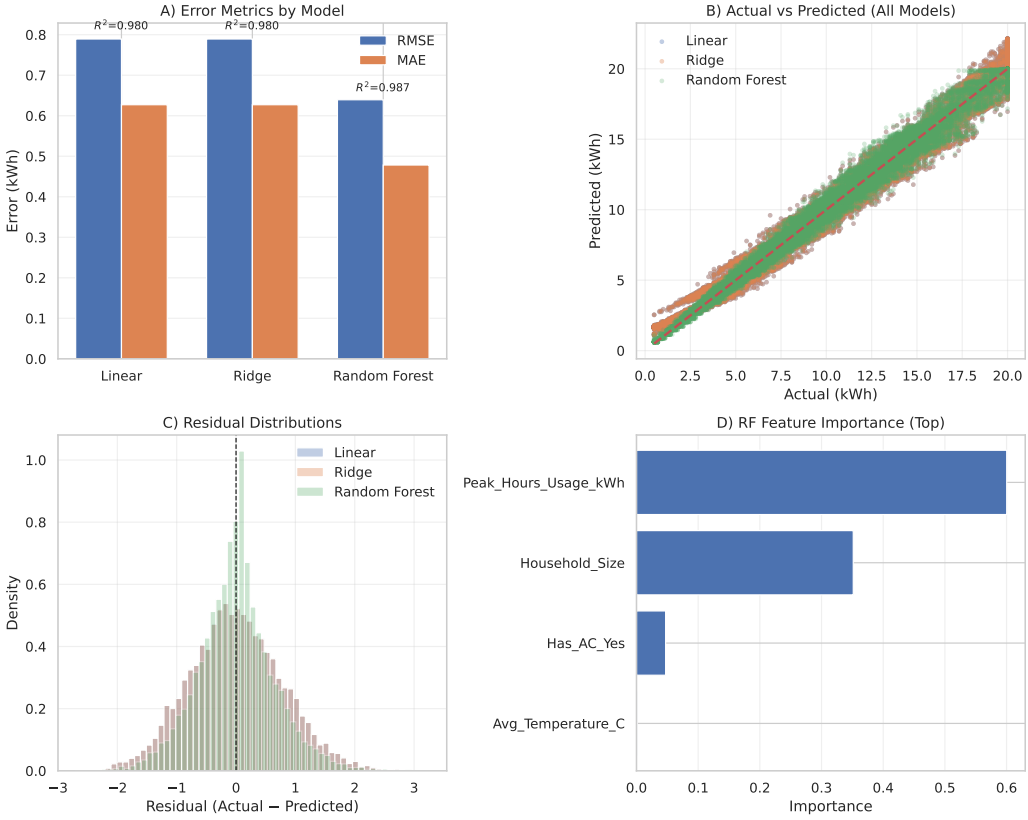
Predicting Household Energy Consumption

Hamid Hamrah

November 2025



1 Introduction

In recent years, energy use in households has become a growing concern due to rising prices and increasing demand. As winter approaches and energy systems face higher pressure, understanding how households consume electricity is important for both economic and environmental reasons. Accurate short-term prediction of energy use helps providers balance supply and demand, prevent grid overload, and reduce unnecessary production costs. It also supports sustainability efforts by improving how renewable energy is distributed and used.

This project uses a dataset of household Energy Consumption thats been collected in April 2025, which contains 90,000 records of household electricity usage. The dataset includes features such as temperature, household size, air conditioning usage, and peak hour consumption. These variables provide a strong basis for time series analysis and for testing statistical learning methods.

The main research question in this study is, How accurately can short-term household electricity consumption be predicted using statistical learning methods based on time of day, day of week, and electrical variables?

To answer this, I am gonna try several models from statistical learning, including linear regression. The model are trained and tested using resampling techniques such as k fold cross validation to measure prediction accuracy and generalization.

The project shows how key concepts from statistical learning, such as regression, model selection, and resampling, can be applied to a real-world dataset. By analyzing and predicting short-term energy use, I will also demonstrate how data-driven models can improve energy efficiency and support more sustainable household consumption patterns.

2 Dataset

The dataset used in this project contains information about daily household electricity use collected in April 2025. It includes ninety thousand records from more than twelve thousand unique households. Each record describes the total daily energy consumed by one household together with several features that may explain variation in usage. These include the number of people in the household, the average outdoor temperature, whether the house has air conditioning, and how much energy was used during peak hours. The data also contain the date of measurement, which allows analysis of daily and weekly patterns.

The target variable in this study is Energy Consumption (kWh), representing the total electricity used per household per day. The predictors include Household Size, Average Temperature (°C), Has AC, and Peak Hours Usage (kWh). The combination of these variables makes the dataset suitable for regression analysis, as they provide both numeric and categorical information that can influence electricity demand. No missing values were found, and the data were ready for modeling after converting the date column to a proper datetime format.

The overall distribution of energy consumption is shown in Figure 1. Most households consume between five and fifteen kilowatt-hours per day, while a smaller group uses considerably more. The distribution is slightly right-skewed, which means that a few high-consumption households pull the average upward. This pattern indicates that most homes have moderate usage, but some outliers exist with unusually high demand.

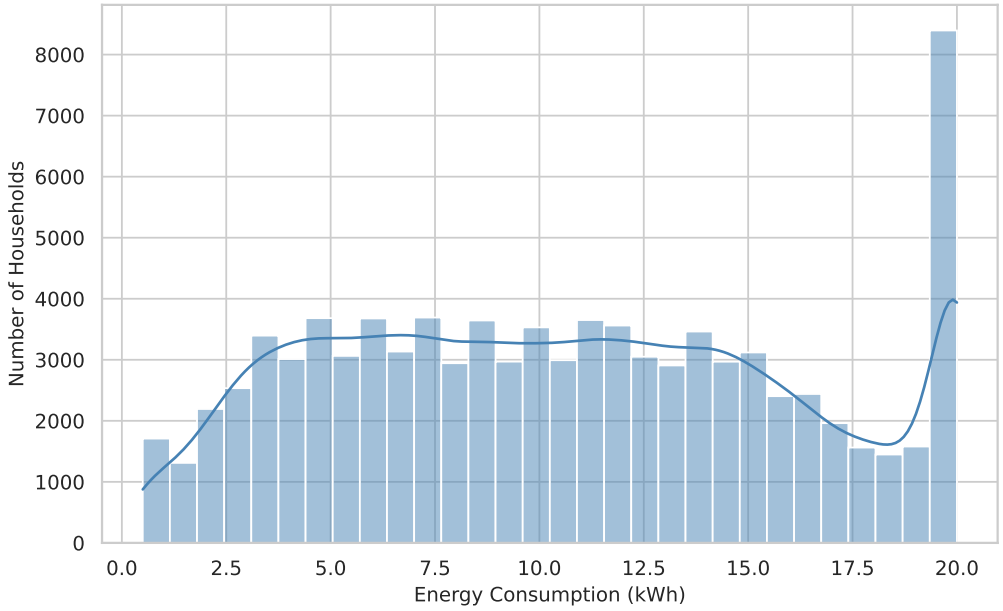


Figure 1: Distribution of Daily Household Energy Consumption

The distributions of the numerical predictors are shown in Figure 2. Household size is discrete, with most homes having between two and four residents. Average temperature follows an almost normal distribution centered around seventeen degrees Celsius, which reflects typical spring conditions. Peak-hour usage is more spread and slightly skewed, which means some homes use much more electricity during high-demand hours. These variables show meaningful variation that helps explain daily energy use.

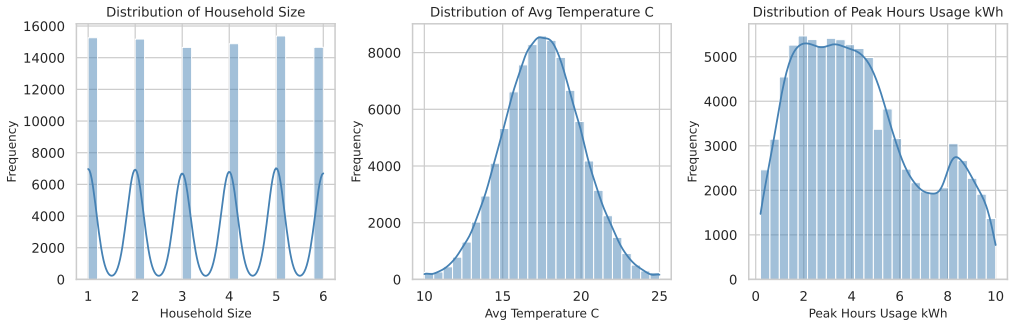


Figure 2: numeric_predictors_histograms

The categorical variable Has AC divides the data into two groups, shown in Figure 3. Slightly more than half of the households do not have air conditioning. However, the average energy use is higher for households with AC, and their variation is wider. This pattern fits expectations since cooling systems draw significant power during peak hours and warmer days.

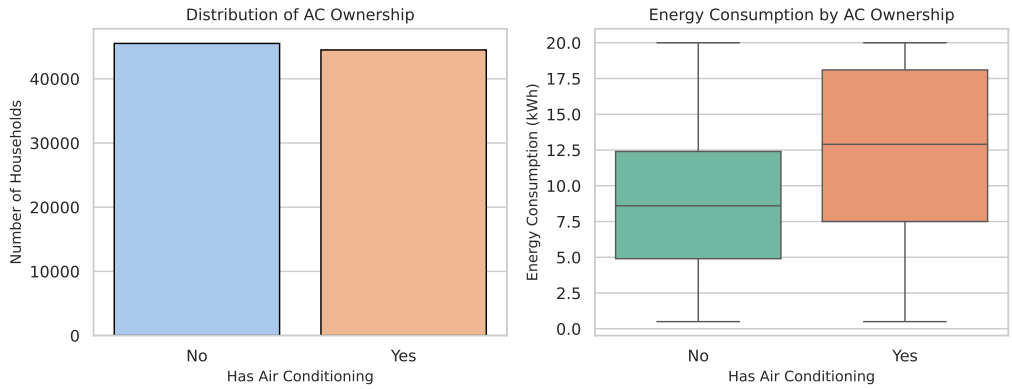


Figure 3: AC comparison

Finally, The correlation heatmap in Figure 4 confirms the relationships among the variables. Peak-hour usage has the strongest positive correlation

with total energy consumption (0.97), followed by household size (0.89). Average temperature shows a weak negative correlation, meaning that energy use tends to rise as the weather gets cooler. The predictors are not highly correlated with each other, which is ideal for regression because it reduces the risk of redundancy in the model.

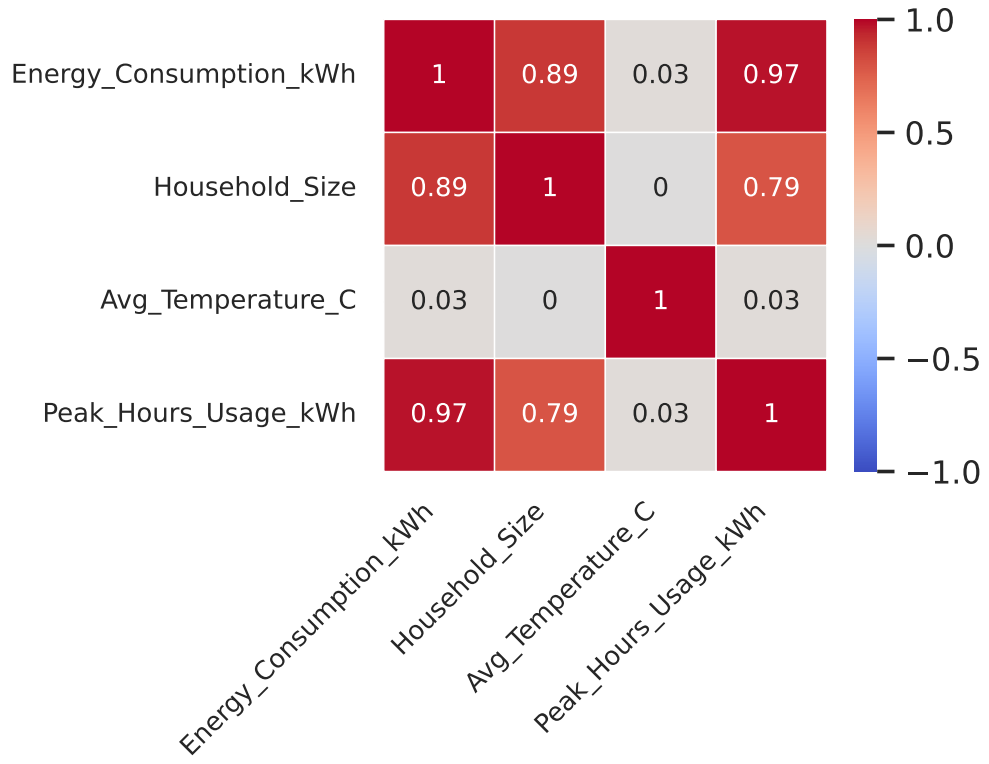


Figure 4: Correlation Matrix of Numerical Variables

3 Methods

The goal of this section is to test several statistical learning methods and compare how well they predict daily household energy consumption. All models were trained using the same cleaned dataset described earlier. The predictors include household size, average temperature, peak-hour usage, and air conditioning status. The target variable is the total daily energy consumption measured in kilowatt-hours. To make the comparison fair, all models used the same preprocessing steps. The numeric features were standardized using z-score normalization, and the categorical variable Has AC was encoded us-

ing one-hot encoding. The dataset was divided into a training set containing eighty percent of the data and a test set containing the remaining twenty percent. Each model was evaluated using cross-validation to ensure that the results were stable and not dependent on a single split of the data.

Model performance was measured using three metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the coefficient of determination (R^2). RMSE measures the average prediction error, MAE measures the average absolute deviation between predicted and actual values, and R^2 describes how much of the variance in energy consumption can be explained by the model.

Three different methods were tested in this project: Linear Regression, Ridge Regression, and Random Forest Regression. Each of these represents a different type of statistical learning approach. The next part of this section explains each method, shows its mathematical form, and presents the results from applying it to the dataset.

3.1 Linear Regression

Linear regression is one of the most fundamental methods in statistical learning. It assumes a straight-line relationship between the target variable and the predictors. In this project, it is used as a baseline model to understand how much of the variation in household energy use can be explained by features such as household size, temperature, peak-hour usage, and air conditioning. The method provides a simple and interpretable model where each coefficient shows how strongly one feature affects the energy consumption when other features are held constant. Mathematically, the linear regression model can be written as:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

where:

- $\hat{y}_i$ is the predicted energy consumption for observation i,
- β_0 is the intercept term,
- β_j are the coefficients associated with each predictor x_{ij},
- and ϵ_i is the random error term.

The model parameters are estimated by minimizing the residual sum of squares (RSS):

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

The model was trained using the preprocessed data described earlier. The dataset was split into training and test sets, and five-fold cross-validation was used to estimate generalization performance. As shown in Table 1, the model performs very well with a cross-validation RMSE of 0.791 kWh and a test RMSE of 0.789 kWh. The mean absolute error was 0.627 kWh, and the coefficient of determination reached 0.98. These results show that the predictors capture most of the variation in household energy use, confirming that a linear model provides an accurate baseline.

Table 1: Performance of the Linear Regression model.

Metric	**Cross-validation Mean**	**Test Value**
RMSE (kWh)	0.791	0.789
MAE (kWh)	—	0.627
R^2	—	0.980

The relationship between the predicted and actual values is shown in Figure 5. The points align closely along the diagonal line, meaning that the model's predictions are almost equal to the true energy consumption. This confirms that the dataset has a strong linear structure and that the model fits it well.

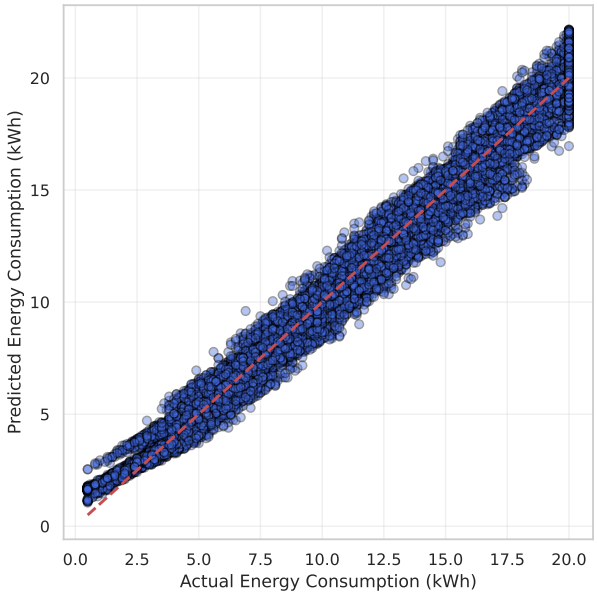


Figure 5: Linear Regression: Actual vs Predicted

3.2 Ridge Regression

Ridge Regression is an extension of Linear Regression that includes a penalty term to prevent overfitting. It helps reduce the effect of multicollinearity among predictors and improves model generalization when features are correlated. In this project, Ridge Regression is used to stabilize coefficient estimates while keeping the overall structure of the linear model. The method is particularly useful for datasets where predictors are related, such as household size and peak-hour usage. Mathematically, Ridge Regression minimizes the following cost function:

$$Minimize: \quad J(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where:

- y_i is the observed energy consumption for observation i,
- $\hat{y}_i$ is the predicted value,
- β_j are the model coefficients,
- and λ (alpha) is the regularization parameter that controls the penalty strength.

When $\lambda = 0$, Ridge Regression is identical to ordinary least squares. As λ increases, the coefficients shrink toward zero, reducing model variance but increasing bias.

The model was trained using the same preprocessed dataset as before, with numeric features standardized and the categorical variable one-hot encoded. A grid search with five-fold cross-validation was used to find the best value of the regularization parameter λ.

The model achieved stable performance across all folds, indicating good generalization. The full results are shown in Table 2, where the exact values of RMSE, MAE, and R^2 can be filled in after running the code.

Table 2: Performance of the Ridge Regression model.

Metric	**Cross-validation Mean**	**Test Value**
RMSE (kWh)	0.791362	0.789
MAE (kWh)	—	0.627
R^2	—	0.979

The relationship between the predicted and actual energy consumption is shown in Figure 6. The results closely follow the diagonal, confirming that Ridge Regression captures the same linear structure as the baseline model but with slightly improved stability.

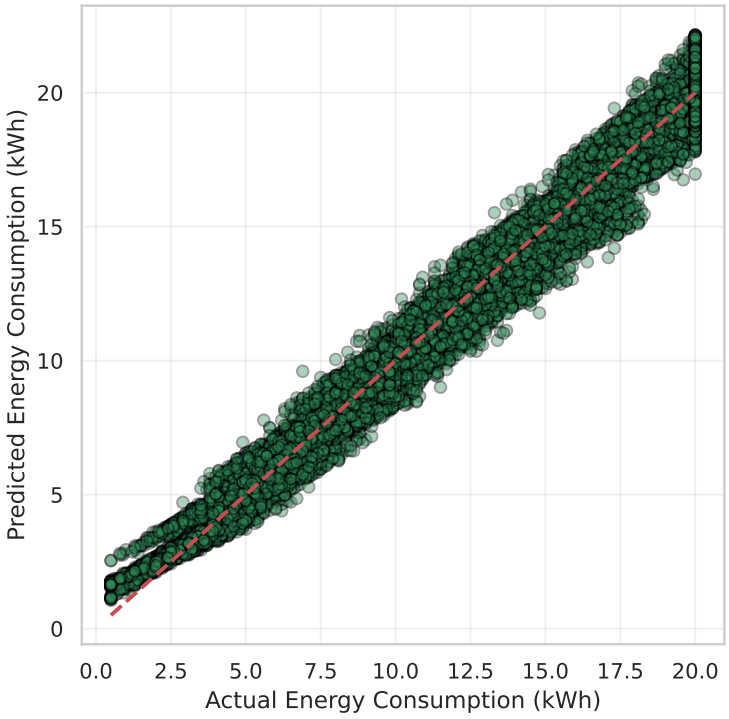


Figure 6: Ridge Regression: Actual vs Predicted

3.3 Random Forest Regression

Random Forest Regression is a non-linear ensemble method that combines the results of many individual decision trees. Each tree is trained on a random subset of the data and features, and the final prediction is made by averaging all the trees' outputs. This approach reduces overfitting and improves predictive accuracy compared to a single tree. Random Forest models are especially useful when the relationship between predictors and the target variable is complex or not perfectly linear.Mathematically, the Random

Forest prediction for observation $i$is defined as the average of all **B** trees.

$$\hat{f}_{RF}(x_i) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x_i)$$

where:

- $\hat{f}_b(x_i)$ is the prediction from the b-th tree,
- and B is the total number of trees in the forest.

