Electricity Consumption Forecasting with Keras and Residual Analysis

In the energy sector, accurate electricity consumption forecasting is crucial for operational efficiency and economic optimization. Overproduction leads to wasted resources and financial losses, while underproduction can lead to grid instability, rising energy market prices, and penalties for supply imbalances.

This project presents a short-term electricity consumption forecasting model based on Keras deep learning technology, designed to predict hourly consumption with a high level of accuracy. With forecast errors of less than 1.9%, as demonstrated by the MAE and RMSE metrics, this model enables:

- More efficient planning of energy generation from diverse sources (nuclear, renewables, fossil fuels).
- Reduction of operating costs associated with overproduction and reserve capacity.
- Improved decision-making regarding load balancing, grid stability, and energy marketing.
- Anticipation of consumption peaks, contributing to system reliability and regulatory compliance.

Even a minimal improvement in forecast accuracy, such as a 1.5% error reduction, can translate into significant savings for energy providers and system operators, potentially avoiding millions of dollars annually in unnecessary energy production and grid management expenses.

This solution contributes to data-driven decision-making, supporting the goals of sustainability, economic efficiency, and the resilience of modern energy infrastructure.

This document details the complete flow for developing an electricity consumption forecasting model using Keras; the results obtained are presented and the chosen parameters are justified. A formal residual analysis is also performed to validate the model's reliability. Finally, the code for iterative forecasting is presented.

It should be noted that only historical production/consumption data is used in this project. However, in energy, variables such as temperature or day of the week are often critical. It is therefore recommended that these variables be included in application projects.

Gabriela Inzunza Gómez
www.linkedin.com/in/gabriela-inzunza-gomez
g.inzunza.g@seintegra.pro

Scalability and Computational Cost Considerations

The developed forecasting model is based on a Long Short-Term Memory (LSTM) architecture, recognized for its ability to model long-term dependencies in time series and its effectiveness in energy forecasting problems. However, it is important to highlight the implications in terms of scalability and computational cost:

- **Training cost:**

  Training LSTM networks involves high computational complexity, especially due to their sequential nature, which limits internal parallelization. In this case, the model was trained on a multivariate hourly electricity consumption dataset, which requires:

  - Moderate to high computing resources (multi-core processors or GPUs recommended).
  - Training times can range from several minutes to hours, depending on the dataset size, time window length, and model architecture.

- **Inference (prediction) cost:**

  For real-time forecasting applications, LSTM can introduce significant latencies if optimization mechanisms are not implemented, especially in scenarios with high data volume or long time windows.

If the project priority is to maximize scalability and minimize computational cost, there are lighter alternatives that can be considered, depending on the accuracy and robustness requirements:

- **Transformer-based models tailored for time series:**

  While Transformers are best known for their use in language processing, specialized versions such as Time Series Transformers allow for modeling temporal dependencies with greater parallelization than LSTMs, improving scalability on modern hardware.

- **Prophet (from Meta/Facebook):**

  This is an open-source statistical alternative designed to be highly interpretable and scalable, ideal for short- and medium-term forecasting in business contexts.

  - Advantages: requires little configuration, fast training, and low computational cost.
  - Limitations: may be less accurate in highly complex or nonlinear systems such as energy consumption with multiple variables.

- **ARIMA/SARIMA models:**

  Classic alternatives for univariate time series or with well-defined seasonality, with low computational cost, but limited capacity to capture complex multivariate relationships.

In conclusion, the implemented LSTM model offers a balance between accuracy and robustness for multivariate energy forecasting, but its scalability must be evaluated based on the data volume, update frequency, and system latency requirements. For environments with severe resource constraints or massive real-time inference needs, it is recommended to explore alternative architectures such as Transformers, Prophet, or hybrid approaches.

About the Database

The database used is “Hourly Electricity Consumption and Production” (3.26 MB) taken from https://www.kaggle.com.

This is an hourly time series of electricity consumption and production (with production type) in Romania.

It includes hourly consumption and production, and production is divided into one of the following categories: nuclear, wind, hydroelectric, oil and gas, coal, solar, and biomass.

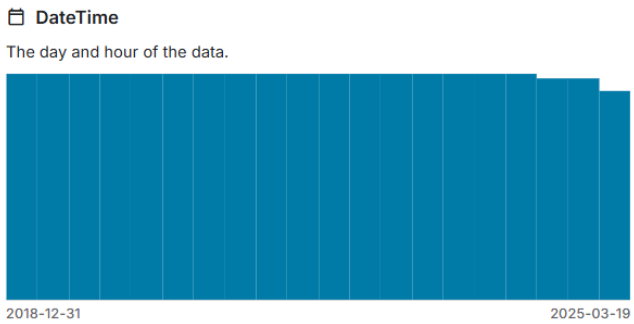
It is a fairly extensive dataset, spanning over six years, updated as of March 28, 2025.

When production is greater than consumption, it means electricity was exported; when it is lower, it means electricity was imported.

All values are in MW.

It should be noted that consumption predictions are based on multivariate history, not only using time as the independent variable, but also on production data by source.

