Big Data Project

Machine Learning Module

1 Introduction

This document presents the complete machine learning workflow implemented for the Big Data project. The goal of this module is to build predictive models that estimate the next hour's electricity consumption using the UCI Household Power Consumption dataset. The process includes data ingestion, preprocessing, feature engineering, model training, evaluation, visualization, and AutoML experimentation.

The methods demonstrated here reflect a standard machine learning pipeline suitable for time series prediction tasks.

---

2 Dataset Description

The dataset used is the **UCI Household Power Consumption** dataset, containing minute-level measurements of electricity usage across several variables. For this project, only the `Global_active_power` variable is required for prediction.

Key characteristics:

- Data granularity: one measurement per minute
- Time span: multiple years
- Contains missing and non-numeric values
- Date and Time fields must be merged for temporal analysis

The machine learning team is responsible for building hourly prediction models, so the raw data is aggregated to hourly averages.

---

3 Workflow Summary

The workflow consists of:

1. Dataset loading and cleaning
2. Timestamp creation and indexing
3. Resampling to hourly frequency
4. Feature engineering (hour, day of week, month)
5. Target creation (next hour's consumption)
6. Model training: Linear Regression, Random Forest, Gradient Boosting
7. Error metric computation using MAE and RMSE
8. Performance visualization using bar and line plots
9. H2O AutoML execution and leaderboard analysis

Each stage is described in detail in the following sections.

---

4 Data Processing Pipeline

4.1 Reading the raw dataset

```
df = pd.read_csv(file_path, sep=";", na_values="?", low_memory=False)
```

This loads the text file and converts all missing values represented by “?” into NaN. The `low_memory=False` flag ensures more reliable type inference.

4.2 Timestamp generation

```
df['timestamp'] = pd.to_datetime(df['Date'] + " " + df['Time'], dayfirst=True)
df = df.set_index('timestamp')
```

Purpose:

- Merge separate date and time columns
- Convert to a datetime index
- Allow resampling and temporal feature extraction

4.3 Numeric conversion

```
df['Global_active_power'] = pd.to_numeric(df['Global_active_power'], errors='coerce')
```

Many values are non-numeric. Setting `errors='coerce'` forces invalid values to NaN, which allows mathematical operations later.

4.4 Resampling to hourly averages

```
hourly = df['Global_active_power'].resample('h').mean()
```

This aggregates the raw minute-level data to one averaged value per hour. Hourly data is required for model accuracy evaluation across time periods.

4.5 Feature engineering

```
hourly_df = hourly.to_frame(name='consumption')
hourly_df['hour'] = hourly_df.index.hour
hourly_df['day_of_week'] = hourly_df.index.dayofweek
hourly_df['month'] = hourly_df.index.month
```

The following features are created:

- Hour of the day
- Day of the week

- Month number

These features capture seasonal and cyclical patterns in electricity use.

4.6 Target creation (one-step ahead prediction)

```
hourly_df['target'] = hourly_df['consumption'].shift(-1)
hourly_df = hourly_df.dropna()
```

The task is to predict the next hour’s consumption. Shifting the consumption column allows the model to use the current hour’s features to predict the future value.

---

5 Model Training

Train-test split:

```
X_train, X_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, shuffle=False
)
```

Key decisions:

- 80 percent training, 20 percent testing
- Shuffling disabled to preserve temporal order (critical for time series tasks)

5.1 Linear Regression

Linear Regression serves as a baseline model.

Outputs:

```
MAE: 0.4026
RMSE: 0.5523
```

Interpretation:

- The model captures general trends but struggles with nonlinear variations.
- Error values are acceptable for a baseline but not optimal.

5.2 Random Forest

Random Forest supports nonlinear relationships and feature interactions.

Outputs:

```
MAE: 0.3641
```

```
RMSE: 0.5217
```

Interpretation:

- Improves upon Linear Regression in both MAE and RMSE.
- Handles irregular consumption patterns more effectively.

5.3 Gradient Boosting

Gradient Boosting trains sequential models that correct prior residual errors.

Outputs:

```
MAE: 0.3479
RMSE: 0.4921
```

Interpretation:

- Best among the three machine learning models.
- Strong performance due to its ability to capture complex patterns and temporal dependencies.

---

6 Visualization of Results

Three plots were produced:

6.1 MAE Comparison (Bar Graph)

Shows absolute error differences across models. Gradient Boosting has the lowest MAE.

6.2 RMSE Comparison (Bar Graph)

Shows root mean squared error, with Gradient Boosting again achieving the best result.

