Importing the libraries

- *Models to be used are bagging regressor, extra tree regressor, voting regressor and random forest regressor on the dataset to find the best model.*

```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
%matplotlib inline
import pymongo
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import BaggingRegressor, ExtraTreesRegressor,
VotingRegressor, RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score



import warnings
warnings.filterwarnings('ignore')
```

Loading data from mongoDB

```
# Creating connection
try:
    client =
pymongo.MongoClient("mongodb+srv://ineuron:Project1@cluster0.rp4qzrr.m
ongodb.net/?retryWrites=true&w=majority")
    print("Connection to MongoDB server is successful.")
except Exception as e:
    print("Error is: ", e)
else:

    # Fetching data
    db = client.ml_algo
    collection = db.power_consumption_data

    try:
        # Creating dataframe
        df = pd.DataFrame(list(collection.find()))
    except Exception as e:
        print("Error is: ", e)
    else:
        df.drop(['_id'],axis = 1,inplace = True)
finally:
    print("\nDataframe created successfully.\n")
```

```
Connection to MongoDB server is successful.

Dataframe created successfully.
```

```python
# Checking the dataframe

df
```

```
       month  Global_active_power  Global_reactive_power  Voltage  \
0          1                1.376                  0.080   239.34
1          3                1.384                  0.096   242.14
2         11                0.224                  0.000   244.44
3          4                0.370                  0.128   244.31
4         12                0.216                  0.000   244.75
...      ...                  ...                    ...      ...
49387      3                1.734                  0.000   244.87
49388      2                0.534                  0.138   242.03
49389      9                1.814                  0.224   237.71
49390      3                0.318                  0.096   240.26
49391      4                0.406                  0.202   240.51

       Global_intensity  Total_energy_consumed
0                   5.6                   18.0
1                   5.6                   19.0
2                   0.8                    1.0
3                   1.6                    1.0
4                   1.0                    0.0
...                 ...                    ...
49387               7.0                   19.0
49388               2.2                    2.0
49389               7.6                   18.0
49390               1.4                    2.0
49391               1.8                    1.0

[49392 rows x 6 columns]
```

5. Feature Engineering

5.1 Spliting the data into train and test data

```python
X_train, X_test, y_train, y_test =
train_test_split(df.drop('Total_energy_consumed', axis=1),

df['Total_energy_consumed'],
                                                    test_size=0.33,
                                                    random_state=42)

X_train.shape, X_test.shape
```

```
((33092, 5), (16300, 5))
```

Observations:

- So now we have 33092 rows for training and 16300 for test datasets.

5.2 Standardizing or feature scaling the dataset

- Although there is no need for standardization as we are mainly going to use **Decision Trees** for solving the problem.

```
scaler = StandardScaler()
scaler
```

```
StandardScaler()
```

```
# Creating the scale by training with train data and then save it to use in future

scale = scaler.fit(X_train)
print(scale.mean_)
```

```
[6.42880454e+00 1.05362205e+00 1.22219932e-01 2.40885507e+02
 4.44457271e+00]
```

Saving the `scale` to use it later to transform the data and predict the values

```
# To save a Standard scaler object


with open('scaled.pkl', 'wb') as f:
    pickle.dump(scale, f)

# Loading the scaled object to transform the data

with open('scaled.pkl', 'rb') as f:
    scaled = pickle.load(f)

# Now transforming the train and test dataset

X_train_tf = scaled.transform(X_train)
X_test_tf = scaled.transform(X_test)
```

