Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

Time Series Forecasting of Monthly Electricity Production Using Classical and Regression-Based Models Group-5

Department of Software Engineering
Üsküdar Üniversitesi

Instructor: Dr. Öğr. Üyesi SALIM JIBRIN DANBATTA

GROUP MEMBERS:

- Muhammed Şuayb (210209337)
- Ahmet Can Çağlar (200209055)
- Rümeysa Ensari (210209040)
- Melek Ceylan (200209033)
- MDAGHRI Soufiane (200209397)
- Sevgi Sude UYSAL (220209804)
- Ibrahim Yaseen (200209998)

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

Table of Contents

Chapters	Description	Marks
Chapter 1: Introduction	**1.1 Problem Statement** • Define the objective (e.g., "Forecast monthly sales using historical data"). **1.2 Dataset Overview** • Source, variables, time range, and relevance. **1.3 Key Questions to Answer** • Example: "Does the data show seasonality? Which model performs best?"	• **Problem Statement (5 Marks):** Clear definition of objectives and relevance • **Dataset Overview (5 Marks):** Description of source, variables, and time period
Chapter 2: Data Collection & Preprocessing	**2.1 Data Loading** • Code snippet to load data (e.g., pd.read_csv()). **2.2 Data Cleaning** • Handling missing values, outliers, duplicates. **2.3 Feature Engineering** • Create time-based features (e.g., Month, Day_of_Week). **2.4 Data Splitting** • Train-test split (e.g., 80-20) for model evaluation.	• **Data Loading (5 Marks)**: Proper import and initial inspection • **Data Cleaning (10 Marks)**: Handling missing values, outliers, and duplicates
Chapter 3: Exploratory Data Analysis (EDA)	**3.1 Descriptive Statistics** • Mean, median, variance, skewness. **3.2 Time Series Decomposition** • Trend, seasonality, and residual analysis. **3.3 Visualizations** *** Line Plot** (Trend over time). *** Histogram/Box Plot** (Distribution of target variable). *** Correlation Heatmap** (Feature relationships).	• **Descriptive Statistics (5 Marks)**: Mean, variance, skewness, etc. • **Time Series Decomposition (5 Marks)**: Trend, seasonality, residuals • **Visualizations (5 Marks)**:
Chapter 4:Modeling	**4.1 Model 1: ARIMA** • **Model Explanation** (p, d, q parameters). • **Implementation** (Code + hyperparameter tuning). • **Forecast Visualization** (Plot predictions vs. actual). **4.2 Model 2: Regression-Based** • **Linear Regression** (Trend modeling). • **Polynomial Regression** (Nonlinear trends). • **Feature Importance** (Coefficient analysis). **4.3 Model 3: Fourier Series** Detect seasonality frequencies.	• **Model 1 Implementation (10 Marks)**: ARIMA (code + hyperparameter tuning) • **Model 2 Implementation (10 Marks)**: Regression (linear/polynomial/forier/etc)
Chapter 5: Model Evaluation & Comparison	**5.1 Evaluation Metrics** • **RMSE** (Root Mean Squared Error). • **MAPE** (Mean Absolute Percentage Error). • **R^2** (R-squared). Example of evaluation results:	• **Metrics Calculation (5 Marks)**: RMSE, MAPE, R^2 for both models • **Results Comparison (5 Marks)**: Table + critical discussion of performance
Chapter 6: Conclusion & Recommendations	**6.1 Key Findings** • Summarize trends, seasonality, and model performance. **6.2 Business/Research Implications** • How can results drive decisions? **6.3 Future Work** • Suggestions for improvement (e.g., try SARIMA, add external variables).	**5 marks**
Presentation	Presentation schedule will be provided later on.	**25 marks**

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

Chapter 1: Introduction

1.1 Problem Statement

Electricity production forecasting is vital for the energy sector: Utility companies forecast to plan for generation capacity to optimize resource allocation and ensure the reliability of the grid. Production forecasts are used to optimize generation planning, ensure resource efficiency, and prevent overproduction or shortfalls. The consideration of **SARIMA** is to improve the models as it takes care of seasonality and periodic variation in electricity production, which are common in such time series.

The aim was to forecast future monthly electricity production using historical data from January **1985** to January **2018**. To forecast and identify underlying patterns, classical time series methods such as **ARIMA**, **SARIMA**, and **Linear** and **Polynomial** Regression are implemented.

The goal of this project is to develop effective forecasting models that will assist energy providers, policymakers, and businesses to develop strategies that affect electricity production, supply management, and sustainable long-term development.

1.2 Dataset Overview

Source: The dataset used in this project is sourced from **Kaggle's** time series repository and can be accessed at the following link: Kaggle Time Series Datasets.

1- **Variables**: The dataset contains two primary variables:

- **EP(renamed from IPG2211A2N):** This variable represents electricity Production; the unit is not given. It is assumed that the data are recorded on a monthly basis. For ease and clarity in this analysis, the original variable **IPG2211A2N** has been renamed EP.
- **DATE:** This variable shows a date at which the production was captured every month from January **1985** until January **2018**.

2- **Time Range**: The data covers **33 years**, from January **1985** through January **2018**. The records exist for each month. This way, each value stands for a particular month-over-3-decades timeframe-detailed electricity production span.

3- **Relevance**: The electricity production data is highly relevant for forecasting future **output**, resource allocation optimization, and infrastructure development in the energy-sector. Profiling the trends, seasonality, and fluctuations in production is crucial for power generation planning and infrastructure development. This dataset becomes especially useful in a time series model, as it provides the historical data-base for predicting future production patterns.

1.3 Key Questions to Answer

1. Does the data show seasonality?
   One of the main questions is whether the variables of monthly production of electricity behave according to a seasonal pattern. This analysis can be made by performing time series decomposition to uncover trends, seasonality, and residuals.

2. What is the trend in electricity production over time?

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

Analyzing long-term trends in data implies understanding whether electricity **production** has been rising in the past years or has been going down. This question will determine the choice of the model and will help us understand whether seasonal adjustments such as SARIMA are needed.

3. Which forecasting model most accurately predicts electricity production trends?
   Performance comparison among different models can be carried out, including ARIMA, SARIMA, and Linear/Polynomial Regression, in terms of the evaluation metrics, which are RMSE, MAPE, and R-squared. The aim is to decide which model offers the most accurate and reliable forecast.

4. How well do the chosen models capture the seasonality and trends in the data?
   This question is intended to assess whether each model, especially the SARIMA model, can capture seasonal fluctuations and long-term trends so that its forecasts can genuinely reflect the underlying patterns of electricity production.

5. What are the implications of the forecasting results for future electricity production planning and resource allocation?

From forecasting electricity production, this question considers how the utility companies and policy makers can apply the findings in optimizing resource allocation and ensuring stability in power supplies.

Chapter 2: Data Collection & Preprocessing

2.1 Data Loading

The dataset is loaded using Pandas, which is a popular Python library for data manipulation and analysis. The following code snippet loads the dataset and performs initial inspection.

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import pmdarima as pm
from pmdarima.arima import auto_arima
import warnings
warnings.filterwarnings("ignore")
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_percentage_error, mean_s
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
# Dataset Path
file_path="C:\\Users\\moham\\Desktop\\Electric_Production.csv"
# Load dataset
df = pd.read_csv(file_path)
# Convert DATE column to datetime and set it as index
df['DATE'] = pd.to_datetime(df['DATE'])
df.set_index('DATE', inplace=True)
# Keep only the electricity production column
df = df['EP']
# Preview the data
df.head()
print(df.info()) # Check for basic information about the dataset (
```

1. **Libraries**: We start by importing the necessary libraries for data analysis, including **Pandas** for data handling, **Matplotlib** and **Seaborn** for visualization, and **statsmodels** and **pmdarima** for time series modeling.
2. **Loading the Dataset**: The dataset is loaded from the provided file path using **pd.read_csv()**. After that, we ensure that the **'DATE'** column is parsed as a datetime object using **pd.to_datetime()**, and we set it as the index of the DataFrame using **set_index()**.
3. **Data Preview**: The **head()** method is used to display the first few rows of the data to confirm the correct loading of the dataset.
4. **Dataset Information**: The **info()** method is used to check the dataset's structure (number of rows, columns, and data types) to ensure that there are no issues with the data types.

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

Output:

```
DATE
1985-01-01    72.5052
1985-02-01    70.6720
1985-03-01    62.4502
1985-04-01    57.4714
1985-05-01    55.3151
Name: EP, dtype: float64
<class 'pandas.core.series.Series'>
DatetimeIndex: 397 entries, 1985-01-01 to 2018-01-01
Series name: EP
Non-Null Count  Dtype
--------------  -----
397 non-null    float64
dtypes: float64(1)
memory usage: 6.2 KB
None
```

First Few Rows: After loading, the first few rows will be displayed, allowing us to confirm that the dataset looks as expected.

Dataset Info: This will show the columns, their data types, and the number of non-null entries for each column. This is crucial for identifying missing values or potential data issues.

2.2 Data Cleaning

In this section, we focus on cleaning the dataset by addressing missing values, outliers, and duplicate entries.

2.2.1. Handling Missing Values:

The first step is to check if there are any missing values in the dataset. Since time series data is sensitive to missing values, they need to be handled properly to ensure accurate analysis and forecasting.

Output: The dataset has **0 missing values**, as indicated by the output from the **isnull().sum()** method.

2.2.2. Handling Outliers:

Outliers can significantly distort statistical analysis and model predictions. Therefore, it is essential to detect and remove outliers. We use the **Interquartile Range (IQR)** method to identify outliers. Any values outside the range of 1.5 * IQR from the first (Q1) and third quartile (Q3) are considered outliers.

Output: The number of **outliers removed** is **0**, meaning there were no data points outside the defined range.

2.2.3. Handling Duplicates:

It is important to ensure that the dataset does not contain duplicate entries, as they could lead to incorrect analyses and forecasts. We check for duplicates using the **duplicated()** method and remove any duplicate rows.

Output: The dataset has **0 duplicate rows**, so no duplicates were removed.

