ARIMA models provide another approach to time series forecasting. Exponential smoothing and ARIMA models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

Before we introduce ARIMA models, we must first discuss the concept of stationarity and the technique of differencing time series.

9.1 Stationarity and differencing

A stationary time series is one whose statistical properties do not depend on the time at which the series is observed.[1] Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time.

Some cases can be confusing — a time series with cyclic behaviour (but with no trend or seasonality) is stationary. This is because the cycles are not of a fixed length, so before we observe the series we cannot be sure where the peaks and troughs of the cycles will be.

In general, a stationary time series will have no predictable patterns in the long-term. Time plots will show the series to be roughly horizontal (although some cyclic behaviour is possible), with constant variance.

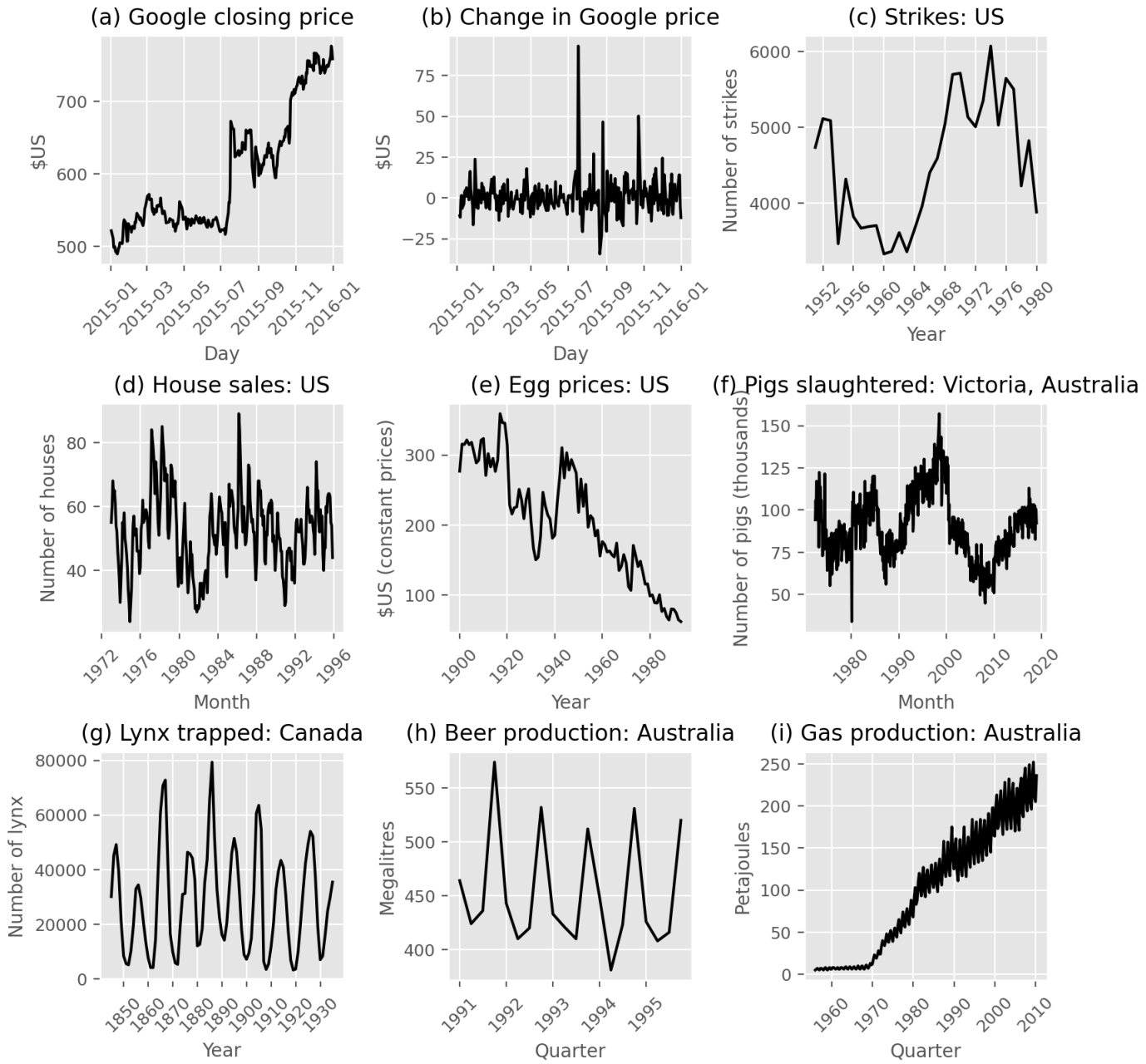

Figure 9.1: Which of these series are stationary? (a) Google closing stock price in 2015; (b) Daily change in the Google stock price in 2015; (c) Annual number of strikes in the US; (d) Monthly sales of new one-family houses sold in the US; (e) Annual price of a dozen eggs in the US (constant dollars); (f) Monthly total of pigs slaughtered in Victoria, Australia; (g) Annual total of Canadian Lynx furs traded by the Hudson Bay Company; (h) Quarterly Australian beer production; (i) Monthly Australian gas production.

Consider the nine series plotted in Figure 9.1. Which of these do you think are stationary?

Obvious seasonality rules out series (d), (h) and (i). Trends and changing levels rules out series (a), (c), (e), (f) and (i). Increasing variance also rules out (i). That leaves only (b) and (g) as stationary series.

At first glance, the strong cycles in series (g) might appear to make it non-stationary. But these cycles are aperiodic — they are caused when the lynx population becomes too large for the available feed, so that they stop breeding and the population falls to low numbers, then the regeneration of their food sources allows the population to grow again, and so on. In the long-term, the timing of these cycles is not predictable. Hence the series is stationary.

Differencing

In Figure 9.1, note that the Google stock price was non-stationary in panel (a), but the daily changes were stationary in panel (b). This shows one way to make a non-stationary time series stationary — compute the differences between consecutive observations. This is known as **differencing**.

Transformations such as logarithms can help to stabilise the variance of a time series. Differencing can help stabilise the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality.

As well as the time plot of the data, the ACF plot is also useful for identifying non-stationary time series. For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly. Also, for non-stationary

data, the value of r_1 is often large and positive.

```
google = gafa_stock.query(
    "unique_id == 'GOOG_Close' and ds.dt.year == 2015"
).reset_index()
```

```
fig = plt.figure()

gs = fig.add_gridspec(1, 2)
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])

plot_acf(google["y"], ax=ax1, zero=False,
         title="Google closing stock price",
         bartlett_confint=False,
         auto_ylims=True)
plot_acf(
    google["y"].diff()[1:],
    ax=ax2,
    zero=False,
    title="Changes in Google closing stock price",
    bartlett_confint=False,
    auto_ylims=True
)

plt.show()
```

Figure 9.2: The ACF of the Google closing stock price in 2015 (left) and of the daily changes in Google closing stock price in 2015 (right).

```
ljung_box = acorr_ljungbox(google["y"].diff()[1:], lags=[10])

ljung_box
```

	lb_stat	lb_pvalue
10	7.914	0.637

The ACF of the differenced Google stock price looks just like that of a white noise series. Only one autocorrelation is outside of the 95% limits, and the Ljung-Box Q^* statistic has a *p*-value of 0.637 (for h=10). This suggests that the *daily change* in the Google stock price is essentially a random amount which is uncorrelated with that of previous days.

Random walk model

The differenced series is the *change* between consecutive observations in the original series, and can be written as y'_t = y_t - y_{t-1}. The differenced series will have only T-1 values, since it is not possible to calculate a difference y_1' for the first observation.

When the differenced series is white noise, the model for the original series can be written as y_t - y_{t-1} = \varepsilon_t, where \varepsilon_t denotes white noise. Rearranging this leads to the “random walk” model y_t = y_{t-1} + \varepsilon_t. Random walk models are widely used for non-stationary data, particularly financial and economic data. Random walks typically have:

- long periods of apparent trends up or down
- sudden and unpredictable changes in direction.

The forecasts from a random walk model are equal to the last observation, as future movements are unpredictable, and are equally likely to be up or down. Thus, the random walk model underpins naïve forecasts, first introduced in Section 5.2.

A closely related model allows the differences to have a non-zero mean. Then y_t - y_{t-1} = c + \varepsilon_t\quad\text{or}\quad {y_t = c + y_{t-1} + \varepsilon_t}\: . The value of c is the average of the changes between consecutive observations. If c is positive, then the average change is an increase in the value of y_t. Thus, y_t will tend to drift upwards. However, if c is negative, y_t will tend to drift downwards.

This is the model behind the drift method, also discussed in Section 5.2.

Second-order differencing

Occasionally the differenced data will not appear to be stationary and it may be necessary to difference the data a second time to obtain a stationary series: \begin{align*} y''_{t} &= y'_{t} - y'_{t - 1} \\ &= (y_t - y_{t-1}) - (y_{t-1}-y_{t-2})\\ &= y_t - 2y_{t-1} +y_{t-2}. \end{align*} In this case, y_t'' will have T-2 values. Then, we would model the “change in the changes” of the original data. In practice, it is almost never necessary to go beyond second-order differences.

Seasonal differencing

A seasonal difference is the difference between an observation and the previous observation from the same season. So y'_t = y_t - y_{t-m}, where m= the number of seasons. These are also called “lag-m differences”, as we subtract the observation after a lag of m periods.

If seasonally differenced data appear to be white noise, then an appropriate model for the original data is y_t = y_{t-m}+\varepsilon_t. Forecasts from this model are equal to the last observation from the relevant season. That is, this model gives seasonal naïve forecasts, introduced in Section 5.2.

The bottom panel in Figure 9.3 shows the seasonal differences of the logarithm of the monthly scripts for A10 (antidiabetic) drugs sold in Australia. The transformation and differencing have made the series look relatively stationary.

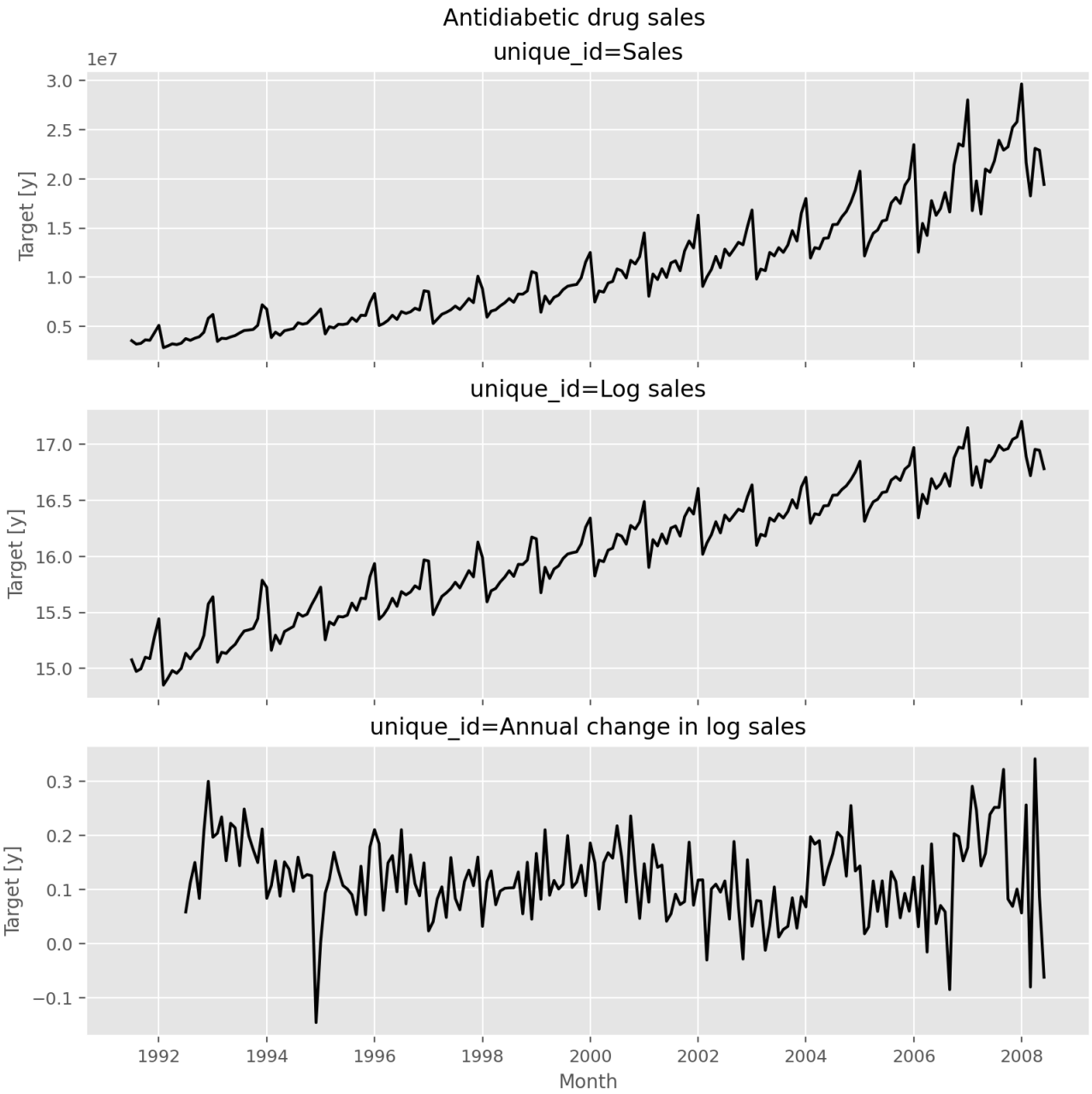

Figure 9.3: Logs and seasonal differences of the A10 (antidiabetic) sales data. The logarithms stabilise the variance, while the seasonal differences remove the seasonality and trend.

To distinguish seasonal differences from ordinary differences, we sometimes refer to ordinary differences as “first differences”, meaning differences at lag 1.

Sometimes it is necessary to take both a seasonal difference and a first difference to obtain stationary data. Figure 9.4 plots Australian corticosteroid drug sales ($AUD) (top panel). Here, the data are first transformed using logarithms (second panel), then seasonal differences are calculated (third panel). The data still seem somewhat non-stationary, and so a further lot of first differences are computed (bottom panel).

```
pbs = pd.read_csv("../data/PBS_unparsed.csv", parse_dates=["Month"])
pbs = pbs.rename(columns={"Month": "ds"})
pbs = pbs.query("ATC2 == 'H02'").reset_index(drop=True)
pbs = pbs.groupby(["ds"])["Cost"].sum().reset_index()
pbs = pbs.rename(columns={"Cost": "Sales"})
pbs["Log sales"] = np.log(pbs["Sales"])
pbs["Annual change in log sales"] = pbs["Log sales"].diff(periods=12)
pbs["Doubly differenced log sales"] = pbs["Annual change in log sales"].diff(periods=12)

pbs = pbs.set_index(["ds"]).stack().reset_index()
pbs = pbs.rename(columns={"level_1": "unique_id", 0: "y"})
pbs = pbs.sort_values(by=["unique_id", "ds"], ascending=[False, True]).reset_index(
    drop=True
)

_, axes = plt.subplots(nrows=4, ncols=1, figsize=(8, 9), sharex=True)
fig = plot_series_utils(pbs, ax=axes)
for ax in fig.axes:
    ax.tick_params(axis='both')
    if ax.get_legend():
        ax.get_legend().remove()
    for label in ax.get_xticklabels():
        label.set_rotation(0)
fig.legends = []
ax.set_xlabel("Month")
fig.suptitle("Corticosteroid drug sales", x=0.525)
fig
```

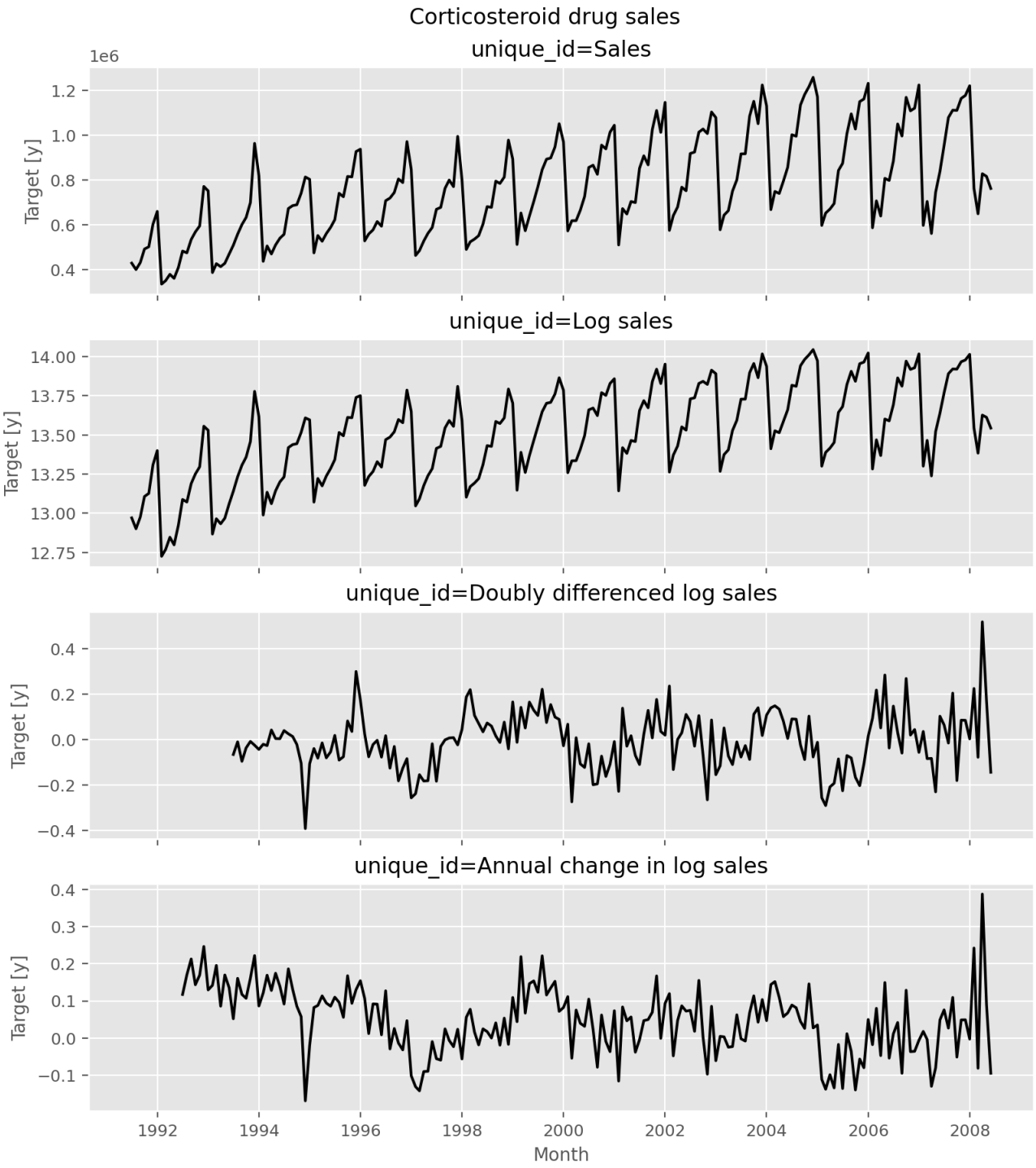

Figure 9.4: Top-left panel: Corticosteroid drug sales ($AUD). Other panels show the same data after transforming and differencing.

There is a degree of subjectivity in selecting which differences to apply. The seasonally differenced data in Figure 9.3 do not show substantially different behaviour from the seasonally differenced data in Figure 9.4. In the latter case, we could have decided to stop with the seasonally differenced data, and not done an extra round of differencing. In the former case, we could have decided that the data were not sufficiently stationary and taken an extra round of differencing. Some formal tests for differencing are discussed below, but there are always some choices to be made in the modelling process, and different analysts may make different choices.

If y'_t = y_t - y_{t-m} denotes a seasonally differenced series, then the twice-differenced series is \begin{align*} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-m}) - (y_{t-1} - y_{t-m-1}) \\ &= y_t -y_{t-1} - y_{t-m} + y_{t-m-1}\: \end{align*} When both seasonal and first differences are applied, it makes no difference which is done first—the result will be the same. However, if the data have a strong seasonal pattern, we recommend that seasonal differencing be done first, because the resulting series will sometimes be stationary and there will be no need for a further first difference. If first differencing is done first, there will still be seasonality present.

Beware that applying more differences than required will induce false dynamics or autocorrelations that do not really exist in the time series. Therefore, do as few differences as necessary to obtain a stationary series.

It is important that if differencing is used, the differences are interpretable. First differences are the change between one observation and the next. Seasonal differences are the change between one year to the next. Other lags are unlikely to make much

interpretable sense and should be avoided.

Unit root tests

One way to determine more objectively whether differencing is required is to use a *unit root test*. These are statistical hypothesis tests of stationarity that are designed for determining whether differencing is required.

A number of unit root tests are available, which are based on different assumptions and may lead to conflicting answers. In our analysis, we use the *Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test* (Kwiatkowski et al. 1992). In this test, the null hypothesis is that the data are stationary, and we look for evidence that the null hypothesis is false. Consequently, small p-values (e.g., less than 0.05) suggest that differencing is required. The test can be computed using the `kpss()` function.

For example, let us apply it to the Google stock price data.

```
goog_kpss_stat, goog_kpss_pvalue, _, _ = kpss(google["y"], nlags=5)

print(f"kpss_stat: {goog_kpss_stat:.3f}, kpss_pvalue: {goog_kpss_pvalue:.2f}")
```

```
kpss_stat: 3.561, kpss_pvalue: 0.01
```

The KPSS test p-value is reported as a number between 0.01 and 0.1. If the actual p-value is less than 0.01, it is reported as 0.01; and if the actual p-value is greater than 0.1, it is reported as 0.1. In this case, the p-value is shown as 0.01 (and therefore it may be smaller than that), indicating that the null hypothesis is rejected. That is, the data are not stationary. We can difference the data, and apply the test again.

```
googdiff_kpss_stat, googdiff_kpss_pvalue, _, _ = kpss(google["y"].diff()[1:], nlags=5)

print(f"kpss_stat: {googdiff_kpss_stat:.3f}, kpss_pvalue: {googdiff_kpss_pvalue:.2f}")
```

```
kpss_stat: 0.099, kpss_pvalue: 0.10
```

This time, the p-value is reported as 0.1 (and so it could be larger than that). We can conclude that the differenced data appear stationary.

