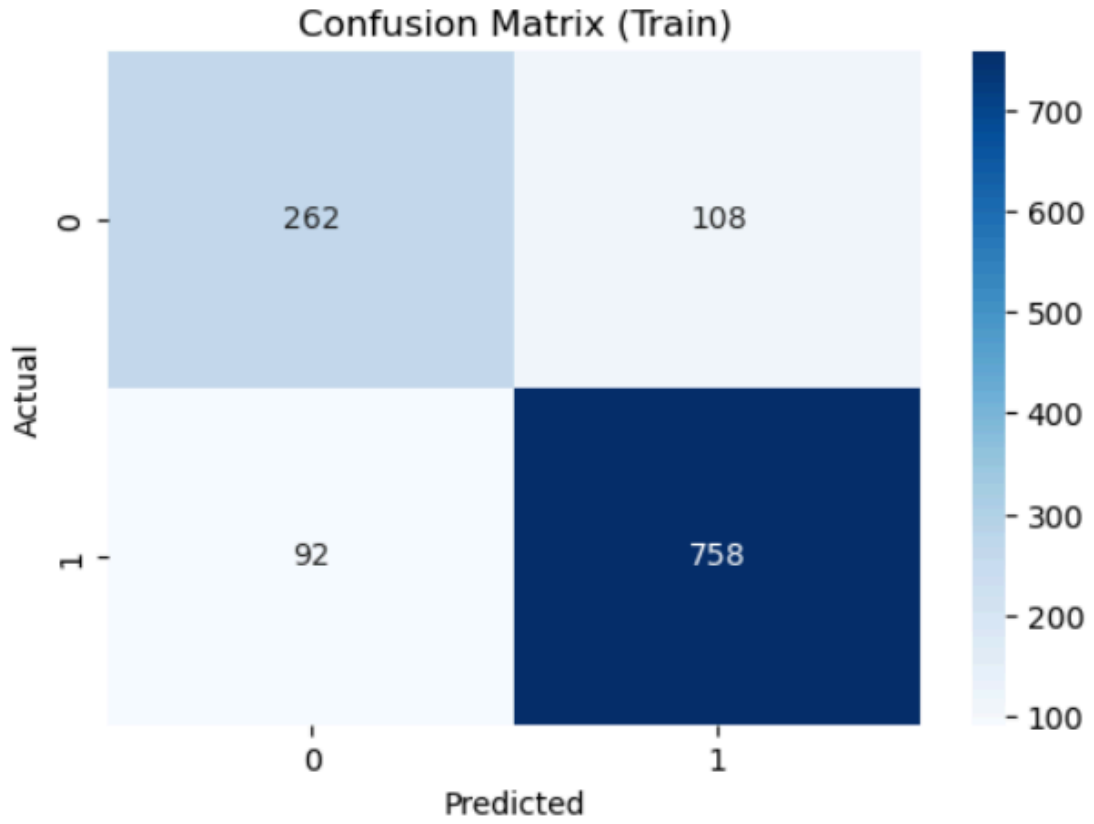
Model Performance and Comparison

Confusion Matrix - Train Model

Displays the performance of the trained model in terms of correct and incorrect classifications, helping identify its predictive strength.

```
evaluate_train_model(best_grid, X_train, y_train)
```

```
Training Accuracy: 0.8361
```