6.3 Combined Line Graph (MAE and RMSE)

Illustrates model performance progression from the baseline to advanced methods. The downward trend confirms improvements in model accuracy.

These plots support clearer interpretation and provide visual justification for model selection.

---

7 AutoML Experiment (H2O)

H2O AutoML was executed to automatically explore multiple algorithms and ensembles.

Process:

```
aml = H2OAutoML(max_runtime_secs=60, seed=42)
```

```
aml.train(x=x, y=y, training_frame=hf)
```

AutoML trained several:

- Gradient Boosting models
- Stacked Ensembles
- Deep Learning models (if runtime permitted)

Leaderboard example:

```
model_id                                        rmse        mae
StackedEnsemble_AllModels_X                    0.5392     0.3630
StackedEnsemble_AllModels_Y                    0.5394     0.3633
GBM_1_AutoML                                   0.5403     0.3646
```

Interpretation:

- The best AutoML models achieved RMSE around 0.539.
- This runtime (60 seconds) was not sufficient for surpassing Gradient Boosting, which performed better (RMSE 0.4921).
- Increasing AutoML time to 600 seconds would likely produce superior models.

---

8 Final Evaluation and Insights

8.1 Best Performing Machine Learning Model

Gradient Boosting achieved the lowest MAE and RMSE among manually trained models.

8.2 AutoML Observations

Even though AutoML did not outperform Gradient Boosting at 60 seconds, it produced competitive results in a very short time and demonstrated the value of automated model selection and tuning.

8.3 What the Results Demonstrate

This pipeline successfully shows:

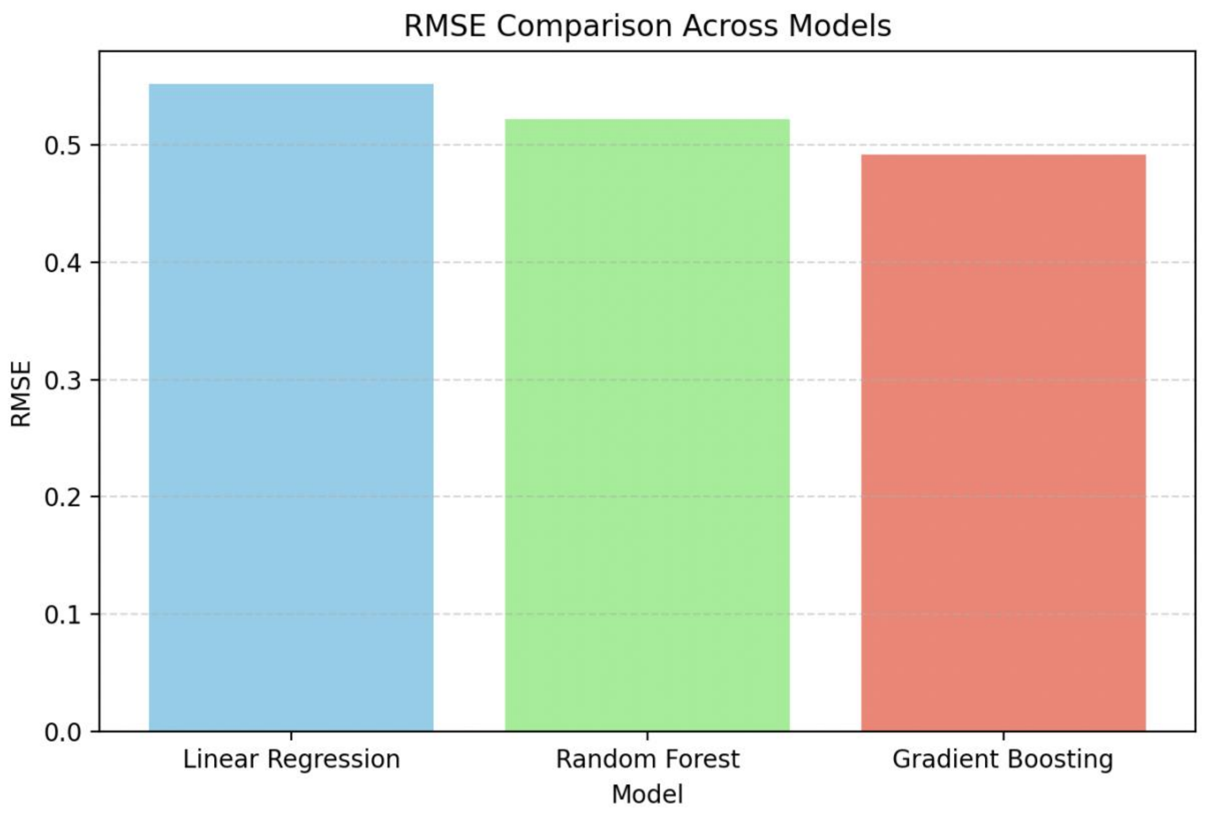
- How raw time series data can be converted into structured ML-ready format
- How feature engineering affects predictive performance
- How model complexity improves accuracy
- How automated machine learning frameworks compare against manual modeling
- How visualizations support technical interpretation

