# 🚀 Gradient Boosting From Scratch – Regression

## ⚡ Application: Energy Consumption Forecasting

## 📌 Problem Statement

Electric power companies must accurately **forecast daily energy consumption** to ensure:

- grid stability
- optimal power generation
- reduced operational cost

Energy consumption depends on multiple interacting, non-linear factors such as:

- Ambient temperature
- Humidity levels
- Number of occupants
- Duration of daylight

### 🎯 Objective

Build a **Gradient Boosting Regressor from scratch** that:

- Predicts daily energy consumption (kWh)
- Minimizes **Mean Squared Error (MSE)**
- Learns incrementally by correcting residual errors

### 📥 Input Features

| Feature | Description |
|---|---|
| temperature | Average daily temperature (°C) |
| humidity | Average humidity (%) |
| occupancy | Number of people |
| daylight | Hours of daylight |

## 📤 Output

- Continuous numeric value representing **energy consumption (kWh)**

## 🔧 Common Utilities

In [1]:
```
import numpy as np
```

## 📊 Dataset Generation

In [2]:
```
np.random.seed(1)
n = 350

temperature = np.random.uniform(5, 40, n)
humidity = np.random.uniform(20, 90, n)
occupancy = np.random.randint(1, 6, n)
daylight = np.random.uniform(8, 14, n)

energy = (
    50
    + 1.8 * temperature
    + 0.6 * humidity
    + 12 * occupancy
    - 3 * daylight
    + np.random.normal(0, 10, n)
)

X = np.column_stack((temperature, humidity, occupancy, daylight))
y = energy
```

In [3]:
```
idx = np.random.permutation(n)
train, test = idx[:280], idx[280:]
X_train, X_test = X[train], X[test]
y_train, y_test = y[train], y[test]
```

## 🌱 Weak Learner: Regression Tree

In [4]:
```
class Node:
    def __init__(self, f=None, t=None, l=None, r=None, v=None):
        self.f, self.t, self.l, self.r, self.v = f, t, l, r, v
```

In [5]:
```
class RegressionTree:
    def __init__(self, max_depth=2):
        self.max_depth = max_depth

    def fit(self, X, y):
        self.nf = X.shape[1]
        self.root = self._grow(X, y, 0)

    def _grow(self, X, y, d):
        if d >= self.max_depth or len(y) < 2:
            return Node(v=np.mean(y))

        bf, bt, bl = None, None, float("inf")
        for f in range(self.nf):
            for t in np.unique(X[:, f]):
                l = y[X[:, f] <= t]
                r = y[X[:, f] > t]
                if len(l) == 0 or len(r) == 0:
                    continue
                loss = len(l)*np.var(l) + len(r)*np.var(r)
                if loss < bl:
                    bf, bt, bl = f, t, loss

        if bf is None:
            return Node(v=np.mean(y))

        m = X[:, bf] <= bt
        return Node(bf, bt,
                    self._grow(X[m], y[m], d+1),
                    self._grow(X[~m], y[~m], d+1))

    def _pred(self, x, n):
        if n.v is not None:
            return n.v
        return self._pred(x, n.l if x[n.f] <= n.t else n.r)

    def predict(self, X):
        return np.array([self._pred(x, self.root) for x in X])
```

## 🚀 Gradient Boosting Regressor (From Scratch)

In [6]:
```
class GradientBoostingRegressorScratch:
    def __init__(self, n_estimators=120, lr=0.1, depth=2):
        self.n_estimators, self.lr, self.depth = n_estimators, lr, depth
        self.trees = []

    def fit(self, X, y):
        self.init = np.mean(y)
        pred = np.full(len(y), self.init)

        for _ in range(self.n_estimators):
            residuals = y - pred
            t = RegressionTree(self.depth)
            t.fit(X, residuals)
            pred += self.lr * t.predict(X)
            self.trees.append(t)
    def predict(self, X):
        pred = np.full(len(X), self.init)
        for t in self.trees:
            pred += self.lr * t.predict(X)
        return pred
```

In [7]:
```
model = GradientBoostingRegressorScratch()
model.fit(X_train, y_train)

pred = model.predict(X_test)
rmse = np.sqrt(np.mean((pred - y_test)**2))
print("RMSE:", rmse)
```

```
RMSE: 12.62789164348103
```

## 🧪 Asset-Based Test Cases

These tests validate the **internal correctness** of the Gradient Boosting Regression pipeline.

In [8]:
```
# Asset 1: Initial prediction equals mean
assert np.isclose(model.init, np.mean(y_train))
print("✅ Initial prediction test passed")
```

```
✅ Initial prediction test passed
```

In [9]:
```
# Asset 2: Residuals decrease after boosting
baseline_pred = np.full(len(y_train), np.mean(y_train))
baseline_rmse = np.sqrt(np.mean((baseline_pred - y_train)**2))

boosted_rmse = np.sqrt(np.mean((model.predict(X_train) - y_train)**2))

assert boosted_rmse < baseline_rmse
print("✅ Residual reduction test passed")
```

```
✅ Residual reduction test passed
```

In [10]:
```
# Asset 3: Weak learner prediction shape
tree = RegressionTree()
tree.fit(X_train, y_train)
preds = tree.predict(X_test)
assert preds.shape[0] == X_test.shape[0]
print("✅ Weak learner shape test passed")
```

```
✅ Weak learner shape test passed
```

In [11]:
```
# Asset 4: Ensemble size check
assert len(model.trees) == model.n_estimators
print("✅ Ensemble size test passed")
```

```
✅ Ensemble size test passed
```

In [12]:
```
# Asset 5: Prediction continuity (no NaNs)
```

```python
assert not np.isnan(pred).any()
print("✅ Prediction continuity test passed")
```

```
✅ Prediction continuity test passed
```

## ✅ Conclusion

This notebook demonstrated:

- Gradient Boosting **Regression** from scratch
- Residual-based learning with squared loss
- Asset-based testing for internal validation