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

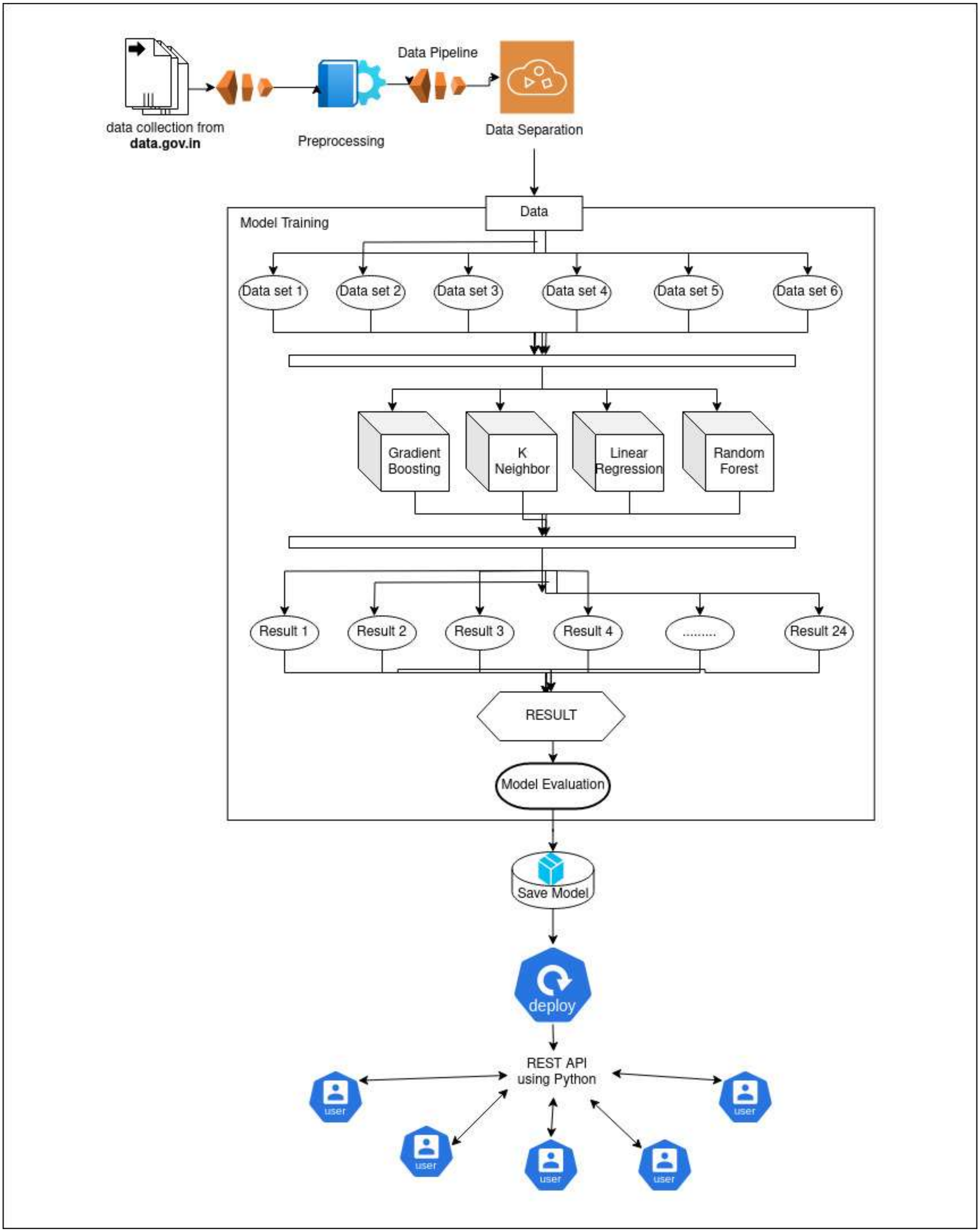
Problem Statement

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



data collection from
data.gov.in
Preprocessing
Data Pipeline
Data Separation
Data
Model Training
Data set 1
Data set 2
Data set 3
Data set 4
Data set 5
Data set 6
Gradient Boosting
K Neighbor
Linear Regression
Random Forest
Result 1
Result 2
Result 3
Result 4
....
Result 24
RESULT
Model Evaluation
Save Model
deploy
REST API using Python
user
user
user
user
user

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)
```

```
The Shape of the Data set :  (73081, 11)
```

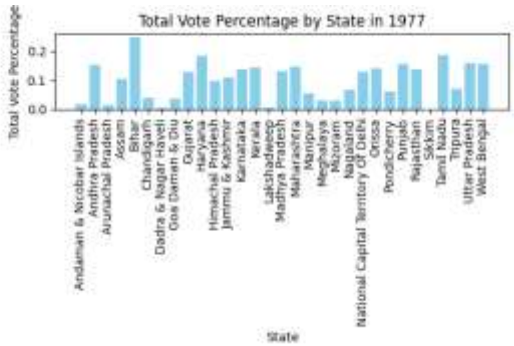
2) Display the Attributes and datatypes

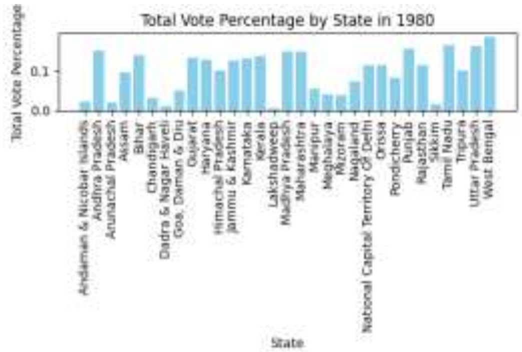
```
print("The Data Types of The Attributes are\n\n",df.dtypes)
```

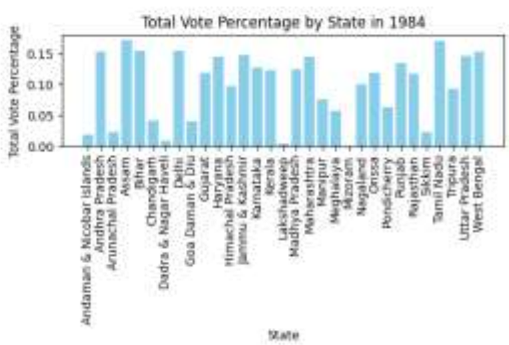
```
The Data Types of The Attributes are

 st_name       object
year           int64
pc_no          int64
pc_name       object
pc_type       object
cand_name     object
cand_sex      object
partyname     object
partyabbre    object
totvotpoll     int64
electors       int64
dtype: object
```

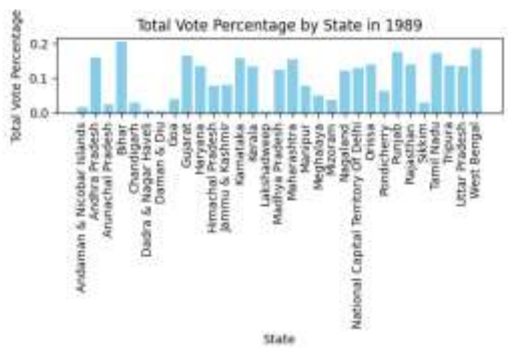
3) Display the State Wise Vote Percentage in Each Election

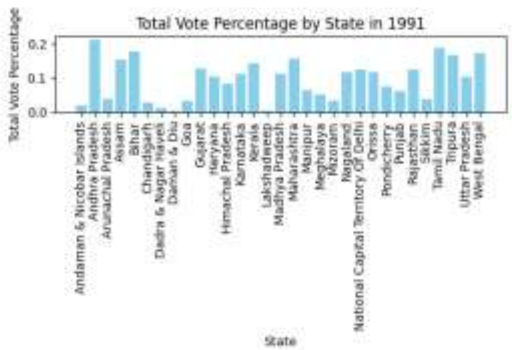
```
grouped_data = df.groupby('year')