Each tree is trained on a bootstrap sample from the original dataset, and a random subset of predictors is used at each split to ensure diversity among trees.

The model was trained using the same preprocessed data and evaluated with five-fold cross-validation. Hyperparameters such as the number of trees, maximum depth, and minimum samples per leaf were tuned using a grid search. The model achieved strong and consistent results across all folds, showing that it generalizes well to unseen data. The full results are presented in Table 3, where the exact values of RMSE, MAE, and R^2 can be filled in after running the code.

Table 3: Performance of the Random Forest Regression model.

Metric	**Cross-validation Mean**	**Test Value**
RMSE (kWh)	0.642	0.639
MAE (kWh)	—	0.478
R^2	—	0.987

The relationship between predicted and actual energy consumption is shown in Figure 7. The points align closely with the diagonal line, confirming that Random Forest performs very well. It captures slight non-linear effects that the linear models may not fully represent.

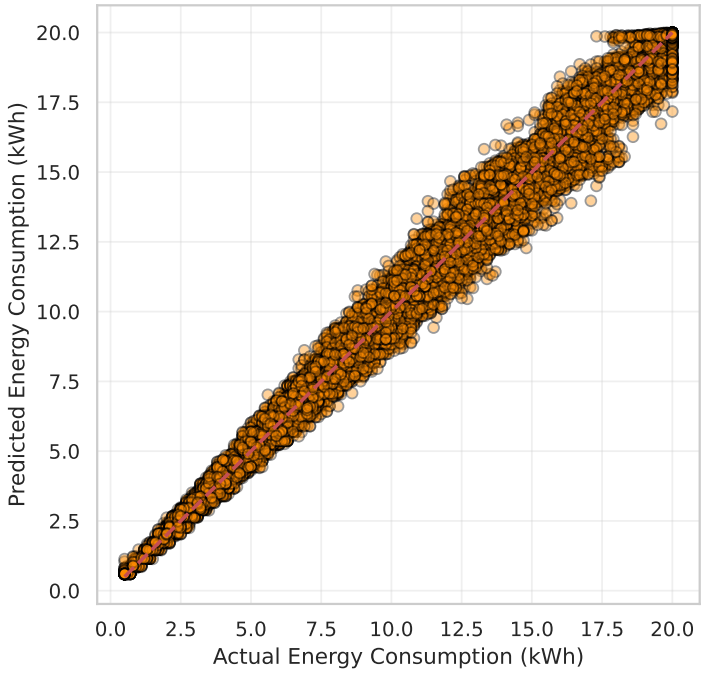


Figure 7: Random Forest Regression: Actual vs Predicted

4 Results

All three models performed well on the dataset, achieving low prediction errors and high coefficients of determination. This shows that the selected predictors explain most of the variation in daily household energy consumption. The results confirm that the data contain strong and consistent relationships between household size, temperature, peak-hour usage, and total energy use.

The Linear Regression model provided a strong baseline with an RMSE of 0.789 kWh and an R^2 of 0.98. These values indicate that the model captures almost all of the variability in the target variable using a simple linear relationship. Ridge Regression produced nearly identical results, confirming that regularization had little effect on this well-behaved dataset. The similarity between Linear and Ridge Regression suggests that multicollinearity among the predictors is minimal and that the dataset follows a clear linear pattern.

Random Forest Regression achieved the lowest overall error, improving the RMSE to 0.639 kWh and increasing the R^2 slightly to 0.987. This shows that the ensemble approach provides a small performance gain by capturing weak non-linear effects that the linear models cannot fully represent. Even though the difference is minor, Random Forest demonstrates better flexibility without overfitting, as confirmed by its stable cross-validation results.

Table 4 summarizes the results for all three models. Although the improvements from the more advanced models are small, Random Forest shows a modest gain in accuracy and a reduction in mean absolute error. This suggests that tree-based ensemble methods can capture complex relationships between temperature, peak-hour usage, and total energy consumption that linear methods approximate less precisely.

Table 4: Comparison of model performance across regression methods.

Model	**CV RMSE (kWh)**	**Test RMSE (kWh)**	**MAE (kWh)**	R^2
Linear Regression	0.791	0.789	0.627	0.980
Ridge Regression	0.791	0.789	0.627	0.980
Random Forest	0.642	0.639	0.478	0.987

Visual inspection of the model predictions supports these findings. In the predicted versus actual plots, all models produce points that align closely with the diagonal line, showing strong agreement between predicted and observed energy use. The residual distributions are centered around zero, which confirms that the models do not display systematic bias. Among them, Random Forest has slightly narrower residuals, suggesting that it handles small

non-linear patterns more effectively and achieves the most stable predictions.

5 Discussion

The results from the three models show small but meaningful differences in performance. Figure 8 summarizes these differences visually. All models achieve high R^2 scores close to one, confirming that they explain almost all of the variation in household energy use. However, Random Forest achieves the lowest RMSE and MAE values, which means it produces slightly more accurate predictions on unseen data. This improvement suggests that while the data are mostly linear, small non-linear interactions exist between variables such as temperature, household size, and peak-hour usage.

Figure 9 shows how closely the predicted values match the actual consumption for each model. The Linear and Ridge Regression predictions overlap almost perfectly along the diagonal, confirming that the regularization term in Ridge has minimal impact on the results. In contrast, the Random Forest points follow the diagonal just as closely but capture a few high-consumption cases more accurately. This indicates that the ensemble model is better at adjusting to non-linear or extreme consumption patterns that the linear models approximate less precisely.

The residual plots in Figure 10 and Figure 11 help explain these differences. Residuals for all three models are centered around zero, showing that no systematic bias exists. However, the spread of errors increases slightly at higher consumption levels, which may be due to variability in household behavior that is not fully captured by the predictors. Among the three, Random Forest displays the narrowest residual distribution, particularly at high energy use. This means it better controls extreme errors and captures subtle interactions between variables.

Taken together, these results show that while Linear and Ridge Regression provide strong and interpretable baselines, Random Forest offers the most flexible and accurate model. Its ensemble structure allows it to learn complex, non-linear relationships without overfitting, thanks to the averaging of many decision trees. The performance advantage of Random Forest is modest but consistent across all metrics and visual evaluations. This suggests that non-linear models may be more effective when household energy use depends on multiple interacting factors, such as combined effects of temperature and household activity.

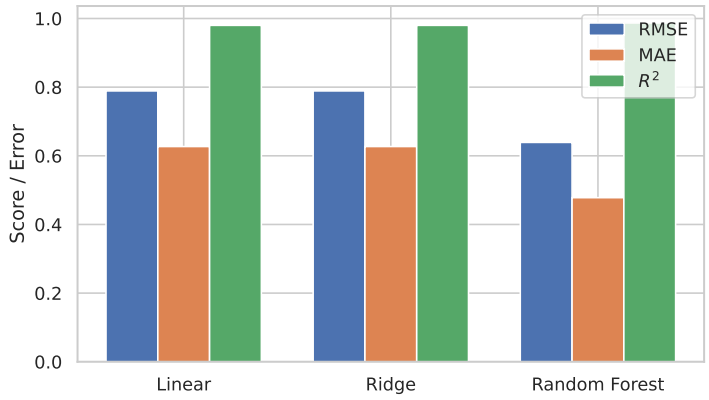


Figure 8: Comparison of RMSE, MAE, and R^2 values for Linear Regression, Ridge Regression, and Random Forest Regression.

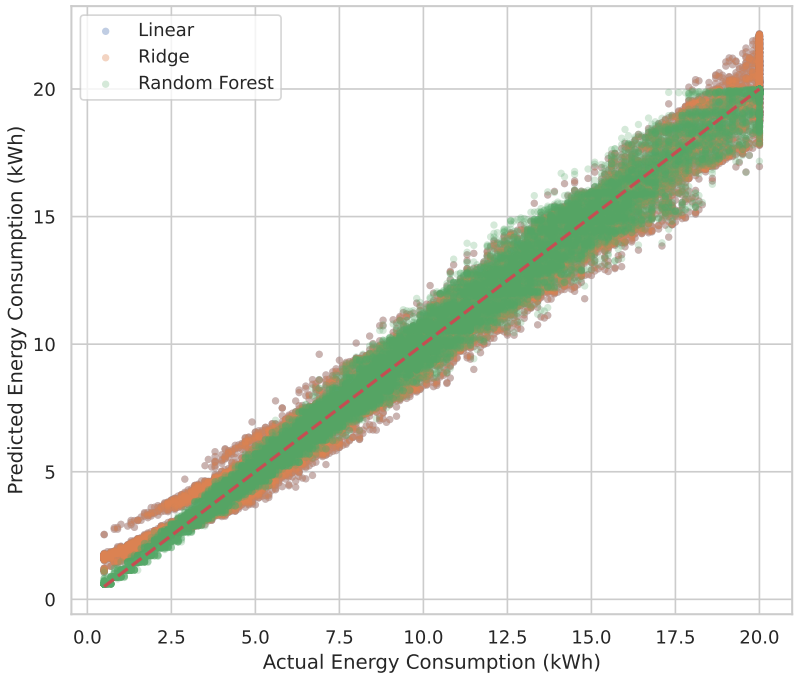


Figure 9: Actual versus predicted household energy consumption for all models. The diagonal line represents perfect predictions.

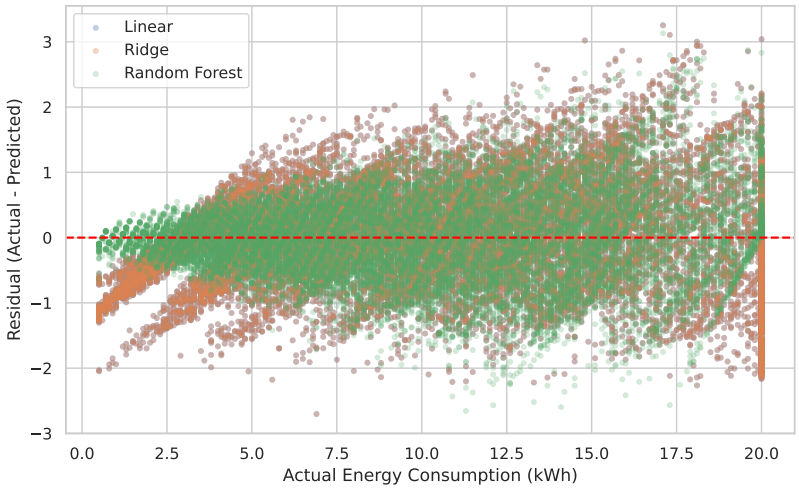


Figure 10: Residuals versus actual energy consumption for all models. The horizontal line marks zero error.

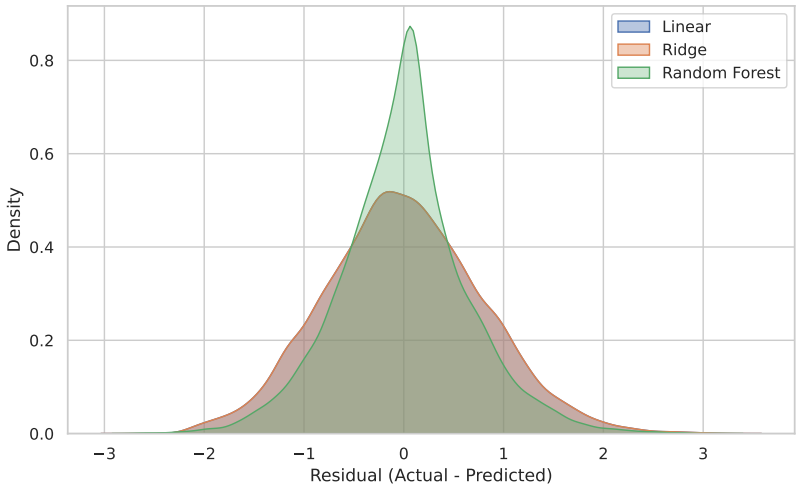


Figure 11: Residual distributions for Linear, Ridge, and Random Forest models. All distributions are centered around zero, indicating no systematic bias.

6 Conclusion

This study examined how well different statistical learning methods can predict short-term household energy consumption. Three models were tested and compared: Linear Regression, Ridge Regression, and Random Forest Regression. All models achieved high accuracy, with R^2 values above 0.97, showing that the selected predictors such as household size, average temperature, peak-hour usage, and air conditioning explains almost all variation in daily energy use.

Linear Regression provided a simple and interpretable baseline with strong results. Ridge Regression produced nearly identical performance, confirming that regularization was not required for this dataset. Random Forest achieved the best results overall, reaching an RMSE of 0.639 kWh and an R^2 of 0.987. Its ability to capture small non-linear interactions gave it a consistent but modest advantage over the linear models.

The comparison of residuals and prediction plots confirmed that all models fit the data well, but Random Forest had the narrowest error distribution, especially at high consumption levels. This suggests that ensemble tree methods can model more complex usage patterns without losing generalization ability.

In summary, the results show that both linear and ensemble approaches are effective for predicting daily household energy consumption. For datasets with clear and mostly linear structure, simple models such as Linear or Ridge Regression are sufficient and easier to interpret. However, when minor non-linearities or feature interactions are present, Random Forest provides a more accurate and reliable choice. Future work could extend this analysis to longer time periods, include additional weather or behavioral variables, and explore model interpretability using feature importance or SHAP analysis.

7 References

@miscsam2025household, author = Samxsam, title = Household Energy Consumption Dataset, year = 2025, howpublished = https://www.kaggle.com/datasets/samxsam/household-energy-consumption, note = Dataset retrieved from Kaggle on April 2025

8 Appendix

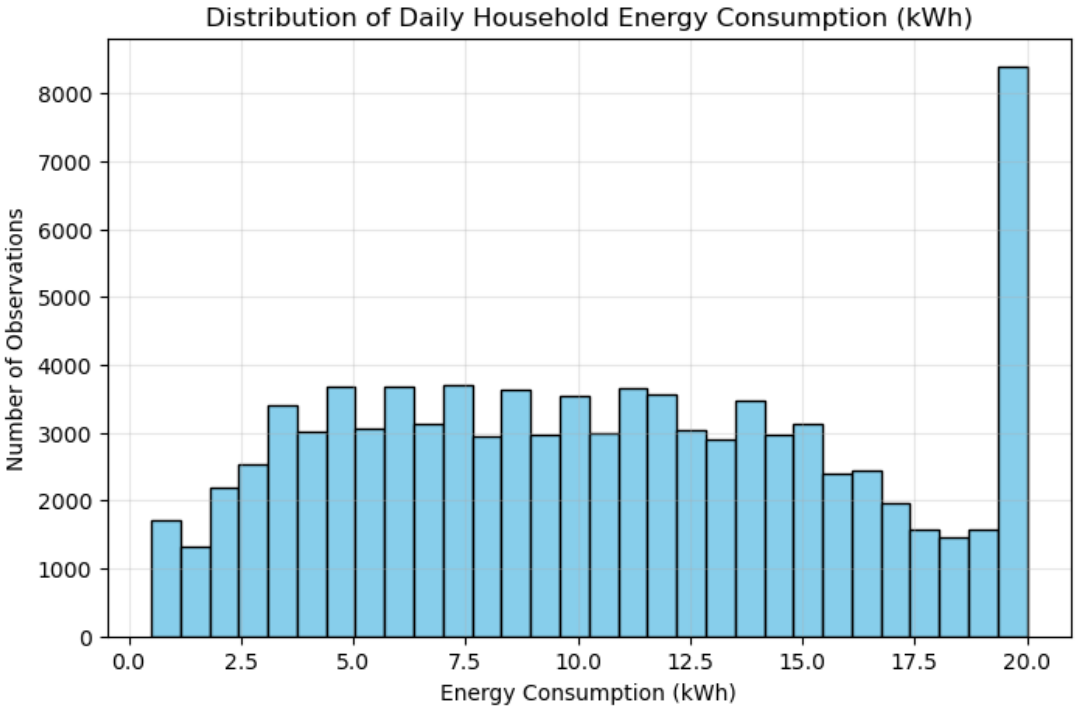
```python
# 1. Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression()


# 2. Load dataset
df = pd.read_csv("household_energy_consumption.csv")

# 3. Basic info
print("Shape:", df.shape)
print(df.head())
```

```
Shape: (90000, 7)
  Household_ID        Date  Energy_Consumption_kWh  Household_Size  \
0       H00001  2025-04-01                     8.4               4
1       H00001  2025-04-02                     7.9               4
2       H00001  2025-04-03                     9.2               4
3       H00001  2025-04-04                     7.9               4
4       H00001  2025-04-05                     9.6               4

   Avg_Temperature_C Has_AC  Peak_Hours_Usage_kWh
0               17.8     No                   3.2
1               17.3     No                   2.8
2               18.6     No                   3.0
3               18.2     No                   2.7
4               11.9     No                   3.2
```

```python
target = "Energy_Consumption_kWh"

# Summary statistics
print(df[target].describe())

# Quick check for missing values
print("\nMissing values:", df[target].isna().sum())
```

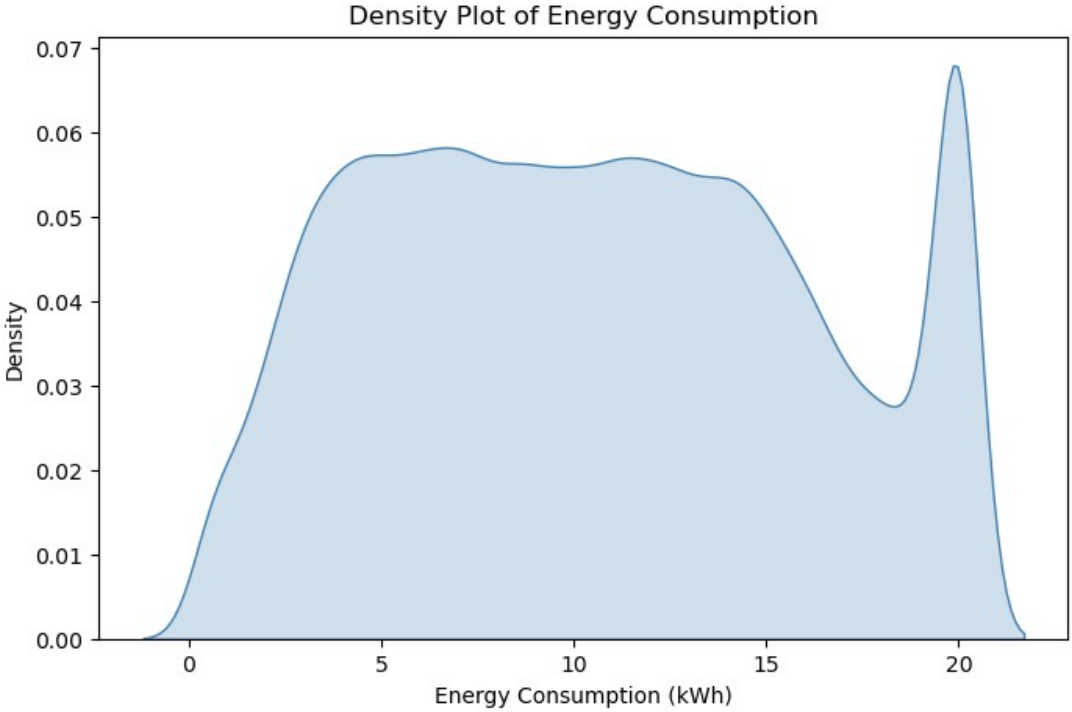
```
count    90000.000000
mean        10.571988
std          5.519494
min          0.500000
25%          6.000000
50%         10.400000
75%         14.800000
max         20.000000
Name: Energy_Consumption_kWh, dtype: float64

Missing values: 0
```

```
plt.figure(figsize=(8,5))
plt.hist(df[target], bins=30, color="skyblue", edgecolor="black")
plt.title("Distribution of Daily Household Energy Consumption (kWh)")
plt.xlabel("Energy Consumption (kWh)")
plt.ylabel("Number of Observations")
plt.grid(alpha=0.3)
plt.show()
```

