The time series models in the previous two chapters allow for the inclusion of information from past observations of a series, but not for the inclusion of other information that may also be relevant. For example, the effects of holidays, competitor activity, changes in the law, the wider economy, or other external variables, may explain some of the historical variation and may lead to more accurate forecasts. On the other hand, the regression models in Chapter 7 allow for the inclusion of a lot of relevant information from predictor variables, but do not allow for the subtle time series dynamics that can be handled with ARIMA models. In this chapter, we consider how to extend ARIMA models in order to allow other information to be included in the models.

In Chapter 7 we considered regression models of the form $y_t = \beta_0 + \beta_1 x_{1,t} + \dots + \beta_k x_{k,t} + \varepsilon_t$, where y_t is a linear function of the k predictor variables $(x_{1,t},\dots,x_{k,t})$, and ε_t is usually assumed to be an uncorrelated error term (i.e., it is white noise). We considered tests such as the Ljung-Box test for assessing whether the resulting residuals were significantly correlated.

In this chapter, we will allow the errors from a regression to contain autocorrelation. To emphasise this change in perspective, we will replace ε_t with η_t in the equation. The error series η_t is assumed to follow an ARIMA model. For example, if η_t follows an ARIMA(1,1,1) model, we can write $$\begin{align*} y_t &= \beta_0 + \beta_1 x_{1,t} + \dots + \beta_k x_{k,t} + \eta_t,\\ & (1-\phi_1B)(1-B)\eta_t = (1+\theta_1B)\varepsilon_t, \end{align*}$$ where ε_t is a white noise series.

Notice that the model has two error terms here — the error from the regression model, which we denote by η_t, and the error from the ARIMA model, which we denote by ε_t. Only the ARIMA model errors are assumed to be white noise.

10.1 Estimation

When we estimate the parameters from the model, we need to minimise the sum of squared ε_t values. If we minimise the sum of squared η_t values instead (which is what would happen if we estimated the regression model ignoring the autocorrelations in the errors), then several problems arise.

1. The estimated coefficients $\hat{\beta}_0,\dots,\hat{\beta}_k$ are no longer the best estimates, as some information has been ignored in the calculation;
2. Any statistical tests associated with the model (e.g., t-tests on the coefficients) will be incorrect.
3. The AICc values of the fitted models are no longer a good guide as to which is the best model for forecasting.
4. In most cases, the p-values associated with the coefficients will be too small, and so some predictor variables will appear to be important when they are not. This is known as “spurious regression”.

Minimizing the sum of squared ε_t values avoids these problems. Alternatively, maximum likelihood estimation can be used; this will give similar estimates of the coefficients.

An important consideration when estimating a regression with ARMA errors is that all of the variables in the model must first be stationary. Thus, we first have to check that y_t and all of the predictors $(x_{1,t},\dots,x_{k,t})$ appear to be stationary. If we estimate the model when any of these are non-stationary, the estimated coefficients will not be consistent estimates (and therefore may not be meaningful). One exception to this is the case where non-stationary variables are co-integrated. If there exists a linear combination of the non-stationary y_t and the predictors that is stationary, then the estimated coefficients will be consistent.[1]

We therefore first difference the non-stationary variables in the model. It is often desirable to maintain the form of the relationship between y_t and the predictors, and consequently it is common to difference all of the variables if any of them need differencing. The resulting model is then called a “model in differences”, as distinct from a “model in levels”, which is what is obtained when the original data are used without differencing.

If all of the variables in the model are stationary, then we only need to consider an ARMA process for the errors. It is easy to see that a regression model with ARIMA errors is equivalent to a regression model in differences with ARMA errors. For example, if the above regression model with ARIMA(1,1,1) errors is differenced we obtain the model $$\begin{align*} y'_t &= \beta_1 x'_{1,t} + \dots + \beta_k x'_{k,t} + \eta'_t,\\ & (1-\phi_1B)\eta'_t = (1+\theta_1B)\varepsilon_t, \end{align*}$$ where $y'_t=y_t-y_{t-1}$, $x'_{t,i}=x_{t,i}-x_{t-1,i}$ and $\eta'_t=\eta_t-\eta_{t-1}$, which is a regression model in differences with ARMA errors.

