```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
#Importing warnings
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

#reading csv file
data = pd.read_csv('/content/drive/MyDrive/new_election_dataset (1).csv')
data.head()
```

	TimeElapsed	time	territoryName	totalMandates	availableMandates	numParishes	n
0	0	8	0	0	16	147	
1	0	8	0	0	16	147	
2	0	8	0	0	16	147	
3	0	8	0	0	16	147	
4	0	8	0	0	16	147	

5 rows × 29 columns

```
# Preparing Data set
# dropping Final Mandate variable from X
#assign the value of y for training
x = data.drop(columns=['FinalMandates'])
y = data[["FinalMandates"]]
y = y.values.reshape(-1, 1)  # Reshape to make it a 2D array
```

```
#Standardizing value of x by using standardscalar to make the data normally distributed
sc = StandardScaler()
a = sc.fit_transform(x)
df_new_x = pd.DataFrame(a,columns=x.columns)
df_new_x.head()
```

	TimeElapsed	time	territoryName	totalMandates	availableMandates	numParishe
0	-1.752045	-1.206238	-1.741356	-0.767282	0.979379	-0.09922
1	-1.752045	-1.206238	-1.741356	-0.767282	0.979379	-0.09922
2	-1.752045	-1.206238	-1.741356	-0.767282	0.979379	-0.09922
3	-1.752045	-1.206238	-1.741356	-0.767282	0.979379	-0.09922
4	-1.752045	-1.206238	-1.741356	-0.767282	0.979379	-0.09922

5 rows × 28 columns

```
#Splitting the data into training and testing data
x_train,x_test,y_train,y_test=train_test_split(df_new_x,y,test_size=0.3,random_state=45)
```

The top model from step-1 is Random Forest Regressor.

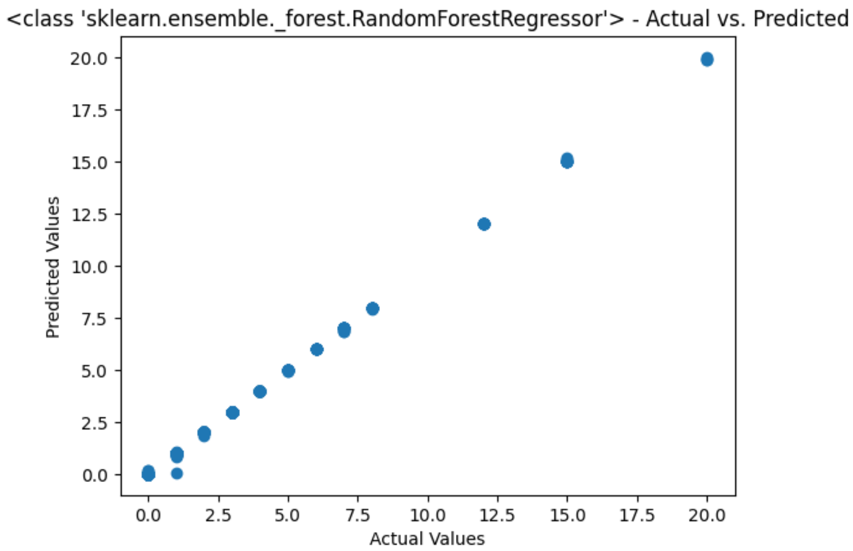
```
model= RandomForestRegressor(random_state=42)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
print(f"{RandomForestRegressor} - MSE: {mse:.4f}, MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2 Score: {rmse:.4f}")
```

```
<class 'sklearn.ensemble._forest.RandomForestRegressor'> - MSE: 0.0002, MAE: 0.0009, RMSE: 0.0150, R2 Score: 0.0150
```

```
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title(f'{RandomForestRegressor} - Actual vs. Predicted')
plt.show()
```



The scatter plot depicts that the actual values alligned correctly with the predicted values.