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()
```

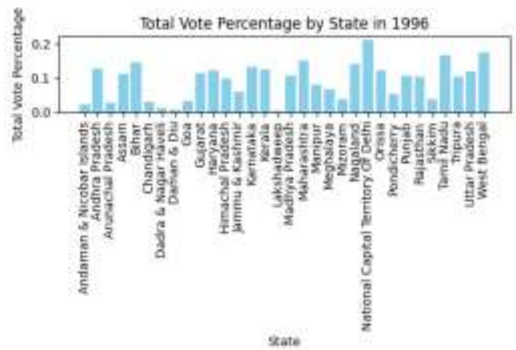

Total Vote Percentage by State in 1977
Total Vote Percentage
State

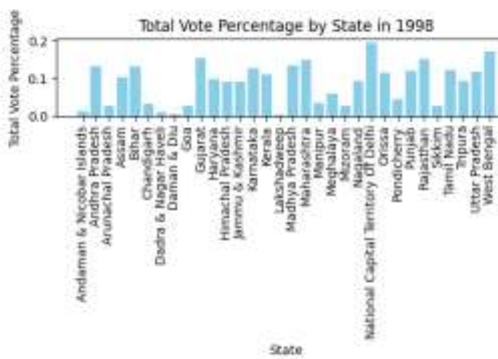

Total Vote Percentage by State in 1980
Total Vote Percentage
State

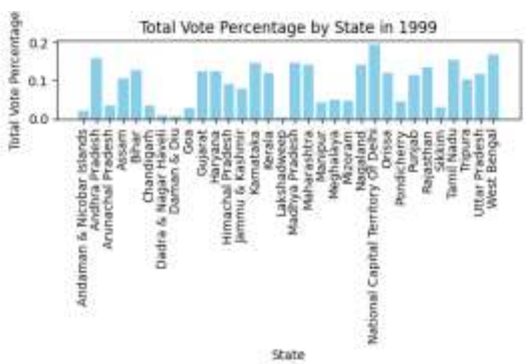

Total Vote Percentage by State in 1984
Total Vote Percentage
State

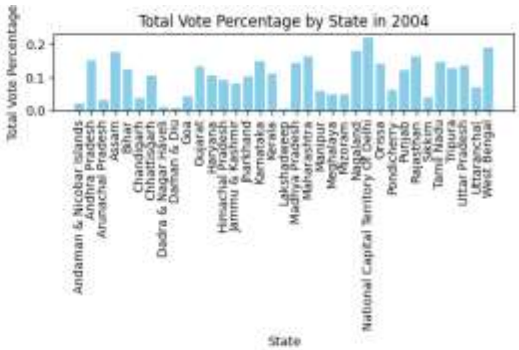

Total Vote Percentage by State in 1989
Total Vote Percentage
State

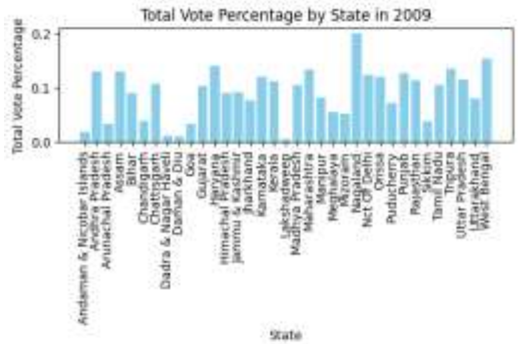

Total Vote Percentage by State in 1991
Total Vote Percentage
State

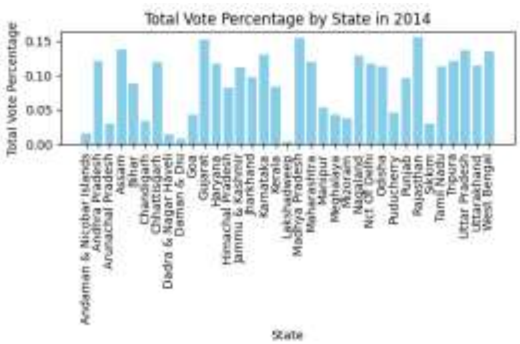

Total Vote Percentage by State in 1996
Total Vote Percentage
State


Total Vote Percentage by State in 1998
Total Vote Percentage
State


Total Vote Percentage by State in 1999
Total Vote Percentage
State







4) 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)
```

```
       year                      pc_name partyabbre  totvotpoll
230    1977                     Adilabad        INC      167410
26001  1977                        Adoor        CPI      227939
54809  1977                         Agra        BLD      257472
15633  1977                    Ahmedabad        INC      187715
33241  1977                   Ahmednagar        INC      179550
...     ...                          ...        ...         ...
38715  2014                       Wardha        BJP      537518
27595  2014                      Wayanad        INC      377035
42001  2014  West Delhi                         BJP      651395
38858  2014               Yavatmal-Washim       SHS      477905
4759   2014                     Zahirabad       TRS      508661
```

5) 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)
```

```
     year partyabbre  count
0    1977        ADK     18
1    1977        BLD    292
2    1977        CPI      7
3    1977        CPM     22
4    1977        DMK      2
..    ...        ...    ...
323  2014         SP      5
324  2014        SWP      1
325  2014        TDP     16
326  2014        TRS     11
327  2014      YSRCP      9
```

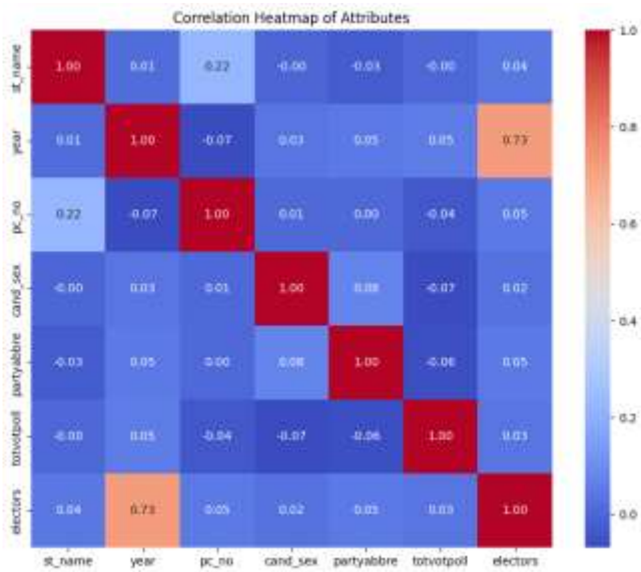
```
Year = int(input("Enter The Year : "))
year_counts = party_counts[party_counts['year'] == Year]
sorted_party_counts = year_counts.sort_values(by='count', ascending=False)
print(sorted_party_counts)
```

```
     year partyabbre  count
300  2014        BJP    279
304  2014        INC     44
294  2014       ADMK     37
297  2014       AITC     34
299  2014        BJD     20
322  2014        SHS     18
325  2014        TDP     16
326  2014        TRS     11
327  2014      YSRCP      9
303  2014        CPM      9
313  2014        LJP      6
314  2014        NCP      6
323  2014         SP      5
320  2014        SAD      4
318  2014        RJD      4
292  2014       AAAP      4
305  2014        IND      3
301  2014       BLSP      3
298  2014      AIUDF      3
310  2014      JKPDP      3
307  2014       IUML      2
306  2014       INLD      2
309  2014      JD(U)      2
311  2014        JMM      2
...
321  2014        SDF      1
324  2014        SWP      1
296  2014      AINRC      1
295  2014      AIMIM      1
```

6) 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'])
```

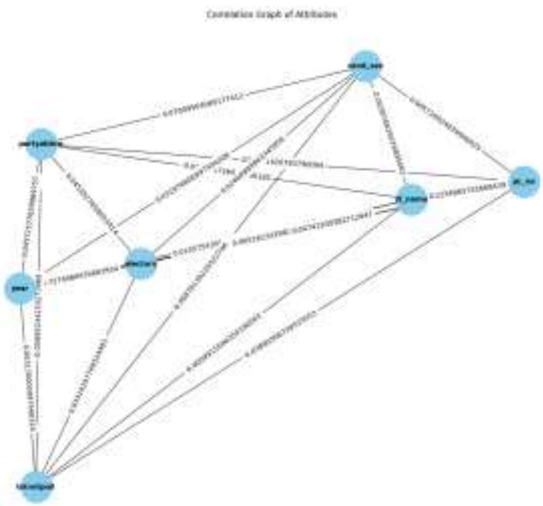
```python
df['cand_sex'] = label_encoder.fit_transform(df['cand_sex'])
print(df.head())

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()
```



7) Display the Correlation Graph between Each Attributes

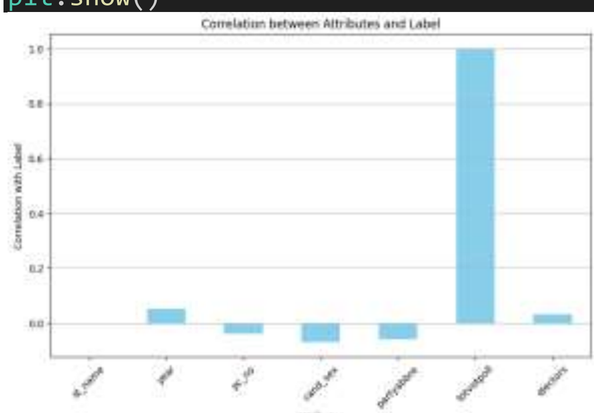
```python
import networkx as nx
attributes_to_scale = df.columns.tolist()
G = nx.Graph()
for i, attribute1 in enumerate(attributes_to_scale):
    for j, attribute2 in enumerate(attributes_to_scale):
        if i != j:
            correlation = df[attribute1].corr(df[attribute2])
            G.add_edge(attribute1, 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()
```