10.2 Regression with ARIMA errors using `StatsForecast`

The class `ARIMA()` will fit a regression model with ARIMA errors if exogenous regressors are included in the dataframe. As introduced in Section 9.5, `order` specifies the order of the ARIMA error model following the convention `(p, d, q)`. If differencing is specified, then the differencing is applied to all variables in the regression model before the model is estimated. For example, if `train_df` contains one additional exogenous variable, the following code

```
sf = StatsForecast(models=[ARIMA(order=(1, 1, 0))])
sf.fit(df=train_df)
```

will fit the model y_t' = \beta_1 x'_t + \eta'_t, where \eta'_t = \phi_1 \eta'_{t-1} + \varepsilon_t is an AR(1) error. This is equivalent to the model y_t = \beta_0 + \beta_1 x_t + \eta_t, where \eta_t is an ARIMA(1,1,0) error. Notice that the constant term disappears due to the differencing. To include a constant in the differenced model, we would add `1` to the model formula.

The `AutoARIMA()` can be used to select the best ARIMA model for the errors. Whether differencing is required is determined by applying a KPSS test to the residuals from the regression model estimated using ordinary least squares. If differencing is required, then all variables are differenced and the model re-estimated using maximum likelihood estimation. The final model will be expressed in terms of the original variables, even if it has been estimated using differenced variables.

The AICc is calculated for the final model, and this value can be used to determine the best predictors. That is, the procedure should be repeated for all subsets of predictors to be considered, and the model with the lowest AICc value selected.

Example: US Personal Consumption and Income

Figure 10.1 shows the quarterly changes in personal consumption expenditure and personal disposable income from 1970 to 2019 Q2. We would like to forecast changes in expenditure based on changes in income. A change in income does not necessarily translate to an instant change in consumption (e.g., after the loss of a job, it may take a few months for expenses to be reduced to allow for the new circumstances). However, we will ignore this complexity in this example and try to measure the instantaneous effect of the average change of income on the average change of consumption expenditure.

```
us_change = pd.read_csv('../data/US_change.csv', parse_dates=['ds'])
us_change = us_change.rename(columns={"y": "Consumption"})

fig, axes = plt.subplots(nrows=2, ncols=1)
sns.lineplot(data=us_change,
             x='ds', y='Consumption', ax=axes[0], color='black')
sns.lineplot(data=us_change, x='ds', y='Income', ax=axes[1], color='black')
fig.suptitle("US consumption and personal income")
plt.show()
```