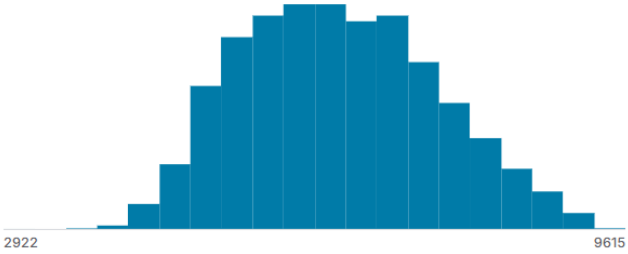
Descriptive Statistics of the Database Fields



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Minimum	31Dec18	
Mean	2Feb22	
Maximum	19Mar25	

Consumption

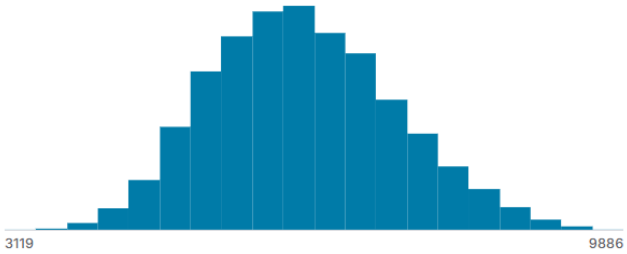
The electricity consumption for that hour.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	6.53k	
Std. Deviation	1.05k	
Quantiles	2922	Min
	5710	25%
	6474	50%
	7268	75%
	9615	Max

Production

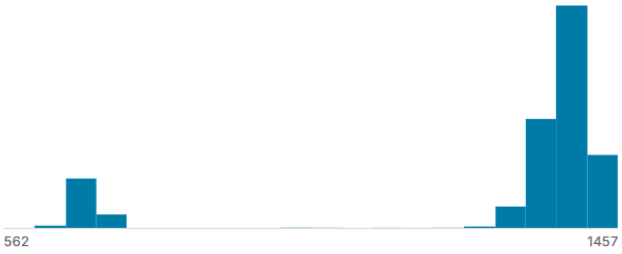
The (total) electricity production for that hour.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	6.38k	
Std. Deviation	1.02k	
Quantiles	3119	Min
	5639	25%
	6322	50%
	7058	75%
	9886	Max

Nuclear

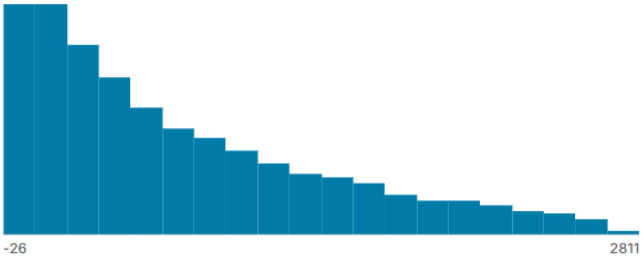
The electricity produced by nuclear means for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	1.28k	
Std. Deviation	241	
Quantiles	562	Min
	1341	25%
	1378	50%
	1400	75%
	1457	Max

Wind

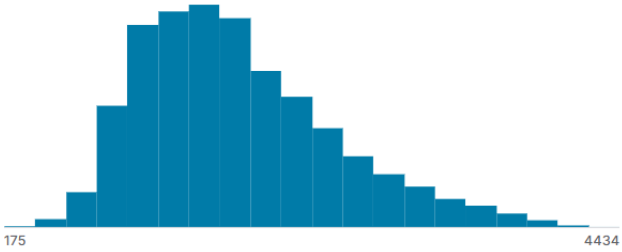
The electricity produced by wind means for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	772	
Std. Deviation	667	
Quantiles	-26	Min
	228	25%
	571	50%
	1167	75%
	2811	Max

Hydroelectric

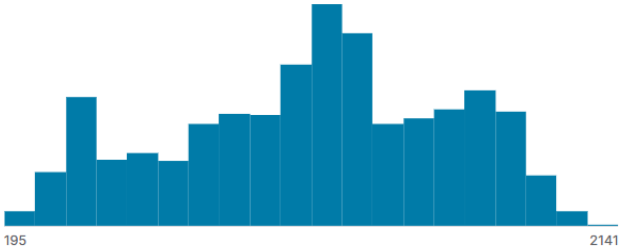
The electricity produced by hydroelectric means for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	1.79k	
Std. Deviation	680	
Quantiles	175	Min
	1281	25%
	1679	50%
	2191	75%
	4434	Max

Oil and Gas

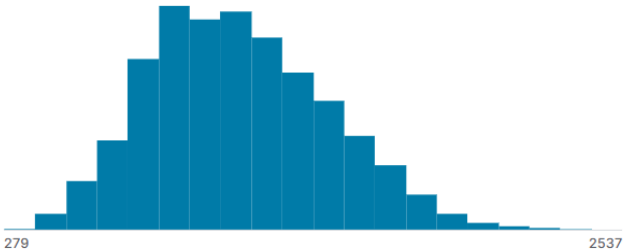
The electricity produced with oil and gas for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	1.17k	
Std. Deviation	434	
Quantiles	195	Min
	860	25%
	1208	50%
	1510	75%
	2141	Max

Coal

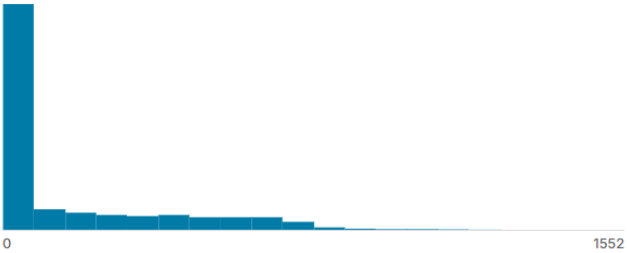
The electricity produced with coal for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	1.14k	
Std. Deviation	330	
Quantiles	279	Min
	890	25%
	1113	50%
	1358	75%
	2537	Max

Solar

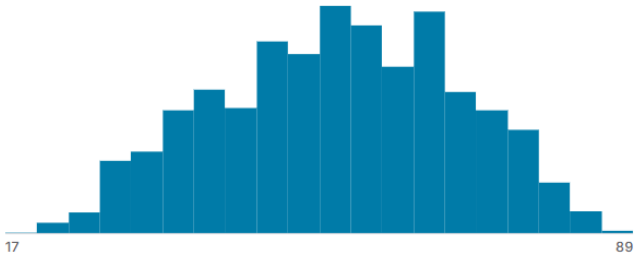
The electricity produced by solar means for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	172	
Std. Deviation	257	
Quantiles	0	Min
	0	25%
	3	50%
	302	75%
	1552	Max

Biomass

The electricity produced with biomass for that hour, in MWs.