8) 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())
```

```
The Missing Values in the Dataset

 st_name       0
year          0
pc_no         0
cand_sex      0
partyabbre    0
totvotpoll    0
electors      0
dtype: int64
```

2) Converting the categorical values into Numerical Values

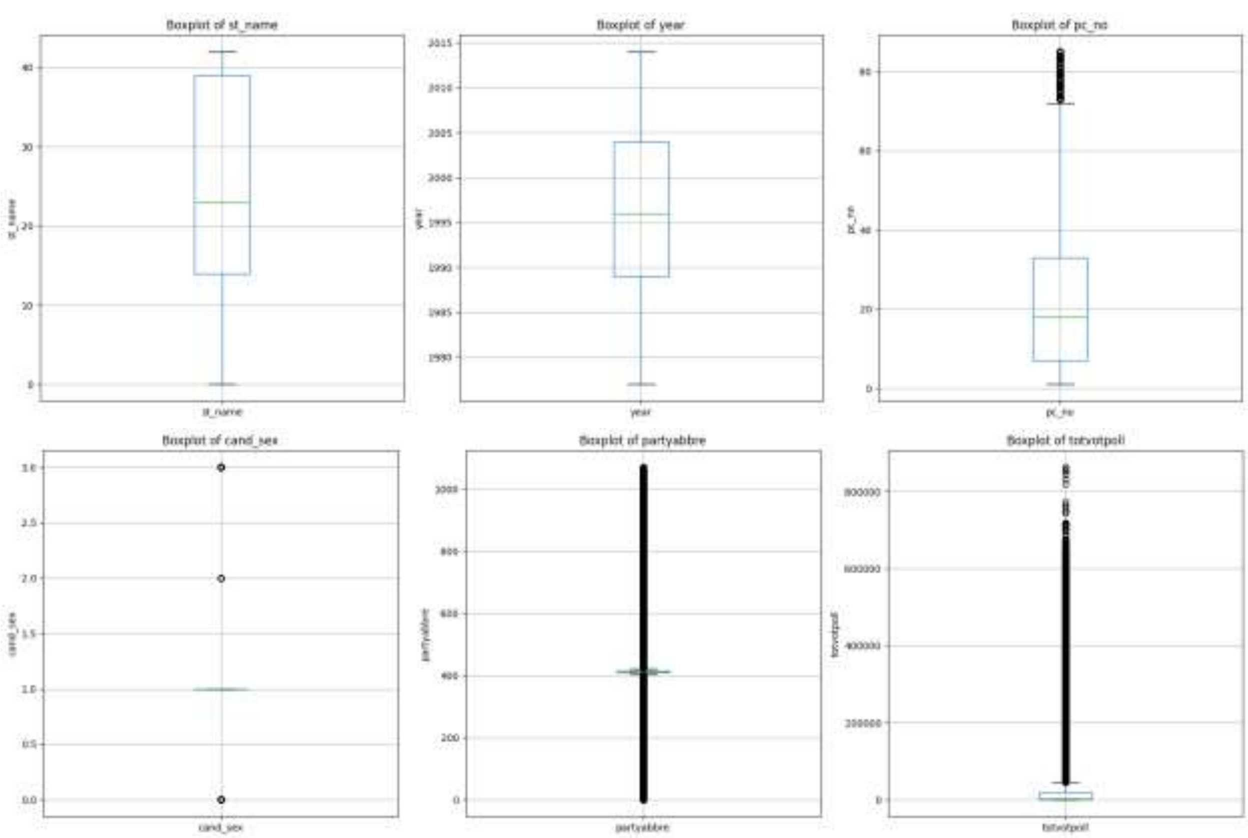
```
label_encoder = LabelEncoder()

#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())
```

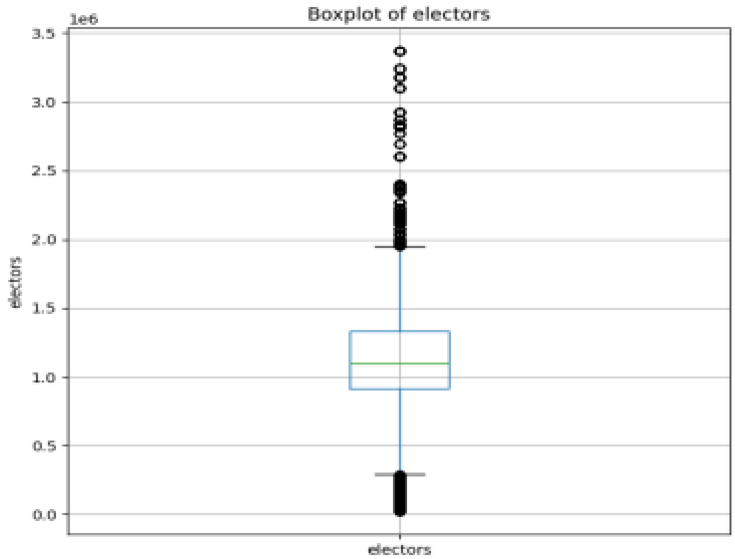
```
   st_name  year  pc_no  cand_sex  partyabbre  totvotpoll  electors
0        0  1977      1         1         414       25168     85308
1        0  1977      1         1         410       35400     85308
2        0  1980      1         1         414         109     96084
3        0  1980      1         1         414         125     96084
4        0  1980      1         1         414         405     96084
```

3) 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=[]
```

```
After removing the Unnecessary Attributes The Remaining Attributes in The Dataset are

 st_name         int32
year            int64
pc_no           int64
cand_sex        int32
partyabbre      int32
totvotpoll      int64
electors      float64
dtype: object
```

2) 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 %

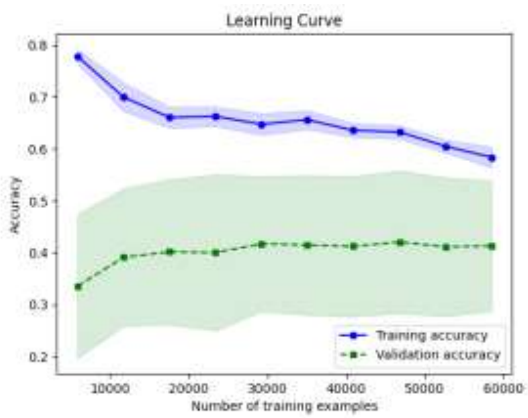
Gradient Boosting Regression Model6 Using LDA:

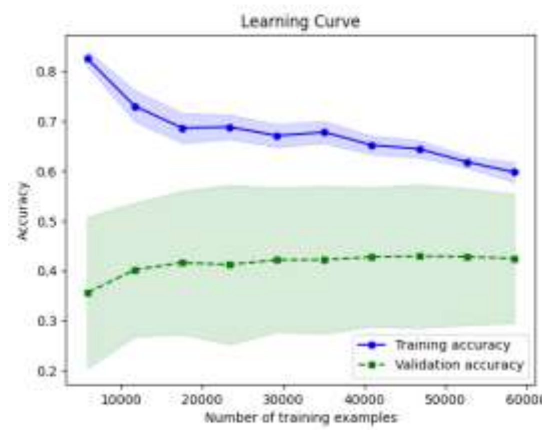
```
lda = LinearDiscriminantAnalysis(n_components=5)
lda.fit(x, y)

x_lda = lda.transform(x)

print("Selected Features (Linear Discriminant Components):")
print(x_lda.shape)
```

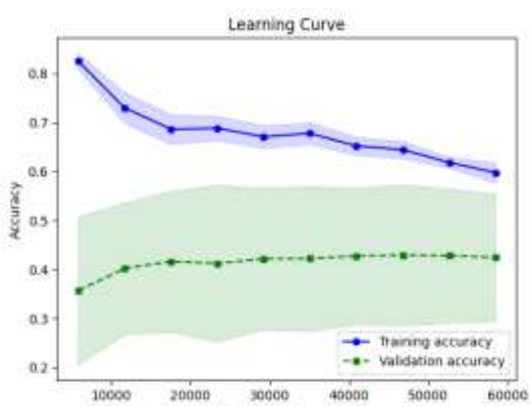
```
x = pd.DataFrame(x_lda)
y = df['totvotpoll']

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)
```

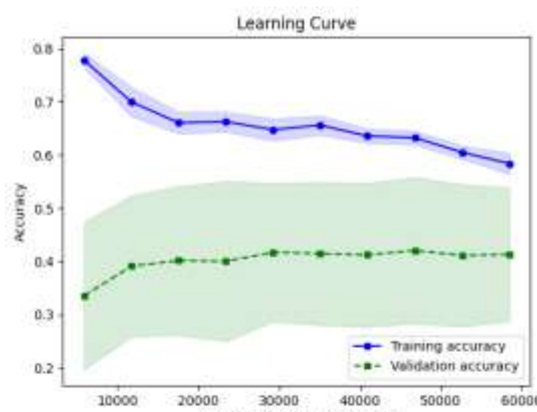
```
mse_gb = mean_squared_error(y_test, y_pred_gb)
r_squared_gb = r2_score(y_test, y_pred_gb)

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), "%")
```

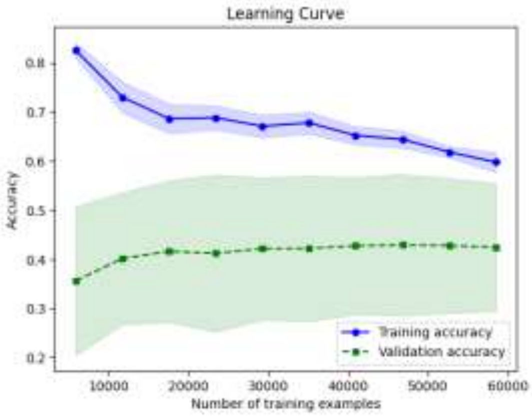
Gradient Boosting Mean Squared Error : 8481187310.78
Gradient Boosting R-squared Value : 0.22683108552706288
Gradient Boosting Accuracy Percentage : 22.68 %