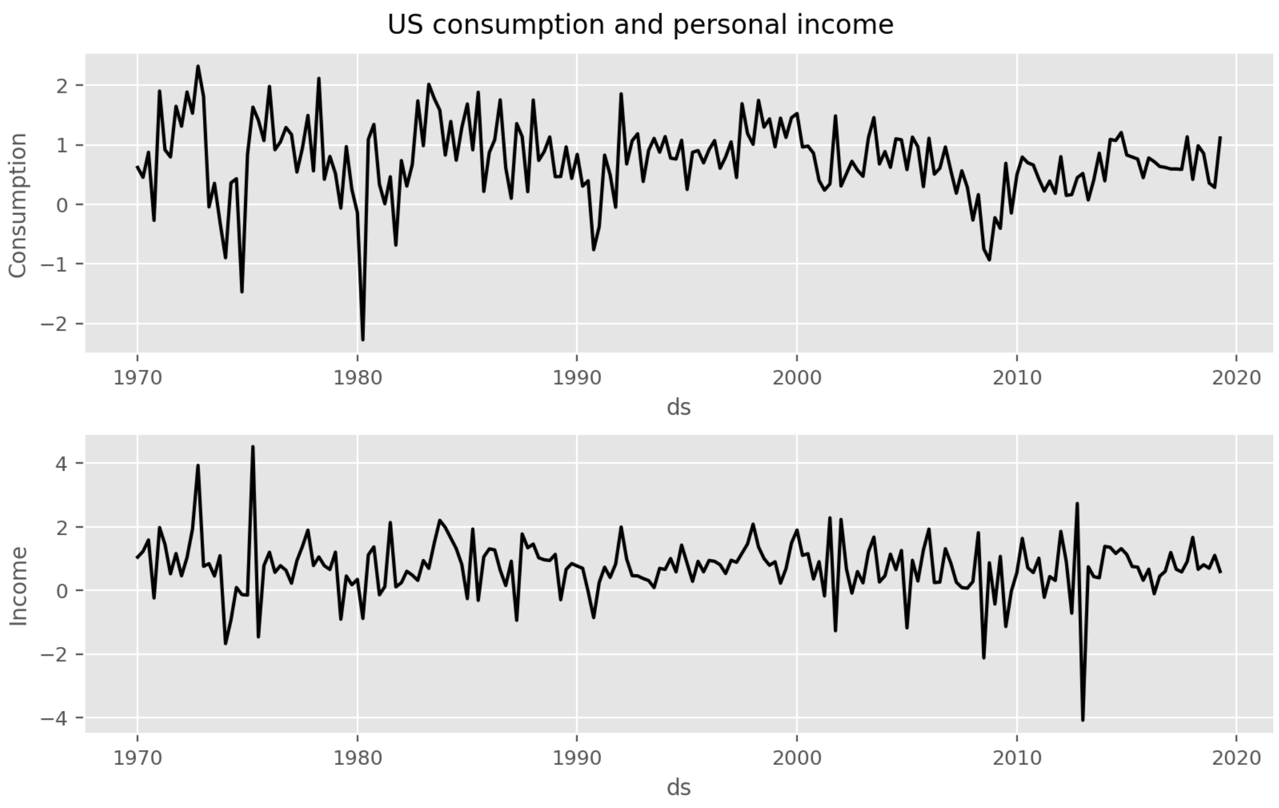

Figure 10.1: Percentage changes in quarterly personal consumption expenditure and personal disposable income for the USA, 1970 Q1 to 2019 Q2.

```
Regression with ARIMA(1,0,2) errors
Coefficients: {'ar1': 0.7070114069745624, 'ma1': -0.6172228681521843, 'ma2': 0.20663432656409253, 'ex_1': 0.197
62556963397948, 'mean': 0.5949262342120432}
sigma^2      : 0.31
loglik       : -163.04
aic          : 338.07
aicc         : 338.51
bic          : 357.80
```

The data are clearly already stationary (as we are considering percentage changes rather than raw expenditure and income), so there is no need for any differencing. The fitted model is \begin{align*} y_t &= 0.595 + 0.198 x_t + \eta_t, \\ \eta_t &= 0.707 \eta_{t-1} + \varepsilon_t - 0.617 \varepsilon_{t-1} + 0.207 \varepsilon_{t-2},\\ \varepsilon_t &\sim \text{NID}(0, 0.311). \end{align*}

We can recover estimates of both the \eta_t and \varepsilon_t series using the following,

```
innov_residuals = sf.fitted_[0, 0].model_["residuals"]
reg_residuals = us_change["Consumption"] - sf.fitted_[0, 0].model_['coef']["ex_1"] * us_change["In

resids_df = pd.DataFrame({"Regression residuals": reg_residuals, "ARIMA residuals": innov_residual

resids_df["ds"] = us_change["ds"]
fig, axes = plt.subplots(nrows=2, ncols=1)
sns.lineplot(data=resids_df,
             x='ds', y='Regression residuals', ax=axes[0], color='black')
sns.lineplot(data=resids_df, x='ds', y='ARIMA residuals', ax=axes[1], color='black')
plt.show()
```

