Assignment 1 - Statistics for Data Science

Dilirici Mihai, 411

June 1, 2025

1. Electric Production Time Series

1.1 Dataset and Objective

This project uses a monthly electric production dataset covering the period from January 1985 onward. It contains a single time series representing the volume of electric power production measured in gigawatt-hours (GWh). The aim is to model this series using ARIMA techniques and to forecast future values. Additionally, a synthetic dataset is used to validate an implementation of the Durbin-Levinson algorithm.

Source

`https://www.kaggle.com/datasets/kandij/electric-production`

1.2. Initial Time Series Exploration

The original time series shows strong seasonal fluctuations, with visible peaks and troughs that repeat annually. While there is no dramatic long-term trend, the seasonal variation is consistent. Monthly subseries plots and boxplots confirm the presence of seasonality, which is an important characteristic to address in modeling.

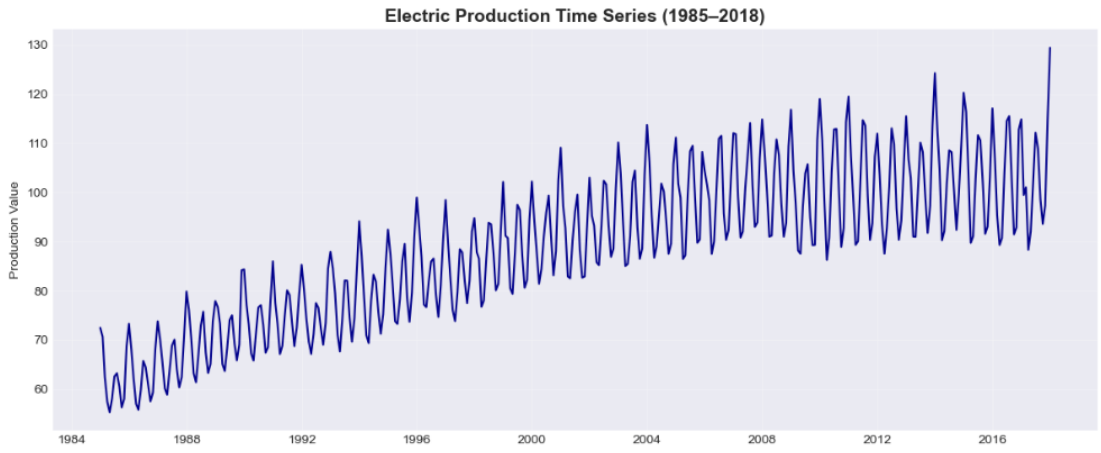

Figure 1: Electric Production Time Series: shows seasonality and trend.

1.3. Stationarity and Differencing

To determine whether the original time series was stationary, the Augmented Dickey-Fuller (ADF) test was applied. The test yielded a p-value greater than 0.05, indicating that the series was non-stationary. As a result, a first-order differencing transformation was performed to address both trend and seasonality. After differencing, the ADF test confirmed that the modified series exhibited stationarity, making it appropriate for ARIMA modeling.