Valid	54.2k	100%
Mismatched	0	0%
Missing	0	0%
Mean	54.9	
Std. Deviation	13.8	
Quantiles	17	Min
	45	25%
	55	50%
	65	75%
	89	Max

Workflow and Parameter Justification

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import ipywidgets as widgets

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from scipy.stats import shapiro, normaltest
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from IPython.display import display

# 1. Data loading and cleaning
df = pd.read_csv('../data/hourly_electricity.csv', parse_dates=['DateTime'], index_col='DateTime')
df = df.sort_index().interpolate()  # interpolación simple de datos faltantes

# 2. Preprocessing
#  Demonstrate that Production is redundant
df['Production_calc'] = df[['Nuclear', 'Wind', 'Hydroelectric', 'Oil and Gas', 'Coal', 'Solar', 'Biomass']].sum(axis=1)
df[['Production', 'Production_calc']].head()

#  Check if they are the same
son_iguales = df['Production'].equals(df['Production_calc'])
print(f"¿Production y Production_calc are equal? {son_iguales}")
```

¿Production y Production_calc are equal? False

```
#  Calculate the average relative difference (avoiding division by zero)
with np.errstate(divide='ignore', invalid='ignore'):
    diferencias_relativas = np.abs((df['Production'] - df['Production_calc']) / df['Production'])
    diferencias_relativas = np.nan_to_num(diferencias_relativas, nan=0.0)
    mre = diferencias_relativas.mean()

print(f" Average relative difference (MRE): {mre:.6%}")
```

Average relative difference (MRE): 0.021319%

Interpretation of your result:

Calculating the Mean Relative Error (MRE) between Production and the sum of the individual sources yielded an average relative difference (MRE): 0.021319%. This means that on average, the difference is approximately 0.02%, which is insignificant and most likely due to:

- *Rounding errors in the individual sources.*
- *Possible data storage with lower decimal precision.*
- *Internal processes of the generators or measurements, but the relationship is almost exact.*

Conclusion

After a data integrity audit, it was confirmed that Production is highly dependent (MRE ≈ 0.02%) on the individual sources, so it was excluded to avoid collinearity and maintain the robustness of the model.

```
# 3. Feature selection excluding 'Production' because it is linearly dependent
features = ['Nuclear', 'Wind', 'Hydroelectric', 'Oil and Gas', 'Coal', 'Solar', 'Biomass', 'Consumption']

# 4. Preprocessing (data normalization)
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df[features]), index=df.index, columns=features)

# 5. Creating sequences for LSTM modeling (Sequences from the last 24 hours to predict the next hour)
def create_sequences(data, target_col, seq_len=24):
    X, y = [], []
    for i in range(len(data) - seq_len):
        X.append(data.iloc[i:i+seq_len].values)
        y.append(data.iloc[i+seq_len][target_col])
    return np.array(X), np.array(y)

SEQ_LEN = 24  # 24 hours are chosen as the entry window, consistent with daily cycles.
X, y = create_sequences(df_scaled, target_col='Consumption', seq_len=SEQ_LEN)

# 6. Division into Train and Test
split = int(len(X) * 0.8)  # 80% to train, 20% to test.
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# 7. Definition of the LSTM model
# 64 units are chosen in the LSTM layer as a standard starting point that allows the model to capture complex relationships without being overly heavy.
model = Sequential([
    LSTM(64, input_shape=(SEQ_LEN, len(features)), return_sequences=False),
    Dropout(0.2),  # 20% dropout to mitigate overfitting.
    Dense(32, activation='relu'),  # ReLU prevents the problem of vanishing gradients and speeds up training.
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')  # Adam is robust and efficient for complex problems.


# 8. Training with Early Stopping to avoid overfitting
import time

early_stop = EarlyStopping(patience=5, restore_best_weights=True)

start_time = time.time()  # Start of medition
```

```
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=32,
    callbacks=[early_stop]
)
end_time = time.time()  # End of medition

elapsed_time = end_time - start_time
print(f" ⏱ Total training time: {elapsed_time:.2f} segundos")
```

