

Data Science Challenge: Predicting Energy Prices

[Updates & Additional Resources: https://btw2025.gi.de/data-science-challenge](https://btw2025.gi.de/data-science-challenge)

Submission Deadline: February 17 2025, 10:00 UTC (12:00 GMT).

Submitted by:

<your names here>

<your affiliation here>

Introduction

Day-ahead prices are the prices set for electricity delivery on the next day, determined through a daily auction process based on anticipated demand and supply. These prices provide a forecast that helps producers and consumers plan their generation and consumption, reflecting expected market conditions. Predicting these prices is crucial for energy companies and traders to optimize their operations, manage risks, and improve market efficiency.

As the goal of the upcoming BTW 2025 Data Science Challenge, you will build a model to predict hourly day-ahead energy prices for Germany on **February 18, 2025** as reported by the [SMARD.de information platform](https://smard.de) (see Figure 1 below for an example showing data for a different date in the past).

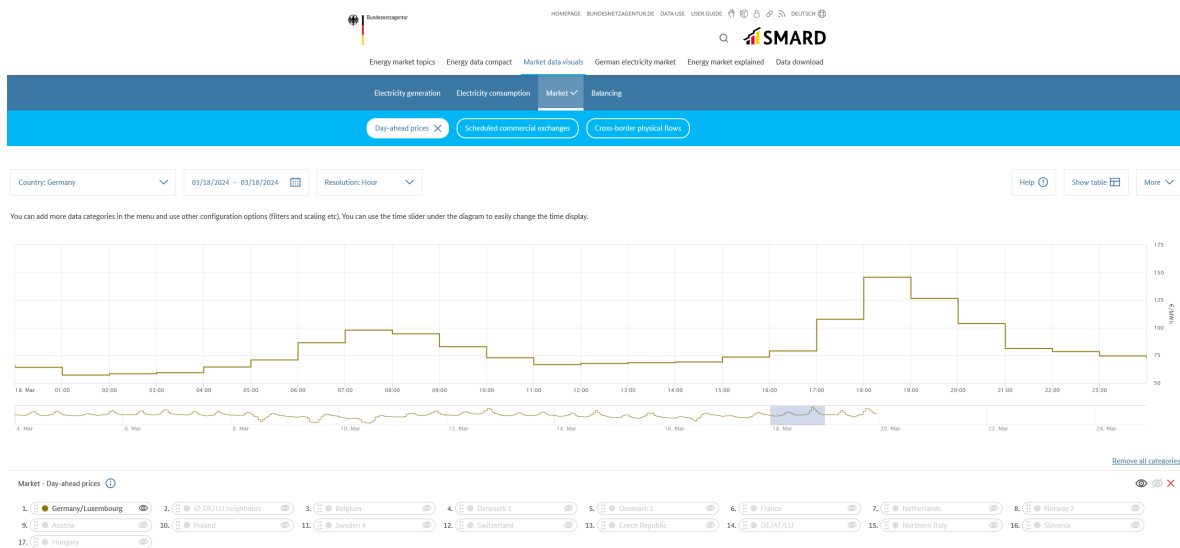


Figure 1: Day-Ahead Prices for March 18, 2024.

Submission Structure & Task Overview

The task of creating a submission for the BTW 2025 Data Science Challenge can be partitioned into 4 phases as described below.

Phase 1: Gathering Domain Knowledge & Data Sources

For building a model that predicts energy prices, you are required to gather not only historical data on energy prices, but also gain a strong understanding of the energy sector's dynamics. This entails collecting data for recent years on energy prices as well as demand and production capacities, weather conditions, supply metrics, and any significant events impacting energy prices. Additionally, acquiring domain knowledge is crucial for understanding the factors that influence energy markets.

You can think of your submission as a story about data discovery, where the background information sets the stage for understanding the significance and narrative of your analytical findings. It should therefore include important facts on the domain that you are exploring, for example:

- Research the energy market's structure, including how prices are set and the role of different energy sources.
- Understand regulatory policies, market reforms, and technological advancements that could impact energy prices.
- Review energy sector publications, expert analyses, and videos or other resources.

Possible data sources include energy market databases, weather archives, government publications on energy, and news articles reporting on the energy sector. Combining comprehensive data collection with in-depth domain knowledge will lay a solid foundation for developing accurate predictive models and meaningful analysis.