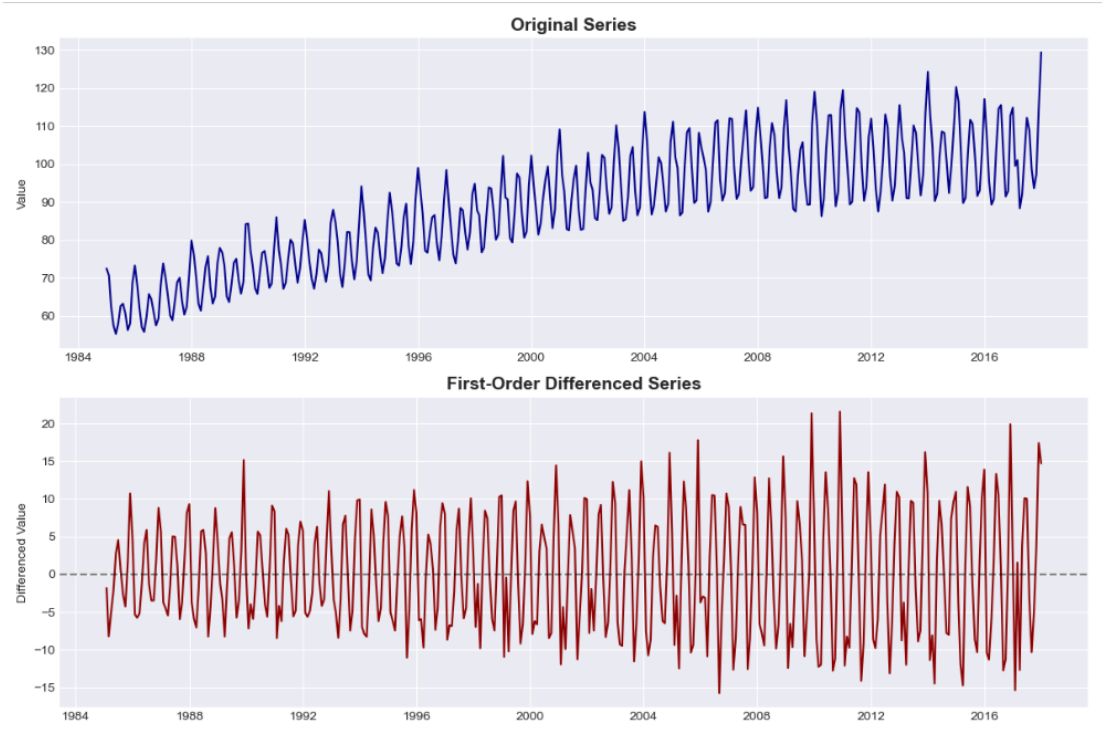

Figure 2: First-order Differenced Series (stationary)

1.4. Model Construction (Manual ARIMA)

ACF and PACF Analysis

To guide the selection of initial ARIMA model parameters, the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots of the differenced series were examined. Based on these diagnostics, multiple candidate models with varying (p, d, q) configurations—such as ARIMA(1,1,1), ARIMA(2,1,1), and ARIMA(3,1,0)—were evaluated. Model performance was compared using the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC), which helped identify the most suitable specification.

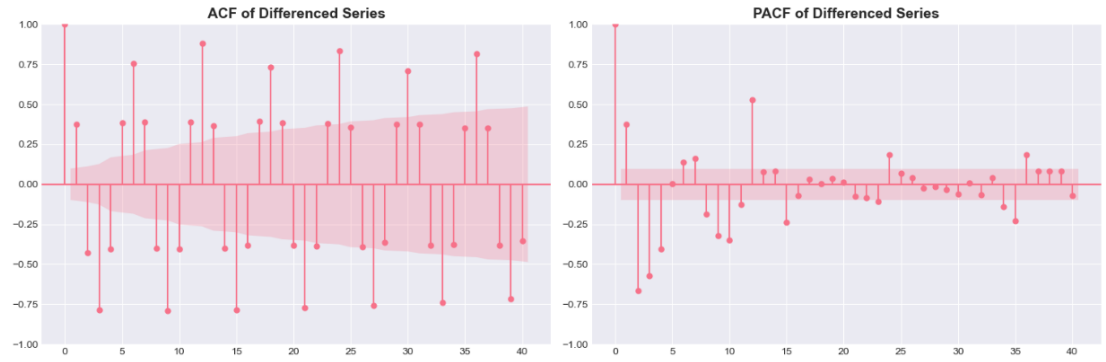

Figure 3: ACF and PACF of Differenced Series

1.5. Forecasting

The ARIMA model with the lowest AIC was selected as the best model. Using the best-fit ARIMA model, forecasts for the next 20 months were generated and visualized. The forecasted values extend the overall level and direction of the historical data, while remaining consistent with recent patterns observed in the series.