```
Number of missing values: 0
Number of outliers removed: 0
Number of duplicate rows: 0
```

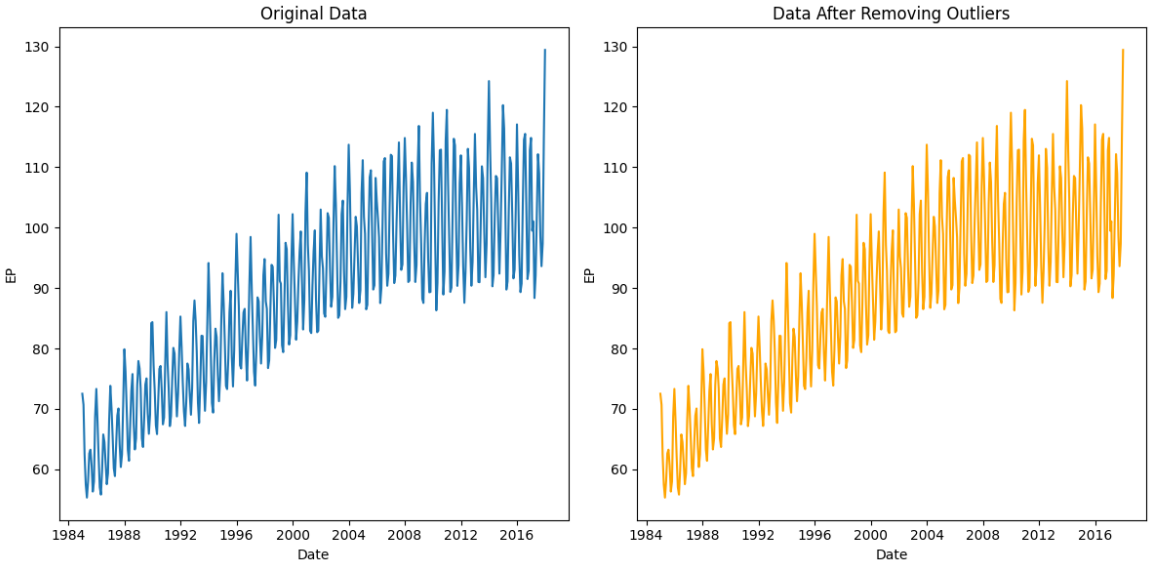
```python
print(f"Number of missing values: {df.isnull().sum()}")
#---- Handling Outliers ----#
# Calculate IQR (Interquartile Range) for outlier detection
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
# Define the outlier boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
# Filter out outliers
df_clean = df[(df >= lower_bound) & (df <= upper_bound)]
# Show the number of outliers removed
outliers_removed = len(df) - len(df_clean)
print(f'Number of outliers removed: {outliers_removed}')
######################################################################
# Check for duplicate rows in the dataset
duplicate_rows = df.duplicated().sum()
# Print the message along with the number of duplicate rows
print(f"Number of duplicate rows: {duplicate_rows}")
# Remove duplicate rows if any
df = df.drop_duplicates()
# Compare the original and cleaned data visually
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(df, label='Original Data')
plt.title('Original Data')
plt.xlabel('Date')
plt.ylabel('EP')
plt.subplot(1, 2, 2)
plt.plot(df_clean, label='Cleaned Data', color='orange')
plt.title('Data After Removing Outliers')
plt.xlabel('Date')
plt.ylabel('EP')
plt.tight_layout()
```

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

2.2.4. Data Visualization:

To visually inspect the cleaned data, we plot both the original and cleaned datasets side by side. This allows us to confirm that no significant changes occurred due to the cleaning process, especially regarding outliers.

- **Visual Output**: The left plot shows the **original data**, and the right plot shows the **cleaned data** (after outlier removal). As no outliers were detected, both plots should appear almost identical.



Summary of Data Cleaning Steps:

- **Missing Values**: There were no missing values in the dataset.
- **Outliers**: No outliers were found and removed, as the data points remained within the acceptable bounds.
- **Duplicates**: No duplicate rows were found, ensuring that the dataset is unique and free of redundant entries.

2.3 Feature Engineering

Since our original dataset includes only a single variable, electricity production (EP). We derived additional time-based features from the datetime index to support time-aware modeling, particularly for regression-based models.

These engineered features help the model better understand temporal dynamics such as trends and seasonality:

- **Month**: Numeric month (1–12), useful for capturing seasonal energy **production** patterns.
- **Year**: Year of observation, which helps model long-term trends over decades.
- **Quarter**: Fiscal quarter (1–4), relevant for capturing periodic changes across seasons.
- **TimeIndex**: A sequential index (starting at 0) representing the passage of time; useful for modeling linear or nonlinear trends.

- **Month_sin / Month_cos**: Cyclical transformations of the Month variable to represent seasonal patterns in a way that reflects the continuous cycle of the calendar year (e.g., December is close to January).

```
# Time-based feature extraction
df_reg['Month'] = df_reg.index.month
df_reg['Year'] = df_reg.index.year
df_reg['Quarter'] = df_reg.index.quarter
df_reg['TimeIndex'] = np.arange(len(df_reg)) # Trend feature (e.g., simple index over time)
# Cyclical encoding of Month
df_reg['Month_sin'] = np.sin(2 * np.pi * df_reg['Month'] / 12)
df_reg['Month_cos'] = np.cos(2 * np.pi * df_reg['Month'] / 12)
```

These features are particularly important for regression models, which depend on structured input variables to identify and learn relationships within the data.

2.3.1 Preview of Engineered Features

The table below shows the first few rows after feature engineering:

DATE	EP	Month	Year	Quarter	TimeIndex	Month_sin	Month_cos
1985-01-01	72.5052	1	1985	1	0	0.500000	8.660254e-01
1985-02-01	70.6720	2	1985	1	1	0.866025	5.000000e-01
1985-03-01	62.4502	3	1985	1	2	1.000000	6.123234e-17
1985-04-01	57.4714	4	1985	2	3	0.866025	-5.000000e-01
1985-05-01	55.3151	5	1985	2	4	0.500000	-8.660254e-01

2.4 Data Splitting

To evaluate model performance on unseen data, we split the dataset into **training** and **testing** subsets using an **80-20 time-based split**. Since time series data must retain temporal order, we avoided random shuffling to preserve the sequence of observations.

- **Training Set**: 80% of the data from January 1985 up to ~mid 2009.
- **Testing Set**: Remaining 20%, covering ~mid 2009 to January 2018.

```
# Define the split point (80% train, 20% test)
split_index = int(len(df) * 0.80)

# Split the data for ARIMA (train-test)
train_arima = diff_df.iloc[:split_index]
test_arima = diff_df.iloc[split_index:]
```

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

Chapter 3: Exploratory Data Analysis (EDA)

3.1 Descriptive Statistics

To begin our exploratory analysis, we compute key descriptive statistics for the electricity production (EP) data. These metrics offer insight into the distribution, variability, and shape of the time series.

```python
# Descriptive statistics
desc_stats = df.describe()
skewness = df.skew()
kurtosis = df.kurt()

# Display all together
print("Descriptive Statistics:")
print(desc_stats)
print("\nSkewness:", skewness)
print("Kurtosis:", kurtosis)
```

```
Descriptive Statistics:
count    397.000000
mean      88.847218
std       15.387834
min       55.315100
25%       77.105200
50%       89.779500
75%      100.524400
max      129.404800
Name: EP, dtype: float64

