**UCS2612 - Machine Learning Lab**

**Mini Project**

**Project Title: Predicting the Candidates vote in Indian general Election**

Team Members:

Anandh K 3122 21 5001 009

Tejaswi Kakarla 3122 21 5001 039

Karthikeyan A 3122 21 5001 041

Mega V 3122 21 5001 051

Class: CSE

Section: A

Year: III

**Introduction**

Indian general elections are among the largest democratic exercises in the world, where billion of eligible voters participate in selecting representatives for the Lok Sabha, the lower house of India's Parliament. Governed by the Election Commission of India, these elections occur every five years. The electoral machinery, comprising polling booths, voter registration drives, and ballot counting procedures, orchestrates the voices of billions of voters, ensuring their collective will is translated into governance.

Opinion polls in Indian elections serve as barometers of public sentiment, providing valuable insights into voter preferences and potential electoral outcomes. Conducted by various research organizations and media outlets, these polls employ sampling techniques to gauge the pulse of the electorate. While opinion polls offer predictive insights and shape campaign strategies, they also face challenges such as sampling errors and methodological limitations. Nonetheless, they play a pivotal role in fostering democratic engagement and informing voters about prevailing trends, thereby enriching the electoral discourse in the world's largest democracy.

Our goal to predict the opinion poll results using the existing election results history with machine learning Models

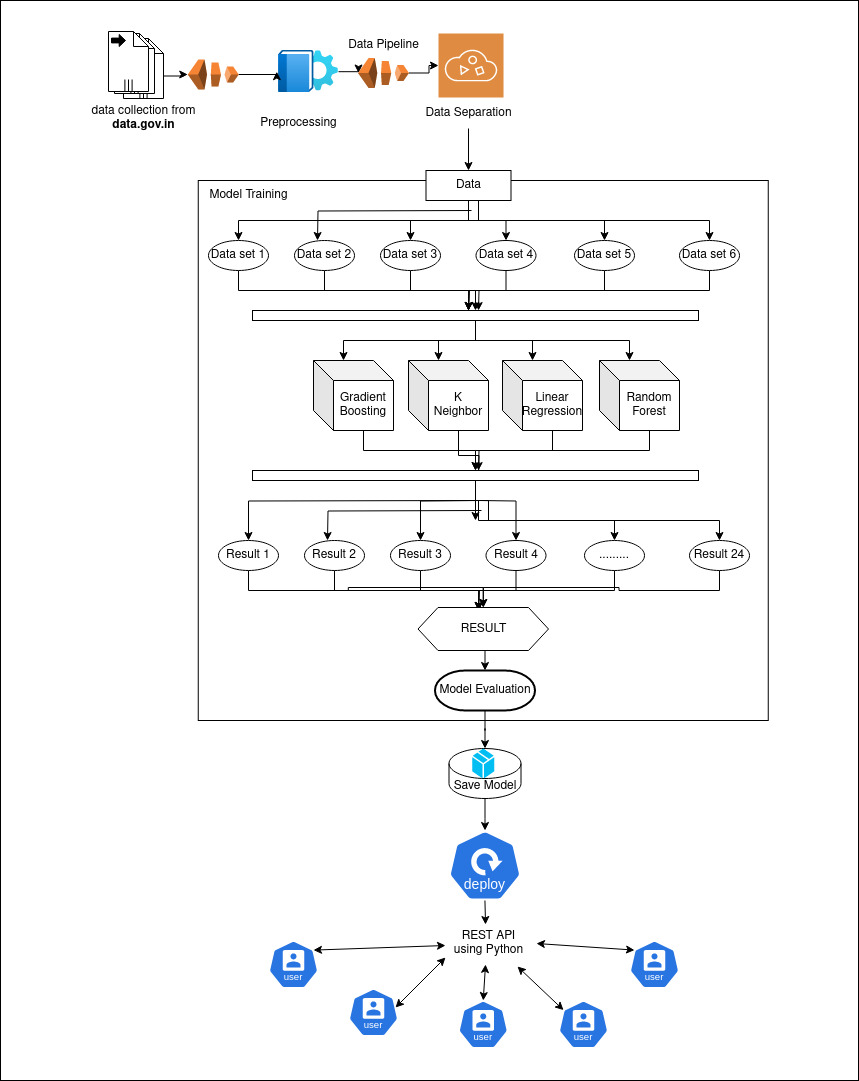
**Problem Statement**

Now a days, opinion polls often employ random sampling techniques to gather data from a representative sample of the population. In the context of telephone-based polling, researchers choose only limited number of phone numbers randomly. This ensures that every phone number in the target population has an equal chance of being selected for the survey. Once a phone number is dialed, interviewers conduct the survey by asking questions about voter preferences, opinions on political issues, and other relevant topics. But that count is nearly less than one percent in our population. But this itself is a very complicated process for our massive population. So, reduce the man power and improve the accuracy of opinion poll results we are trying to build a machine learning models.

**Development Environment**

|  |  |
| --- | --- |
| IDE / Editor | Jupyter Notebook, VS Code |
| Programming Language | Python |
| Libraries/Frameworks | scikit-learn |
| Data Exploration Tools | Pandas, Matplotlib, Numpy |
| Model Evaluation Tools | scikit-learn metrics |
| Deployment Tool | Assure |

**System Architecture Diagram**



**Dataset Collection**

Official Website for Data Collection:

<https://data.gov.in/catalog/statistical-hand-book-2019-legislature-and-election>

<https://www.kaggle.com/>

Unofficial Website for Data Collection:

<https://www.indiavotes.com/>

<https://data.opencity.in/dataset/tamil-nadu-assembly-elections-2021>

**Implementation**

Importing Necessary Libraries:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.feature\_selection import SelectKBest, f\_regression

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.preprocessing import MinMaxScaler

from sklearn.neighbors import KNeighborsRegressor

from sklearn.model\_selection import learning\_curve

import numpy as np

Loading the dataset:

df=pd.read\_csv("indian-national-level-election.csv")

Data Collection:

1. Display the Shape of dataset

print("The Shape of the Data set : ",df.shape)



1. Display the Attributes and datatypes

print("The Data Types of The Attributes are\n\n",df.dtypes)

A screenshot of a computer

Description automatically generated

1. Display the State Wise Vote Percentage in Each Election

grouped\_data = df.groupby('year')

for year, group in grouped\_data:

    total\_votes\_year = group['totvotpoll'].sum()

    group['vote\_percentage'] = (group['totvotpoll'] / total\_votes\_year) \* 100

    plt.figure(figsize=(6, 4))

    plt.bar(group['st\_name'], group['vote\_percentage'], color='skyblue')

    plt.title(f'Total Vote Percentage by State in {year}')

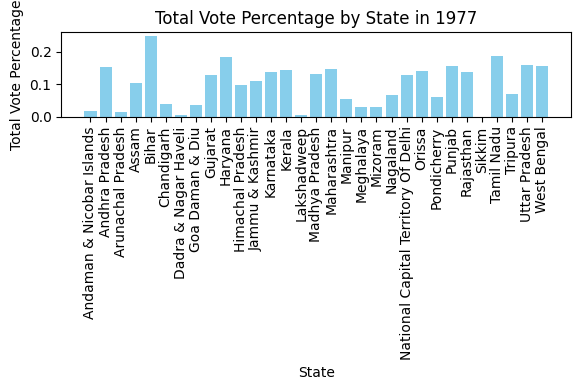
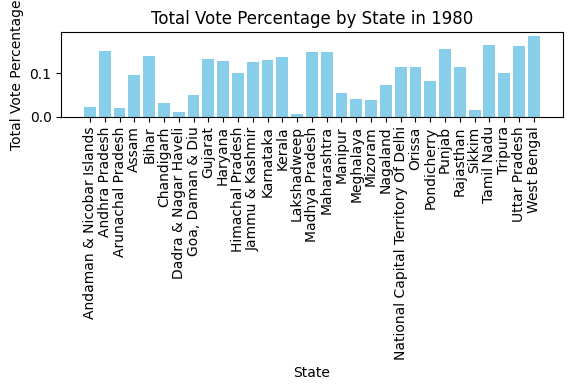
    plt.xlabel('State')

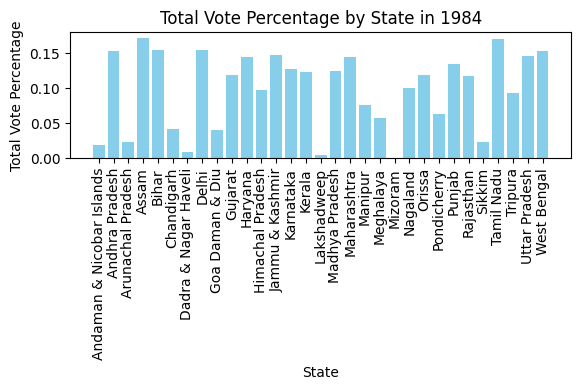
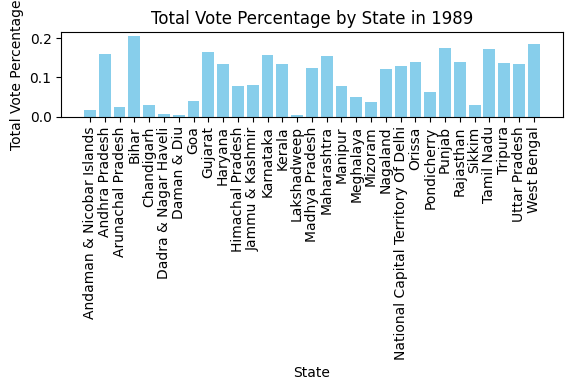
    plt.ylabel('Total Vote Percentage')