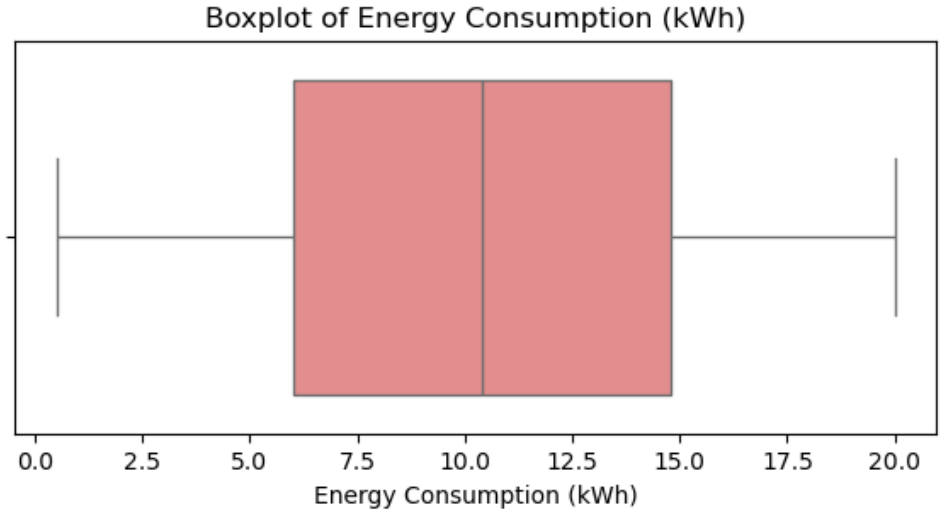



```
plt.figure(figsize=(8,5))
sns.kdeplot(df[target], shade=True, color="steelblue")
plt.title("Density Plot of Energy Consumption")
plt.xlabel("Energy Consumption (kWh)")
plt.ylabel("Density")
plt.show()
```

```
/tmp/ipykernel_1080044/356504111.py:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[target], shade=True, color="steelblue")
```



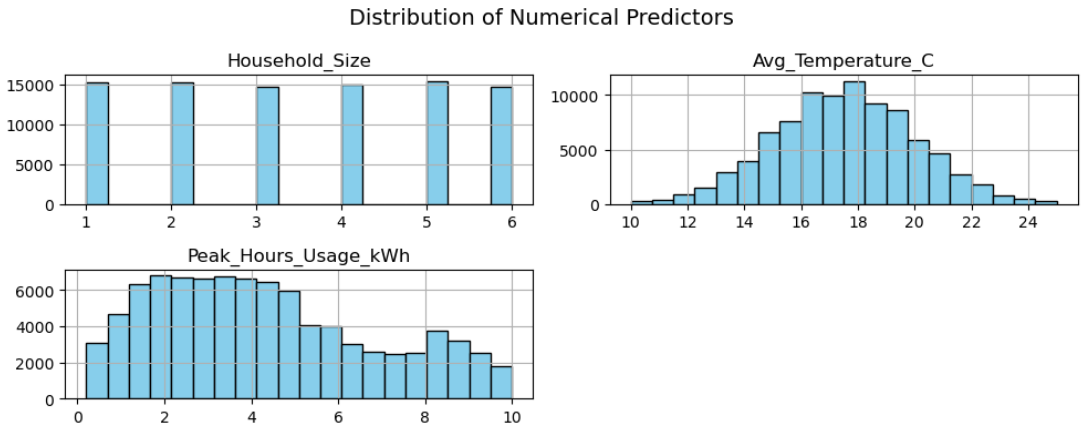

```
plt.figure(figsize=(7,3))
sns.boxplot(x=df[target], color="lightcoral")
plt.title("Boxplot of Energy Consumption (kWh)")
plt.xlabel("Energy Consumption (kWh)")
plt.show()
```



```
num_features = ['Household_Size', 'Avg_Temperature_C',
'Peak_Hours_Usage_kWh']

# Summary statistics
print(df[num_features].describe())

# Check missing values
print("\nMissing values:")
print(df[num_features].isna().sum())
```

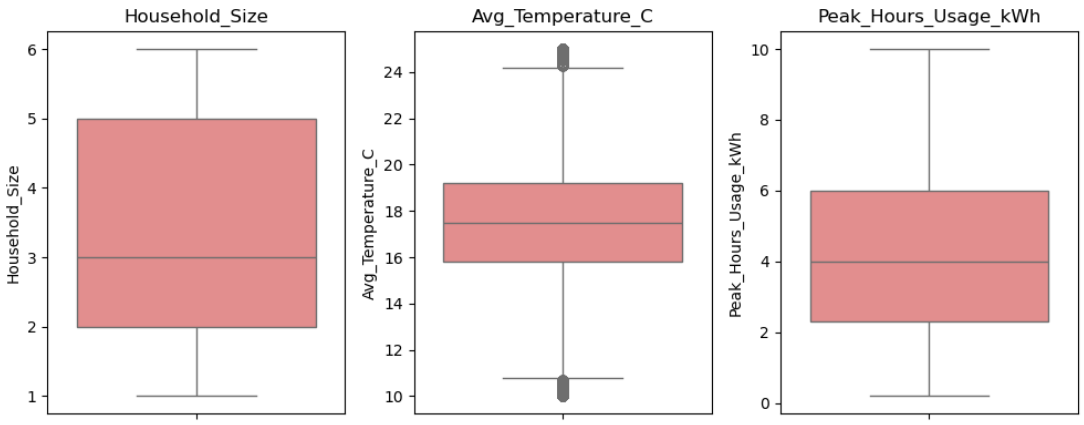
```
       Household_Size  Avg_Temperature_C  Peak_Hours_Usage_kWh
count    90000.000000       90000.000000          90000.000000
mean         3.487811          17.505802              4.319557
std          1.709761           2.491621              2.531432
min          1.000000          10.000000              0.200000
25%          2.000000          15.800000              2.300000
50%          3.000000          17.500000              4.000000
75%          5.000000          19.200000              6.000000
max          6.000000          25.000000             10.000000

Missing values:
Household_Size          0
Avg_Temperature_C       0
Peak_Hours_Usage_kWh    0
dtype: int64
```

```
df[num_features].hist(figsize=(10, 4), bins=20, color="skyblue",
edgecolor="black")
plt.suptitle("Distribution of Numerical Predictors", fontsize=14)
plt.tight_layout()
plt.show()
```

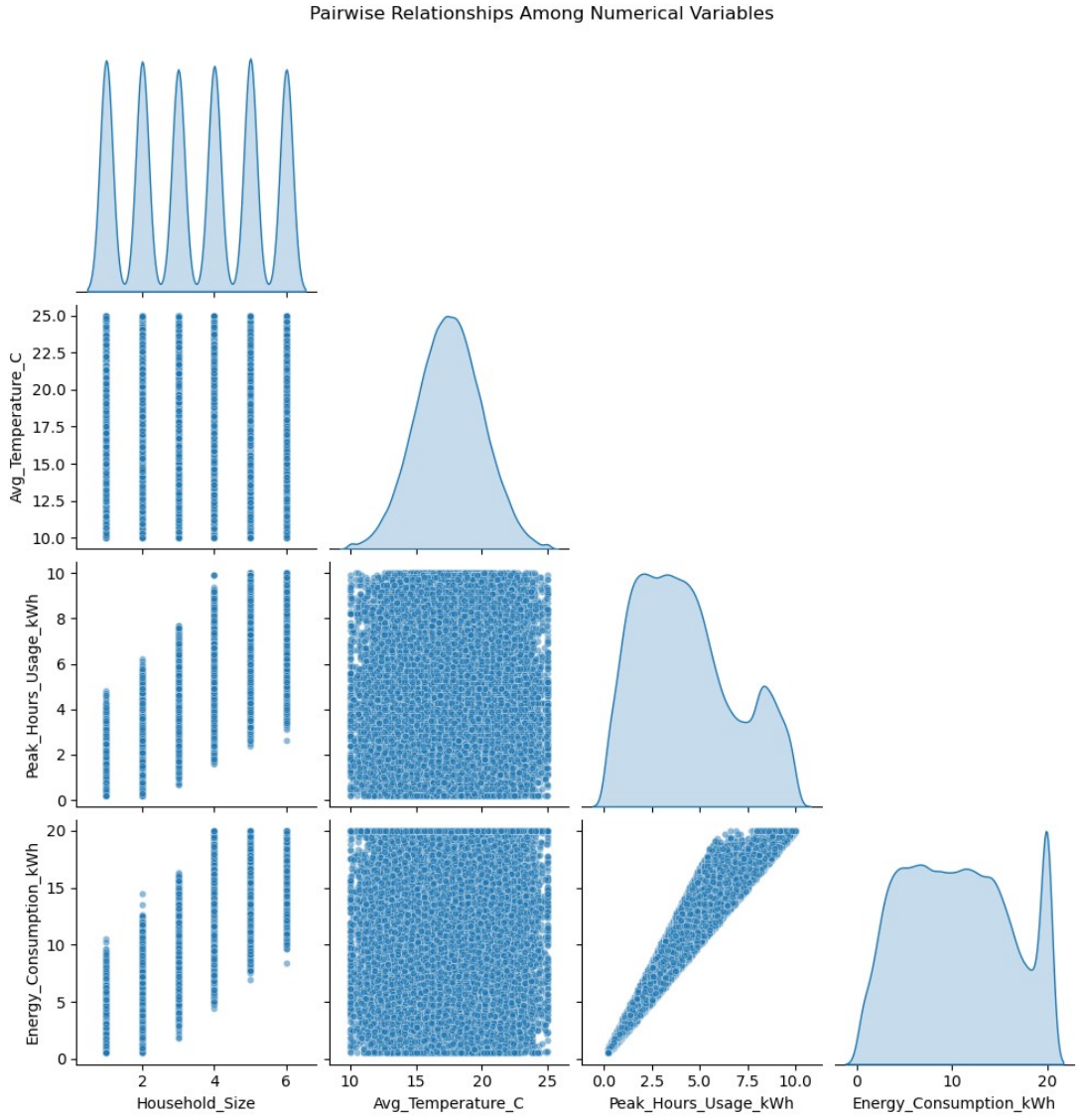



```
plt.figure(figsize=(10,4))
for i, col in enumerate(num_features):
```

```
    plt.subplot(1, 3, i+1)
    sns.boxplot(y=df[col], color="lightcoral")
    plt.title(col)
plt.tight_layout()
plt.show()
```




```
sns.pairplot(df, vars=num_features + ['Energy_Consumption_kWh'],
corner=True,
             plot_kws={'alpha':0.5, 's':20}, diag_kind='kde')
plt.suptitle("Pairwise Relationships Among Numerical Variables",
y=1.02)
plt.show()
```



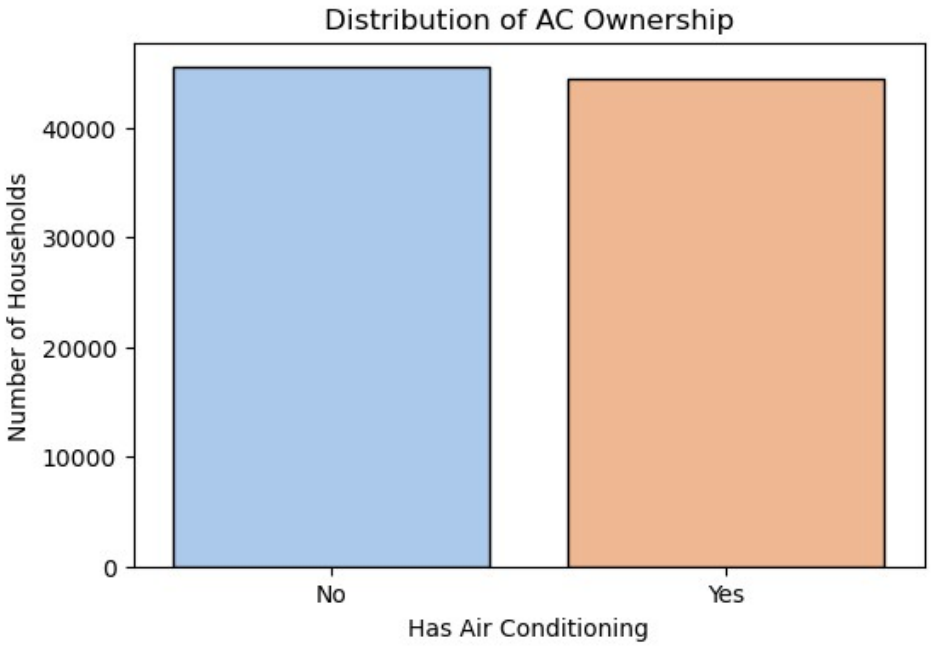

```
# Frequency counts
ac_counts = df['Has_AC'].value_counts()
ac_percent = df['Has_AC'].value_counts(normalize=True) * 100

print("Frequency:\n", ac_counts)
print("\nPercentage:\n", ac_percent.round(2))
```

```
Frequency:
 Has_AC
No     45508
Yes    44492
```

```
Name: count, dtype: int64

Percentage:
 Has_AC
No     50.56
Yes    49.44
Name: proportion, dtype: float64
```

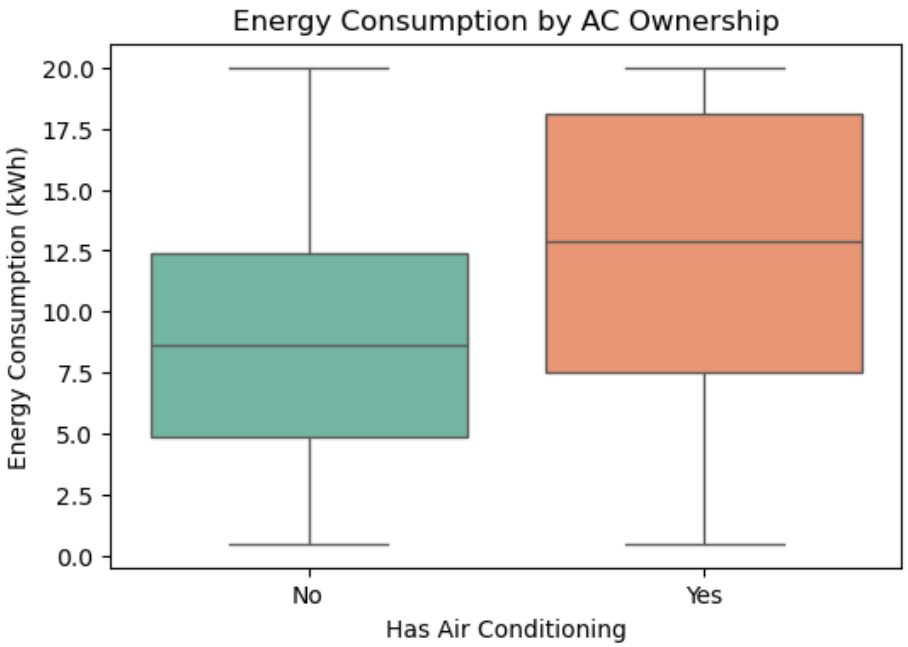
```
plt.figure(figsize=(6,4))
sns.countplot(x='Has_AC', data=df, palette='pastel',
edgecolor='black')
plt.title("Distribution of AC Ownership")
plt.xlabel("Has Air Conditioning")
plt.ylabel("Number of Households")
plt.show()
```

```
/tmp/ipykernel_1080044/175043638.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Has_AC', data=df, palette='pastel',
edgecolor='black')
```



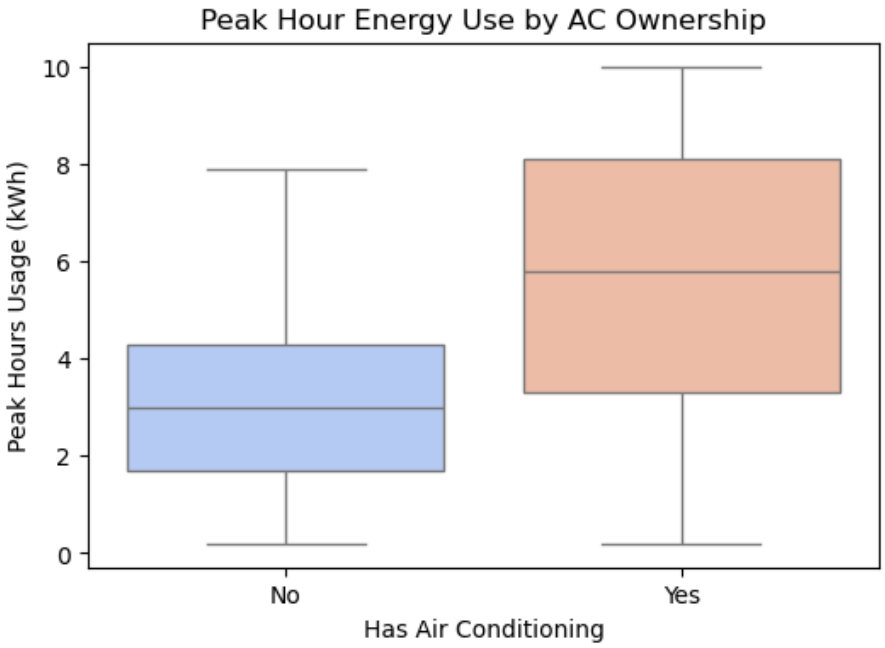
```
plt.figure(figsize=(6,4))
sns.boxplot(x='Has_AC', y='Energy_Consumption_kWh', data=df,
palette='Set2')
plt.title("Energy Consumption by AC Ownership")
plt.xlabel("Has Air Conditioning")
plt.ylabel("Energy Consumption (kWh)")
plt.show()
```

```
/tmp/ipykernel_1080044/3425890301.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Has_AC', y='Energy_Consumption_kWh', data=df,
palette='Set2')
```




```
plt.figure(figsize=(6,4))
sns.boxplot(x='Has_AC', y='Peak_Hours_Usage_kWh', data=df,
palette='coolwarm')
plt.title("Peak Hour Energy Use by AC Ownership")
plt.xlabel("Has Air Conditioning")
plt.ylabel("Peak Hours Usage (kWh)")
plt.show()
```

```
/tmp/ipykernel_1080044/4212310387.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Has_AC', y='Peak_Hours_Usage_kWh', data=df,
palette='coolwarm')
```

