Time Series -2

August 29, 2023

Q1. What is meant by time-dependent seasonal components?

Time-dependent seasonal components refer to recurring patterns in data that occur over regular time intervals, such as daily, weekly, monthly, or yearly cycles. These components capture the systematic variation in data caused by factors like seasonality, holidays, or other recurring events.

The term "time-dependent" indicates that the seasonal patterns are not static but change over time. In other words, the amplitude, phase, or shape of the seasonal pattern may vary from one season to another or from one year to another. This variation could be due to factors like changing consumer behavior, economic conditions, or other external influences.

Time-dependent seasonal components are commonly observed in various fields, including economics, finance, sales forecasting, and weather analysis. By understanding and modeling these components, analysts and statisticians can better interpret data, detect trends, and make accurate predictions or forecasts.

Q2. How can time-dependent seasonal components be identified in time series data?

Identifying time-dependent seasonal components in time series data typically involves analyzing the periodic patterns and fluctuations that occur at regular intervals. Here are some common approaches to identify these components:

1. Visual Inspection: Plot the time series data and visually examine the pattern over time. Look for recurring patterns that repeat at regular intervals, such as daily, weekly, monthly, or yearly cycles. These patterns may indicate the presence of time-dependent seasonal components.
2. Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF): Calculate the ACF and PACF of the time series data. The ACF measures the correlation between the time series observations at different lags, while the PACF measures the correlation between observations after removing the effects of shorter lags. Significant spikes or peaks at specific lags in the ACF and PACF plots may suggest the presence of seasonal components.
3. Seasonal Subseries Plot: Divide the time series data into subseries based on the seasonal cycle (e.g., months, quarters, or weeks). Create separate line plots for each subseries and visually inspect the patterns within each group. If consistent patterns emerge within each subseries, it indicates the presence of time-dependent seasonal components.
4. Decomposition Methods: Apply time series decomposition techniques, such as the additive or multiplicative decomposition. These methods decompose the time series into its trend, seasonal, and residual components. If the seasonal component varies over time, it suggests the presence of time-dependent seasonal components.

5. Time Series Models: Fit time series models that account for time-dependent seasonal components, such as SARIMA (Seasonal Autoregressive Integrated Moving Average) models. These models explicitly model and estimate the seasonal patterns based on the data, allowing for the identification and analysis of time-dependent seasonal components.

It is worth noting that identifying time-dependent seasonal components can sometimes be challenging, especially when the patterns are subtle or influenced by multiple factors. In such cases, a combination of visual inspection, statistical techniques, and domain knowledge can help in effectively identifying and modeling these components in time series data.

Q3. What are the factors that can influence time-dependent seasonal components?

Several factors can influence time-dependent seasonal components in time series data. These factors may vary depending on the specific domain or context of the data. Here are some common factors that can influence time-dependent seasonal components:

1. S1easonal Variations: Natural variations associated with seasons can strongly influence time-dependent seasonal components. For example, in retail, sales may exhibit higher patterns during holiday seasons or summer months compared to other times of the year.
2. Economic Factors: Economic conditions and events can impact seasonal patterns. For instance, consumer spending behavior may vary during different economic cycles, leading to changes in seasonal components. Economic indicators, such as interest rates, inflation, or stock market performance, can affect seasonal patterns in various industries.
3. Calendar Events: Calendar events like holidays, festivals, or specific days of the week can influence time-dependent seasonal components. For example, the sales of certain products may surge during the holiday season or weekends.
4. Weather Conditions: Weather patterns can have a significant impact on certain industries, such as agriculture, tourism, or energy consumption. Seasonal components in these domains can be influenced by weather conditions like temperature, precipitation, or daylight hours.
5. Social and Cultural Factors: Societal and cultural factors can introduce time-dependent seasonal components. For instance, shopping patterns may vary based on cultural celebrations, local events, or social trends.
6. Policy Changes: Policy changes, regulations, or government initiatives can affect seasonal patterns. For example, tax holidays or changes in import/export policies can lead to shifts in consumer behavior and impact seasonal components.
7. Technological Advancements: Technological advancements or innovations can alter seasonal patterns. The introduction of new products, services, or technologies can lead to changes in consumer preferences and behaviors.
8. Demographic Shifts: Changes in population demographics, such as age distribution or migration patterns, can influence seasonal components. These changes can affect demand patterns and consumption behaviors.
9. Competitive Factors: Competitive dynamics within an industry can impact seasonal patterns. Promotions, discounts, or marketing strategies of competitors can influence consumer choices and alter seasonal components.