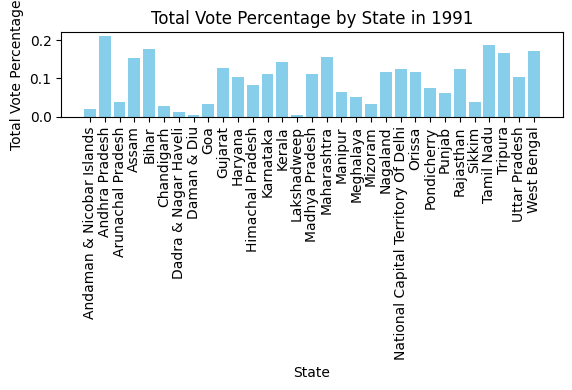
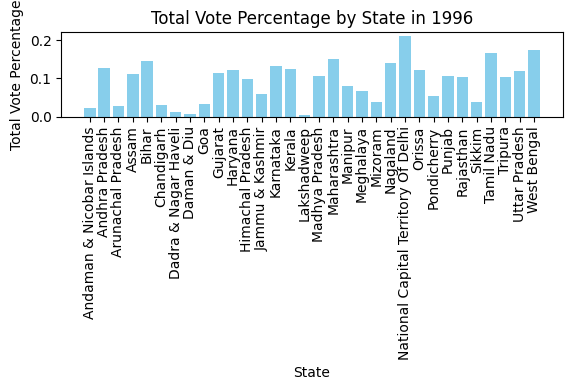
    plt.xticks(rotation=90)

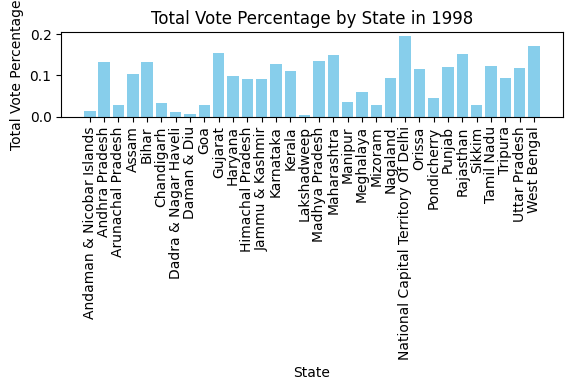
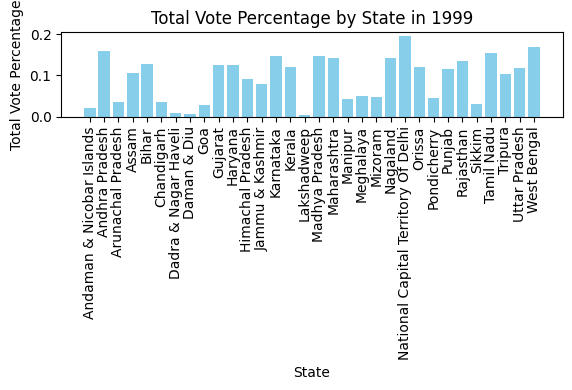
    plt.tight\_layout()

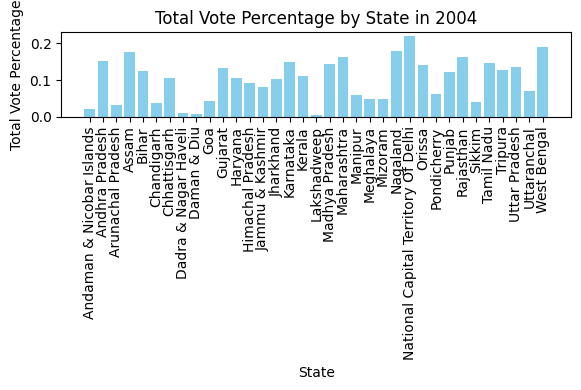
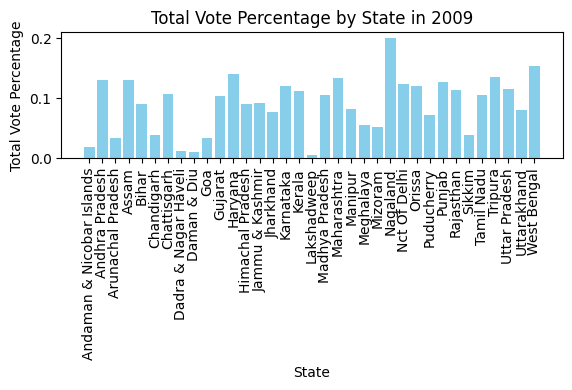
    plt.show()

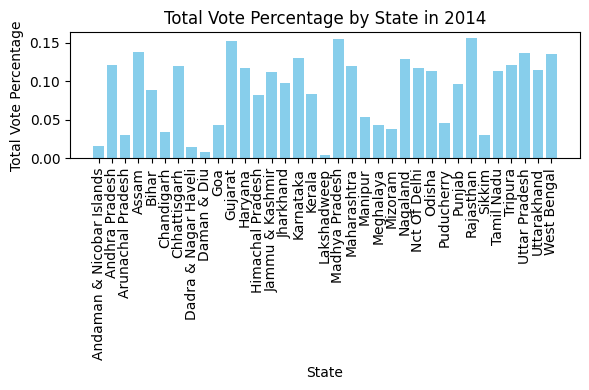
 



1. Display the Each Parliament Constitution Victoried Person in Each election

max\_votes\_index = df.groupby(['year', 'pc\_name'])['totvotpoll'].idxmax()

max\_votes\_parties = df.loc[max\_votes\_index, ['year', 'pc\_name', 'partyabbre', 'totvotpoll']]

print(max\_votes\_parties)

A computer screen with white text

Description automatically generated

1. Display the Election Results in Each year

party\_counts = max\_votes\_parties.groupby(['year', 'partyabbre']).size()

party\_counts = party\_counts.reset\_index(name='count')

print(party\_counts)

Year = int(input("Enter The Year : "))

year\_counts = party\_counts[party\_counts['year'] == Year]

sorted\_party\_counts = year\_counts.sort\_values(by='count', ascending=False)

print(sorted\_party\_counts)

A black and white screen with numbers

Description automatically generated

Year = int(input("Enter The Year : "))

year\_counts = party\_counts[party\_counts['year'] == Year]

sorted\_party\_counts = year\_counts.sort\_values(by='count', ascending=False)

print(sorted\_party\_counts)

A black screen with white text

Description automatically generated

1. Display the heatmap

df=df.drop(columns=['pc\_name','pc\_type','cand\_name','partyname'])

print("After removing the Unnecessary Attributes The Remaining Attributes in The Dataset are\n\n",df.dtypes)

label\_encoder = LabelEncoder()

df['st\_name'] = label\_encoder.fit\_transform(df['st\_name'])

df['partyabbre'] = label\_encoder.fit\_transform(df['partyabbre'])

df['cand\_sex'] = label\_encoder.fit\_transform(df['cand\_sex'])

print(df.head())

import seaborn as sns

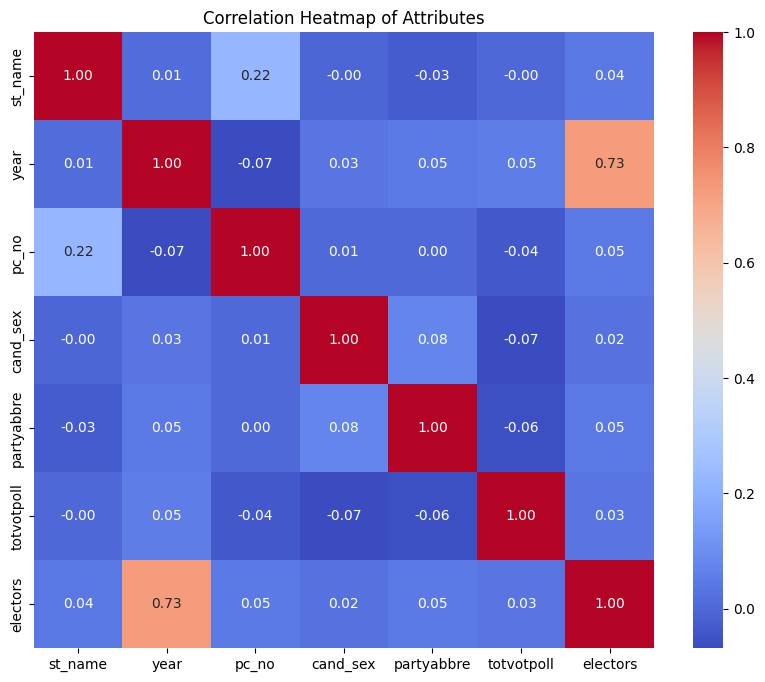
correlation\_matrix = df.corr()

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Heatmap of Attributes')

plt.show()



1. Display the Correlation Graph between Each Attributes

import networkx as nx

attributes\_to\_scale = df.columns.tolist()

G = nx.Graph()

for i, attribute1 in enumerate(attributes\_to\_scale):

    for j, attribute2 in enumerate(attributes\_to\_scale):

        if i != j:

            correlation = df[attribute1].corr(df[attribute2])

            G.add\_edge(attribute1, attribute2, weight=correlation)

plt.figure(figsize=(12, 10))

pos = nx.spring\_layout(G)