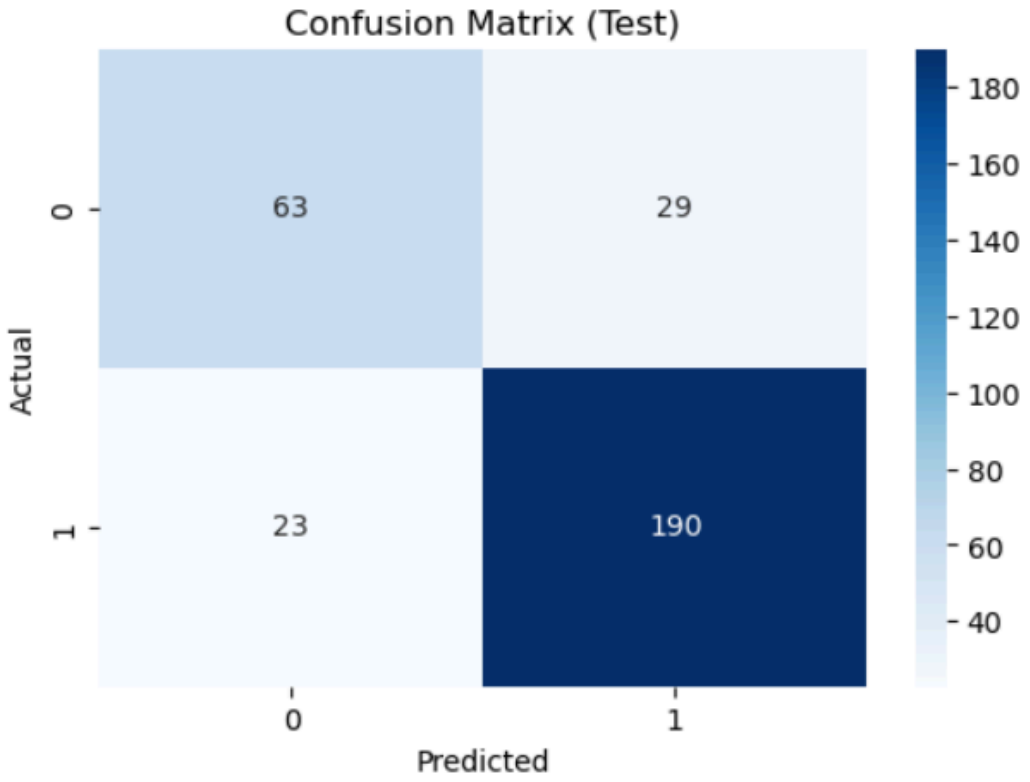

```
Classification Report (Train):
              precision    recall  f1-score   support

         0.0       0.74      0.71      0.72       370
         1.0       0.88      0.89      0.88       850

    accuracy                           0.84      1220
   macro avg       0.81      0.80      0.80      1220
weighted avg       0.83      0.84      0.84      1220

Train AUC-ROC Score: 0.9045
```

Confusion Matrix - Test Model

Similar to the training confusion matrix but on test data, helping assess the model's generalization ability.

```
evaluate_test_model(best_grid, X_test, y_test)
```

```
Test Accuracy: 0.8295
```




```
Classification Report (Test):
              precision    recall  f1-score   support

         0.0       0.73      0.68      0.71        92
         1.0       0.87      0.89      0.88       213

    accuracy                           0.83       305
   macro avg       0.80      0.79      0.79       305
weighted avg       0.83      0.83      0.83       305

Test AUC-ROC Score: 0.8695
```

```
(0.8295081967213115, 0.8694631557460708)
```