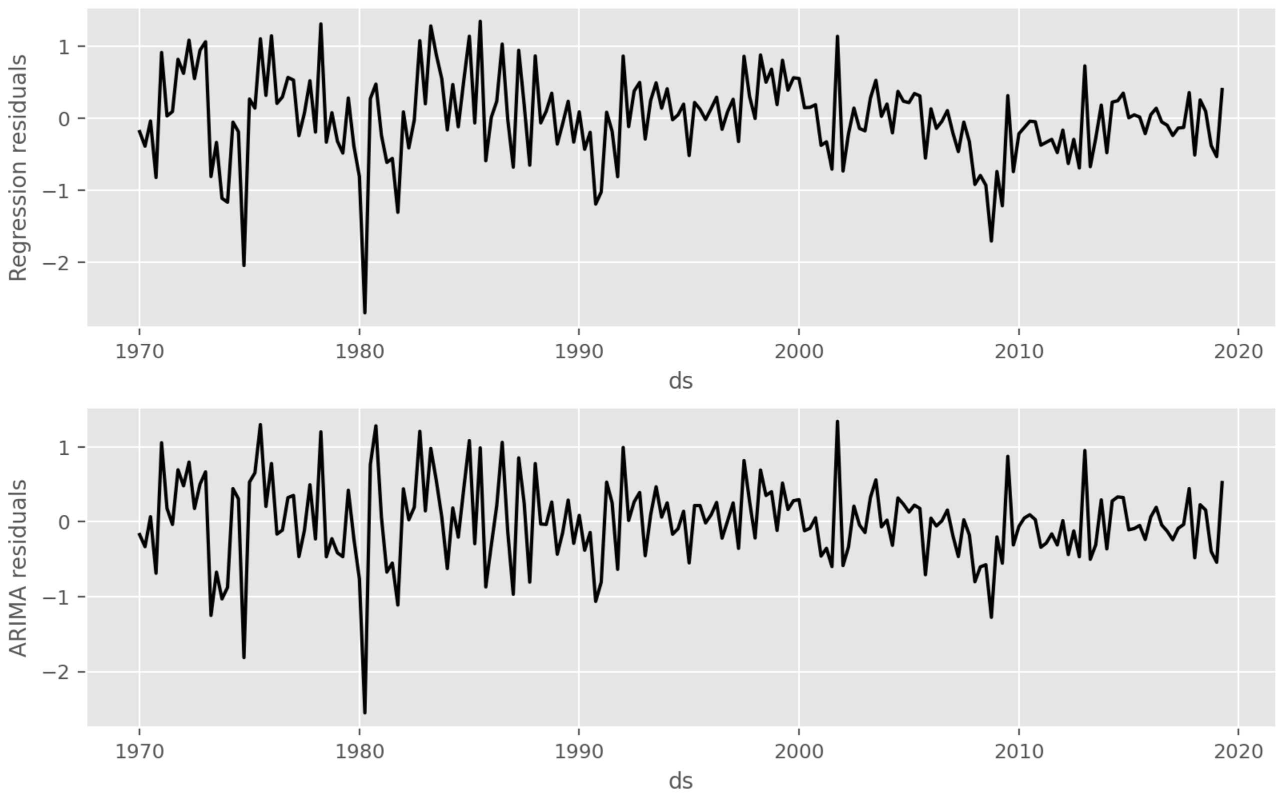

Figure 10.2: Regression residuals (\eta_t) and ARIMA residuals (\varepsilon_t) from the fitted model.

It is the ARIMA estimated errors (the innovation residuals) that should resemble a white noise series.

```
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])
resids = sf.fitted_[0, 0].model_["residuals"]
ax1.plot(us_change['ds'], resids, marker="o")
ax1.set_ylabel('ARIMA residuals')
ax1.set_xlabel('Date')

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

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

plt.show()
```