It's important to note that not all time series data will be influenced by all these factors. The factors influencing time-dependent seasonal components will vary based on the specific context, industry, and nature of the data being analyzed.

Q4. How are autoregression models used in time series analysis and forecasting?

Autoregression models, specifically autoregressive (AR) models, are widely used in time series analysis and forecasting. These models capture the dependence of a variable on its own past values, making them useful for analyzing and predicting the behavior of time series data. Here's how autoregression models are used:

1. Modeling Time Series Data: Autoregressive models are used to understand the underlying patterns and dynamics within a time series. By examining the autocorrelation structure of the data, AR models can identify and quantify the lagged relationships between observations. This helps in understanding the persistence or memory of the time series.

2. Forecasting: Autoregressive models are utilized for making future predictions or forecasts. Once an AR model is fitted to historical data, it can be used to generate forecasts by extending the model into the future. By incorporating the lagged values of the variable, AR models can capture the inherent patterns and trends in the data, making them valuable for short-term or long-term predictions.

3. Model Selection: Autoregressive models provide a framework for model selection in time series analysis. The order of the autoregressive model, denoted as AR(p), determines the number of lagged values considered in the model. The selection of the optimal order involves techniques like analyzing autocorrelation and partial autocorrelation functions, evaluating information criteria (e.g., AIC, BIC), or conducting cross-validation to identify the most suitable model order.

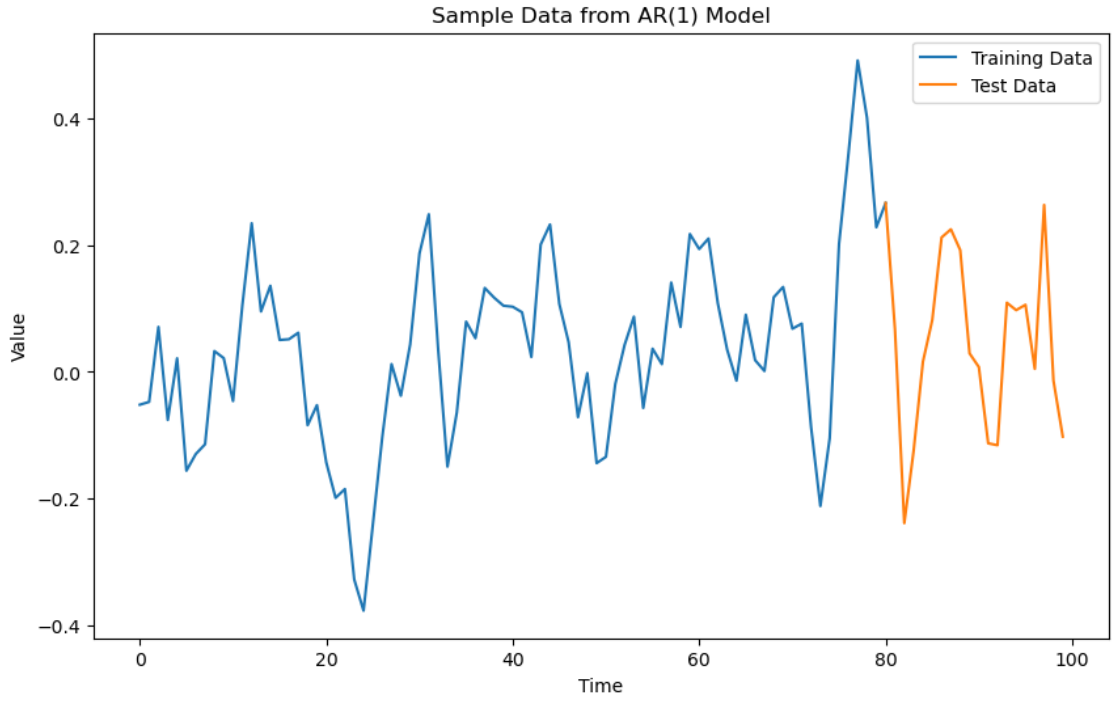
4. Error Analysis: Autoregressive models enable the analysis of residuals or errors, which are the differences between the observed values and the predicted values. By examining the residual patterns, model assumptions, such as independence and homoscedasticity, can be assessed. Deviations from these assumptions can indicate the presence of additional patterns or factors that need to be incorporated into the model.