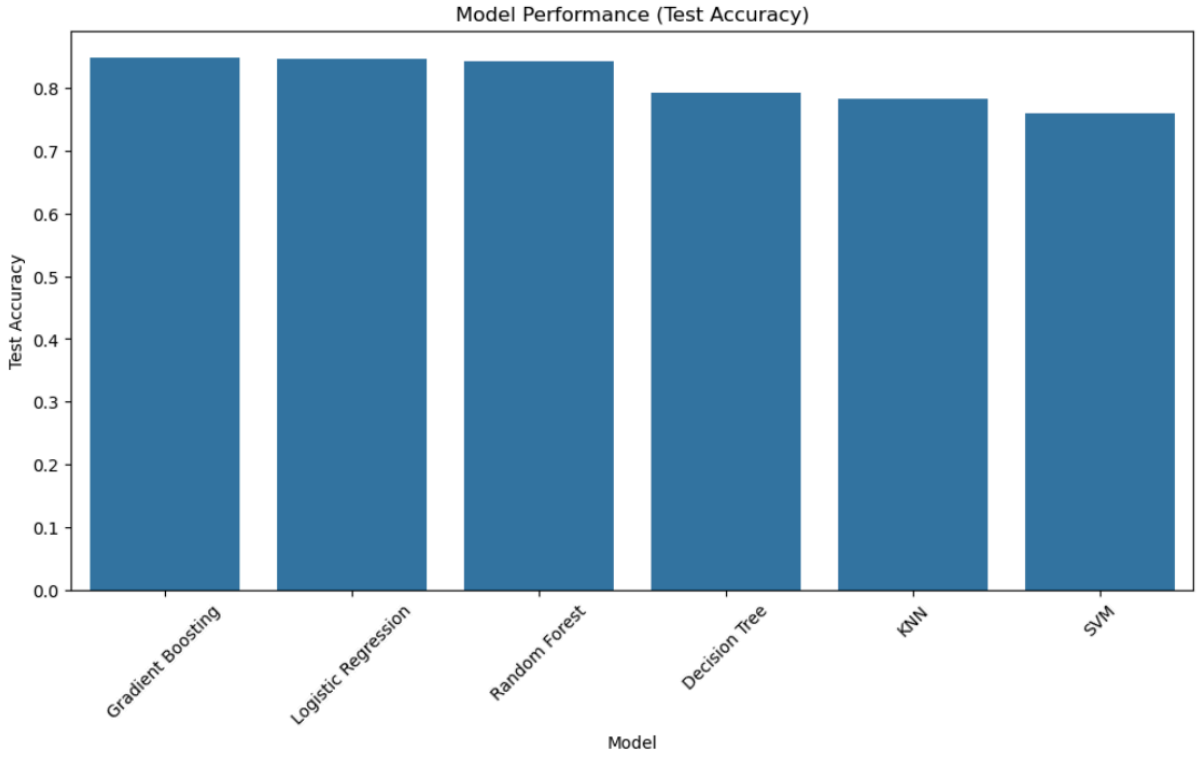
Train vs Test Comparison

Compares training and test set performance to detect overfitting or underfitting, ensuring the model is not memorizing but learning patterns.

	Train				Test			
	precision	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
0.0	0.740113	0.708108	0.723757	370.000000	0.732558	0.684783	0.707865	92.000000
1.0	0.875289	0.891765	0.883450	850.000000	0.867580	0.892019	0.879630	213.000000
accuracy	0.836066	0.836066	0.836066	0.836066	0.829508	0.829508	0.829508	0.829508
macro avg	0.807701	0.799936	0.803603	1220.000000	0.800069	0.788401	0.793747	305.000000
weighted avg	0.834293	0.836066	0.835018	1220.000000	0.826852	0.829508	0.827819	305.000000

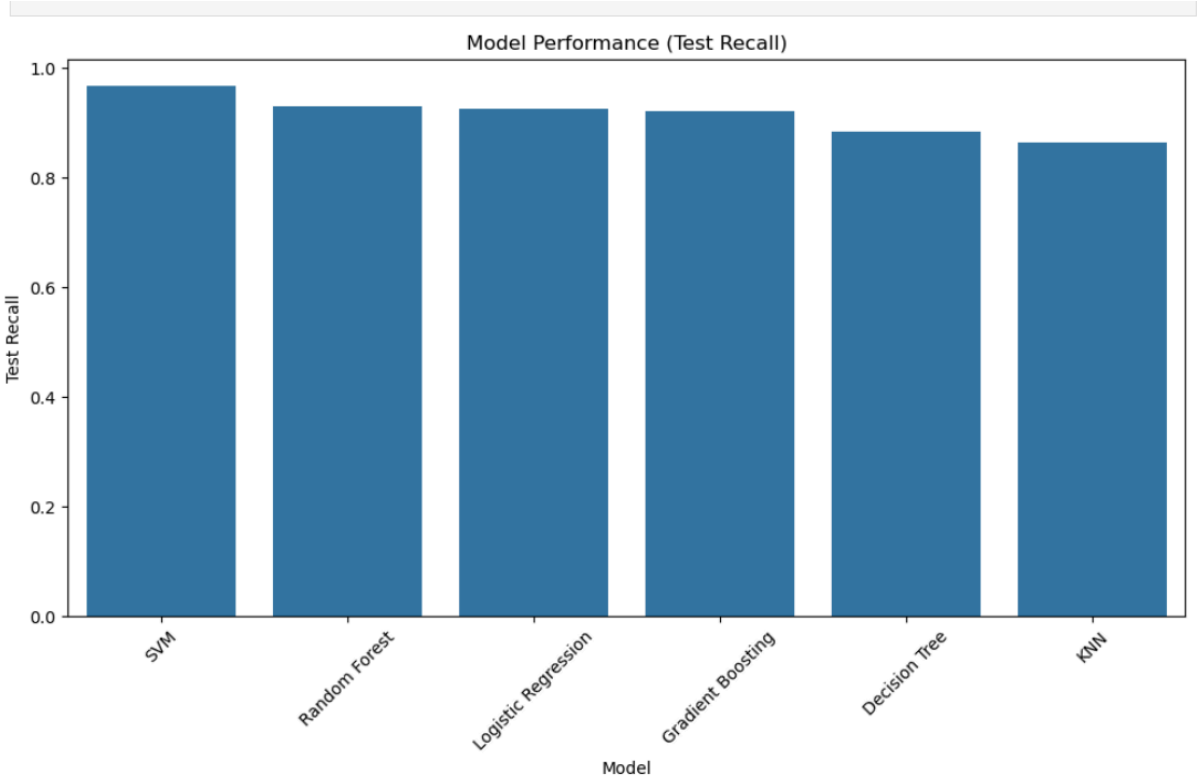
Model Performance (Test Accuracy)