Phase 2: Data Cleaning & Exploratory Data Analysis (EDA)

Understanding and optimizing the quality of your data is critical for uncovering insights from your data sources and setting the stage for predictive modeling, for example:

- Prepare your data for analysis by handling missing values, outliers, and ensuring data quality.
- Use statistical summaries and visualizations to understand the basic characteristics of the data. Explore the distribution of key variables and identify patterns or anomalies.
- Investigate the relationships between different variables, especially how various factors such as weather conditions, demand, and significant events correlate with energy prices. Utilize statistical methods like Pearson's or Spearman's rank correlation coefficient to quantify the strength and direction of these relationships. This step is crucial for identifying potential predictors for your models.

This initial analysis will not only help you understand the data better but also guide your choice of features for the predictive modeling phase. It's essential to document any insights or interesting findings during this stage, as they could be valuable for the final report and presentation.

Phase 3: Visualization & Storytelling

Effective presentation is key to interpreting the data and predictive modeling results comprehensively, for example:

- Create visualizations that illustrate the historical trends in energy prices and highlight the key factors influencing price fluctuations.
- Use graphs to compare actual vs. predicted energy prices to evaluate the model's performance visually.
- Explain your findings by connecting the dots between visuals, forecasts, and insights, making it easier to understand how energy prices change and what that means for people and businesses.

Visualizations should not only aim to support the narrative of your findings, but also uncover insights that might not be immediately apparent from the raw data or statistical metrics alone. A well-crafted visualization can make complex data more accessible and insights more compelling.

Phase 4: Predictive Modeling

To forecast average hourly energy prices on the target date, you may consider various modeling techniques such as:

- Linear Regression
- Time Series Analysis (e.g. ARIMA)
- Machine Learning models (e.g. Random Forest, Neural Networks)

Evaluate different models based on reasonable performance metrics (e.g. RMSE, MAE) to select the best-performing model.

Evaluation Criteria

Your submission will be evaluated based on the following criteria:

- **Data Analysis Depth:** The thoroughness of your initial data analysis and insights.
- **Presentation:** The clarity and effectiveness of your data visualization and presentation.
- **Model Complexity and Justification:** The complexity of your model and the rationale behind choosing it.
- **Innovativeness:** The novelty of your approach in predicting energy prices.
- **Accuracy of Predictions:** The precision of your model's energy price forecasts.

Submission

Details on how to submit have not been finalized as of writing. Please check <https://btw2025.gi.de/data-science-challenge> for updates.

UPDATE From January 27, 2025: Clarification & Supporting Material

Clarification on the Submission Artifacts

Your submission should include:

1. a **PDF file** generated from your notebook that can be read as a report without executing the python code,
2. a **CSV file** containing your prediction, and
3. a **zip archive** containing the python notebook file and optionally also additional artifacts such as crawled data.

Python Functions for Measuring Prediction Accuracy

Below are two Python functions:

1. Function for Downloading Data

This function automates the retrieval of day-ahead energy price data from the SMARD.de website and saves it in the format that will be used for evaluating the submitted predictions.

2. Prediction Evaluation Function

This function evaluates prediction CSV files against the target values for the given date, ranks predictions based on a selection of comparison functions, and generates a visualization of the results.

3. Generate Sample Predictions

This function generates sample CSV files to illustrate how the objective part of our evaluation is supposed to work and in what format your predictions should be submitted.

We will use this code as a basis for determining the most accurate prediction in the review process. (While we may implement minor changes for review, you should use this function to check whether your prediction CSV file is formatted correctly.)

