Forecasting Electricity Production - Time Series Project

Alex Makassiouk

November 2024

NTNU, Trondheim

TMA4285

Abstract

This project examines the relationship between precipitation and hydroelectric electricity production in Trøndelag county, Norway. We use monthly precipitation data spanning from 1950 to 2019 and monthly electricity production data spanning from 1993 to 2011 to investigate patterns, seasonality and the potential influence of precipitation on electricity production. Seasonal trends were found, with electricity production peaking in winter. Statistical approaches, including ARIMA and ARMAX were employed to capture the dynamics and forecast electricity production.

1 Introduction

In this project we are working on two datasets. The main data is data on electricity production in Trøndelag county in Norway. This data is monthly production in MWh from January 1993 to December 2011. We know from [1] that hydroelectric power was the source for 83 percent of Norwegian electricity. An important part of producing hydroelectric power is water which comes in the form of precipitation. The other dataset we will work with is monthly precipitation data in Trondheim which is located in Trøndelag county, and we will assume that this precipitation will generalize to the rest of the county and to where the electricity is produced. This data contains monthly precipitation in mm from 1950 until 2019. So we have far more data on precipitation.
One natural question which we will investigate in this project is how the precipitation influences the production of electricity, and so precipitation will be treated as an exogenous variable to the amount of production, and one hypothesis is that production of electricity in Trøndelag correlates with the amount of precipitation, and even stronger is *caused* by the precipitation levels.
Another fair hypothesis to make is that electricity production is somewhat seasonal due to higher electricity demand in colder winter months. A similar argument can be made for precipitation that it is seasonal because of the purest example of what seasonality is: That the planet earth goes through seasons and a change in climate because of the planet's tilt and rotation around the sun.
Let us first visualize the data that we have to work with. This is a safe first move in determining if the time series data is stationary.

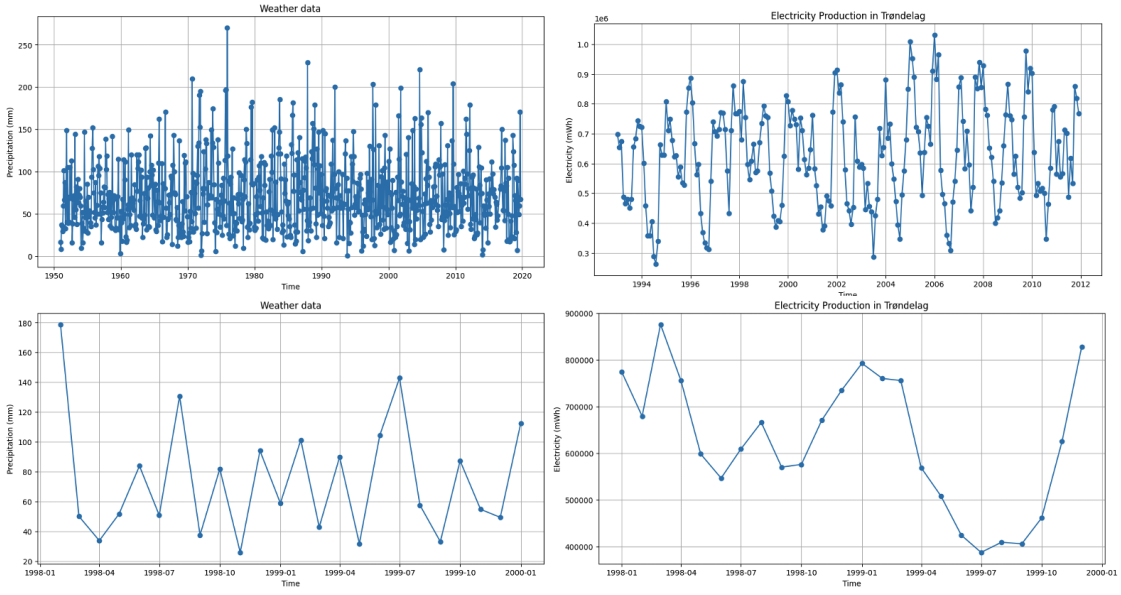

Figure 1: Data visualization

In figure 1 we see all of our data with precipitation on the left and electricity production on the right, as well as zoomed in plots on the years 1998-1999. Looking at the full data, it's hard to see any clear patterns. The electricity production does seem to slightly trend upwards, but it is hard to tell. We need to dig deeper to see if the data exhibits any trends, seasonality and if it is stationary, and hopefully we can later predict electricity production based on time and lagged precipitation.