This process of using a sequence of KPSS tests to determine the appropriate number of first differences is carried out using the `ndiffs()` feature.

```
ndiffs(google["y"].values)
```

```
1
```

As we saw from the KPSS tests above, one difference is required to make the `google_2015` data stationary.

A similar feature for determining whether seasonal differencing is required is `nsdiffs()`, which uses the measure of seasonal strength introduced in Section 4.3 to determine the appropriate number of seasonal differences required. No seasonal differences are suggested if F_S<0.64, otherwise one seasonal difference is suggested.

We can apply `nsdiffs()` to the monthly total Australian retail turnover.

```
aus_retail = pd.read_csv("../data/aus_retail.csv", parse_dates=["Month"])
aus_total_retail = aus_retail.groupby(["Month"])["Turnover"].sum().reset_index()
aus_total_retail["log(Turnover)"] = np.log(aus_total_retail["Turnover"])
```

```
nsdiffs(aus_total_retail["log(Turnover)"].values, period=12)
```

```
1
```

```
ndiffs(aus_total_retail["log(Turnover)"].diff(12)[1:].values)
```

```
1
```

Because `nsdiffs()` returns 1 (indicating one seasonal difference is required), we apply the `ndiffs()` function to the seasonally differenced data. These functions suggest we should do both a seasonal difference and a first difference.

9.2 Backshift notation

The backward shift operator B is a useful notational device when working with time series lags: B y_{t} = y_{t - 1} \: . (Some references use L for “lag” instead of B for “backshift”.) In other words, B, operating on y_{t}, has the effect of shifting the data back one period. Two applications of B to y_{t} shifts the data back two periods: B(By_{t}) = B^{2}y_{t} = y_{t-2}\: . For monthly data, if we wish to consider “the same month last year,” the notation is B^{12}y_{t} = y_{t-12}.

The backward shift operator is convenient for describing the process of *differencing*. A first difference can be written as y'_{t} = y_{t} - y_{t-1} = y_t - By_{t} = (1 - B)y_{t}\: . So a first difference can be represented by (1 - B). Similarly, if second-order differences have to be computed, then: y''_{t} = y_{t} - 2y_{t - 1} + y_{t - 2} = (1-2B+B^2)y_t = (1 - B)^{2} y_{t}\: . In general, a dth-order difference can be written as (1 - B)^{d} y_{t}.

Backshift notation is particularly useful when combining differences, as the operator can be treated using ordinary algebraic rules. In particular, terms involving B can be multiplied together.

For example, a seasonal difference followed by a first difference can be written as \begin{align*} (1-B)(1-B^m)y_t &= (1 - B - B^m + B^{m+1})y_t \\ &= y_t-y_{t-1}-y_{t-m}+y_{t-m-1}, \end{align*} the same result we obtained earlier.

9.3 Autoregressive models

In a multiple regression model, introduced in Chapter 7, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of *past values of the variable*. The term *auto*regression indicates that it is a regression of the variable against itself.

Thus, an autoregressive model of order p can be written as y_{t} = c + \phi_{1}y_{t-1} + \phi_{2}y_{t-2} + \dots + \phi_{p}y_{t-p} + \varepsilon_{t}, where \varepsilon_t is white noise. This is like a multiple regression but with *lagged values* of y_t as predictors. We refer to this as an **AR(p) model**, an autoregressive model of order p.

Autoregressive models are remarkably flexible at handling a wide range of different time series patterns. The two series in Figure 9.5 show series from an AR(1) model and an AR(2) model. Changing the parameters \phi_1,\dots,\phi_p results in different time series patterns. The variance of the error term \varepsilon_t will only change the scale of the series, not the patterns.

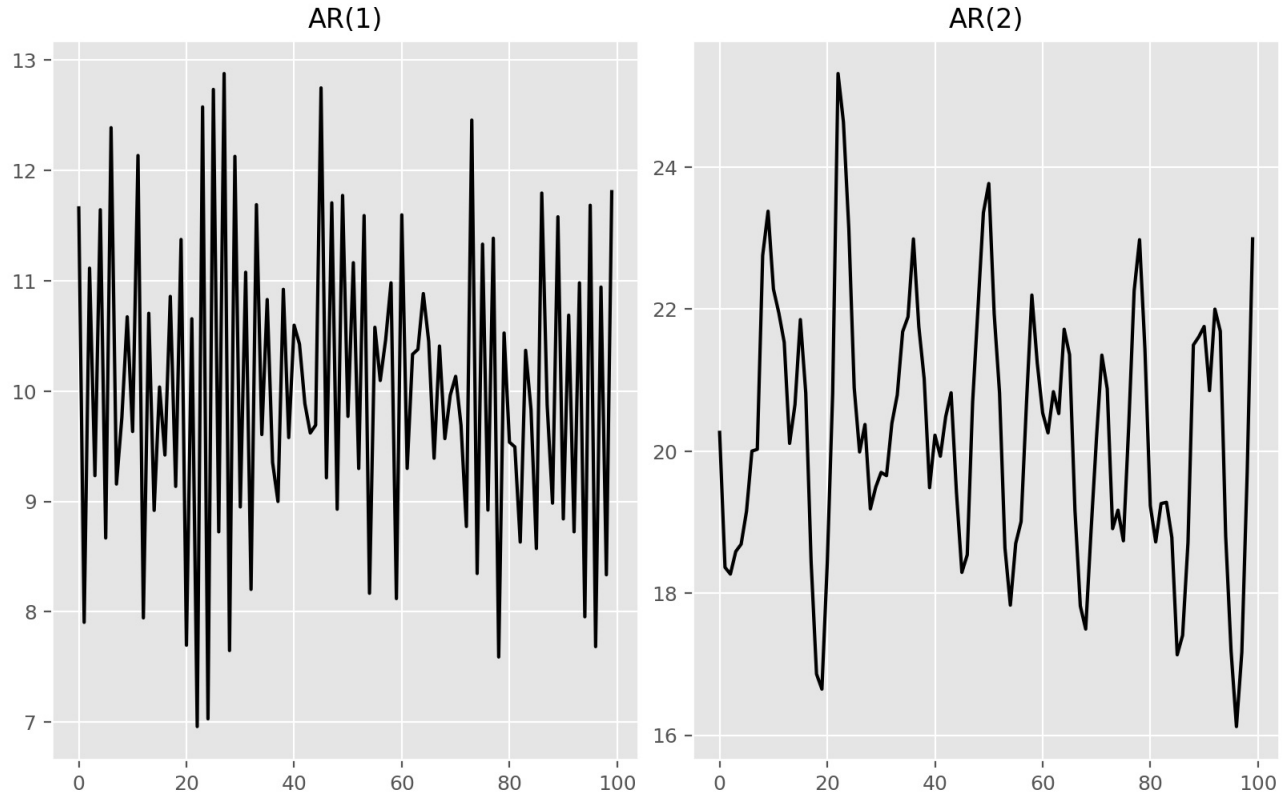

Figure 9.5: Two examples of data from autoregressive models with different parameters. Left: AR(1) with y_t = 18 -0.8y_{t-1} + \varepsilon_t. Right: AR(2) with y_t = 8 + 1.3y_{t-1}-0.7y_{t-2}+\varepsilon_t. In both cases, $arepsilon_t$ is normally distributed white noise with mean zero and variance one.

For an AR(1) model:

- when \phi_1=0 and c=0, y_t is equivalent to white noise;
- when \phi_1=1 and c=0, y_t is equivalent to a random walk;
- when \phi_1=1 and c\ne0, y_t is equivalent to a random walk with drift;
- when \phi_1<0, y_t tends to oscillate around the mean.

We normally restrict autoregressive models to stationary data, in which case some constraints on the values of the parameters are required.

- For an AR(1) model: -1 < \phi_1 < 1.
- For an AR(2) model: -1 < \phi_2 < 1, \phi_1+\phi_2 < 1, \phi_2-\phi_1 < 1.

When p\ge3, the restrictions are much more complicated, however software packages such as `statsforecast` take care of that.

9.4 Moving average models

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model, y_{t} = c + \varepsilon_t + \theta_{1}\varepsilon_{t-1} + \theta_{2}\varepsilon_{t-2} + \dots + \theta_{q}\varepsilon_{t-q}, where \varepsilon_t is white noise. We refer to this as an **MA(q) model**, a moving average model of order q. Of course, we do not *observe* the values of \varepsilon_t, so it is not really a regression in the usual sense.

Notice that each value of y_t can be thought of as a weighted moving average of the past few forecast errors (although the coefficients will not normally sum to one). However, moving average *models* should not be confused with the moving average *smoothing* we discussed in Chapter 3. A moving average model is used for forecasting future values, while moving average smoothing is used for estimating the trend-cycle of past values.

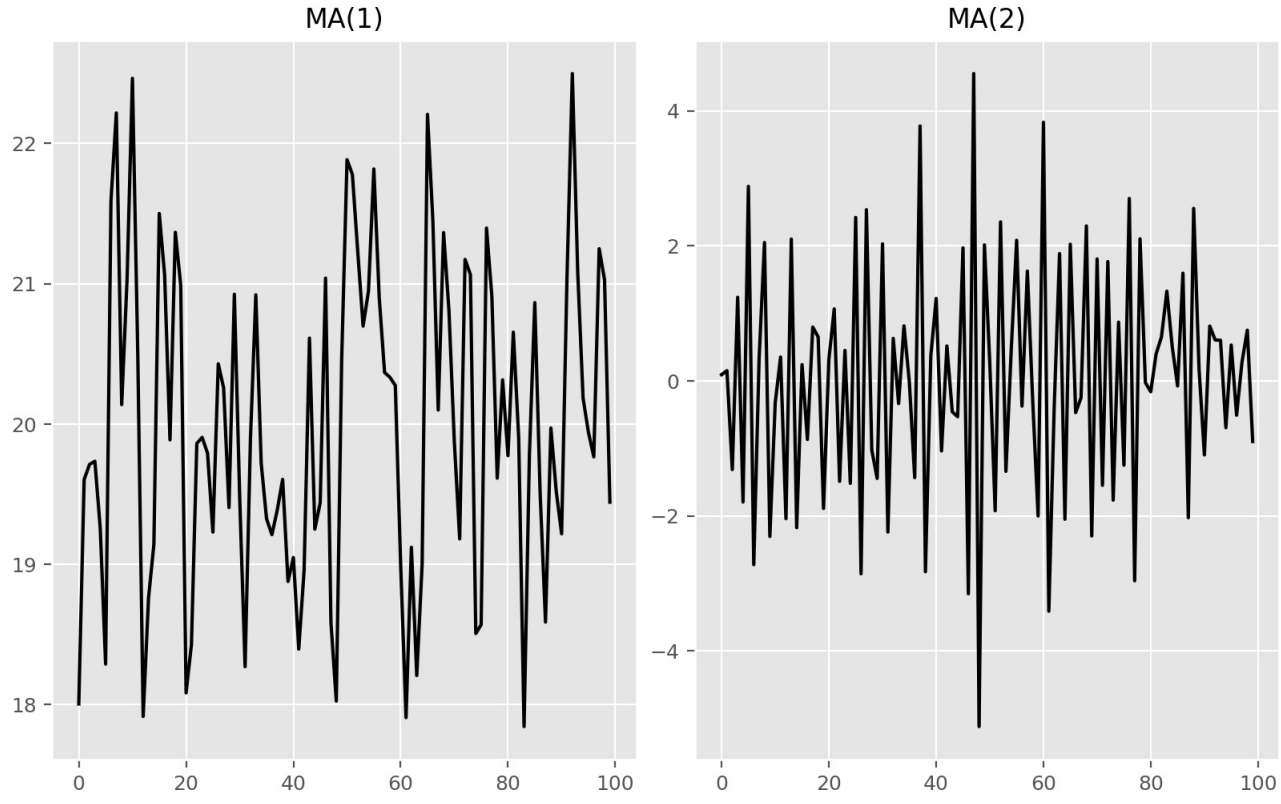

Figure 9.6: Two examples of data from moving average models with different parameters. Left: MA(1) with y_t = 20 + \varepsilon_t + 0.8\varepsilon_{t-1}. Right: MA(2) with y_t = \varepsilon_t- \varepsilon_{t-1}+0.8\varepsilon_{t-2}. In both cases, \varepsilon_t is normally distributed white noise with mean zero and variance one.

Figure 9.6 shows some data from an MA(1) model and an MA(2) model. Changing the parameters \theta_1,\dots,\theta_q results in different time series patterns. As with autoregressive models, the variance of the error term \varepsilon_t will only change the scale of the series, not the patterns.

It is possible to write any stationary AR(p) model as an MA(\infty) model. For example, using repeated substitution, we can demonstrate this for an AR(1) model: \begin{align*} y_t &= \phi_1y_{t-1} + \varepsilon_t\\ &= \phi_1(\phi_1y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t\\ &= \phi_1^2y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t\\ &= \phi_1^3y_{t-3} + \phi_1^2\varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t\\ &\text{etc.} \end{align*} Provided -1 < \phi_1 < 1, the value of \phi_1^k will get smaller as k gets larger. So eventually we obtain y_t = \varepsilon_t + \phi_1 \varepsilon_{t-1} + \phi_1^2 \varepsilon_{t-2} + \phi_1^3 \varepsilon_{t-3} + \cdots, an MA(\infty) process.

The reverse result holds if we impose some constraints on the MA parameters. Then the MA model is called **invertible**. That is, we can write any invertible MA(q) process as an AR(\infty) process. Invertible models are not simply introduced to enable us to convert from MA models to AR models. They also have some desirable mathematical properties.

For example, consider the MA(1) process, y_{t} = \varepsilon_t + \theta_{1}\varepsilon_{t-1}. In its AR(\infty) representation, the most recent error can be written as a linear function of current and past observations: \varepsilon_t = \sum_{j=0}^\infty (-\theta_1)^j y_{t-j}. When |\theta_1| > 1, the weights increase as lags increase, so the more distant the observations the greater their influence on the current error. When |\theta_1|=1, the weights are constant in size, and the distant observations have the same influence as the recent observations. As neither of these situations make much sense, we require |\theta_1|<1, so the most recent observations have higher weight than observations from the more distant past. Thus, the process is invertible when |\theta_1|<1.

The invertibility constraints for other models are similar to the stationarity constraints.

- For an MA(1) model: -1<\theta_1<1.
- For an MA(2) model: -1<\theta_2<1,~ \theta_2+\theta_1 >-1,~ \theta_1 -\theta_2 < 1.

More complicated conditions hold for q\ge3. Again, `statsforecast` will take care of these constraints when estimating the models.

9.5 Non-seasonal ARIMA models

If we combine differencing with autoregression and a moving average model, we obtain a non-seasonal ARIMA model. ARIMA is an acronym for AutoRegressive Integrated Moving Average (in this context, “integration” is the reverse of differencing). The full model can be written as y'_{t} = c + \phi_{1}y'_{t-1} + \cdots + \phi_{p}y'_{t-p} + \theta_{1}\varepsilon_{t-1} + \cdots + \theta_{q}\varepsilon_{t-q} + \varepsilon_{t}, \tag{9.1} where y'_{t} is the differenced series (it may have been differenced more than once). The “predictors” on the right hand side include both lagged values of y_t and lagged errors. We call this an **ARIMA(p, d, q) model**, where

- p = order of the autoregressive part;
- d = degree of first differencing involved;
- q = order of the moving average part.

The same stationarity and invertibility conditions that are used for autoregressive and moving average models also apply to an ARIMA model.

Many of the models we have already discussed are special cases of the ARIMA model:

Special case	ARIMA(p, d, q)
White noise	ARIMA(0,0,0) with no constant
Random walk	ARIMA(0,1,0) with no constant
Random walk with drift	ARIMA(0,1,0) with a constant
Autoregression	ARIMA(p,0,0)
Moving average	ARIMA(0,0,q)

Once we start combining components in this way to form more complicated models, it is much easier to work with the backshift notation. For example, Equation 9.1 can be written in backshift notation as \begin{array}{c c c} (1-\phi_1B - \cdots - \phi_p B^p) & (1-B)^d y_{t} &= &c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t\\ {\uparrow} & {\uparrow} & &{\uparrow}\\ \text{AR(p)} & \text{d differences} & & \text{MA(q)}\\ \end{array} \tag{9.2} Selecting appropriate values for p, d and q can be difficult. However, the `AutoARIMA()` function from the `statsforecast` package will do it for you automatically. In Section 9.7, we will learn how this function works, along with some methods for choosing these values yourself.

Example: Egyptian exports

Figure 9.7 shows Egyptian exports as a percentage of GDP from 1960 to 2017.

```
global_economy = pd.read_csv("../data/global_economy.csv", parse_dates=[2])
global_economy = global_economy.rename(columns={"Exports": "y"})
egyptian_economy = global_economy.query("Code == 'EGY'").reset_index(drop=True)

plot_series(egyptian_economy, xlabel="Year [1Y]", ylabel="% of GDP", title="Egyptian exports")
```

Figure 9.7: Annual Egyptian exports as a percentage of GDP since 1960.

The following Python code selects a non-seasonal ARIMA model automatically.

```
from statsforecast import StatsForecast
from statsforecast.models import AutoARIMA
from statsforecast.arima import ARIMASummary
from copy import deepcopy

models = [AutoARIMA(allowmean=True)]

sf = StatsForecast(models=models, freq="A", n_jobs=-1)

sf.fit(df=egyptian_economy[["ds", "y", "unique_id"]])

print(ARIMASummary(sf.fitted_[0, 0].model_))
coefs = deepcopy(sf.fitted_[0, 0].model_['coef'])
coefs["mean"] = coefs.pop("intercept")
print(f"Coefficients: {coefs}")
print(f"sigma^2     : {sf.fitted_[0, 0].model_['sigma2']:.2f}")
print(f"loglik      : {sf.fitted_[0, 0].model_['loglik']:.2f}")
print(f"aic         : {sf.fitted_[0, 0].model_['aic']:.2f}")
print(f"aicc        : {sf.fitted_[0, 0].model_['aicc']:.2f}")
print(f"bic         : {sf.fitted_[0, 0].model_['bic']:.2f}")
```

```
ARIMA(2,0,1) with non-zero mean
Coefficients: {'ar1': 1.6764281936479777, 'ar2': -0.8034081643875923, 'ma1': -0.6896288223396578, 'mean': 20.17
9017921618534}
sigma^2     : 8.05
loglik      : -141.57
aic         : 293.13
aicc        : 294.29
bic         : 303.43
```

This is an ARIMA(2,0,1) model: y_t = c + 1.68 y_{t-1} - 0.80 y_{t-2} -0.69 \varepsilon_{t-1} + \varepsilon_{t}, where c = 20.18 * (1 - 1.68 + 0.8) = 2.43 and \varepsilon_t is white noise with a standard deviation of 2.837 = \sqrt{8.046}. Forecasts from the model are shown in Figure 9.8. Notice how they have picked up the cycles evident in the Egyptian economy over the last few decades.

```
levels = [80, 95]
forecasts = sf.predict(h=10, level=levels)
plot_series(
    df=egyptian_economy[["ds", "y", "unique_id"]], forecasts_df=forecasts, level=levels,
    xlabel="Year [1Y]", ylabel="% of GDP", title="Egyptian exports", rm_legend=False,
)
```

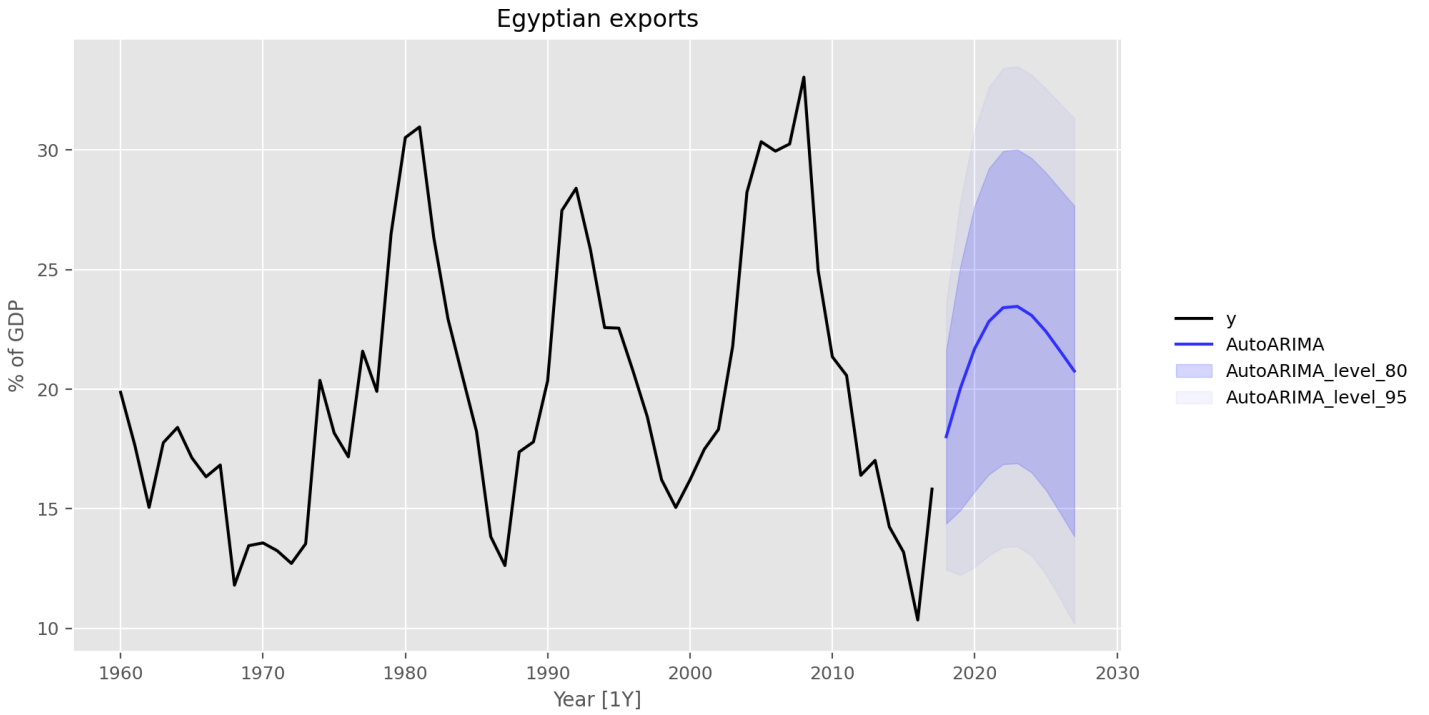

Figure 9.8: Forecasts of Egyptian exports.