Please make sure that your submitted CSV file work as input for the evaluation function below. Thank you!

```
In [6]: import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
from datetime import datetime
import pytz
from typing import Callable, List, Dict, Tuple
```

```
In [7]: def download_smard_data(target_date: str, output_dir: str):
        """
        Downloads day-ahead price data for a specified date from the SMARD.de website
        and retrieves the 24-hour data for that specific day.

        Args:
            target_date (str): Date in the format 'YYYY-MM-DD'.
            output_dir (str): Directory to save the downloaded CSV file.
        """
        # Define timezone
        local_timezone = pytz.timezone("Europe/Berlin")

        # Parse the target date
        date_string = f"{target_date} 00:00:00"
        local_date_object = datetime.strptime(date_string, "%Y-%m-%d %H:%M:%S")
        localized_date_object = local_timezone.localize(local_date_object)
        epoch_timestamp = int(localized_date_object.timestamp())

        # Calculate the offset based on the weekday (0 = Monday, 6 = Sunday)
        weekday = localized_date_object.weekday() # Monday = 0, Sunday = 6
        hour_offset = weekday * 24

        timestamp_in_milliseconds = epoch_timestamp * 1000 - (hour_offset * 3600 * 1000)

        # Fetch data from the SMARD.de API (returns 168 hours of data starting from the time
        url = f"https://www.smard.de/app/chart_data/4169/DE/4169_DE_hour_{timestamp_in_milli
        response = requests.get(url)
        if response.status_code != 200:
            print(f"Failed to fetch data: {response.status_code}")
            return
```

```

data = response.json()

if "series" not in data or not data["series"]:
    print("No data available for the specified date.")
    return

series = data["series"]

# Extract the 24-hour data for the given date
start_index = hour_offset
end_index = hour_offset + 24
day_series = series[start_index:end_index]
observed_prices = [item[1] for item in day_series]

print(f"Data Series for {target_date}: {day_series}")
print(f"Hourly Prices for {target_date}: {observed_prices}")

# Prepare the data as a DataFrame
df = pd.DataFrame({
    "Hour": list(range(1, 25)),
    "Germany/Luxembourg in €/MWh": observed_prices
})

# Create the output directory structure
date_dir = os.path.join(output_dir, target_date)
os.makedirs(date_dir, exist_ok=True)

# Save the CSV file
output_file = os.path.join(date_dir, "actual_values.csv")
df.to_csv(output_file, index=False)
print(f"Data saved to: {output_file}")

def generate_sample_predictions(actual_values_filename: str, output_dir: str):
    """
    Generates sample prediction files that deviate from the actual values.

    Args:
        actual_values_path (str): Path to the actual values CSV file.
        output_dir (str): Directory to save the generated prediction files.

    Example:
        generate_sample_predictions('actual_values.csv', './data/2024-03-18')
    """
    # Load the actual values
    actual_values_path = os.path.join(output_dir, actual_values_filename)
    actual_values_df = pd.read_csv(actual_values_path)
    actual_values = actual_values_df["Germany/Luxembourg in €/MWh"].tolist()

    # Generate deviating predictions
    predictions = {
        "prediction_1.csv": [v * 1.1 for v in actual_values], # 10% higher
        "prediction_2.csv": [v * 0.9 for v in actual_values], # 10% lower
        "prediction_3.csv": [v + np.random.uniform(-50, 50) for v in actual_values], #
        "prediction_4.csv": [v if i % 2 == 0 else v * 1.2 for i, v in enumerate(actual_v
    ]

    # Save the generated predictions as CSV files
    os.makedirs(output_dir, exist_ok=True)
    for filename, prediction in predictions.items():
        prediction_df = pd.DataFrame({
            "Hour": list(range(1, 25)),
            "Germany/Luxembourg in €/MWh": prediction
        })
        output_path = os.path.join(output_dir, filename)

```

```
prediction_df.to_csv(output_path, index=False)
print(f"Prediction saved to: {output_path}")
```

```
In [8]: # Evaluation metric functions
EVALUATION_METRICS: List[Tuple[str, Callable[[List[float], List[float]], float]]] = [
    ("MAE", lambda target, pred: sum(abs(t - p) for t, p in zip(target, pred)) / len(target, pred)),
    ("MSE", lambda target, pred: sum((t - p) ** 2 for t, p in zip(target, pred)) / len(target, pred)),
    ("RMSE", lambda target, pred: (sum((t - p) ** 2 for t, p in zip(target, pred)) / len(target, pred)) ** 0.5),
    ("MAPE", lambda target, pred: sum(abs((t - p) / t) for t, p in zip(target, pred) if t != 0) / len(target, pred))
]

# Function to evaluate predictions
def evaluate_predictions_from_directory(output_dir: str, actual_values_filename: str = "actual_values.csv"):
    """
    Evaluates and ranks prediction files in a directory against the actual values file.

    Args:
        output_dir (str): Directory containing the actual values file and prediction files.
        actual_values_filename (str): Name of the actual values file (default is "actual_values.csv").

    Returns:
        pd.DataFrame: Ranking of prediction files with their scores for each function.
    """
    # Locate the actual values file
    actual_values_path = os.path.join(output_dir, actual_values_filename)
    if not os.path.exists(actual_values_path):
        raise FileNotFoundError(f"Actual values file not found at {actual_values_path}")

    # Load the actual values
    actual_df = pd.read_csv(actual_values_path)
    actual_values = actual_df["Germany/Luxembourg in €/MWh"].tolist()

    # Find all prediction files in the directory
    prediction_files = [
        os.path.join(output_dir, f) for f in os.listdir(output_dir)
        if f.endswith(".csv") and f != actual_values_filename
    ]

    results = []
    for file in prediction_files:
        # Load the prediction data
        prediction_df = pd.read_csv(file)
        prediction_values = prediction_df["Germany/Luxembourg in €/MWh"].tolist()

        # Compute scores for all functions
        scores = {}
        for name, func in EVALUATION_METRICS:
            scores[name] = func(actual_values, prediction_values)
        results.append({"File": os.path.basename(file), **scores})

    # Create a DataFrame for scores and ranks separately
    rankings_df = pd.DataFrame(results)

    # Create scores DataFrame
    scores_df = rankings_df[["File"] + [metric[0] for metric in EVALUATION_METRICS]]

    # Add ranking columns for each metric
    for name in [metric[0] for metric in EVALUATION_METRICS]:
        rankings_df[f"Rank_{name}"] = rankings_df[name].rank(ascending=True)

    # Compute total ranking position
    rank_cols = [col for col in rankings_df.columns if col.startswith("Rank_")]
    rankings_df["Average_Rank"] = rankings_df[rank_cols].mean(axis=1)

    # Create ranks DataFrame
    ranks_df = rankings_df[["File", "Average_Rank"] + rank_cols]
```

```

ranks_df = ranks_df.sort_values(by="Average_Rank")

# Visualization
plt.figure(figsize=(12, 6))
plt.step(range(1, 25), actual_values, label="Actual Values", linewidth=3, color="black")

for file in prediction_files:
    prediction_df = pd.read_csv(file)
    prediction_values = prediction_df["Germany/Luxembourg in €/MWh"].tolist()
    # plt.step(range(1, 25), prediction_values, label=os.path.basename(file), alpha=0.2)
    plt.fill_between(
        range(1, 25), prediction_values, actual_values,
        step="post", alpha=0.2, label=f"Error: {os.path.basename(file)}"
    )

plt.xlabel("Hour")
plt.ylabel("Energy Price (€/MWh)")
plt.title("Comparison of Actual Values and Predictions")
plt.legend()
plt.show()

return scores_df, ranks_df