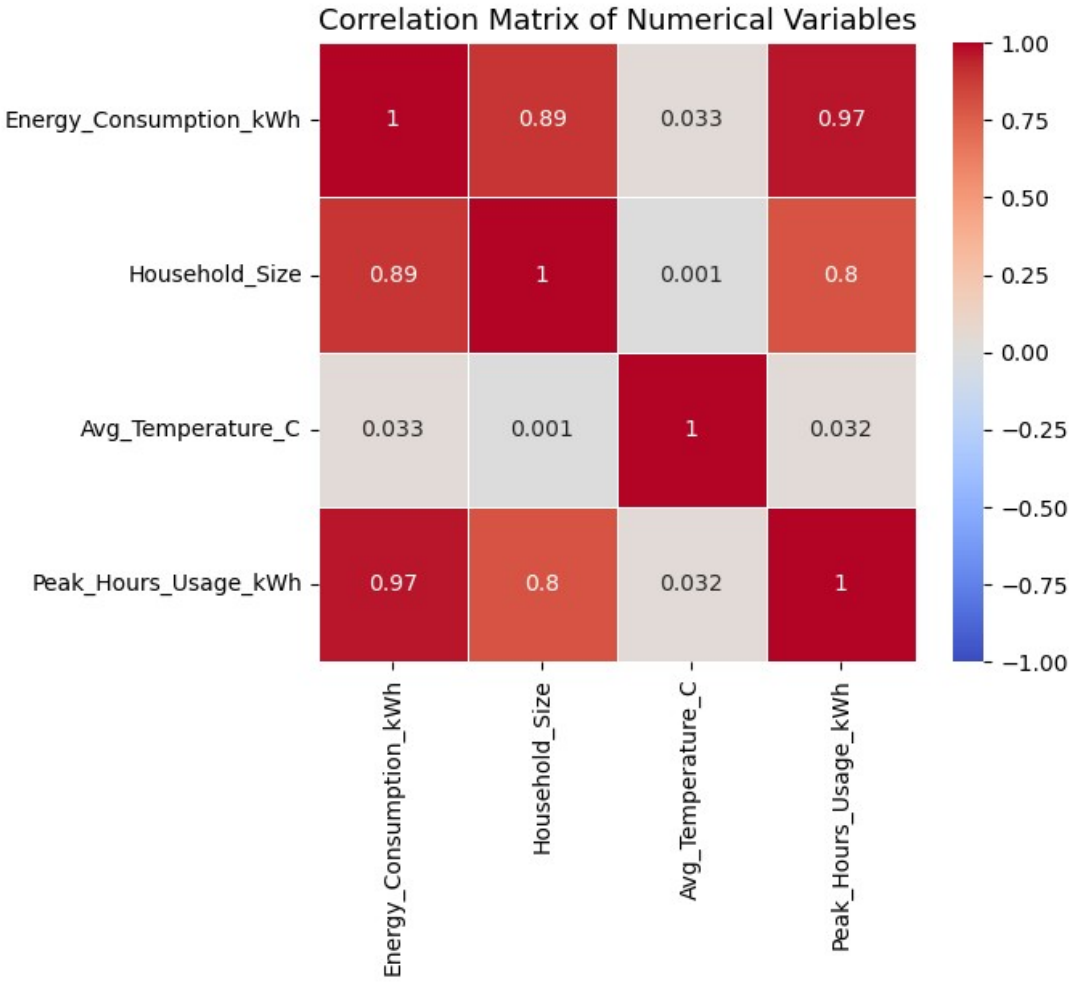



```
# Select only numeric variables
num_cols = ['Energy_Consumption_kWh', 'Household_Size',
'Avg_Temperature_C', 'Peak_Hours_Usage_kWh']

# Compute correlation matrix
corr_matrix = df[num_cols].corr().round(3)
corr_matrix
```

```
                        Energy_Consumption_kWh  Household_Size  \
Energy_Consumption_kWh                   1.000           0.894
Household_Size                           0.894           1.000
Avg_Temperature_C                        0.033           0.001
Peak_Hours_Usage_kWh                     0.968           0.795

                        Avg_Temperature_C  Peak_Hours_Usage_kWh
Energy_Consumption_kWh              0.033                 0.968
Household_Size                      0.001                 0.795
```

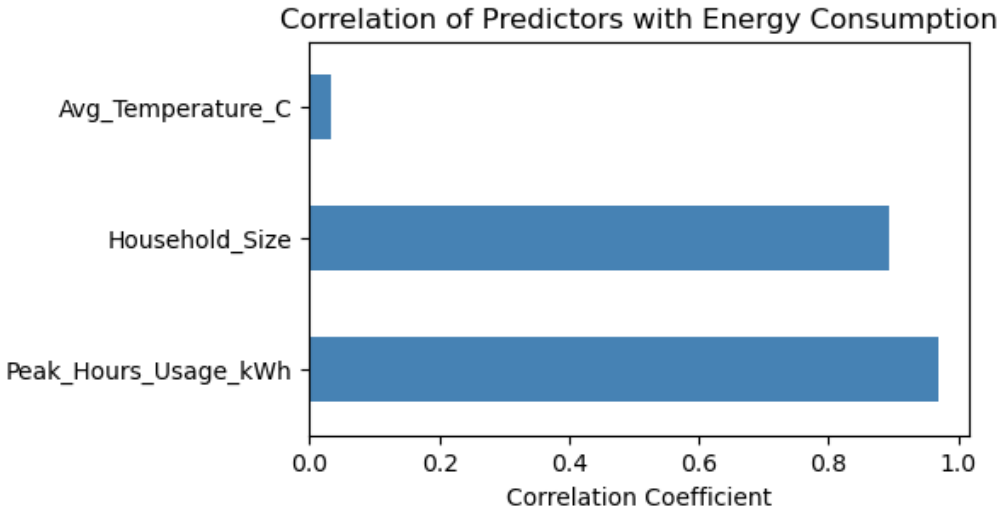
```
Avg_Temperature_C                  1.000                  0.032
Peak_Hours_Usage_kWh               0.032                  1.000
```

```
plt.figure(figsize=(6,5))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", vmin=-1, vmax=1,
linewidths=0.5)
plt.title("Correlation Matrix of Numerical Variables", fontsize=13)
plt.show()
```

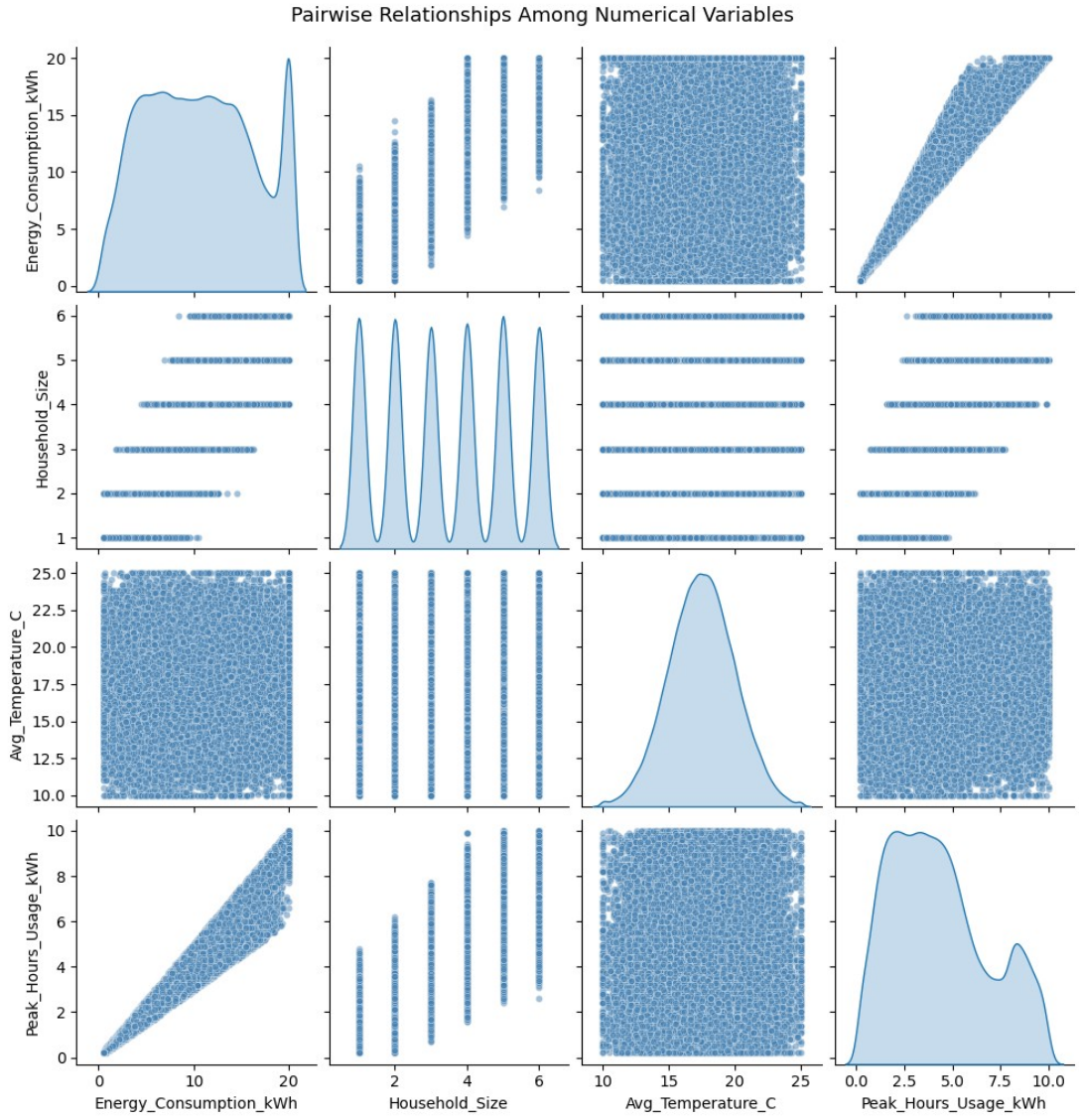



```
target_corr =
corr_matrix['Energy_Consumption_kWh'].drop('Energy_Consumption_kWh').s
ort_values(ascending=False)
plt.figure(figsize=(5,3))
target_corr.plot(kind='barh', color='steelblue')
plt.title("Correlation of Predictors with Energy Consumption")
```

```
plt.xlabel("Correlation Coefficient")
plt.show()
```

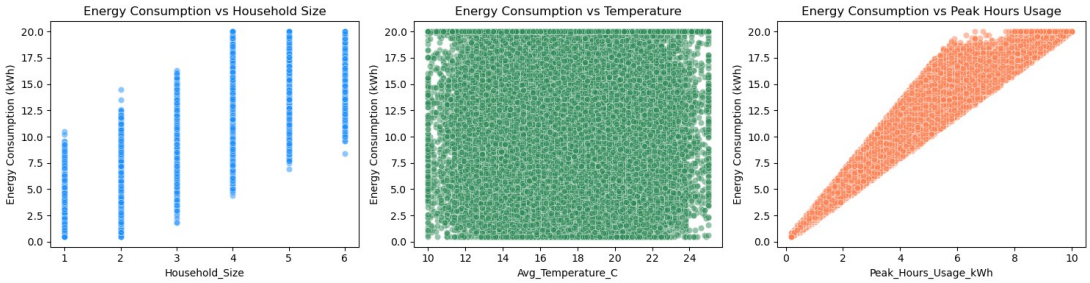



```
sns.pairplot(
    df,
    vars=['Energy_Consumption_kWh', 'Household_Size',
'Avg_Temperature_C', 'Peak_Hours_Usage_kWh'],
    diag_kind='kde',
    plot_kws={'alpha': 0.5, 's': 20, 'color': 'steelblue'}
)
plt.suptitle("Pairwise Relationships Among Numerical Variables",
y=1.02, fontsize=13)
plt.show()
```

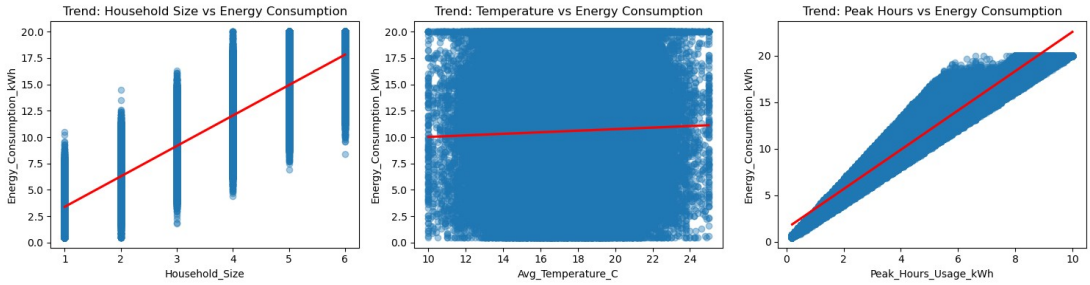



```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.scatterplot(ax=axes[0], x='Household_Size',
y='Energy_Consumption_kWh', data=df, alpha=0.5, color='dodgerblue')
axes[0].set_title("Energy Consumption vs Household Size")

sns.scatterplot(ax=axes[1], x='Avg_Temperature_C',
y='Energy_Consumption_kWh', data=df, alpha=0.5, color='seagreen')
axes[1].set_title("Energy Consumption vs Temperature")

sns.scatterplot(ax=axes[2], x='Peak_Hours_Usage_kWh',
```

```
y='Energy_Consumption_kWh', data=df, alpha=0.5, color='coral')
axes[2].set_title("Energy Consumption vs Peak Hours Usage")

for ax in axes:
    ax.set_xlabel(ax.get_xlabel())
    ax.set_ylabel("Energy Consumption (kWh)")

plt.tight_layout()
plt.show()
```



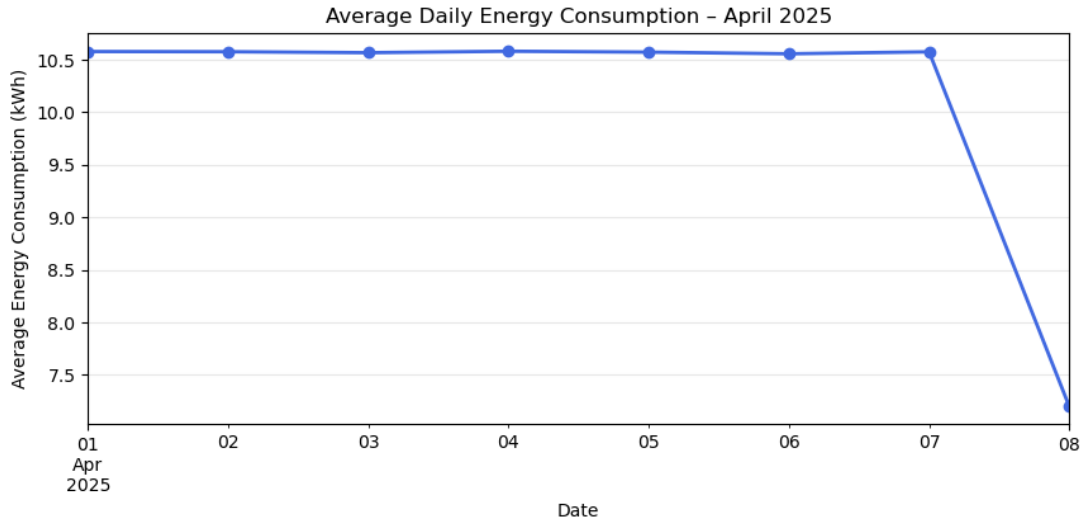

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.regplot(ax=axes[0], x='Household_Size',
y='Energy_Consumption_kWh', data=df, scatter_kws={'alpha':0.4},
line_kws={'color':'red'})
axes[0].set_title("Trend: Household Size vs Energy Consumption")

sns.regplot(ax=axes[1], x='Avg_Temperature_C',
y='Energy_Consumption_kWh', data=df, scatter_kws={'alpha':0.4},
line_kws={'color':'red'})
axes[1].set_title("Trend: Temperature vs Energy Consumption")

sns.regplot(ax=axes[2], x='Peak_Hours_Usage_kWh',
y='Energy_Consumption_kWh', data=df, scatter_kws={'alpha':0.4},
line_kws={'color':'red'})
axes[2].set_title("Trend: Peak Hours vs Energy Consumption")

plt.tight_layout()
plt.show()
```




```
# Convert to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Extract time features
df['DayOfWeek'] = df['Date'].dt.dayofweek        # Monday=0, Sunday=6
df['DayName'] = df['Date'].dt.day_name()
df['IsWeekend'] = df['DayOfWeek'].isin([5,6]).astype(int)

df[['Date','DayName','IsWeekend']].head()
```

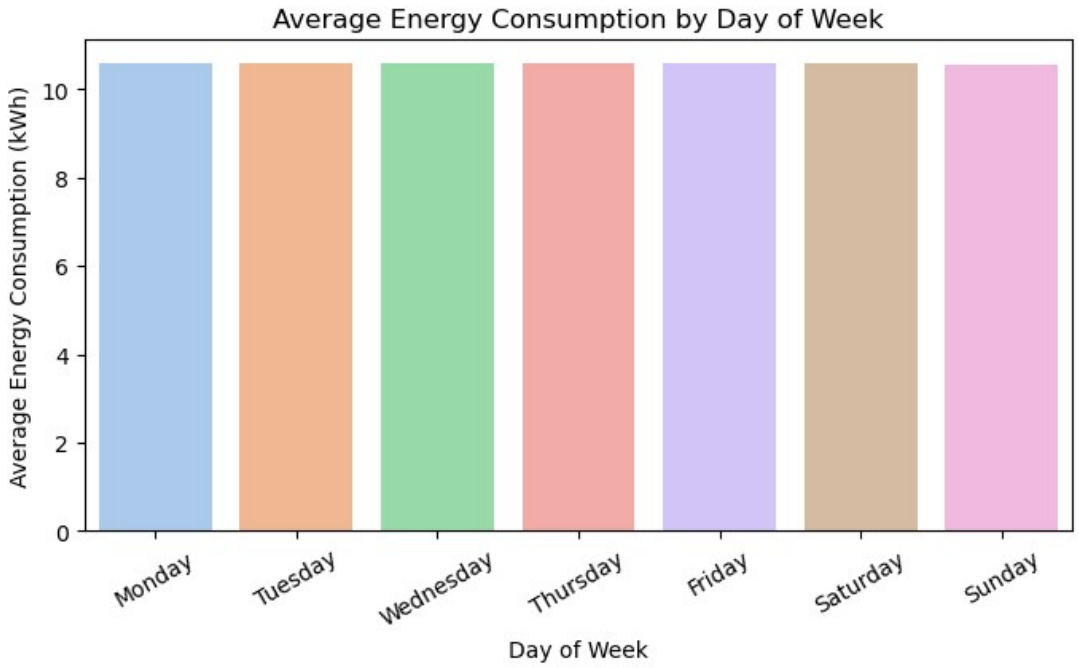
	Date	DayName	IsWeekend
0	2025-04-01	Tuesday	0
1	2025-04-02	Wednesday	0
2	2025-04-03	Thursday	0
3	2025-04-04	Friday	0
4	2025-04-05	Saturday	1

```
daily_avg = df.groupby('Date')['Energy_Consumption_kWh'].mean()

plt.figure(figsize=(10,4))
daily_avg.plot(color='royalblue', marker='o', linewidth=2)
plt.title("Average Daily Energy Consumption – April 2025")
plt.xlabel("Date")
plt.ylabel("Average Energy Consumption (kWh)")
plt.grid(alpha=0.3)
plt.show()
```

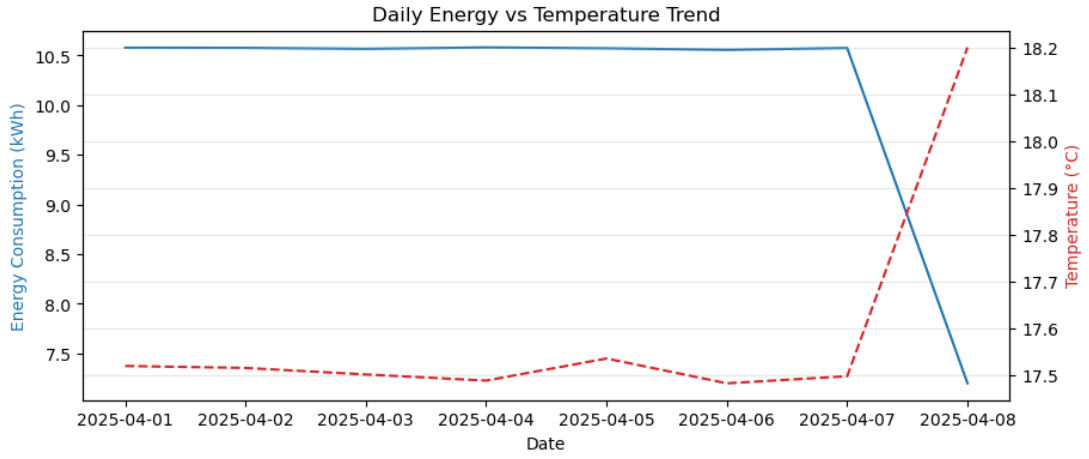



```
weekday_avg = df.groupby('DayName')
['Energy_Consumption_kWh'].mean().reindex(

['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday
']
)

plt.figure(figsize=(8,4))
sns.barplot(x=weekday_avg.index, y=weekday_avg.values,
palette='pastel')
plt.title("Average Energy Consumption by Day of Week")
plt.xlabel("Day of Week")
plt.ylabel("Average Energy Consumption (kWh)")
plt.xticks(rotation=30)
plt.show()
```