5. Time Series Decomposition: Autoregressive models are often used as components of more complex time series models, such as SARIMA (Seasonal Autoregressive Integrated Moving Average). SARIMA models incorporate autoregressive components to capture temporal dependencies and combine them with other components like seasonal, trend, and moving average terms to provide a comprehensive representation of the time series.

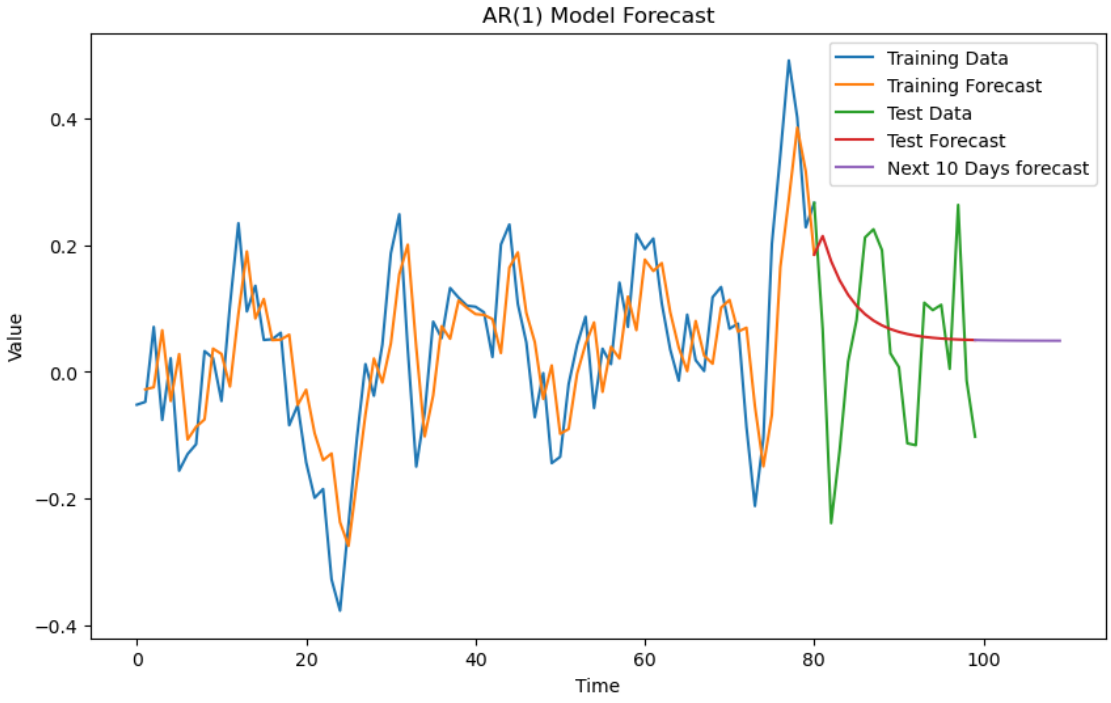
It's important to note that autoregressive models assume that the underlying time series is stationary, meaning that its statistical properties remain constant over time. If the time series exhibits non-stationary behavior, preprocessing techniques like differencing or more advanced models like ARIMA (Autoregressive Integrated Moving Average) may be employed to handle the non-stationarity before applying autoregressive modeling.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
```

```
num_samples = 100  # Number of data points
true_coefficient = 0.7  # True coefficient for the AR(1) model

# Generate sample data for AR(1) model
np.random.seed(21)
ar_data = np.zeros(num_samples)
ar_data[0] = np.random.normal(0, 1)  # Initial value

for i in range(1, num_samples):
    ar_data[i] = true_coefficient * ar_data[i-1] + np.random.normal(0, 0.1)

# Split data into train and test sets
train_data = ar_data[:81]
test_data = ar_data[80:]

# Plot the sample data
plt.figure(figsize=(10, 6))
plt.plot(np.arange(81), train_data, label='Training Data')
plt.plot(np.arange(80, 100), test_data, label='Test Data')
plt.title('Sample Data from AR(1) Model')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```