Understanding ARIMA models

The `AutoARIMA()` function is useful, but anything automated can be a little dangerous, and it is worth understanding something of the behaviour of the models even when you rely on an automatic procedure to choose the model for you.

The constant c has an important effect on the long-term forecasts obtained from these models.

- If c=0 and d=0, the long-term forecasts will go to zero.
- If c=0 and d=1, the long-term forecasts will go to a non-zero constant.
- If c=0 and d=2, the long-term forecasts will follow a straight line.
- If c\ne0 and d=0, the long-term forecasts will go to the mean of the data.
- If c\ne0 and d=1, the long-term forecasts will follow a straight line.
- If c\ne0 and d=2, the long-term forecasts will follow a quadratic trend. (This is not recommended.)

The value of d also has an effect on the prediction intervals — the higher the value of d, the more rapidly the prediction intervals increase in size. For d=0, the long-term forecast standard deviation will go to the standard deviation of the historical data, so the prediction intervals will all be essentially the same.

This behaviour is seen in Figure 9.8 where d=0 and c\ne0. In this figure, the prediction intervals are almost the same width for the last few forecast horizons, and the final point forecasts are close to the mean of the data.

The value of p is important if the data show cycles. To obtain cyclic forecasts, it is necessary to have p\ge2, along with some additional conditions on the parameters. For an AR(2) model, cyclic behaviour occurs if \phi_1^2+4\phi_2<0 (as is the case for the Egyptian exports model). In that case, the average period of the cycles is^2 \frac{2\pi}{\text{arc cos}(-\phi_1(1-\phi_2)/(4\phi_2))}.

ACF and PACF plots

It is usually not possible to tell, simply from a time plot, what values of p and q are appropriate for the data. However, it is sometimes possible to use the ACF plot, and the closely related PACF plot, to determine appropriate values for p and q.

Recall that an ACF plot shows the autocorrelations which measure the relationship between y_t and y_{t-k} for different values of k. Now if y_t and y_{t-1} are correlated, then y_{t-1} and y_{t-2} must also be correlated. However, then y_t and y_{t-2} might be correlated, simply because they are both connected to y_{t-1}, rather than because of any new information contained in y_{t-2} that could be used in forecasting y_t.

To overcome this problem, we can use **partial autocorrelations**. These measure the relationship between y_{t} and y_{t-k} after removing the effects of lags 1, 2, 3, \dots, k - 1. So the first partial autocorrelation is identical to the first autocorrelation, because there is nothing between them to remove. Each partial autocorrelation can be estimated as the last coefficient in an autoregressive model. Specifically, \alpha_k, the kth partial autocorrelation coefficient, is equal to the estimate of \phi_k in an AR(k) model. In practice, there are more efficient algorithms for computing \alpha_k than fitting all of these autoregressions, but they give the same results.

Figures 9.9 and 9.10 shows the ACF and PACF plots for the Egyptian exports data shown in Figure 9.7. The partial autocorrelations have the same critical values of \pm 1.96/\sqrt{T} as for ordinary autocorrelations, and these are typically shown on the plot as in Figure 9.10.

```
from matplotlib.ticker import MaxNLocator
fig, ax = plt.subplots()
plot_acf(egyptian_economy["y"], lags=20, zero=False,
         bartlett_confint=False,
         ax=ax,
         auto_ylims=True)
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

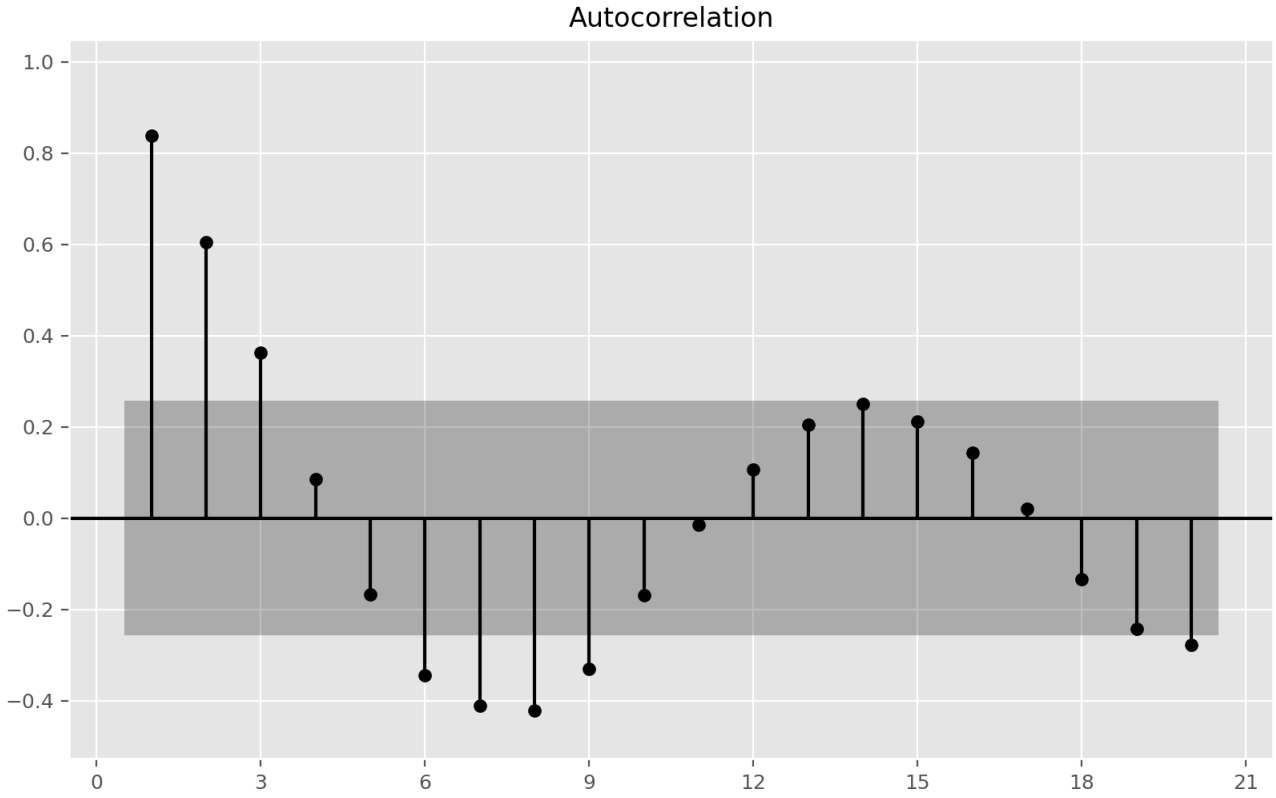

Figure 9.9: ACF of Egyptian exports.

```
fig, ax = plt.subplots()
fig = plot_pacf(egyptian_economy["y"], lags=20,
                zero=False,
                ax=ax,
                auto_ylims=True)
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

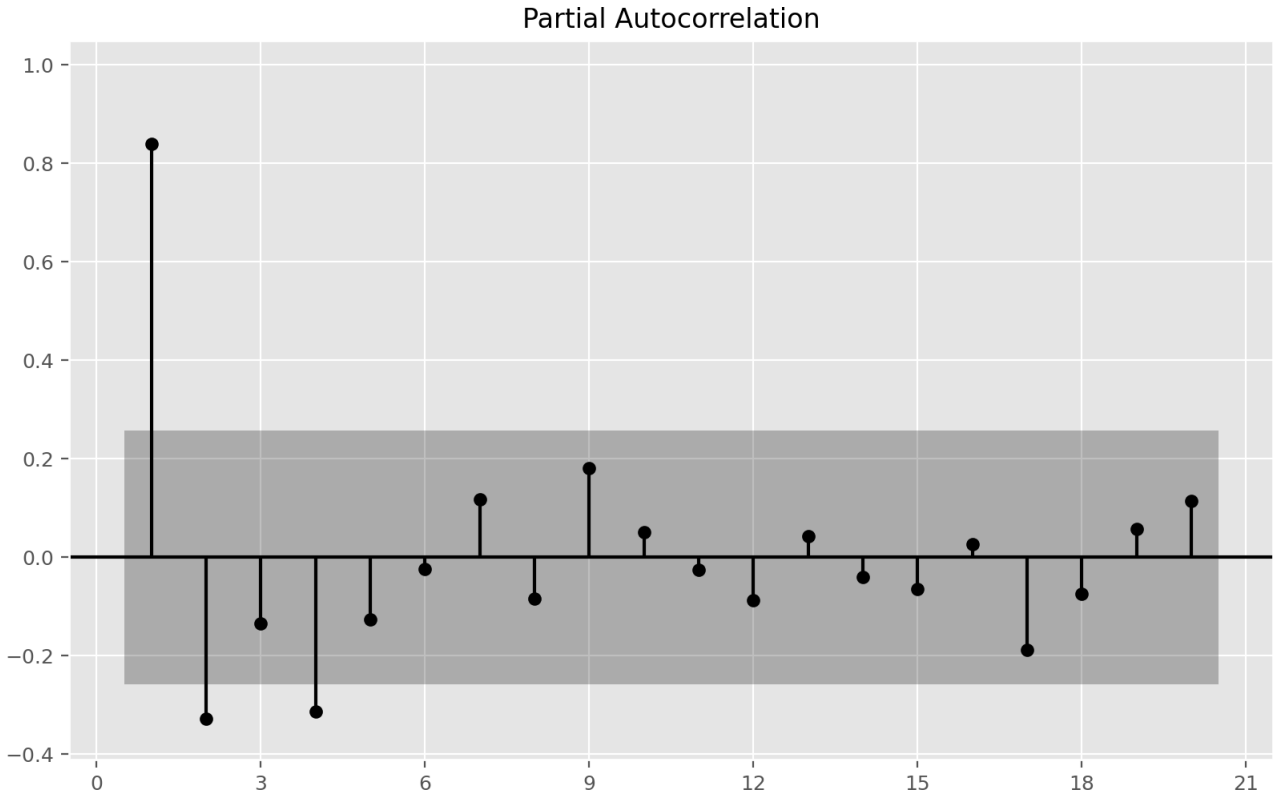

Figure 9.10: PACF of Egyptian exports.

If the data are from an ARIMA(p,d,0) or ARIMA(0,d,q) model, then the ACF and PACF plots can be helpful in determining the value of p or q. If p and q are both positive, then the plots do not help in finding suitable values of p and q.

The data may follow an ARIMA(p,d,0) model if the ACF and PACF plots of the differenced data show the following patterns:

- the ACF is exponentially decaying or sinusoidal;
- there is a significant spike at lag p in the PACF, but none beyond lag p.

The data may follow an ARIMA(0,d,q) model if the ACF and PACF plots of the differenced data show the following patterns:

- the PACF is exponentially decaying or sinusoidal;
- there is a significant spike at lag q in the ACF, but none beyond lag q.

In Figure 9.9, we see that there is a decaying sinusoidal pattern in the ACF, and in Figure 9.10 the PACF shows the last significant spike at lag 4. This is what you would expect from an ARIMA(4,0,0) model.

```
from statsforecast.models import ARIMA

models = [ARIMA(order=(4, 0, 0))]

sf = StatsForecast(models=models, freq="Y", n_jobs=-1)

sf.fit(df=egyptian_economy[["ds", "y", "unique_id"]])

print(ARIMASummary(sf.fitted_[0, 0].model_))
coefs = deepcopy(sf.fitted_[0, 0].model_['coef'])
coefs["mean"] = coefs.pop("intercept")
print(f"Coefficients: {coefs}")
print(f"sigma^2     : {sf.fitted_[0, 0].model_['sigma2']:.2f}")
print(f"loglik      : {sf.fitted_[0, 0].model_['loglik']:.2f}")
print(f"aic         : {sf.fitted_[0, 0].model_['aic']:.2f}")
print(f"aicc        : {sf.fitted_[0, 0].model_['aicc']:.2f}")
print(f"bic         : {sf.fitted_[0, 0].model_['bic']:.2f}")
```

```
ARIMA(4,0,0) with non-zero mean
Coefficients: {'ar1': 0.9860926757531944, 'ar2': -0.17152907675987353, 'ar3': 0.18072506411563896, 'ar4': -0.32
825638752480574, 'mean': 20.098620164617397}
sigma^2     : 7.88
loglik      : -140.53
aic         : 293.05
aicc        : 294.70
bic         : 305.41
```

This model is only slightly worse than the ARIMA(2,0,1) model identified by `AutoARIMA()` (with an AICc value of 294.70 compared to 294.29).

We can also specify particular values of `p`, `d` and `q` that `AutoARIMA()` can search for. For example, to find the best ARIMA model with p\in\{1,2,3\}, q\in\{0,1,2\} and d=1, you could use `AutoARIMA(start_p=1, max_p=3, start_q=0, max_q=2, d=1)`.

9.6 Estimation and order selection

Maximum likelihood estimation

Once the model order has been identified (i.e., the values of p, d and q), we need to estimate the parameters c, \phi_1,\dots,\phi_p, \theta_1,\dots,\theta_q. When `statsforecast` estimates the ARIMA model, it uses *maximum likelihood estimation* (MLE). This technique finds the values of the parameters which maximise the probability of obtaining the data that we have observed. For ARIMA models, MLE is similar to the *least squares* estimates that would be obtained by minimising \sum_{t=1}^T\varepsilon_t^2. (For the regression models considered in Chapter 7, MLE gives exactly the same parameter estimates as least squares estimation.) Note that ARIMA models are much more complicated to estimate than regression models, and different software will give slightly different answers as they use different methods of estimation, and different optimisation algorithms.

In practice, the `statsforecast` package will report the value of the *log likelihood* of the data; that is, the logarithm of the probability of the observed data coming from the estimated model. For given values of p, d and q, `AutoARIMA()` will try to maximise the log likelihood when finding parameter estimates.

Information Criteria

Akaike's Information Criterion (AIC), which was useful in selecting predictors for regression (see Section 7.5), is also useful for determining the order of an ARIMA model. It can be written as \text{AIC} = -2 \log(L) + 2(p+q+k+1), where L is the likelihood of the data, k=1 if c\ne0 and k=0 if c=0. Note that the last term in parentheses is the number of parameters in the model (including \sigma^2, the variance of the residuals).

For ARIMA models, the corrected AIC can be written as \text{AICc} = \text{AIC} + \frac{2(p+q+k+1)(p+q+k+2)}{T-p-q-k-2}, and the Bayesian Information Criterion can be written as \text{BIC} = \text{AIC} + [\log(T)-2](p+q+k+1). Good models are obtained by minimising the AIC, AICc or BIC. Our preference is to use the AICc.

It is important to note that these information criteria tend not to be good guides to selecting the appropriate order of differencing (d) of a model, but only for selecting the values of p and q. This is because the differencing changes the data on which the likelihood is computed, making the AIC values between models with different orders of differencing not comparable. So we need to use some other approach to choose d, and then we can use the AICc to select p and q.

9.7 ARIMA modelling in `statsforecast`

How does `AutoARIMA()` work?

`AutoARIMA()` uses a variation of the Hyndman-Khandakar algorithm (Hyndman & Khandakar, 2008), which combines unit root tests, minimisation of the AICc and MLE to obtain an ARIMA model. The arguments to `ARIMA()` provide for many variations on the algorithm. What is described here is the default behaviour.

Hyndman-Khandakar algorithm for automatic ARIMA modelling

1. The number of differences $0 \leq d \leq 2$ is determined using repeated KPSS tests.

2. The values of p and q are then chosen by minimising the AICc after differencing the data d times. Rather than considering every possible combination of p and q, the algorithm uses a stepwise search to traverse the model space.

    a. Four initial models are fitted:
        - ARIMA$(0, d, 0)$,
        - ARIMA$(2, d, 2)$,
        - ARIMA$(1, d, 0)$,
        - ARIMA$(0, d, 1)$.

    A constant is included unless $d = 2$. If $d \leq 1$, an additional model is also fitted:
        - ARIMA$(0, d, 0)$ without a constant.

    b. The best model (with the smallest AICc value) fitted in step (a) is set to be the "current model".

    c. Variations on the current model are considered:
        - vary p and/or q from the current model by ± 1;
        - include/exclude c from the current model.

    The best model considered so far (either the current model or one of these variations) becomes the new current model.

    d. Repeat Step 2(c) until no lower AICc can be found.

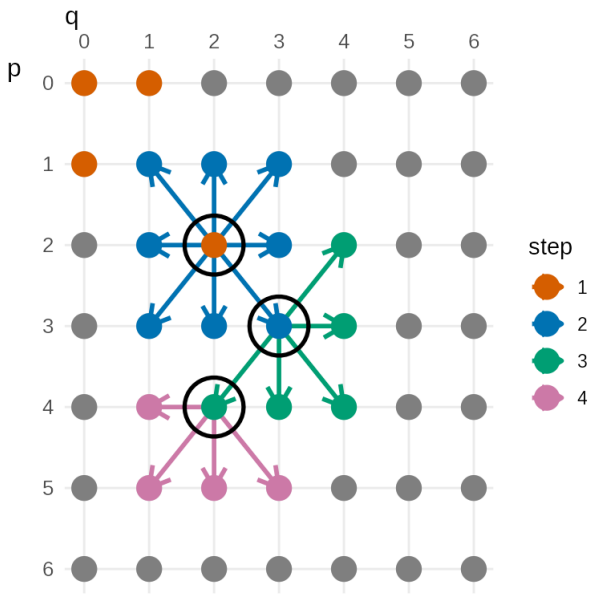

Figure 9.11: An illustrative example of the Hyndman-Khandakar stepwise search process

Figure 9.11 illustrates diagrammatically how the Hyndman-Khandakar algorithm traverses the space of the ARMA orders, through an example. The grid covers combinations of ARMA(p,q) orders starting from the top-left corner with an ARMA(0,0), with the AR order increasing down the vertical axis, and the MA order increasing across the horizontal axis.

The orange cells show the initial set of models considered by the algorithm. In this example, the ARMA(2,2) model has the lowest AICc value amongst these models. This is called the "current model" and is shown by the black circle. The algorithm then searches over neighbouring models as shown by the blue arrows. If a better model is found then this becomes the new "current model". In this example, the new "current model" is the ARMA(3,3) model. The algorithm continues in this fashion until no better model can be found. In this example the model returned is an ARMA(4,2) model.

The default procedure will switch to a new “current model” as soon as a better model is identified, without going through all the neighbouring models.

The default procedure also uses some approximations to speed up the search. These approximations can be avoided with the argument `approximation=False`. It is possible that the minimum AICc model will not be found due to these approximations, or because of the use of the stepwise procedure. A much larger set of models will be searched if the argument `stepwise=False` is used. See the help file for a full description of the arguments.

Modelling procedure

When fitting an ARIMA model to a set of (non-seasonal) time series data, the following procedure provides a useful general approach.

1. Plot the data and identify any unusual observations.
2. If necessary, transform the data (using a Box-Cox transformation) to stabilise the variance.
3. If the data are non-stationary, take first differences of the data until the data are stationary.
4. Examine the ACF/PACF: Is an ARIMA(p,d,0) or ARIMA(0,d,q) model appropriate?
5. Try your chosen model(s), and use the AICc to search for a better model.
6. Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
7. Once the residuals look like white noise, calculate forecasts.

The Hyndman-Khandakar algorithm only takes care of steps 3–5. So even if you use it, you will still need to take care of the other steps yourself.

The process is summarised in Figure 9.12.

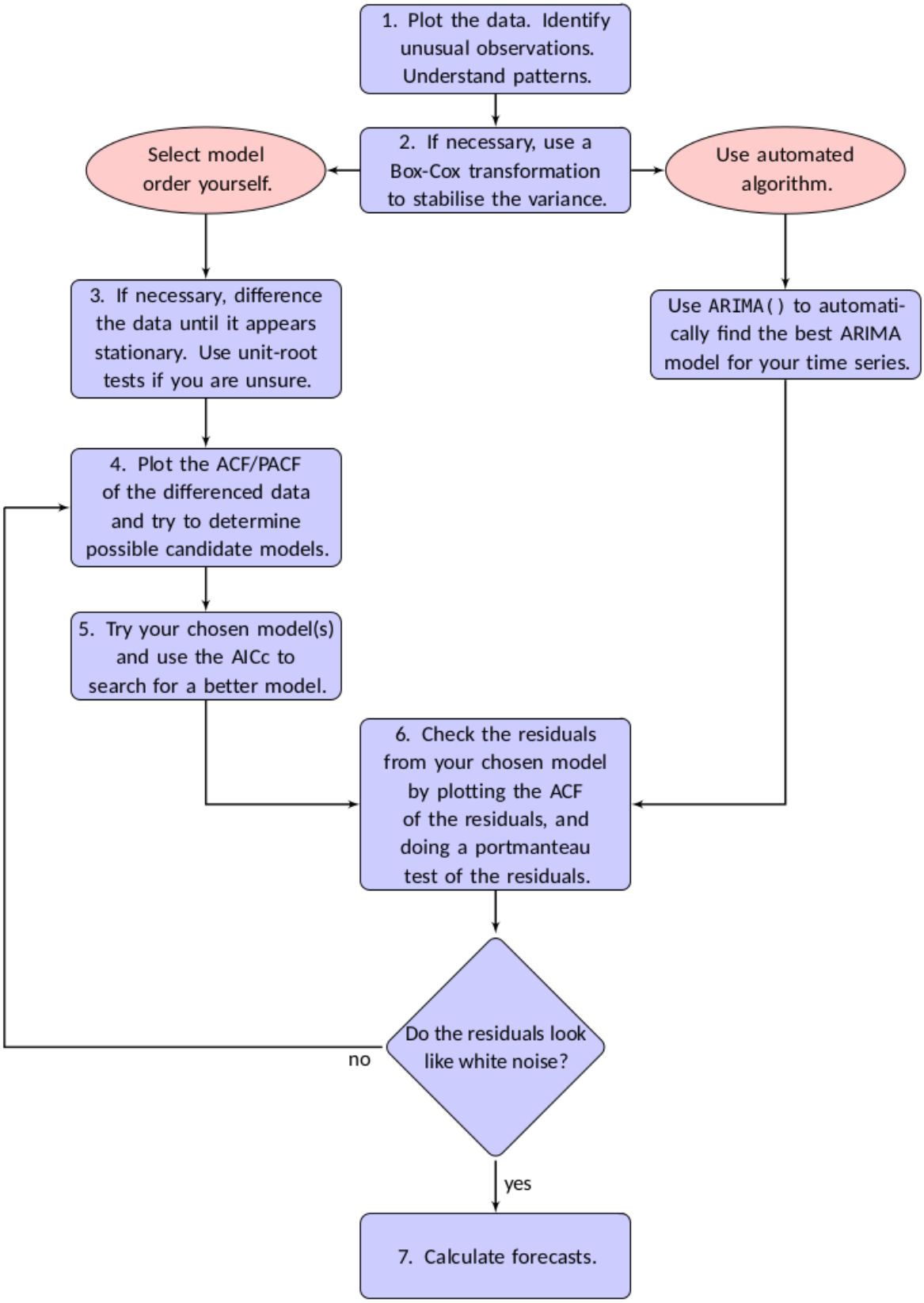

Figure 9.12: General process for forecasting using an ARIMA model.