6. Model Building

6.1 Create a Function to evaluate all the models

```
def evaluate_model(actual, predicted, X_test_tf):
    mae = mean_absolute_error(actual, predicted)
    mse = mean_squared_error(actual, predicted)
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    r2_square = r2_score(actual, predicted)
    adj_r2 = 1 - (1 - r2_square)*(len(actual)-1)/(len(actual) -
X_test_tf.shape[1] - 1)
    return mae, rmse, r2_square, adj_r2
```

```
models = {
    "Bagging Regressor": BaggingRegressor(),
    "Extra Tree Regressor": ExtraTreesRegressor(),
    "Random Forest Regressor": RandomForestRegressor()
}

model_list = []
r2_list =[]
adj_r2_list = []

for i in range(len(list(models))):
    model = list(models.values())[i]

    # Train model
    model.fit(X_train_tf, y_train)

    # Make predictions
    y_train_pred = model.predict(X_train_tf)
    y_test_pred = model.predict(X_test_tf)

    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2,
model_train_adjusted_r2 = evaluate_model(y_train, y_train_pred,
X_test_tf)

    model_test_mae , model_test_rmse, model_test_r2,
model_test_adjusted_r2 = evaluate_model(y_test, y_test_pred,
X_test_tf)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('\nModel performance for Training set')
    print(f"- Root Mean Squared Error: {model_train_rmse:.4f}")
    print(f"- Mean Absolute Error: {model_train_mae:.4f}")
    print(f"- R2 Score: {model_train_r2:.4f}".format())
    print(f"- Adjusted R2 Score: {model_train_adjusted_r2:.4f}")

    print('----------------------------------')

    print('Model performance for Test set')
    print(f"- Root Mean Squared Error: {model_test_rmse:.4f}")
    print(f"- Mean Absolute Error: {model_test_mae:.4f}")
    print(f"- R2 Score: {model_test_r2:.4f}")
    print(f"- Adjusted R2 Score: {model_test_adjusted_r2:.4f}")
    r2_list.append(model_test_r2)
    adj_r2_list.append(model_test_adjusted_r2)
```

```
    print('='*50)
    print('\n')
```

```
Bagging Regressor

Model performance for Training set
- Root Mean Squared Error: 2.3730
- Mean Absolute Error: 1.1251
- R2 Score: 0.9560
- Adjusted R2 Score: 0.9559
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.5926
- Mean Absolute Error: 2.9170
- R2 Score: 0.7566
- Adjusted R2 Score: 0.7565
==================================================


Extra Tree Regressor

Model performance for Training set
- Root Mean Squared Error: 0.3346
- Mean Absolute Error: 0.0094
- R2 Score: 0.9991
- Adjusted R2 Score: 0.9991
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.5411
- Mean Absolute Error: 2.8260
- R2 Score: 0.7610
- Adjusted R2 Score: 0.7610
==================================================


Random Forest Regressor

Model performance for Training set
- Root Mean Squared Error: 2.0600
- Mean Absolute Error: 1.0602
- R2 Score: 0.9668
- Adjusted R2 Score: 0.9668
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3849
- Mean Absolute Error: 2.8269
- R2 Score: 0.7743
- Adjusted R2 Score: 0.7743
==================================================
```

Now doing the same for `Voting Regressor` using the other 3 regression models

```
r1 = BaggingRegressor()
r2 = ExtraTreesRegressor()
r3 = RandomForestRegressor()

Vt_reg = VotingRegressor([('Bg_regr', r1), ('Et_regr', r2),
('Rf_regr', r3)])

Vt_reg.fit(X_train_tf, y_train)
```

```
VotingRegressor(estimators=[('Bg_regr', BaggingRegressor()),
                            ('Et_regr', ExtraTreesRegressor()),
                            ('Rf_regr', RandomForestRegressor())])
```

```
y_train_pred_vt = Vt_reg.predict(X_train_tf)
y_test_pred_vt = Vt_reg.predict(X_test_tf)

# Evaluate Train and Test dataset
model_train_mae , model_train_rmse, model_train_r2,
model_train_adjusted_r2 = evaluate_model(y_train, y_train_pred_vt,
X_test_tf)
model_test_mae , model_test_rmse, model_test_r2,
model_test_adjusted_r2 = evaluate_model(y_test, y_test_pred_vt,
X_test_tf)


print("Voting Regressor")
model_list.append("Voting Regressor")

print('\nModel performance for Training set')
print(f"- Root Mean Squared Error: {model_train_rmse:.4f}")
print(f"- Mean Absolute Error: {model_train_mae:.4f}")
print(f"- R2 Score: {model_train_r2:.4f}".format())
print(f"- Adjusted R2 Score: {model_train_adjusted_r2:.4f}")

print('----------------------------------')

print('Model performance for Test set')
print(f"- Root Mean Squared Error: {model_test_rmse:.4f}")
print(f"- Mean Absolute Error: {model_test_mae:.4f}")
print(f"- R2 Score: {model_test_r2:.4f}")
print(f"- Adjusted R2 Score: {model_test_adjusted_r2:.4f}")
r2_list.append(model_test_r2)
adj_r2_list.append(model_test_adjusted_r2)
```

```
Voting Regressor
```

```
Model performance for Training set
- Root Mean Squared Error: 1.4454
- Mean Absolute Error: 0.7220
- R2 Score: 0.9837
- Adjusted R2 Score: 0.9837
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3997
- Mean Absolute Error: 2.8247
- R2 Score: 0.7731
- Adjusted R2 Score: 0.7730
```

```
pd.DataFrame(list(zip(model_list, r2_list, adj_r2_list)),
             columns=['Model Name', 'R2_Score', 'Adjusted
R2_Score']).sort_values(by=["R2_Score"], ascending=False)
```

```
                Model Name  R2_Score  Adjusted R2_Score
2  Random Forest Regressor  0.774330           0.774261
3        Voting Regressor  0.773089           0.773020
1     Extra Tree Regressor  0.761050           0.760976
0       Bagging Regressor  0.756587           0.756513
```

Observations:

- All the regression models are showing **Overfitting** condition.