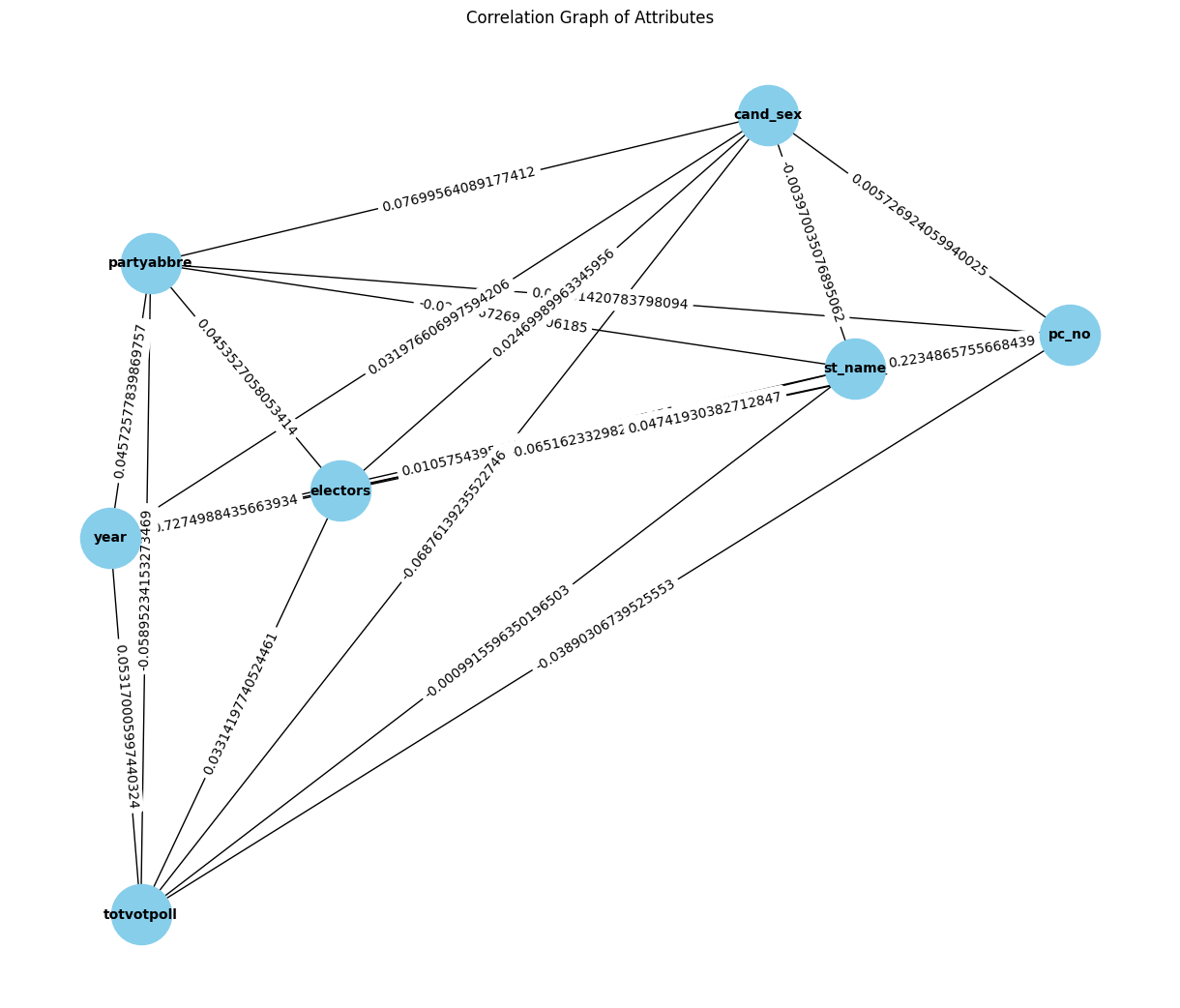
nx.draw(G, pos, with\_labels=True, node\_size=2000, node\_color='skyblue', font\_size=10, font\_weight='bold')

edge\_labels = nx.get\_edge\_attributes(G, 'weight')

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels)

plt.title('Correlation Graph of Attributes')

plt.show()



1. Correlation between Attributes and Label

attribute\_label\_correlations = df.corrwith(df['totvotpoll'])

plt.figure(figsize=(10, 6))

attribute\_label\_correlations.plot(kind='bar', color='skyblue')

plt.title('Correlation between Attributes and Label')

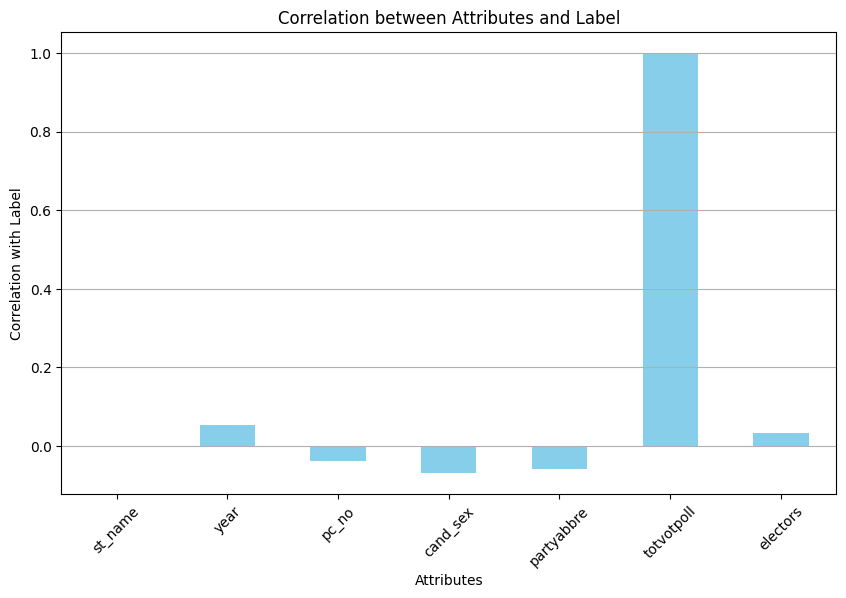
plt.xlabel('Attributes')

plt.ylabel('Correlation with Label')

plt.xticks(rotation=45)

plt.grid(axis='y')

plt.show()



Data Preprocessing:

1. Handling Missing Values in the Dataset

print("The Missing Values in the Dataset\n\n",df.isnull().sum())

most\_frequent\_train\_gender = df['cand\_sex'].mode()[0]

df['cand\_sex'] = df['cand\_sex'].fillna(most\_frequent\_train\_gender)

print("The Missing Values in the Dataset\n\n",df.isnull().sum())

A black screen with white text

Description automatically generated

1. Converting the categorical values into Numerical Values

label\_encoder = LabelEncoder()

#df['st\_name'] = label\_encoder.fit\_transform(df['st\_name'])

#df['partyabbre'] = label\_encoder.fit\_transform(df['partyabbre'])

#df['cand\_sex'] = label\_encoder.fit\_transform(df['cand\_sex'])

print(df.head())

A black and white screen with white text

Description automatically generated

1. Normalization

attributes\_to\_scale = df.columns.tolist()

attributes\_per\_line = 3

total\_rows = (len(attributes\_to\_scale) + attributes\_per\_line - 1) // attributes\_per\_line

for row in range(total\_rows):

    plt.figure(figsize=(18, 6))

    start\_index = row \* attributes\_per\_line

    end\_index = min((row + 1) \* attributes\_per\_line, len(attributes\_to\_scale))

    for idx, attribute in enumerate(attributes\_to\_scale[start\_index:end\_index], start=1):

        plt.subplot(1, attributes\_per\_line, idx)

        df.boxplot(column=attribute)

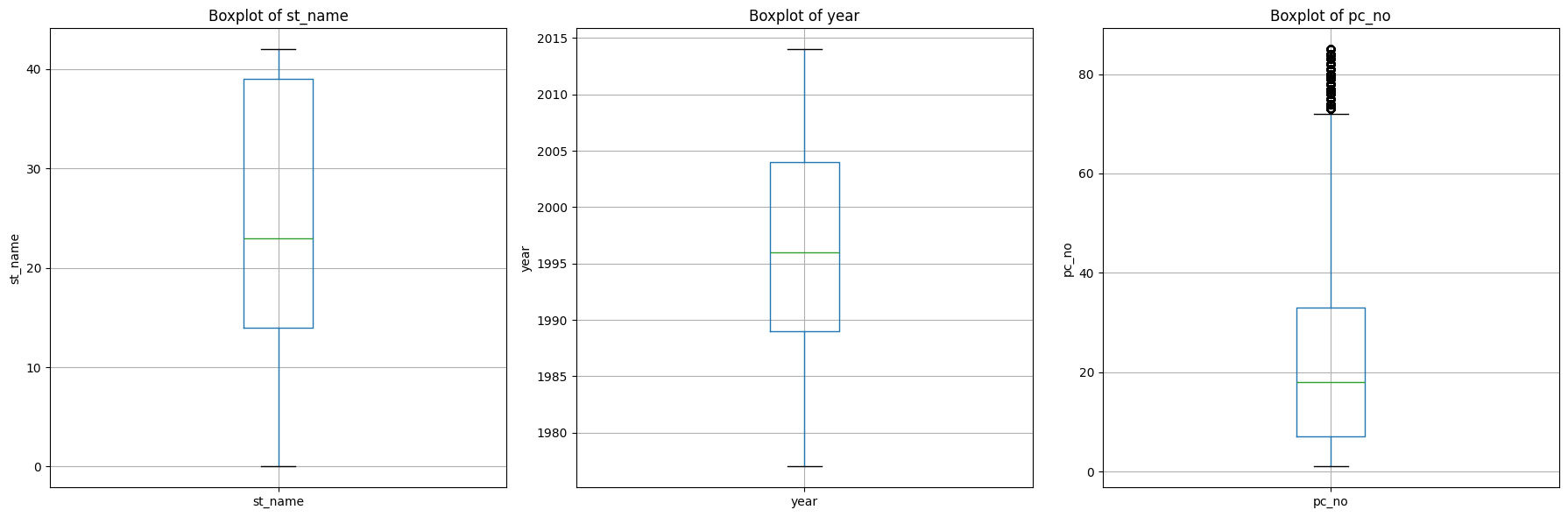
        plt.title(f'Boxplot of {attribute}')