Portmanteau tests of residuals for ARIMA models

With ARIMA models, more accurate portmanteau tests are obtained if the degrees of freedom of the test statistic are adjusted to take account of the number of parameters in the model. Specifically, we use \ell - K degrees of freedom in the test, where K is the number of AR and MA parameters in the model. So for the non-seasonal models that we have considered so far, K=p+q. The value of K is passed to the `acorr_ljungbox` function via the argument `model_df`, as shown in the example below.

Example: Central African Republic exports

We will apply this procedure to the exports of the Central African Republic shown in Figure 9.13.

```
caf_economy = global_economy.query("Code == 'CAF'").reset_index(drop=True)

plot_series(caf_economy, xlabel="Year [1Y]", ylabel="% of GDP", title="Central African Republic ex
```

Figure 9.13: Exports of the Central African Republic as a percentage of GDP.

1. The time plot shows some non-stationarity, with an overall decline. The improvement in 1994 was due to a new government which overthrew the military junta and had some initial success, before unrest caused further economic decline.
2. There is no evidence of changing variance, so we will not do a Box-Cox transformation.
3. To address the non-stationarity, we will take a first difference of the data. The differenced data are shown in Figure 9.14.

```
difference_caf_economy = caf_economy["y"].diff()[1:]

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(difference_caf_economy, marker="o")
ax1.set_ylabel("difference(Exports)")
ax1.set_xlabel("Year")

plot_acf(difference_caf_economy, ax2, zero=False,
         bartlett_confint=False,
         auto_ylims=True)
plot_pacf(difference_caf_economy, ax3, zero=False,
          auto_ylims=True)
ax3.xaxis.set_major_locator(MaxNLocator(integer=True))
ax2.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```

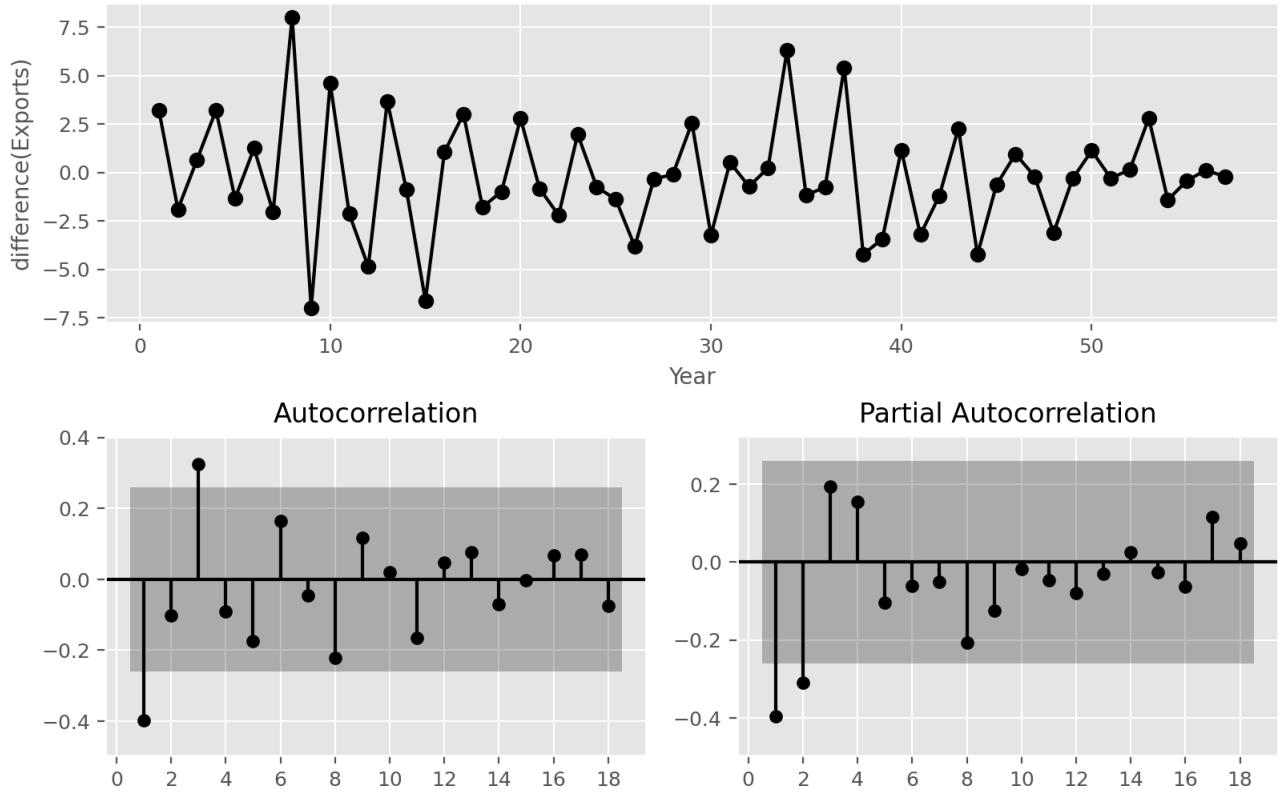

Figure 9.14: Time plot and ACF and PACF plots for the differenced Central African Republic Exports.

These now appear to be stationary.

4. The PACF shown in Figure 9.14 is suggestive of an AR(2) model; so an initial candidate model is an ARIMA(2,1,0). The ACF suggests an MA(3) model; so an alternative candidate is an ARIMA(0,1,3).
5. We fit both an ARIMA(2,1,0) and an ARIMA(0,1,3) model along with two automated model selections, one using the default stepwise procedure, and one working harder to search a larger model space.

```
models = [
    ARIMA(order=(2, 1, 0), alias="arima210"),
    ARIMA(order=(0, 1, 3), alias="arima013"),
    AutoARIMA(alias="stepwise"),
    AutoARIMA(stepwise=False, alias="search"),
]

sf = StatsForecast(models=models, freq="A", n_jobs=-1)

sf.fit(df=caf_economy[["ds", "y", "unique_id"]])

summaries = []
for model in sf.fitted_[0]:
    summary_model = {
        "model": model,
        "Orders": ARIMASummary(model.model_),
        "sigma2": model.model_["sigma2"],
        "loglik": model.model_["loglik"],
        "aic": model.model_["aic"],
        "aicc": model.model_["aicc"],
        "bic": model.model_["bic"],
    }
    summaries.append(summary_model)

pd.DataFrame(sorted(summaries, key=lambda d: d["aicc"]))
```

	model	Orders	sigma2	loglik	aic	aicc	bic
0	search	ARIMA(3,1,0)	6.519	-133.006	274.012	274.781	282.184
1	arima210	ARIMA(2,1,0)	6.706	-134.269	274.539	274.992	280.668
2	arima013	ARIMA(0,1,3)	6.537	-133.130	274.261	275.030	282.433
3	stepwise	ARIMA(2,1,2)	6.417	-132.124	274.248	275.425	284.463

The four models have almost identical AICc values. Of the models fitted, the full search has found that an ARIMA(3,1,0) gives the lowest AICc value, closely followed by the ARIMA(2,1,0) and ARIMA(0,1,3). The automated stepwise selection has identified an ARIMA(2,1,2) model, which has the highest AICc value of the four models.

6. The ACF plot of the residuals from the ARIMA(3,1,0) model shows that all autocorrelations are within the threshold limits, indicating that the residuals are behaving like white noise.

```
forecasts = sf.forecast(
    df=caf_economy[["ds", "y", "unique_id"]], h=10, fitted=True, level=[80, 95]
)
fitted_values = sf.forecast_fitted_values()
insample_forecasts = fitted_values["search"]
residuals = fitted_values["y"] - insample_forecasts

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(caf_economy["ds"], residuals, marker="o")
ax1.set_ylabel("Innovation Residuals")
ax1.set_xlabel("Year")

plot_acf(residuals, ax2, zero=False,
         bartlett_confint=False,
         auto_ylims=True)

ax3.hist(residuals, bins=20)
ax3.set_ylabel("count")
ax3.set_xlabel("residuals")
ax2.xaxis.set_major_locator(MaxNLocator(integer=True))

plt.show()
```

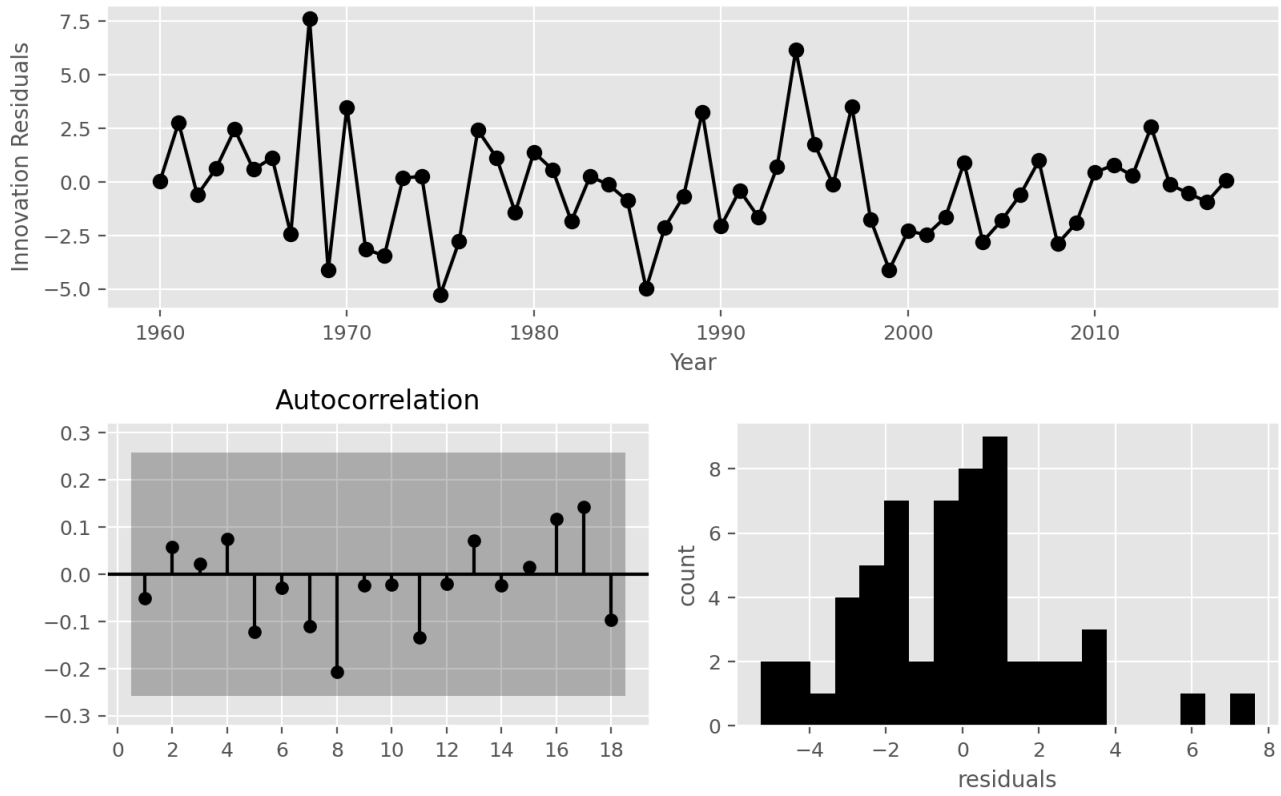

Figure 9.15: Residual plots for the ARIMA(3,1,0) model.

A portmanteau test (setting K = 3) returns a large p-value, also suggesting that the residuals are white noise.

```
ljung_box = acorr_ljungbox(residuals, lags=[10], model_df=3)

ljung_box
```

	lb_stat	lb_pvalue
10	5.704	0.575

7. Forecasts from the chosen model are shown in Figure 9.16.

```
plot_series(
    df=caf_economy,
    forecasts_df=forecasts.reset_index()[
        [
            "unique_id",
            "ds",
            "search",
            "search-lo-80",
            "search-lo-95",
            "search-hi-80",
            "search-hi-95",
        ]
    ],
    level=[80, 95],
    xlabel="Year", ylabel="Exports", title="Central African Republic exports",
    rm_legend=False,
)
```

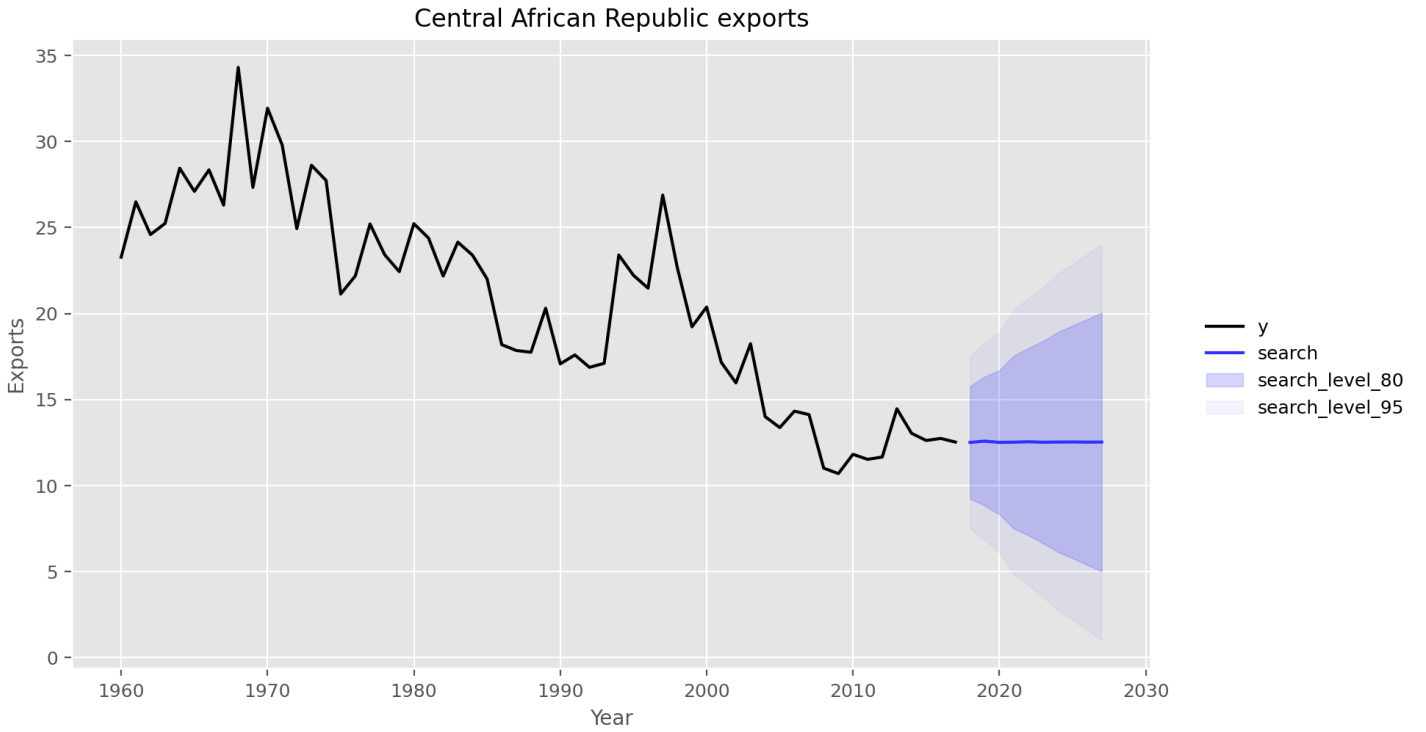

Figure 9.16: Forecasts of Central African Republic exports.

Note that the mean forecasts look very similar to what we would get with a random walk (equivalent to an ARIMA(0,1,0)). The extra work to include AR and MA terms has made little difference to the point forecasts in this example, although the prediction intervals are much narrower than for a random walk model.

Understanding constants

A non-seasonal ARIMA model can be written as (1-\phi_1B - \cdots - \phi_p B^p)(1-B)^d y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t, \tag{9.3} or equivalently as (1-\phi_1B - \cdots - \phi_p B^p)(1-B)^d (y_t - \mu t^d/d!) = (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t, \tag{9.4} where c = \mu(1-\phi_1 - \cdots - \phi_p) and \mu is the mean of (1-B)^d y_t. `StatsForecast` uses the parameterisation of Equation 9.4.

Thus, the inclusion of a constant in a non-stationary ARIMA model is equivalent to inducing a polynomial trend of order d in the forecasts. (If the constant is omitted, the forecasts include a polynomial trend of order d-1.) When d=0, we have the special case that \mu is the mean of y_t.

By default, the `ARIMA()` class sets c = \mu = 0 when d > 0 and provides an estimate of \mu when d = 0. It will be close to the sample mean of the time series but usually not identical to it, as the sample mean is not the maximum likelihood estimate when p + q > 0.

The argument `include_mean` only has an effect when d = 0 and is `True` by default. Setting `include_mean = False` will force \mu = c = 0.

The argument `include_drift` allows \mu \neq 0 when d = 1. For d > 1, no constant is allowed, as a quadratic or higher-order trend is particularly dangerous when forecasting. The parameter \mu is called the “drift” in the model output when d = 1.

There is also an argument `include_constant` which, if `True`, will set `include_mean = True` if d = 0 and `include_drift = True` when d = 1. If `include_constant = False`, both `include_mean` and `include_drift` will be set to `False`. If `include_constant` is used, the values of `include_mean = True` and `include_drift = False` are ignored.

The `AutoARIMA()` class automates the inclusion of a constant. By default, for d = 0 or d = 1, a constant will be included if it improves the AICc value; for d > 1 the constant is always omitted. If `allowdrift = False` is specified, then the constant is only allowed when d = 0.

Plotting the characteristic roots

(This is a more advanced section and can be skipped if desired.)

We can re-write Equation 9.3 as \phi(B) (1-B)^d y_t = c + \theta(B) \varepsilon_t where \phi(B)= (1-\phi_1B - \cdots - \phi_p B^p) is a pth order polynomial in B and \theta(B) = (1 + \theta_1 B + \cdots + \theta_q B^q) is a qth order polynomial in B.

The stationarity conditions for the model are that the p complex roots of \phi(B) lie outside the unit circle, and the invertibility conditions are that the q complex roots of \theta(B) lie outside the unit circle. So we can see whether the model is close to invertibility or stationarity by a plot of the roots in relation to the complex unit circle.

For the ARIMA(3,1,0) model fitted to the Central African Republic Exports, we obtain Figure 9.17.

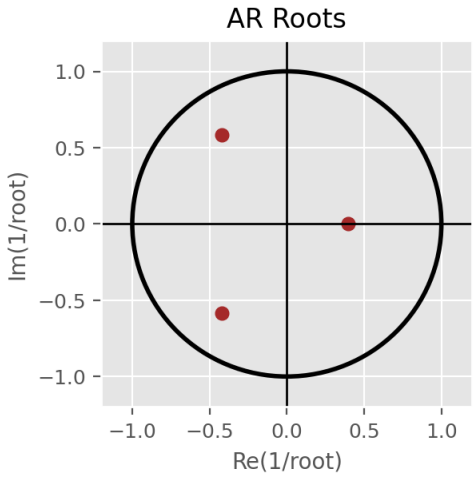

Figure 9.17: Inverse characteristic roots for the ARIMA(3,1,0) model fitted to the Central African Republic Exports.

The three orange dots in the plot correspond to the roots of the polynomials \phi(B). They are all inside the unit circle, as we would expect because `statsforecast` ensures the fitted model is both stationary and invertible. Any roots close to the unit circle may be numerically unstable, and the corresponding model will not be good for forecasting.

The `AutoARIMA()` function will never return a model with inverse roots outside the unit circle. Models automatically selected by the `AutoARIMA()` function will not contain roots close to the unit circle either. Consequently, it is sometimes possible to find a model with better AICc value than `AutoARIMA()` will return, but such models will be potentially problematic.

9.8 Forecasting

Point forecasts

Although we have calculated forecasts from the ARIMA models in our examples, we have not yet explained how they are obtained. Point forecasts can be calculated using the following three steps.

1. Expand the ARIMA equation so that y_t is on the left hand side and all other terms are on the right.
2. Rewrite the equation by replacing t with T+h.
3. On the right hand side of the equation, replace future observations with their forecasts, future errors with zero, and past errors with the corresponding residuals.

Beginning with h=1, these steps are then repeated for h=2,3,\dots until all forecasts have been calculated.

The procedure is most easily understood via an example. We will illustrate it using a ARIMA(3,1,1) model which can be written as follows: (1-\hat{\phi}_1B -\hat{\phi}_2B^2-\hat{\phi}_3B^3)(1-B) y_t = (1+\hat{\theta}_1B)\varepsilon_{t}. Then we expand the left hand side to obtain \left[1-(1+\hat{\phi}_1)B +(\hat{\phi}_1-\hat{\phi}_2)B^2 + (\hat{\phi}_2-\hat{\phi}_3)B^3 +\hat{\phi}_3B^4\right] y_t = (1+\hat{\theta}_1B)\varepsilon_{t}, and applying the backshift operator gives y_t - (1+\hat{\phi}_1)y_{t-1} +(\hat{\phi}_1-\hat{\phi}_2)y_{t-2} + (\hat{\phi}_2-\hat{\phi}_3)y_{t-3} +\hat{\phi}_3y_{t-4} = \varepsilon_t+\hat{\theta}_1\varepsilon_{t-1}. Finally, we move all terms other than y_t to the right hand side: y_t = (1+\hat{\phi}_1)y_{t-1} -(\hat{\phi}_1-\hat{\phi}_2)y_{t-2} - (\hat{\phi}_2-\hat{\phi}_3)y_{t-3} -\hat{\phi}_3y_{t-4} + \varepsilon_t+\hat{\theta}_1\varepsilon_{t-1}. \tag{9.5} This completes the first step. While the equation now looks like an ARIMA(4,0,1), it is still the same ARIMA(3,1,1) model we started with. It cannot be considered an ARIMA(4,0,1) because the coefficients do not satisfy the stationarity conditions.

For the second step, we replace t with T+1 in Equation 9.5: y_{T+1} = (1+\hat{\phi}_1)y_{T} -(\hat{\phi}_1-\hat{\phi}_2)y_{T-1} -(\hat{\phi}_2-\hat{\phi}_3)y_{T-2} -\hat{\phi}_3y_{T-3} + \varepsilon_{T+1}+\hat{\theta}_1\varepsilon_{T}. Assuming we have observations up to time T, all values on the right hand side are known except for \varepsilon_{T+1}, which we replace with zero, and \varepsilon_T, which we replace with the last observed residual e_T: \hat{y}_{T+1|T} = (1+\hat{\phi}_1)y_{T} -(\hat{\phi}_1-\hat{\phi}_2)y_{T-1} - (\hat{\phi}_2-\hat{\phi}_3)y_{T-2} -\hat{\phi}_3y_{T-3} + \hat{\theta}_1e_{T}.

A forecast of y_{T+2} is obtained by replacing t with T+2 in Equation 9.5 . All values on the right hand side will be known at time T except y_{T+1} which we replace with \hat{y}_{T+1|T}, and \varepsilon_{T+2} and \varepsilon_{T+1}, both of which we replace with zero: \hat{y}_{T+2|T} = (1+\hat{\phi}_1)\hat{y}_{T+1|T} -(\hat{\phi}_1-\hat{\phi}_2)y_{T} - (\hat{\phi}_2-\hat{\phi}_3)y_{T-1} - \hat{\phi}_3y_{T-2}.

The process continues in this manner for all future time periods. In this way, any number of point forecasts can be obtained.

Prediction intervals

The calculation of ARIMA prediction intervals is more difficult, and the details are largely beyond the scope of this book. We will only give some simple examples.

The first prediction interval is easy to calculate. If $\hat{\sigma}$ is the standard deviation of the residuals, then a 95% prediction interval is given by $\hat{y}_{T+1|T} \pm 1.96\hat{\sigma}$. This result is true for all ARIMA models regardless of their parameters and orders.

Multi-step prediction intervals for ARIMA(0,0,q) models are relatively easy to calculate. We can write the model as $y_t = \varepsilon_t + \sum_{i=1}^q \theta_i \varepsilon_{t-i}$. Then, the estimated forecast variance can be written as $\hat\sigma_h^2 = \hat{\sigma}^2 \left[1 + \sum_{i=1}^{h-1} \hat{\theta}_i^2\right], \qquad\text{for $h=2,3,\dots$,}$ where $\hat{\theta}_i=0$ for $i>q$, and a 95% prediction interval is given by $\hat{y}_{T+h|T} \pm 1.96\hat\sigma_h$.

In Section 9.4, we showed that an AR(1) model can be written as an MA(∞) model. Using this equivalence, the above result for MA(q) models can also be used to obtain prediction intervals for AR(1) models.

More general results, and other special cases of multi-step prediction intervals for an ARIMA(p,d,q) model, are given in more advanced textbooks such as Brockwell and Davis (2016).

The prediction intervals for ARIMA models are based on assumptions that the residuals are uncorrelated and normally distributed. If either of these assumptions does not hold, then the prediction intervals may be incorrect. For this reason, always plot the ACF and histogram of the residuals to check the assumptions before producing prediction intervals.

If the residuals are uncorrelated but not normally distributed, then bootstrapped intervals can be obtained instead, as discussed in Section 5.5.

In general, prediction intervals from ARIMA models increase as the forecast horizon increases. For stationary models (i.e., with d=0) they will converge, so that prediction intervals for long horizons are all essentially the same. For $d\ge1$, the prediction intervals will continue to grow into the future.

As with most prediction interval calculations, ARIMA-based intervals tend to be too narrow. This occurs because only the variation in the errors has been accounted for. There is also variation in the parameter estimates, and in the model order, that has not been included in the calculation. In addition, the calculation assumes that the historical patterns that have been modelled will continue into the forecast period.

9.9 Seasonal ARIMA models

So far, we have restricted our attention to non-seasonal data and non-seasonal ARIMA models. However, ARIMA models are also capable of modelling a wide range of seasonal data.

A seasonal ARIMA model is formed by including additional seasonal terms in the ARIMA models we have seen so far. It is written as follows:

$$\text{ARIMA} \underbrace{(p, d, q)}_{\text{Non-seasonal part of the model \, \,}} \underbrace{(P, D, Q)_m}_{\text{Seasonal part of the model}}$$

where m = the seasonal period (e.g., number of observations per year). We use uppercase notation for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model.

The seasonal part of the model consists of terms that are similar to the non-seasonal components of the model, but involve backshifts of the seasonal period. For example, an ARIMA(1,1,1)(1,1,1)$_4$ model (without a constant) is for quarterly data (m=4), and can be written as $(1 - \phi_{1}B)~(1 - \Phi_{1}B^{4}) (1 - B) (1 - B^{4})y_{t} = (1 + \theta_{1}B)~ (1 + \Theta_{1}B^{4})\varepsilon_{t}$.

The additional seasonal terms are simply multiplied by the non-seasonal terms.

ACF/PACF

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF. For example, an ARIMA(0,0,0)(0,0,1)$_{12}$ model will show:

- a spike at lag 12 in the ACF but no other significant spikes;
- exponential decay in the seasonal lags of the PACF (i.e., at lags 12, 24, 36, …).

Similarly, an ARIMA(0,0,0)(1,0,0)$_{12}$ model will show:

- exponential decay in the seasonal lags of the ACF;

- a single significant spike at lag 12 in the PACF.

In considering the appropriate seasonal orders for a seasonal ARIMA model, restrict attention to the seasonal lags.

The modelling procedure is almost the same as for non-seasonal data, except that we need to select seasonal AR and MA terms as well as the non-seasonal components of the model. The process is best illustrated via examples.

Example: Monthly US leisure and hospitality employment

We will describe seasonal ARIMA modelling using monthly US employment data for leisure and hospitality jobs from January 2001 to September 2019, shown in Figure 9.18.