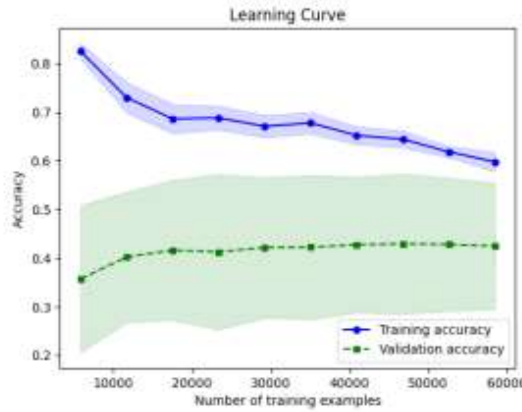




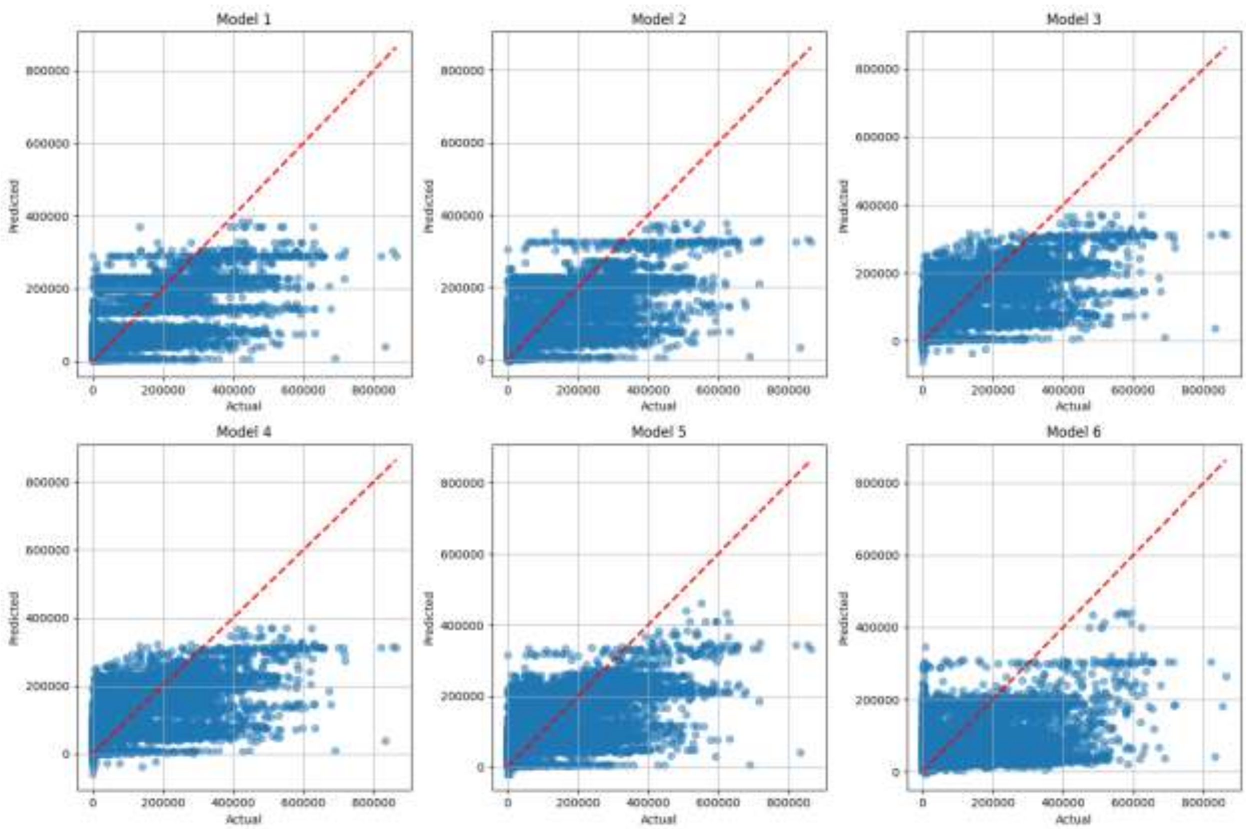








Plot the Results in Graph



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

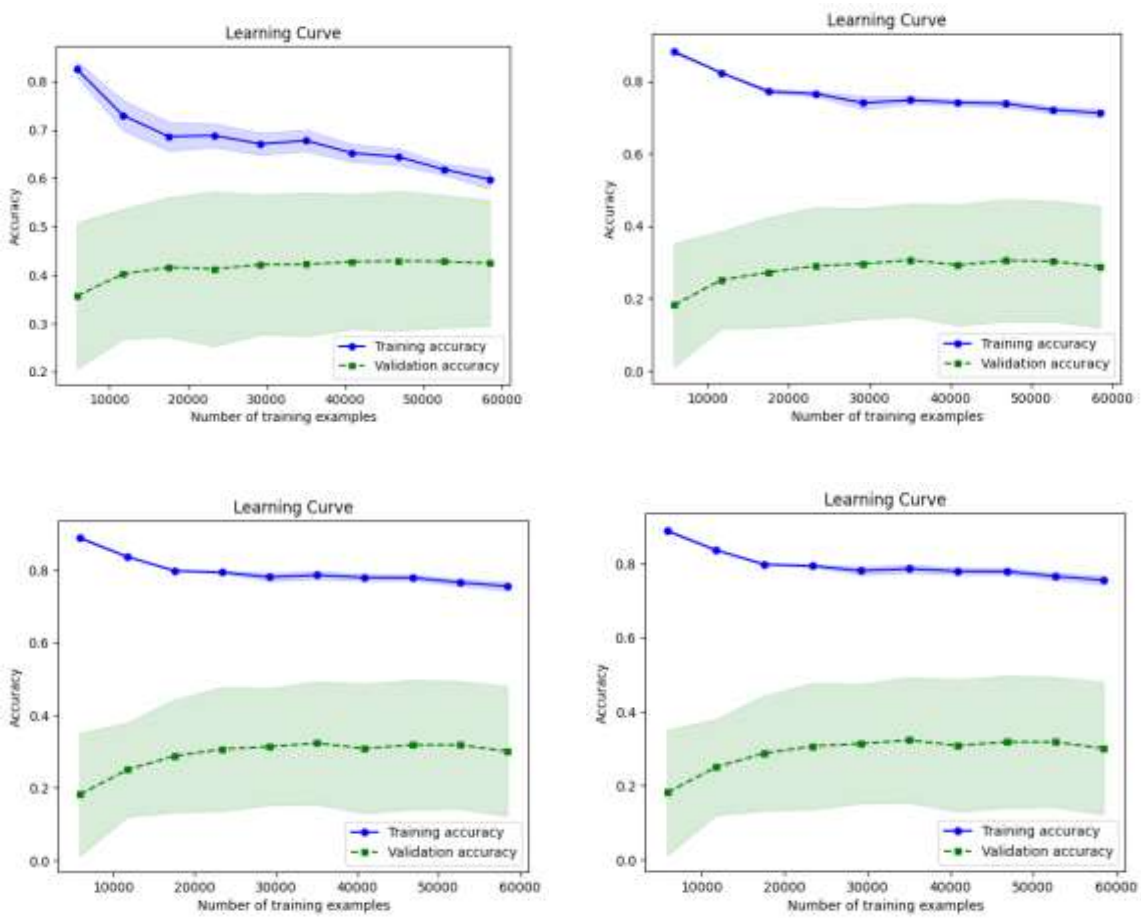
K-Nearest Neighbors R-squared Value   : 0.7421324841492635

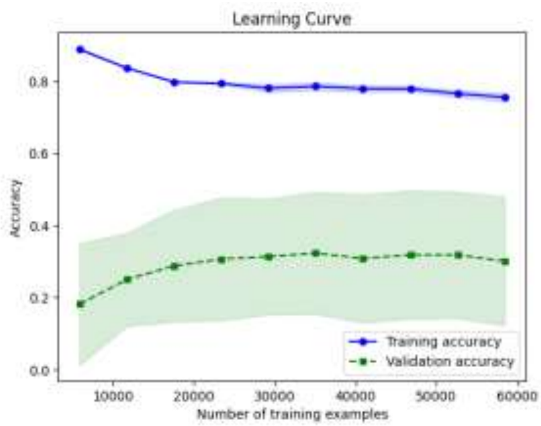
K-Nearest Neighbors Accuracy Percentage : 74.21 %

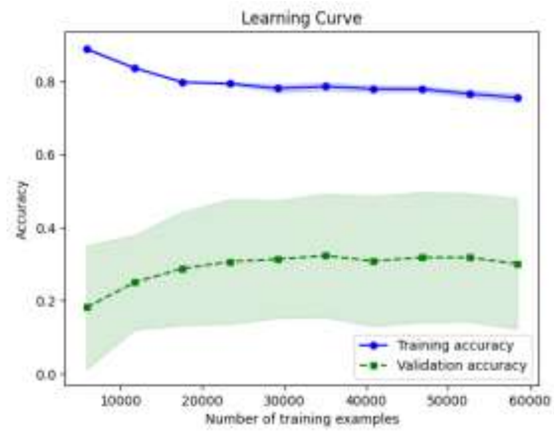
K Neighbor Regression Model6 Using LDA:

```
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)
```