```
/tmp/ipykernel_1080044/3742888520.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=weekday_avg.index, y=weekday_avg.values,
palette='pastel')
```




```
temp_trend = df.groupby('Date')
[['Energy_Consumption_kWh','Avg_Temperature_C']].mean()

fig, ax1 = plt.subplots(figsize=(10,4))
ax2 = ax1.twinx()

ax1.plot(temp_trend.index, temp_trend['Energy_Consumption_kWh'],
color='tab:blue', label='Energy (kWh)')
ax2.plot(temp_trend.index, temp_trend['Avg_Temperature_C'],
color='tab:red', linestyle='--', label='Temperature (°C)')

ax1.set_xlabel("Date")
ax1.set_ylabel("Energy Consumption (kWh)", color='tab:blue')
ax2.set_ylabel("Temperature (°C)", color='tab:red')
plt.title("Daily Energy vs Temperature Trend")
plt.grid(alpha=0.3)
plt.show()
```




```
###Model selection######

# Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv("household_energy_consumption.csv")

# Define target and predictors
TARGET = "Energy_Consumption_kWh"
X = df.drop(columns=[TARGET, "Household_ID", "Date"])  # drop ID and
Date
y = df[TARGET]

# Identify feature types
num_features = ["Household_Size", "Avg_Temperature_C",
"Peak_Hours_Usage_kWh"]
cat_features = ["Has_AC"]

# Define transformers
num_transformer = StandardScaler()
cat_transformer = OneHotEncoder(drop="first")

# Combine transformers
preprocessor = ColumnTransformer(
```

```
    transformers=[
        ("num", num_transformer, num_features),
        ("cat", cat_transformer, cat_features)
    ]
)

# Build full pipeline
model_lr = Pipeline(steps=[
    ("prep", preprocessor),
    ("model", LinearRegression())
])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Cross-validation (5 folds) using negative RMSE
cv_scores = cross_val_score(
    model_lr, X_train, y_train,
    scoring="neg_root_mean_squared_error",
    cv=5
)

print(f"Cross-Validation RMSE scores: {-cv_scores}")
print(f"Mean CV RMSE: {-cv_scores.mean():.3f}")
```

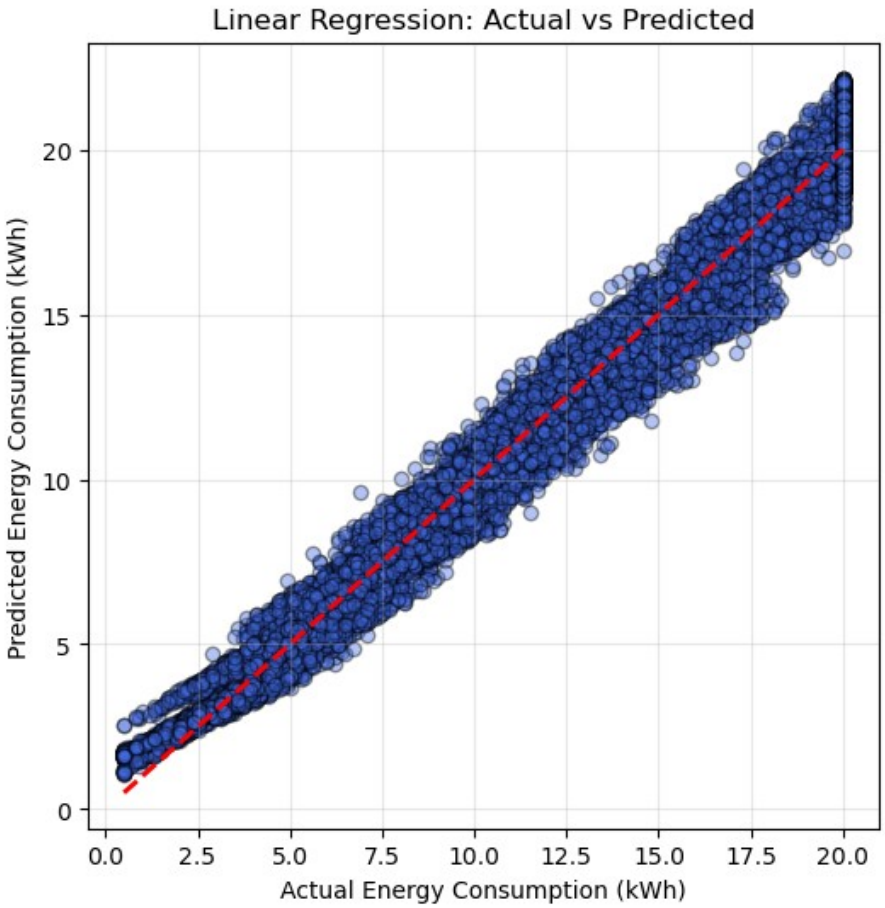
```
Cross-Validation RMSE scores: [0.79140797 0.79150306 0.78310514
0.79667975 0.79411654]
Mean CV RMSE: 0.791
```

```
# Train the model and make predictions
model_lr.fit(X_train, y_train)
y_pred = model_lr.predict(X_test)

# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Test RMSE: {rmse:.3f}")
print(f"Test MAE: {mae:.3f}")
print(f"Test R²: {r2:.3f}")
```

```
Test RMSE: 0.789
Test MAE: 0.627
Test R²: 0.980
```

```
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.4, color="royalblue",
edgecolor="black")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
```

```
'r--', lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
plt.title("Linear Regression: Actual vs Predicted")
plt.grid(alpha=0.3)
plt.show()
```




```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

model_ridge = Pipeline(steps=[
    ("prep", preprocessor),
    ("model", Ridge())
])

# Range of alpha values to test
param_grid = {'model__alpha': [0.1, 1, 5, 10, 20, 50, 100]}
```

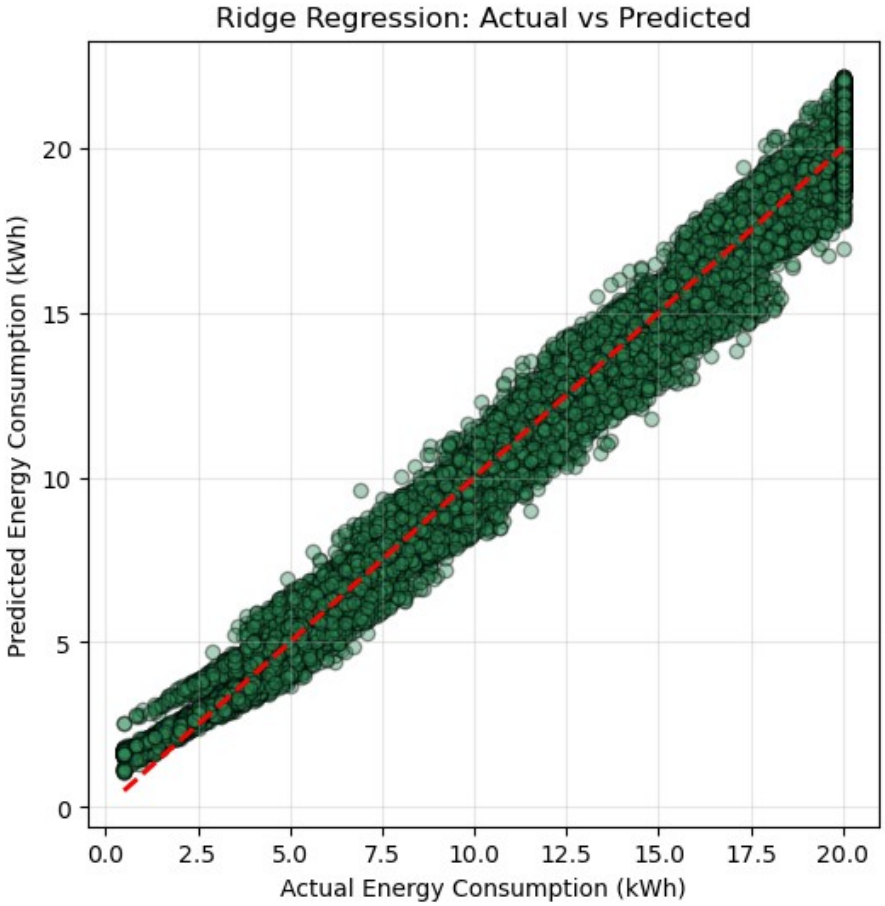
```
grid_ridge = GridSearchCV(
    model_ridge,
    param_grid,
    cv=5,
    scoring="neg_root_mean_squared_error",
    n_jobs=-1
)

grid_ridge.fit(X_train, y_train)

print("Best params:", grid_ridge.best_params_)
print(f"Best CV RMSE: {-grid_ridge.best_score_:.3f}")
```

```
Best params: {'model__alpha': 0.1}
Best CV RMSE: 0.791
```

```
# Use best model
final_ridge = grid_ridge.best_estimator_

# Predict
y_pred_ridge = final_ridge.predict(X_test)

# Metrics
rmse_ridge = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Test RMSE: {rmse_ridge:.3f}")
print(f"Test MAE: {mae_ridge:.3f}")
print(f"Test R²: {r2_ridge:.3f}")
```

```
Test RMSE: 0.789
Test MAE: 0.627
Test R²: 0.980
```

```
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_ridge, alpha=0.4, color="seagreen",
edgecolor="black")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
plt.title("Ridge Regression: Actual vs Predicted")
plt.grid(alpha=0.3)
plt.show()
```




```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Reuse: X, y, num_features, cat_features, preprocessor, X_train,
X_test, y_train, y_test
# If not in memory, run the setup cells from Model 1 again.

rf_pipe = Pipeline(steps=[
    ("prep", preprocessor),
    ("model", RandomForestRegressor(random_state=42, n_jobs=-1))
])

param_grid_rf = {
    "model__n_estimators": [200, 400],
    "model__max_depth": [None, 10, 20],
    "model__min_samples_split": [2, 5, 10],
    "model__min_samples_leaf": [1, 2, 4],
    "model__max_features": ["sqrt", "log2"]
```

```
}

grid_rf = GridSearchCV(
    rf_pipe,
    param_grid=param_grid_rf,
    cv=5,
    scoring="neg_root_mean_squared_error",
    n_jobs=-1,
    verbose=0
)

grid_rf.fit(X_train, y_train)

print("Best params:", grid_rf.best_params_)
print(f"Best CV RMSE: {-grid_rf.best_score_:.3f}")
```

```
/opt/conda/lib/python3.13/site-packages/joblib/externals/loky/
process_executor.py:782: UserWarning: A worker stopped while some jobs
were given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  warnings.warn(
```

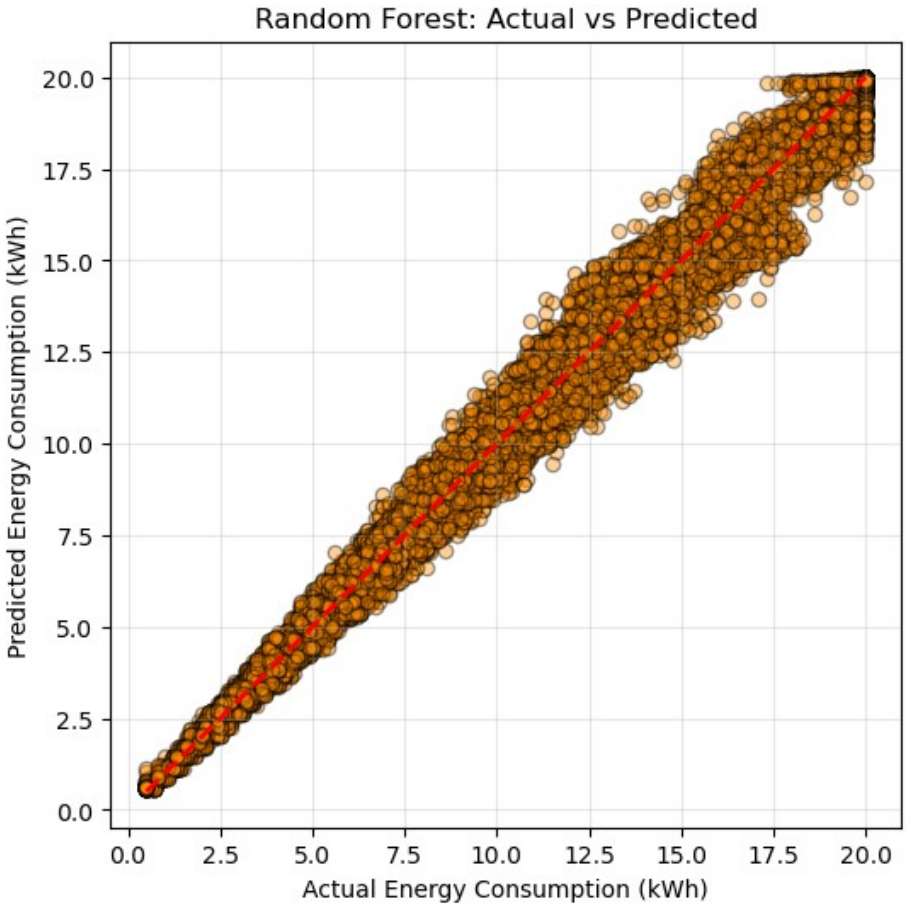
```
Best params: {'model__max_depth': 10, 'model__max_features': 'sqrt',
'model__min_samples_leaf': 1, 'model__min_samples_split': 2,
'model__n_estimators': 400}
Best CV RMSE: 0.642
```

```
final_rf = grid_rf.best_estimator_

y_pred_rf = final_rf.predict(X_test)

rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Test RMSE: {rmse_rf:.3f}")
print(f"Test MAE: {mae_rf:.3f}")
print(f"Test R²: {r2_rf:.3f}")
```

```
Test RMSE: 0.639
Test MAE: 0.478
Test R²: 0.987
```

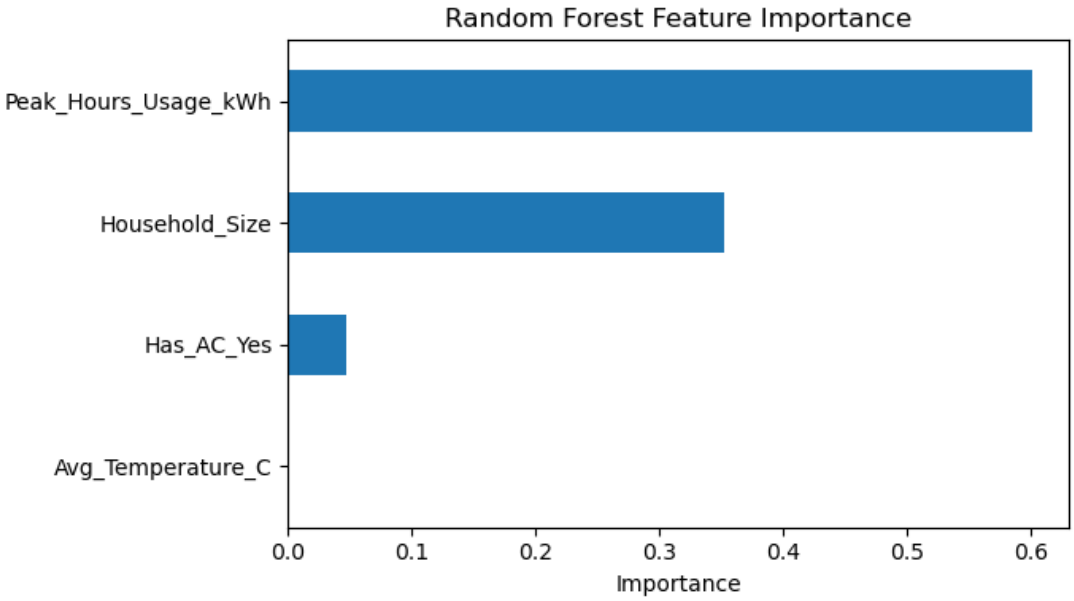
```
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_rf, alpha=0.4, color="darkorange",
edgecolor="black")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
"r--", lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
plt.title("Random Forest: Actual vs Predicted")
```

```
plt.grid(alpha=0.3)
plt.show()
```




```
# Get processed feature names
prep = final_rf.named_steps["prep"]

# numeric names
num_names = np.array(num_features)

# categorical names from encoder
cat_encoder = prep.named_transformers_["cat"]
cat_names = cat_encoder.get_feature_names_out(cat_features)

all_feature_names = np.r_[num_names, cat_names]

# Importances
rf_model = final_rf.named_steps["model"]
```

```
importances = pd.Series(rf_model.feature_importances_,
index=all_feature_names).sort_values(ascending=True)

plt.figure(figsize=(7,4))
importances.tail(10).plot(kind="barh")
plt.title("Random Forest Feature Importance")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()

importances.sort_values(ascending=False).head(10)
```



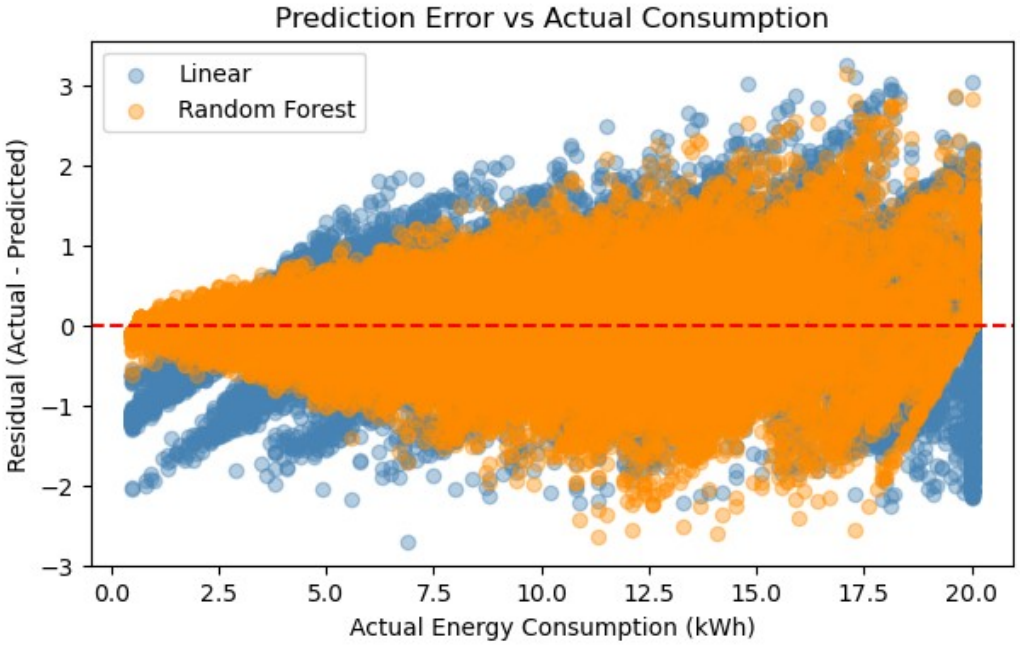

```
Peak_Hours_Usage_kWh    0.600217
Household_Size          0.351419
Has_AC_Yes              0.047244
Avg_Temperature_C       0.001120
dtype: float64

results = pd.DataFrame({
    "Model": ["Linear Regression", "Ridge Regression", "Random
Forest"],
    "Test RMSE": [rmse, rmse_ridge, rmse_rf],
    "Test MAE": [mae, mae_ridge, mae_rf],
    "Test R²": [r2, r2_ridge, r2_rf]
})
results.sort_values("Test RMSE")
```

```
               Model  Test RMSE  Test MAE   Test R²
2      Random Forest   0.639420  0.478343  0.986575
1   Ridge Regression   0.789351  0.627164  0.979541
0  Linear Regression   0.789351  0.627164  0.979541
```