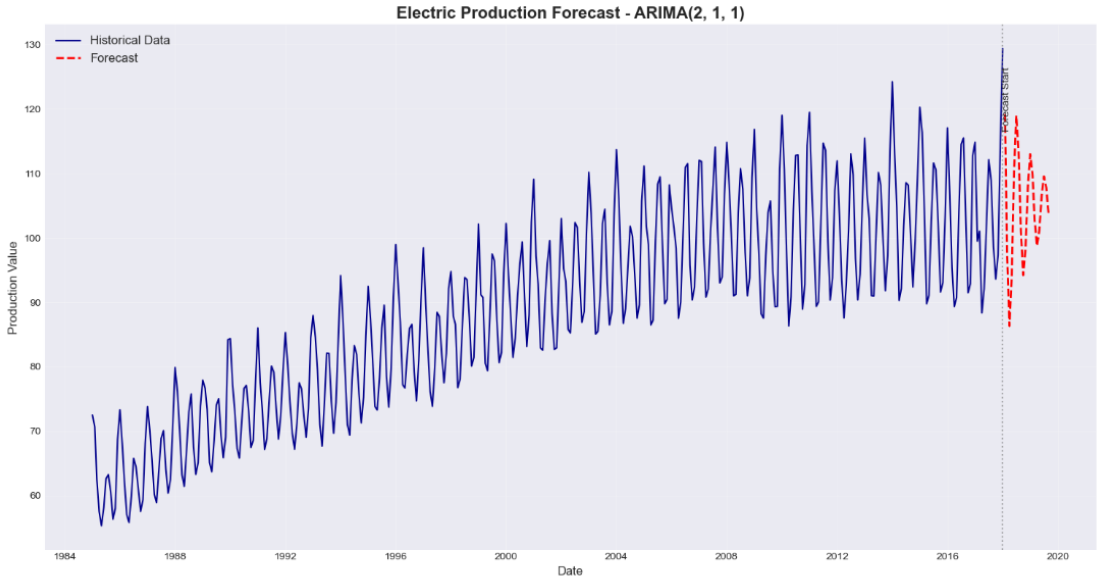

Figure 4: Manual ARIMA Forecast (20 steps)

1.6. Auto.ARIMA Comparison

To validate the manual model, the same time series was fitted using the `auto_arima()` function from the `pmdarima` library. The automatically selected model was very similar to the manual one in terms of structure and forecast output. The small differences between the forecasts confirm the validity of the manual model choice.

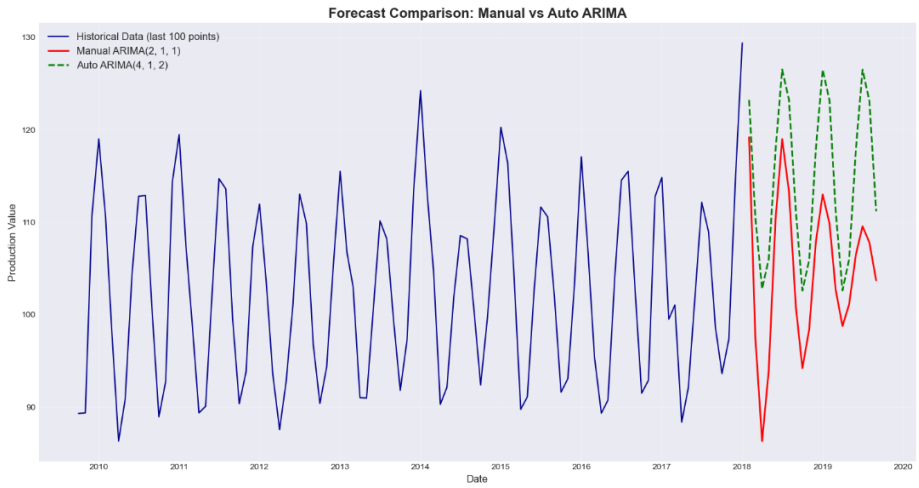

Figure 5: Comparison of Manual vs Auto ARIMA Forecasts

2. Durbin-Levinson Algorithm Implementation

To validate the Durbin-Levinson (D-L) algorithm, a synthetic autoregressive time series was generated. The simulated data follows an AR(2) process defined by the parameters $\Phi = (1, -0.9)$, meaning each value in the series depends linearly on its two preceding values, plus a random white noise term.