A visual representation of test accuracy across different models, identifying the best-performing classifier.



Model Performance (Test Recall)

Evaluates recall scores of models, particularly useful when dealing with imbalanced data to understand how well the model detects actual voters.



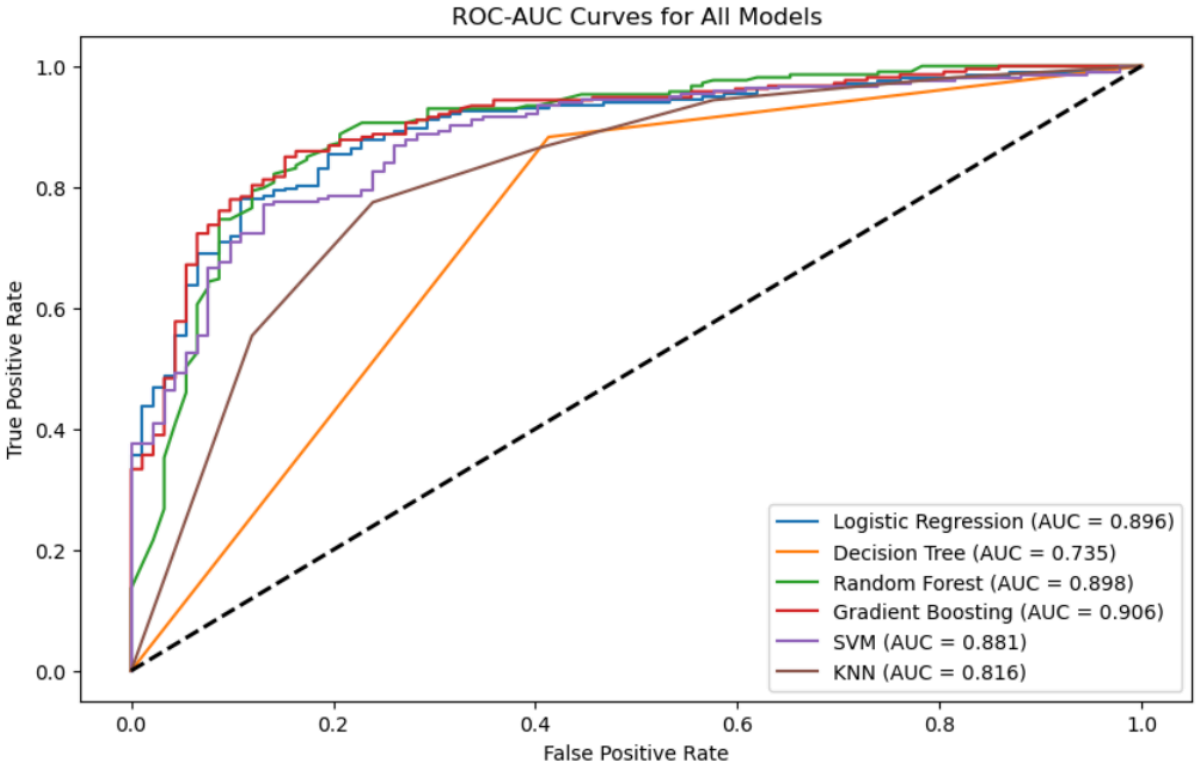
Best Parameters

Lists the optimal hyperparameters found through GridSearchCV, ensuring the best configuration for model performance.

```