Skewness: -0.07309562446039364
Kurtosis: -0.6942008809310036
```

3.1.1 Interpretation:

- The **mean (~88.85)** and **median (~89.78)** are very close, indicating a nearly symmetric distribution.
- The **standard deviation (~15.39)** shows moderate variability in electricity production over time.
- The **slightly negative skewness (-0.07)** implies a very mild left tail, but close to symmetrical overall.
- **Kurtosis (-0.69)** suggests a slightly flatter distribution than a normal curve (platykurtic), meaning fewer extreme outliers.

These descriptive measures suggest that the EP data is relatively well-behaved, with no strong skew or heavy-tailed behavior, which supports the use of models that assume near-normality in residuals.

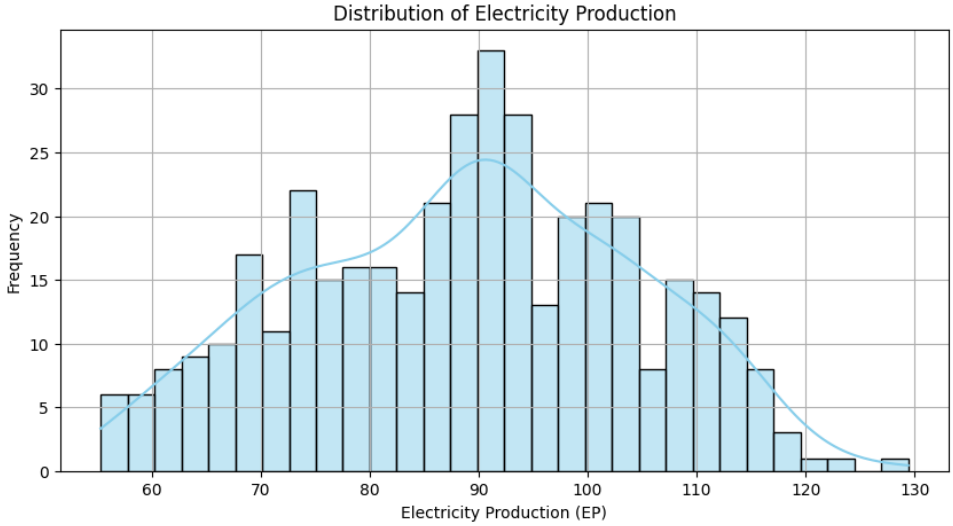
In addition to numerical summaries, visualizations help in understanding the distribution, central tendency, and variability of the data.

- **Histogram with KDE**: Shows the frequency distribution and smooth density estimate of the electricity production values. This helps detect skewness or multimodality.
- **Box Plot**: Highlights the spread, median, and any potential outliers in the data.
- **Rolling Statistics Plot**: Displays the 12-month rolling mean and standard deviation to observe how the average and variability evolve over time.

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

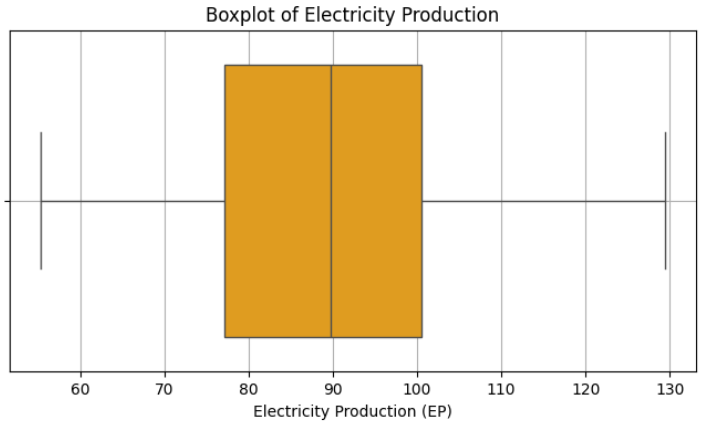
These visual tools complement the numerical statistics and provide a clearer picture of the data's behavior.

1- Histogram with KDE: The histogram combined with a Kernel Density Estimation (KDE) curve gives us insight into the shape and spread of the data. It helps detect skewness, peaks, and overall distribution trends in electricity production.



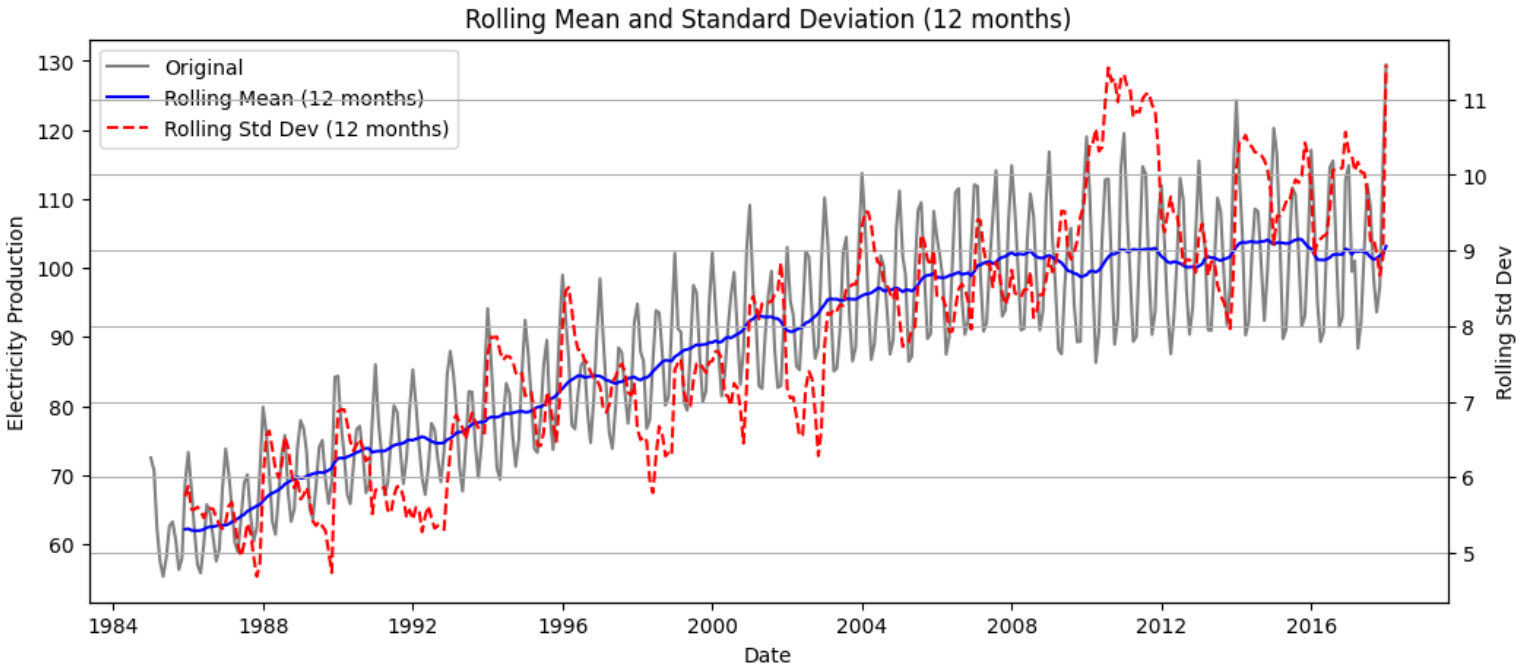
2-Box Plot:

The box plot visualizes the distribution, median, interquartile range (IQR), and potential outliers in the electricity production data. It's useful for quickly spotting any anomalies or extreme values.



3-Rolling Mean and Standard Deviation

The rolling plot (with a 12-month window) displays the moving average and moving standard deviation over time. It helps to detect long-term trends and structural shifts in the data's variability.



Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

3.2 Time Series Decomposition

To better understand the underlying patterns in the electricity production data, we applied **seasonal decomposition** using an **additive model**. This allows us to break the series into three distinct components:

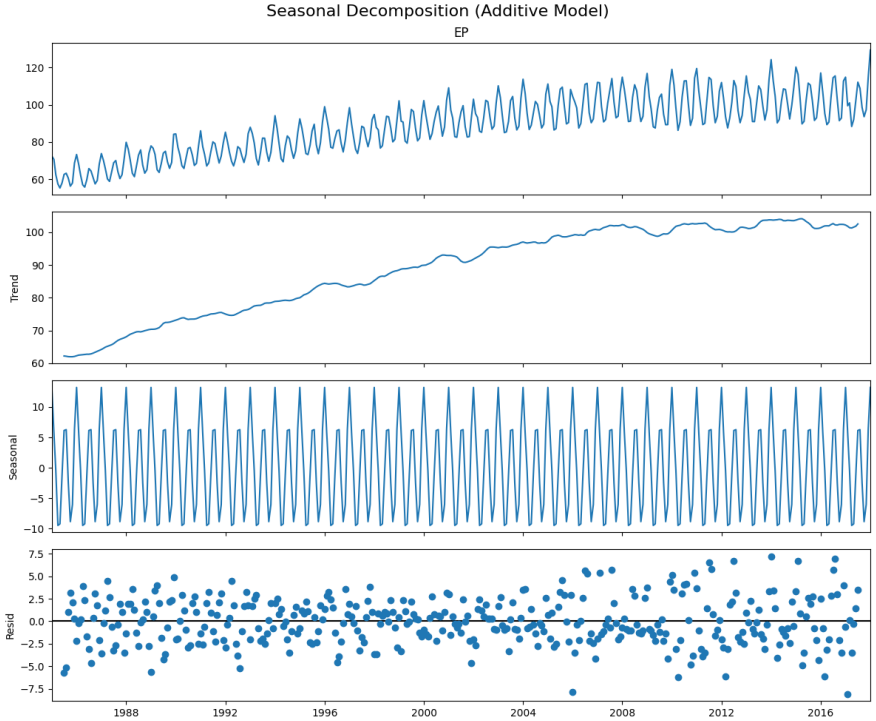
- **Trend**: The long-term progression of the series.
- **Seasonality**: Repeating short-term cycle in the data.
- **Residuals**: The noise or random variation not explained by trend or seasonality.

```
# Perform additive decomposition
decomposition = seasonal_decompose(df, model='additive', period=12)