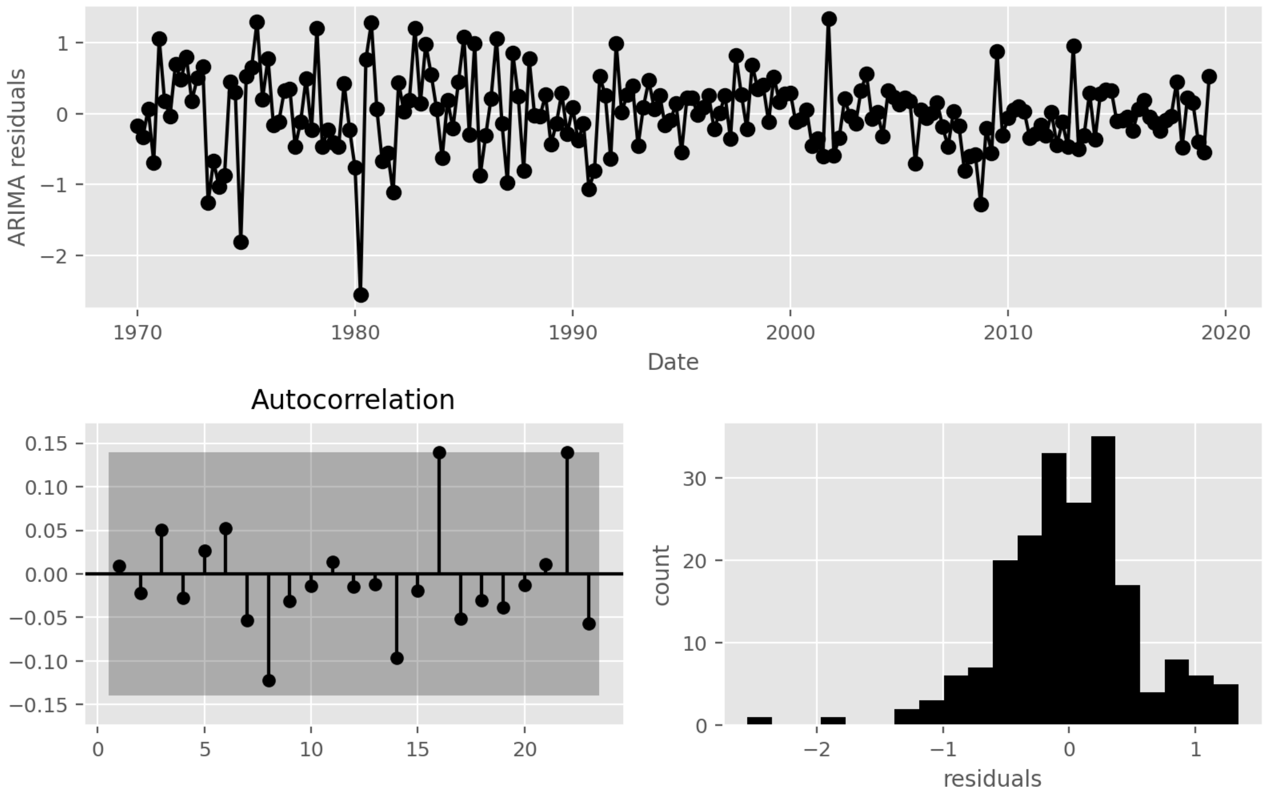

Figure 10.3: The innovation residuals (i.e., the estimated ARIMA errors) are not significantly different from white noise.

```
from statsmodels.stats.diagnostic import acorr_ljungbox
ljung_box = acorr_ljungbox(resids, lags=[8], model_df=3)

ljung_box
```

	lb_stat	lb_pvalue
8	5.207175	0.391123

10.3 Forecasting

To forecast using a regression model with ARIMA errors, we need to forecast the regression part of the model and the ARIMA part of the model, and combine the results. As with ordinary regression models, in order to obtain forecasts we first need to forecast the predictors. When the predictors are known into the future (e.g., calendar-related variables such as time, day-of-week, etc.), this is straightforward. But when the predictors are themselves unknown, we must either model them separately, or use assumed future values for each predictor.