[2]:
```
from statsmodels.tsa.ar_model import AutoReg

# Train the AR model
model = AutoReg(train_data, lags=1)
model_fit = model.fit()

# Perform the forecast on training data
train_forecast = model_fit.predict(start=0, end=80)

# Perform the forecast on test data
test_forecast = model_fit.predict(start=80, end=99)

# Next 10 days forecast
forecast = model_fit.predict(start=99,end=109)

# Plot the forecasted values
plt.figure(figsize=(10, 6))
plt.plot(np.arange(81), train_data, label='Training Data')
plt.plot(np.arange(81), train_forecast, label='Training Forecast')
plt.plot(np.arange(80, 100), test_data, label='Test Data')
plt.plot(np.arange(80, 100), test_forecast, label='Test Forecast')
plt.plot(np.arange(99,110),forecast,label='Next 10 Days forecast')
plt.title('AR(1) Model Forecast')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```



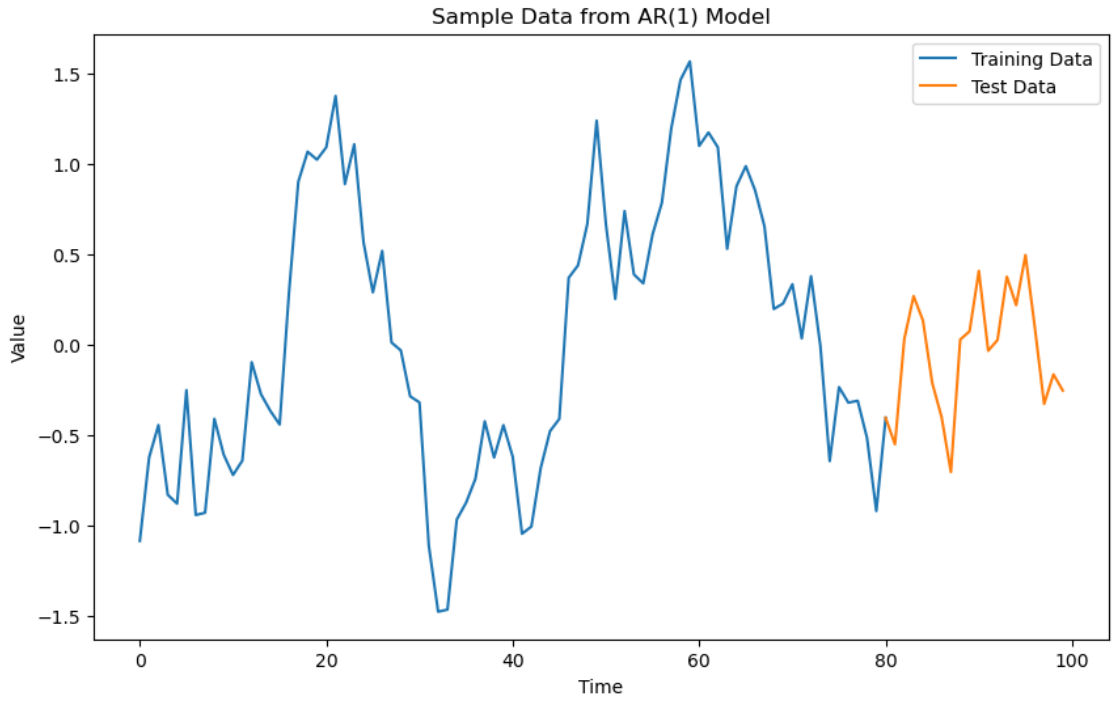
Q5. How do you use autoregression models to make predictions for future time points?

To use autoregression models to make predictions for future time points, you follow these general steps:

1. Train the Autoregression Model: Fit an autoregression model on historical time series data. The order of the autoregression model (e.g., AR(1), AR(2), etc.) determines the number of lagged values considered in the model. The choice of the order depends on the autocorrelation and partial autocorrelation analysis or model selection techniques.
2. Obtain Model Parameters: Once the model is trained, obtain the estimated parameters, including the intercept and coefficients for the lagged values in the autoregression model. These parameters capture the relationship between the current observation and its lagged values.