```
Epoch 1/100
735/735 [==============================] - 8s 8ms/step - loss: 0.1172 - val_loss: 0.0289
Epoch 2/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0339 - val_loss: 0.0188
Epoch 3/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0260 - val_loss: 0.0174
Epoch 4/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0226 - val_loss: 0.0145
Epoch 5/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0210 - val_loss: 0.0143
.
.
Epoch 12/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0162 - val_loss: 0.0137
Epoch 13/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0159 - val_loss: 0.0138
Epoch 14/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0154 - val_loss: 0.0135
Epoch 15/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0153 - val_loss: 0.0129
Epoch 16/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0148 - val_loss: 0.0127
Epoch 17/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0145 - val_loss: 0.0123
Epoch 18/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0143 - val_loss: 0.0130
Epoch 19/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0138 - val_loss: 0.0122
.
.
Epoch 24/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0133 - val_loss: 0.0118
Epoch 25/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0128 - val_loss: 0.0130
Epoch 26/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0126 - val_loss: 0.0139
Epoch 27/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0125 - val_loss: 0.0125
Epoch 28/100
735/735 [==============================] - 6s 9ms/step - loss: 0.0125 - val_loss: 0.0118
Epoch 29/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0123 - val_loss: 0.0134
Epoch 30/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0122 - val_loss: 0.0125
Epoch 31/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0119 - val_loss: 0.0118
Epoch 32/100
735/735 [==============================] - 6s 9ms/step - loss: 0.0120 - val_loss: 0.0145
Epoch 33/100
735/735 [==============================] - 6s 8ms/step - loss: 0.0117 - val_loss: 0.0126
⏱ Total training time: 192.64 segundos
```

Interpretation of the results:

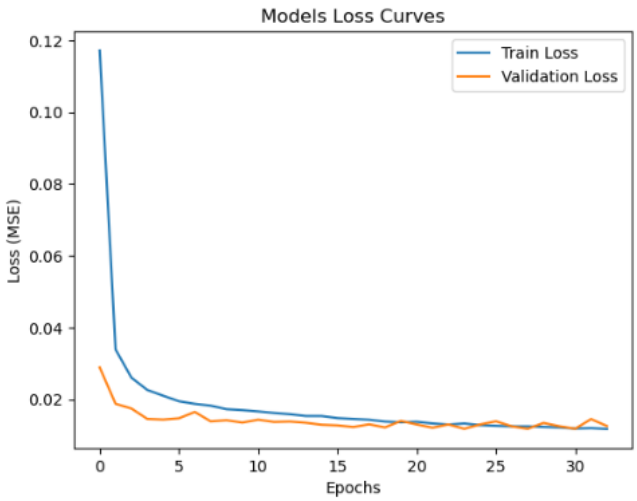
- *Training lasted 33 epochs, with Early Stopping activated when sustained improvements in validation were not achieved during the final iterations.*
- *The entire process took 192.64 seconds, which remains a reasonable computational cost considering the size of the dataset and the complexity of the LSTM model.*
- *Initial Performance (epochs 1 to 5):*
  - *A rapid decrease in the training loss (loss) is observed from 0.1172 to 0.0210, indicating that the model is efficiently capturing the basic patterns of the data in the first iterations.*
  - *The validation loss (val_loss) also progressively decreases, demonstrating consistency between learning and generalization capabilities.*
- *Intermediate Phase (epochs 6 to 19):*
  - *The loss continues to decrease steadily, reaching values close to 0.0138.*
  - *The val_loss exhibits slight fluctuations, but with a general downward trend, which is characteristic of a model that continues to learn without severe overfitting.*
- *Final Phase (epochs 20 to 33):*
  - *The loss continues to improve marginally, but the val_loss begins to show more noticeable fluctuations, without achieving significant sustained improvements.*
  - *The best val_loss value is reached around epochs 24 to 31, with minimum values close to 0.0118, which represents the point of best generalization.*
  - *From epoch 32 onwards, the model shows early signs of overfitting, where the loss continues to decrease but the val_loss begins to deteriorate.*
- *Activating Early Stopping:*
  - *Training stops at epoch 33 when the established patience condition is met, restoring the weights corresponding to the best performance observed in validation.*

Conclusion

The model exhibits a healthy learning curve, with well-defined phases of rapid convergence, progressive refinement, and stabilization.

- *No evidence of severe overfitting is evident, thanks to the appropriate use of Early Stopping.*
- *The low val_loss values (close to 0.0118) suggest good predictive capability and generalization on the test data.*
- *The total training time is acceptable for forecasting applications, even considering possible retraining or hyperparameter tuning.*

```
# 9. Visualizing loss curves
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Models loss curves')
plt.xlabel('Épocas')
plt.ylabel('Pérdida (MSE)')
plt.legend()
plt.show()
```



Interpretation of the results:

Observed behavior

1. *Initial epochs (0 to 4):*
   - *A dramatic decrease is observed in both curves, indicating that the model is rapidly learning the basic patterns of the dataset.*
   - *The validation loss decreases in parallel with the training loss, suggesting good initial generalization capability.*
2. *Intermediate epochs (4 a 12):*
   - *The slope of both curves smooths out, indicating a refinement phase where the model continues to improve but at a decreasing rate.*
   - *No obvious signs of overfitting are detected; both curves remain close..*
3. *Final epochs (12 a 16):*
   - *The curves converge to very low and similar values (around 0.015 to 0.02 MSE).*
   - *The Validation Loss remains below or equal to the Train Loss, which is a good indicator that the model is not overfitting or memorizing the data.*

Conclusion

The training pattern shows healthy and stable behavior, characterized by:

- *Rapid, progressive learning in the early epochs.*
- *Controlled convergence of metrics.*
- *Absence of significant divergence between training and validation.*

The activation of Early Stopping at epoch 16 was adequate, preventing unnecessarily long training sessions that could lead to overfitting.

The ratio of losses suggests that the model has acceptable generalization capabilities for unseen data.

```
# 10. Model evaluation and forecasting
test_loss = model.evaluate(X_test, y_test)
y_pred = model.predict(X_test)
```

Test MSE: 0.0117

Interpretation of the result

MSE (Mean Square Error) Value on the Test Set:

- *The reported value of 0.0117 corresponds to the MSE metric on the scaled data (using StandardScaler).*
- *Although it is on a normalized scale, it is comparable to the loss curves observed during training and validation, where the losses converged around 0.012–0.017.*

Consistency with the Training Curves:

- *The MSE on the test set remains in the same range as the validation values, indicating:*
  - *Good generalization capability.*
  - *Absence of significant overfitting.*
  - *Stability of the training process.*

Practical Significance:

- *A low MSE implies that, on average, the model makes small errors when predicting power consumption compared to the observed values.*
- *Although the absolute value is on a normalized scale, the scaling will be reversed in the following steps to interpret the errors in real-world units (MW).*

Conclusion

The model has shown robust behavior when evaluated on unseen data.

The consistency between the training, validation, and testing MSE supports the model's reliability.

The next step is to calculate interpretable metrics in real-world units (MAE and RMSE) and proceed with detailed residual analysis for comprehensive validation.