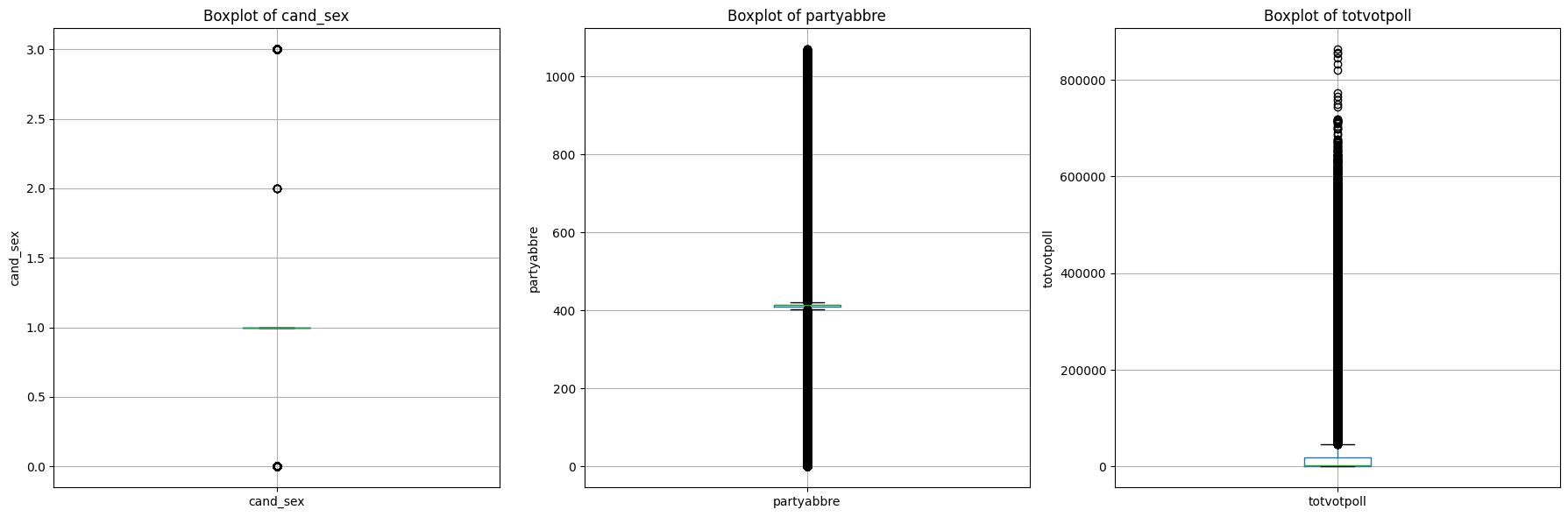
        plt.ylabel(attribute)

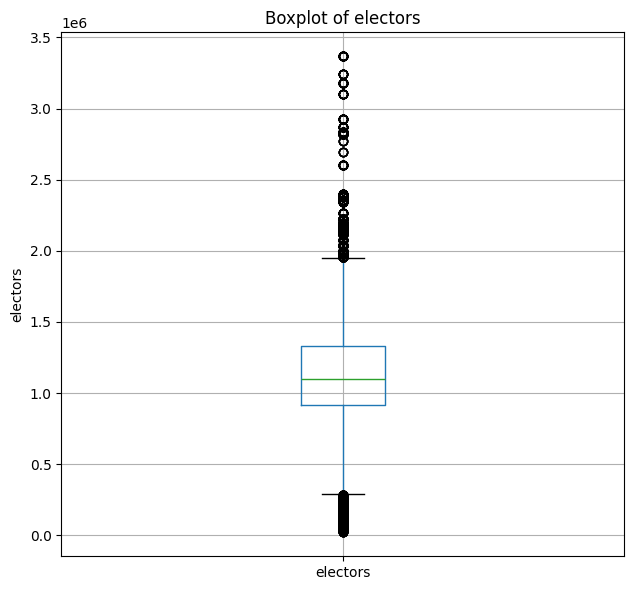
        plt.grid(True)

    plt.tight\_layout()

plt.show()







attributes\_to\_scale = [ 'electors']

min\_max\_scaler = MinMaxScaler()

data\_normalized = df.copy()

df[attributes\_to\_scale] = min\_max\_scaler.fit\_transform(data\_normalized[attributes\_to\_scale])

Feature Engineering:

1. Remove the Unnecessary data from dataset:

#df=df.drop(columns=['pc\_name','pc\_type','cand\_name','partyname'])

print("After removing the Unnecessary Attributes, The Remaining Attributes in The Dataset are\n\n",df.dtypes)

test=[]

prediction=[]

A screen shot of a computer

Description automatically generated

1. Select K best with K=3,4,5,6:

Gradient Boosting Regression Model1:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=3)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_gb = GradientBoostingRegressor()

model\_gb.fit(X\_train, y\_train)

y\_pred\_gb = model\_gb.predict(X\_test)

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r\_squared\_gb = r2\_score(y\_test, y\_pred\_gb)

test.append(y\_test)

prediction.append(y\_pred\_gb)

print("Gradient Boosting Mean Squared Error  : ", round(mse\_gb, 2))

print("Gradient Boosting R-squared Value     : ", r\_squared\_gb)

print("Gradient Boosting Accuracy Percentage : ", round(100 \* r\_squared\_gb, 2), "%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model\_gb, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

Gradient Boosting Mean Squared Error : 4689954366.56

Gradient Boosting R-squared Value : 0.5724505551346817

Gradient Boosting Accuracy Percentage : 57.25 %

Gradient Boosting Regression Model2:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=4)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_gb = GradientBoostingRegressor()

model\_gb.fit(X\_train, y\_train)

y\_pred\_gb = model\_gb.predict(X\_test)

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r\_squared\_gb = r2\_score(y\_test, y\_pred\_gb)

test.append(y\_test)

prediction.append(y\_pred\_gb)

print("Gradient Boosting Mean Squared Error  : ", round(mse\_gb, 2))

print("Gradient Boosting R-squared Value     : ", r\_squared\_gb)

print("Gradient Boosting Accuracy Percentage : ", round(100 \* r\_squared\_gb, 2), "%")

Gradient Boosting Mean Squared Error : 4691383891.1

Gradient Boosting R-squared Value : 0.5723202356526427

Gradient Boosting Accuracy Percentage : 57.23 %

Gradient Boosting Regression Model3:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_gb = GradientBoostingRegressor()

model\_gb.fit(X\_train, y\_train)

y\_pred\_gb = model\_gb.predict(X\_test)

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r\_squared\_gb = r2\_score(y\_test, y\_pred\_gb)

test.append(y\_test)

prediction.append(y\_pred\_gb)

print("Gradient Boosting Mean Squared Error  : ", round(mse\_gb, 2))

print("Gradient Boosting R-squared Value     : ", r\_squared\_gb)

print("Gradient Boosting Accuracy Percentage : ", round(100 \* r\_squared\_gb, 2), "%")

Gradient Boosting Mean Squared Error : 4651468408.58

Gradient Boosting R-squared Value : 0.5759590434232739

Gradient Boosting Accuracy Percentage : 57.6 %

Gradient Boosting Regression Model4:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_gb = GradientBoostingRegressor()

model\_gb.fit(X\_train, y\_train)

y\_pred\_gb = model\_gb.predict(X\_test)

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r\_squared\_gb = r2\_score(y\_test, y\_pred\_gb)

test.append(y\_test)

prediction.append(y\_pred\_gb)

print("Gradient Boosting Mean Squared Error  : ", round(mse\_gb, 2))

print("Gradient Boosting R-squared Value     : ", r\_squared\_gb)

print("Gradient Boosting Accuracy Percentage : ", round(100 \* r\_squared\_gb, 2), "%")

Gradient Boosting Mean Squared Error : 4651567699.12

Gradient Boosting R-squared Value : 0.5759499918180762

Gradient Boosting Accuracy Percentage : 57.59 %

Gradient Boosting Regression Model5 using PCA:

pca = PCA(n\_components=5)

pca.fit(x)

x\_pca = pca.transform(x)

print("Selected Features (Principal Components):")

print(x\_pca.shape)

x = pd.DataFrame(x\_pca)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model\_gb = GradientBoostingRegressor()

model\_gb.fit(X\_train, y\_train)

y\_pred\_gb = model\_gb.predict(X\_test)

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r\_squared\_gb = r2\_score(y\_test, y\_pred\_gb)

test.append(y\_test)

prediction.append(y\_pred\_gb)

print("Gradient Boosting Mean Squared Error  : ", round(mse\_gb, 2))

print("Gradient Boosting R-squared Value     : ", r\_squared\_gb)

print("Gradient Boosting Accuracy Percentage : ", round(100 \* r\_squared\_gb, 2), "%")

Gradient Boosting Mean Squared Error : 4491402030.83

Gradient Boosting R-squared Value : 0.590551145094216

Gradient Boosting Accuracy Percentage : 59.06 %

Gradient Boosting Regression Model6 Using LDA:

lda = LinearDiscriminantAnalysis(n\_components=5)

lda.fit(x, y)

x\_lda = lda.transform(x)

print("Selected Features (Linear Discriminant Components):")

print(x\_lda.shape)

x = pd.DataFrame(x\_lda)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model\_gb = GradientBoostingRegressor()

model\_gb.fit(X\_train, y\_train)

y\_pred\_gb = model\_gb.predict(X\_test)

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r\_squared\_gb = r2\_score(y\_test, y\_pred\_gb)

test.append(y\_test)

prediction.append(y\_pred\_gb)

print("Gradient Boosting Mean Squared Error  : ", round(mse\_gb, 2))

print("Gradient Boosting R-squared Value     : ", r\_squared\_gb)

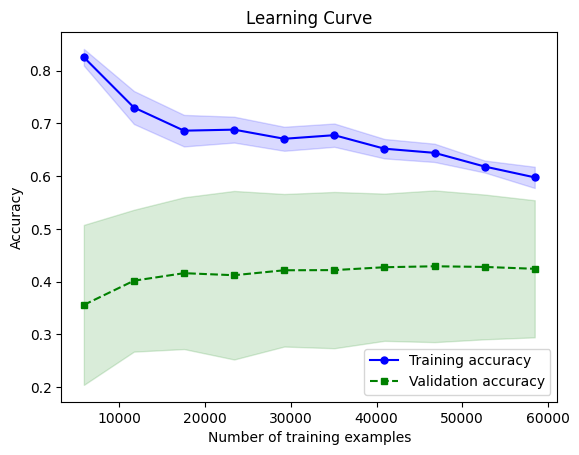
print("Gradient Boosting Accuracy Percentage : ", round(100 \* r\_squared\_gb, 2), "%")

Gradient Boosting Mean Squared Error : 8481187310.78

Gradient Boosting R-squared Value : 0.22683108552706288

Gradient Boosting Accuracy Percentage : 22.68 %

A graph showing the number of training and validation examples

Description automatically generated 

A graph showing the number of training and validation examples

Description automatically generated A graph showing the number of training and validation examples