XGB Regressor without Hyper parameter Tuning

```
xgb_before_tuning = XGBRegressor(random_state=42)
xgb_before_tuning.fit(x_train, y_train)
y_pred_before = xgb_before_tuning.predict(x_test)
mse_before_tuning_XGB = mean_squared_error(y_test, y_pred_before)
```

```
mse = mean_squared_error(y_test, y_pred_before)
mae = mean_absolute_error(y_test, y_pred_before)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_before)
```

```
print(f"{XGBRegressor} - MSE: {mse:.4f}, MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2 Score: {rmse:.4f}")
```

```
<class 'xgboost.sklearn.XGBRegressor'> - MSE: 0.0005, MAE: 0.0019, RMSE: 0.0228, R2 Score: 0.0228
```

```
plt.scatter(y_test, y_pred_before)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title(f'{XGBRegressor} - Actual vs. Predicted')
plt.show()
```

The scatter plot depicts that there is some misalignment between the actual and predicted values.

Extreme Machine Learning Model

```
class ELMRegressor:
    def __init__(self, n_input, n_hidden, activation_function=np.tanh):
        self.n_input = n_input
        self.n_hidden = n_hidden
        self.activation_function = activation_function
        self.weights_input_hidden = None
        self.bias_hidden = None
        self.weights_hidden_output = None

    def _initialize_weights(self):
        self.weights_input_hidden = np.random.rand(self.n_input, self.n_hidden)
        self.bias_hidden = np.random.rand(1, self.n_hidden)
        self.weights_hidden_output = np.random.rand(self.n_hidden, 1)

    def _activation(self, x):
        return self.activation_function(x)

    def train(self, X, y):
        self._initialize_weights()

        # Calculate hidden layer output
        hidden_output = self._activation(np.dot(X, self.weights_input_hidden) + self.bias_hidden)

        # Calculate output layer weights using the Moore-Penrose pseudoinverse
        self.weights_hidden_output = np.dot(np.linalg.pinv(hidden_output), y)

    def predict(self, X):
        hidden_output = self._activation(np.dot(X, self.weights_input_hidden) + self.bias_hidden)
        output = np.dot(hidden_output, self.weights_hidden_output)
        return output
    def mse(self, y_true, y_pred):
        return np.mean((y_true - y_pred) ** 2)
```

```
elm = ELMRegressor(n_input = x_train.shape[1], n_hidden=50)
elm.train(x_train, y_train)
```

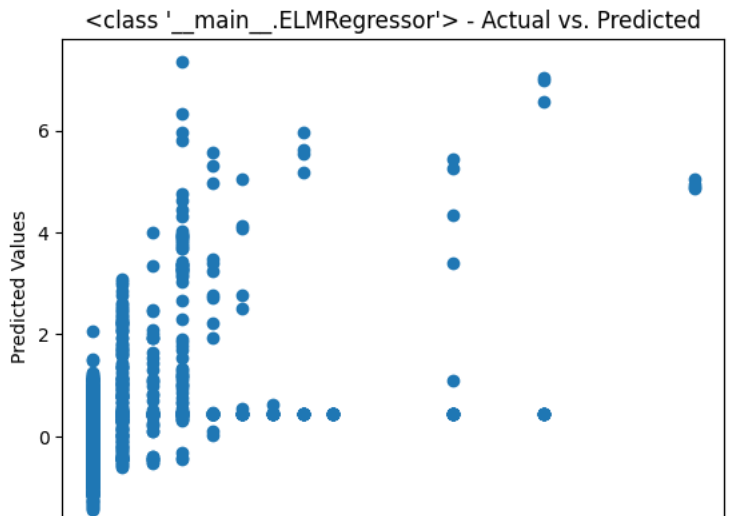
```
y_pred_train_ELM = elm.predict(x_train)
y_pred_test_ELM = elm.predict(x_test)
```

```
mse = mean_squared_error(y_test, y_pred_test_ELM)
mae = mean_absolute_error(y_test, y_pred_test_ELM)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_test_ELM)
```

```
print(f"{ELMRegressor} - MSE: {mse:.4f}, MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2 Score: {rmse:.4f}")
```

```
<class '__main__.ELMRegressor'> - MSE: 1.8147, MAE: 0.6146, RMSE: 1.3471, R2 Score: 1.3471
```

```
plt.scatter(y_test, y_pred_test_ELM)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title(f'{ELMRegressor} - Actual vs. Predicted')
plt.show()
```



The scatter plot depicts that significant observations misaligned between the actual and predicted values.