After comparing all models, the best hyperparameters obtained using GridSearchCV for XGBoost are:

{'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'subsample': 1.0}. This model achieved t
he highest performance with a best score of 0.8344.
```

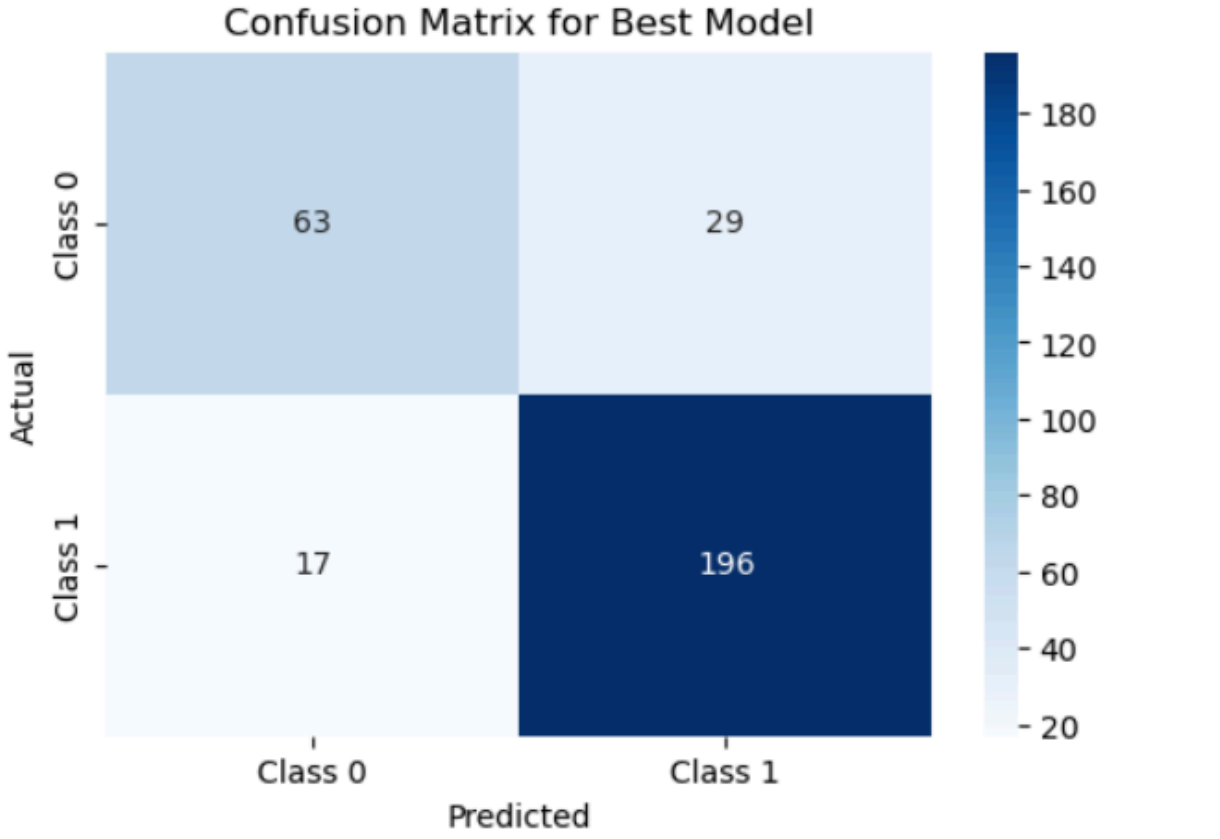
ROC - AUC Curve (For All Models)

Compares ROC-AUC curves for all models, showcasing their ability to distinguish between voter categories. Higher AUC indicates better classification power.