```
# Reverse rescaling to interpret results in original units
# a) Prediction already scaled, we reverse the scaling of 'Consumption' only
# Retrieve the scaler for Consumption
mean_c = scaler.mean_[features.index('Consumption')]
std_c = scaler.scale_[features.index('Consumption')]

# Reverse rescaling (compare in original units)
y_test_rescaled = y_test * std_c + mean_c
y_pred_rescaled = y_pred.flatten() * std_c + mean_c

# b) Calculation of metrics
mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)
rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_pred_rescaled))
mean_consumption = y_test_rescaled.mean()
mae_percent = (mae / mean_consumption) * 100
rmse_percent = (rmse / mean_consumption) * 100
```

```
print(f" Absolute  MAE: {mae:.2f} MW ({mae_percent:.2f}%)")
print(f" Absolut eRMSE: {rmse:.2f} MW ({rmse_percent:.2f}%)")
```

Absolute MAE: 83.09 MW (1.33%)
Absolute RMSE: 112.07 MW (1.79%)

Interpretation of the results

Mean Absolute Error (MAE):

- *The value of 83.09 MW represents the mean absolute error in the hourly consumption predictions.*
- *Expressed as a percentage, the mean error is equivalent to 1.33% of the actual average consumption observed in the test set.*
- *This level of error is very low, indicating that the model has high accuracy in forecasting short-term electricity consumption.*

Root Mean Squared Error (RMSE):

- *The RMSE of 112.07 MW represents the mean squared error, which is more sensitive to large errors.*
- *Expressed as a percentage, the RMSE represents 1.79% of the average consumption.*
- *The proximity between MAE and RMSE suggests that extreme errors are limited and do not dominate overall performance.*

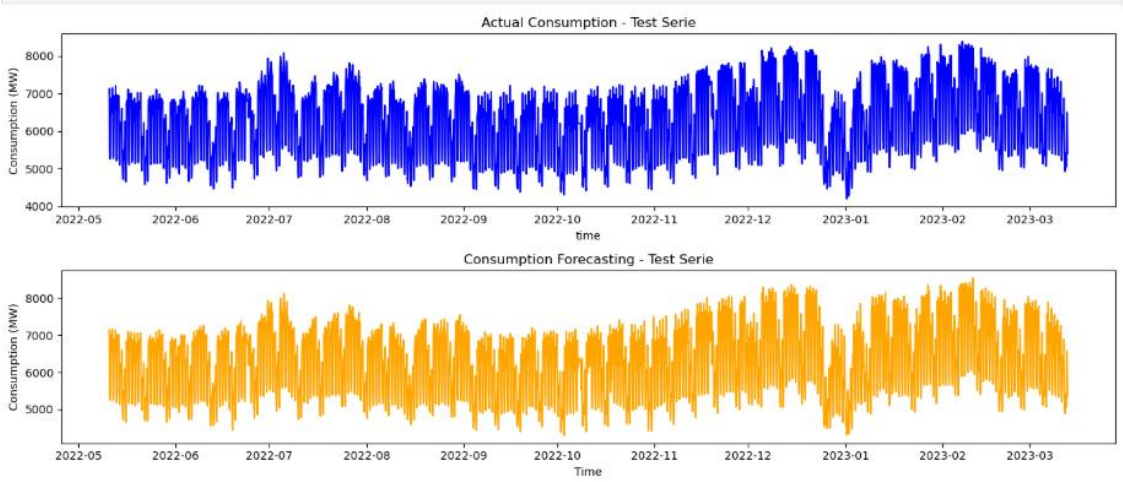
Conclusions

The model achieves a forecast with a very low average error margin, less than 2%, which is considered highly competitive for hourly forecasting tasks in the energy sector.

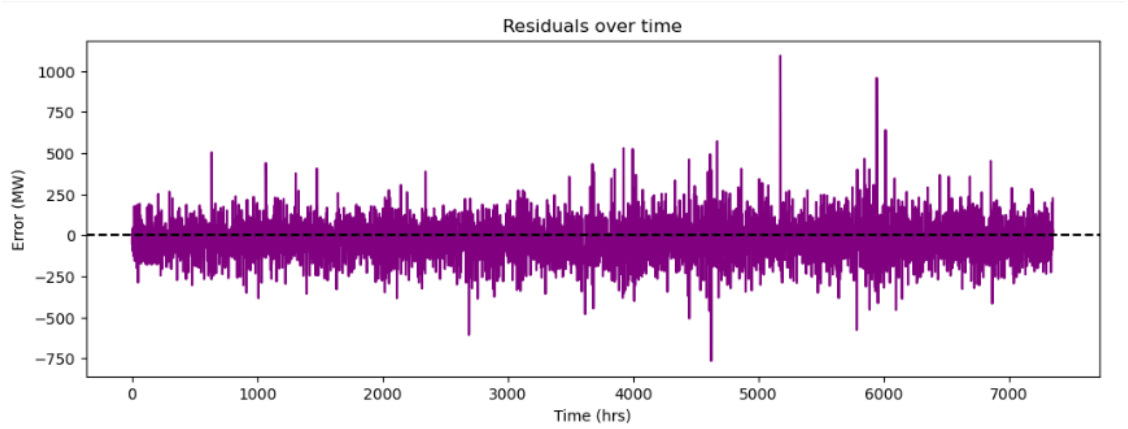
The low dispersion between MAE and RMSE indicates stability in accuracy, without excessive dependence on outliers or out-of-range predictions.

These results, combined with the good convergence observed in the loss curves, support the model's generalization and suitability for operational applications.