6.2 Hyper Parameter Tuning (using GridSearchCV)

Here we will use the `Bagging Regressor model`, `Extra Tree model` and `Random Forest model`

```
# Initialize the parameters

bg_params = {"n_estimators": [100, 200, 500, 1000],
             "max_features": [5, 7, "auto", 8]}

et_params = {"max_depth": [5, 8, 15, None, 10],
             "max_features": [5, 7, "auto", 8],
             "min_samples_split": [2, 8, 15, 20],
             "n_estimators": [100, 200, 500, 1000]}

rf_params = {"max_depth": [5, 8, 15, None, 10],
             "max_features": [5, 7, "auto", 8],
             "min_samples_split": [2, 8, 15, 20],
             "n_estimators": [100, 200, 500, 1000]}

# Models

gridcv_models = [('BGR', BaggingRegressor(), bg_params),
                 ('ETR', ExtraTreesRegressor(), et_params),
```

```
                 ('RF', RandomForestRegressor(), rf_params)
                ]

model_param = {}

for name, model, params in gridcv_models:
    grid = GridSearchCV(estimator=model,
                        param_grid=params,
                        cv=3,
                        verbose=2,
                        n_jobs=-1)

    grid.fit(X_train_tf, y_train)
    model_param[name] = grid.best_params_

for model_name in model_param:
    print(f"---------------- Best Params for {model_name}
-------------------")
    print(model_param[model_name])
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 320 candidates, totalling 960 fits
Fitting 3 folds for each of 320 candidates, totalling 960 fits
---------------- Best Params for BGR -------------------
{'max_features': 5, 'n_estimators': 1000}
---------------- Best Params for ETR -------------------
{'max_depth': 15, 'max_features': 5, 'min_samples_split': 20,
'n_estimators': 500}
---------------- Best Params for RF -------------------
{'max_depth': 15, 'max_features': 8, 'min_samples_split': 20,
'n_estimators': 1000}
```

Retraining the Models with best Parameters

```
models = {
    "Bagging Regressor": BaggingRegressor(max_features=5,
n_estimators=1000),
    "Extra Tree Regressor": ExtraTreesRegressor(max_depth=15,
max_features=5, min_samples_split=20, n_estimators=500),
    "Random Forest Regressor": RandomForestRegressor(max_depth=15,
max_features=8, min_samples_split=20, n_estimators=1000)
}

model_list = []
r2_list =[]
adj_r2_list = []

for i in range(len(list(models))):
    model = list(models.values())[i]

    # Train model
```

```python
    model.fit(X_train_tf, y_train)

    # Make predictions
    y_train_pred = model.predict(X_train_tf)
    y_test_pred = model.predict(X_test_tf)

    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2,
model_train_adjusted_r2 = evaluate_model(y_train, y_train_pred,
X_test_tf)

    model_test_mae , model_test_rmse, model_test_r2,
model_test_adjusted_r2 = evaluate_model(y_test, y_test_pred,
X_test_tf)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('\nModel performance for Training set')
    print(f"- Root Mean Squared Error: {model_train_rmse:.4f}")
    print(f"- Mean Absolute Error: {model_train_mae:.4f}")
    print(f"- R2 Score: {model_train_r2:.4f}".format())
    print(f"- Adjusted R2 Score: {model_train_adjusted_r2:.4f}")

    print('----------------------------------')

    print('Model performance for Test set')
    print(f"- Root Mean Squared Error: {model_test_rmse:.4f}")
    print(f"- Mean Absolute Error: {model_test_mae:.4f}")
    print(f"- R2 Score: {model_test_r2:.4f}")
    print(f"- Adjusted R2 Score: {model_test_adjusted_r2:.4f}")
    r2_list.append(model_test_r2)
    adj_r2_list.append(model_test_adjusted_r2)

    print('='*50)
    print('\n')
```

```
Bagging Regressor

Model performance for Training set
- Root Mean Squared Error: 2.0267
- Mean Absolute Error: 1.0506
- R2 Score: 0.9679
- Adjusted R2 Score: 0.9679
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3647
- Mean Absolute Error: 2.8207
- R2 Score: 0.7760
```

```
- Adjusted R2 Score: 0.7760
==================================================


Extra Tree Regressor

Model performance for Training set
- Root Mean Squared Error: 4.6010
- Mean Absolute Error: 2.4936
- R2 Score: 0.8344
- Adjusted R2 Score: 0.8344
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.1921
- Mean Absolute Error: 2.8247
- R2 Score: 0.7902
- Adjusted R2 Score: 0.7901
==================================================


Random Forest Regressor

Model performance for Training set
- Root Mean Squared Error: 4.3765
- Mean Absolute Error: 2.3650
- R2 Score: 0.8502
- Adjusted R2 Score: 0.8502
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.2406
- Mean Absolute Error: 2.8218
- R2 Score: 0.7863
- Adjusted R2 Score: 0.7862
==================================================
```