3. Prepare Input for Prediction: Determine the lagged values of the time series that will be used as input to the autoregression model for prediction. The number of lagged values needed depends on the order of the autoregression model. These lagged values should correspond to the most recent historical data points available.
4. Make Predictions: Use the trained autoregression model and the lagged values of the time series as input to make predictions for future time points. The prediction process involves multiplying the lagged values by their corresponding coefficients and summing them up, including the intercept term.
5. Update Lagged Values: After making a prediction for a future time point, update the lagged values used for prediction by shifting them one step forward. Drop the oldest lagged value and include the newly predicted value as the most recent lagged value.
6. Repeat for Multiple Time Points: Repeat the prediction process for the desired number of future time points, updating the lagged values at each step.

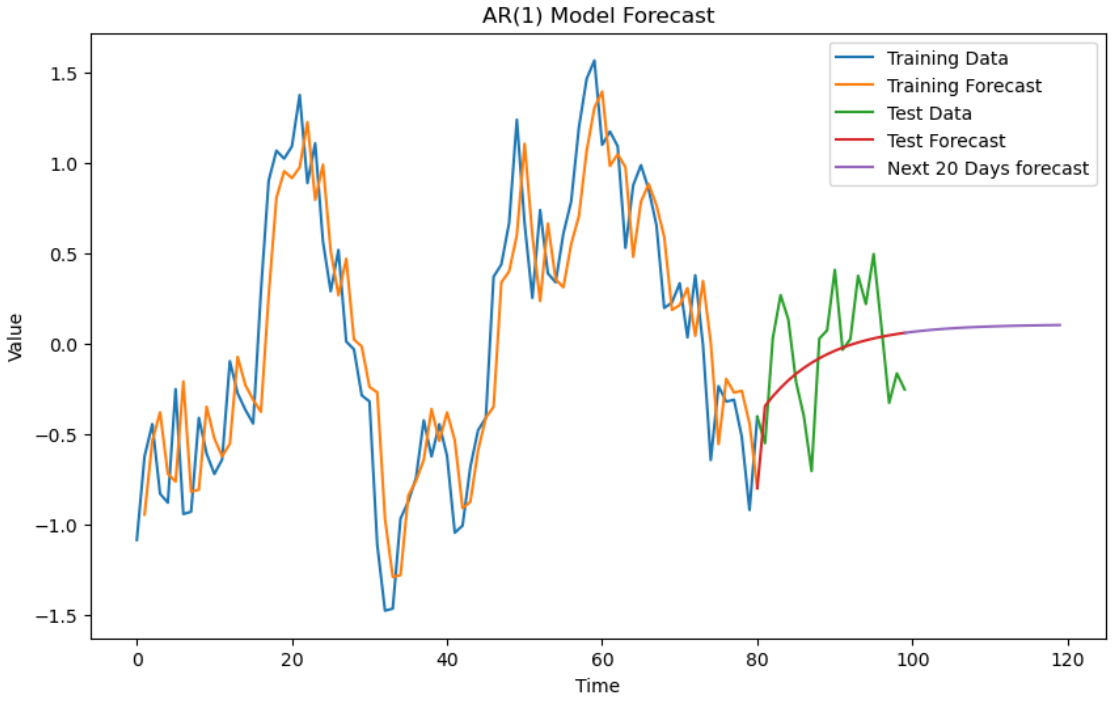
[5]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
num_samples = 100  # Number of data points
true_coefficient = 0.85  # True coefficient for the AR(1) model

# Generate sample data for AR(1) model
np.random.seed(123)
ar_data = np.zeros(num_samples)
ar_data[0] = np.random.normal(0, 1)  # Initial value

for i in range(1, num_samples):
    ar_data[i] = true_coefficient * ar_data[i-1] + np.random.normal(0, 0.3)

# Split data into train and test sets
```

```
train_data = ar_data[:81]
test_data = ar_data[80:]

# Plot the sample data
plt.figure(figsize=(10, 6))
plt.plot(np.arange(81), train_data, label='Training Data')
plt.plot(np.arange(80, 100), test_data, label='Test Data')
plt.title('Sample Data from AR(1) Model')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```



[6]:
```
from statsmodels.tsa.ar_model import AutoReg