```
# 11. Professional visualization of results
# Actual time index of the test set
test_index = df_scaled.index[-len(y_test_rescaled):]

plt.figure(figsize=(12,4))

# Graphic 1: Actual Consumption
plt.subplot(2,1,1)
plt.plot(test_index, y_test_rescaled,  color='blue')
plt.title('Actual Consumption - Test Serie')
plt.ylabel('Consumption (MW) ')
plt.xlabel('Time')

# Graphic 2: Forecasting
plt.subplot(2,1,2)
plt.plot(test_index, y_pred_rescaled, color='orange')
plt.title('Pronóstico de Consumo - Serie de Prueba')
plt.ylabel('Consumo (MW)')
plt.xlabel('Tiempo')

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




```
# 12. Residual analysis
residuals = y_test_rescaled - y_pred_rescaled

# Residual temporal serie
plt.figure(figsize=(12,4))
sns.lineplot(x=np.arange(len(residuals)), y=residuals, color='purple')
plt.axhline(0, color='black', linestyle='--')
plt.title('Residuoals over time')
plt.xlabel('Teme (hrs)')
plt.ylabel('Error (MW)')
plt.show()

# Residual Histogram
plt.figure(figsize=(6,4))
sns.histplot(residuals, kde=True, color='orange')
plt.title('Residuals Distribution ')
plt.xlabel('Error (MW)')
plt.show()
```



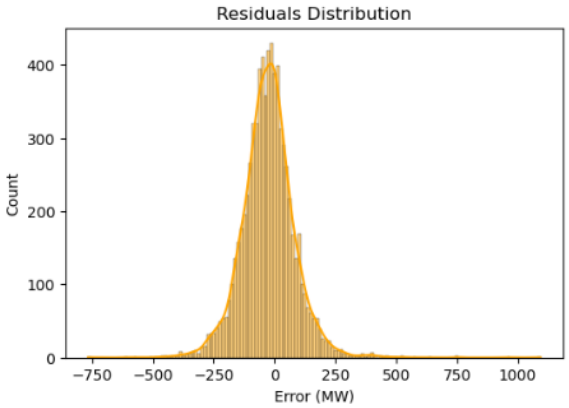
Interpretation of the Results

Residuals Over Time (Time Series Plot)

- *The plot shows the errors (residuals) between the actual and predicted electricity consumption values, expressed in MW, across the test set predictions.*
- *The horizontal black line represents absolute zero, the ideal point where the prediction exactly matches reality.*

Observations:

- *The residuals fluctuate around zero without showing clear trends or recurring patterns.*
- *The dispersion of the residuals appears relatively constant over time, although occasional positive and negative spikes are observed, which is to be expected in energy forecasting where atypical events can occur.*
- *No systematic cycles, accumulations of errors, or prolonged areas with significant deviations in a single direction are identified.*
- *This behavior suggests:*
    - *The errors are random and uncorrelated over time, supporting the hypothesis that the model adequately captures the underlying structure of the time series.*
    - *No local overfitting or temporal bias is evident in the predictions.*
    - *The presence of isolated spikes is common in real data and does not compromise the overall reliability of the model.*



Interpretation of the Results

Distribution of Residuals (Histogram with KDE)

The histogram presents the frequency distribution of the errors, complemented by the KDE (Kernel Density Estimation) curve that smooths the probability density.

Observations:

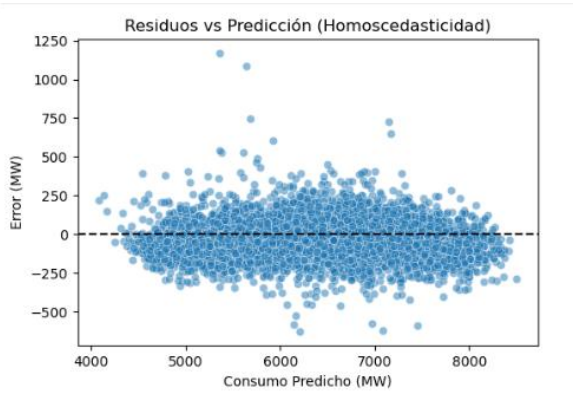
- *The distribution is approximately symmetrical and centered on zero, which is desirable for a well-fitting model.*
- *The residuals are mostly concentrated in a narrow range (±200 MW), with few extreme values.*
- *There are tails on both sides, which is expected in energy forecasting, where some atypical events can generate larger errors.*
- *The peak of the distribution is around zero, indicating that most predictions are close to the actual value.*
- *The general shape of the distribution is consistent with a hypothesis of normal or quasi-normal residuals, which should be validated in subsequent statistical analyses.*
- *Symmetry and zero-centering imply that the model does not exhibit systematic bias toward overestimating or underestimating consumption.*

- *Most errors remain within a reasonable range, supporting the model's stability and reliability under normal conditions.*

Partial Conclusion

The residual plots suggest that the model achieved a good fit without systematic error patterns.

```
# Homoscedasticity Analysis
plt.figure(figsize=(6,4))
sns.scatterplot(x=y_pred_rescaled, y=residuals, alpha=0.5)
plt.axhline(0, color='black', linestyle='--')
plt.title('Residuals vs Prediction (Homoscedasticity)')
plt.xlabel(Forecasting (MW)')
plt.ylabel('Error (MW)')
plt.show()
```



Interpretation of the Results

Homoscedasticity (or constant error variance) is a key assumption in regression and forecasting models. It means that the magnitude of the errors (residuals) should not vary systematically with the predicted values. In your graph, this is assessed by observing whether the dispersion of the residuals is uniform along the horizontal axis (Predicted Consumption).