```
us_employment = pd.read_csv("../data/us_employment.csv", parse_dates=["ds"])

leisure = us_employment.query(
    "unique_id == 'Leisure and Hospitality' and ds.dt.year > 2000"
).reset_index(drop=True)

leisure["y"] /= 1000
plot_series(leisure, xlabel="Month [1M]", ylabel="Number of people (millions)", title="US employme
```

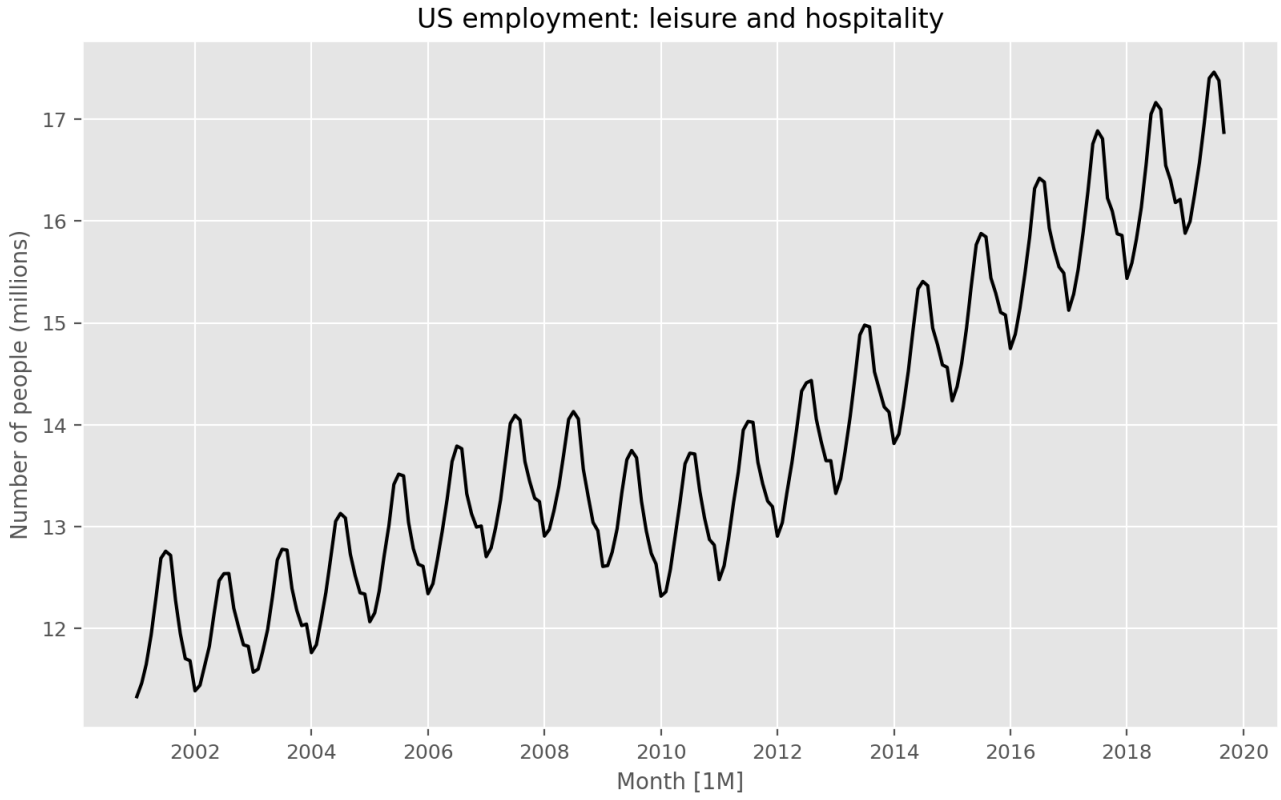

Figure 9.18: Monthly US leisure and hospitality employment, 2001-2019.

The data are clearly non-stationary, with strong seasonality and a nonlinear trend, so we will first take a seasonal difference. The seasonally differenced data are shown in Figure 9.19.

```
leisure_difference = leisure["y"].diff(12)[12:]

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(leisure_difference, marker="o")
ax1.set_ylabel("difference(Exports)")
ax1.set_xlabel("Year")

plot_acf(leisure_difference, ax2, zero=False, lags=24, bartlett_confint=False,
         auto_ylims=True)
plot_pacf(leisure_difference, ax3, zero=False, lags=24,
          auto_ylims=True)
plt.show()
```

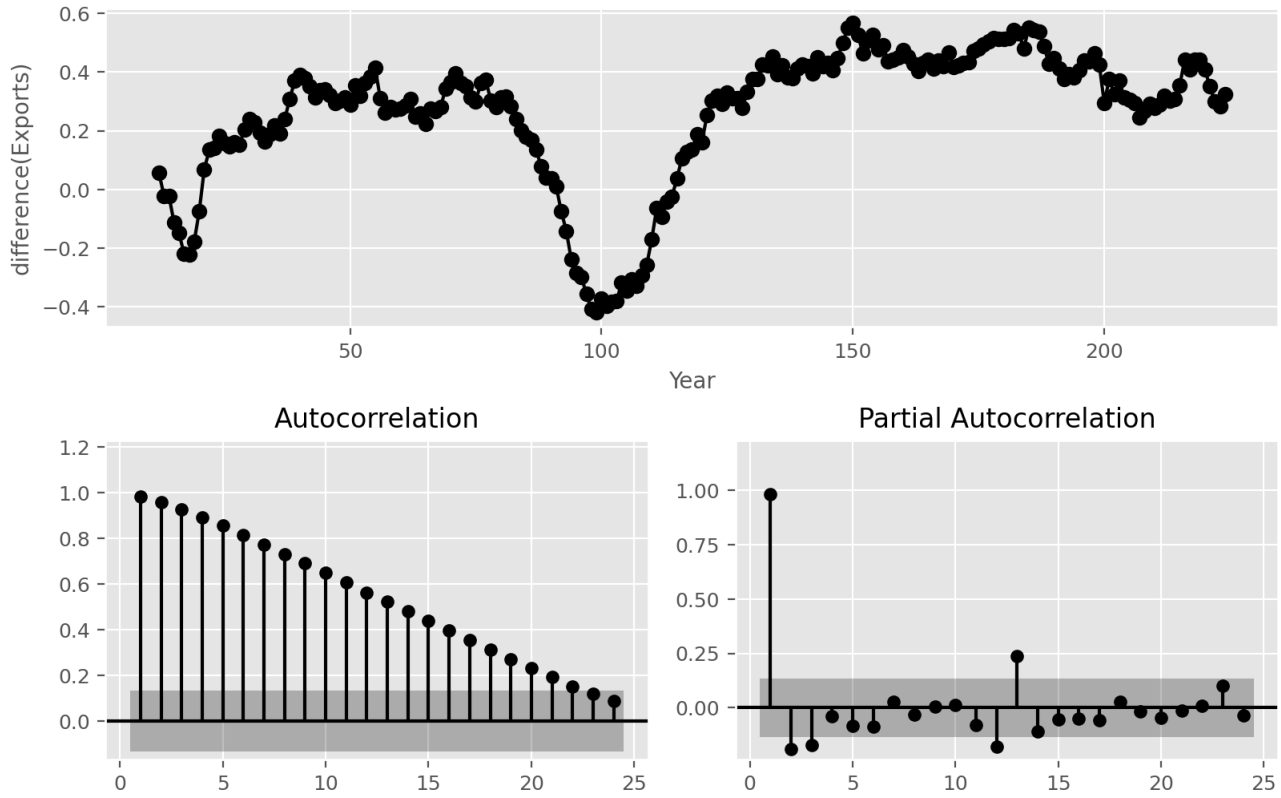

Figure 9.19: Seasonally differenced Monthly US leisure and hospitality employment.

These are also clearly non-stationary, so we take a further first difference in Figure 9.20.

```
leisure_double_difference = leisure_difference.diff()[1:]

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(leisure_double_difference, marker="o")
ax1.set_ylabel("difference(Exports)")
ax1.set_xlabel("Year")

plot_acf(leisure_double_difference, ax2, zero=False, lags=24,
         bartlett_confint=False,
         auto_ylims=True)
plot_pacf(leisure_double_difference, ax3, zero=False, lags=24,
          auto_ylims=True)
plt.show()
```

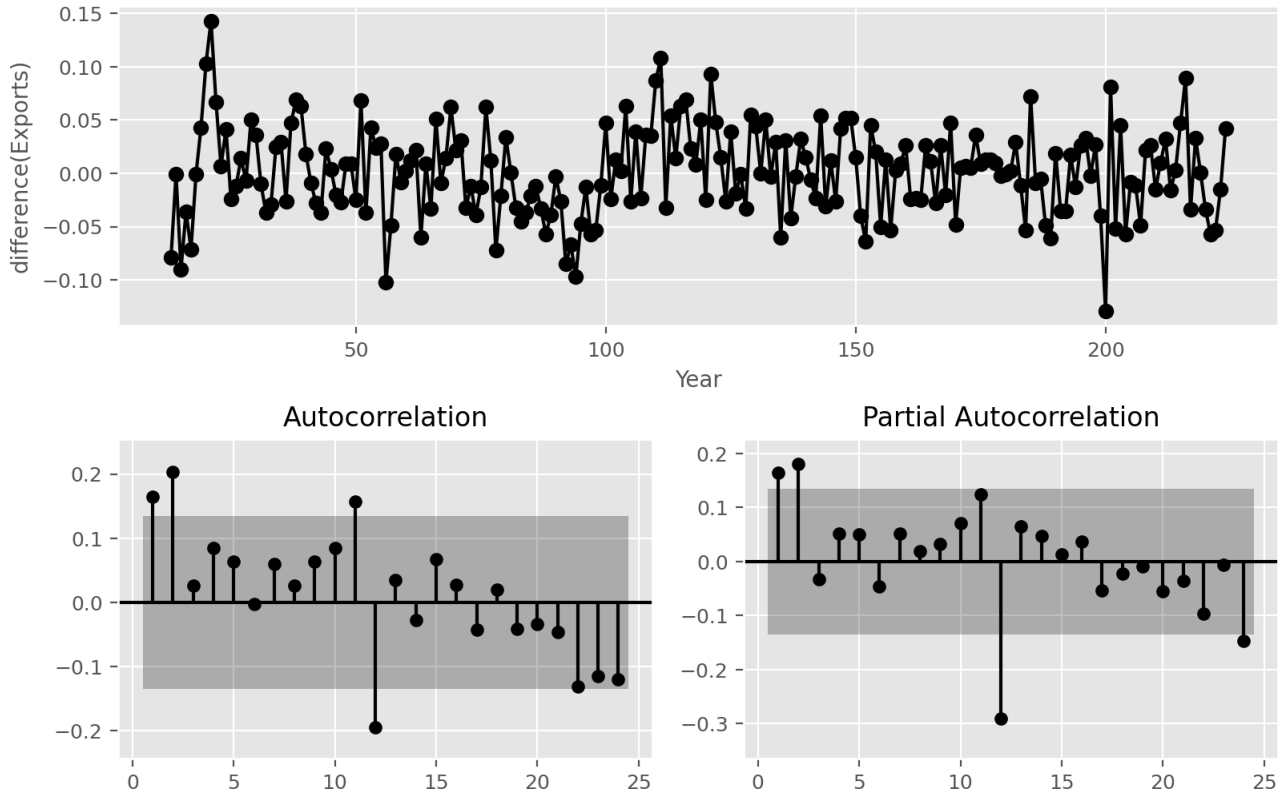

Figure 9.20: Doubly differenced Monthly US leisure and hospitality employment.

Our aim now is to find an appropriate ARIMA model based on the ACF and PACF shown in Figure 9.20. The significant spike at lag 2 in the ACF suggests a non-seasonal MA(2) component. The significant spike at lag 12 in the ACF suggests a seasonal MA(1) component. Consequently, we begin with an ARIMA(0,1,2)(0,1,1)_{12} model, indicating a first difference, a seasonal difference, and non-seasonal MA(2) and seasonal MA(1) component. If we had started with the PACF, we may have selected an ARIMA(2,1,0)(0,1,1)_{12} model — using the PACF to select the non-seasonal part of the model and the ACF to select the seasonal part of the model. We will also include an automatically selected model. By setting `stepwise=False` and `approximation=False`, we are making Python work extra hard to find a good model. This takes much longer, but with only one series to model, the extra time taken is not a problem.

```
models = [
    ARIMA(order=(0, 1, 2), seasonal_order=(0, 1, 1), season_length=12, alias="arima012011"),
    ARIMA(order=(2, 1, 0), seasonal_order=(0, 1, 1), season_length=12, alias="arima210011"),
    AutoARIMA(stepwise=False, approximation=False, alias="auto", season_length=12),
]

sf = StatsForecast(models=models, freq="MS", n_jobs=-1)

sf.fit(df=leisure[["ds", "y", "unique_id"]])

summaries = []
for model in sf.fitted_[0]:
    summary_model = {
        "model": model,
        "Orders": ARIMASummary(model.model_),
        "sigma2": model.model_["sigma2"],
        "loglik": model.model_["loglik"],
        "aic": model.model_["aic"],
        "aicc": model.model_["aicc"],
        "bic": model.model_["bic"],
    }
    summaries.append(summary_model)

pd.DataFrame(sorted(summaries, key=lambda d: d["aicc"]))
```

	model	Orders	sigma2	loglik	aic	aicc	bic
0	auto	ARIMA(2,1,0)(0,1,2)[12]	0.001	394.470	-778.940	-778.649	-762.157
1	arima210011	ARIMA(2,1,0)(0,1,1)[12]	0.001	392.020	-776.040	-775.847	-762.613
2	arima012011	ARIMA(0,1,2)(0,1,1)[12]	0.001	391.293	-774.586	-774.392	-761.159

The `AutoARIMA()` function uses `nsdiffs()` to determine D (the number of seasonal differences to use), and `ndiffs()` to determine d (the number of ordinary differences to use), when these are not specified. The selection of the other model parameters (p,q,P and Q) are all determined by minimizing the AICc, as with non-seasonal ARIMA models.

The three fitted models have similar AICc values, with the automatically selected model being a little better. Our second “guess” of \text{ARIMA}(2,1,0)(0,1,1)_{12} turned out to be very close to the automatically selected model of \text{ARIMA}(2,1,0)(0,1,2)_{12}.

The residuals for the best model are shown in Figure 9.21.