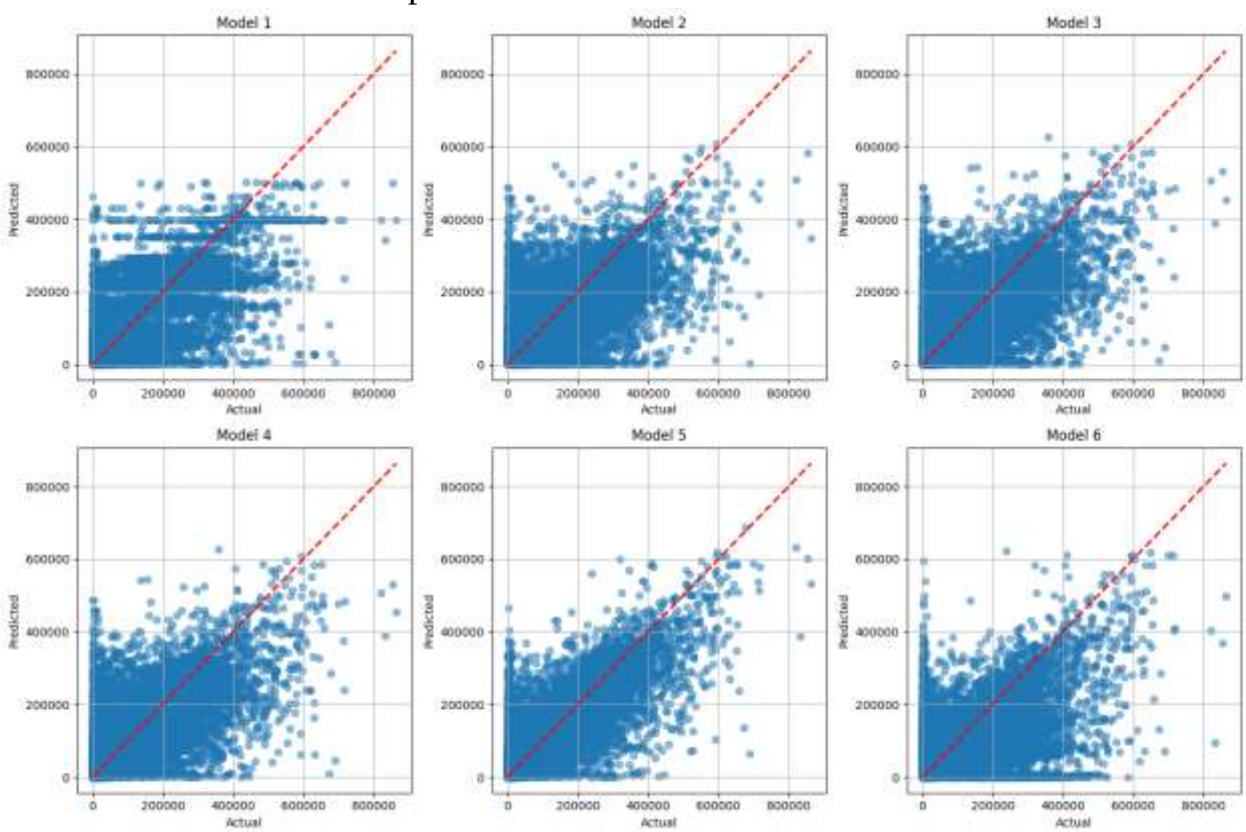
```
y_pred_knn = model_knn.predict(X_test)

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:







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)
```

```python
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,),"%")
```

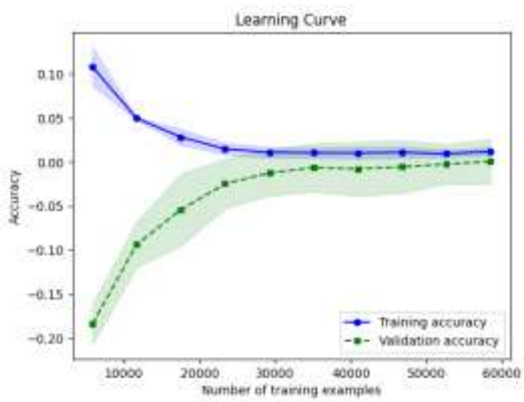
Mean Squared Error : 10857162242.02
R-squared Value : 0.010230521114152125
Accuracy Percentage : 1.02 %

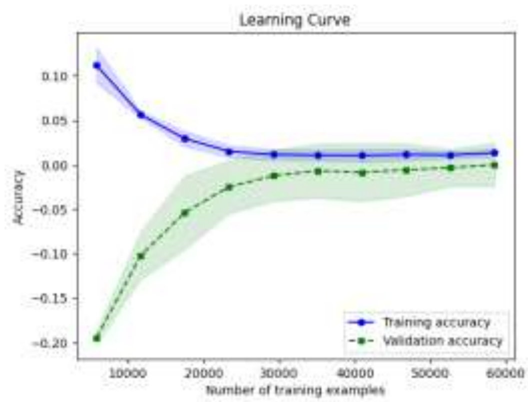
Linear Regression Model5 using LDA:

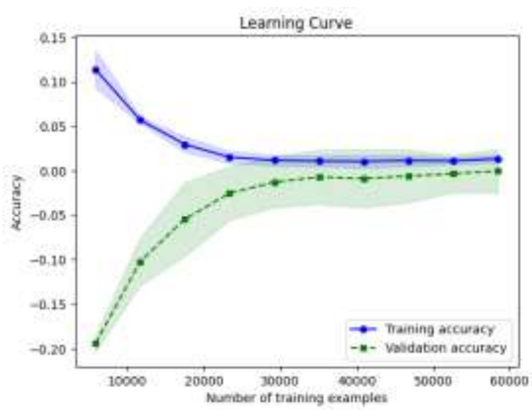
```
x = pd.DataFrame(x_lda)
y = df['totvotpoll']

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)
```

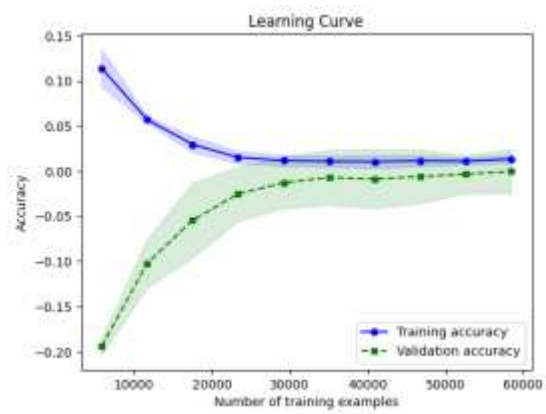
```python
test.append(y_test)
prediction.append(y_pred)

print("Mean Squared Error  : ", round(mse,2))
print("R-squared Value     : ", r_squared)
print("Accuracy Percentage : ", round(100*r_squared,2,),"%")
```

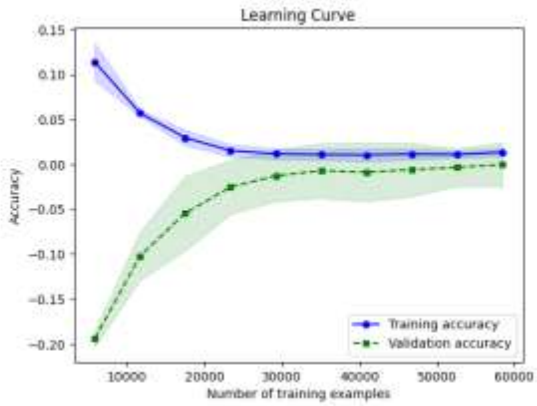
Mean Squared Error : 10857162242.02
R-squared Value   : 0.010230521114152125
Accuracy Percentage : 1.02 %