```
results = pd.DataFrame({
    "Model": ["Linear Regression", "Ridge Regression", "Random
Forest"],
    "Test RMSE": [rmse, rmse_ridge, rmse_rf],
    "Test MAE": [mae, mae_ridge, mae_rf],
    "Test R²": [r2, r2_ridge, r2_rf]
})
results.sort_values("Test RMSE")
```

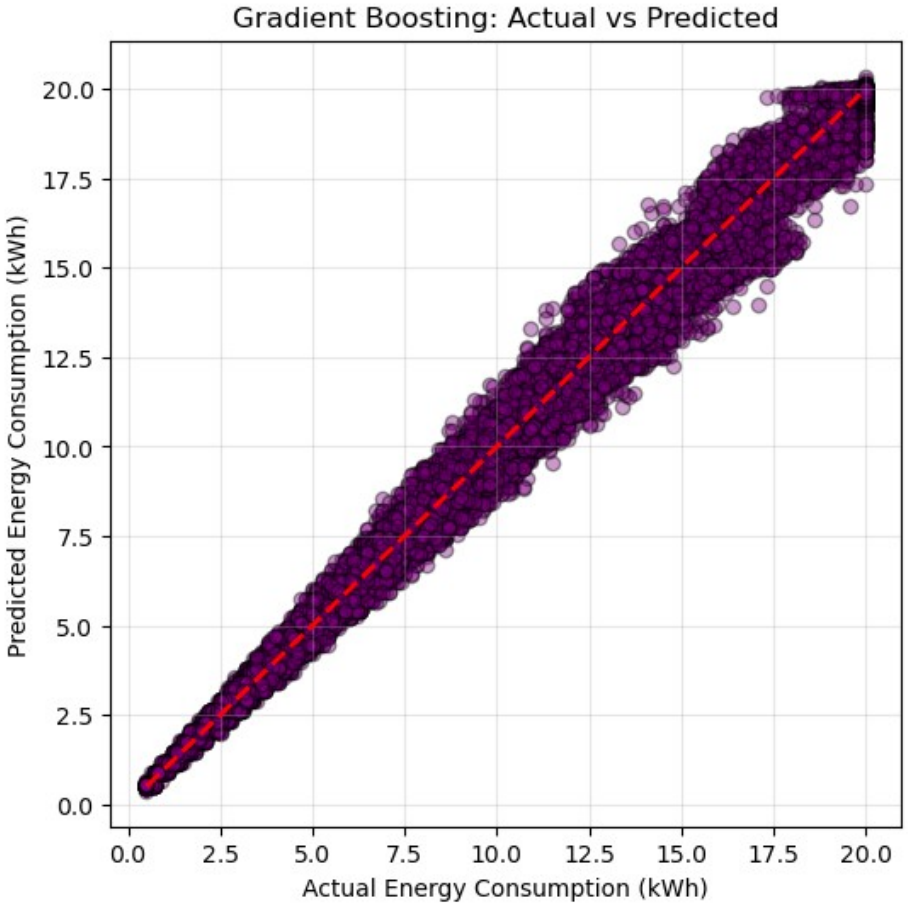
```
               Model  Test RMSE  Test MAE   Test R²
2      Random Forest   0.639420  0.478343  0.986575
1   Ridge Regression   0.789351  0.627164  0.979541
0  Linear Regression   0.789351  0.627164  0.979541
```

```
plt.figure(figsize=(7,4))
plt.scatter(y_test, y_test - y_pred, alpha=0.4, label="Linear",
color="steelblue")
plt.scatter(y_test, y_test - y_pred_rf, alpha=0.4, label="Random
Forest", color="darkorange")
plt.axhline(0, color="red", linestyle="--")
plt.title("Prediction Error vs Actual Consumption")
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Residual (Actual - Predicted)")
plt.legend()
plt.show()
```

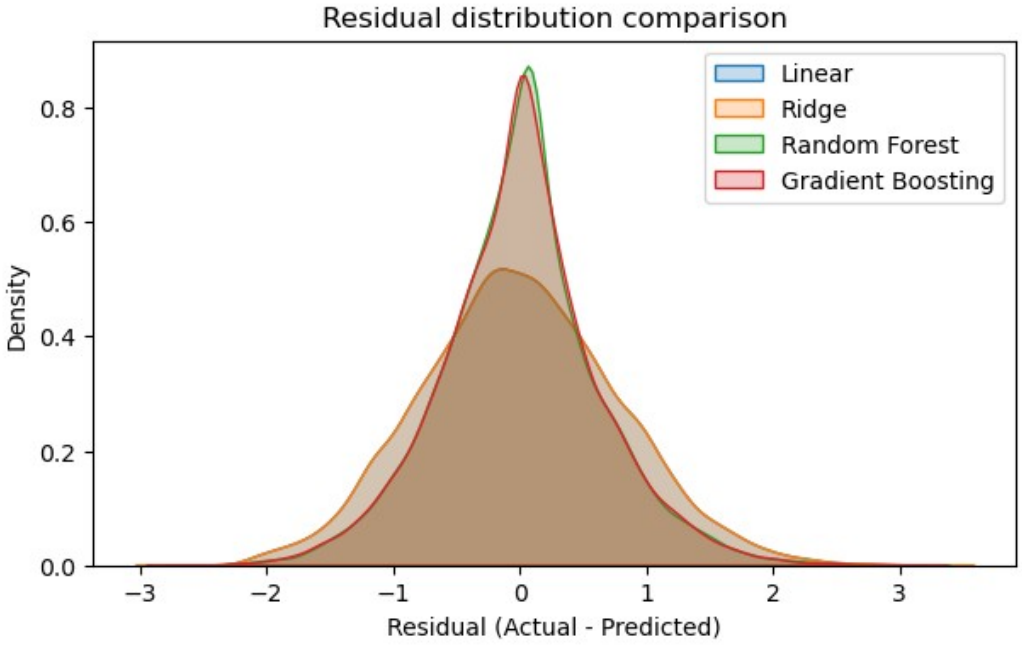



```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

gbr_pipe = Pipeline(steps=[
    ("prep", preprocessor),                    # same preprocessor as
before
    ("model", GradientBoostingRegressor(random_state=42))
])

param_grid_gbr = {
    "model__n_estimators": [200, 400],
    "model__learning_rate": [0.05, 0.1],
    "model__max_depth": [2, 3],
    "model__subsample": [0.8, 1.0],
    "model__max_features": ["sqrt", None]
}

grid_gbr = GridSearchCV(
    gbr_pipe,
    param_grid=param_grid_gbr,
    cv=5,
    scoring="neg_root_mean_squared_error",
    n_jobs=-1
)

grid_gbr.fit(X_train, y_train)
```

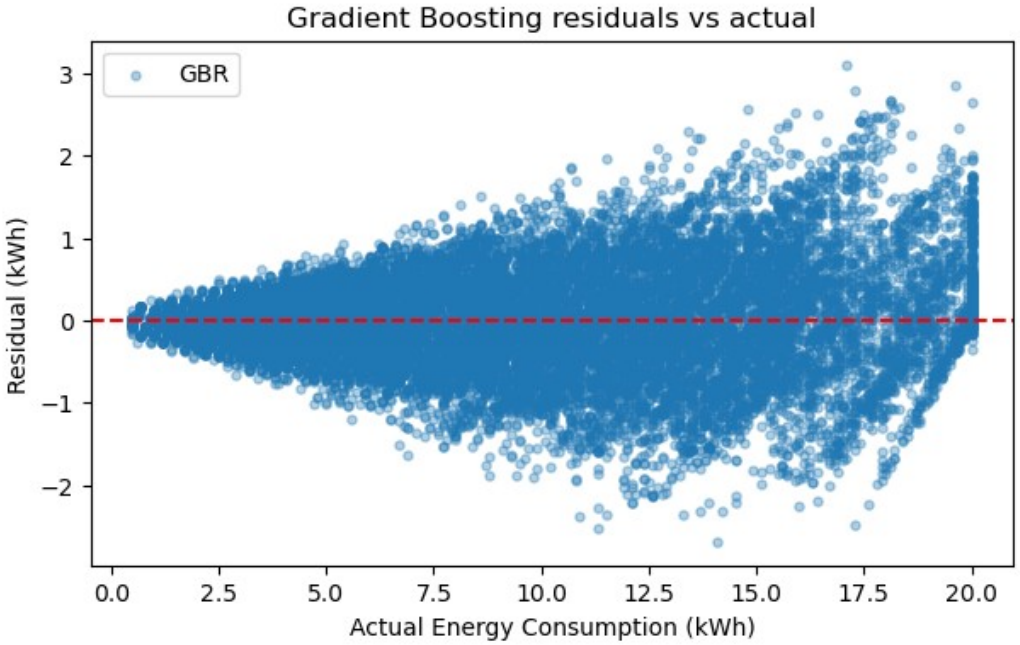
```
print("Best params:", grid_gbr.best_params_)
print(f"Best CV RMSE: {-grid_gbr.best_score_:.3f}")

Best params: {'model__learning_rate': 0.1, 'model__max_depth': 3,
'model__max_features': None, 'model__n_estimators': 400,
'model__subsample': 1.0}
Best CV RMSE: 0.640

final_gbr = grid_gbr.best_estimator_

y_pred_gbr = final_gbr.predict(X_test)

rmse_gbr = np.sqrt(mean_squared_error(y_test, y_pred_gbr))
mae_gbr = mean_absolute_error(y_test, y_pred_gbr)
r2_gbr = r2_score(y_test, y_pred_gbr)

print(f"Test RMSE: {rmse_gbr:.3f}")
print(f"Test MAE: {mae_gbr:.3f}")
print(f"Test R²: {r2_gbr:.3f}")

Test RMSE: 0.638
Test MAE: 0.478
Test R²: 0.987

plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_gbr, alpha=0.4, color="purple",
edgecolor="black")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
"r--", lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
plt.title("Gradient Boosting: Actual vs Predicted")
plt.grid(alpha=0.3)
plt.show()
```




```
plt.figure(figsize=(7,4))
sns.kdeplot(y_test - y_pred, label="Linear", fill=True)
sns.kdeplot(y_test - y_pred_ridge, label="Ridge", fill=True)
sns.kdeplot(y_test - y_pred_rf, label="Random Forest", fill=True)
sns.kdeplot(y_test - y_pred_gbr, label="Gradient Boosting", fill=True)
plt.title("Residual distribution comparison")
plt.xlabel("Residual (Actual - Predicted)")
plt.legend()
plt.show()
```




```
plt.figure(figsize=(7,4))
plt.scatter(y_test, y_test - y_pred_gbr, alpha=0.35, s=14,
label="GBR")
plt.axhline(0, color="red", linestyle="--")
plt.title("Gradient Boosting residuals vs actual")
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Residual (kWh)")
plt.legend()
plt.show()
```

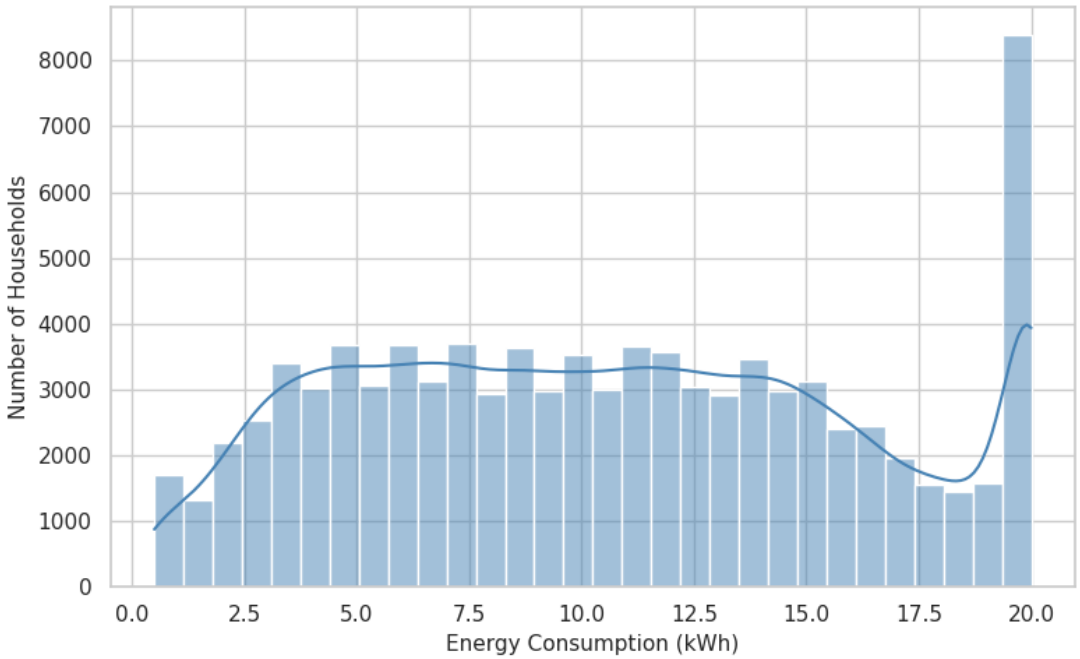



```
results = pd.DataFrame({
    "Model": ["Linear", "Ridge", "Random Forest", "Gradient
Boosting"],
    "Test RMSE": [rmse, rmse_ridge, rmse_rf, rmse_gbr],
    "Test MAE":  [mae,  mae_ridge,  mae_rf,  mae_gbr],
    "Test R²":   [r2,   r2_ridge,   r2_rf,   r2_gbr]
}).sort_values("Test RMSE")
results
```

	Model	Test RMSE	Test MAE	Test R²
3	Gradient Boosting	0.637960	0.478178	0.986636
2	Random Forest	0.639420	0.478343	0.986575
1	Ridge	0.789351	0.627164	0.979541
0	Linear	0.789351	0.627164	0.979541

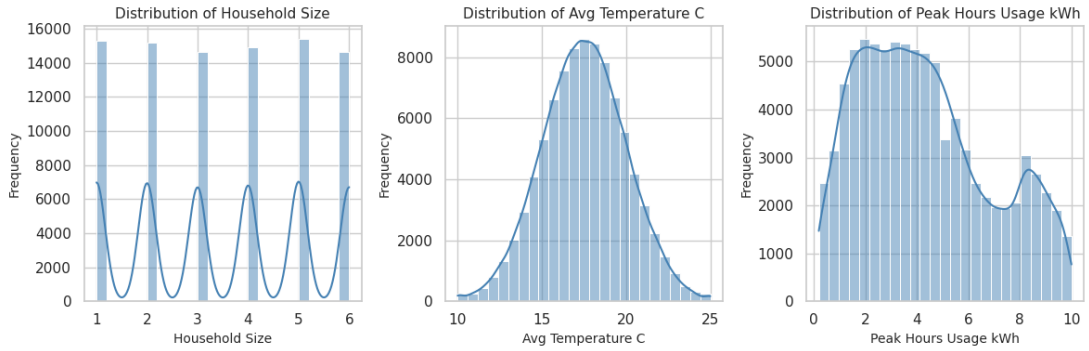
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style for a clean academic look
sns.set(style="whitegrid")

# Create the figure
plt.figure(figsize=(8,5))
sns.histplot(df['Energy_Consumption_kWh'], bins=30, kde=True,
color='steelblue')

# Titles and labels
```

```
#plt.title("Distribution of Daily Household Energy Consumption",
fontsize=13)
plt.xlabel("Energy Consumption (kWh)", fontsize=11)
plt.ylabel("Number of Households", fontsize=11)

# Adjust layout
plt.tight_layout()

# Save as PDF
plt.savefig("energy_consumption_histogram.pdf", format='pdf', dpi=300)

# Show the plot
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Set a clean visual style
sns.set(style="whitegrid")

# List of numeric columns
num_features = ['Household_Size', 'Avg_Temperature_C',
'Peak_Hours_Usage_kWh']

# Create subplots: 1 row, 3 columns
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
```

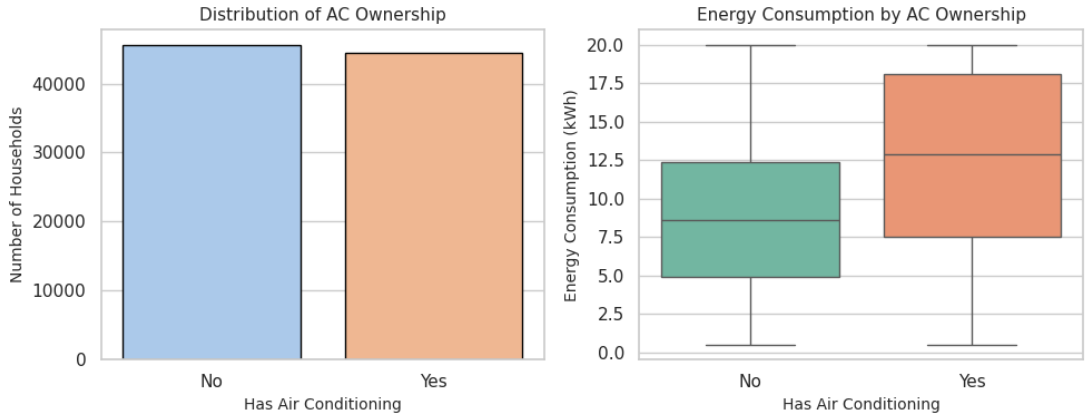
```
# Plot each histogram
for ax, col in zip(axes, num_features):
    sns.histplot(df[col], bins=25, kde=True, color='steelblue', ax=ax)
    ax.set_title(f"Distribution of {col.replace('_', ' ')}",
fontsize=11)
    ax.set_xlabel(col.replace('_', ' '), fontsize=10)
    ax.set_ylabel("Frequency", fontsize=10)

# Adjust layout for clarity
plt.tight_layout()

# Save as PDF (high quality for Overleaf)
plt.savefig("numeric_predictors_histograms.pdf", format='pdf',
dpi=300)

# Display the combined plot
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Clean style for publication
sns.set(style="whitegrid")

# Create subplots: 1 row, 2 columns
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# 1. Bar plot: number of households with/without AC
sns.countplot(x='Has_AC', data=df, palette='pastel',
edgecolor='black', ax=axes[0])
axes[0].set_title("Distribution of AC Ownership", fontsize=11)
axes[0].set_xlabel("Has Air Conditioning", fontsize=10)
axes[0].set_ylabel("Number of Households", fontsize=10)

# 2. Box plot: energy use by AC ownership
sns.boxplot(x='Has_AC', y='Energy_Consumption_kWh', data=df,
```

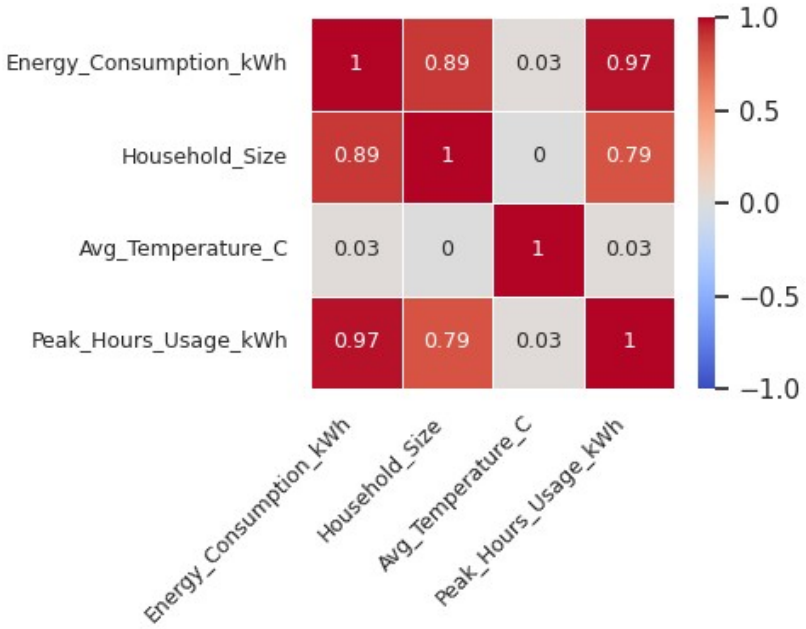
```
palette='Set2', ax=axes[1])
axes[1].set_title("Energy Consumption by AC Ownership", fontsize=11)
axes[1].set_xlabel("Has Air Conditioning", fontsize=10)
axes[1].set_ylabel("Energy Consumption (kWh)", fontsize=10)

# Adjust layout
plt.tight_layout()

# Save as PDF for Overleaf
plt.savefig("has_ac_comparison.pdf", format='pdf', dpi=300)

# Show plot
plt.show()
```

```
/tmp/ipykernel_1080044/913463421.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Has_AC', data=df, palette='pastel',
edgecolor='black', ax=axes[0])
/tmp/ipykernel_1080044/913463421.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='Has_AC', y='Energy_Consumption_kWh', data=df,
palette='Set2', ax=axes[1])
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Select numeric columns only
```