A total of 200 observations were created using this AR(2) structure. The following steps were followed:

1. **Sample Autocovariance Computation:** For lags $h = 0$ to 20, the sample autocovariances $\hat{\gamma}(h)$ were estimated using the formula from Course 5, Slide 2:

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-h} (x_t - \bar{x})(x_{t+h} - \bar{x})$$

   where $\bar{x}$ is the sample mean of the series.

2. **Durbin-Levinson Recursion:** The PACF values $\hat{\phi}_{mm}$ were estimated recursively using the D-L algorithm. The procedure updates coefficients iteratively using past estimates and residual variances.

3. **Comparison with Built-in PACF:** To validate the manual implementation, the estimated PACF values were compared with those produced by the `pacf()` function from the `statsmodels` package in Python.

The results are shown in Figure 6. The orange line (Durbin-Levinson) and the blue line (built-in PACF) nearly overlap for all lags, confirming the accuracy of the implementation. As expected from theory, the PACF cuts off after lag 2, indicating a correct recovery of the AR(2) structure.

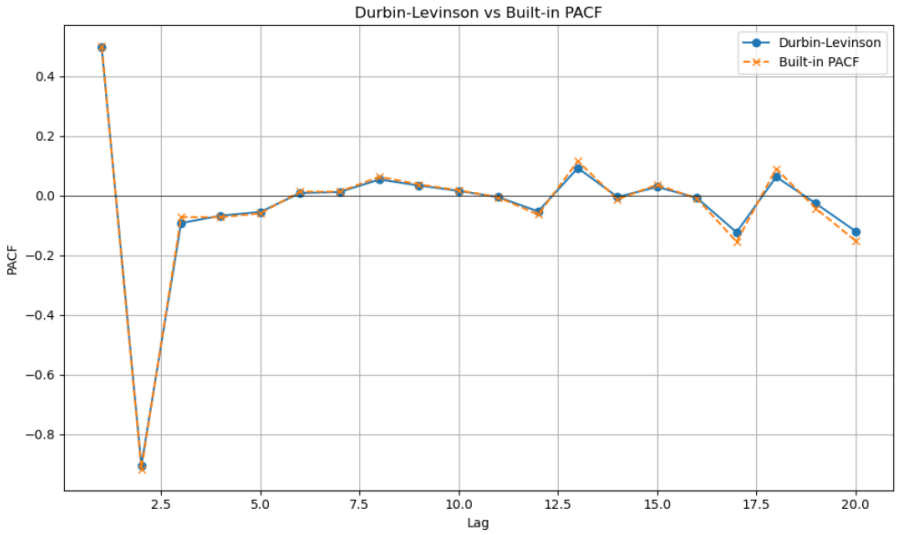

Figure 6: Durbin-Levinson PACF vs Built-in PACF

3: Python Code Used for Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as mplot
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.gofplots import qqplot
from scipy import stats
from pmdarima import auto_arima
import warnings
warnings.filterwarnings('ignore')

mplot.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")

df = pd.read_csv('Electric_Production.csv')
df['DATE'] = pd.to_datetime(df['DATE'])
df.set_index('DATE', inplace=True)

mplot.figure(figsize=(12, 5))
mplot.plot(df.index, df['Value'], linewidth=1.5, color='darkblue')
mplot.title('Electric Production Time Series (1985-2018)', fontsize=14, fontweight='bold')
mplot.ylabel('Production Value')
mplot.grid(True, alpha=0.3)
mplot.tight_layout()
mplot.show()

def adf_test(series):
    result = adfuller(series.dropna())
    return result[1] <= 0.05