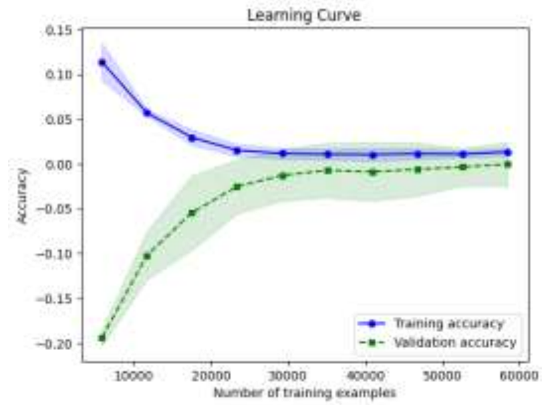




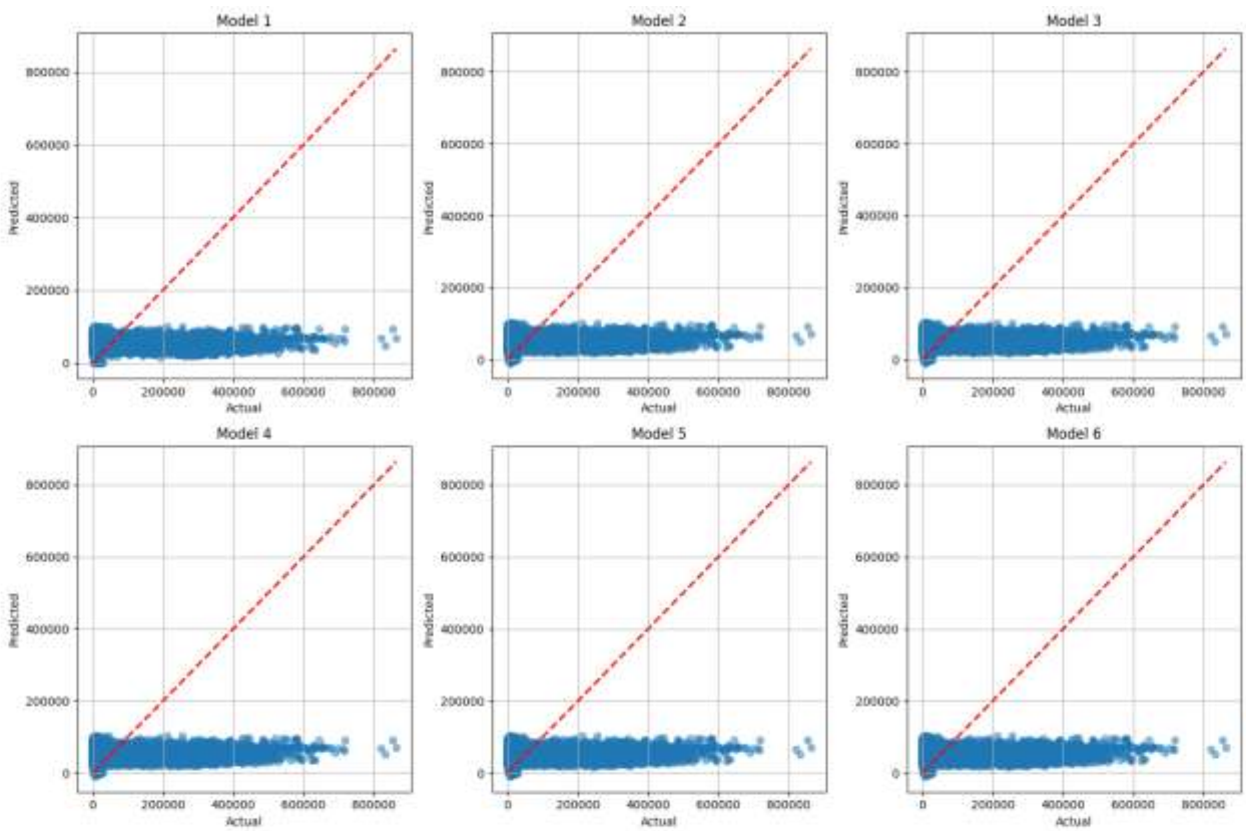








Plot the Results in Graph



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']
```

```python
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')

```python
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)
```

```python
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()
```

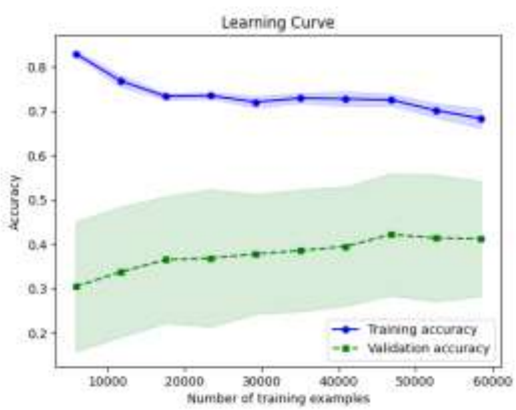
```
Mean Squared Error  :  2089034810.02
R-squared Value     :  0.8095577049332217
Accuracy Percentage :  80.96 %
```

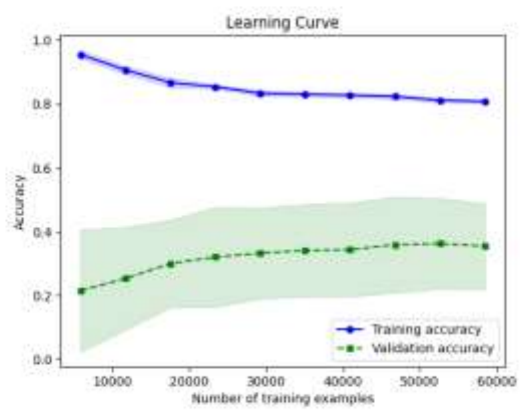
Random Forest Regression Model6 Using LDA:

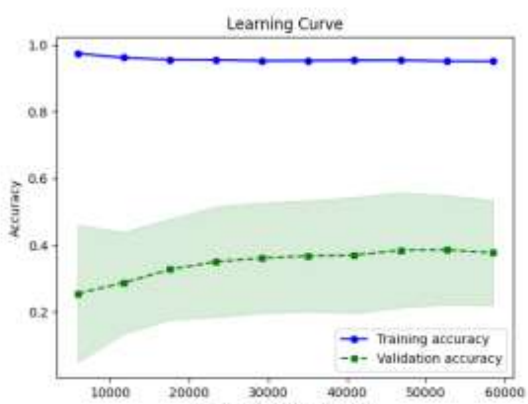
```python
x = pd.DataFrame(x_lda)
y = df['totvotpoll']

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)
```

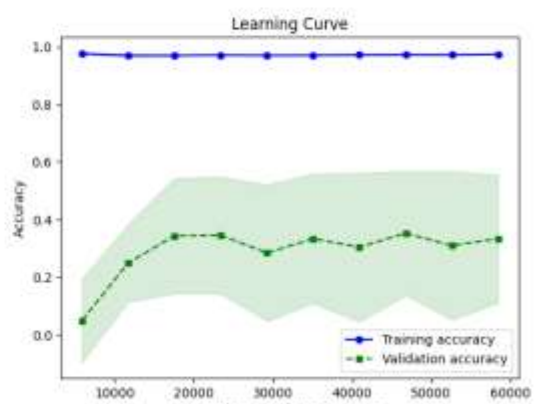
```python
y_pred = model.predict(X_test)

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,),"%")
```

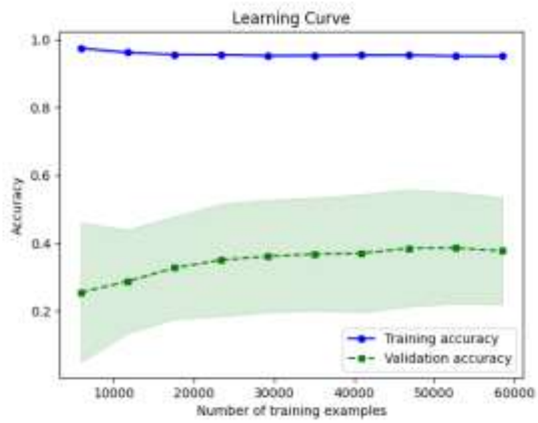
```
Mean Squared Error  :  3847991575.76
R-squared Value     :  0.6492062537338011
Accuracy Percentage :  64.92 %
```