```
forecasts = sf.forecast(
    df=leisure[["ds", "y", "unique_id"]], h=36, fitted=True, level=[80, 95]
)
fitted_values = sf.forecast_fitted_values()
insample_forecasts = fitted_values["auto"]
residuals = fitted_values["y"] - insample_forecasts

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(leisure["ds"], residuals, marker="o")
ax1.set_ylabel("Innovation Residuals")
ax1.set_xlabel("Year")

plot_acf(residuals, ax2, zero=False,
         bartlett_confint=False,
         auto_ylims=True)

ax3.hist(residuals, bins=20)
ax3.set_ylabel("count")
ax3.set_xlabel("residuals")

plt.show()
```

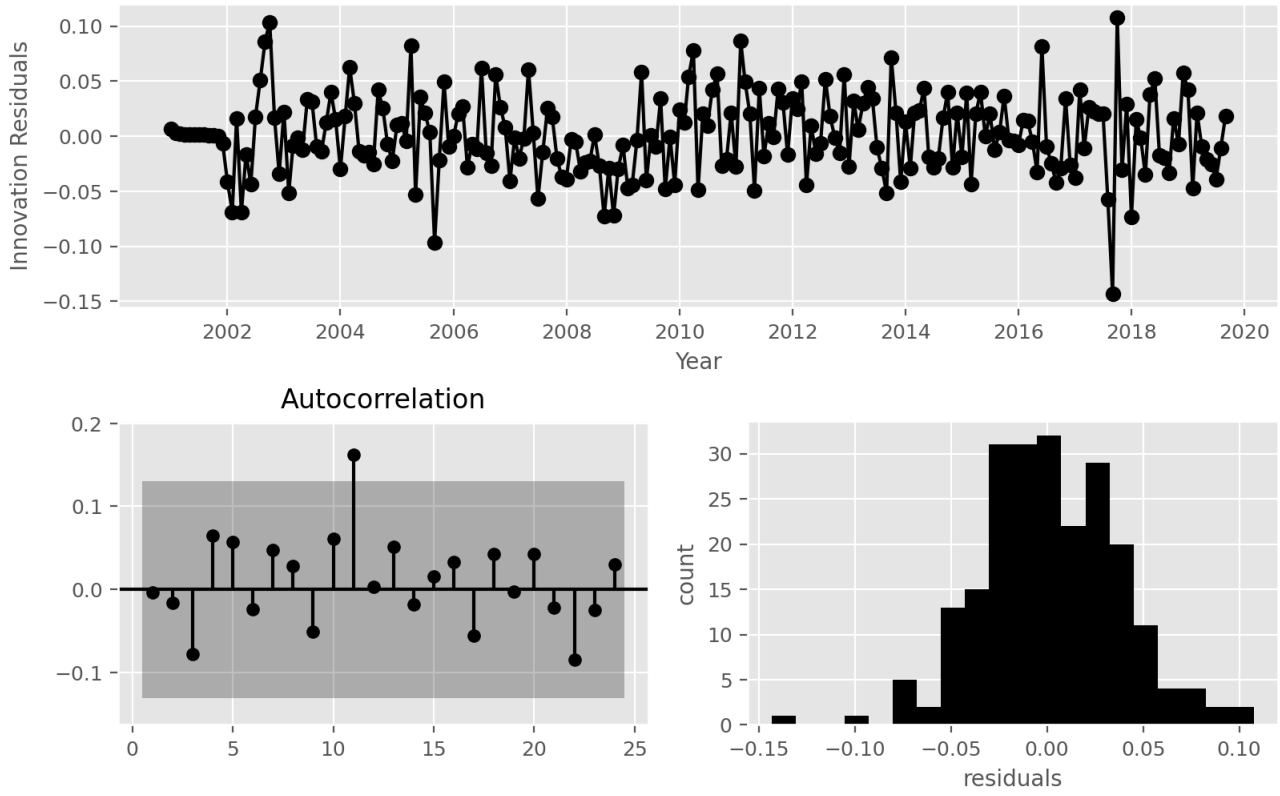

Figure 9.21: Residuals from the fitted ARIMA(2,1,0)(1,1,1)_12 model.

One small but significant spike (at lag 11) out of 36 is still consistent with white noise. To be sure, we use a Ljung-Box test, being careful to set the degrees of freedom to match the number of parameters in the model.

```
ljung_box = acorr_ljungbox(residuals, lags=[24], model_df=4)

ljung_box
```

	lb_stat	lb_pvalue
24	16.808	0.665

The large p-value confims that the residuals are similar to white noise.

Thus, we now have a seasonal ARIMA model that passes the required checks and is ready for forecasting. Forecasts from the model for the next three years are shown in Figure 9.22. The forecasts have captured the seasonal pattern very well, and the increasing trend extends the recent pattern. The trend in the forecasts is induced by the double differencing.

```
plot_series(
    df=leisure,
    forecasts_df=forecasts.reset_index()[
        [
            "unique_id",
            "ds",
            "auto",
            "auto-lo-80",
            "auto-lo-95",
            "auto-hi-80",
            "auto-hi-95",
        ]
    ],
    level=[80, 95],
    rm_legend=False,
    xlabel="Month [1M]", ylabel="Number of people (millions)", title="US employment: leisure and h

)
```

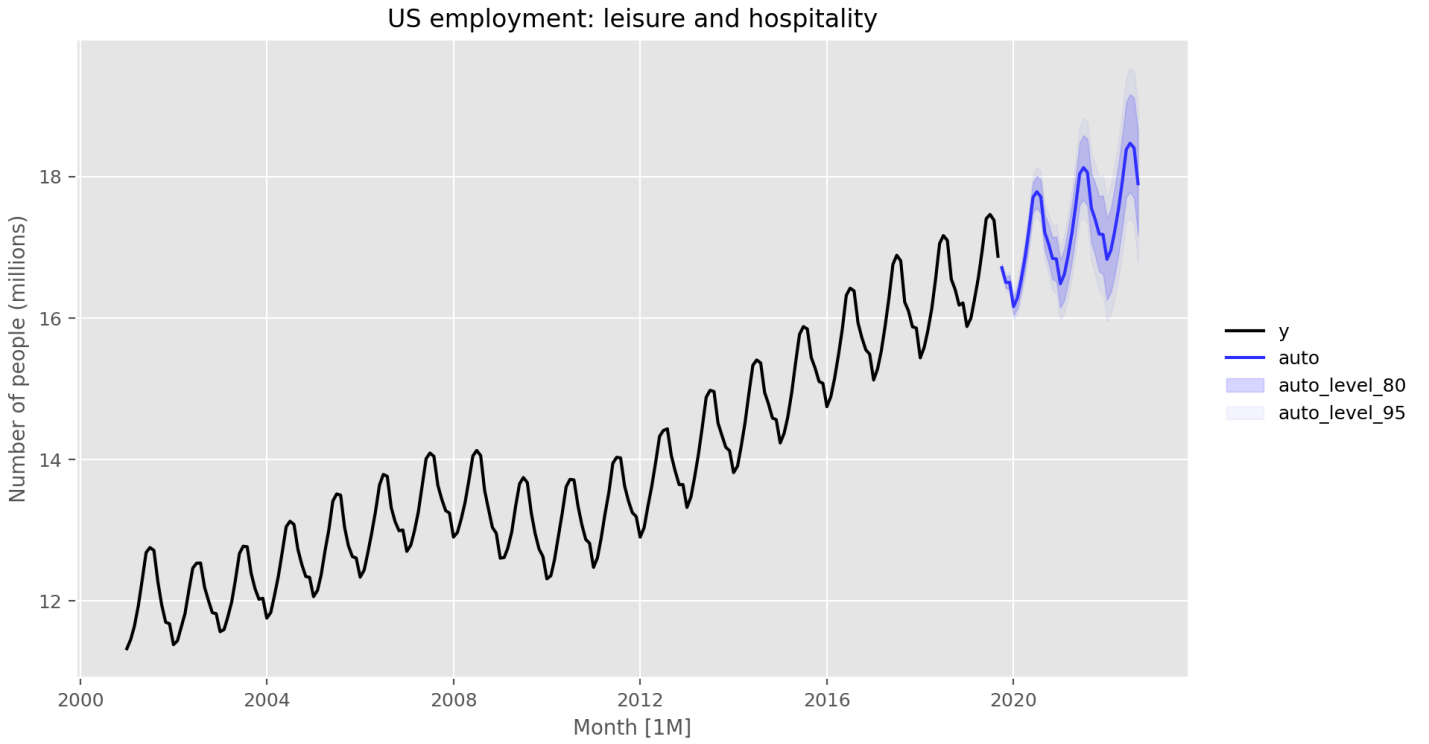

Figure 9.22: Forecasts of monthly US leisure and hospitality employment using the ARIMA(2,1,0)(1,1,1)_12model. 80% and 95% prediction intervals are shown.

Example: Corticosteroid drug sales in Australia

For our second example, we will try to forecast monthly corticosteroid drug sales in Australia. These are known as H02 drugs under the Anatomical Therapeutic Chemical classification scheme.

```
pbs = pd.read_csv("../data/PBS_unparsed.csv", parse_dates=["Month"])
pbs = pbs.rename(columns={"Month": "ds"})
pbs = pbs.query("ATC2 == 'H02'").reset_index(drop=True)
pbs["Cost"] /= 1e6
pbs = pbs.groupby(["ds"])["Cost"].sum().reset_index()
pbs["Log(cost)"] = np.log(pbs["Cost"])
pbs["unique_id"] = "Corticosteroid drug scripts (H02)"
pbs = pbs.set_index(["ds", "unique_id"]).stack().reset_index()
pbs.columns = ["ds", "drug_type", "unique_id", "y"]
pbs = pbs.drop(columns="drug_type")
_, axes = plt.subplots(ncols=1, nrows=2, figsize=(8, 7))
fig = plot_series_utils(pbs, ax=axes)
for ax in fig.axes:
    ax.tick_params(axis='both')
    if ax.get_legend():
        ax.get_legend().remove()
    for label in ax.get_xticklabels():
        label.set_rotation(0)
fig.legends = []
fig.axes[1].set_xlabel("Month")
fig.suptitle("Corticosteroid drug scripts (H02)", x=0.525)
fig
```

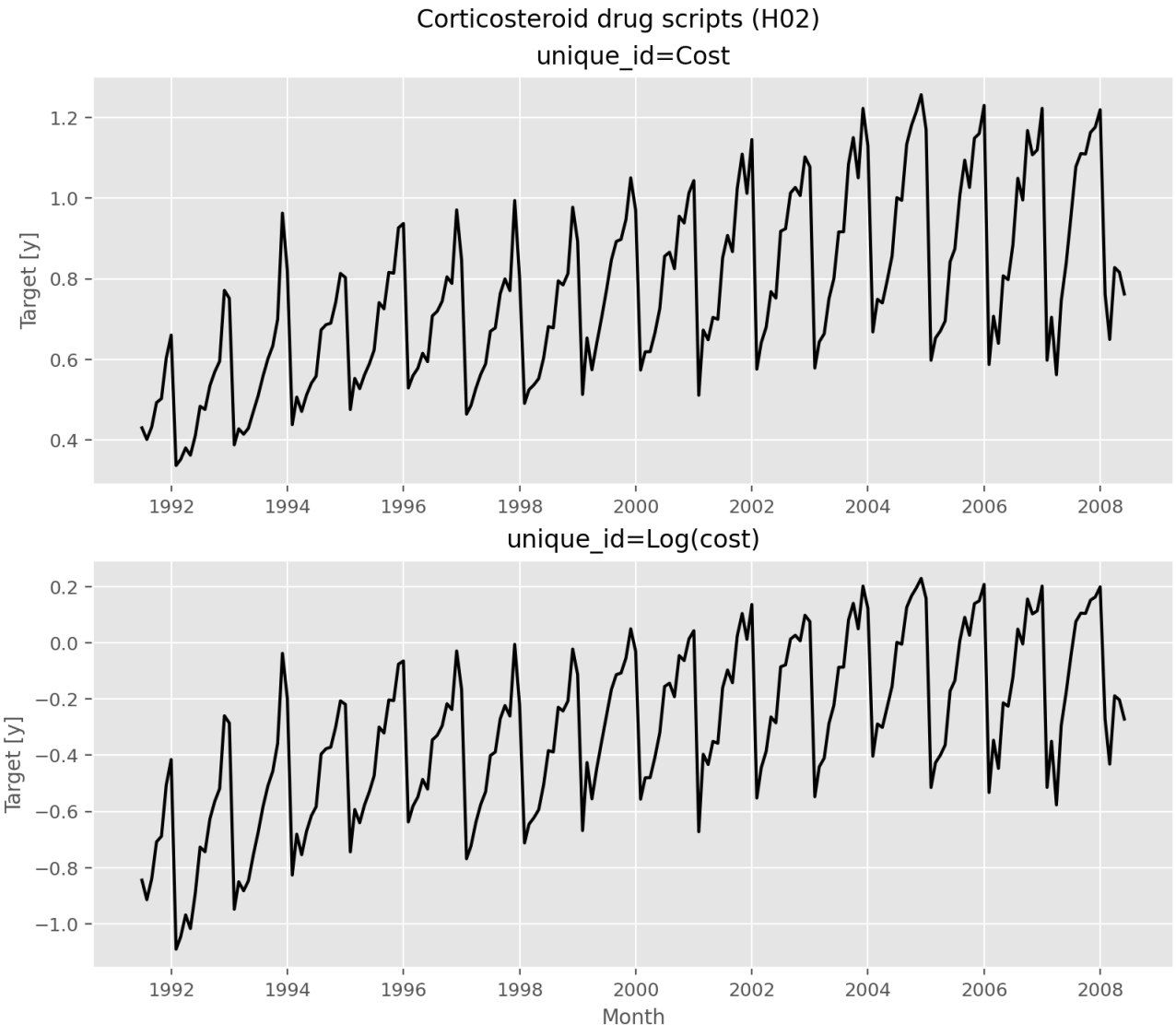

Figure 9.23: Corticosteroid drug sales in Australia (in millions of scripts per month). Logged data shown in right panel.

Data from July 1991 to June 2008 are plotted in Figure 9.23. There is a small increase in the variance with the level, so we take logarithms to stabilise the variance.

The data are strongly seasonal and obviously non-stationary, so seasonal differencing will be used. The seasonally differenced data are shown in Figure 9.24. It is not clear at this point whether we should do another difference or not. We decide not to, but the choice is not obvious.

The last few observations appear to be different (more variable) from the earlier data. This may be due to the fact that data are sometimes revised when earlier sales are reported late.

```
log_cost = pbs.query("unique_id == 'Log(cost)'").reset_index(drop=True)
difference_log_cost = log_cost["y"].diff(12)[12:]

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(difference_log_cost, marker="o")
ax1.set_ylabel("difference(log(Cost))")
ax1.set_xlabel("Year")

plot_acf(difference_log_cost, ax2, zero=False,
         bartlett_confint=False,
         auto_ylims=True)
plot_pacf(difference_log_cost, ax3, zero=False,
         auto_ylims=True)
plt.show()
```

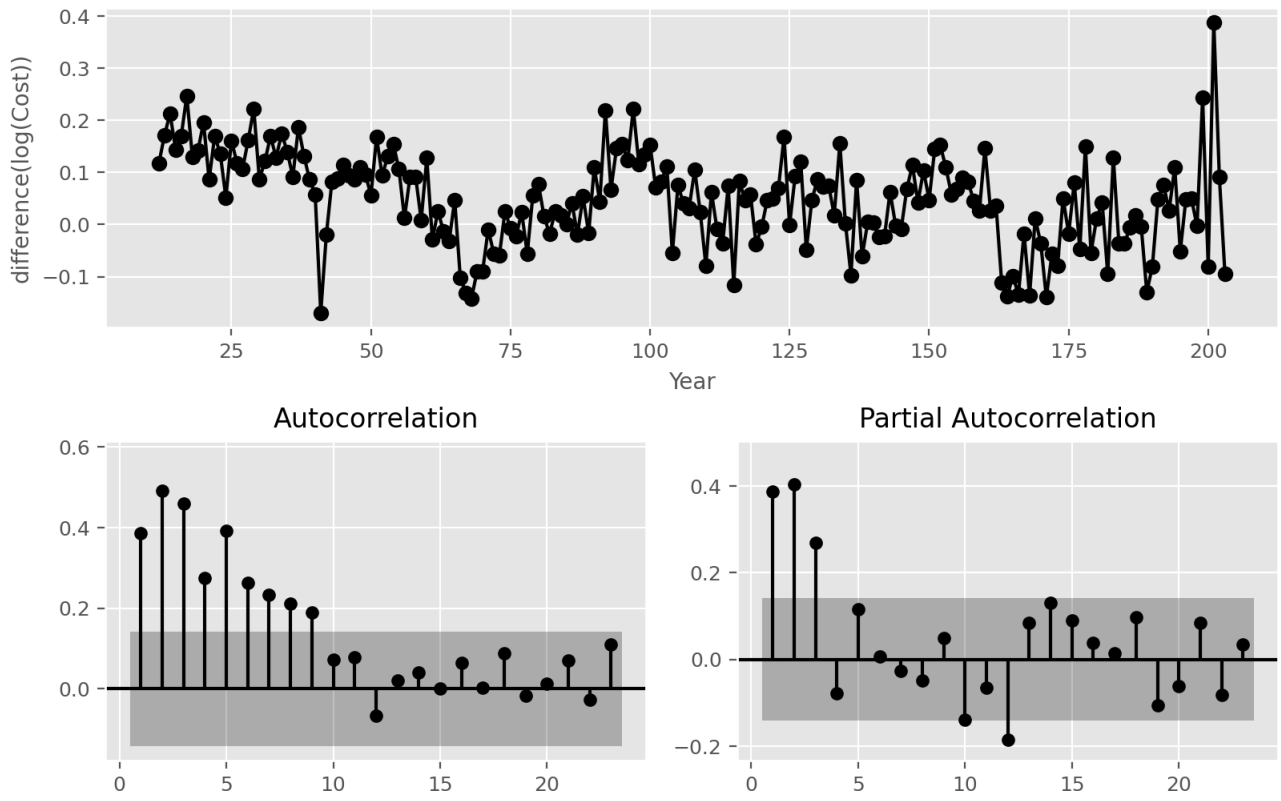

Figure 9.24: Seasonally differenced corticosteroid drug sales in Australia (in millions of scripts per month).

In the plots of the seasonally differenced data, there are spikes in the PACF at lags 12 and 24, but nothing at seasonal lags in the ACF. This may be suggestive of a seasonal AR(2) term. In the non-seasonal lags, there are three significant spikes in the PACF, suggesting a possible AR(3) term. The pattern in the ACF is not indicative of any simple model.

Consequently, this initial analysis suggests that a possible model for these data is an ARIMA(3,0,0)(2,1,0)_{12}. We fit this model, along with some variations on it, and compute the AICc values shown in Table 9.1.

Table 9.1: AICc values for various ARIMA models applied for H02 monthly script sales data.

	model	Orders	aicc
0	arima301012	ARIMA(3,0,1)(0,1,2)[12]	-482.842
1	arima301011	ARIMA(3,0,1)(0,1,1)[12]	-481.147
2	arima301111	ARIMA(3,0,1)(1,1,1)[12]	-479.318
3	arima301210	ARIMA(3,0,1)(2,1,0)[12]	-471.765
4	arima302210	ARIMA(3,0,2)(2,1,0)[12]	-469.914
5	arima300210	ARIMA(3,0,0)(2,1,0)[12]	-469.642
6	arima301110	ARIMA(3,0,1)(1,1,0)[12]	-462.217

Of these models, the best is the ARIMA(3,0,1)(0,1,2)_{12} model (i.e., it has the smallest AICc value). The innovation residuals from this model are shown in Figure 9.25.

```
forecasts = sf.forecast(df=log_cost, h=24, fitted=True, level=[80, 95])
fitted_values = sf.forecast_fitted_values()
insample_forecasts = fitted_values["arima301012"]
residuals = fitted_values["y"] - insample_forecasts

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(log_cost["ds"], residuals, marker="o")
ax1.set_ylabel("Innovation Residuals")
ax1.set_xlabel("Year")

plot_acf(residuals, ax2, zero=False, lags=36,
         bartlett_confint=False,
         auto_ylims=True)

ax3.hist(residuals, bins=20)
ax3.set_ylabel("count")
ax3.set_xlabel("residuals")

plt.show()
```

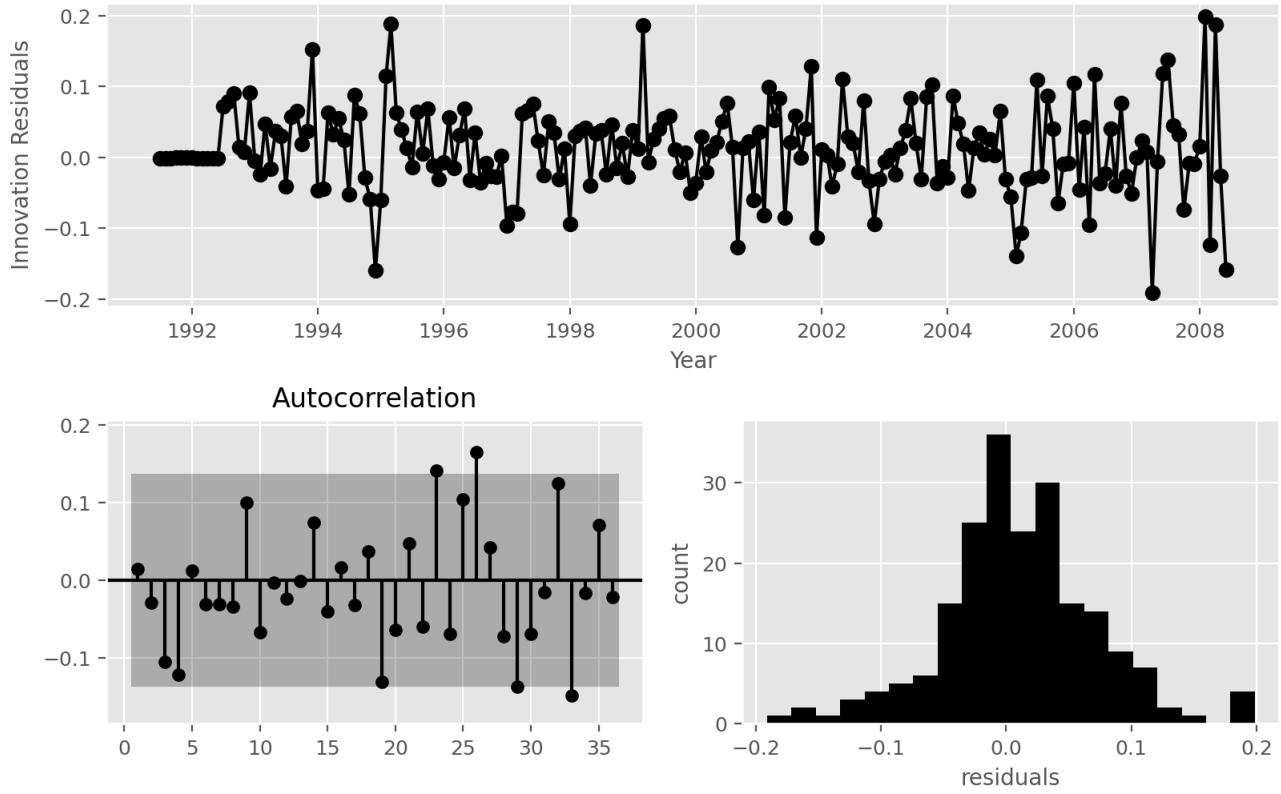

Figure 9.25: Innovation residuals from the ARIMA(3,0,1)(0,1,2)_12 model applied to the H02 monthly script sales data.