2 Results

The first thing we do with our data is analyzing the autocorrelation and identifying seasonality based on the initial thoughts from Section 1.

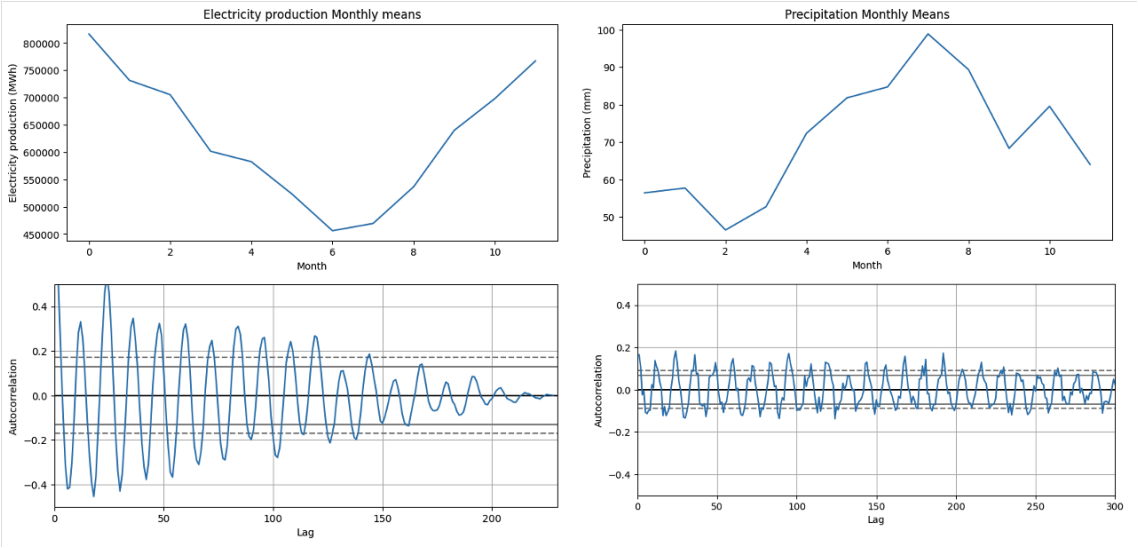

Figure 2: Monthly means and autocorrelation functions for both datasets

Figure 2 provides several insights. The autocorrelation functions in the bottom row gives valuable insight into how the data correlates with itself over time and we see cyclical behavior in both datasets with a period of 12 months. While the ACF for electricity production decays after about 150 months, the weather data seems to stay seasonal for all lags, confirming the seasonal nature of nature itself. The top row shows the means of the data grouped by month, giving a long-term insight which strengthens the seasonal feeling of both the weather data and electricity data. We can see that for electricity production, the mean over all years is lowest for July and highest for the winter months February, December and January. The precipitation cycle is shifted with the peak being in August and minimum in March.

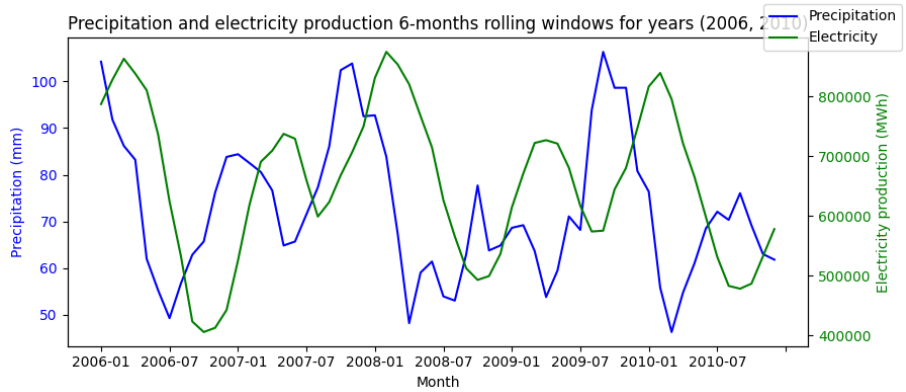

Figure 3: 6 Month rolling window mean

In figure 3 we see a rolling window of length 6 zoomed in on the years 2006-2010, giving a nice lag effect between the precipitation and electricity production, with electricity peaks following precipitation peaks.