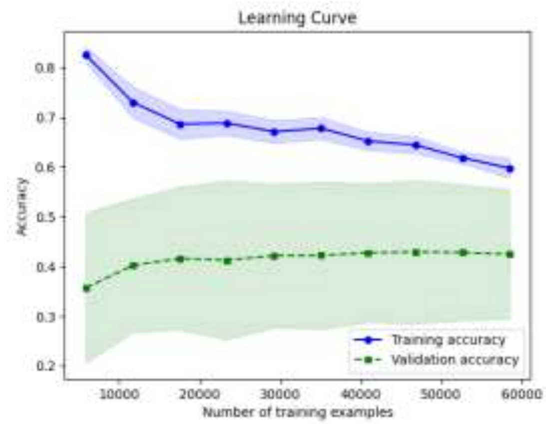




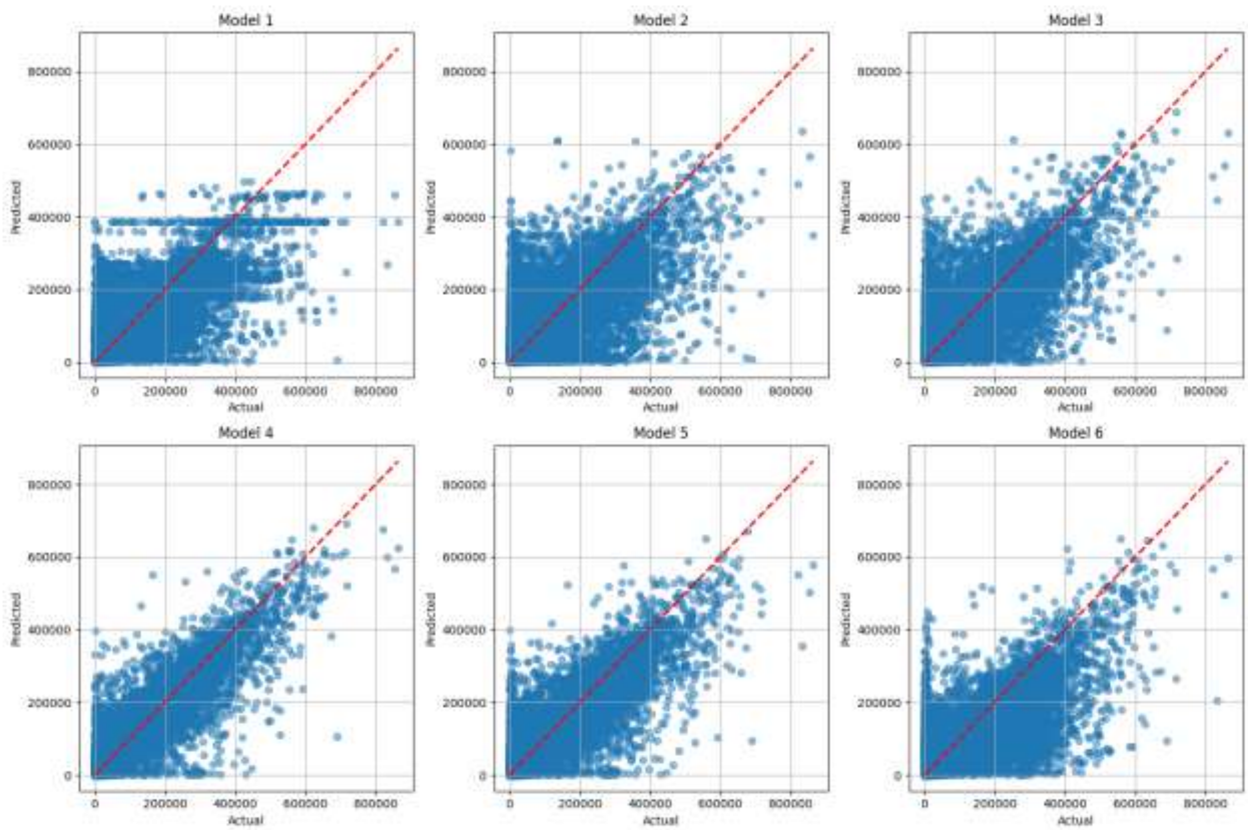




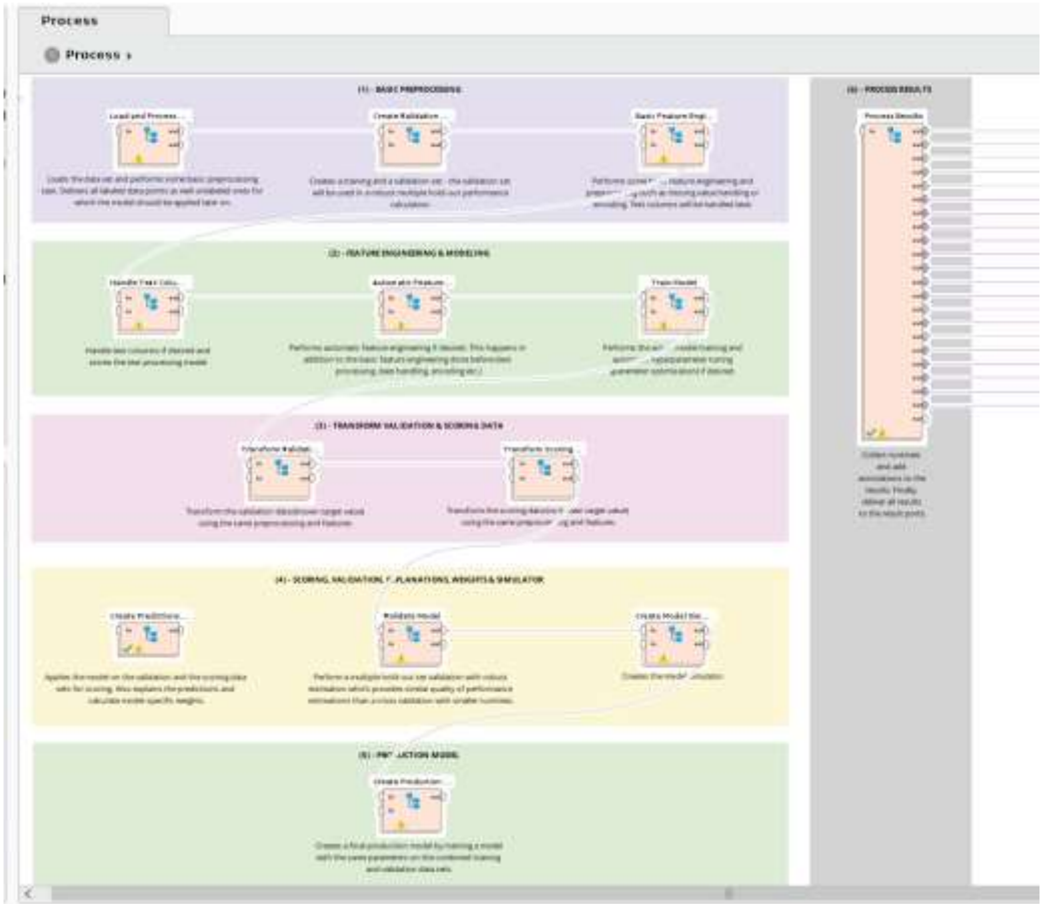


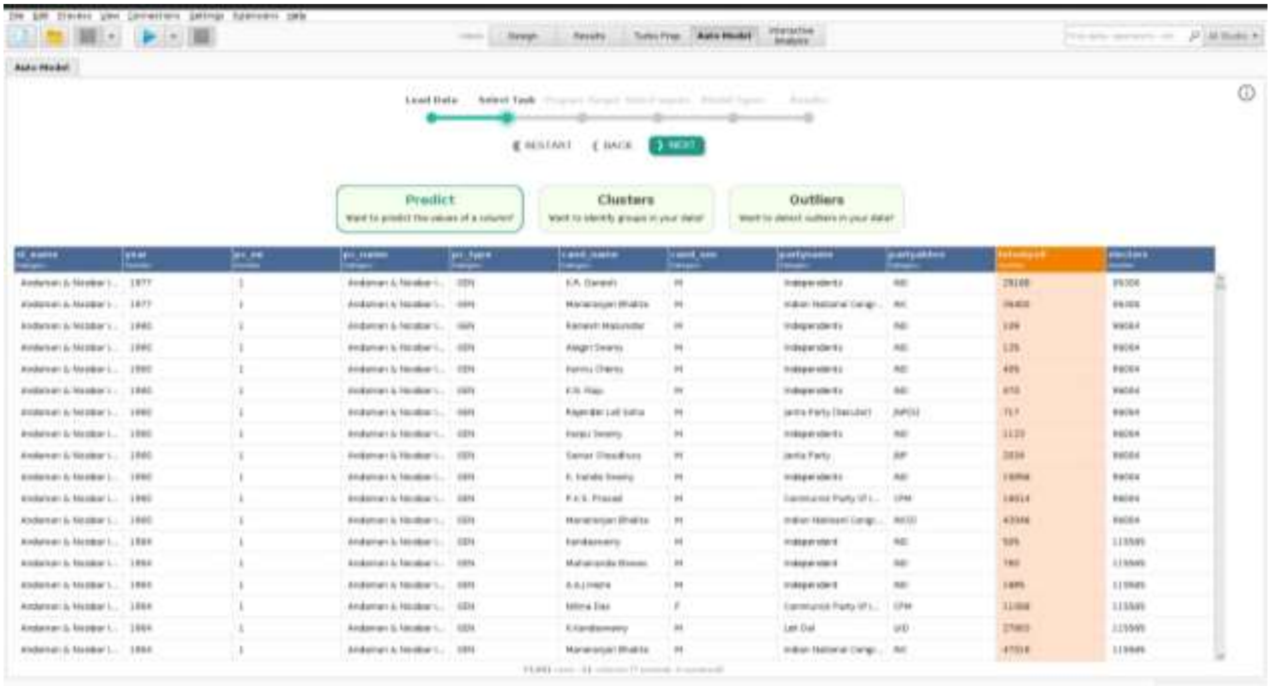


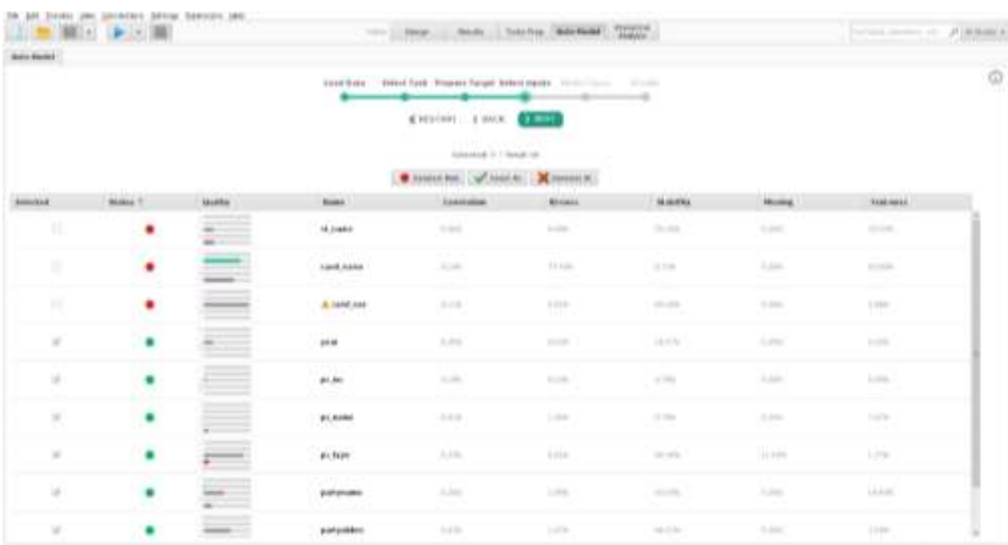
Plot the results using Graph



Model Deployment in Rapid Miner

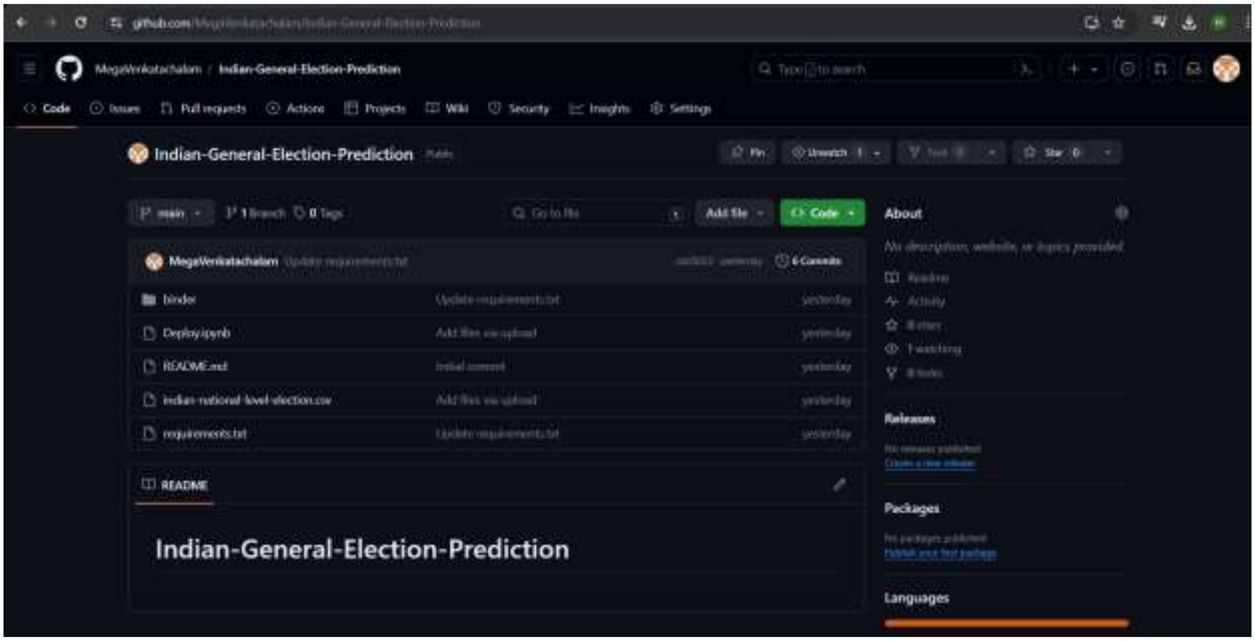



Predict
Clusters
Outliers



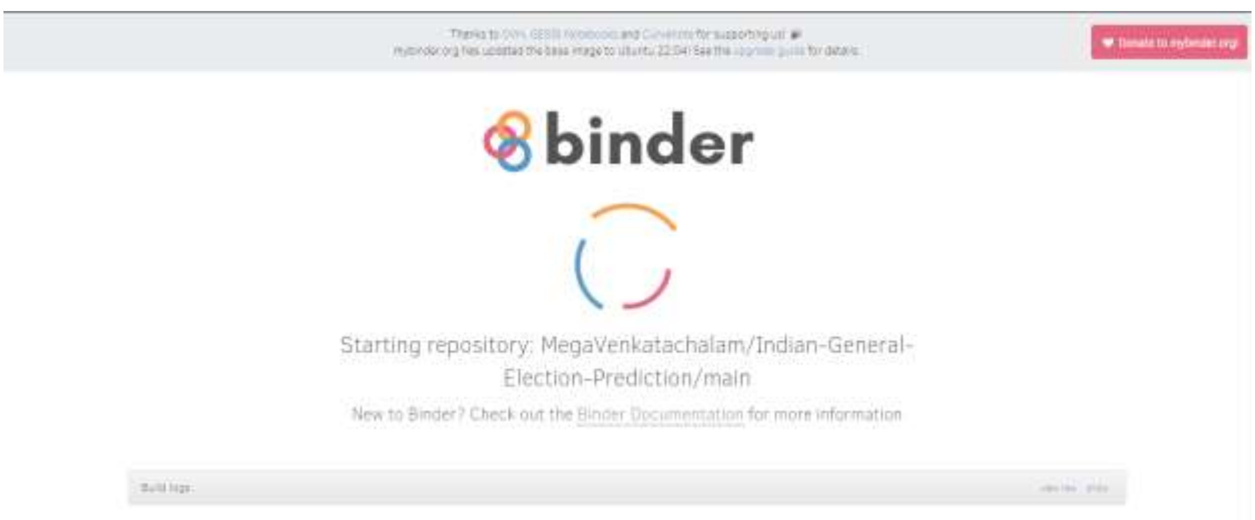
Model Deploy in Binder

Upload all codes in GitHub

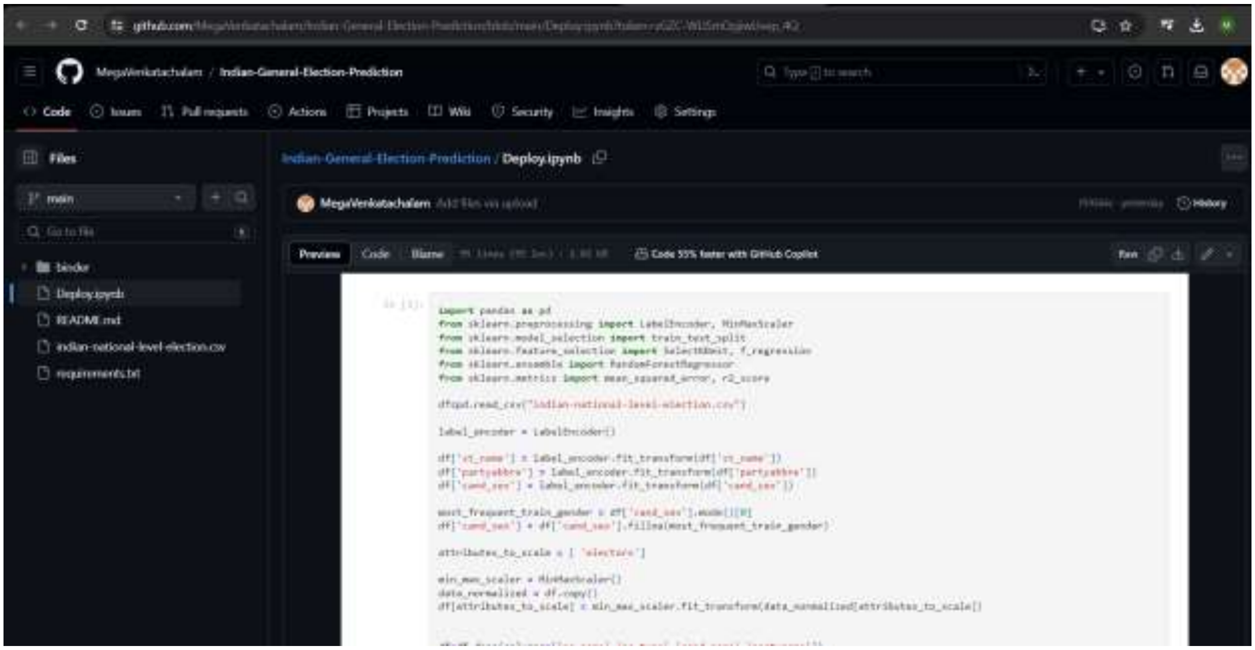