is_stationary = adf_test(df['Value'])
```

```
fig, axes = mplot.subplots(2, 1, figsize=(12, 8))
plot_acf(df['Value'], lags=40, ax=axes[0], alpha=0.05)
axes[0].set_title('ACF of Original Series', fontsize=14, fontweight='bold')
plot_pacf(df['Value'], lags=40, ax=axes[1], alpha=0.05)
axes[1].set_title('PACF of Original Series', fontsize=14, fontweight='bold')
mplot.tight_layout()
mplot.show()

df['Value_diff'] = df['Value'].diff()
df_diff = df['Value_diff'].dropna()


fig, axes = mplot.subplots(2, 1, figsize=(12, 8))
axes[0].plot(df.index, df['Value'], linewidth=1.5, color='darkblue')
axes[0].set_title('Original Series', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Value')
axes[1].plot(df_diff.index, df_diff, linewidth=1.5, color='darkred')
axes[1].set_title('First-Order Differenced Series', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Differenced Value')
axes[1].axhline(y=0, color='black', linestyle='--', alpha=0.5)
mplot.tight_layout()
mplot.show()

is_diff_stationary = adf_test(df_diff)

fig, axes = mplot.subplots(1, 2, figsize=(15, 5))

plot_acf(df_diff, lags=40, ax=axes[0], alpha=0.05)
axes[0].set_title('ACF of Differenced Series', fontsize=14, fontweight='bold')
plot_pacf(df_diff, lags=40, ax=axes[1], alpha=0.05)
axes[1].set_title('PACF of Differenced Series', fontsize=14, fontweight='bold')
mplot.tight_layout()
mplot.show()


acf_values = acf(df_diff, nlags=20)
pacf_values = pacf(df_diff, nlags=20)

models = [
    (1, 1, 0),
    (2, 1, 0),
    (0, 1, 1),
    (1, 1, 1),
    (2, 1, 1),
    (3, 1, 0),
    (0, 1, 2),
    (1, 1, 2),
]

results = []
for order in models:
    model = ARIMA(df['Value'], order=order)
    fit_model = model.fit()
    results.append({
        'Order': order,
        'AIC': fit_model.aic,
        'BIC': fit_model.bic,
        'Model': fit_model
```

```
    })

results_df = pd.DataFrame(results).sort_values('AIC')

best_model = results_df.iloc[0]['Model']
best_order = results_df.iloc[0]['Order']


n_periods = 20
forecast_result = best_model.forecast(steps=n_periods)
forecast_index = pd.date_range(start=df.index[-1] + pd.DateOffset(months=1),
                               periods=n_periods, freq='MS')

forecast_df = pd.DataFrame({
    'forecast': forecast_result,
}, index=forecast_index)

mplot.figure(figsize=(15, 8))

mplot.plot(df.index, df['Value'], label='Historical Data', color='darkblue', linewidth=1.5)
mplot.plot(forecast_df.index, forecast_df['forecast'], label='Forecast',
        color='red', linewidth=2, linestyle='--')

mplot.title(f'Electric Production Forecast - ARIMA{best_order}', fontsize=16, fontweight='bold')
mplot.xlabel('Date', fontsize=12)
mplot.ylabel('Production Value', fontsize=12)
mplot.legend(loc='upper left', fontsize=12)
mplot.grid(True, alpha=0.3)

mplot.axvline(x=df.index[-1], color='gray', linestyle=':', alpha=0.7)
mplot.text(df.index[-1], mplot.ylim()[1]*0.95, 'Forecast Start',
        rotation=90, verticalalignment='top')

mplot.tight_layout()
mplot.show()


auto_model = auto_arima(df['Value'],
                        start_p=0, start_q=0,
                        max_p=5, max_q=5,
                        seasonal=False,
                        stepwise=True,
                        suppress_warnings=True,
                        trace=False)

auto_forecast = auto_model.predict(n_periods=n_periods)
auto_forecast_df = pd.DataFrame({
    'auto_forecast': auto_forecast
}, index=forecast_index)

mplot.figure(figsize=(15, 8))

mplot.plot(df.index[-100:], df['Value'][-100:], label='Historical Data (last 100 points)',
        color='darkblue', linewidth=1.5)