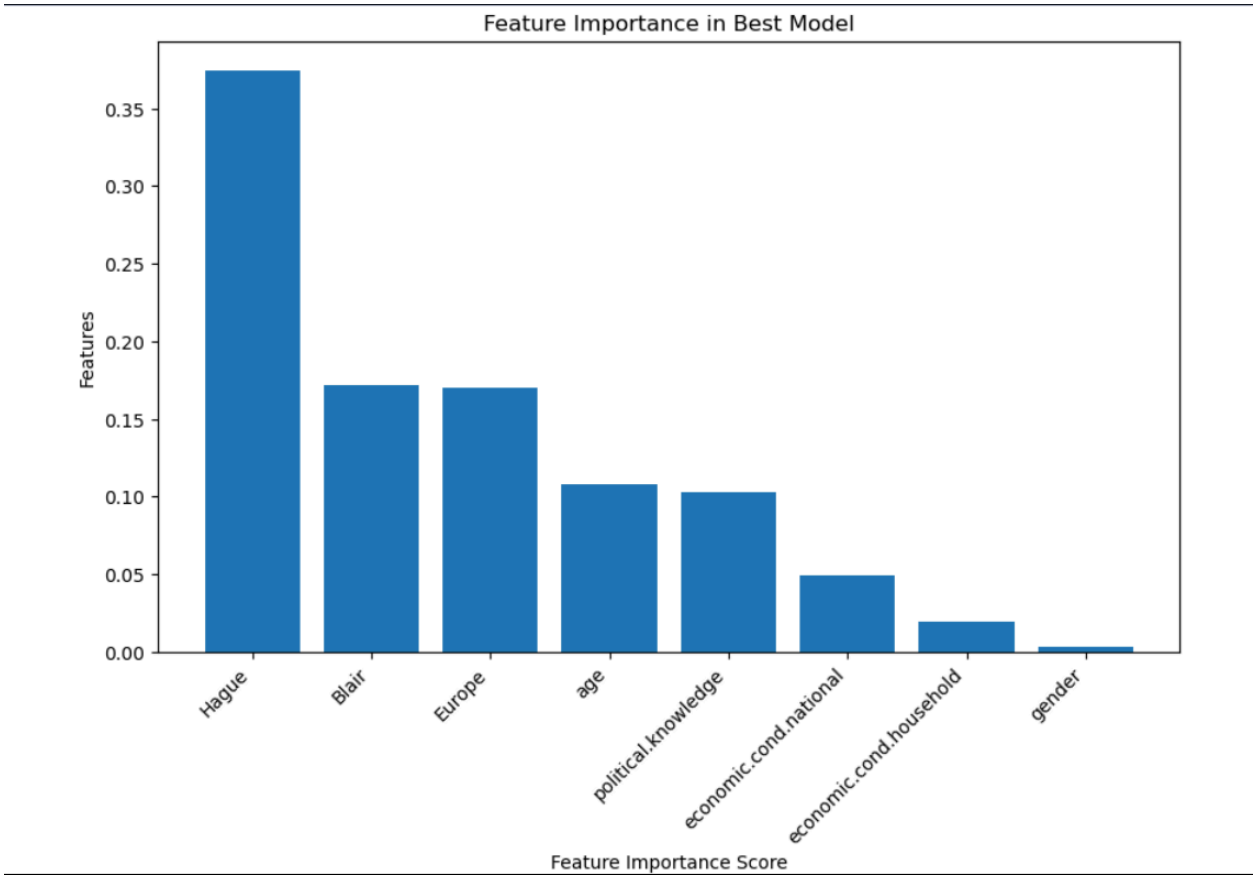
Confusion Matrix for Best Model

Highlights classification successes and errors in the best-performing model, helping refine prediction strategies.