# Train the AR model
model = AutoReg(train_data, lags=1)
model_fit = model.fit()

# Perform the forecast on training data
train_forecast = model_fit.predict(start=0, end=80)

# Perform the forecast on test data
test_forecast = model_fit.predict(start=80, end=99)
```

```
# Next 10 days forecast
forecast = model_fit.predict(start=99,end=119)

# Plot the forecasted values
plt.figure(figsize=(10, 6))
plt.plot(np.arange(81), train_data, label='Training Data')
plt.plot(np.arange(81), train_forecast, label='Training Forecast')
plt.plot(np.arange(80, 100), test_data, label='Test Data')
plt.plot(np.arange(80, 100), test_forecast, label='Test Forecast')
plt.plot(np.arange(99,120),forecast,label='Next 20 Days forecast')
plt.title('AR(1) Model Forecast')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```



Q6. What is a moving average (MA) model and how does it differ from other time series models?

A moving average (MA) model is a type of time series model that is used to describe and forecast the behavior of a time series based on the past values and the error terms. In an MA model, the value of the time series at a particular time point is modeled as a linear combination of the error terms at previous time points.

In an MA model, the forecasted value at time t (denoted as Y_t) is calculated as the weighted sum of the error terms at previous time points. The weights assigned to the error terms are called the

model parameters or coefficients.

The key characteristics of an MA model are:

1. Reliance on Error Terms: An MA model assumes that the current value of the time series depends on the error terms at previous time points, rather than on the actual observed values of the time series itself.
2. Finite Order: An MA model is specified by its order, denoted as MA(q), where 'q' represents the number of lagged error terms considered in the model. The order determines the number of coefficients or weights assigned to the error terms.
3. No Autocorrelation: An MA model assumes that there is no autocorrelation or dependence between the error terms at different time points. In other words, the error terms are assumed to be independent and identically distributed.

It is important to note that an MA model is different from other time series models, such as autoregressive (AR) models and autoregressive moving average (ARMA) models, in terms of the underlying assumptions and the way they model the time series behavior.

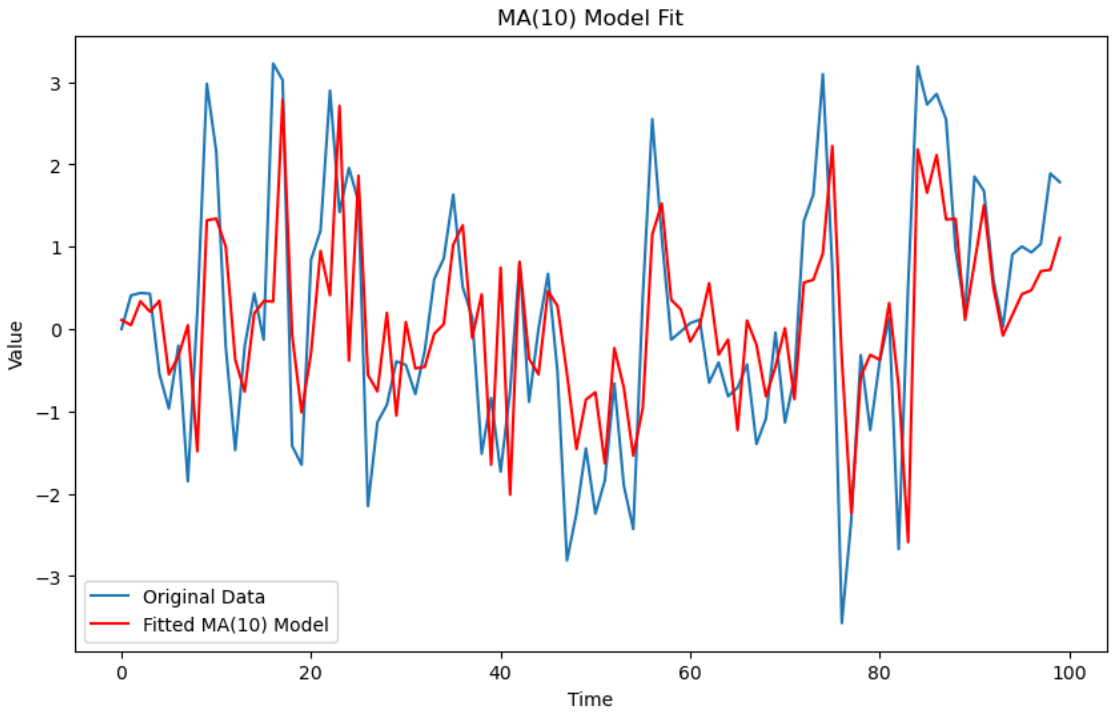
While an AR model considers the past values of the time series itself to forecast future values, an MA model considers the past error terms. In an ARMA model, both the past values of the time series and the past error terms are considered. The choice between different time series models (AR, MA, ARMA, etc.) depends on the characteristics of the data, the presence of autocorrelation, and the specific modeling requirements. Additionally, more advanced models like autoregressive integrated moving average (ARIMA) and seasonal ARIMA (SARIMA) can handle both autoregressive and moving average components, as well as seasonal patterns in the data.