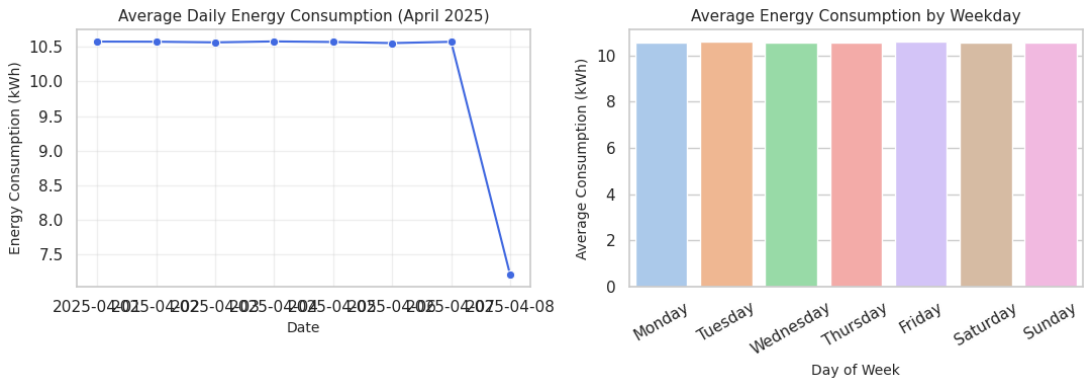
```
num_features = ['Energy_Consumption_kWh', 'Household_Size',
'Avg_Temperature_C', 'Peak_Hours_Usage_kWh']
corr = df[num_features].corr().round(2)

# Set style
sns.set(style="whitegrid")

# Create figure
plt.figure(figsize=(5,4))
ax = sns.heatmap(
    corr,
    annot=True,          # show correlation values
    cmap="coolwarm",
    vmin=-1, vmax=1,
    linewidths=0.5,
    annot_kws={"size": 9}
)

# Rotate labels to save space
plt.xticks(rotation=45, ha="right", fontsize=9)
plt.yticks(rotation=0, fontsize=9)

# Titles
#plt.title("Correlation Matrix of Numerical Variables", fontsize=11,
pad=10)

# Adjust layout
plt.tight_layout()

# Save as PDF
plt.savefig("correlation_heatmap.pdf", format='pdf', dpi=300)

# Show plot
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Make sure the Date column is in datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Add weekday column
df['DayName'] = df['Date'].dt.day_name()

# Compute daily averages
daily_avg = df.groupby('Date')
['Energy_Consumption_kWh'].mean().reset_index()

# Compute weekday averages (ordered)
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']
weekday_avg = df.groupby('DayName')
['Energy_Consumption_kWh'].mean().reindex(weekday_order)

# Set style
sns.set(style="whitegrid")

# Create figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(11, 4))

# --- Left plot: Daily trend ---
```

```
sns.lineplot(data=daily_avg, x='Date', y='Energy_Consumption_kWh',
marker='o', color='royalblue', ax=axes[0])
axes[0].set_title("Average Daily Energy Consumption (April 2025)",
fontsize=11)
axes[0].set_xlabel("Date", fontsize=10)
axes[0].set_ylabel("Energy Consumption (kWh)", fontsize=10)
axes[0].grid(alpha=0.3)

# --- Right plot: Weekday average ---
sns.barplot(x=weekday_avg.index, y=weekday_avg.values,
palette='pastel', ax=axes[1])
axes[1].set_title("Average Energy Consumption by Weekday",
fontsize=11)
axes[1].set_xlabel("Day of Week", fontsize=10)
axes[1].set_ylabel("Average Consumption (kWh)", fontsize=10)
axes[1].tick_params(axis='x', rotation=30)

# Adjust layout
plt.tight_layout()

# Save as PDF for Overleaf
plt.savefig("daily_weekday_energy.pdf", format='pdf', dpi=300)

# Show plots
plt.show()
```

```
/tmp/ipykernel_1080044/3048891584.py:32: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=weekday_avg.index, y=weekday_avg.values,
palette='pastel', ax=axes[1])
```



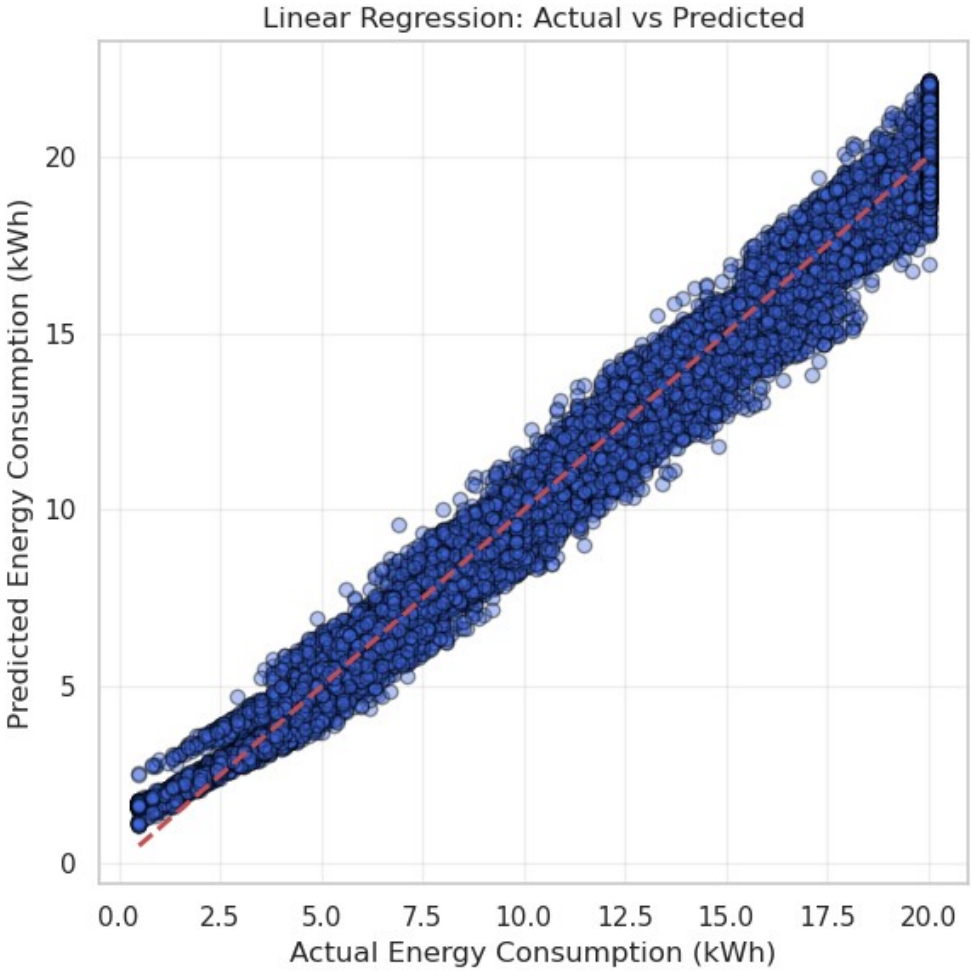
```
# ======================================================
# 1. Imports
# ======================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# ======================================================
# 2. Load and prepare data
# ======================================================
df = pd.read_csv("household_energy_consumption.csv")

# Define target and predictors
TARGET = "Energy_Consumption_kWh"
X = df.drop(columns=[TARGET, "Household_ID", "Date"])
y = df[TARGET]

# Identify feature types
num_features = ["Household_Size", "Avg_Temperature_C",
"Peak_Hours_Usage_kWh"]
cat_features = ["Has_AC"]

# ======================================================
# 3. Preprocessing pipeline
# ======================================================
num_transformer = StandardScaler()
cat_transformer = OneHotEncoder(drop="first")

preprocessor = ColumnTransformer(
    transformers=[
        ("num", num_transformer, num_features),
        ("cat", cat_transformer, cat_features)
    ]
)

# ======================================================
# 4. Define model pipeline
# ======================================================
model_lr = Pipeline(steps=[
    ("prep", preprocessor),
    ("model", LinearRegression())
```

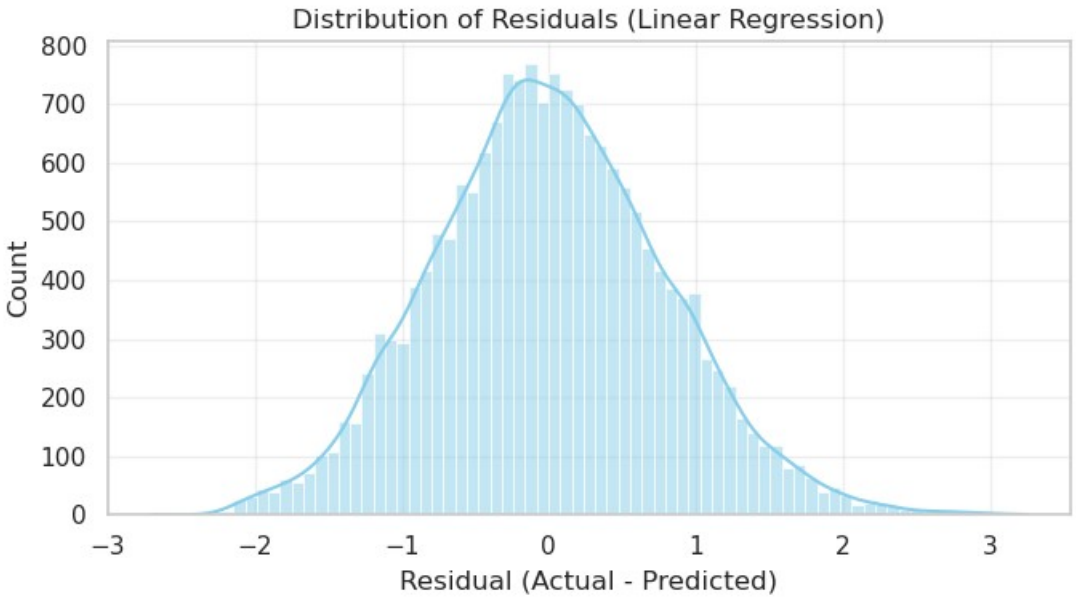
```
])

# ======================================================
# 5. Train-test split
# ======================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ======================================================
# 6. Cross-validation
# ======================================================
cv_scores = cross_val_score(
    model_lr, X_train, y_train,
    scoring="neg_root_mean_squared_error",
    cv=5
)

print(f"Cross-validation RMSE: {-cv_scores}")
print(f"Mean CV RMSE: {-cv_scores.mean():.3f}")

# ======================================================
# 7. Final training and evaluation
# ======================================================
model_lr.fit(X_train, y_train)
y_pred = model_lr.predict(X_test)

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

print(f"Test RMSE: {rmse:.3f}")
print(f"Test MAE: {mae:.3f}")
print(f"Test R²: {r2:.3f}")

# ======================================================
# 8. Create results summary table
# ======================================================
results_lr = pd.DataFrame({
    "Metric": ["RMSE", "MAE", "R²"],
    "Cross-validation Mean": [(-cv_scores.mean()), None, None],
    "Test Value": [rmse, mae, r2]
})
display(results_lr)

# ======================================================
# 9. Actual vs Predicted plot
# ======================================================
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.4, color="royalblue",
```

```
edgecolor="black")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
plt.title("Linear Regression: Actual vs Predicted")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig("Linear regression.pdf", format='pdf', dpi=300)
plt.show()

# ======================================================
# 10. Residual distribution plot
# ======================================================
residuals = y_test - y_pred
plt.figure(figsize=(7,4))
sns.histplot(residuals, kde=True, color="skyblue")
plt.title("Distribution of Residuals (Linear Regression)")
plt.xlabel("Residual (Actual - Predicted)")
plt.grid(alpha=0.3)
plt.tight_layout()

plt.show()
```

```
Cross-validation RMSE: [0.79140797 0.79150306 0.78310514 0.79667975
0.79411654]
Mean CV RMSE: 0.791
Test RMSE: 0.789
Test MAE: 0.627
Test R²: 0.980
```

	Metric	Cross-validation Mean	Test Value
0	RMSE	0.791362	0.789351
1	MAE	NaN	0.627164
2	R²	NaN	0.979541


Linear Regression: Actual vs Predicted
Predicted Energy Consumption (kWh)
Actual Energy Consumption (kWh)
0
5
10
15
20
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0




```
# ====================================================
# Ridge Regression Setup
# ====================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# ====================================================
# 1. Load data
# ====================================================
df = pd.read_csv("household_energy_consumption.csv")

# Define target and predictors
TARGET = "Energy_Consumption_kWh"
X = df.drop(columns=[TARGET, "Household_ID", "Date"])
y = df[TARGET]

# Identify feature types
```

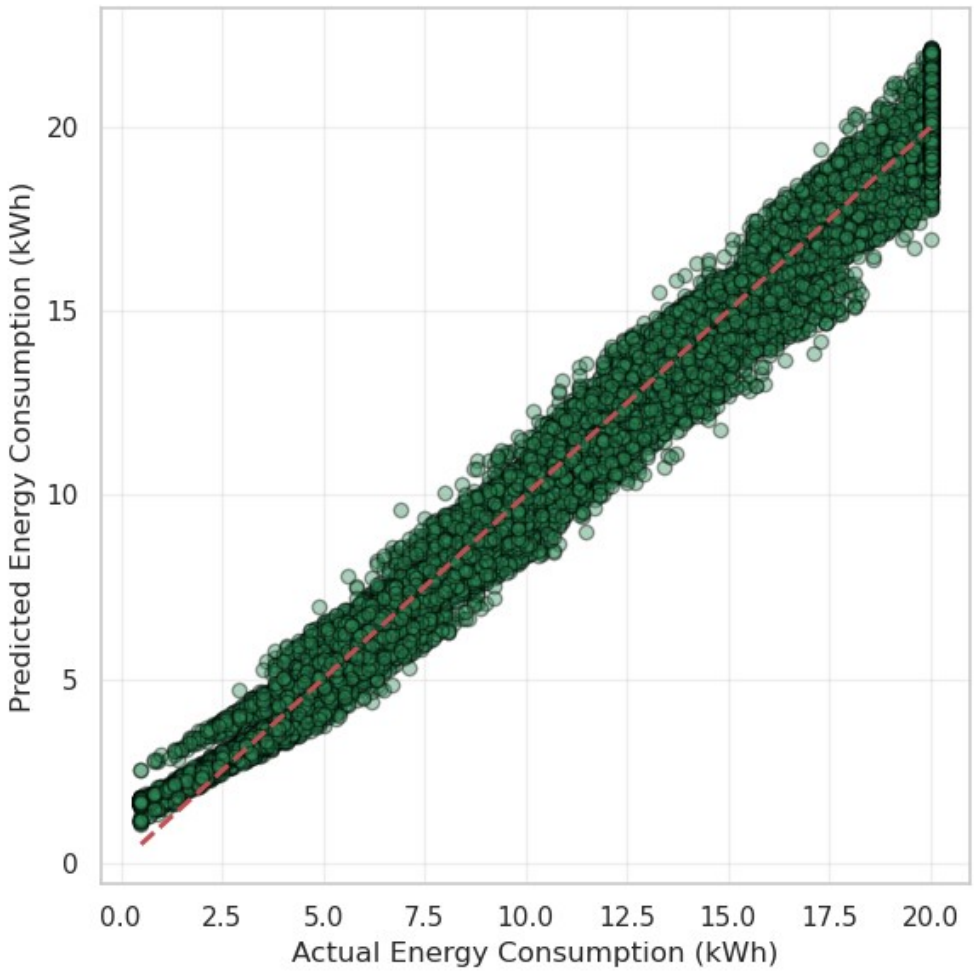
```
num_features = ["Household_Size", "Avg_Temperature_C",
"Peak_Hours_Usage_kWh"]
cat_features = ["Has_AC"]

# ========================================================
# 2. Preprocessing pipeline
# ========================================================
num_transformer = StandardScaler()
cat_transformer = OneHotEncoder(drop="first")

preprocessor = ColumnTransformer(
    transformers=[
        ("num", num_transformer, num_features),
        ("cat", cat_transformer, cat_features)
    ]
)

# ========================================================
# 3. Define Ridge model with pipeline
# ========================================================
ridge_pipe = Pipeline(steps=[
    ("prep", preprocessor),
    ("model", Ridge())
])

# ========================================================
# 4. Split data
# ========================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ========================================================
# 5. Grid search for best alpha (λ)
# ========================================================
param_grid = {"model__alpha": [0.1, 1, 5, 10, 20, 50, 100]}

grid_ridge = GridSearchCV(
    ridge_pipe,
    param_grid,
    cv=5,
    scoring="neg_root_mean_squared_error",
    n_jobs=-1
)

grid_ridge.fit(X_train, y_train)

print("Best alpha (λ):", grid_ridge.best_params_)
print(f"Best CV RMSE: {-grid_ridge.best_score_:.3f}")
```

```python
# ======================================================
# 6. Final evaluation on test data
# ======================================================
final_ridge = grid_ridge.best_estimator_
y_pred_ridge = final_ridge.predict(X_test)

rmse_ridge = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Test RMSE: {rmse_ridge:.3f}")
print(f"Test MAE: {mae_ridge:.3f}")
print(f"Test R²: {r2_ridge:.3f}")

# ======================================================
# 7. Summary results table
# ======================================================
results_ridge = pd.DataFrame({
    "Metric": ["RMSE", "MAE", "R²"],
    "Cross-validation Mean": [(-grid_ridge.best_score_), None, None],
    "Test Value": [rmse_ridge, mae_ridge, r2_ridge]
})
display(results_ridge)

# ======================================================
# 8. Predicted vs Actual Plot
# ======================================================
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_ridge, alpha=0.4, color="seagreen",
edgecolor="black")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
#plt.title("Ridge Regression: Actual vs Predicted")
plt.grid(alpha=0.3)
plt.savefig("Ridge regression.pdf", format='pdf', dpi=300)
plt.tight_layout()
plt.show()
```

```
Best alpha (λ): {'model__alpha': 0.1}
Best CV RMSE: 0.791
Test RMSE: 0.789
Test MAE: 0.627
Test R²: 0.980
```

	Metric	Cross-validation Mean	Test Value
0	RMSE	0.791362	0.789351

```
1    MAE                    NaN    0.627164
2     R²                    NaN    0.979541
```




```
# ====================================================
# Random Forest Regression Setup
# ====================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

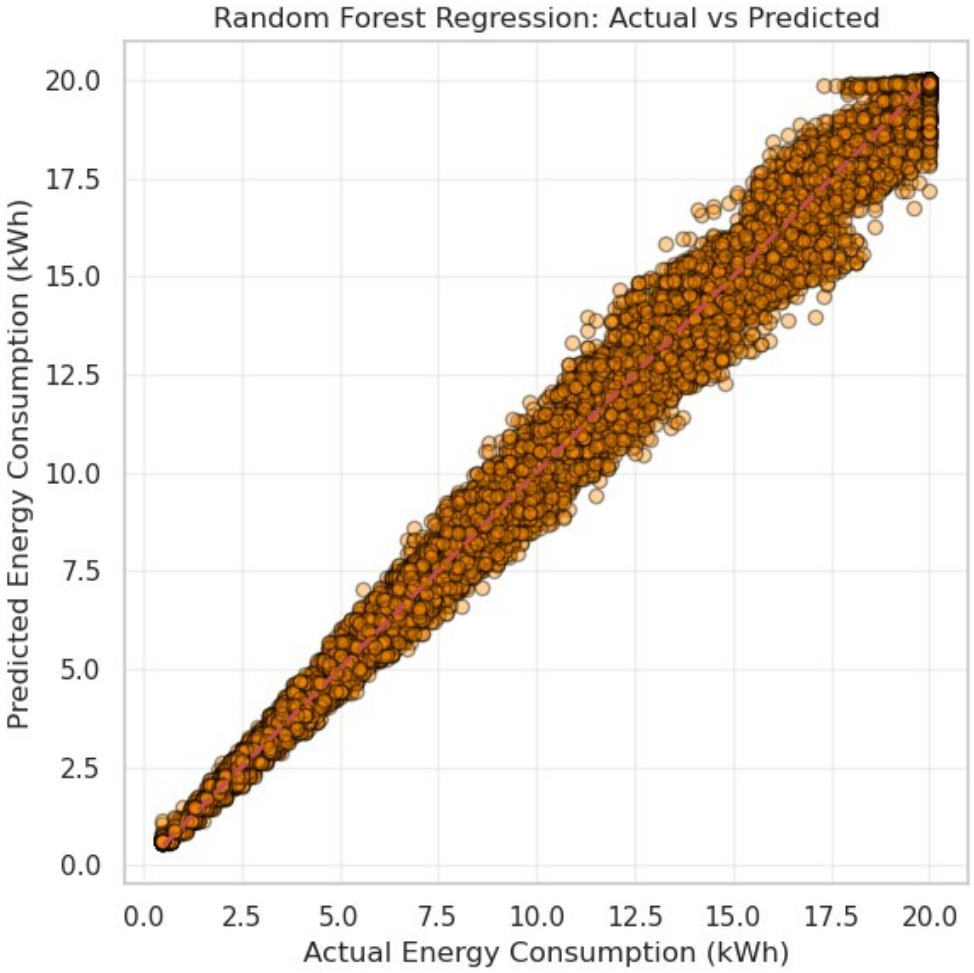
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# =======================================================
# 1. Load and prepare data
# =======================================================
df = pd.read_csv("household_energy_consumption.csv")

TARGET = "Energy_Consumption_kWh"
X = df.drop(columns=[TARGET, "Household_ID", "Date"])
y = df[TARGET]

num_features = ["Household_Size", "Avg_Temperature_C",
"Peak_Hours_Usage_kWh"]
cat_features = ["Has_AC"]

# =======================================================
# 2. Preprocessing pipeline
# =======================================================
num_transformer = StandardScaler()
cat_transformer = OneHotEncoder(drop="first")

preprocessor = ColumnTransformer(
    transformers=[
        ("num", num_transformer, num_features),
        ("cat", cat_transformer, cat_features)
    ]
)

# =======================================================
# 3. Define model pipeline
# =======================================================
rf_pipe = Pipeline(steps=[
    ("prep", preprocessor),
    ("model", RandomForestRegressor(random_state=42, n_jobs=-1))
])

# =======================================================
# 4. Train-test split
# =======================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# =======================================================
# 5. Grid Search for Hyperparameter Tuning
# =======================================================
param_grid_rf = {
    "model__n_estimators": [200, 400],
```

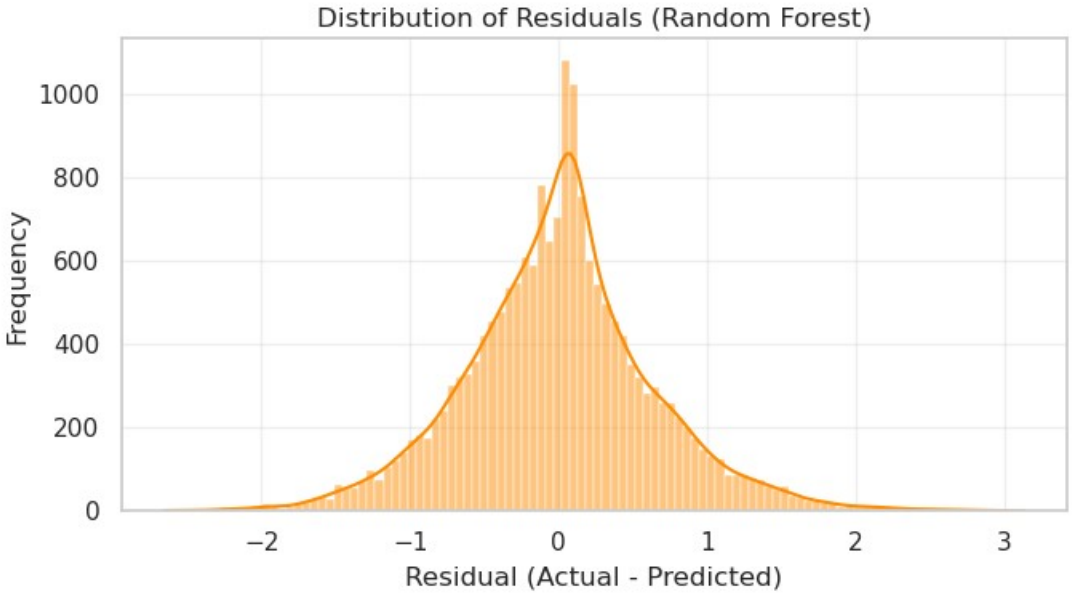
```
    "model__max_depth": [None, 10, 20],
    "model__min_samples_split": [2, 5],
    "model__min_samples_leaf": [1, 2],
    "model__max_features": ["sqrt", "log2"]
}

grid_rf = GridSearchCV(
    rf_pipe,
    param_grid=param_grid_rf,
    cv=5,
    scoring="neg_root_mean_squared_error",
    n_jobs=-1,
    verbose=1
)

grid_rf.fit(X_train, y_train)

print("Best parameters:", grid_rf.best_params_)
print(f"Best CV RMSE: {-grid_rf.best_score_:.3f}")

# ======================================================
# 6. Final Model Evaluation
# ======================================================
final_rf = grid_rf.best_estimator_
y_pred_rf = final_rf.predict(X_test)

rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Test RMSE: {rmse_rf:.3f}")
print(f"Test MAE: {mae_rf:.3f}")
print(f"Test R²: {r2_rf:.3f}")

# ======================================================
# 7. Results Summary Table
# ======================================================
results_rf = pd.DataFrame({
    "Metric": ["RMSE", "MAE", "R²"],
    "Cross-validation Mean": [(-grid_rf.best_score_), None, None],
    "Test Value": [rmse_rf, mae_rf, r2_rf]
})
display(results_rf)

# ======================================================
# 8. Predicted vs Actual Plot
# ======================================================
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_rf, alpha=0.4, color="darkorange",
edgecolor="black")
```

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
plt.title("Random Forest Regression: Actual vs Predicted")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# ======================================================
# 9. Residual Distribution
# ======================================================
residuals_rf = y_test - y_pred_rf
plt.figure(figsize=(7,4))
sns.histplot(residuals_rf, kde=True, color="darkorange")
plt.title("Distribution of Residuals (Random Forest)")
plt.xlabel("Residual (Actual - Predicted)")
plt.ylabel("Frequency")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

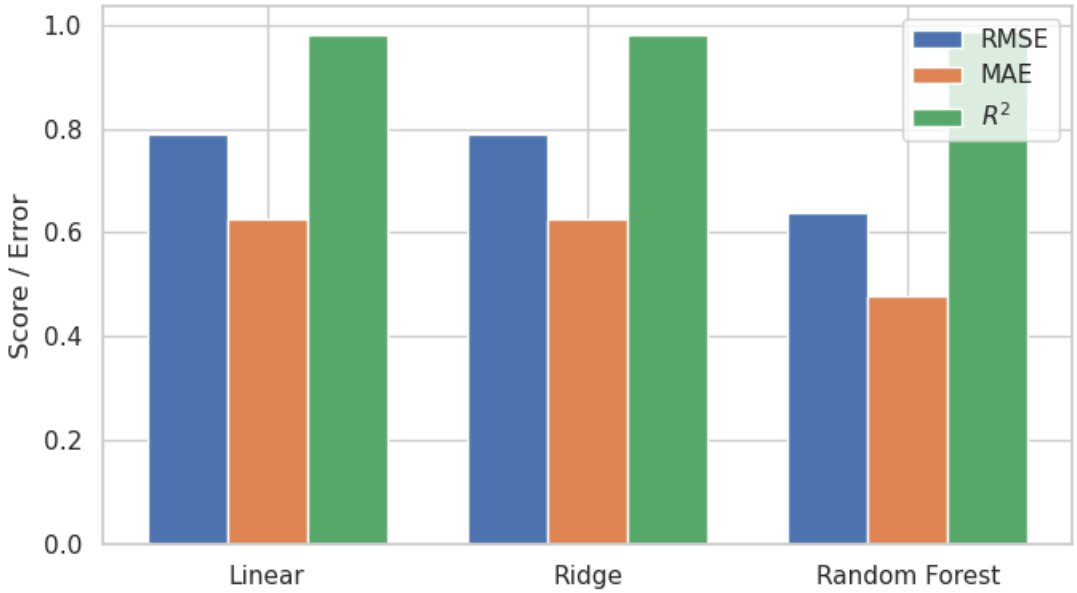
```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best parameters: {'model__max_depth': 10, 'model__max_features':
'sqrt', 'model__min_samples_leaf': 1, 'model__min_samples_split': 2,
'model__n_estimators': 400}
Best CV RMSE: 0.642
Test RMSE: 0.639
Test MAE: 0.478
Test R²: 0.987
```

```
  Metric  Cross-validation Mean  Test Value
0   RMSE               0.642465    0.639420
1    MAE                    NaN    0.478343
2     R²                    NaN    0.986575
```

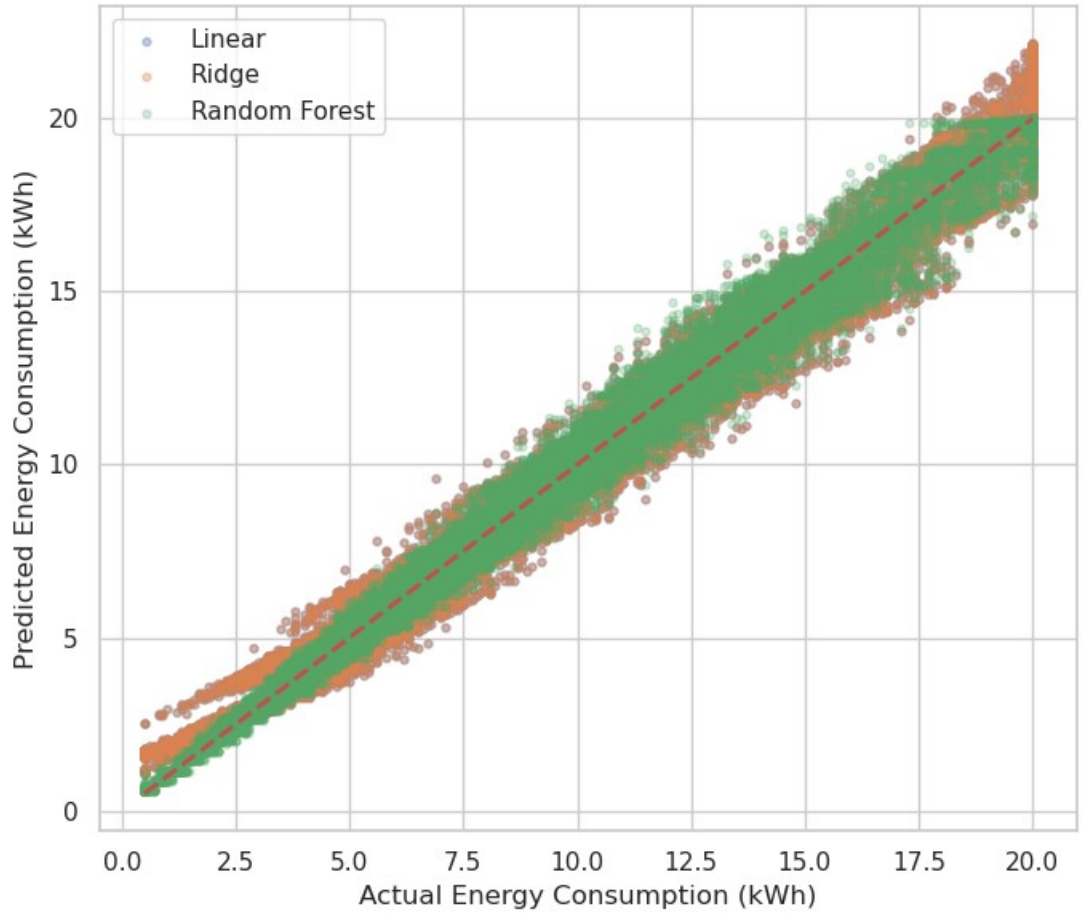

Random Forest Regression: Actual vs Predicted
Predicted Energy Consumption (kWh)
Actual Energy Consumption (kWh)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



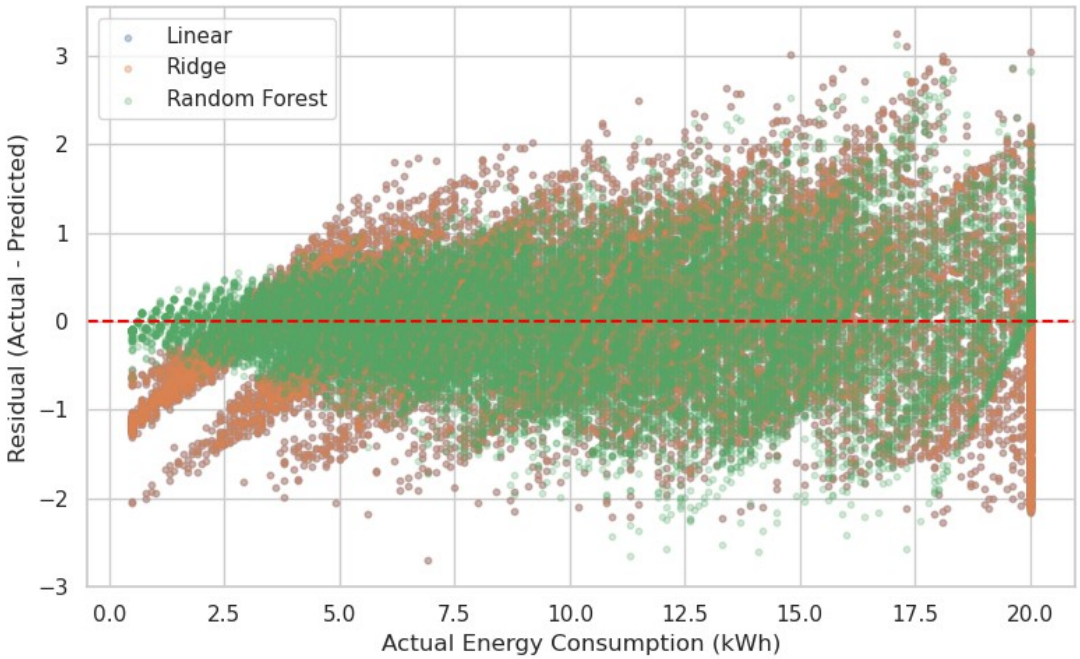

```
import matplotlib.pyplot as plt

models = ["Linear", "Ridge", "Random Forest"]
rmse = [0.789, 0.789, 0.639]
mae = [0.627, 0.627, 0.478]
r2  = [0.980, 0.980, 0.987]

x = np.arange(len(models))
width = 0.25

fig, ax = plt.subplots(figsize=(7,4))
ax.bar(x - width, rmse, width, label="RMSE")
ax.bar(x, mae, width, label="MAE")
ax.bar(x + width, r2, width, label="$R^2$")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.set_ylabel("Score / Error")
#ax.set_title("Comparison of Model Performance")
ax.legend()
plt.savefig("Compersion.pdf", format='pdf', dpi=300)
plt.tight_layout()
plt.show()
```



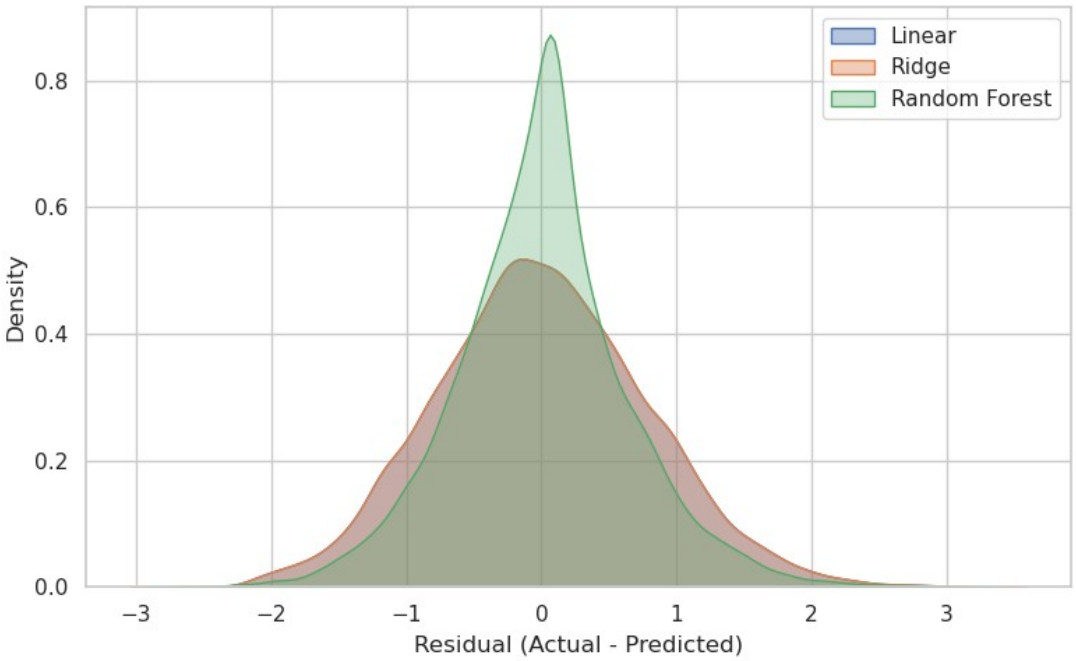

```
# assumes you already have: y_test, y_pred  (Linear), y_pred_ridge,
y_pred_rf
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(7,6))
plt.scatter(y_test, y_pred,       s=12, alpha=0.35, label="Linear")
plt.scatter(y_test, y_pred_ridge, s=12, alpha=0.35, label="Ridge")
plt.scatter(y_test, y_pred_rf,    s=12, alpha=0.25, label="Random
Forest")
mn, mx = y_test.min(), y_test.max()
plt.plot([mn, mx], [mn, mx], "r--", lw=2)
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Predicted Energy Consumption (kWh)")
#plt.title("Actual vs Predicted, All Models")
plt.legend()
plt.tight_layout()
plt.savefig("actual_pred_all_models.pdf", bbox_inches="tight")
plt.show()
```




```
res_lin   = y_test - y_pred
res_ridge = y_test - y_pred_ridge
res_rf    = y_test - y_pred_rf

plt.figure(figsize=(8,5))
plt.axhline(0, color="red", linestyle="--", linewidth=1.5)
plt.scatter(y_test, res_lin,   s=10, alpha=0.35, label="Linear")
plt.scatter(y_test, res_ridge, s=10, alpha=0.35, label="Ridge")
plt.scatter(y_test, res_rf,    s=10, alpha=0.25, label="Random
Forest")
plt.xlabel("Actual Energy Consumption (kWh)")
plt.ylabel("Residual (Actual - Predicted)")
#plt.title("Prediction Error vs Actual, All Models")
plt.legend()
plt.tight_layout()
plt.savefig("residuals_vs_actual_all_models.pdf", bbox_inches="tight")
plt.show()
```




```
import seaborn as sns

plt.figure(figsize=(8,5))
sns.kdeplot(res_lin,   fill=True, alpha=0.4, label="Linear")
sns.kdeplot(res_ridge, fill=True, alpha=0.4, label="Ridge")
sns.kdeplot(res_rf,    fill=True, alpha=0.3, label="Random Forest")
#plt.title("Residual Distribution, All Models")
plt.xlabel("Residual (Actual - Predicted)")
plt.ylabel("Density")
plt.legend()
plt.tight_layout()
plt.savefig("residual_distribution_all_models.pdf",
bbox_inches="tight")
plt.show()
```


Linear
Ridge
Random Forest
Density
0.0
0.2
0.4
0.6
0.8
−3
−2
−1
0
1
2
3
Residual (Actual - Predicted)