Deployment

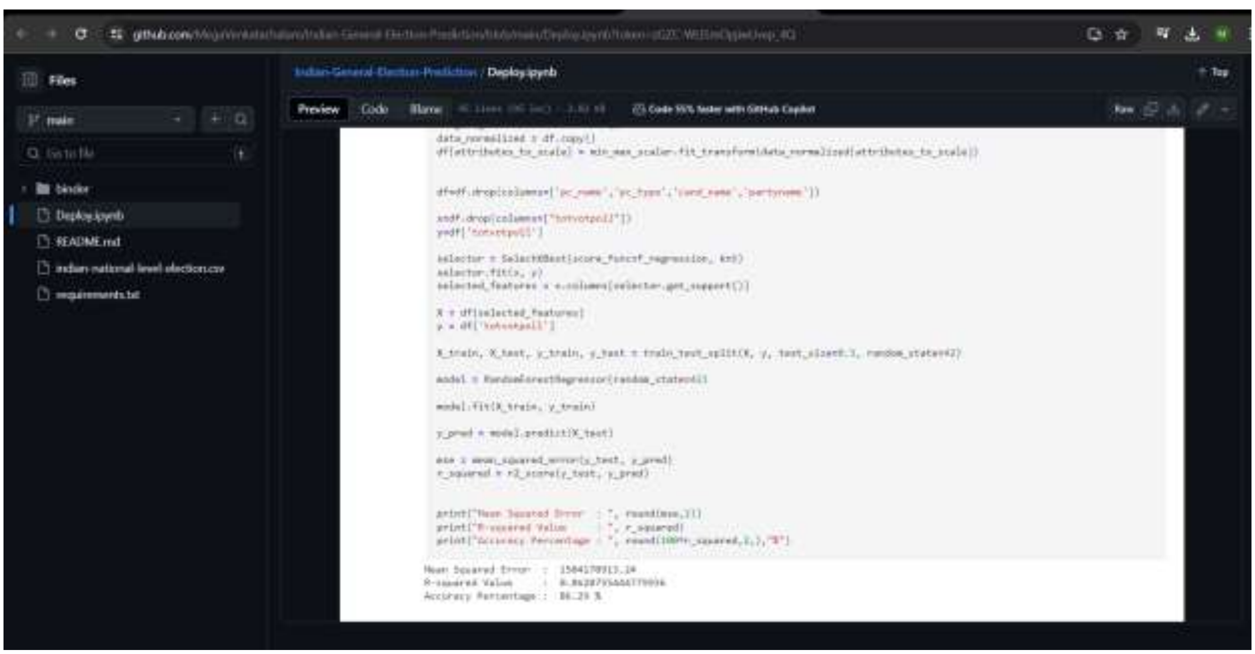
https://mybinder.org/v2/gh/MegaVenkatachalam/Indian-General-Election-Prediction/main?urlpath=https%3A%2F%2Fgithub.com%2FMegaVenkatachalam%2FIndian-General-Election-Prediction%2Fblob%2Fmain%2FDeploy.ipynb



After Deploy The Model See The Results in GitHub



Model results in Github



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

Github Link For Project

https://github.com/MegaVenkatachalam/Indian-General-Election-Prediction