2.1 ARIMA Modeling

Now we will aim to fit variants of ARIMA models with extensions. An ARIMA model can in general be written as:

$$\phi(B)(1-B)^d x_t = \delta + \theta(B)w_t$$
$$\delta := \mu(1-\phi_1 - \cdots \phi_p)$$
$$\mu = E(\nabla^d x_t)$$

These equations can be found in definition 3.11 in [2].

We will use Python 3.11 (uncorrelated with the definition above) for modeling with the essential data processing modules and the statsmodels module for where we want richer functionality than plain implementations.
We will focus on two main approaches to the ARIMA modeling. Either with or without precipitation data as an exogenous explanatory variable for the electricity production. Many varieties can be considered of the ARIMA itself and we search through these in various ways. One first questions we ask ourself is if we difference our data. This is explored first with plots by differencing on values like 1, 6, 12 and looking at the data.

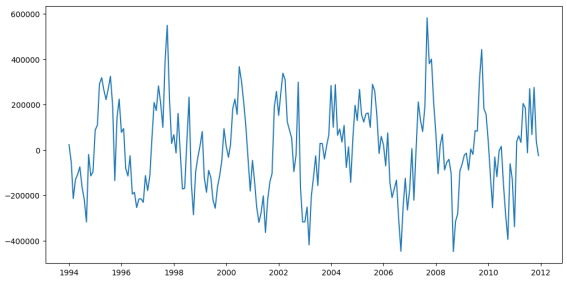

Figure 4: 12 months differenced electricity production

The differenced data essentially looks similar as our normal data but with a shifted mean to 0. We decide to not go forward with transformed data only but rather test with different values for d in the ARIMA model.

Our approach when finding the correct model is doing a grid search over possible parameters. We start with the simpler ARIMA model without the exogenous weather data and fit a model on the pure electricity production time series.
We define a range for the parameters, iterate over these and fit a model to a subset of our data which will be our training data. We then do one step predictions of the remaining test data, and calculate the total mean squared error of that model's predictions over the whole test data. When doing one step predictions we do a prediction and then get an observation and add the observation to our training data and fit a new model which includes this data before we predict the next one.
For searching we start with a broader range of parameters before we narrow the values down, in order to make the search feasible on time. At first, the setup $(p, d, q) = (2, 0, 1)$ looked like a good candidate, but was in the end beaten by the parameters $(p, d, q) = (12, 0, 12)$. We see that differencing did not help the ARIMA model, but the best parameters make it so that the ARIMA model uses lagged terms from the whole cycle of 12 months.

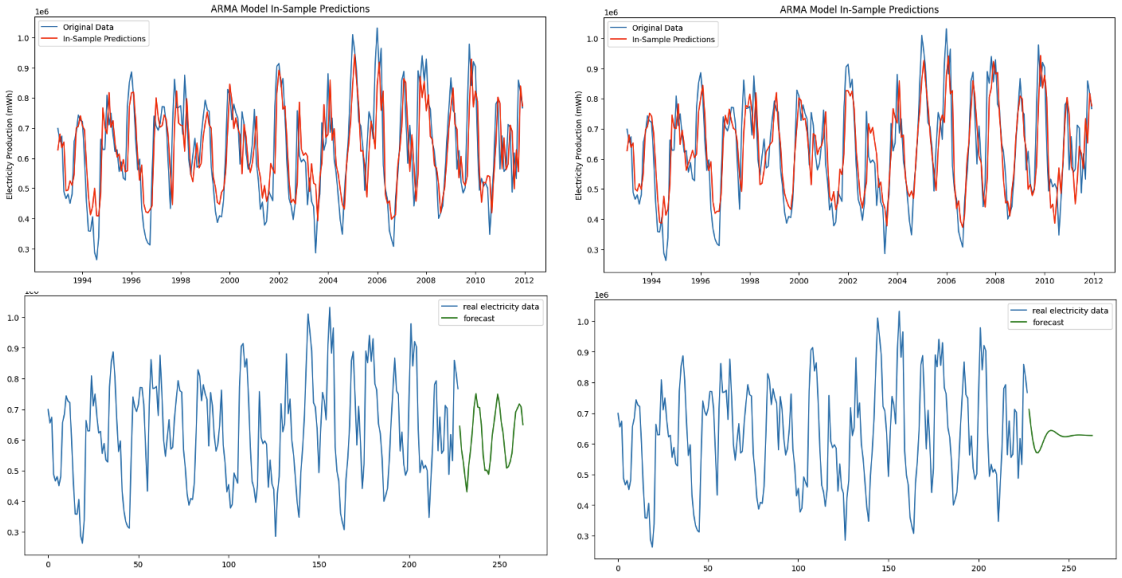

Figure 5: In-sample predictions and forecasts for (12,0,12) and (2,0,1) ARIMA models