[7]:
```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Generate sample data for MA model
np.random.seed(523)
num_samples = 100
error_terms = np.random.normal(0, 1, num_samples)
ma_data = np.zeros(num_samples)
for i in range(1, num_samples):
    ma_data[i] = 0.7 * error_terms[i-1] + error_terms[i]

# Fit the MA(1) model
model = ARIMA(ma_data, order=(0, 0, 10))
model_fit = model.fit()

# Get the model parameters
ma_coefficient = model_fit.params[1]

# Print the model summary
print(model_fit.summary())
```

```python
# Plot the original data and the fitted MA model
plt.figure(figsize=(10, 6))
plt.plot(ma_data, label='Original Data')
plt.plot(model_fit.fittedvalues, color='red', label='Fitted MA(10) Model')
plt.title('MA(10) Model Fit')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  100
Model:                 ARIMA(0, 0, 10)   Log Likelihood                -152.625
Date:                Tue, 29 Aug 2023   AIC                            329.249
Time:                        09:35:45   BIC                            360.511
Sample:                             0   HQIC                           341.901
                                  - 100
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1106      0.259      0.427      0.669      -0.397       0.618
ma.L1          0.8439      0.107      7.904      0.000       0.635       1.053
ma.L2          0.1366      0.150      0.914      0.361      -0.156       0.430
ma.L3          0.1967      0.143      1.371      0.170      -0.084       0.478
ma.L4          0.0650      0.167      0.388      0.698      -0.263       0.393
ma.L5         -0.0499      0.224     -0.222      0.824      -0.490       0.390
ma.L6          0.0297      0.176      0.168      0.866      -0.316       0.375
ma.L7          0.1362      0.121      1.129      0.259      -0.100       0.373
ma.L8          0.0280      0.127      0.221      0.825      -0.220       0.276
ma.L9         -0.1213      0.153     -0.793      0.428      -0.421       0.178
ma.L10        -0.1798      0.127     -1.414      0.157      -0.429       0.069
sigma2         1.2143      0.188      6.471      0.000       0.847       1.582
==============================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
1.89
Prob(Q):                              0.94   Prob(JB):
0.39
Heteroskedasticity (H):               1.04   Skew:
0.08
Prob(H) (two-sided):                  0.91   Kurtosis:
3.65
==============================================================================
===
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```



Q7. What is a mixed ARMA model and how does it differ from an AR or MA model?

A mixed autoregressive moving average (ARMA) model, also known as an ARMA(p, q) model, is a type of time series model that combines both autoregressive (AR) and moving average (MA) components. It incorporates the past values of the time series (AR) as well as the error terms (MA) to describe and forecast the behavior of the time series.