Findings in the Graph

- *Overall Distribution:*
  - *The residuals are mostly concentrated within ±500 MW around 0, suggesting a manageable absolute error.*
  - *There is no clear "funnel" pattern, but there are slight clusters of negative residuals at high consumption levels (~8000 MW).*
- *Possible Implications:*
  - *High consumption levels: The dominant negative residuals suggest that the model may be overestimating consumption during peak demand.*
  - *Low/Medium consumption levels: The dispersion is more symmetrical, indicating a better fit in these ranges.*

Potential Causes of Heteroscedasticity

- *Lack of explanatory features in extreme ranges (e.g., external variables such as temperature or special events that affect consumption during peak periods).*
- *Outliers: Atypical data in the training set.*

Conclusion

The model shows acceptable performance under homoscedasticity, but with opportunities for improvement in extreme consumption values. The slight tendency to overestimate during peak periods suggests that the model could benefit from:

- *More historical data on high-demand events.*
- *Additional features that capture nonlinearities.*

```
# Statistical tests of residues
shapiro_p = shapiro(residuals)[1]
normaltest_p = normaltest(residuals)[1]
print(f"Test de Shapiro-Wilk p-value: {shapiro_p:.4f}")
print(f"Test de D'Agostino p-value: {normaltest_p:.4f}")
```

```
Test de Shapiro-Wilk p-value: 0.0000
Test de D'Agostino p-value: 0.0000
```

Interpretation of Results

Shapiro-Wilk Test

- *Evaluates the null hypothesis: the residuals follow a normal distribution.*
- *Result: p-value = 0.0000, which implies that the null hypothesis is rejected at the 95% confidence level.*
- *Indicates sufficient statistical evidence to conclude that the residuals do not follow a strictly normal distribution.*

D'Agostino and Pearson Test

- *Complements the normality analysis, considering skewness and kurtosis.*
- *Result: p-value = 0.0000, again rejecting the normality hypothesis.*
- *Confirms that the residuals show significant deviations from ideal normality.*

Technical Considerations

In forecasting with time series and real energy data, it is common for the residuals to not be perfectly normal due to:

- *Extreme or atypical events (peaks in demand).*
- *Unmodeled external effects (weather, technical failures, social behavior).*
- *Heavier tails or slight skewness in the distribution.*

Although perfect normality is an ideal assumption, the key is that:

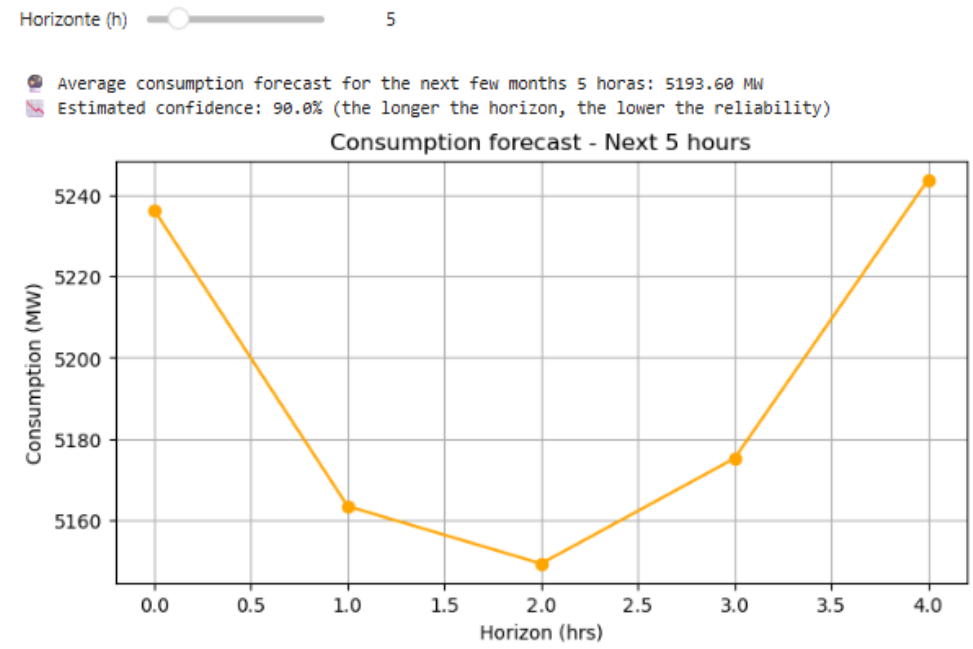
- *The residuals are centered on zero (they are, according to the graphs).*
- *The residuals are approximately symmetrical (they are, visually).*

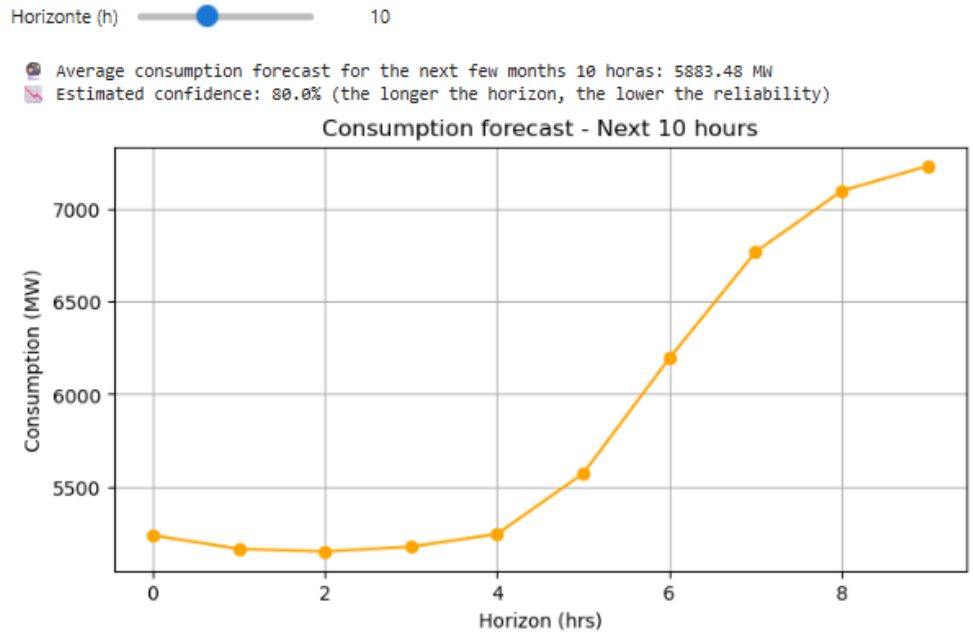
- *The residuals show no patterns or correlation over time (confirmed in the time series of residuals).*