```
ljung_box = acorr_ljungbox(residuals, lags=[36], model_df=6)

ljung_box
```

	lb_stat	lb_pvalue
36	50.631	0.011

There are a few significant spikes in the ACF, and the model fails the Ljung-Box test. The model can still be used for forecasting, but the prediction intervals may not be accurate due to the correlated residuals.

Next we will try using the automatic ARIMA algorithm. Running `AutoARIMA()` with all arguments left at their default values led to an ARIMA(2,1,0)(0,1,1)_{12}. model. Running `ARIMA()` with `stepwise=False` and `approximation=False` gives an ARIMA(2,1,3)(0,1,1)_{12} model. However, both models still fail the Ljung-Box test for 36 lags. Sometimes it is just not possible to find a model that passes all of the tests.

Test set evaluation:

We will compare some of the models fitted so far using a test set consisting of the last two years of data. Thus, we fit the models using data from July 1991 to June 2006, and forecast the script sales for July 2006 – June 2008. The results are summarised in Table 9.2.

Table 9.2: RMSE values for various ARIMA models applied for H02 monthly script sales data over test set July 2006 – June 2008.

	model	Orders	rmse
0	arima302210	ARIMA(3,0,2)(2,1,0)[12]	0.242
1	arima301210	ARIMA(3,0,1)(2,1,0)[12]	0.242
2	arima300210	ARIMA(3,0,0)(2,1,0)[12]	0.242
3	arima301012	ARIMA(3,0,1)(0,1,2)[12]	0.244
4	arima213011	ARIMA(2,1,3)(0,1,1)[12]	0.245
5	arima214011	ARIMA(2,1,4)(0,1,1)[12]	0.246
6	arima212011	ARIMA(2,1,2)(0,1,1)[12]	0.246
7	arima211011	ARIMA(2,1,1)(0,1,1)[12]	0.246
8	arima210011	ARIMA(2,1,0)(0,1,1)[12]	0.247
9	arima210010	ARIMA(2,1,0)(0,1,0)[12]	0.249
10	arima303011	ARIMA(3,0,3)(0,1,1)[12]	0.249
11	arima302011	ARIMA(3,0,2)(0,1,1)[12]	0.251
12	arima301011	ARIMA(3,0,1)(0,1,1)[12]	0.251
13	arima301111	ARIMA(3,0,1)(1,1,1)[12]	0.251
14	arima301110	ARIMA(3,0,1)(1,1,0)[12]	0.261
15	arima210110	ARIMA(2,1,0)(1,1,0)[12]	0.262

The models chosen manually are close to the best model over this test set based on the RMSE values, while those models chosen automatically with `AutoARIMA()` are not far behind.

When models are compared using AICc values, it is important that all models have the same orders of differencing. However, when comparing models using a test set, it does not matter how the forecasts were produced — the comparisons are always valid. Consequently, in the table above, we can include some models with only seasonal differencing and some models with both first and seasonal differencing, while in the earlier table containing AICc values, we only compared models with seasonal differencing but no first differencing.

None of the models considered here pass all of the residual tests. In practice, we would normally use the best model we could find, even if it did not pass all of the tests.

Forecasts from the ARIMA(3,0,1)(0,1,2)_{12} model are shown in Figure 9.26.

```
arima_cols = [
    "arima301012",
    "arima301012-lo-80",
    "arima301012-lo-95",
    "arima301012-hi-80",
    "arima301012-hi-95",
]
original_fcst = forecasts.reset_index()[
        [
            "ds",
            "unique_id",
        ] + arima_cols
    ]
for col in arima_cols:
    original_fcst[col] = np.exp(original_fcst[col])
plot_series(
    df=log_cost[:-24].assign(y = lambda df: np.exp(df["y"])),
    forecasts_df=original_fcst,
    level=[80, 95],
    xlabel="Month",
    ylabel="$AU (millions)",
    title="Corticosteroid drug scripts (H02) sales",
    rm_legend=False,
)
```

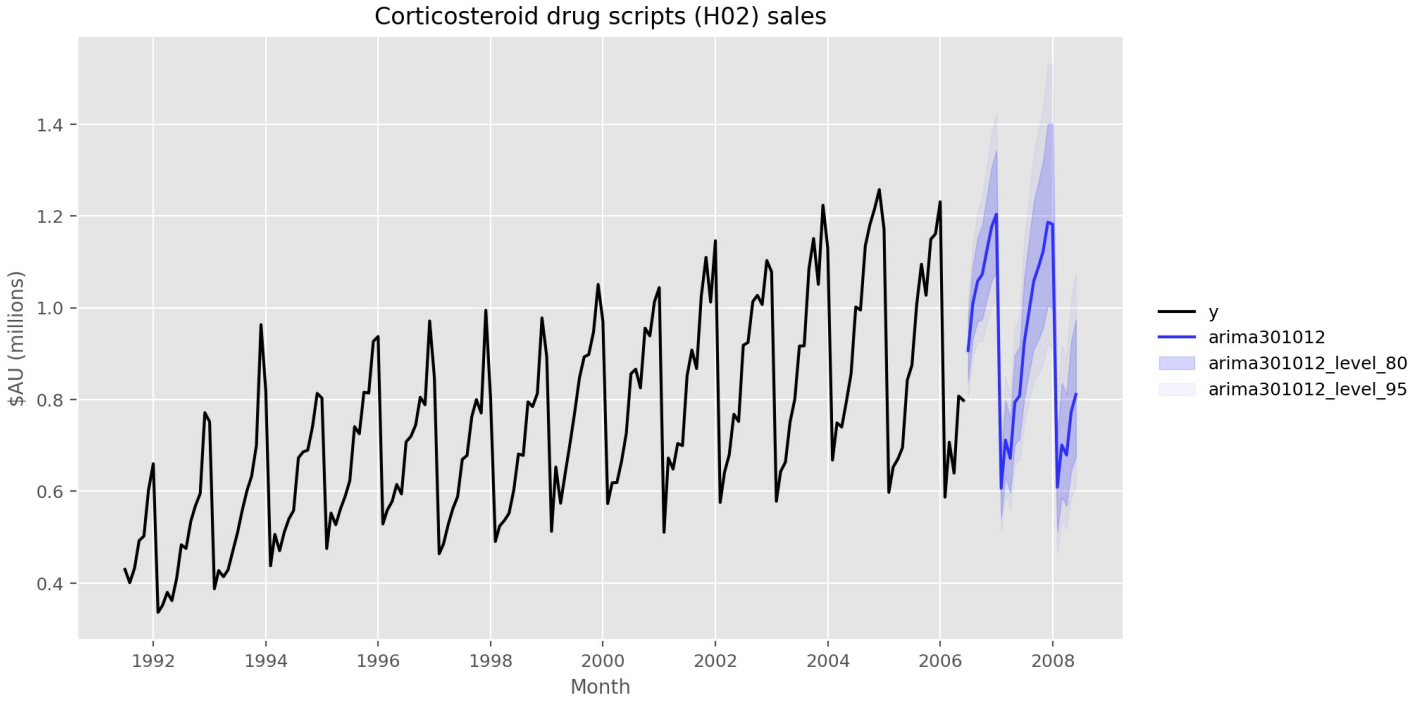

Figure 9.26: Forecasts from the ARIMA(3,0,1)(0,1,2)_12 model applied to the H02 monthly script sales data.

9.10 ARIMA vs ETS

It is a commonly held myth that ARIMA models are more general than exponential smoothing. While linear exponential smoothing models are all special cases of ARIMA models, the non-linear exponential smoothing models have no equivalent ARIMA counterparts. On the other hand, there are also many ARIMA models that have no exponential smoothing counterparts. In particular, all ETS models are non-stationary, while some ARIMA models are stationary. Figure 9.27 shows the overlap between the two model classes.

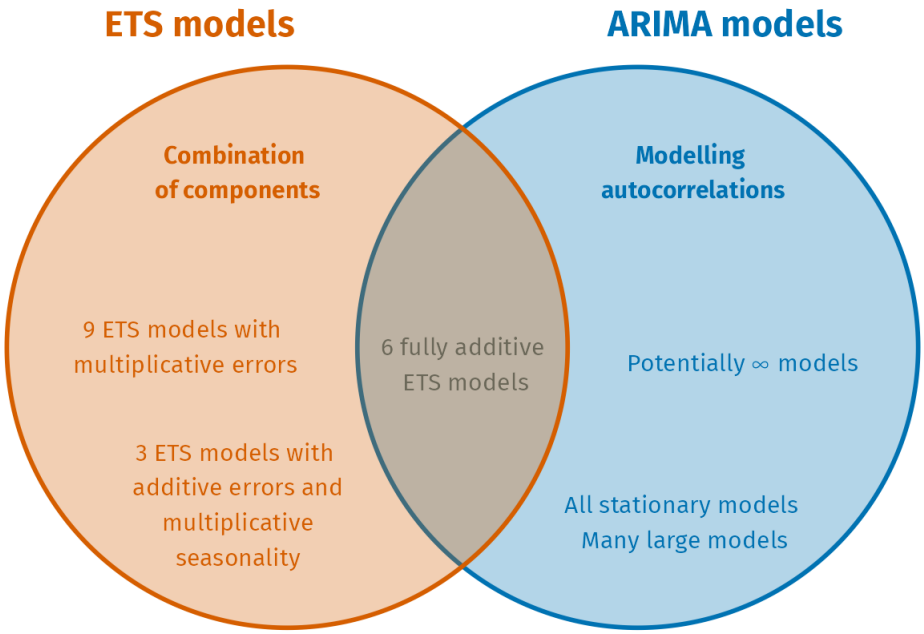

Figure 9.27: The ETS and ARIMA model classes overlap with the additive ETS models having equivalent ARIMA forms.

The ETS models with seasonality or non-damped trend or both have two unit roots (i.e., they need two levels of differencing to make them stationary). All other ETS models have one unit root (they need one level of differencing to make them stationary).

Figure 9.28 gives the equivalence relationships for the two classes of models. For the seasonal models, the ARIMA parameters have a large number of restrictions.

ETS model	ARIMA model	Parameters
ETS(A,N,N)	ARIMA(0,1,1)	$\theta_1 = \alpha - 1$
ETS(A,A,N)	ARIMA(0,2,2)	$\theta_1 = \alpha + \beta - 2$ $\theta_2 = 1 - \alpha$
ETS(A,A_d,N)	ARIMA(1,1,2)	$\phi_1 = \phi$ $\theta_1 = \alpha + \phi\beta - 1 - \phi$ $\theta_2 = (1 - \alpha)\phi$
ETS(A,N,A)	ARIMA(0,1,m)(0,1,0)$_m$	
ETS(A,A,A)	ARIMA(0,1,$m+1$)(0,1,0)$_m$	
ETS(A,A_d,A)	ARIMA(1,0,$m+1$)(0,1,0)$_m$	

Figure 9.28: Equivalence relationships between ETS and ARIMA models.

The AICc is useful for selecting between models in the same class. For example, we can use it to select an ARIMA model between candidate ARIMA models[3] or an ETS model between candidate ETS models. However, it cannot be used to compare between ETS and ARIMA models because they are in different model classes, and the likelihood is computed in different ways. The examples below demonstrate selecting between these classes of models.

Comparing `AutoARIMA()` and `AutoETS()` on non-seasonal data

We can use time series cross-validation to compare ARIMA and ETS models. Let's consider the Australian population from the `global_economy` dataset, as introduced in Section 8.2.

```
aus_economy = pd.read_csv("../data/aus_economy.csv", parse_dates=["ds"])
aus_economy["y"] /= 1e6

models = [AutoARIMA(), AutoETS()]

sf = StatsForecast(models=models, freq="A", n_jobs=-1)

cv_forecasts = sf.cross_validation(h=1, n_windows=10, df=aus_economy)

error_metrics = [rmse, mae, smape]
errors = pd.DataFrame()
for error_metric in error_metrics:
    error = error_metric(cv_forecasts.reset_index(), models=["AutoARIMA", "AutoETS"])
    error.rename(index={0: error_metric.__name__}, inplace=True)
    errors = pd.concat((errors, error))

errors.drop(columns="unique_id").T
```

	rmse	mae	smape
AutoARIMA	0.082	0.060	0.001
AutoETS	0.131	0.113	0.003

In this case the AutoARIMA model has higher accuracy on the cross-validated performance measures. Below we generate and plot forecasts for the next 5 years generated from an ARIMA model.

```
models = [AutoARIMA()]

sf = StatsForecast(models=models, freq="A", n_jobs=-1)

forecasts = sf.forecast(h=5, df=aus_economy, level=[80, 95])

plot_series(
    aus_economy, forecasts, level=[80, 95], max_insample_length=20,
    xlabel="Year",
    ylabel="People (millions)",
    title="Australian population",
    rm_legend=False,
)
```

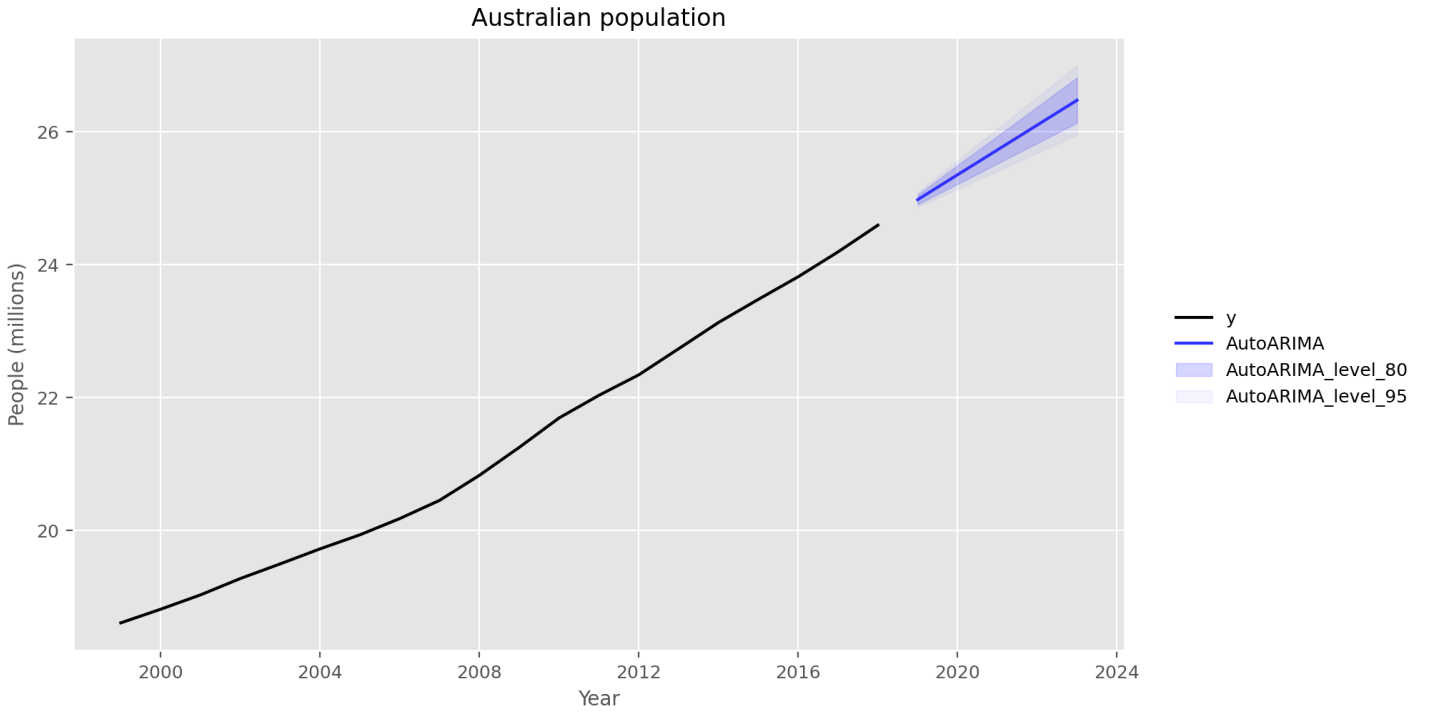

Figure 9.29: Forecasts from an ETS model fitted to the Australian population.

Comparing `AutoARIMA()` and `AutoETS()` on seasonal data

In this case we want to compare seasonal ARIMA and ETS models applied to the quarterly cement production data (from `aus_production`). Because the series is relatively long, we can afford to use a training and a test set rather than time series cross-validation. The advantage is that this is much faster. We create a training set from the beginning of 1988 to the end of 2007 and select an ARIMA and an ETS model using the `AutoARIMA()` and `AutoETS()` functions.

```
aus_production = pd.read_csv("../data/aus_production.csv", parse_dates=["ds"])
cement = aus_production.query("ds.dt.year >= 1988")[["ds", "Cement"]]
cement.insert(0, "unique_id", "Cement")
cement.rename(columns={"Cement": "y"}, inplace=True)

cement_train = cement.query("ds.dt.year < 2008").reset_index(drop=True)
```

The output below shows the model selected and estimated by `AutoARIMA()`. The ARIMA model does well in capturing all the dynamics in the data as the residuals seem to be white noise.

```
models = [AutoARIMA(season_length=4, allowdrift=True)]

sf = StatsForecast(models=models, freq="Q", n_jobs=-1)

sf.fit(df=cement_train)

print(ARIMASummary(sf.fitted_[0, 0].model_))
print(f"Coefficients: {sf.fitted_[0, 0].model_['coef']}")
print(f"sigma^2     : {sf.fitted_[0, 0].model_['sigma2']:.2f}")
print(f"loglik      : {sf.fitted_[0, 0].model_['loglik']:.2f}")
print(f"aic         : {sf.fitted_[0, 0].model_['aic']:.2f}")
print(f"aicc        : {sf.fitted_[0, 0].model_['aicc']:.2f}")
print(f"bic         : {sf.fitted_[0, 0].model_['bic']:.2f}")
```

```
ARIMA(1,0,0)(1,1,2)[4] with drift
Coefficients: {'ar1': 0.7813013118289077, 'sar1': -0.5071077342582854, 'sma1': -0.08277598835791783, 'sma2': -0
.6241194101203863, 'drift': 9.338609534785933}
sigma^2     : 12314.39
loglik      : -465.49
aic         : 942.98
aicc        : 944.20
bic         : 956.97
```

```
forecasts = sf.forecast(df=cement_train, h=10, fitted=True)
fitted_values = sf.forecast_fitted_values()
insample_forecasts = fitted_values["AutoARIMA"]
residuals = fitted_values["y"] - insample_forecasts

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(cement_train["ds"], residuals, marker="o")
ax1.set_ylabel("Innovation Residuals")
ax1.set_xlabel("Year")

plot_acf(residuals, ax2, zero=False,
        bartlett_confint=False,
        auto_ylims=True)

ax3.hist(residuals, bins=20)
ax3.set_ylabel("count")
ax3.set_xlabel("residuals")
ax2.xaxis.set_major_locator(MaxNLocator(integer=True))

plt.show()
```

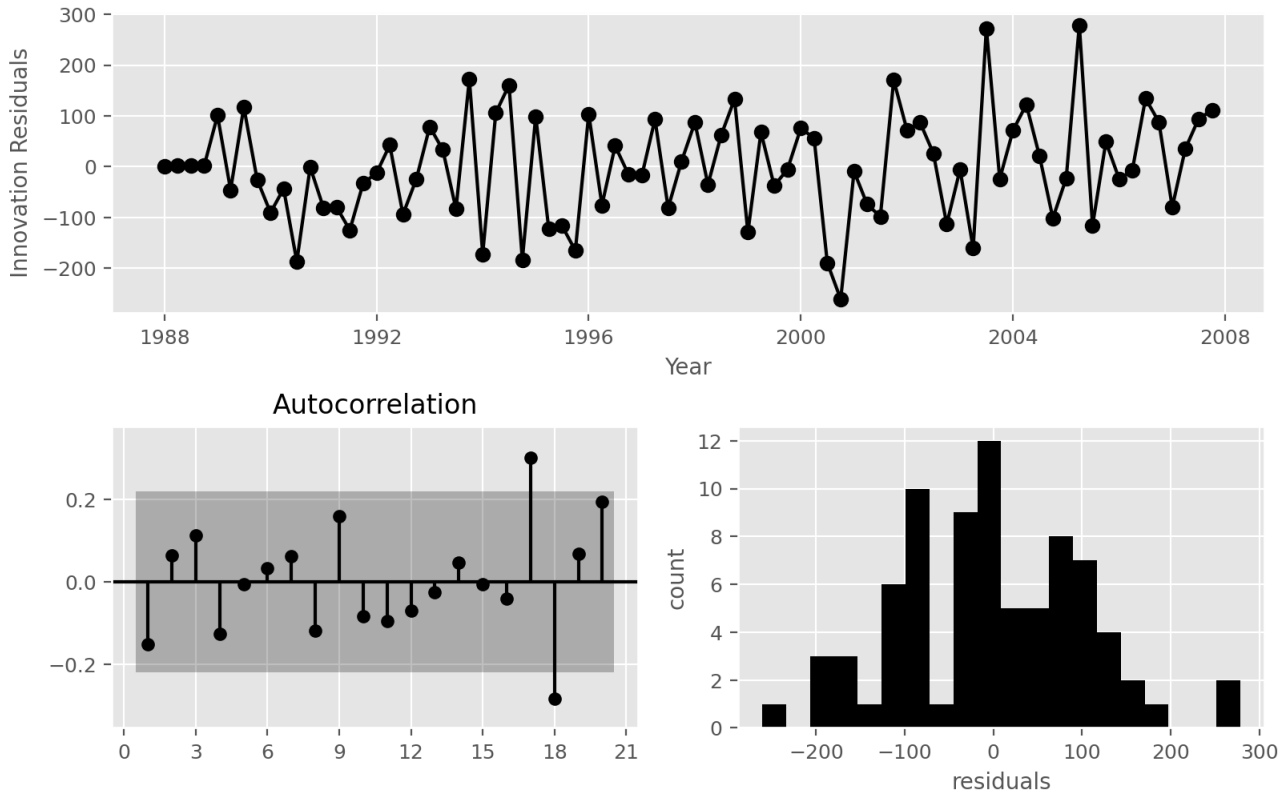

Figure 9.30: Residual diagnostic plots for the ARIMA model fitted to the quarterly cement production training data.