We do however see a major concern in all of the p-values for the coefficients $\boldsymbol{\theta}, \boldsymbol{\phi}$. Meanwhile all parameters for the $(2, 0, 1)$ - model end up being significant, but with lower forecasting power as seen in figure 5. We note that both the best-scoring models do not difference and that the forecasts from the $(12, 0, 12)$ model forecasts the same kind of cycles as we see in for example figure 2.

2.2 ARMAX

Now we want to use the weather data as an exogenous explanatory variable to the ARIMA model so that we add a term to the model that depends on the previous value in the weather data. This gives us the ARMAX model from (5.91) in [2]:

$$x_t = \Gamma u_t + \sum_{j=1}^{p} \Phi_j x_{t-j} + \sum_{k=1}^{q} \Theta_k w_{t-k} + w_t$$

Using this model we can fit it on a training set, then forecast on the validation data we held out from fitting so that we can compare the model forecast with the real data.

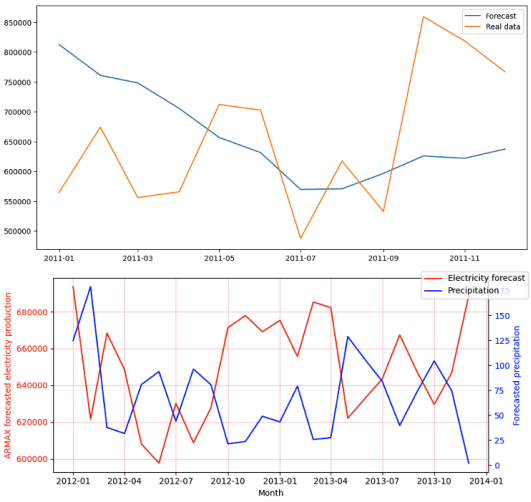

Figure 6: ARMAX validation and forecast

3 Discussion

We see that especially ARMAX gives interesting results. The plot of the ARMAX looks to capture the seasonality with peaks in the winter and lower values in the summer as well as local peaks lagged a month or two after periods of high precipitation values. The forecast done on the validation data also looks promising and looks like a smoothed out version of the ground truth electricity production. The standard ARIMA models look a little bit limited in this sense in that the forecast looks very homogeneous or decays quickly, so we see much value from adding in the weather data as explanatory variable, and a deeper analysis of the explanatory power of the precipitation data should be done. Because of the highly volatile nature of both of these datasets, it seems like trying out GARCH should also give interesting results.
In terms of implementation of the models, these could be done from scratch but more educational value was found in tuning existing models and looking for interesting results to improve the practical skills of time series modeling on a real life project like this.
Another point I would like to address is how the grid search was done when finding the best hyperparameters for the ARIMA modeling. The inference was done using only a one-step ahead prediction and choosing hyperparameters based on the most accurate one-step ahead predictions, when in reality, long-term forecasting should also be implemented to find more general hyperparameters that can perform well on forecasting longer into the future.

4 Conclusion

In this project we investigated precipitation and hydroelectric electricity production and the relationship between these, in Trøndelag county in Norway, leveraging a few time series techniques. By applying models such as ARIMA and ARMAX, we explored how precipitation, treated as an exogenous variable, influences the electricity production and the seasonal patterns.

The results highlighted distinct seasonlity in both datasets, with electricity production peaking during winter, and precipitation during late summer to autumn. The autocorrelation analysis and monthly means emphasized the cyclical nature of the data. We found the ARIMA-model that used 12-month terms to be best-performing. However, when we incorporated the precipitation data in the ARMAX-model the model began to look more interesting in the forecasting sense, as it also revealed lagged effects of precipitation on electricity production.

A lot of areas for improvement remain. The time series project in TDT4173: Modern machine learning in practice had a deadline the week before and took a lot of the author's capacity. However some experience from the machine learning project came in handy when working on this time series data but using classical statistical methods instead. We can enhance the hyperparameter selection for long-term forecasting as well as explore volatility models like GARCH to further provide clarity and accuracy in the modeling. Overall, this project gave a very nice practice in the hands-on statistical modeling to understand and forecast complex systems.