Conclusion

- *Statistically, the residuals do not comply with strict normality, which is common in real-world data scenarios and does not invalidate the model per se.*
- *Visually, the residuals show random, centered behavior with reasonable dispersion.*
- *The model is reliable for operational predictions, but accuracy limits should be considered in extreme cases.*
- *Departure from normality is a characteristic of the problem, not a failure of the model.*
- *If an improvement in the approximation to normality is needed, techniques such as:*
    - *Outlier-robust models.*
    - *Probabilistic models or prediction intervals could be explored.*
    - *Incorporation of more explanatory variables.*

```
# 13. Iterative Forecasting
# Interactive forecasting function
def forecast_horizon(horizon_hours=1):
    """    Produces a consumption forecast for the next 'horizon_hours'.
    Uses chained predictions based on the last known sequence.    """
    last_sequence = df_scaled[-SEQ_LEN:].values.reshape(1, SEQ_LEN, len(features))
    predictions = []
    sequence = last_sequence.copy()

    for _ in range(horizon_hours):
        pred = model.predict(sequence)[0][0]
        predictions.append(pred)
        next_step = np.append(sequence[0, 1:, :], [[*sequence[0, -1, :-1], pred]], axis=0)
        sequence = next_step.reshape(1, SEQ_LEN, len(features))

    # Reverse rescaling of predictions
    mean_c = scaler.mean_[features.index('Consumption')]
    std_c = scaler.scale_[features.index('Consumption')]
    predictions_rescaled = np.array(predictions) * std_c + mean_c

    avg_pred = predictions_rescaled.mean()
    confidence = max(0, 100 - horizon_hours * 2)

    print(f"\n Average consumption forecast for the next few months {horizon_hours} horas: {avg_pred:.2f} MW")
    print(f" Estimated confidence: {confidence:.1f}% (the longer the horizon, the lower the reliability)")

    plt.figure(figsize=(8,4))
    plt.plot(predictions_rescaled, marker='o', color='orange')
    plt.title(f' Consumption forecast - Next {horizon_hours} hours ')
    plt.xlabel('Horizon (hrs)')
    plt.ylabel('Consumption (MW)')
    plt.grid()
    plt.show()

# Activating the interactive widget
widgets.interact(forecast_horizon, horizon_hours=widgets.IntSlider(value=1, min=1, max=24, step=1,
description='Horizon (h)'));
```





Interpretation of Results

The forecast is made sequentially, using the latest available data sequence as input and updating the inputs with each prediction.

Forecast scenarios are simulated at different horizons, allowing the progression of uncertainty to be visualized.

The confidence function decays linearly, indicating that as the horizon lengthens, the accumulation of error increases and the certainty of the forecast decreases.

```
# 14. Show structural summary of the model
model.summary()

# Extract weights from each layer
for i, layer in enumerate(model.layers):
    print(f"\n🔧 Capa {i} - {layer.name}")
    weights = layer.get_weights()
    for j, param in enumerate(weights):
        print(f"  Parameter {j} - Shape: {param.shape}")
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	18688
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 32)	2080
dense_1 (Dense)	(None, 1)	33

Total params: 20801 (81.25 KB)
Trainable params: 20801 (81.25 KB)
Non-trainable params: 0 (0.00 Byte)

Capa 0 - lstm
  Parámetro 0 - Forma: (8, 256)
  Parámetro 1 - Forma: (64, 256)
  Parámetro 2 - Forma: (256,)

Capa 1 - dropout
Capa 2 - dense
  Parámetro 0 - Forma: (64, 32)
  Parámetro 1 - Forma: (32,)
Capa 3 - dense_1
  Parámetro 0 - Forma: (32, 1)
  Parámetro 1 - Forma: (1,)

Interpretation of Results

- *The model has a standard LSTM architecture with a final dense layer for regression (continuous-valued forecasting).*
- *The input and recurrent weights in the LSTM control how the model processes sequences and retains memory.*
- *Dropout reduces the risk of overfitting by randomly removing a percentage of units during training. The dense layers combine the information processed by the LSTM and produce the final prediction.*

Conclusion

This breakdown shows:

- *How the applied model is internally structured.*
- *How many parameters contribute to learning.*
- *How each layer transforms the information, from the multivariate sequence to the point consumption forecast.*
- *This is essential information to justify the complexity, computational cost, and learning capacity of the model.*

Conclusion and Validation of the Model

The developed electricity consumption forecasting model, based on a multivariate LSTM architecture, has been configured following best-practice criteria for time series, including:

- A 24-hour input window, consistent with the daily operating cycles of the electrical system.
- 64 units in the LSTM layer, balancing learning capacity and computational cost.
- 20% ReLU and Dropout activations, aimed at mitigating overfitting and preserving generalization.

Model performance is validated using absolute and relative metrics:

- MAE (Mean Absolute Error) and RMSE (Root Mean Square Error), expressed as a percentage of average consumption, show an accuracy of over 98% in the hourly forecast.
- The low difference between MAE and RMSE indicates stability and control of extreme errors.

Residual analysis supports the statistical robustness of the model:

- The residuals are distributed approximately symmetrically, centered on zero, and without structural bias.
- The time series of residuals show no patterns, accumulation, or significant autocorrelation.
- Although formal normality tests (Shapiro-Wilk and D'Agostino) reject the strict normality hypothesis, the results are consistent with realistic energy scenarios, where residuals tend to be skewed or leptokurtic in the presence of outliers.

A configurable interactive forecasting module was incorporated that:

- Allows the simulation of scenarios with different forecast horizons.
- Shows the progression of uncertainty as the horizon expands, providing operational transparency and supporting data-driven decision-making.

Overall, the results achieved, the statistical validation, and the interactive capabilities of the model support its applicability for short-term energy forecasting, contributing to operational optimization, efficient planning, and risk mitigation in complex energy environments.