```
ljung_box = acorr_ljungbox(residuals, lags=[16], model_df=5)

ljung_box
```

	lb_stat	lb_pvalue
16	11.177	0.429

The output below also shows the ETS model selected and estimated by `AutoETS()`. This model also does well in capturing all the dynamics in the data, as the residuals similarly appear to be white noise.

```
models = [AutoETS(season_length=4)]

sf = StatsForecast(models=models, freq="Q", n_jobs=-1)

sf.fit(df=cement_train)

print(f"sigma^2     : {sf.fitted_[0, 0].model_['sigma2']:.4f}")
print(f"loglik      : {sf.fitted_[0, 0].model_['loglik']:.2f}")
print(f"aic         : {sf.fitted_[0, 0].model_['aic']:.2f}")
print(f"aicc        : {sf.fitted_[0, 0].model_['aicc']:.2f}")
print(f"bic         : {sf.fitted_[0, 0].model_['bic']:.2f}")
```

```
sigma^2     : 0.0034
loglik      : -544.10
aic         : 1102.21
aicc        : 1103.76
bic         : 1118.88
```

```
forecasts = sf.forecast(df=cement_train, h=10, fitted=True)
fitted_values = sf.forecast_fitted_values()
insample_forecasts = fitted_values["AutoETS"]
residuals = fitted_values["y"] - insample_forecasts

fig = plt.figure()

gs = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, 0])
ax3 = fig.add_subplot(gs[1, 1])

ax1.plot(cement_train["ds"], residuals, marker="o")
ax1.set_ylabel("Innovation Residuals")
ax1.set_xlabel("Year")

plot_acf(residuals, ax2, zero=False,
         bartlett_confint=False,
         auto_ylims=True)

ax3.hist(residuals, bins=20)
ax3.set_ylabel("count")
ax3.set_xlabel("residuals")
ax2.xaxis.set_major_locator(MaxNLocator(integer=True))

plt.show()
```

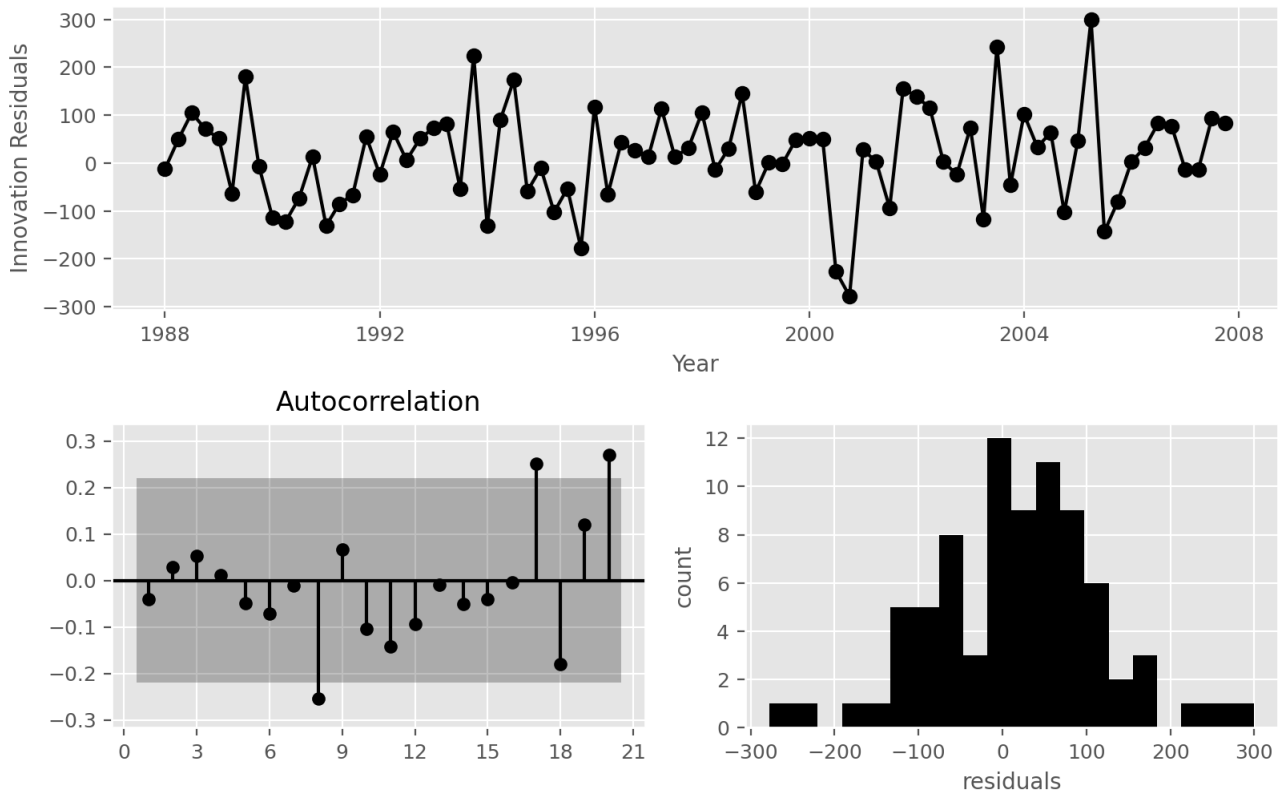

Figure 9.31: Residual diagnostic plots for the ETS model fitted to the quarterly cement production training data.

```
ljung_box = acorr_ljungbox(residuals, lags=[16])

ljung_box
```

	lb_stat	lb_pvalue
16	11.644	0.768

The output below evaluates the forecasting performance of the two competing models over the test set. In this case the ARIMA model seems to be the slightly more accurate model based on the test set RMSE, MAPE and MASE.

```
models = [AutoARIMA(season_length=4), AutoETS(season_length=4)]

sf = StatsForecast(models=models, freq="Q", n_jobs=-1)

cv_forecasts = sf.cross_validation(h=10, n_windows=1, df=cement)

error_metrics = [rmse, mae, smape]
errors = pd.DataFrame()
for error_metric in error_metrics:
    error = error_metric(cv_forecasts.reset_index(), models=["AutoARIMA", "AutoETS"])
    error.rename(index={0: error_metric.__name__}, inplace=True)
    errors = pd.concat((errors, error))

errors.drop(columns="unique_id").T
```

	rmse	mae	smape
AutoARIMA	195.204	169.078	0.037
AutoETS	222.180	191.152	0.042

Below we generate and plot forecasts from the ARIMA model for the next 3 years.

```
models = [AutoARIMA(season_length=4)]

sf = StatsForecast(models=models, freq="Q", n_jobs=-1)

forecasts = sf.forecast(h=12, df=cement, level=[80, 95])

plot_series(
    cement, forecasts, level=[80, 95],
    xlabel="Quarter",
    ylabel="Tonnes ('000)",
    title="Cement production in Australia",
    rm_legend=False,
)
```

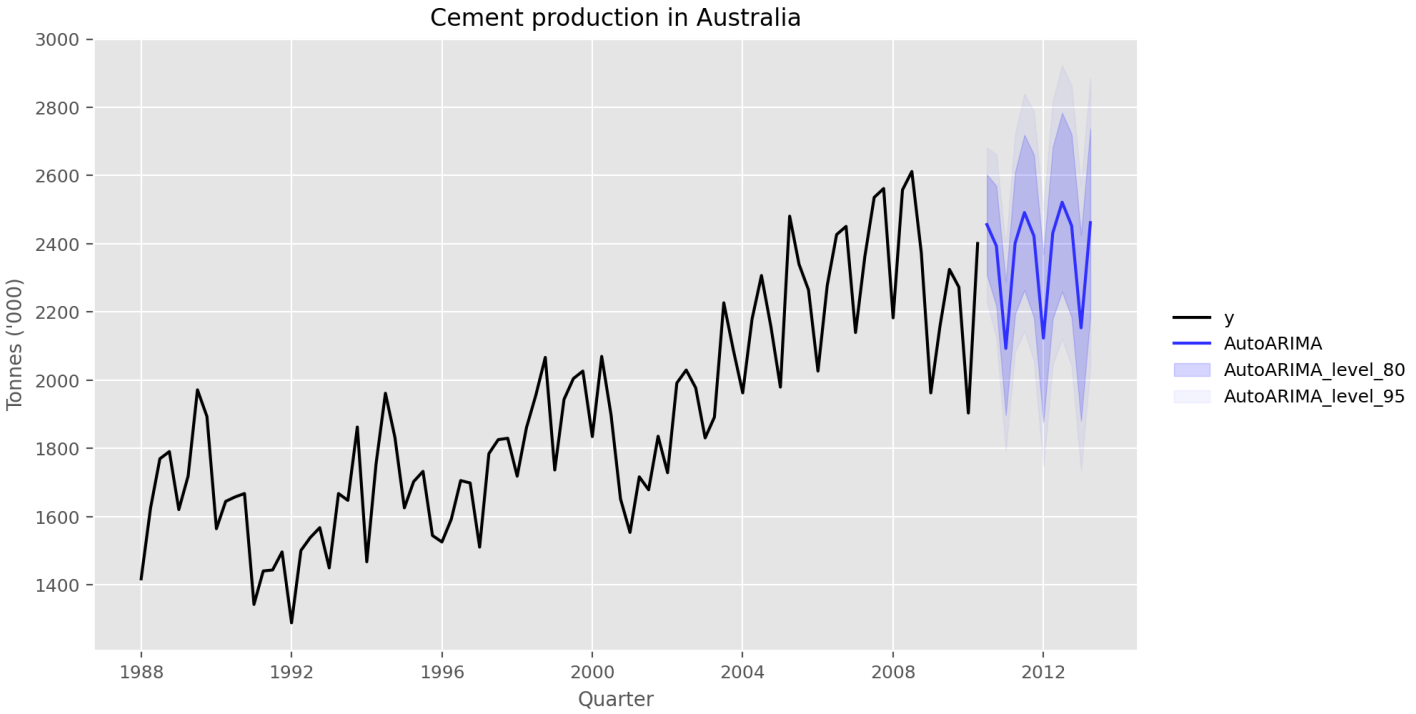

Figure 9.32: Forecasts from an ARIMA model fitted to all of the available quarterly cement production data since 1988.

9.11 Exercises

1. Figure 9.33 shows the ACFs for 36 random numbers, 360 random numbers and 1,000 random numbers.
   a. Explain the differences among these figures. Do they all indicate that the data are white noise?
   b. Why are the critical values at different distances from the mean of zero? Why are the autocorrelations different in each figure when they each refer to white noise?

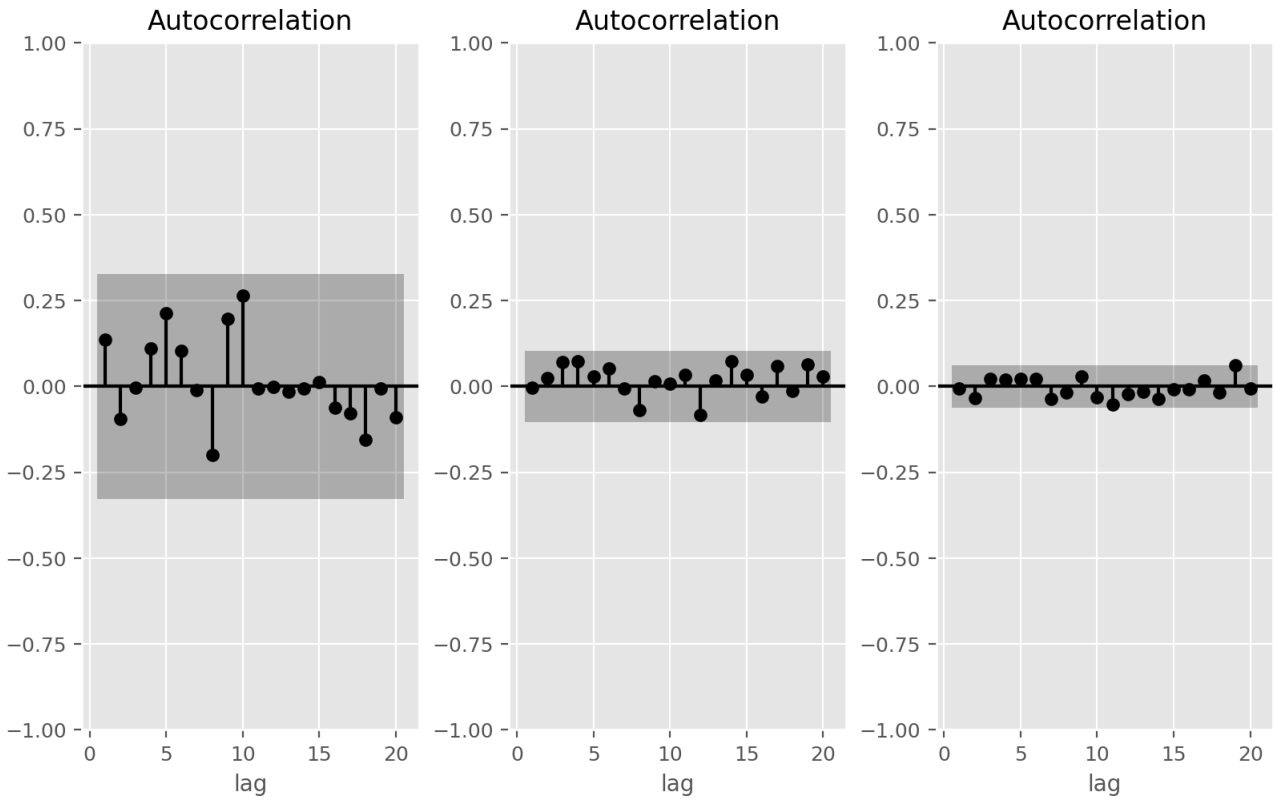

Figure 9.33: Left: ACF for a white noise series of 36 numbers. Middle: ACF for a white noise series of 360 numbers. Right: ACF for a white noise series of 1,000 numbers.

2. A classic example of a non-stationary series are stock prices. Plot the daily closing prices for Amazon stock (contained in `gafa_stock`), along with the ACF and PACF. Explain how each plot shows that the series is non-stationary and should be differenced.

3. For the following series, find an appropriate Box-Cox transformation and order of differencing in order to obtain stationary data.

    a. Turkish GDP from `global_economy`.
    b. Accommodation takings in the state of Tasmania from `aus_accommodation`.
    c. Monthly sales from `souvenirs`.

4. For the `souvenirs` data, write down the differences you chose above using backshift operator notation.

5. For your retail data (from Exercise 7 in Section 2.10), find the appropriate order of differencing (after transformation if necessary) to obtain stationary data.

6. Simulate and plot some data from simple ARIMA models.

    a. Use the following Python code to generate data from an AR(1) model with \phi_{1} = 0.6 and \sigma^2=1. The process starts with y_1=0.

```
y = np.zeros(100)
e = np.random.normal(size=100)

for i in range(1, 100):
    y[i] = 0.6 * y[i-1] + e[i]

sim = pd.DataFrame({'y': y})
```

    b. Produce a time plot for the series. How does the plot change as you change \phi_1?
    c. Write your own code to generate data from an MA(1) model with \theta_{1} = 0.6 and \sigma^2=1.
    d. Produce a time plot for the series. How does the plot change as you change \theta_1?
    e. Generate data from an ARMA(1,1) model with \phi_{1} = 0.6, \theta_{1} = 0.6 and \sigma^2=1.
    f. Generate data from an AR(2) model with \phi_{1} =-0.8, \phi_{2} = 0.3 and \sigma^2=1. (Note that these parameters will give a non-stationary series.)
    g. Graph the latter two series and compare them.

7. Consider `aus_airpassengers`, the total number of passengers (in millions) from Australian air carriers for the period 1970-

2011.

a. Use `AutoARIMA()` to find an appropriate ARIMA model. What model was selected. Check that the residuals look like white noise. Plot forecasts for the next 10 periods.
b. Write the model in terms of the backshift operator.
c. Plot forecasts from an ARIMA(0,1,0) model with drift and compare these to part a.
d. Plot forecasts from an ARIMA(2,1,2) model with drift and compare these to parts a and c. Remove the constant and see what happens.
e. Plot forecasts from an ARIMA(0,2,1) model with a constant. What happens?

8. For the United States GDP series (from `global_economy`):

a. If necessary, find a suitable Box-Cox transformation for the data;
b. Fit a suitable ARIMA model to the transformed data using `AutoARIMA()`;
c. Try some other plausible models by experimenting with the orders chosen;
d. Choose what you think is the best model and check the residual diagnostics;
e. Produce forecasts of your fitted model. Do the forecasts look reasonable?
f. Compare the results with what you would obtain using `AutoETS()` (with no transformation).

9. Consider `aus_arrivals`, the quarterly number of international visitors to Australia from several countries for the period 1981 Q1 – 2012 Q3.

a. Select one country and describe the time plot.
b. Use differencing to obtain stationary data.
c. What can you learn from the ACF graph of the differenced data?
d. What can you learn from the PACF graph of the differenced data?
e. What model do these graphs suggest?
f. Does `AutoARIMA()` give the same model that you chose? If not, which model do you think is better?
g. Write the model in terms of the backshift operator, then without using the backshift operator.

10. Choose a series from `us_employment`, the total employment in different industries in the United States.

a. Produce an MSTL decomposition of the data and describe the trend and seasonality.
b. Do the data need transforming? If so, find a suitable transformation.
c. Are the data stationary? If not, find an appropriate differencing which yields stationary data.
d. Identify a couple of ARIMA models that might be useful in describing the time series. Which of your models is the best according to their AICc values?
e. Estimate the parameters of your best model and do diagnostic testing on the residuals. Do the residuals resemble white noise? If not, try to find another ARIMA model which fits better.
f. Forecast the next 3 years of data. Get the latest figures from https://fred.stlouisfed.org/categories/11 to check the accuracy of your forecasts.
g. Eventually, the prediction intervals are so wide that the forecasts are not particularly useful. How many years of forecasts do you think are sufficiently accurate to be usable?

11. Choose one of the following seasonal time series: the Australian production of electricity, cement, or gas (from `aus_production`).

a. Do the data need transforming? If so, find a suitable transformation.
b. Are the data stationary? If not, find an appropriate differencing which yields stationary data.
c. Identify a couple of ARIMA models that might be useful in describing the time series. Which of your models is the best according to their AIC values?
d. Estimate the parameters of your best model and do diagnostic testing on the residuals. Do the residuals resemble white noise? If not, try to find another ARIMA model which fits better.
e. Forecast the next 24 months of data using your preferred model.
f. Compare the forecasts obtained using `AutoETS()`.

12. For the same time series you used in the previous exercise, try using a non-seasonal model applied to the seasonally adjusted data obtained from the MSTL. Compare the forecasts with those obtained in the previous exercise. Which do you think is the best approach?

13. For the Australian tourism data (from `tourism`):

a. Fit ARIMA models for each time series.
b. Produce forecasts of your fitted models.
c. Check the forecasts for the “Snowy Mountains” and “Melbourne” regions. Do they look reasonable?

14. For your retail time series (Exercise 5 above):

a. Develop an appropriate seasonal ARIMA model;
b. Compare the forecasts with those you obtained in earlier chapters;
c. Obtain up-to-date retail data from the ABS website (Cat 8501.0, Table 11), and compare your forecasts with the actual

numbers. How good were the forecasts from the various models?

15. Consider the number of Snowshoe Hare furs traded by the Hudson Bay Company between 1845 and 1935 (data set `pelt`).

    a. Produce a time plot of the time series.
    b. Assume you decide to fit the following model: y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_3 y_{t-3} + \phi_4 y_{t-4} + \varepsilon_t, where \varepsilon_t is a white noise series. What sort of ARIMA model is this (i.e., what are p, d, and q)?
    c. By examining the ACF and PACF of the data, explain why this model is appropriate.
    d. The last five values of the series are given below:

Year	1931	1932	1933	1934	1935
Number of hare pelts	19520	82110	89760	81660	15760

    The estimated parameters are c=30993, \phi_1=0.82, \phi_2=-0.29, \phi_3=-0.01, and \phi_4=-0.22. Calculate the forecasts for the next three years (1936–1939) using this model.

    e. Now fit the model with `StatsForecast`. How are they different from yours? Why?

16. The population of Switzerland from 1960 to 2017 is in data set `global_economy`.

    a. Produce a time plot of the data.
    b. You decide to fit the following model to the series: y_t = c + y_{t-1} + \phi_1 (y_{t-1} - y_{t-2}) + \phi_2 (y_{t-2} - y_{t-3}) + \phi_3(y_{t-3} - y_{t-4}) + \varepsilon_t where y_t is the Population in year t and \varepsilon_t is a white noise series. What sort of ARIMA model is this (i.e., what are p, d, and q)?
    c. Explain why this model was chosen using the ACF and PACF of the differenced series.
    d. The last five values of the series are given below.

Year	2013	2014	2015	2016	2017
Population (millions)	8.09	8.19	8.28	8.37	8.47

    The estimated parameters are c=0.0053, \phi_1 = 1.64, \phi_2 = -1.17, and \phi_3 = 0.45. Calculate forecasts for the next three years (2018–2020).

    e. Now fit the model with `StatsForecast` and obtain the forecasts from the same model. How are they different from yours? Why?

9.12 Further reading

- The classic text which popularised ARIMA modelling was Box and Jenkins (1970). The most recent edition is Box et al. (2015), and it is still an excellent reference for all things ARIMA.
- Brockwell and Davis (2016) provides a good introduction to the mathematical background to the models.
- The Hyndman-Khandakar algorithm for automatically selecting an ARIMA model is described in Hyndman and Khandakar (2008).
- Peña, Tiao, and Tsay (2001) describes some alternative automatic algorithms to the one used by `AutoARIMA()`.

9.13 Used modules and classes

StatsForecast

- `StatsForecast` class- Core forecasting engine
- `AutoARIMA` model - For automatic ARIMA modeling
- `ARIMA` model - For manual ARIMA modeling

UtilsForecast

- `plot_series` utility - For creating time series visualizations
- `rmse`, `mae`, `smape` metrics - For model evaluation

. More precisely, if \{y_t\} is a **stationary** time series, then for all s, the distribution of (y_t,\dots,y_{t+s}) does not depend on t. □□

2. arc cos is the inverse cosine function. You should be able to find it on your calculator. It may be labelled acos or cos^{-1}. □□

3. As already noted, comparing information criteria is only valid for ARIMA models of the same orders of differencing. □□

← Chapter 8 Exponential smoothing

Chapter 10 Dynamic regression models →