A Basic Deep Learning Model with two layers

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Build the model
model = Sequential([
    # Input layer (specify the input shape for the first layer)
    Dense(units=10, activation='relu', input_shape=(x_train.shape[1],)),

    # Hidden layer
    Dense(units=5, activation='relu'),

    # Output layer
    Dense(units=1, activation='linear')  # Assuming it's a regression task
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')  # Assuming it's a regression task

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                290

 dense_1 (Dense)             (None, 5)                 55

 dense_2 (Dense)             (None, 1)                 6

=================================================================
Total params: 351 (1.37 KB)
Trainable params: 351 (1.37 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

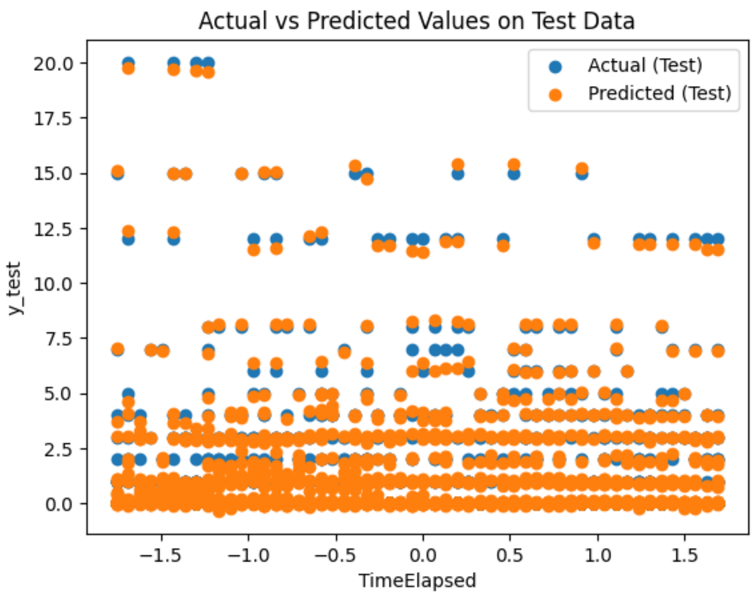
```
# Train the model
history = model.fit(x_train, y_train, epochs=100, batch_size=32, verbose=0)

# Make predictions on training and test data
y_pred_train = model.predict(x_train).flatten()
y_pred_test = model.predict(x_test).flatten()

mse = mean_squared_error(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_test)
```

```
402/402 [==============================] - 1s 1ms/step
172/172 [==============================] - 0s 2ms/step
```

```
print(f"MSE: {mse:.4f}, MAE: {mae:.4f}, RMSE: {rmse:.4f}, R2 Score: {rmse:.4f}")
```

```
MSE: 0.0070, MAE: 0.0248, RMSE: 0.0839, R2 Score: 0.0839
```

```
# Plot actual vs predicted values on test data
plt.scatter(x_test.iloc[:, 0], y_test, label='Actual (Test)')
plt.scatter(x_test.iloc[:, 0], y_pred_test, label='Predicted (Test)')
plt.xlabel(x_test.columns[0])
plt.ylabel('y_test')
plt.legend()
plt.title('Actual vs Predicted Values on Test Data')
plt.show()
```



An Ensemble model containing the top 3 models overall

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
# Train XGBoost model
xgb_model = XGBRegressor()
xgb_model.fit(x_train, y_train)
y_pred_xgb = xgb_model.predict(x_test)

# Train Extreme Learning Machine (ELM) model
elm_model = ELMRegressor(n_input = x_train.shape[1], n_hidden=50)
elm_model.train(x_train, y_train)
y_pred_elm = elm_model.predict(x_test)

# Train basic deep learning model with two layers using TensorFlow and Keras
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(x_train.shape[1],)),
    layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=10, batch_size=32, verbose=0)
y_pred_dl = model.predict(x_test).flatten()

# Ensure dimensions match by reshaping
y_pred_xgb = y_pred_xgb.reshape(-1)
y_pred_elm = y_pred_elm.reshape(-1)
y_pred_dl = y_pred_dl.flatten()

# Create ensemble predictions
ensemble_predictions = (y_pred_xgb + y_pred_elm + y_pred_dl) / 3.0

# Calculate ensemble mean squared error
ensemble_mse = mean_squared_error(y_test, ensemble_predictions)
```

```
172/172 [==============================] - 0s 1ms/step
```

```
print("MSE on an Ensemble Model:", ensemble_mse)
```

```
MSE on an Ensemble Model: 0.24346072831973858
```