mplot.plot(forecast_df.index, forecast_df['forecast'],
        label=f'Manual ARIMA{best_order}', color='red', linewidth=2)

mplot.plot(auto_forecast_df.index, auto_forecast_df['auto_forecast'],
```

```
        label=f'Auto ARIMA{auto_model.order}', color='green', linewidth=2, linestyle='--')

mplot.title('Forecast Comparison: Manual vs Auto ARIMA', fontsize=16, fontweight='bold')
mplot.xlabel('Date', fontsize=12)
mplot.ylabel('Production Value', fontsize=12)
mplot.legend(loc='upper left', fontsize=12)
mplot.grid(True, alpha=0.3)
mplot.tight_layout()
mplot.show()

forecast_comparison = pd.DataFrame({
    'Manual_ARIMA': forecast_df['forecast'],
    'Auto_ARIMA': auto_forecast_df['auto_forecast'],
    'Difference': forecast_df['forecast'] - auto_forecast_df['auto_forecast']
})

forecast_df.to_csv('electric_production_forecasts.csv')

###Exercise 2

import numpy as np
import matplotlib.pyplot as mplot
from statsmodels.tsa.stattools import pacf

np.random.seed(23)
n = 200
phi = np.array([1, -0.9])
x = np.zeros(n)
e = np.random.normal(0, 1, n)
for t in range(2, n):
    x[t] = phi[0] * x[t - 1] + phi[1] * x[t - 2] + e[t]

def sample_autocov(x, h):
    n = len(x)
    m = np.mean(x)
    return np.sum((x[:n - h] - m) * (x[h:] - m)) / n

max_lag = 20
gamma = np.array([sample_autocov(x, h) for h in range(max_lag + 1)])

def durbin_levinson(g, m):
    p = np.zeros((m + 1, m + 1))
    v = np.zeros(m + 1)
    p[1, 1] = g[1] / g[0]
    v[1] = g[0] * (1 - p[1, 1]**2)

    for i in range(2, m + 1):
        s = sum(p[i - 1, k] * g[i - k] for k in range(1, i))
        p[i, i] = (g[i] - s) / v[i - 1]

        for k in range(1, i):
            p[i, k] = p[i - 1, k] - p[i, i] * p[i - 1, i - k]
        v[i] = v[i - 1] * (1 - p[i, i]**2)

    return [p[i, i] for i in range(1, m + 1)]

dl_pacf = durbin_levinson(gamma, max_lag)
lib_pacf = pacf(x, nlags=max_lag)[1:]
```

```
mplot.figure(figsize=(10, 6))
lags = np.arange(1, max_lag + 1)
mplot.plot(lags, dl_pacf, 'o-', label='Durbin-Levinson')
mplot.plot(lags, lib_pacf, 'x--', label='Built-in PACF')
mplot.axhline(0, color='black', lw=0.5)
mplot.xlabel('Lag')
mplot.ylabel('PACF')
mplot.title('Durbin-Levinson vs Built-in PACF')
mplot.legend()
mplot.grid(True)
mplot.tight_layout()
mplot.show()
```

Listing 1: Python code for loading and preparing the dataset

References

- Statistics for Data Science, Marina Anca Cidota, Faculty of Mathematics and Computer Science, University of Bucharest
- Hyndman, R.J., & Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3rd ed.). OTexts. https://otexts.com/fpp3/
- Box, G.E.P., Jenkins, G.M., Reinsel, G.C., & Ljung, G.M. (2015). *Time Series Analysis: Forecasting and Control* (5th ed.). Wiley.
- DevGenius. (n.d.). *Finding optimal p, d, and q values for your ARIMA model*. Medium. https://blog.devgenius.io/finding-optimal-p-d-and-q-values-for-your-arima-model-94669a909
- Baeldung. (n.d.). *Understanding ACF and PACF in ARMA Modeling*. https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling
- Kaggle. (n.d.). *Electric Production Dataset*. https://www.kaggle.com/datasets/kandij/electric-production