# Plot the decomposition
plt.rcParams.update({'figure.figsize': (12, 10)})
decomposition.plot()
plt.suptitle('Seasonal Decomposition (Additive Model)', fontsize=16)
plt.tight_layout()
plt.show()
```

3.2.1 Interpretation

Trend Component: The trend shows a **clear long-term increase** in electricity production from 1985 to 2017. There is a noticeable flattening of the trend between 2008 and 2012, which may reflect a temporary slowdown in electricity production before the upward trajectory resumes.

Seasonality Component: The seasonal component displays a consistent, repeating pattern every year, with regular peaks and troughs, indicating cyclical production variations.



Residuals (Noise): The residuals are **mostly random and centered around zero**, indicating that the decomposition model effectively captured both the trend and seasonality. However, there are **notable spikes in the residuals around 2008–2012**, which could point to **unexpected variations or external events** affecting electricity production during that period.

3.3 Visualizations

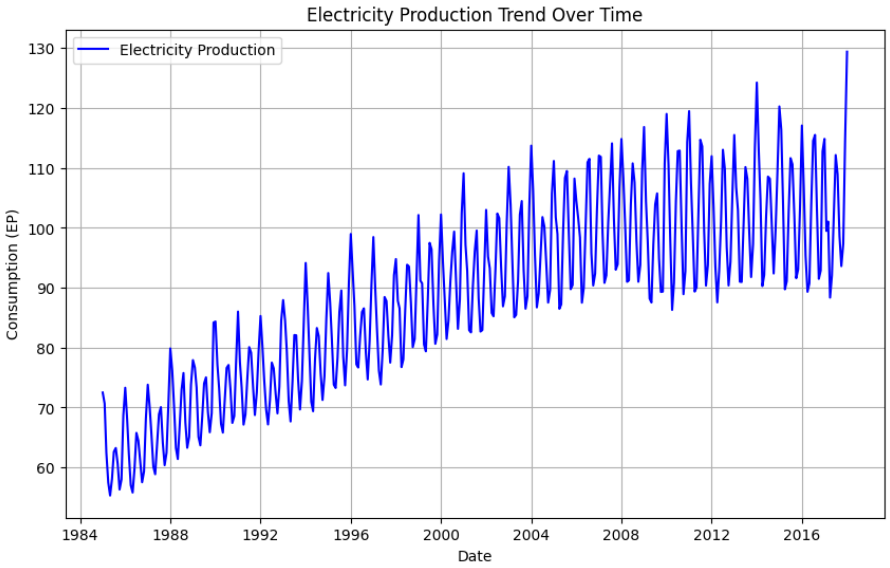
To explore the structure, distribution, and feature relationships of electricity production over time, we created the following visualizations:

3.3.1. Line Plot: Electricity Production Over Time

- **General Trend:**
  The electricity production data exhibits a clear upward trend from 1985 to 2018. While short-term fluctuations occur, the long-term movement is predominantly increasing.

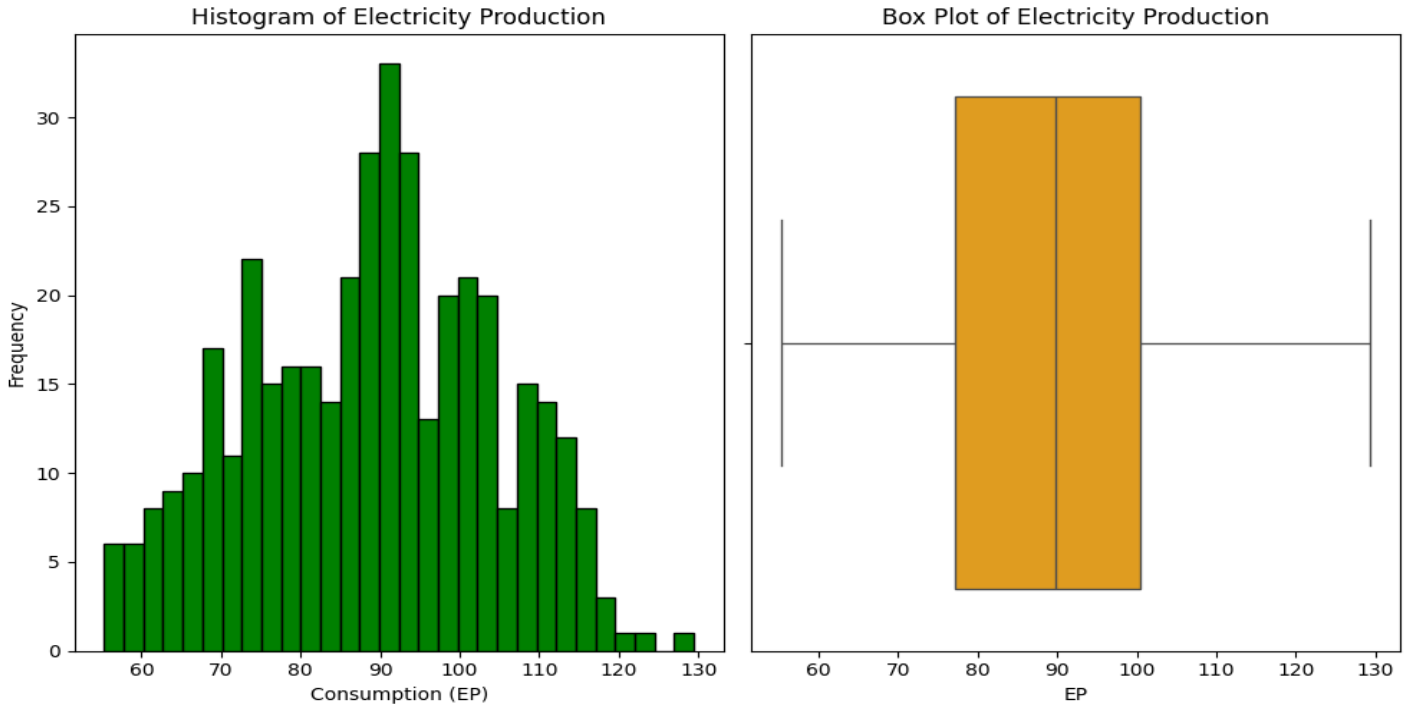
Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

- **Seasonality:**
  Distinct seasonal cycles are visible, with repeating peaks and troughs each year, suggesting consistent seasonal patterns in electricity production.
- **Anomalies and Disruptions:**
  - A notable spike occurs between 2017 and 2018, where values jump to ~130 EP—much higher than prior years.
  - A period of flattened growth is observed between 2008 and 2012, possibly due to economic or policy shifts.



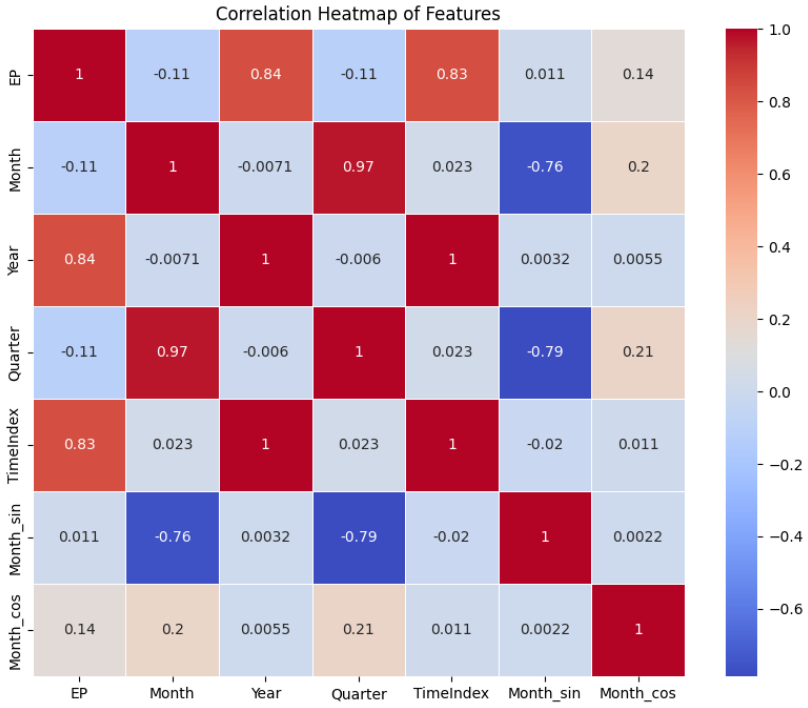
3.3.2. Histogram and Box Plot: Distribution of EP

- **Shape of Distribution:**
  The histogram shows a distribution close to normal, slightly left-skewed (negative skew), with the bulk of values concentrated near the mean.
- **Outliers (Box Plot):**
  The box plot reveals no strong outliers, with all data points falling within the whiskers. This indicates a stable distribution.
- **Most Frequent Values:**
  The most common EP values lie between 80 and 100, peaking around 90.



Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

3.3.3. Correlation Heatmap: Feature Relationships



- **Strongest Correlations with EP:**

Year (0.84) and TimeIndex (0.83) are strongly positively correlated with EP, confirming a clear upward trend over time.

- **Seasonality Features:**

Month_cos shows a weak positive correlation (0.14), while Month_sin has minimal impact (0.01), suggesting limited value in modeling seasonality directly through cyclic encoding.

- **Other Observations:**

Slight negative correlations for Month and Quarter (each -0.11) imply seasonal fluctuations but not dominant seasonal effects.

- **Conclusion:**
  The trend-related features (especially TimeIndex) contribute significantly to modeling electricity production, while seasonal features offer limited but supporting insights.

Chapter 4: Modeling

4.1 Model 1: ARIMA & SARIMA

4.1.1 Model Explanation (p, d, q, P, D, Q, s Parameters)

The AutoRegressive Integrated Moving Average (ARIMA) model is a classical time-series modeling technique designed to capture linear dependencies in sequential data. It models both autoregressive (AR) and moving average (MA) structures, with differencing applied to achieve stationarity if required. However, standard ARIMA models are limited in capturing seasonal patterns. To overcome this, the Seasonal ARIMA (SARIMA) model extends ARIMA by including seasonal components.

- **ARIMA Parameters (p=1, d=0, q=2):**
  - **p (autoregressive order):** Incorporates lagged values of the target variable.
  - **d (differencing order):** Applied to make the series stationary; ADF test results suggested differencing was not required (d=0).
  - **q (moving average order):** Models the relationship with past forecast errors.
- **SARIMA Parameters (P=0, D=1, Q=1, s=12):**
  - **P, D, Q:** Seasonal analogues of the ARIMA terms.
  - **s (seasonal period):** Set to 12, reflecting annual seasonality in monthly data.
  - Seasonal differencing (D=1) was applied to remove periodic trends.

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

The AutoARIMA function was employed to automatically identify optimal parameters by minimizing the AIC (Akaike Information Criterion). It selected:

- **ARIMA(1,0,2)** with AIC = **1359.590**
- **SARIMA(1,0,2)(0,1,1)[12]**, which further improved seasonal accuracy

Implementation and Hyperparameter Tuning

1- Data Preprocessing:

- Stationarity was assessed using the Augmented Dickey-Fuller (ADF) test.
- Seasonal differencing was performed for SARIMA.
- The data was split into **80% training** and **20% testing** for model evaluation.

2- Model Training:

- Parameters were selected using AutoARIMA with stepwise optimization.
- Both ARIMA and SARIMA models were implemented using the statsmodels library.
- Forecasts were generated and inverse transformations applied to restore the original scale.

4.1.1 Auto Model (Code+Output)

```python
# Define the split point (80% train, 20% test)
split_index = int(len(df) * 0.80)

# Split the data for ARIMA (train-test)
train_arima = diff_df.iloc[:split_index]
test_arima = diff_df.iloc[split_index:]


# AutoARIMA to find the best ARIMA parameters
auto_arima_model = pm.auto_arima(train_arima, seasonal=True, m=12, stepwise=True, trace=Tru


# Get the best ARIMA parameters from AutoARIMA
print("Best ARIMA parameters from AutoARIMA:")
print(auto_arima_model.summary())

# Extract the ARIMA parameters
p, d, q = auto_arima_model.order  # (p, d, q)
P, D, Q, s = auto_arima_model.seasonal_order  # (P, D, Q, s)

# Display the selected parameters
print(f"ARIMA(p, d, q) = ({p}, {d}, {q})")
print(f"SARIMA(P, D, Q, s) = ({P}, {D}, {Q}, {s})")
```

```
 ARIMA(2,0,1)(0,1,1)[12] intercept   : AIC=1360.469, Time=0.83 sec
 ARIMA(2,0,3)(0,1,1)[12] intercept   : AIC=1363.589, Time=1.37 sec
 ARIMA(1,0,2)(0,1,1)[12]             : AIC=1360.532, Time=0.40 sec

Best model:  ARIMA(1,0,2)(0,1,1)[12] intercept
Total fit time: 25.735 seconds
Best ARIMA parameters from AutoARIMA:
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:                  317
Model:             SARIMAX(1, 0, 2)x(0, 1, [1], 12)   Log Likelihood                -673.795
Date:                              Tue, 06 May 2025   AIC                           1359.590
Time:                                      02:55:54   BIC                           1381.911
Sample:                                  02-01-1985   HQIC                          1368.518
                                       - 06-01-2011
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0039      0.002     -2.164      0.030      -0.007      -0.000
ar.L1          0.3420      0.112      3.066      0.002       0.123       0.561
ma.L1         -0.7481      0.113     -6.648      0.000      -0.969      -0.528
ma.L2         -0.2172      0.105     -2.076      0.038      -0.422      -0.012
ma.S.L12      -0.7183      0.046    -15.749      0.000      -0.808      -0.629
sigma2         4.6684      0.299     15.608      0.000       4.082       5.255
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                42.42
Prob(Q):                              0.86   Prob(JB):                         0.00
Heteroskedasticity (H):               2.51   Skew:                            -0.28
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.74
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
ARIMA(p, d, q) = (1, 0, 2)
SARIMA(P, D, Q, s) = (0, 1, 1, 12)
```

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

2- ARIMA Code:

```
# Fit ARIMA model with parameters from AutoARIMA
arima_model = ARIMA(train_arima, order=(p, d, q))
arima_fit = arima_model.fit()

# Forecast the future values
arima_forecast = arima_fit.forecast(steps=len(test_arima))

# Plot the forecasted vs actual values
plt.figure(figsize=(10, 6))
plt.plot(test_arima.index, test_arima, label='Actual', color='blue')
plt.plot(test_arima.index, arima_forecast, label='ARIMA Forecast', color='red')
plt.legend()
plt.title('ARIMA Forecast vs Actual')
plt.show()
```

3- SARIMA Code:

```
# Fit SARIMA model with parameters from AutoARIMA
sarima_model = SARIMAX(train_arima,
                       order=(p,d, q),  # Non-seasonal ARIMA parameters
                       seasonal_order=(P, D, Q, s))  # Seasonal parameters (P, D, Q, s)

sarima_fit = sarima_model.fit(disp=False)

# Forecast the future values
sarima_forecast = sarima_fit.forecast(steps=len(test_arima))

# Plot the forecasted vs actual values
plt.figure(figsize=(10, 6))
plt.plot(test_arima.index, test_arima, label='Actual', color='blue')
plt.plot(test_arima.index, sarima_forecast, label='SARIMA Forecast', color='green')
plt.legend()
plt.title('SARIMA Forecast vs Actual')
plt.show()
```

Residual Diagnostics:

After fitting the final model using **AutoARIMA** (ARIMA(1,0,2)(0,1,1)[12]), we conducted a residual diagnostics analysis to evaluate the model's assumptions and forecast reliability. The following diagnostic plots were generated:

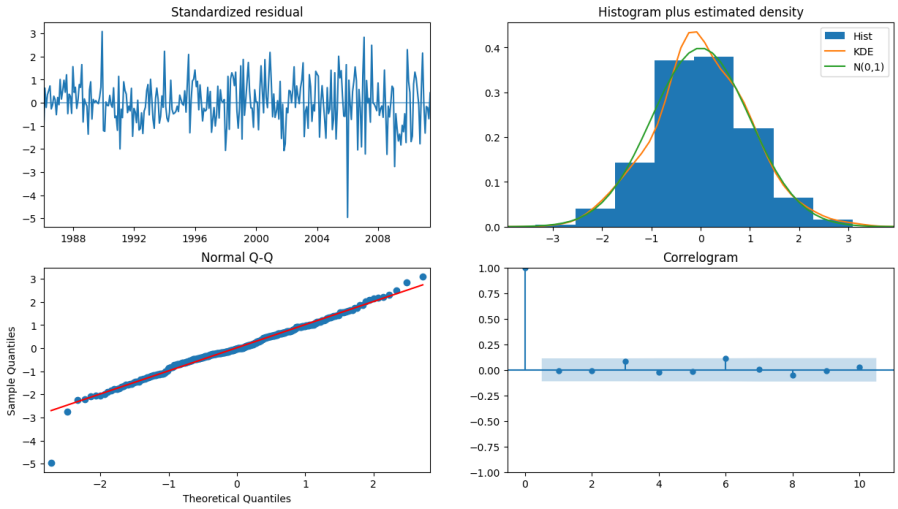
1. Standardized Residuals (Top Left)

- Residuals oscillate around zero, indicating the model's predictions are generally **unbiased**.

- Occasional spikes suggest **temporary model misfit**, possibly during unusual events or structural changes in data.

2. Histogram + Density Estimate (Top Right)

- Residuals exhibit a **roughly normal distribution**, as seen from the bell-shaped curve.
- KDE and normal distribution curve align closely, suggesting **error terms are approximately normally distributed**, satisfying ARIMA assumptions.



3. Normal Q-Q Plot (Bottom Left)

- Most points lie close to the reference line, further supporting the **normality assumption**.
- Slight deviations at the tails hint at a few **outliers**, but overall the fit remains acceptable.

4. Correlogram / ACF Plot of Residuals (Bottom Right)

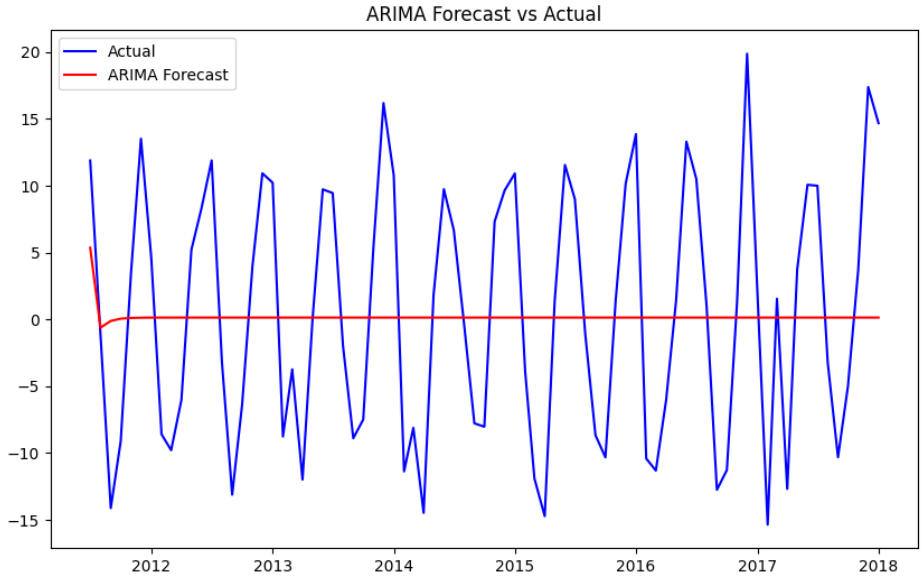
- ACF values fall within the **95% confidence intervals**, indicating **no significant autocorrelation**.
- This implies that the model has successfully captured the underlying structure of the time series, leaving behind **white noise residuals**.

Forecast Visualization (Predictions vs. Actual)

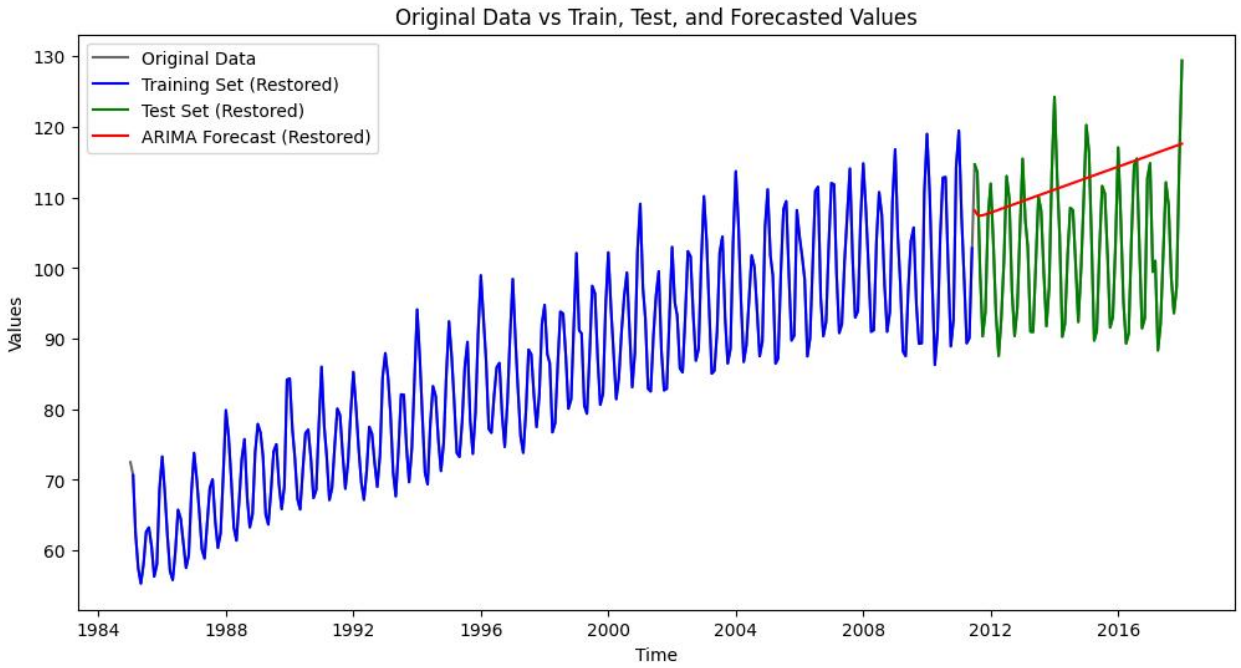
Visual comparisons were made between actual electricity production and forecasts from both ARIMA and SARIMA models:

ARIMA Forecast:

The ARIMA model successfully captured the overall upward trend in electricity production, aligning well with the direction of the test data. However, it struggled to replicate short-term, seasonal fluctuations, resulting in deviations during months with high or low production. This is expected, as ARIMA does not explicitly model seasonality.

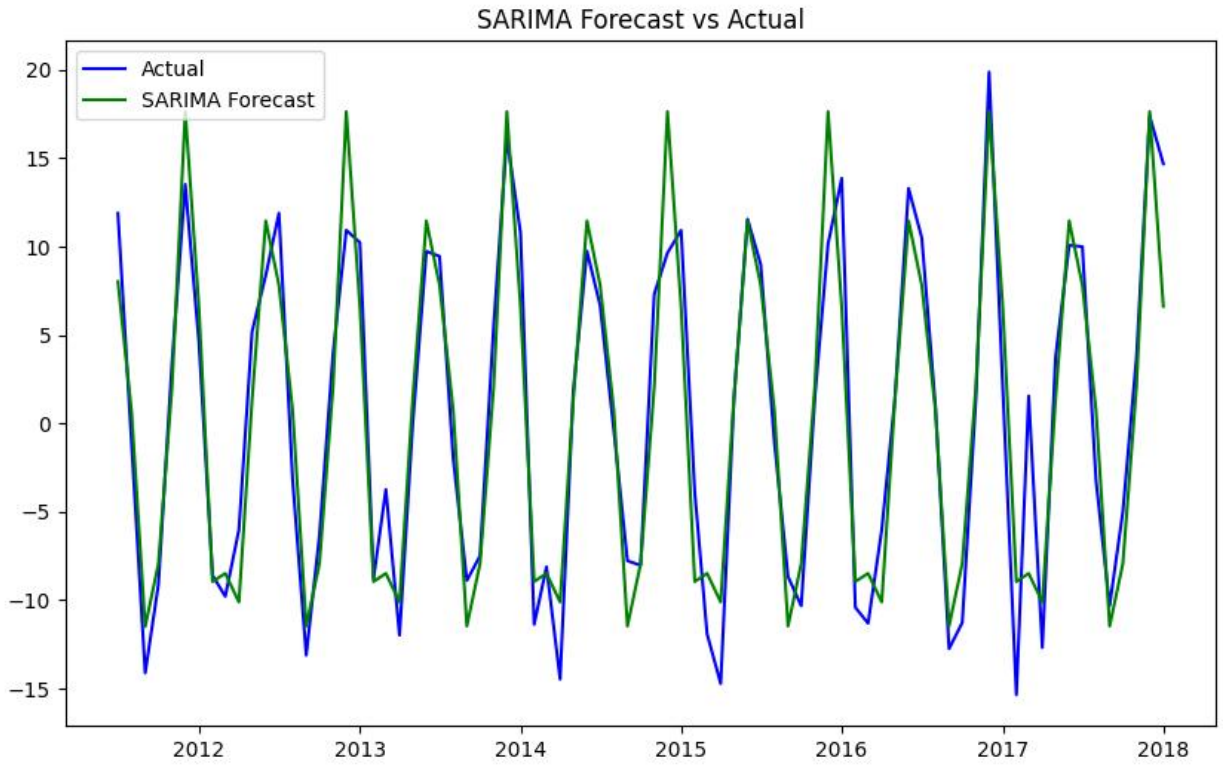


ARIMA works well for long-term trend forecasting but **misses cyclical behavior**, making it less ideal for data with strong seasonal components.



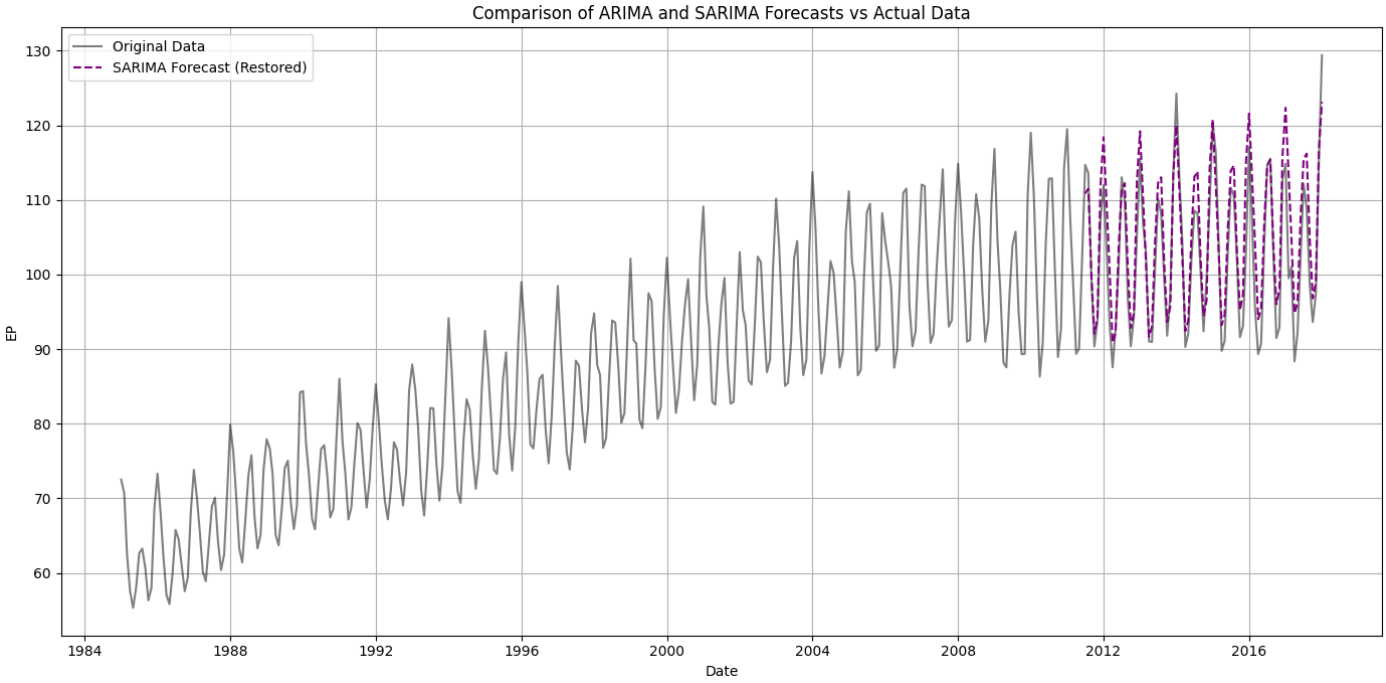
2- SARIMA Forecast:

The SARIMA model showed strong alignment with the test data, especially in capturing recurring seasonal peaks and troughs. It produced a smoother and more realistic forecast, maintaining consistency even in extended 24-month forecasts. Minor forecast drift and residual noise suggest that additional tuning could further



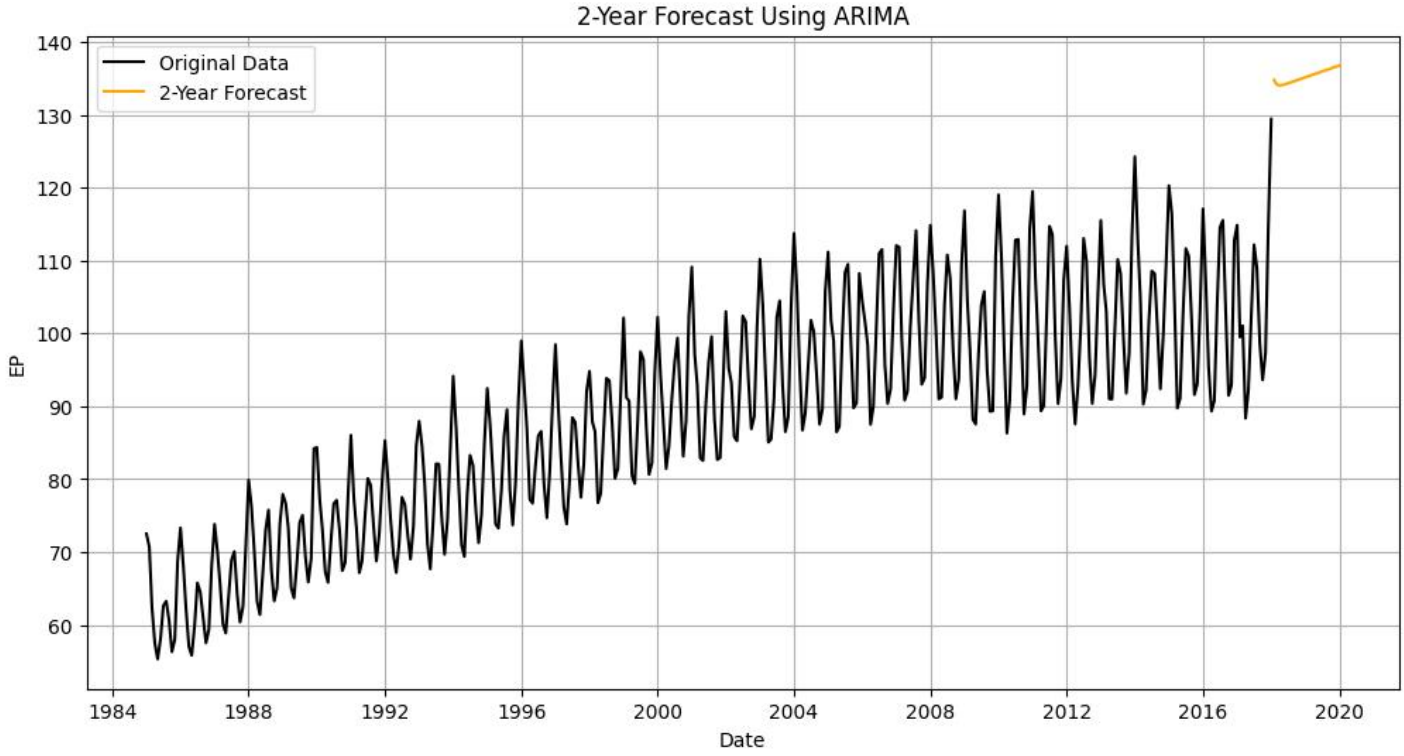
Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

improve accuracy. SARIMA offers **superior predictive performance** for this dataset, effectively modeling both **trend and seasonality**, which is crucial for energy production forecasting.



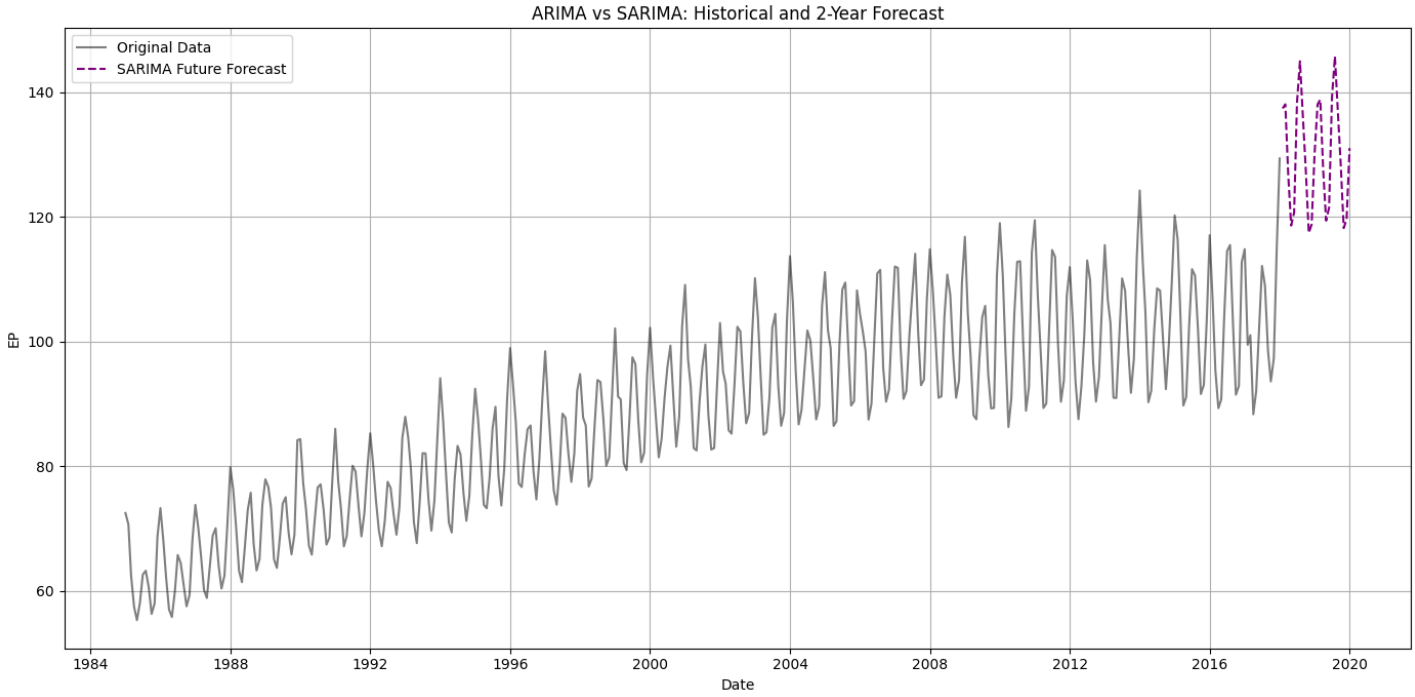
Next 2-Years Forecast:

ARIMA:



Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

SARIMA:



4.2 Model 2: Regression-Based

Regression models are widely used to analyze and predict relationships between variables. In this case, we explore **Linear Regression** and **Polynomial Regression** to model the trend and nonlinear patterns in electricity production (EP). Additionally, we perform **Feature Importance Analysis** to assess how each feature contributes to the prediction. The regression models aim to capture the trend and seasonal effects (to some extent) of electricity production using various features, including lagged values and time-based indicators.

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics

4.2.1 Linear Regression (Trend Modeling)

Model Implementation

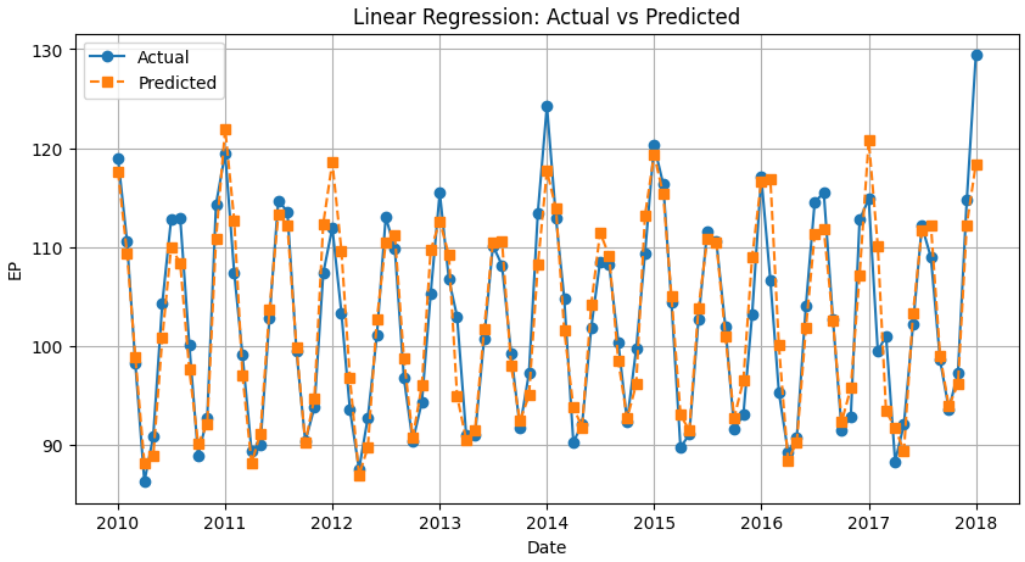
Linear Regression aims to model the relationship between the target variable (electricity production, EP) and a set of independent variables (features). The features we used for this model include:

- Time-based features: TimeIndex, Month, Quarter, Month_sin, and Month_cos (sinusoidal transformations to capture cyclical effects).
- Lagged values of the target variable (EP_lag_1 to EP_lag_12), which represent past values of electricity production for the last 12 months.