References

[1] Norges vassdrags- og energidirektorat. *Hvor kommer strømmen fra?* https://www.nve.no/energi/energisystem/kraftproduksjon/hvor-kommer-stroemmen-fra

[2] Shumway, Robert H. Stoffer, David S. *Time Series Analysis and Its Applications 4th edition*

5 Appendix

5.1 Note on the use of Generative AI

A personal approach to the growing use of GenAI has been to limit it in order to maximize the learning, which is the whole reason of studying at the university. My personal opinion is that it is in general used too much among students, and that valuable learning is lost underway in practical projects like these. I do however see the value at certain times. The main use of GenAI in this project has been Github's Copilot extension in the IDE Visual Studio Code. Primarily when I had a specific need for my code which consists of around 1-10 lines of code. For example making a plot correct for setting two different y-axes or something similar to this. The copilot is however limited in this sense as well as it does not always look to be updated on the latest package versions etc. and so diving into documentation is still necessary at times to surpass the Copilot's limitations.

5.2 Code

The repository for this project can be found on my personal GitHub (github.com/alexmakassiouk/ntnu-time-series) as soon as it is made public after the deadline of the project. A summary of some of the code from this repository will be added here.

```
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import pandas as pd
from pandas.plotting import autocorrelation_plot
from sklearn.metrics import mean_squared_error
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.tsa.stattools as ts
import warnings


def process_data(weather_data_filepath, electricity_data_filepath):
    df_weather = pd.read_csv(weather_data_filepath, delimiter=",")
    df_electricity = pd.read_csv(
        electricity_data_filepath, delimiter=","
    )
    df_weather["time"] = pd.to_datetime(df_weather["referenceTime"])
    df_electricity["time"] = pd.to_datetime(
        df_electricity["m ned"], format="%YM%m"
    )
    df_weather = df_weather.drop(columns="referenceTime")
    df_electricity = df_electricity.drop(columns="m ned")
    df_weather["month_year"] = df_weather["time"].dt.strftime("%Y-%m")
    df_electricity["month_year"] = df_electricity["time"].dt.strftime(
        "%Y-%m"
    )
    df_weather["rolling_mean_precipitation"] = (
        df_weather["value"].rolling(window=6).mean()
    )
```

```
    df_electricity["rolling_mean_electricity"] = (
        df_electricity["value"].rolling(window=6).mean()
    )
    df_filtered_weather = df_weather[
        (
            df_weather["month_year"]
            >= df_electricity["month_year"].min()
        )
        & (
            df_weather["month_year"]
            <= df_electricity["month_year"].max()
        )
    ]
    df_filtered_weather.reset_index(drop=True, inplace=True)

    df_merged = pd.merge(
        df_weather.drop(columns="time"),
        df_electricity.drop(columns="time"),
        on="month_year",
        suffixes=["_precipitation", "_electricity"],
        how="outer",
    )
    df_merged.sort_values(by="month_year", inplace=True)

    return df_weather, df_electricity, df_merged


def get_cropped_df(df, start_year, end_year):
    df["month_year"] = pd.to_datetime(
        df["month_year"], format="%Y-%m"
    )
    cropped_df_merged = df[
        (df["month_year"].dt.year >= start_year)
        & (df["month_year"].dt.year <= end_year)
    ]
    cropped_df_merged.reset_index(drop=True, inplace=True)
    return cropped_df_merged


def plot_values(
    data: pd.DataFrame,
    x_var="time",
    y_var="value",
    title: str = None,
    x_label="Time",
    y_label="Value",
):
    plt.figure(figsize=(12, 6))
    plt.plot(data[x_var], data[y_var], marker="o")
    plt.title(title)
    plt.xlabel(x_label)
```

```
    plt.ylabel(y_label)
    plt.grid(True)
    plt.show()


def print_data_summary(df: pd.DataFrame):
    summary = df["value"].describe()
    print("\n")
    print(f"Data Summary:")
    print(summary)
    print("\n")


def plot_rolling_mean_together(
    df: pd.DataFrame, start_year: int, end_year: int