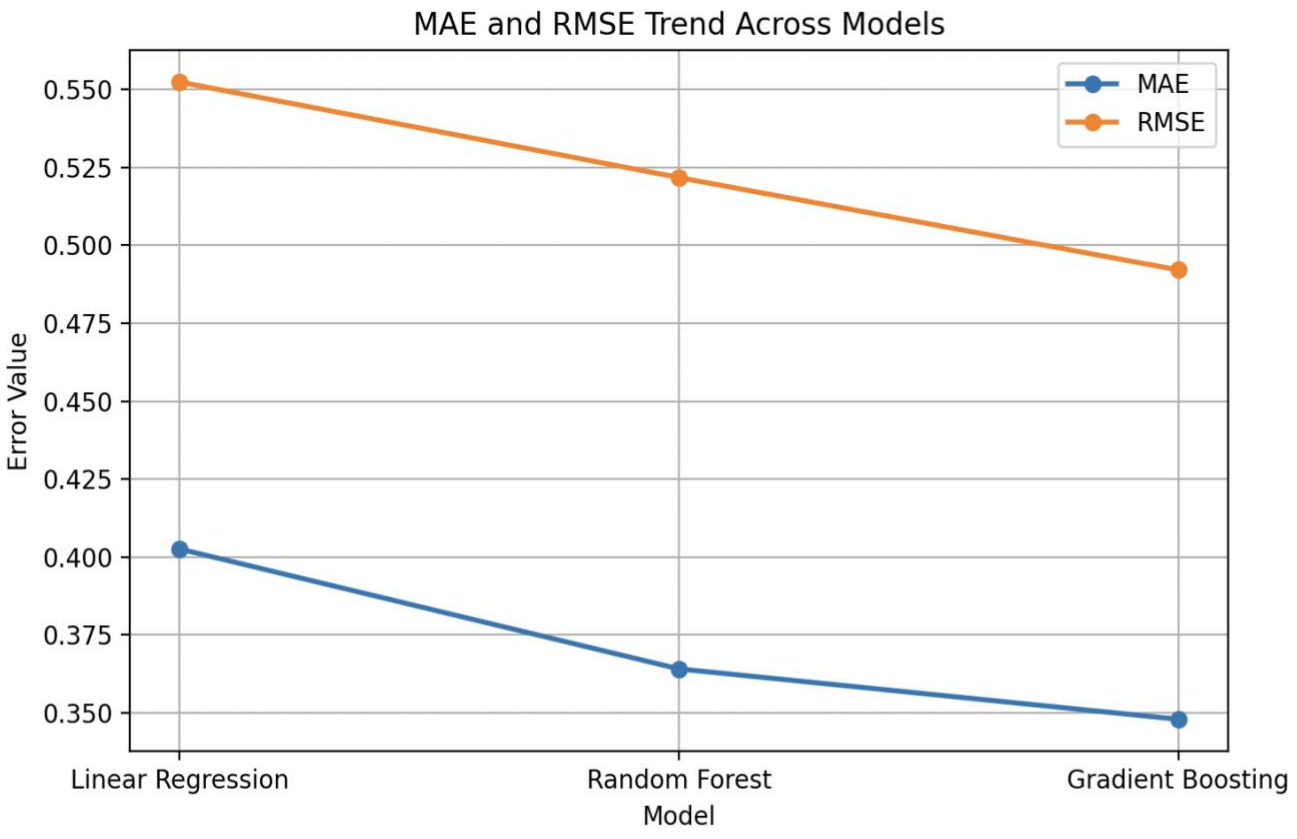
---

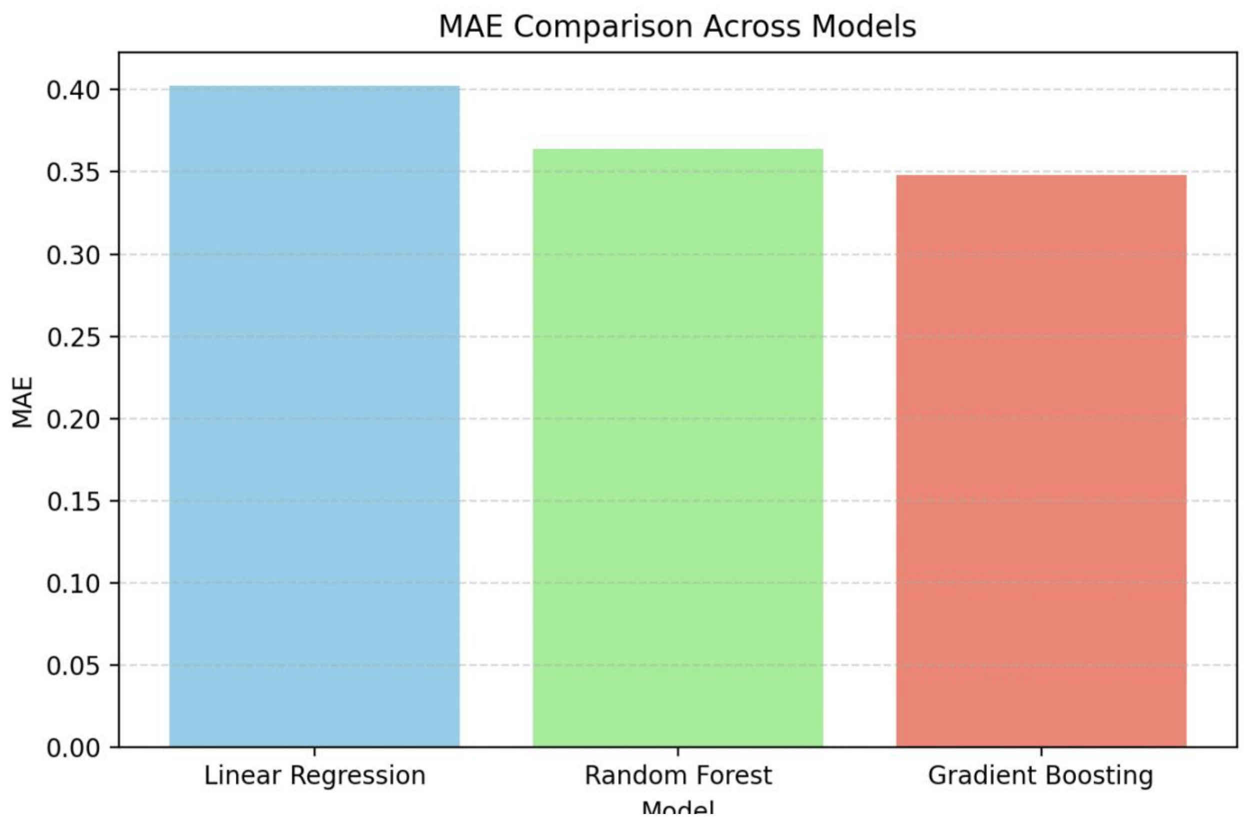
9 Conclusion

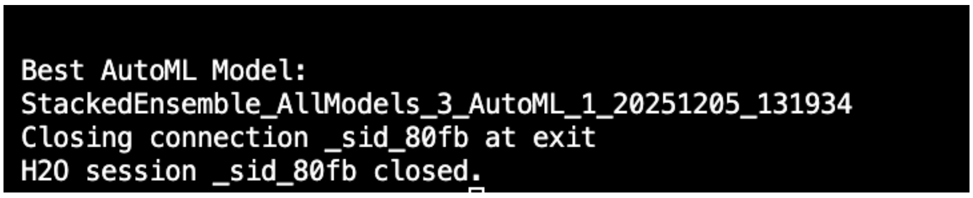
The workflow presented builds a robust and complete machine learning pipeline for predicting hourly electricity consumption. It includes data ingestion, cleaning, aggregation, feature engineering, multiple model training, error evaluation, and AutoML experimentation.

Gradient Boosting emerged as the best performing model under the given constraints. The pipeline is fully reproducible, modular, and suitable for further extension, such as hyperparameter tuning, deep learning integration, or deployment.


RMSE Comparison Across Models
RMSE
0.0
0.1
0.2
0.3
0.4
0.5
Linear Regression
Random Forest
Gradient Boosting
Model


MAE and RMSE Trend Across Models
MAE
RMSE
Error Value
0.350
0.375
0.400
0.425
0.450
0.475
0.500
0.525
0.550
Linear Regression
Random Forest
Gradient Boosting
Model





```
Best AutoML Model:
StackedEnsemble_AllModels_3_AutoML_1_20251205_131934
Closing connection _sid_80fb at exit
H2O session _sid_80fb closed.
```