Description automatically generated

A graph showing the number of training and validation examples

Description automatically generated A graph showing the number of training and validation examples

Description automatically generated

Plot the Results in Graph

A graph of blue and red lines

Description automatically generated

K Neighbor Regression Model1:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=3)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_knn = KNeighborsRegressor()

model\_knn.fit(X\_train, y\_train)

y\_pred\_knn = model\_knn.predict(X\_test)

mse\_knn = mean\_squared\_error(y\_test, y\_pred\_knn)

r\_squared\_knn = r2\_score(y\_test, y\_pred\_knn)

test.append(y\_test)

prediction.append(y\_pred\_knn)

print("K-Nearest Neighbors Mean Squared Error  : ", round(mse\_knn, 2))

print("K-Nearest Neighbors R-squared Value     : ", r\_squared\_knn)

print("K-Nearest Neighbors Accuracy Percentage : ", round(100 \* r\_squared\_knn, 2), "%")

K-Nearest Neighbors Mean Squared Error : 4358090945.67

K-Nearest Neighbors R-squared Value : 0.6027041589615263

K-Nearest Neighbors Accuracy Percentage : 60.27 %

K Neighbor Regression Model2:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=4)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_knn = KNeighborsRegressor()

model\_knn.fit(X\_train, y\_train)

y\_pred\_knn = model\_knn.predict(X\_test)

mse\_knn = mean\_squared\_error(y\_test, y\_pred\_knn)

r\_squared\_knn = r2\_score(y\_test, y\_pred\_knn)

test.append(y\_test)

prediction.append(y\_pred\_knn)

print("K-Nearest Neighbors Mean Squared Error  : ", round(mse\_knn, 2))

print("K-Nearest Neighbors R-squared Value     : ", r\_squared\_knn)

print("K-Nearest Neighbors Accuracy Percentage : ", round(100 \* r\_squared\_knn, 2), "%")

K-Nearest Neighbors Mean Squared Error : 4620176737.24

K-Nearest Neighbors R-squared Value : 0.5788116802862894

K-Nearest Neighbors Accuracy Percentage : 57.88 %

K Neighbor Regression Model3:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_knn = KNeighborsRegressor()

model\_knn.fit(X\_train, y\_train)

y\_pred\_knn = model\_knn.predict(X\_test)

mse\_knn = mean\_squared\_error(y\_test, y\_pred\_knn)

r\_squared\_knn = r2\_score(y\_test, y\_pred\_knn)

test.append(y\_test)

prediction.append(y\_pred\_knn)

print("K-Nearest Neighbors Mean Squared Error  : ", round(mse\_knn, 2))

print("K-Nearest Neighbors R-squared Value     : ", r\_squared\_knn)

print("K-Nearest Neighbors Accuracy Percentage : ", round(100 \* r\_squared\_knn, 2), "%")

K-Nearest Neighbors Mean Squared Error : 4353191882.72

K-Nearest Neighbors R-squared Value : 0.6031507713337951

K-Nearest Neighbors Accuracy Percentage : 60.32 %

K Neighbor Regression Model4:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model\_knn = KNeighborsRegressor()

model\_knn.fit(X\_train, y\_train)

y\_pred\_knn = model\_knn.predict(X\_test)

mse\_knn = mean\_squared\_error(y\_test, y\_pred\_knn)

r\_squared\_knn = r2\_score(y\_test, y\_pred\_knn)

test.append(y\_test)

prediction.append(y\_pred\_knn)

print("K-Nearest Neighbors Mean Squared Error  : ", round(mse\_knn, 2))

print("K-Nearest Neighbors R-squared Value     : ", r\_squared\_knn)

print("K-Nearest Neighbors Accuracy Percentage : ", round(100 \* r\_squared\_knn, 2), "%")

K-Nearest Neighbors Mean Squared Error : 4353191882.72

K-Nearest Neighbors R-squared Value : 0.6031507713337951

K-Nearest Neighbors Accuracy Percentage : 60.32 %

K Neighbor Regression Model5 Using PCA:

x = pd.DataFrame(x\_pca)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model\_knn = KNeighborsRegressor()

model\_knn.fit(X\_train, y\_train)

y\_pred\_knn = model\_knn.predict(X\_test)

mse\_knn = mean\_squared\_error(y\_test, y\_pred\_knn)

r\_squared\_knn = r2\_score(y\_test, y\_pred\_knn)

test.append(y\_test)

prediction.append(y\_pred\_knn)

print("K-Nearest Neighbors Mean Squared Error  : ", round(mse\_knn, 2))

print("K-Nearest Neighbors R-squared Value     : ", r\_squared\_knn)

print("K-Nearest Neighbors Accuracy Percentage : ", round(100 \* r\_squared\_knn, 2), "%")

K-Nearest Neighbors Mean Squared Error : 2828647999.62

K-Nearest Neighbors R-squared Value : 0.7421324841492635

K-Nearest Neighbors Accuracy Percentage : 74.21 %

K Neighbor Regression Model6 Using LDA:

x = pd.DataFrame(x\_lda)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model\_knn = KNeighborsRegressor()

model\_knn.fit(X\_train, y\_train)

y\_pred\_knn = model\_knn.predict(X\_test)

mse\_knn = mean\_squared\_error(y\_test, y\_pred\_knn)

r\_squared\_knn = r2\_score(y\_test, y\_pred\_knn)

test.append(y\_test)

prediction.append(y\_pred\_knn)

print("K-Nearest Neighbors Mean Squared Error  : ", round(mse\_knn, 2))

print("K-Nearest Neighbors R-squared Value     : ", r\_squared\_knn)

print("K-Nearest Neighbors Accuracy Percentage : ", round(100 \* r\_squared\_knn, 2), "%")

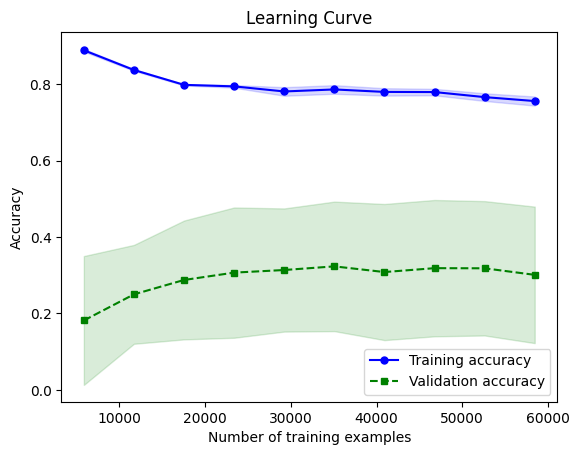
Learning Curves for The Models:

A graph showing the number of training and validation examples

Description automatically generated A graph showing the number of training indicators

Description automatically generated

A graph showing the number of training indicators

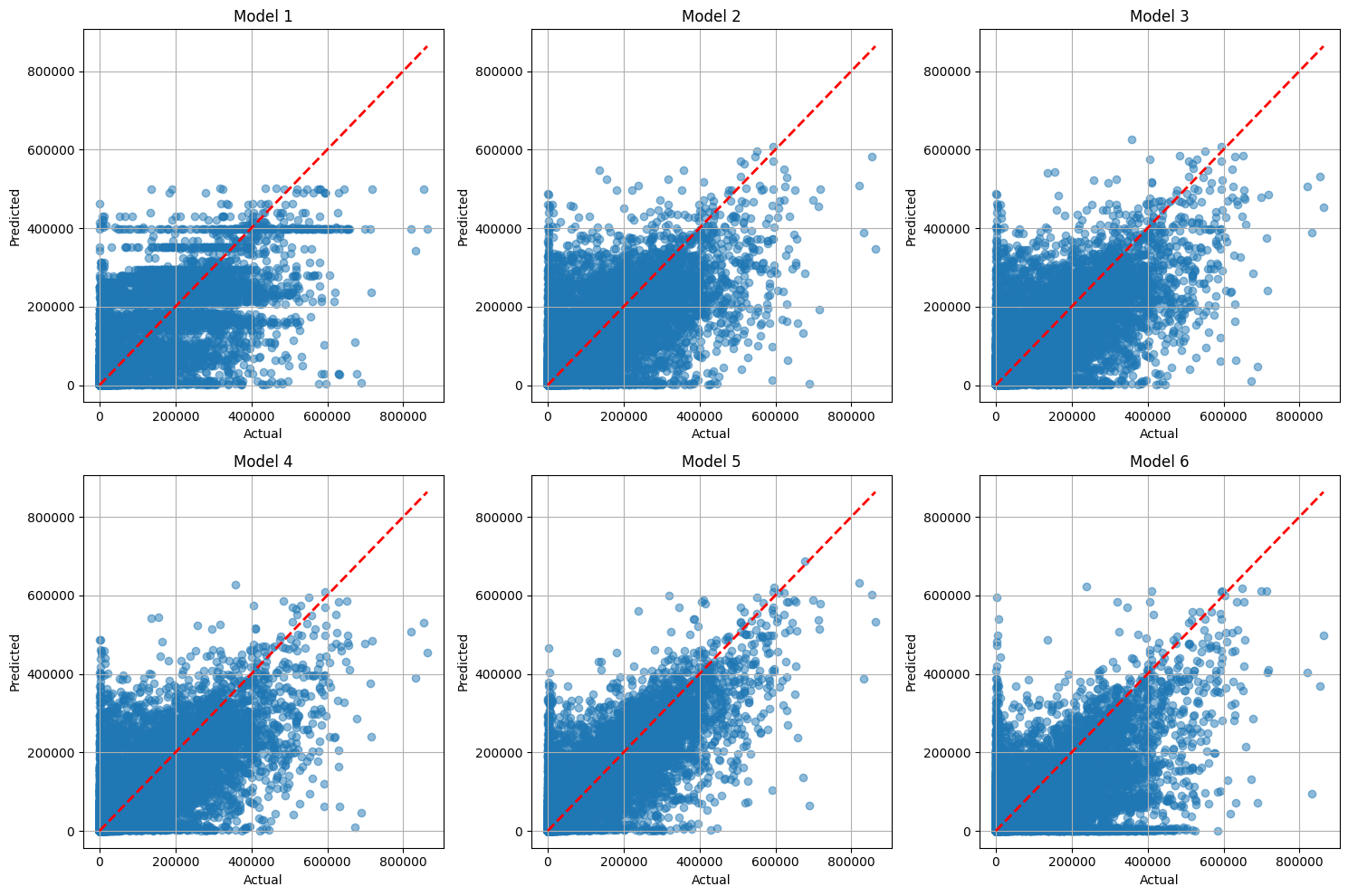
Description automatically generated 

A graph showing the number of training indicators

Description automatically generated A graph showing the number of training indicators