):
    df = get_cropped_df(df, start_year, end_year)
    fig, ax1 = plt.subplots(figsize=(12, 6))
    ax1.plot(
        df["month_year"],
        df["rolling_mean_precipitation"],
        color="blue",
        label="Precipitation",
    )
    ax1.set_xlabel("Month")
    ax1.set_ylabel("Precipitation (mm)", color="blue")
    ax1.tick_params(axis="y", labelcolor="blue")
    ax1.xaxis.set_major_locator(MaxNLocator(nbins=12))

    ax2 = ax1.twinx()
    ax2.plot(
        df["month_year"],
        df["rolling_mean_electricity"],
        color="green",
        label="Electricity",
    )
    ax2.set_ylabel("Electricity production (MWh)", color="green")
    ax2.tick_params(axis="y", labelcolor="green")
    if end_year - start_year == 0:
        plt.title(
            f"""Precipitation and electricity production
            6-months rolling windows for year {start_year}"""
        )
    else:
        plt.title(
            f"""Precipitation and electricity production
             6-months rolling windows for years {start_year, end_year}"""
        )
    fig.tight_layout()
    fig.legend()
    plt.show()
```

```
def print_adf(df: pd.DataFrame):
    adf_result = ts.adfuller(df["value"])
    print("ADF Statistic:", adf_result[0])
    print("p-value:", adf_result[1])
    print("Critical Values:")
    for key, value in adf_result[4].items():
        print(f"   {key}: {value}")
    print("\n")


def plot_monthly_means(df: pd.DataFrame, ylabel: str, title: str):
    df = df.reset_index(drop=False)
    df = df.set_index("month_year", drop=False)
    df.index = pd.to_datetime(df.index, format="%Y-%m")
    monthly_avg = df.groupby(df.index.month)["value"].mean()
    plt.figure(figsize=(12, 6))
    plt.plot(range(1, 13), monthly_avg, marker="o")
    plt.xticks(
        range(1, 13),
        [
            "Jan",
            "Feb",
            "Mar",
            "Apr",
            "May",
            "Jun",
            "Jul",
            "Aug",
            "Sep",
            "Oct",
            "Nov",
            "Dec",
        ],
    )
    plt.xlabel("Month")
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()


def plot_acf(df: pd.DataFrame, xlim: tuple, ylim: tuple):
    plt.figure(figsize=(12, 6))
    autocorrelation_plot(df["value"])
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.show()


def plot_decomposed(df: pd.DataFrame, period: int):
```

```
    decomp = seasonal_decompose(
        df["value"], model="additive", period=period
    )
    decomp.plot()
    plt.show()


def plot_diffed_series(df: pd.DataFrame, lag: int):
    df["value_diff"] = df["value"].diff(lag)
    df.dropna()
    plt.figure(figsize=(12, 6))
    plt.plot(df["value_diff"])
    plt.show()


def evaluate_arima_model(X, order=(2, 0, 1)):
    train_size = int(len(X) * 0.66)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=order)
        model_fit = model.fit()
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    error = mean_squared_error(test, predictions)
    return error


def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype("float32")
    sorted_models = {}
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p, d, q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                        print("ARIMA%s␣MSE=%.3f" % (order, mse))
                    sorted_models[f"ARIMA{order}"] = mse
                except:
                    continue
    sorted_models = {
        k: v
        for k, v in sorted(
            sorted_models.items(), key=lambda item: item[1]
        )
```

```
    }
    print("Sorted from best to worst:")
    for model, mse in sorted_models.items():
        print(f"{model}: MSE={mse:.3f}")
    print("\n")
    print("Best ARIMA%s MSE=%.3f" % (best_cfg, best_score))


def plot_insample_preds(
    preds: pd.DataFrame, ground_truth: pd.DataFrame
):
    plt.figure(figsize=(12, 6))
    plt.plot(ground_truth["value"], label="Ground truth")
    plt.plot(preds, label="In-Sample Predictions", color="red")
    plt.xlabel("Date")
    plt.ylabel("Electricity Production (mWh)")
    plt.title("ARMA Model In-Sample Predictions")
    plt.legend()
    plt.show()


def plot_forecast(model_fit, steps: int, ground_truth: pd.DataFrame):