```

```

In [9]: # Example Usage
data_directory = os.path.join('data_science_challenge', 'data')
exampleDate = '2024-03-18'
exampleDate_directory = os.path.join(data_directory, exampleDate)

download_smard_data(exampleDate, data_directory) # downloads the actual values for a given date
generate_sample_predictions('actual_values.csv', exampleDate_directory) # generate sample predictions

scores, ranks = evaluate_predictions_from_directory(
    output_dir=os.path.join(data_directory, exampleDate)
)

# Print scores and ranks without index
print("Scores:")
print(scores.to_string(index=False))
print("\nRanks:")
print(ranks.to_string(index=False))

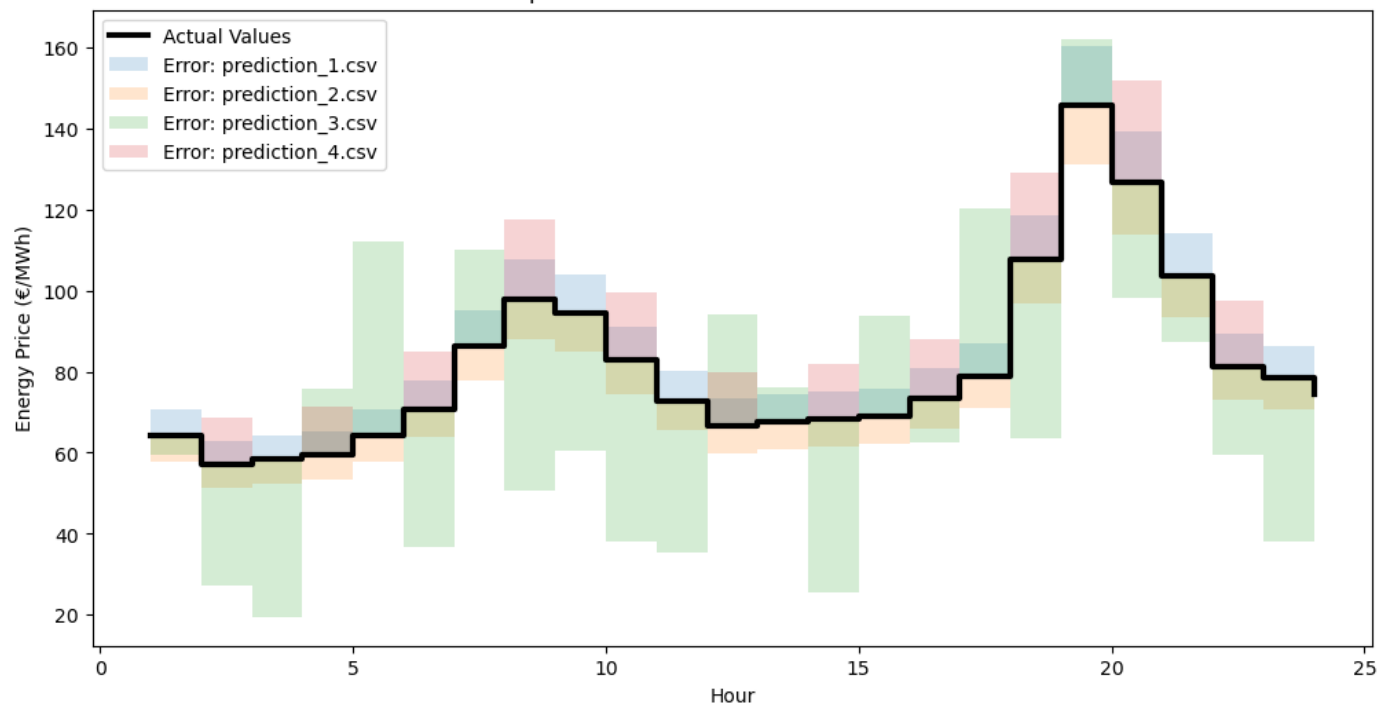
```

```

Data Series for 2024-03-18: [[1710716400000, 64.12], [1710720000000, 57.14], [1710723600000, 58.31], [1710727200000, 59.38], [1710730800000, 64.38], [1710734400000, 70.8], [1710738000000, 86.32], [1710741600000, 97.87], [1710745200000, 94.45], [1710748800000, 82.8], [1710752400000, 72.84], [1710756000000, 66.64], [1710759600000, 67.7], [1710763200000, 68.3], [1710766800000, 68.94], [1710770400000, 73.4], [1710774000000, 78.98], [1710777600000, 107.65], [1710781200000, 145.71], [1710784800000, 126.53], [1710788400000, 103.72], [1710792000000, 81.2], [1710795600000, 78.38], [1710799200000, 74.45]]
Hourly Prices for 2024-03-18: [64.12, 57.14, 58.31, 59.38, 64.38, 70.8, 86.32, 97.87, 94.45, 82.8, 72.84, 66.64, 67.7, 68.3, 68.94, 73.4, 78.98, 107.65, 145.71, 126.53, 103.72, 81.2, 78.38, 74.45]
Data saved to: data_science_challenge\data\2024-03-18\actual_values.csv
Prediction saved to: data_science_challenge\data\2024-03-18\prediction_1.csv
Prediction saved to: data_science_challenge\data\2024-03-18\prediction_2.csv
Prediction saved to: data_science_challenge\data\2024-03-18\prediction_3.csv
Prediction saved to: data_science_challenge\data\2024-03-18\prediction_4.csv

```

Comparison of Actual Values and Predictions



Scores:

File	MAE	MSE	RMSE	MAPE
prediction_1.csv	8.125042	70.627614	8.404024	10.000000
prediction_2.csv	8.125042	70.627614	8.404024	10.000000
prediction_3.csv	30.247598	1080.135382	32.865413	39.193163
prediction_4.csv	8.051333	137.443211	11.723618	10.000000

Ranks:

File	Average_Rank	Rank_MAE	Rank_MSE	Rank_RMSE	Rank_MAPE
prediction_2.csv	1.375	2.0	1.0	1.0	1.5
prediction_4.csv	2.125	1.0	3.0	3.0	1.5
prediction_1.csv	2.500	3.0	2.0	2.0	3.0
prediction_3.csv	4.000	4.0	4.0	4.0	4.0