Description automatically generated

Plot the Results In The Graph:



Linear Regression Model1:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=3)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

Mean Squared Error : 10869082765.09

R-squared Value : 0.009143812668131801

Accuracy Percentage : 0.91 %

Linear Regression Model2:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=4)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

Mean Squared Error : 10857343649.72

R-squared Value : 0.010213983477342037

Accuracy Percentage : 1.02 %

Linear Regression Model3:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

Mean Squared Error : 10857354163.93

R-squared Value : 0.010213024972458573

Accuracy Percentage : 1.02 %

Linear Regression Model4:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

Mean Squared Error : 10857354163.93

R-squared Value : 0.010213024972458573

Accuracy Percentage : 1.02 %

Linear Regression Model5 using PCA:

x = pd.DataFrame(x\_pca)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

Mean Squared Error : 10857162242.02

R-squared Value : 0.010230521114152125

Accuracy Percentage : 1.02 %

Linear Regression Model5 using LDA:

x = pd.DataFrame(x\_lda)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

Mean Squared Error : 10857162242.02

R-squared Value : 0.010230521114152125

Accuracy Percentage : 1.02 %

A graph showing the number of training and validation examples

Description automatically generated A graph showing the number of training and validation examples

Description automatically generated

A graph showing the number of training and validation examples

Description automatically generated A graph showing the number of training and validation examples

Description automatically generated

A graph showing the number of training and validation examples

Description automatically generated A graph showing the number of training and validation examples

Description automatically generated

Plot the Results in Graph

A graph of a graph with a red line

Description automatically generated with medium confidence

Random Forest Regression Model1:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=3)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = RandomForestRegressor(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

Mean Squared Error : 3751661185.88

R-squared Value : 0.6579880032982834

Accuracy Percentage : 65.8 %

Random Forest Regression Model2:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=4)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = RandomForestRegressor(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

Mean Squared Error : 4116258138.62

R-squared Value : 0.6247503185455219

Accuracy Percentage : 62.48 %

Random Forest Regression Model3:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=5)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = RandomForestRegressor(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

Mean Squared Error : 3362184729.95

R-squared Value : 0.6934937736124005

Accuracy Percentage : 69.35 %

Random Forest Regression Model4:

x=df.drop(columns=["totvotpoll"])

y=df['totvotpoll']

selector = SelectKBest(score\_func=f\_regression, k=6)

selector.fit(x, y)

selected\_features = x.columns[selector.get\_support()]

print("Selected Features:")

print(selected\_features)

Selected Features:

Index(['st\_name', 'year', 'pc\_no', 'cand\_sex', 'partyabbre', 'electors'], dtype='object')

X = df[selected\_features]

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = RandomForestRegressor(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

Mean Squared Error : 1504170913.24

R-squared Value : 0.8628755444779936

Accuracy Percentage : 86.29 %

Random Forest Regression Model5 PCA:

pca = PCA(n\_components=5)

pca.fit(x)

x\_pca = pca.transform(x)

print("Selected Features (Principal Components):")

print(x\_pca.shape)

x = pd.DataFrame(x\_pca)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

train\_sizes, train\_scores, test\_scores = learning\_curve(estimator=model, X=X, y=y, train\_sizes=np.linspace(0.1, 1.0, 10), cv=5)

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, color='blue', marker='o', markersize=5, label='Training accuracy')

plt.fill\_between(train\_sizes, train\_mean + train\_std, train\_mean - train\_std, alpha=0.15, color='blue')

plt.plot(train\_sizes, test\_mean, color='green', linestyle='--', marker='s', markersize=5, label='Validation accuracy')

plt.fill\_between(train\_sizes, test\_mean + test\_std, test\_mean - test\_std, alpha=0.15, color='green')

plt.xlabel('Number of training examples')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.title('Learning Curve')

plt.show()

A black background with white text

Description automatically generated

Random Forest Regression Model6 Using LDA:

x = pd.DataFrame(x\_lda)

y = df['totvotpoll']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model = RandomForestRegressor(random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

test.append(y\_test)

prediction.append(y\_pred)

print("Mean Squared Error  : ", round(mse,2))

print("R-squared Value     : ", r\_squared)

print("Accuracy Percentage : ", round(100\*r\_squared,2,),"%")

A black background with white text

Description automatically generated

A graph showing the number of training indicators

Description automatically generated A graph showing the number of training indicators

Description automatically generated

A graph showing the number of training calculations

Description automatically generated A graph showing the number of training indicators

Description automatically generated

A graph showing the number of training calculations

Description automatically generated A graph showing the number of training and validation examples