    forecast = model_fit.forecast(steps=steps)
    plt.figure(figsize=(12, 6))
    plt.plot(
        range(0, len(ground_truth)),
        ground_truth["value"],
        label="Ground truth",
    )
    plt.plot(
        range(len(ground_truth), len(ground_truth) + steps),
        forecast,
        color="green",
        label="forecast",
    )
    plt.legend()
    plt.show()


def plot_forecast_with_exog(
    forecast: pd.DataFrame, exog: pd.DataFrame
):
    fig, ax1 = plt.subplots(figsize=(12, 6))
    ax1.set_xlabel("Month")
    ax1.plot(forecast, label="Electricity forecast", color="red")
    ax1.set_ylabel(
        "ARMAX forecasted electricity production", color="red"
    )
    ax1.tick_params(axis="y", labelcolor="red")
    ax1.grid(color="pink")
    exog.index = forecast.index
```

```
    ax2 = ax1.twinx()
    ax2.plot(exog, label="Precipitation", color="blue")
    ax2.set_ylabel("Forecasted precipitation", color="blue")
    ax2.tick_params(axis="y", labelcolor="blue")
    fig.tight_layout()
    fig.legend()
    plt.show()


def main():
    WEATHER_DATA_FILEPATH = "data/weather_data.csv"
    ELECTRICITY_DATA_FILEPATH = "data/electricity_production_data.csv"
    df_weather, df_electricity, df_merged = process_data(
        WEATHER_DATA_FILEPATH, ELECTRICITY_DATA_FILEPATH
    )
    df_merged_all_values = df_merged.dropna()

    # PART 1: Exploration
    pre_millenium_cropped_weather_data = get_cropped_df(
        df_weather, 1998, 1999
    )
    pre_millenium_cropped_el_data = get_cropped_df(
        df_electricity, 1998, 1999
    )
    plot_values(
        pre_millenium_cropped_weather_data,
        title="Weather data",
        y_label="Precipitation (mm)",
    )
    plot_values(
        pre_millenium_cropped_el_data,
        title="Electricity production",
        y_label="Electricity (MWh)",
    )
    plot_rolling_mean_together(df_merged, 2006, 2010)
    print("Precipitation")
    print_data_summary(df_weather)
    print("Electricity production")
    print_data_summary(df_electricity)
    correlation = df_merged["value_precipitation"].corr(
        df_merged["value_electricity"]
    )
    print(
        f"""\nCorrelation between Precipitation
        and Electricity Production: {correlation}"""
    )

    # PART 2: Transformations
    print("Electricity ADF:")
    print_adf(df_electricity)
    print("Weather ADF:")
```

```
print_adf(df_weather)
plot_monthly_means(
    df_electricity,
    "Average electricity production (MWh)",
    "Average monthly Electricity production",
)
plot_monthly_means(
    df_weather,
    "Average precipitation (mm)",
    "Average monthly Precipitation",
)
plot_acf(df_electricity, (0, 230), (-0.5, 0.5))
plot_acf(df_weather, (0, 300), (-0.5, 0.5))
plot_decomposed(df_electricity, 12)
plot_decomposed(df_weather, 12)
plot_diffed_series(df_electricity, 12)
plot_diffed_series(df_weather, 12)

# PART 3: Modeling
AR = 2
DIFF = 0
MA = 1
order = (AR, DIFF, MA)
params = {"order": order}
p_values_grid = list(range(0, 3)) + [12]
d_values_grid = [0, 1, 6, 12]
q_values_grid = list(range(0, 3)) + [12]
warnings.filterwarnings("ignore")
evaluate_models(
    df_electricity["value"].values,
    p_values_grid,
    d_values_grid,
    q_values_grid,
)
model = ARIMA(df_electricity["value"], **params)
model_fit = model.fit()
print(model_fit.summary())
in_sample_preds = model_fit.predict(
    start=0, end=len(df_electricity) - 1, dynamic=False
)
plot_insample_preds(in_sample_preds, df_electricity)
plot_forecast(model_fit, 36, df_electricity)

## PART 3.2: ARMAX
exog = get_cropped_df(df_weather, 1993, 2011)["value"]
params["exog"] = exog
model_exog = ARIMA(df_electricity["value"], **params)
warnings.filterwarnings("ignore")
model_exog_fit = model_exog.fit()
print(model_exog_fit.summary())
forecast_weather_data = get_cropped_df(df_weather, 2012, 2013)[
```

```
        "value"
    ]
    exog_forecast = model_exog_fit.forecast(
        steps=24, exog=forecast_weather_data
    )
    plot_forecast_with_exog(exog_forecast, forecast_weather_data)
    armax_insample_preds = model_exog_fit.predict(
        start=0, end=len(df_electricity) - 1, dynamic=False
    )
    plot_insample_preds(armax_insample_preds, df_electricity)


main()
```