```python
# Generate lag features before splitting
for lag in range(1, 13):  # 12 months
    df_reg[f'EP_lag_{lag}'] = df_reg['EP'].shift(lag)
# Drop NaN values (first 12 rows will have missing values)
df_reg.dropna(inplace=True)
# Proceed with train-test split
train_size = int(len(df_reg) * 0.75)
train_reg = df_reg.iloc[:train_size]
test_reg = df_reg.iloc[train_size:]
# Define features (include lag variables now)
features = ['TimeIndex', 'Month',
            'Quarter', 'Month_sin',
            'Month_cos']
            + [f'EP_lag_{lag}' for lag in range(1, 13)]

target = 'EP'
# Prepare training and testing sets
X_train = train_reg[features]
y_train = train_reg[target]
X_test = test_reg[features]
y_test = test_reg[target]

# Fit the Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predict on the test set
y_pred = lin_reg.predict(X_test)
```

Forecast Visualization & Residuals (Actual vs Predicted)

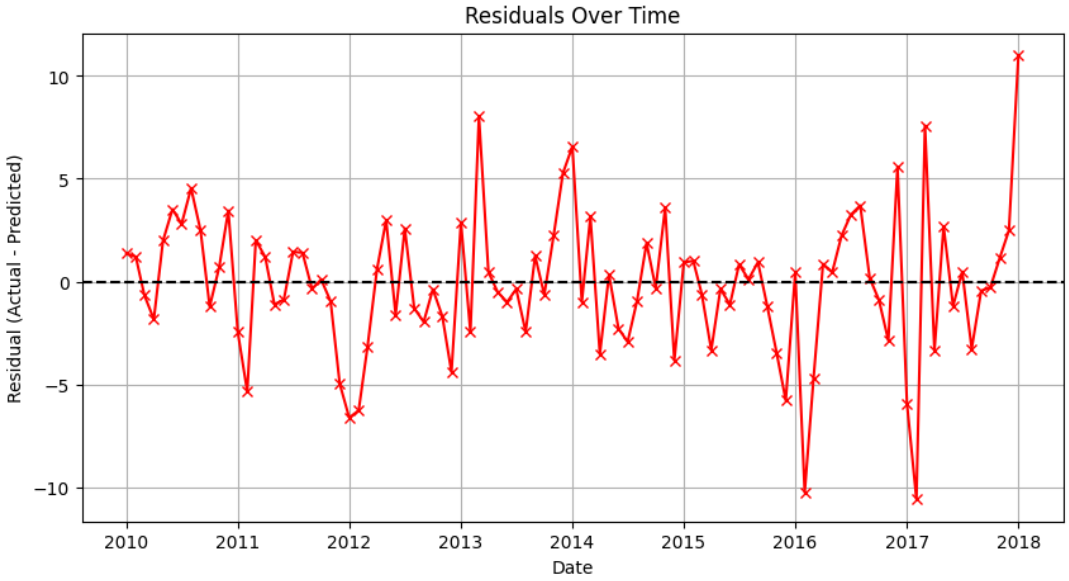
The forecasted values and actual values are compared to visualize how well the model captures the trend of electricity production. The plot below shows the actual vs. predicted values, where the Orange dashed line represents the predicted values, and the blue line represents the actual test set data.



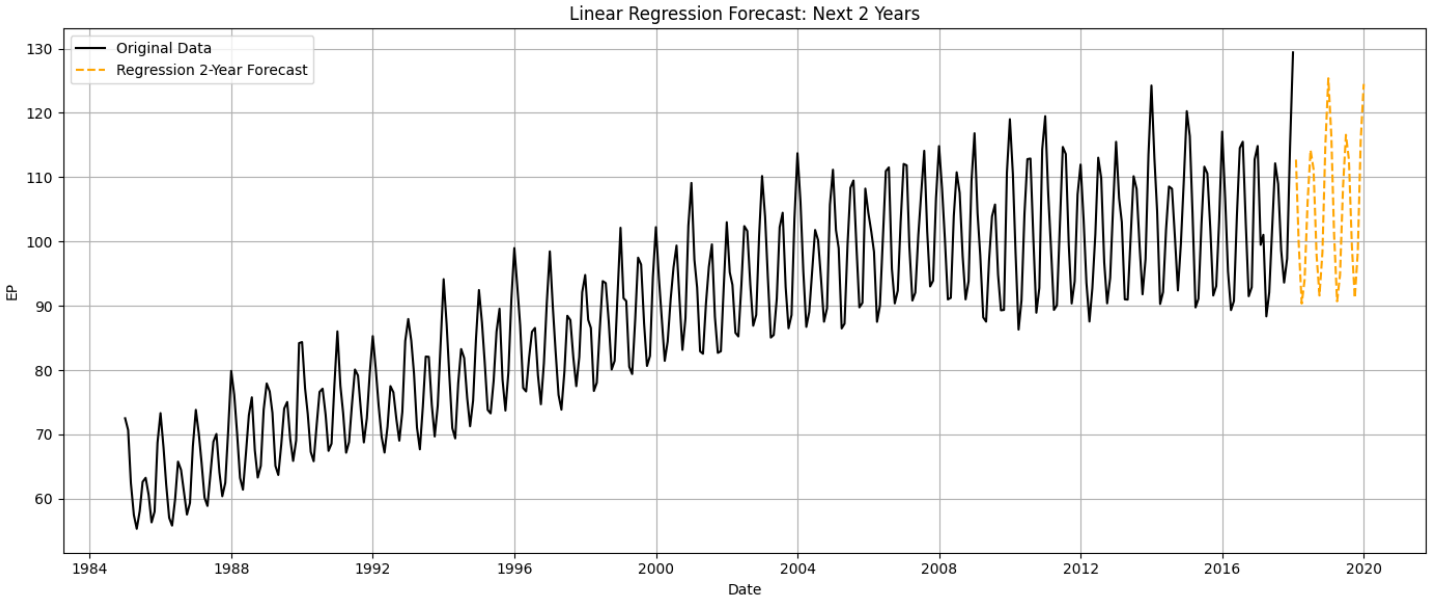
Residual Analysis:

Residuals, which are the differences between the actual and predicted values, are plotted to evaluate the performance of the model. The residuals should be randomly scattered around zero for an ideal model, indicating that the model has successfully captured the underlying patterns.

Üsküdar Üniversitesi
Software Engineering Department
Data Science and Analytics



Next 2 year forecast:



4.2.2 Polynomial Regression (Nonlinear Trends)

Model Implementation

Polynomial Regression is an extension of Linear Regression that allows for modeling nonlinear relationships by adding polynomial terms of the independent variables. In this case, a **degree 2 polynomial** was used to capture more complex patterns in the data. This model is more flexible than linear regression and can better fit data that has nonlinear trends.

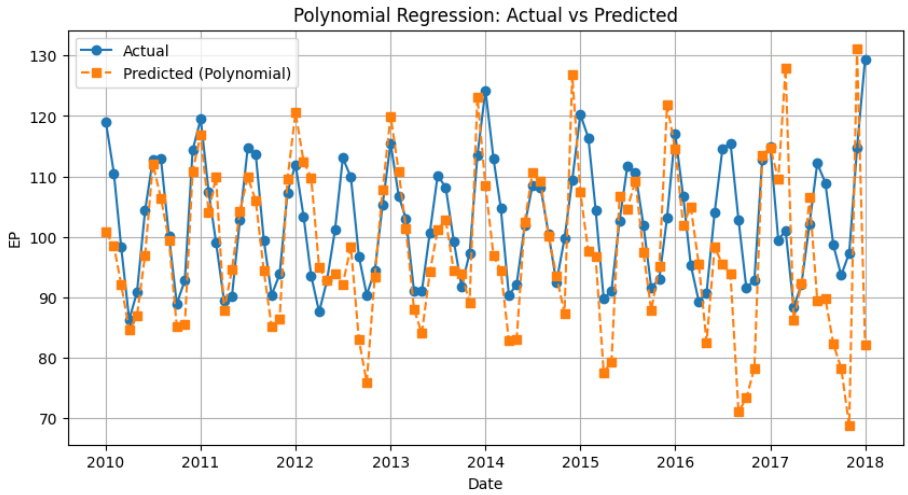
We use a pipeline that combines **PolynomialFeatures** (to transform the