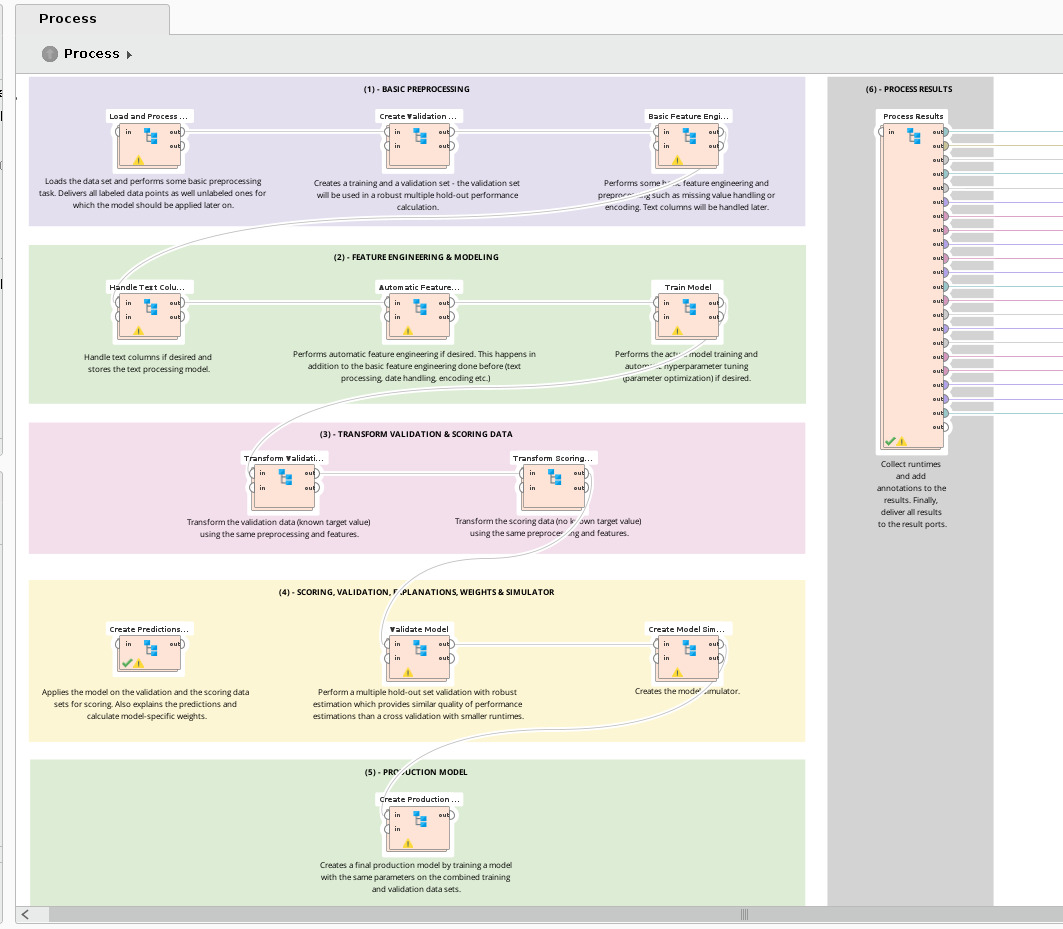
Description automatically generated

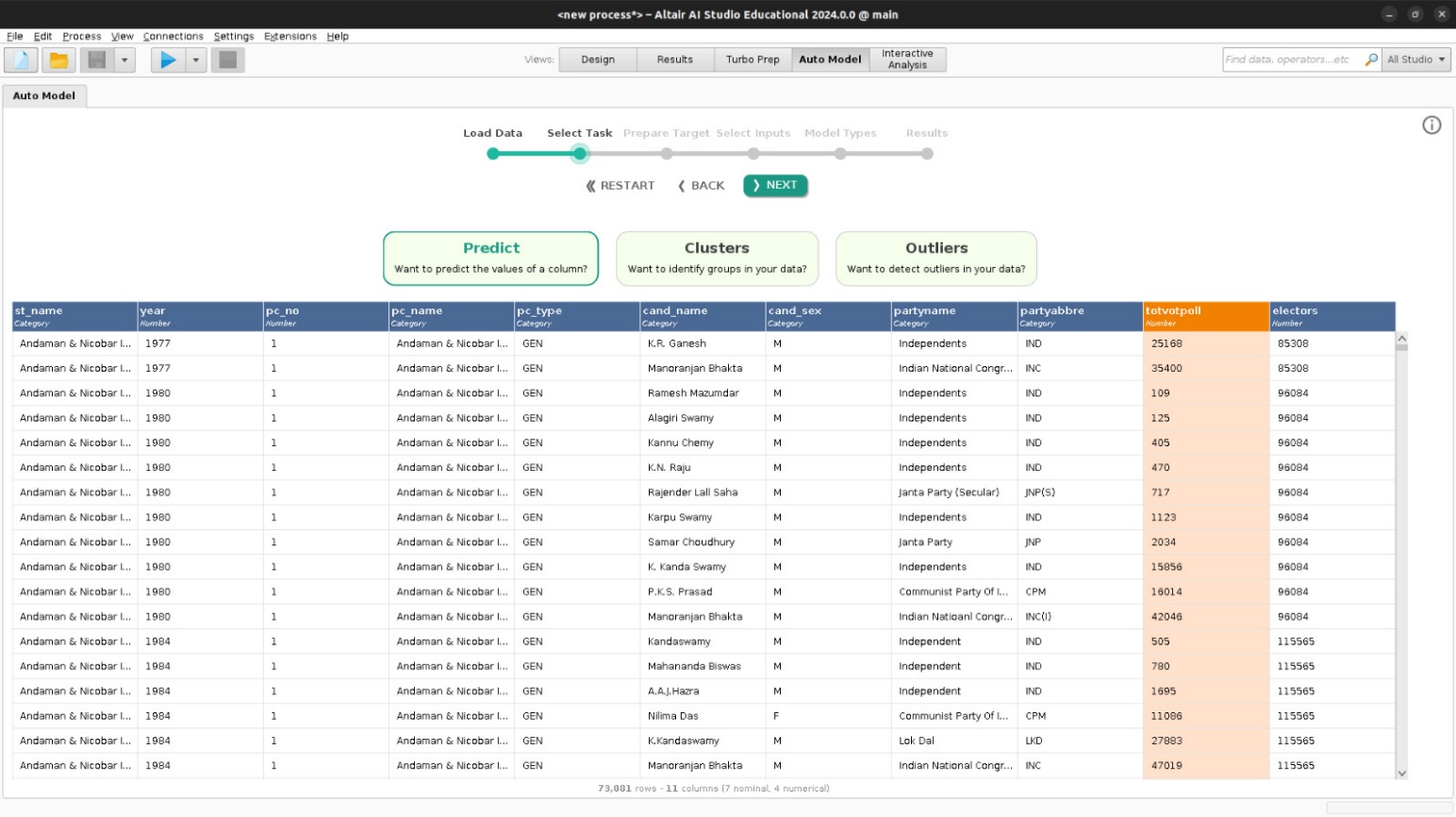
Plot the results using Graph

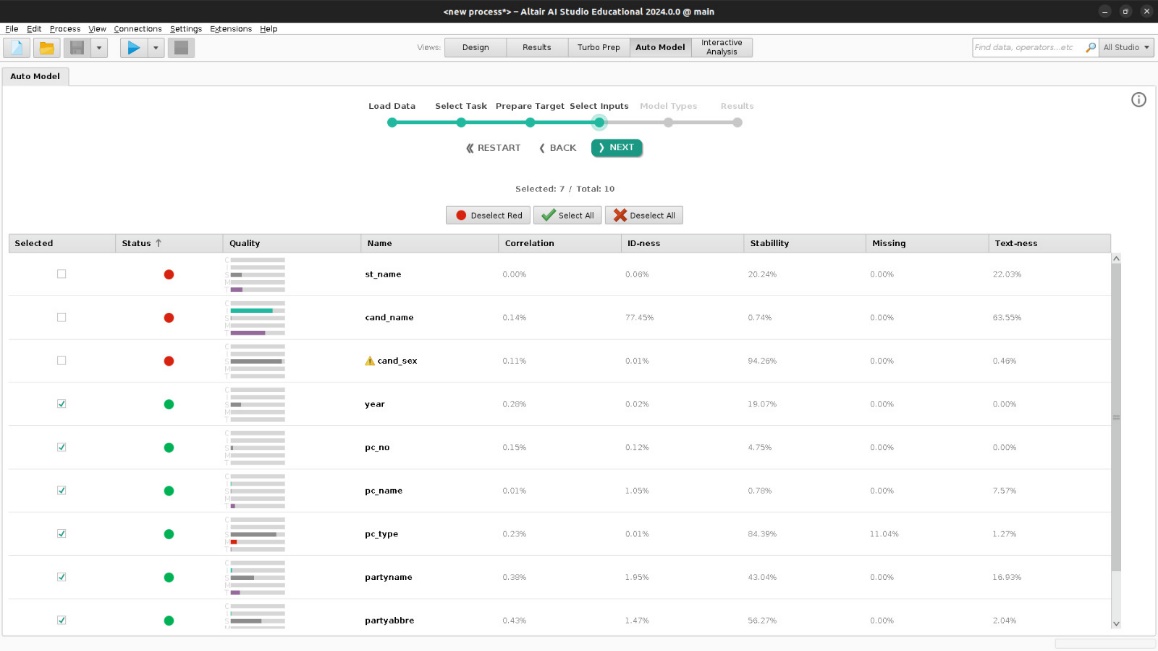
A graph of blue dots

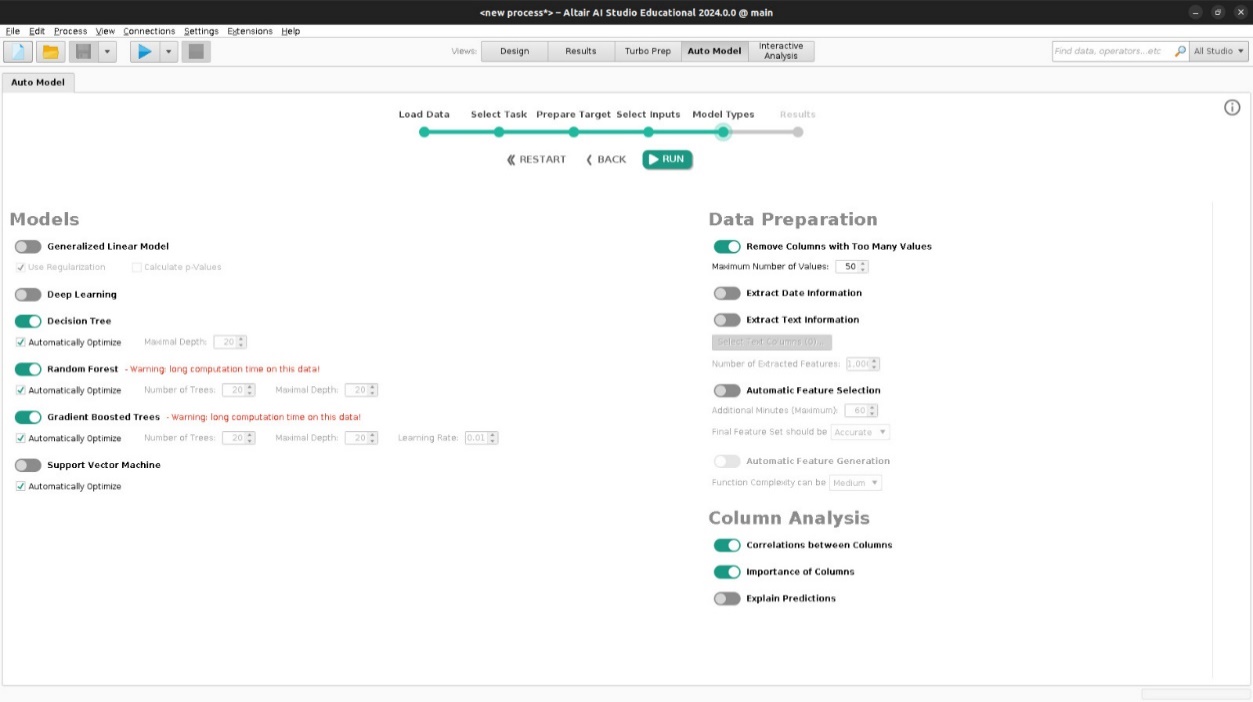
Description automatically generated with medium confidence