Again doing for `VotingRegressor` model

```
r1 = BaggingRegressor(max_features=5, n_estimators=1000)
r2 = ExtraTreesRegressor(max_depth=15, max_features=5,
min_samples_split=20, n_estimators=500)
r3 = RandomForestRegressor(max_depth=15, max_features=8,
min_samples_split=20, n_estimators=1000)

Vt_reg = VotingRegressor([('Bg_regr', r1), ('Et_regr', r2),
('Rf_regr', r3)])

Vt_reg.fit(X_train_tf, y_train)
```

```
VotingRegressor(estimators=[('Bg_regr',
                             BaggingRegressor(max_features=5,
                                              n_estimators=1000)),
                            ('Et_regr',
                             ExtraTreesRegressor(max_depth=15,
max_features=5,
                                                 min_samples_split=20,
                                                 n_estimators=500)),
                            ('Rf_regr',
                             RandomForestRegressor(max_depth=15,
max_features=8,

min_samples_split=20,

n_estimators=1000))])
```

```
y_train_pred_vt = Vt_reg.predict(X_train_tf)
y_test_pred_vt = Vt_reg.predict(X_test_tf)

# Evaluate Train and Test dataset
model_train_mae , model_train_rmse, model_train_r2,
model_train_adjusted_r2 = evaluate_model(y_train, y_train_pred_vt,
X_test_tf)
model_test_mae , model_test_rmse, model_test_r2,
model_test_adjusted_r2 = evaluate_model(y_test, y_test_pred_vt,
X_test_tf)


print("Voting Regressor")
model_list.append("Voting Regressor")

print('\nModel performance for Training set')
print(f"- Root Mean Squared Error: {model_train_rmse:.4f}")
print(f"- Mean Absolute Error: {model_train_mae:.4f}")
print(f"- R2 Score: {model_train_r2:.4f}".format())
print(f"- Adjusted R2 Score: {model_train_adjusted_r2:.4f}")

print('----------------------------------')

print('Model performance for Test set')
print(f"- Root Mean Squared Error: {model_test_rmse:.4f}")
print(f"- Mean Absolute Error: {model_test_mae:.4f}")
print(f"- R2 Score: {model_test_r2:.4f}")
print(f"- Adjusted R2 Score: {model_test_adjusted_r2:.4f}")
r2_list.append(model_test_r2)
adj_r2_list.append(model_test_adjusted_r2)
```

```
Voting Regressor

Model performance for Training set
```

```
- Root Mean Squared Error: 3.6333
- Mean Absolute Error: 1.9607
- R2 Score: 0.8967
- Adjusted R2 Score: 0.8967
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.2155
- Mean Absolute Error: 2.8064
- R2 Score: 0.7883
- Adjusted R2 Score: 0.7882
```

```
pd.DataFrame(list(zip(model_list, r2_list, adj_r2_list)),
             columns=['Model Name', 'R2_Score', 'Adjusted
R2_Score']).sort_values(by=["R2_Score"], ascending=False)
```

```
                Model Name  R2_Score  Adjusted R2_Score
1     Extra Tree Regressor  0.790199           0.790135
3         Voting Regressor  0.788307           0.788242
2  Random Forest Regressor  0.786260           0.786195
0        Bagging Regressor  0.776022           0.775953
```

Observations:

- All the regression models are still showing **Overfitting** condition.
- But out of that the **Extra Tree Regressor** model is giving the best result. It also has less **Overfitting** problem than the other three models.
- So we are going to save this model for future use on this problem.

```
ETR = ExtraTreesRegressor(max_depth=15, max_features=5,
min_samples_split=20, n_estimators=500)
ETR.fit(X_train_tf, y_train)
```

```
ExtraTreesRegressor(max_depth=15, max_features=5,
min_samples_split=20,
                    n_estimators=500)
```

```
# To save the model

try:
    with open('etr_model.pkl', 'wb') as f:
        pickle.dump(ETR, f)
except Exception as err:
    print(err)
else:
    print("Model saved successfully.")
```

```
Model saved successfully.
```

Needed to delete the model when uploading to github.