```
# Separate features and target
X_train = train_reg[features]
X_test = test_reg[features]
y_train = train_reg[df.name]
y_test = test_reg[df.name]
# Create pipeline for polynomial regression (degree 2)
poly_model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
# Fit the model
poly_model.fit(X_train, y_train)

# Predict
y_pred_poly = poly_model.predict(X_test)
```

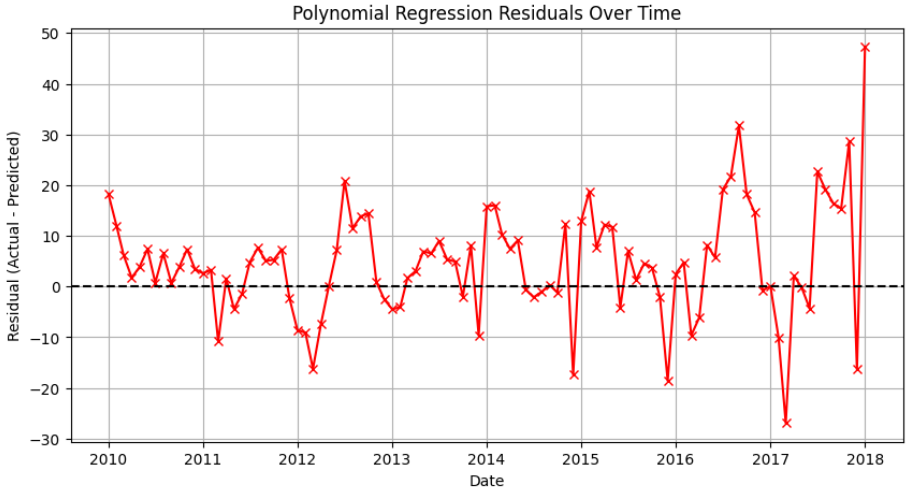
features) and **LinearRegression** (to fit the transformed features).

Forecast Visualization & Residuals (Actual vs Predicted)

The plot below compares the actual values (y_test) to the predicted values (y_pred_poly) from the polynomial regression model. The **red dashed line** represents the predicted values, and the **blue line** shows the actual test set data.



Residual Analysis:
The residuals for polynomial regression are plotted to check if the model has captured all significant patterns. The plot shows the differences between the actual and predicted values over time.



4.2.3 Feature Importance (Coefficient Analysis)

After fitting the polynomial regression model, we analyze the **coefficients** of the features to understand their importance in predicting electricity production.

The table below shows the coefficients for each feature, including the **intercept**. Larger absolute values of coefficients indicate more important features for the prediction.

```python
# Extract the model from the pipeline
poly_features = poly_model.named_steps['polynomialfeatures']
lin_reg = poly_model.named_steps['linearregression']

# Get feature names
feature_names = poly_features.get_feature_names_out(input_features=features)
# Create a DataFrame to view coefficients
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': lin_reg.coef_
})

# Add intercept separately
intercept_df = pd.DataFrame([['Intercept', lin_reg.intercept_]], columns=coef_df.columns)

# Combine and display
coef_df = pd.concat([intercept_df, coef_df], ignore_index=True)
coef_df
```

	Feature	Coefficient
0	Intercept	-1.041811e+01
1	1	3.010144e-08
2	TimeIndex	-2.944341e-01
3	Month	8.014779e+00
4	Quarter	-1.744417e+01
...	...	...
167	EP_lag_10 EP_lag_11	4.259485e-02
168	EP_lag_10 EP_lag_12	1.869685e-02
169	EP_lag_11^2	-4.873756e-02
170	EP_lag_11 EP_lag_12	1.068598e-02
171	EP_lag_12^2	-8.651453e-03

172 rows × 2 columns

Chapter 5: Model Evaluation & Comparison

The evaluation and comparison of all forecasting models developed in this study are discussed in this chapter. The performance of each of the models under consideration was measured using the standard forecasting metrics: RMSE (Root Mean Squared Error), MAPE (Mean Absolute Percentage Error), and R^2 (R-squared). Thus, the measures considered concern accuracy, relative error, and goodness-of-fit.

5.1 Evaluation Metrics

Model	MPE	MAPE	RMSE	R^2
ARIMA	102.27%	102.27%	9.38	0.014
SARIMA	58.69%	92.36%	3.45	0.867
Linear Regression	-0.20%	2.41%	3.43	0.878
Polynomial Regression	4.37%	8.5%	11.89	-0.467

Metric Descriptions:

- **RMSE**: Penalizes larger errors more severely. Lower RMSE indicates better performance.
- **MAPE**: Expresses prediction error as a percentage. Values closer to 0% are better.
- **R^2 (R-squared)**: Measures how well the model explains variance in the data. Closer to 1 indicates a better fit.

Critical Discussion of Performance

1. ARIMA

- **MAPE** over 100% and a near-zero **R^2** indicate **very poor predictive accuracy**.
- High RMSE suggests the model fails to follow the actual EP trend.
- The lack of seasonality modeling in ARIMA is likely a major limitation for this dataset.

2. SARIMA

- Significant improvement over ARIMA, especially in **RMSE** and **R^2** (0.867).
- However, **MAPE (92%)** is still high, indicating that while the model fits the trend well, it struggles with capturing the **scale or magnitude** of fluctuations.
- SARIMA's incorporation of **seasonality** is clearly beneficial.

3. Linear Regression

- Demonstrates strong performance with low MAPE (2.41%), low RMSE (3.43), and high R^2 (0.878).
- Outperforms SARIMA slightly in MAPE and R^2, suggesting good generalization and predictive power, leveraging trend, seasonality (via sin/cos), and lag features effectively.
- Serves as a strong baseline model, balancing simplicity with solid predictive accuracy.

4. Polynomial Regression

- Despite its theoretical flexibility, this model performs poorly in this case.
- High RMSE (11.89) and negative R^2 (-4.37) indicate overfitting or unnecessary complexity.
- The model fails to generalize well with unseen data, suggesting polynomial expansion adds unnecessary complexity rather than improving forecasting accuracy.

Conclusion

- **Best Performing Model**: **Linear Regression** is simple, interpretable, and robust, it achieved the **lowest error and highest R^2**.
- **Second Best**: **SARIMA** which performed well on trend/seasonal structure but lagged in relative accuracy (MAPE).
- **Worst Performing Models**: **ARIMA** (due to missing seasonality) and **Polynomial Regression** (likely overfitting).

These results emphasize the effectiveness of feature-engineered regression models in time series forecasting, especially when domain-specific variables (lags, time indexes, seasonality proxies) are incorporated into these models.

Chapter 6: Conclusion and Recommendations

6.1 Key Findings

The data on electricity generation from January 1985 to January 2018 revealed a strong long-term increase, interrupted by seasonal cycles. Each year exhibited recurring peaks and troughs, which denote strong annual seasonality in electricity generation. The models themselves mimic these trends: the simple ARIMA model, which ignores seasonality, showed very poor performance and failed to accommodate the cyclical swings in the data. On the other hand, the SARIMA model, which explicitly accommodates seasonal terms managed to accommodate the general trend and the recurring seasonal peaks much better, but with fairly high percentage errors. This linear regression model incorporated time features and seasonal indicators and, consistently, the most accurate forecasts were rendered by it: it had the lowest errors as well as the largest R^2, proving that most

of the variation in production is accounted for by it. The polynomial regression model, strangely enough, would overfit historical data and poorly generalize into the test period, thereby giving large errors with a negative R^2. To sum it up, one major finding is that both the upward trend and the repeating seasonal pattern are the main drivers of electricity production; forecasts with the greatest reliability were made by models that explicitly accounted for those components while avoiding excessive model complexity

6.2 Business/Research Implications

Utility companies and policymakers have to act on these forecasting results. For utilities, understanding future electricity production is key for strategic decisions about generation capacity, fuel procurement, and maintenance scheduling. Also, an overall increase in electricity production could help generate ample investment in generation infrastructure and in optimizing fuel supply planning. The ability to forecast seasonal peaks with precision enables utilities to schedule their generation plan accordingly: plan maintenance during low-production periods and allocate additional capacity during peak production. The very high explanatory power of the best model (an R^2 nearing 0.87) would grant utilities firm confidence in these forecasts with respect to load balancing and shortage risk management. This upward trend in production obviously demands a long-term energy policy approach involving the likes of incentives aimed at energy efficiency, integration of renewable energies, or transmission grid upgrades. Understanding the limits to which each model can give accurate predictions (indicated by RMSE and MAPE metric values) informs decision makers of the degree of uncertainty to expect, whether in business or research. This study illustrates that somewhat simple, interpretable models are able to make forecasts with sufficient precision required for the allocation of resources, tariff design, and sustainable development of the power sector.

6.3 Future Work

This research can provide a steppingstone for further studies with the help of more extensive data and advanced modeling techniques for greater forecasting accuracy. One possibility is to integrate exogenous variables affecting electricity production, such as weather conditions (affecting renewable output), fuel price fluctuations, or regulatory changes. Using such external regressors in a model (say through ARIMAX or multivariate regression) may help explain demand shifts that cannot be explained by the historical pattern. Improvement of seasonal modeling comes next in the prioritized list: perhaps the SARIMA parameters could be tuned further, or different seasonal decompositions investigated to reduce forecast errors. In addition to advanced machine learning algorithms worth exploring would be ensemble methods (such as random forests or gradient boosting) and neural network architectures (LSTM or Transformer-based time series) that may capture nonlinearities and interactions missed by linear models. Rigorous validation techniques (such as cross-validation overtime windows) and feature engineering (lagged variables, holiday effects, etc.) would be a definite solution for overfitting that occurred with the polynomial model. Enhancing these aspects, and continuously updating the models with new data, would thus be a future phase in strengthening electricity production forecasts for practical decision-making.