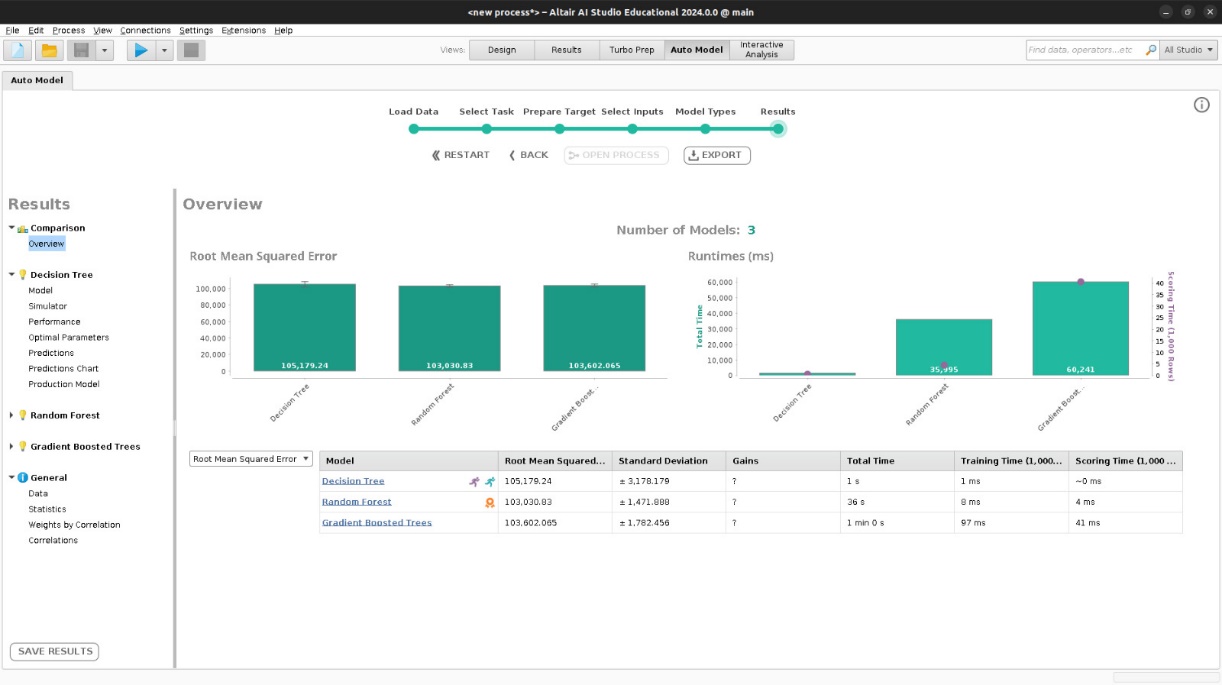
**Model Deployment in Rapid Miner**

****

****

****

****

****

**Comparison of ML Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Feature Engineering | Mean Squared  Error | R Squared  Score | Accuracy |
| Gradient  Boosting  Regression  Model | K = 3 | 4689954366.56 | 0.572450 | 57.25 % |
| K = 4 | 4691351252.52 | 0.572323 | 57.23 % |
| K = 5 | 4651621664.73 | 0.5759450 | 57.59 % |
| K = 6 | 4651487603.76 | 0.575957 | 57.6 % |
| PCA | 4468413418.89 | 0.592646 | 59.26 % |
| LDA | 8283375210.57 | 0.244864 | 24.49 % |
| K Neighbor  Regression  Model | K = 3 | 4358090945.67 | 0.602704 | 60.27 % |
| K = 4 | 4620176737.24 | 0.578811 | 57.88 % |
| K = 5 | 4353191882.72 | 0.603150 | 60.32 % |
| K = 6 | 4353191882.72 | 0.603150 | 60.32 % |
| PCA | 2828647999.62 | 0.742132 | 74.21 % |
| LDA | 6685534694.59 | 0.390527 | 39.05 % |
| Linear  Regression  Model | K = 3 | 10869082765.09 | 0.0091438 | 0.91 % |
| K = 4 | 10857343649.72 | 0.010213 | 1.02 % |
| K = 5 | 10857354163.93 | 0.010213 | 1.02 % |
| K = 6 | 10857354163.93 | 0.010213 | 1.02 % |
| PCA | 10857162242.02 | 0.010230 | 1.02 % |
| LDA | 10857162242.02 | 0.010230 | 1.02 % |
| Random  Forest  Regression  Model | K = 3 | 3751661185.88 | 0.657988 | 65.8 % |
| K = 4 | 4116258138.62 | 0.624750 | 62.48 % |
| K = 5 | 3362184729.95 | 0.693493 | 69.35 % |
| K = 6 | 1504170913.24 | 0.862875 | 86.29 % |
| PCA | 2055588563.91 | 0.812606 | 81.26 % |
| LDA | 3516979397.51 | 0.679382 | 67.94 % |

Random Forest Regression:

This ensemble learning method constructs a multitude of decision trees during training and outputs the mean prediction of the individual trees. It's known for its high accuracy, robustness to overfitting, and effectiveness in handling large datasets with high dimensionality.

K Nearest Neighbors Regression:

KNN is a simple, instance-based learning algorithm where the prediction is based on the majority of the k-nearest neighbors of a query point in feature space. It's intuitive and easy to implement, but its performance can degrade with high dimensionality and large datasets.

Gradient Boosting Regression:

This ensemble technique builds a series of weak learners (usually decision trees) sequentially, each one focusing on the errors made by the previous learners. It's effective in minimizing various loss functions and often yields high accuracy, but it can be computationally expensive and prone to overfitting if not properly tuned.

Linear Regression:

This is a linear approach to modeling the relationship between a dependent variable and one or more independent variables. It's simple and interpretable but assumes a linear relationship between the features and the target variable, which might not always hold true in real-world datasets.

Based on accuracy,

* **Random Forest Regression model with K = 6** achieves the highest accuracy of **86.29%,** followed by the **PCA-based Random Forest model** with an accuracy of **81.26%.**
* The Random Forest regression models with different data provides the higher Accuracy compare than the other three models.
* The K Neighbor Regression model comparatively provides the higher accuracy than Linear Regression models and Gradient Boosting Regression models
* Linear Regression models, along with Gradient Boosting Regression models, generally perform poorly compared to Random Forest and K Nearest Neighbors models.
* Best Models for The dataset based on Accuracy,

1) Random Forest Regression

2) K Neighbor Regression Model

Based on Computational Time,

Computational Time

|  |  |
| --- | --- |
| Model | Computational Time |
| Random Forest Regression | 26.039166927337646 seconds |
| Linear Regression | 0.10006356239318848 seconds |
| K neighbor Regression | 0.15254878997802734 seconds |
| Gradient Boosting Regression | 10.362363815307617 seconds |

* By seeing the above results, the Linear regression model takes the least computational time but provides the very low accuracy which is less than 1%. So, this is worst model for our prediction
* The K Neighbor Regression takes the second least computational time among four and also provide the best accuracy during the testing data. So, K Neighbor regression is the best model for the prediction.
* The Gradient Boosting Regression takes more than 10 sec computational time and also provide the average accuracy during the testing data. So, Gradient Boosting regression is the worst model for the prediction.
* The Random Forest Regression takes more than 20 seconds computational time which is the highest among four and also provide the best accuracy during the testing data. So, Random Forest Regression is the best model for the prediction based on the Accuracy not for the computational time.

Best Models for the dataset based on Computational Time,

1. K Neighbor Regression Model
2. Random Forest Regression

Based on Fitting,

In Random Forest Regression:

while increasing the dataset size the validation accuracy goes nearly to the training accuracy. So, the Random Forest regression fits a Good fit. So, Linear Regression is a Good fit model for this dataset

In Linear Regression:

while increasing the dataset size the training accuracy and validation accuracy both goes nearly to 0. So, the linear regression fits underfitting. So, Linear Regression is a underfit model for this dataset

In K neighbor Regression:

There is a huge difference between the training accuracy and validation accuracy. So, the K neighbor fits an over fit. So, K neighbor is an over fit model for this dataset

In Gradient Boosting Regression:

while increasing the dataset size the validation accuracy goes nearly to the training accuracy. So, the Gradient Boosting regression fits a Good fit. So, Gradient Boosting regression is a Good fit model for this dataset.

|  |  |
| --- | --- |
| Model | Fitting |
| Random Forest Regression | Good fit |
| Linear Regression | Under Fit |
| K neighbor Regression | Over fit |
| Gradient Boosting Regression | Good fit |

**Results**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Accuracy | Computational Time | Fitting |
| Random Forest Regression | High Accuracy | High | Good fit |
| Linear Regression | Worst Accuracy | Low | Under Fit |
| K Neighbor Regression | High Accuracy | Low | Over fit |
| Gradient Boosting Regression | Worst Accuracy | High | Good fit |

**Inferences**

Random Forest Regression:

Achieves high accuracy.

Requires a relatively high computational time.

Provides a good fit to the data.

Inference: Random Forest Regression is suitable for tasks where accuracy is crucial and computational resources are available.

Linear Regression:

Yields the worst accuracy.

Requires low computational time.

Tends to underfit the data.

Inference: Linear Regression is efficient but may not capture the complexity of the data well, making it suitable for simpler problems with fewer features.

K Nearest Neighbors Regression:

Achieves high accuracy.

Requires low computational time.

Tends to overfit the data.

Inference: K Nearest Neighbors Regression is efficient and effective for smaller datasets but may not generalize well to unseen data due to overfitting.

Gradient Boosting Regression:

Yields the worst accuracy.

Requires a relatively high computational time.

Provides a good fit to the data.

Inference: Gradient Boosting Regression provides a good fit but may require more computational resources and tuning compared to other models.

**Conclusion**

The choice of the best model depends on the specific requirements of the problem, such as the importance of accuracy, computational resources available, and the trade-off between overfitting and underfitting. Random Forest Regression and K Nearest Neighbors Regression are suitable for tasks where accuracy is crucial and computational resources are limited, while Linear Regression may be preferred for simpler problems with low computational requirements. Gradient Boosting Regression can be effective but may require more computational resources and tuning to achieve optimal performance.

By considering the results the best Model for predicting the Voters in Indian general election is

Random Forest Regression

**Future Work**

In the current election voting prediction models, only the previous year's election results are considered. This approach relies solely on historical data to make predictions about future elections. However, it doesn't take into account the current situation or any new factors that may influence voter behavior, such as prevailing emotions or emerging issues.

In future work, it is proposed to include these additional factors, such as emotions and sentiments (like "anuthapavam" votes in the election), to enhance the accuracy and relevance of the prediction models. Here's a brief explanation of how this inclusion could improve the models:

* Emotions and Sentiments:

By incorporating emotions and sentiments prevalent among voters during the current election cycle, the models can better capture the mood of the electorate. Analyzing sentiment from social media, news articles, or surveys can provide valuable insights into the prevailing mood and sentiment of voters.

* Improved Predictive Power:

Incorporating current situation factors enhances the predictive power of the models by providing a more comprehensive understanding of voter behavior. By considering both historical trends and present-day dynamics, the models can better anticipate shifts in voter sentiment and behavior, leading to more accurate predictions of election outcomes.

Overall, by integrating emotions, sentiments, and other current situation factors into election voting prediction models, we can create more robust and insightful tools for understanding and forecasting electoral dynamics. This approach enables us to capture the complexities of voter behavior more accurately and adaptively, thereby enhancing the effectiveness of election prediction efforts.

**Learning Outcomes**

* Better understanding about the various machine learning regression model
* We learnt about the strengths, weaknesses, and suitability of ML model
* The Machine learning models are evaluated by several evaluation metrics
* We learnt about the importance of feature engineering in improving the machine model performance
* Better understanding about how to select and preprocess features to enhance the predictive power of machine learning models.
* learnt how to compare different machine learning models based on performance metrics and make inferences about their suitability for specific tasks
* Gain knowledge about the trade-offs between model accuracy, computational efficiency, and fitting.
* We understand that the additional information may increase the performance of the machine learning models.
* We identifying limitations in existing models and proposing future work to address them

**References**

<https://en.wikipedia.org/wiki/Linear_regression>

<https://www.geeksforgeeks.org/random-forest-regression-in-python/>

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>

<https://lms.ssn.edu.in/course/view.php?id=2231>

LMS For Principals Of management