In an ARMA(p, q) model, the value of the time series at a particular time point is modeled as a linear combination of the past values of the time series and the error terms. The AR component captures the linear relationship between the current value of the time series and its past values, while the MA component accounts for the error terms at previous time points.

The key characteristics of an ARMA model are:

1. Autoregressive (AR) Component: The AR component models the relationship between the current value of the time series and its past values. It assumes that the current value is linearly dependent on a specified number of lagged values of the time series. The order of the AR component, denoted as p, determines the number of lagged values considered.

2. Moving Average (MA) Component: The MA component models the relationship between the current value of the time series and the error terms at previous time points. It assumes that the current value is a linear combination of the error terms. The order of the MA component, denoted as q, represents the number of lagged error terms considered.

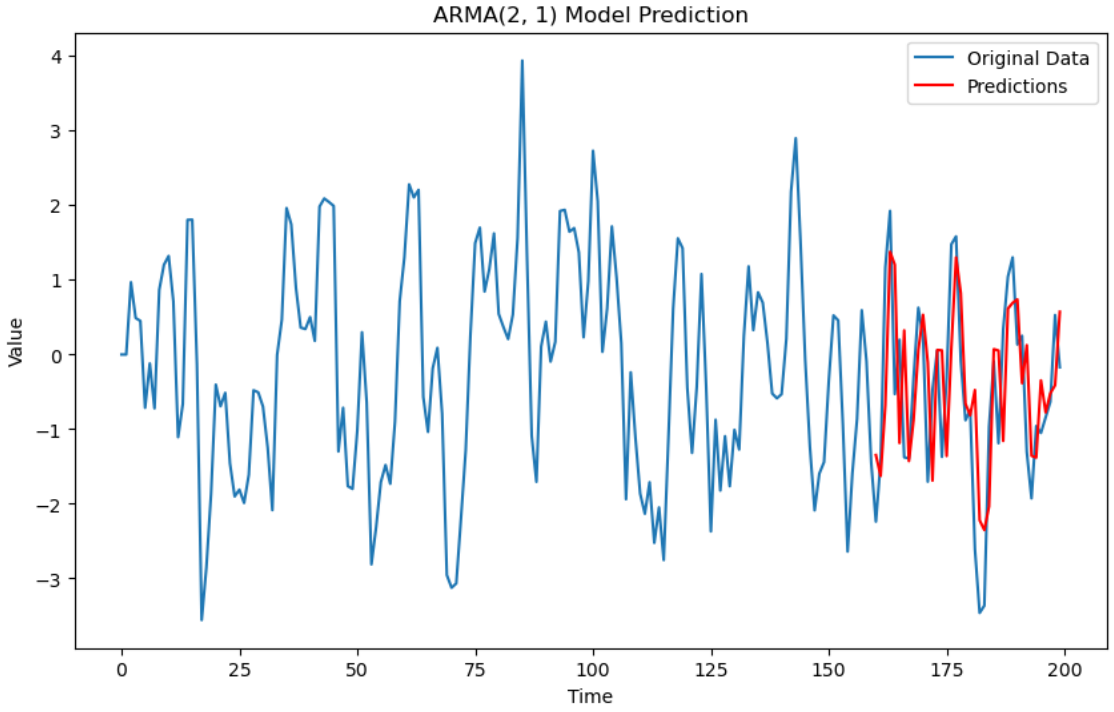
3. Combination of AR and MA: The ARMA model combines the AR and MA components to capture the dynamics of the time series. It allows for modeling both the temporal dependence in the time series itself (AR) and the dependence on the error terms (MA) to account for any residual patterns.

Compared to an AR model, which solely considers the past values of the time series, and an MA model, which solely considers the error terms, an ARMA model incorporates both components, making it more flexible in capturing different types of time series behavior.

It is important to note that an ARMA model assumes stationarity of the time series and does not account for any trend or seasonality. For non-stationary time series or those with trend and seasonality, more advanced models such as autoregressive integrated moving average (ARIMA) or seasonal ARIMA (SARIMA) models are commonly used.

[10]:
```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Set seed for reproducibility
np.random.seed(981)

# Parameters
num_samples = 200  # Number of data points
ar_params = [0.7, -0.2]  # AR parameters
ma_params = [0.3]  # MA parameters

# Generate random data for ARMA(2, 1) model
arma_data = np.zeros(num_samples)
error_terms = np.random.normal(0, 1, num_samples)
for i in range(2, num_samples):
    arma_data[i] = ar_params[0] * arma_data[i-1] + ar_params[1] *␣
 ↪arma_data[i-2] + \
                   ma_params[0] * error_terms[i-1] + error_terms[i]

# Fit the ARMA(2, 1) model
model = ARIMA(arma_data, order=(2, 0, 1))
model_fit = model.fit()

# Make predictions
start_index = int(num_samples * 0.8)  # Start index for test data
predictions = model_fit.predict(start=start_index, end=num_samples-1)

# Plot the original data and the predicted values
plt.figure(figsize=(10, 6))
plt.plot(arma_data, label='Original Data')
plt.plot(range(start_index, num_samples), predictions, color='red',␣
 ↪label='Predictions')
plt.title('ARMA(2, 1) Model Prediction')
```

```
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```



[ ]: