Stage 1 ML Project

Thomas DANIEL / Théo CHICHERY / Jean MARCHEGAY / Julien DE VOS
DIA 3

November 15, 2024

Contents

1 Business Scope

Context :

The decentralized expansion of renewable energies is an important pillar on the way to the energy transition. Self-generation systems on industrial and commercial sites are a major field of application. Primarily, customers consume the electricity they generate themselves and purchase electricity if the quantities generated are insufficient to completely cover their energy load.

The resulting residual load (residual load = energy load - self generated energy) must in any case be provided by the energy supplier (aka. the grid). Note, that the residual load can also be negative if the amount of self generated energy is greater than the load. In this case, the excess energy is fed into the electricity grid.

Predicting energy requirements is necessary for several reasons:

For consumers:

- Managing consumption peaks: According to the French Ministry of Ecology, "consumption increases by around 2.4 GW per degree Celsius below zero" in winter, so that we can always meet demand.
- Optimizing costs to adjust consumption.

For producers:

- Fluctuations in production levels, particularly for renewable energies that depend on the climate.
- The risk of a sudden drop in photovoltaic or wind power production.
- Reduced production costs due to almost zero losses and no need to buy electricity on the market.

All these points make energy forecasting an interesting market.

2 Problem Formalisation and Methods

2.1 Algorithm Description

Linear Regression is a supervised machine learning algorithm that takes different variables and predicts the requested variable by optimizing the equation:

$$Y = B_0 + B_1X_1 + B_2X_2 + \cdots + B_nX_n$$

Where:
Y = Variable sought
X_n = Independent variables
B_n = Coefficients (degrees)
B_0 = Y-intercept

To predict y, the algorithm will minimize the difference between y predicted and y by updating the degrees associated with the different variables.

Knn regression is a supervised model. The algorithm will consider the variable's neighbors to predict its value. To do this, we need to choose the number of neighbors used, and the model will then calculate the distance between neighbors to make a prediction based on the neighbors.

2.2 Limitations

Linear Regression:

- Linearity: If the variables have no linear relationship with each other, the model will have poor accuracy, since it can only use a straight line to make its prediction.
- Dependency: If the variables have no impact on the desired variable.
- Choice of characteristics: If the variables used to predict y are poorly chosen, the prediction will be poorly influenced.

KNN :

- Hyperparameters: the choice of hyperparameters is very important, as the number of neighbors can have a strong impact on accuracy, whether through overfitting or underfitting.
- Calculation: Knn requires the values to be recalculated each time, and the time taken to do so increases greatly with the amount of data.
- Model: We can't have a knn model here, so it can't be reused quickly.Each time it needs to recalculate based on its neighbors.

3 Methodology

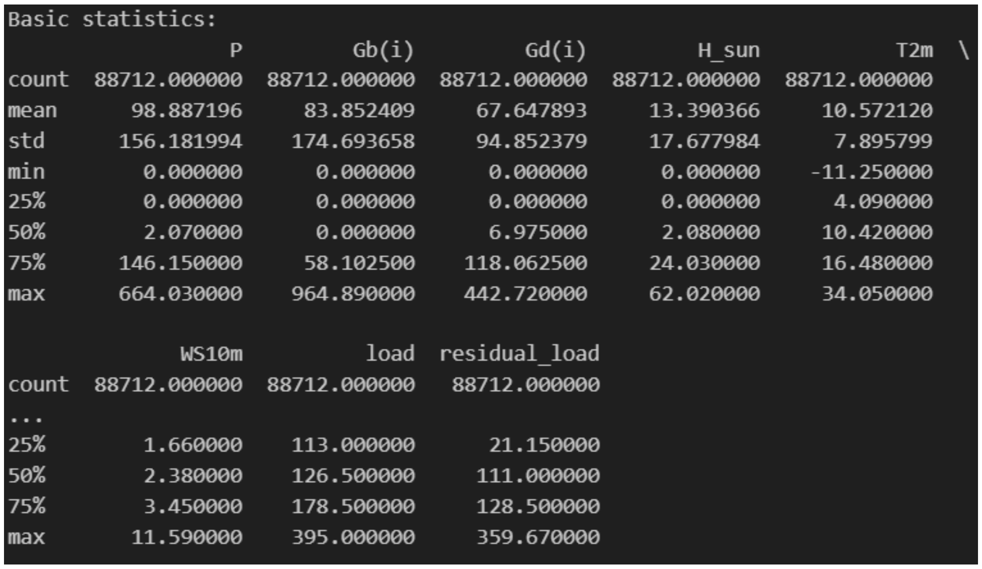
3.1 Data Description and Exploration

For this project, we have 3 csv at our disposal. The first presents the format that our submission csv for Kaggle must have to get the result of our model on the test data. We then have a test csv without the actual values to be predicted (P,load, residual load), which will be used to check the result of our model, and finally a train csv with all the columns including P,load and residual load, to train our models and test their efficiency. In this section, we'll focus on the train.csv file.

Our project is based on multivariate time series, however in this part we will not consider the time because we will use simple models (KNN, linear regression)

We start by putting the csv in a dataframe, the dataframe has 88712 rows and 10 columns. We first remove the id column and index the time as they are useless for us now.

To describe our data, we use the describe command which gives us much information about our data like the average, the maximum and others. We can see that in most columns the maximum is much higher than the average, which will be checked in the outlier's part.



```
Basic statistics:
                 P        Gb(i)        Gd(i)        H_sun          T2m  \
count  88712.000000  88712.000000  88712.000000  88712.000000  88712.000000
mean      98.887196     83.852409     67.647893     13.390366     10.572120
std      156.181994    174.693658     94.852379     17.677984      7.895799
min        0.000000      0.000000      0.000000      0.000000    -11.250000
25%        0.000000      0.000000      0.000000      0.000000      4.090000
50%        2.070000      0.000000      6.975000      2.080000     10.420000
75%      146.150000     58.102500    118.062500     24.030000     16.480000
max      664.030000    964.890000    442.720000     62.020000     34.050000

              WS10m          load  residual_load
count  88712.000000  88712.000000   88712.000000
...
25%        1.660000    113.000000      21.150000
50%        2.380000    126.500000     111.000000
75%        3.450000    178.500000     128.500000
max       11.590000    395.000000     359.670000
```

Figure 1: Variables statistics

We now look at the graphical distributions of each column using the hist function :

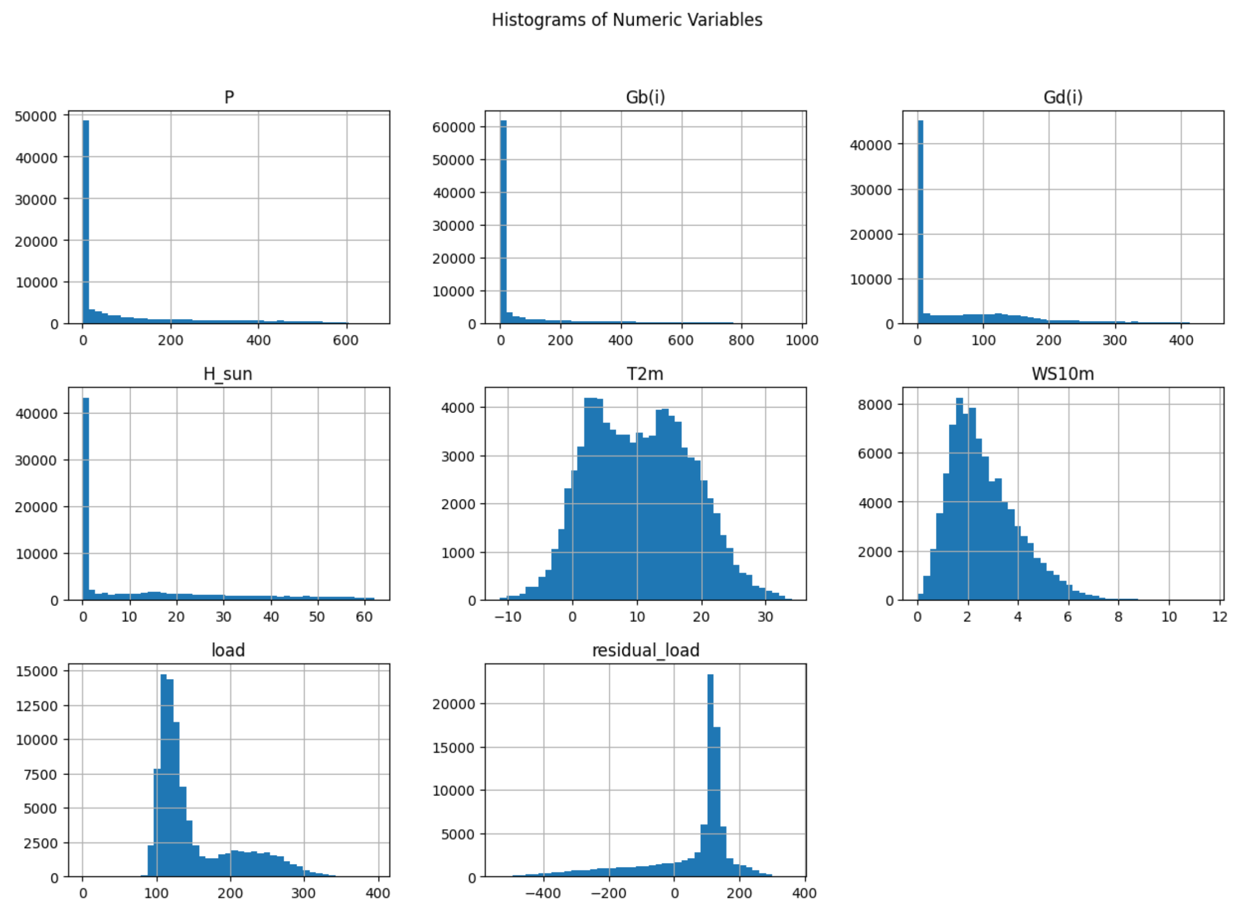


Figure 2: Variables histograms

We observe two types of distributions. The first follows a skewed Gaussian distribution with near symmetry, as is the case with T2M.

The second follows a power law distribution, where most of the data are concentrated around 0, and only a small portion of the data have higher values, as shown in graph P.

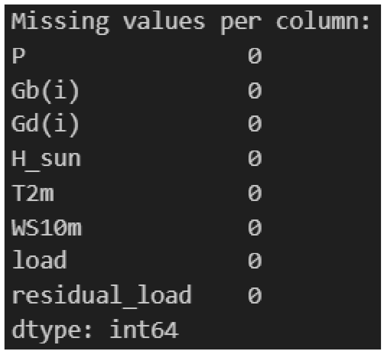
This can help us choose our models more efficiently

To finish this part, we can see that the submission.csv requests two columns time and residual load associated. It will therefore be necessary to take this into account when creating our submission csv

3.1.1 Missing Values

Missing data, or missing values, occurs when you don't have data stored for certain variables or participants. This is important to detect them as it can impact models' performance. Indeed, missing values can introduce bias and lead to inaccurate predictions if not properly handled.

Let's see if we have any missing value for each feature:



```
Missing values per column:
P                0
Gb(i)            0
Gd(i)            0
H_sun            0
T2m              0
WS10m            0
load             0
residual_load    0
dtype: int64
```

Figure 3: Missing values

There are no missing values in our dataset. Let's now visualize our data more concretely to see if we have outliers.

3.1.2 Outliers and Imbalanced Data

An outlier is a data point that significantly deviates from the other observations in a dataset. It often appears unusually high, low, or in some way different from the general pattern or trend. Outliers can result from measurement errors, data entry mistakes, or genuinely rare phenomena.
Outliers can distort statistical summaries (like mean, standard deviation) and lead to misleading conclusions. Removing or handling them ensures cleaner and more reliable data.

Outliers can negatively impact the performance of machine learning models. For example:

- Linear regression models may become skewed as they try to fit the extreme values.
- Distance-based models like k-Nearest Neighbors (k-NN) or clustering algorithms (e.g., k-Means) may assign undue importance to outliers.

To visualize how the data is distributed, we used a boxplot which allows us to ascertain the presence of potential outliers.

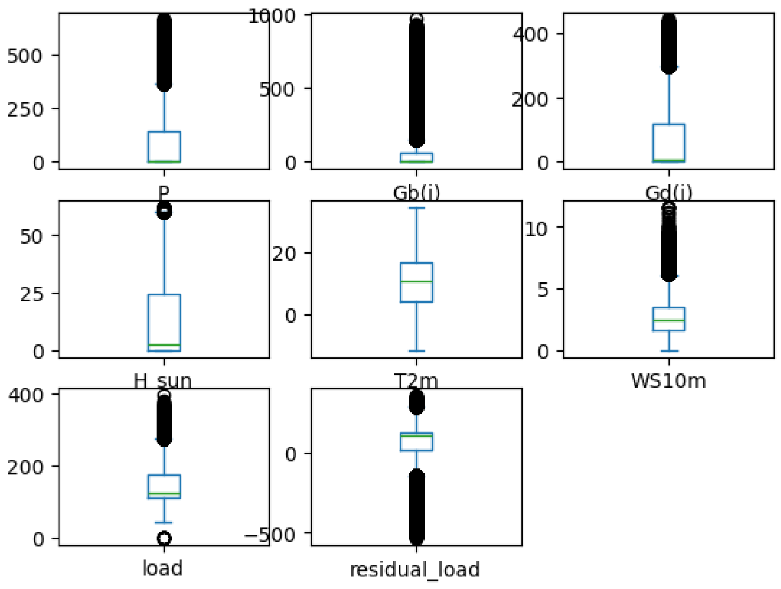


Figure 4: Boxplot for each variable

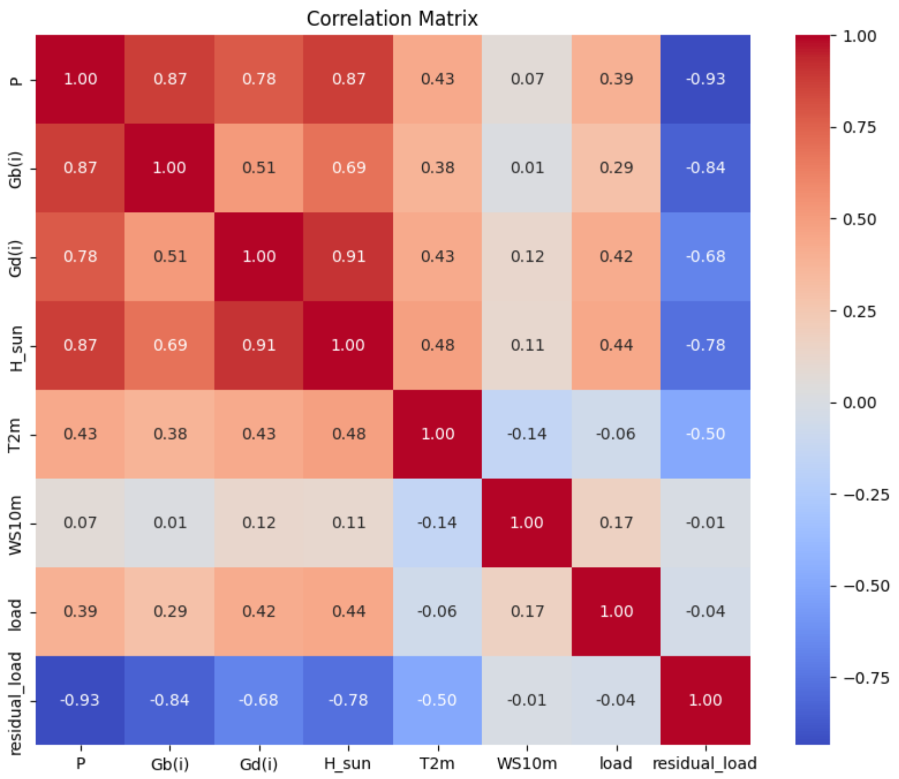
We see that all values except T2m seem to have a great amount of outlier values. However, given that our data is a time series (time dependent data) it is not surprising that our data is grouped around certain values. Gb(i), Gd(i), H sun, and P all depend on the sun (which corresponds to the time of the day) which means that we can observe values which appear to be outliers but can be easily explained by the fact there is no sun during night.

Similarly, wind speed is higher during night which can explain some of the outliers noticed on WS10m.

We can conclude from this analysis that we should not delete or modify outliers value since their presence is explained by the time series nature of our dataset.

3.1.3 Correlation analysis

Let's now see if we observe some correlation between variables by first displaying the correlation matrix of the dataset. A correlation matrix is a matrix that indicates the two-by-two relationship of the variables. It is interesting to reduce the dimensionality of the dataset: if two variables are very correlated, is it useless to keep both variables for the training and the prediction.



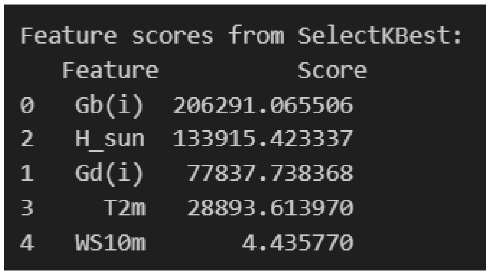
	P	Gb(i)	Gd(i)	H_sun	T2m	WS10m	load	residual_load
P	1.00	0.87	0.78	0.87	0.43	0.07	0.39	-0.93
Gb(i)	0.87	1.00	0.51	0.69	0.38	0.01	0.29	-0.84
Gd(i)	0.78	0.51	1.00	0.91	0.43	0.12	0.42	-0.68
H_sun	0.87	0.69	0.91	1.00	0.48	0.11	0.44	-0.78
T2m	0.43	0.38	0.43	0.48	1.00	-0.14	-0.06	-0.50
WS10m	0.07	0.01	0.12	0.11	-0.14	1.00	0.17	-0.01
load	0.39	0.29	0.42	0.44	-0.06	0.17	1.00	-0.04
residual_load	-0.93	-0.84	-0.68	-0.78	-0.50	-0.01	-0.04	1.00

Figure 5: Correlation matrix

We can observe for example that Gd(i) and Hsun are very correlated (0.91) so if we need to reduce dimensionality, we can delete one of these two features. For the other variables, as we don't know P, load and residual load for the prediction phase, there are no clear correlations between them that could allow us to delete one.

We also see that Gb(i), Gd(i), Hsun and T2m are the most correlated variables with residual load (since P is supposed to be unknown for the prediction), whereas WS10m is the least correlated (-0.01). Thus, this feature could be deleted if necessary to reduce the dataset dimension.

The scores given by a SelectKBest feature selection are in line with the results of the correlation matrix:



```
Feature scores from SelectKBest:
   Feature          Score
0   Gb(i)  206291.065506
2   H_sun  133915.423337
1   Gd(i)   77837.738368
3     T2m   28893.613970
4   WS10m       4.435770
```

Figure 6: Feature selection

To confirm our hypothesis that 2 or 3 features could be enough to train our models, we can use a PCA and check the amount of explained variance by each component.

A PCA (Principal Component Analysis) is a linear dimensionality reduction technique, it linearly transforms a data onto a new coordinate system such that the directions (principal components) capturing the largest variation in the data can be easily identified.

If we apply a PCA to project our data onto as many components as we have variables (5), here are the results:
The explained variance ratio by the PCA components is:

$$[0.5492, 0.2182, 0.1205, 0.0996, 0.0124]$$

We observe that if we need to reduce dimensionality, two or three features would be sufficient to explain around 90% of the data: it confirms our previous conclusion.

For now, since we don't have a lot of features in the dataset, we will keep the original number of variables.

3.2 Data Splitting for Train/Test

We split the data in a "classic" way: X for the features and Y for the target variable. We then split between train and test with 80% of train and 20% of test. For cross validation, 5-fold is used, which allows a good compromise between calculation time and diversity of data represented.

3.3 Algorithm Implementation and Hyperparameters

For this stage we have chosen the regression model seen in the ML course, we will explore some more models and hyperparameters choices in the next coming stages.

We chose two algorithms to test for our problem: Linear regression and KNN.

In both cases we will use sklearn to import the algorithms The sklearn linear regression does not have hyperparameters so we just use the sklearn import model.

For the KNN case we can change the number of neighbors used to calculate predictions. It is very important because a small number of neighbors will create an overfitting while a high number of neighbors will create underfitting.

We test with several neighbors ranging from 20 to 1000. We see that 100 neighbors gives the best precision, so it will be used for Kaggle prediction. Once these choices are made, we use our models on test data to validate our results on Kaggle

4 Results

4.1 Metrics

To measure the effectiveness of our models, we'll use two metrics R^2 and negative root mean squared error. The R-squared (R^2) formula is:

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

Where:
y_i = Actual values
$\hat{y}_i$ = Predicted values
$\bar{y}$ = Mean of actual values
n = Number of data points

The closer the R2 score is to 1, the better the model; for 0, the model is equivalent to the mean value; for less than 0, the model is worse than the mean.

The RMSE (Root Mean Squared Error) formula is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where:
y_i = Actual values
$\hat{y}_i$ = Predicted values
n = Number of data points

This gives us a number that represents the difference between our predicted values and the actual values. The lower this number is, the more accurate our model is, and it gives us an idea of how far our predicted values deviate from the actual ones.

These two metrics give a different view of our results and the accuracy of our models.
In addition the RMSE is the standard metric used in the kaggle competition associated with our data. In the next coming stages we will use this metric as a way to measure our models and to submit solutions on the kaggle leader board.

4.2 Overfitting

4.3 Linear Regression

In linear regression, overfitting occurs when the model becomes too complex, often by incorporating irrelevant features. This leads to a model that fits the training data very well but fails to generalize to new, unseen data. The model essentially learns the noise in the training data instead of the true underlying relationships.

4.4 K-Nearest Neighbors (KNN)

In KNN, overfitting happens when the model becomes overly sensitive to the training data, especially if the number of neighbors, k, is too small. A small k value means the model will base its predictions on very local data points, which can capture noise and anomalies rather than the broader trends in the data.

4.5 Mitigating Overfitting

4.5.1 Train and Test Sets

The principle of using train and test sets is fundamental to evaluating model performance. The **training set** is used to fit the model, allowing it to learn from the data. The **test set** is kept separate and used to assess the model's ability to generalize to new, unseen data. This separation ensures that the model is not just memorizing the training data but instead is learning patterns that can be applied to other datasets.

4.5.2 Cross-Validation

Cross-validation is a technique used to evaluate a model's performance on different subsets of the data. The dataset is divided into several smaller subsets, or "folds." The model is trained on some folds and tested on the remaining ones. This process is repeated multiple times to ensure that the model performs well across different portions of the data, helping to prevent overfitting.

4.5.3 Grid Search

Grid search is used to find the optimal hyperparameters for a model by testing a range of different values systematically. For example, in linear regression, grid search might test different regularization strengths, while in KNN, it could test different values for k. By finding the best combination of parameters, grid search helps to improve model performance and reduce the risk of overfitting.

4.6 Evaluation

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Linear Regression						
Neg_Root_Mean_Squared_Error	-67.81	-62.74	-67.02	-58.64	-68.35	**-64.71**
R^2	0.808	0.729	0.816	0.576	0.811	**0.748**
KNN (k = 20)						
Neg_Root_Mean_Squared_Error	-67.60	-58.99	-68.11	-57.63	-67.40	**-63.94**
R^2	0.809	0.760	0.810	0.591	0.816	**0.757**
KNN (k = 50)						
Neg_Root_Mean_Squared_Error	-66.82	-58.26	-66.96	-56.66	-66.56	**-63.25**
R^2	0.813	0.766	0.816	0.604	0.821	**0.764**
KNN (k = 100)						
Neg_Root_Mean_Squared_Error	-66.81	-58.29	-66.70	-56.45	-66.51	**-62.95**
R^2	0.813	0.766	0.818	0.607	0.821	**0.765**
KNN (k = 1000)						
Neg_Root_Mean_Squared_Error	-68.06	-59.25	-67.35	-57.00	-68.33	**-64.00**
R^2	0.806	0.758	0.814	0.599	0.811	**0.758**

Table 1: Evaluation Results for Regression Models (5 Folds) with Average Scores

The evaluation results for the regression models indicate that the KNN model with k=100 performed the best overall across all folds, with an average R^2 of 0.765 and a negative mean squared error (Neg RMSE) of -62.95.

This model consistently delivered strong results, particularly in comparison to other KNN configurations and linear regression. The KNN model with k=50 also performed well, with an average R^2 of 0.764, but slightly lower than the k=100 model.

Linear regression, while competitive with an average R^2 of 0.748, had a higher average Neg RMSE of -64.71, indicating a relatively weaker fit compared to the KNN models. Overall, the KNN model with k=100 is the best choice due to its superior balance between high predictive accuracy (R^2) and low error (Neg RMSE).

5 Discussion and Conclusion

While the models were able to make satisfactory predictions, there are still several areas for improvement:

- **Time Series Analysis:** The current approach does not fully leverage the time series nature of the data. Future models should consider the temporal dependencies more rigorously. We can explore techniques like autoregressive models, long short-term memory networks (LSTM), or time series cross-validation to improve predictions.

- **Separate Load and Power Prediction:** The residual load is the difference between the load and power production. A more complex approach could involve predicting both load and P separately, and then calculating residual load (i.e., residual load = load - P). This would allow the model to focus on predicting each component more accurately.

- **Feature Engineering:** We can improve feature engineering by incorporating other time-dependent features like rolling averages, seasonal components, and external weather data. Additionally, further dimensionality reduction techniques, like t-SNE or autoencoders, could be explored for more efficient feature extraction.

- **Ensemble Methods:** Ensemble learning methods such as Random Forests, XGBoost, and LightGBM could be tested to combine the strengths of individual models and improve predictive accuracy.