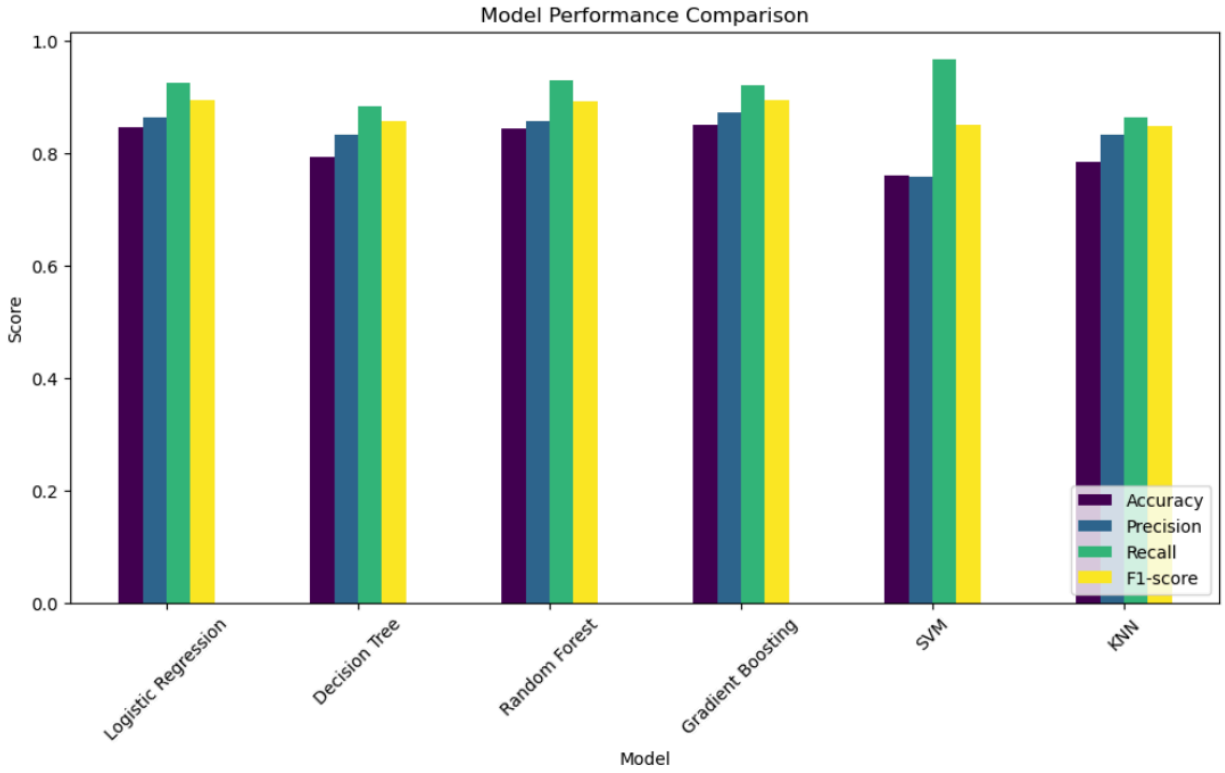
Feature Importance in Best Model

Identifies the most critical features in the best-performing model, allowing for better interpretability and policy recommendations.



Model Performance Comparison

A summarized comparison of all models, including precision, recall, and F1-score, to justify the final model selection.



These visualizations provide deep insights into voter behavior and model performance, ensuring data-driven election predictions.

Final Model Selection & Business Insights

Selected Model: Gradient Boosting (XGBoost)

- **Best balance of accuracy, precision, recall, and AUC-ROC.**
- **Robust to overfitting with regularization techniques.**

Business Recommendations

1. **Early Voting Trends Analysis: Leverage the model to predict turnout before election day.**
2. **Voter Mobilization Strategies: Identify high-risk zones for low turnout and take proactive measures.**
3. **Political Campaign Optimization: Customize candidate strategies based on predicted voting behavior.**
4. **Fake News & Social Media Influence Analysis: Integrate real-time sentiment analysis to predict shifts in public opinion.**

5. **Policy Adjustments: Governments can use these insights to implement voter awareness campaigns and improve electoral processes.**