Example: US Personal Consumption and Income

We will calculate forecasts for the next eight quarters assuming that the future percentage changes in personal disposable income will be equal to the mean percentage change from the last forty years.

```python
sf = StatsForecast(
    models=[AutoARIMA()],
    freq="QS",
)
future_predictors = make_future_dataframe(
    us_change["unique_id"].unique(),
    pd.to_datetime([us_change["ds"].max()]),
    freq="QS",
    h=8,
)
future_predictors['Income'] = us_change['Income'].mean()

fc = sf.forecast(
    df=us_change[["unique_id", "ds", "Consumption", "Income"]],
    h=8, X_df=future_predictors,
    target_col="Consumption",
    fitted=True,
    level=[80, 95]
)
plot_series(us_change, fc, target_col="Consumption", level=[80, 95],
            title="",
            xlabel="Quarter", ylabel="Percentage change",
            rm_legend=False,)
```

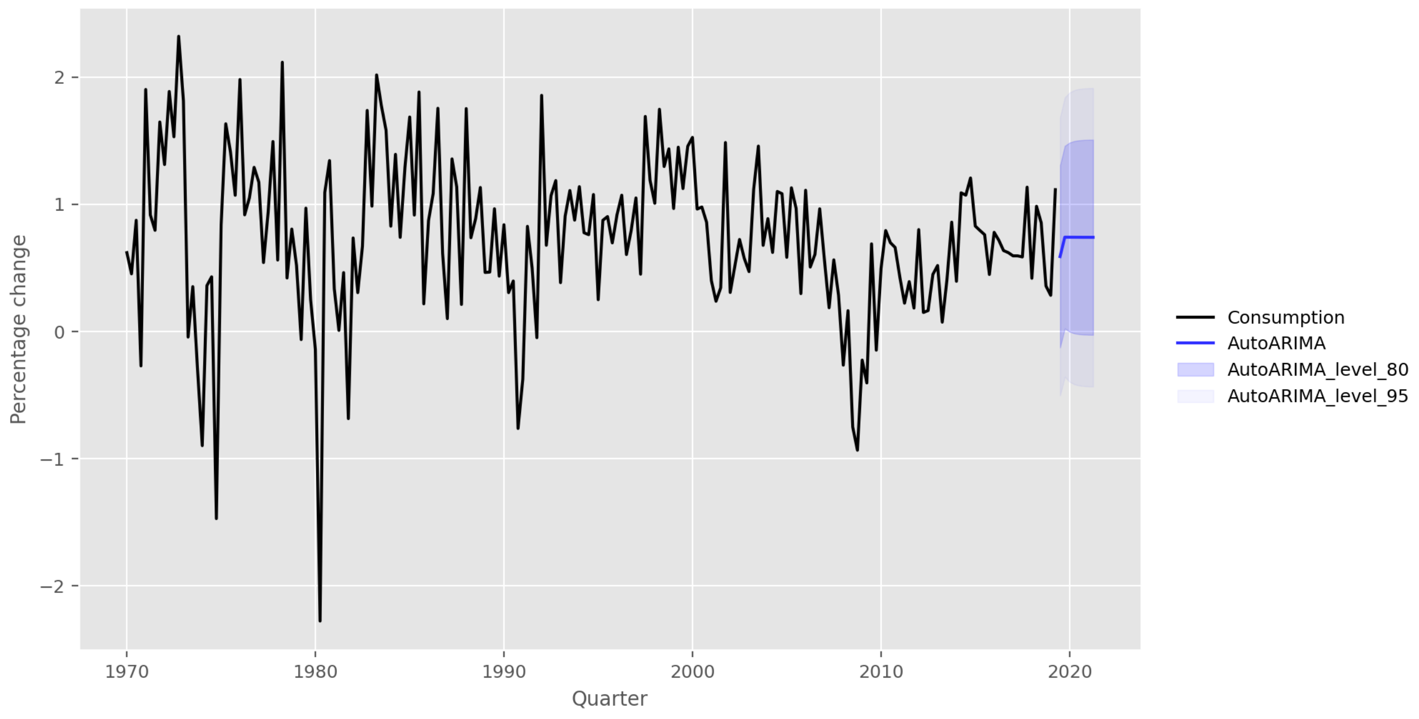

Figure 10.4: Forecasts obtained from regressing the percentage change in consumption expenditure on the percentage change in disposable income, with an ARIMA error model.

The prediction intervals for this model are narrower than if we had fitted an ARIMA model without covariates, because we are now able to explain some of the variation in the data using the income predictor.

It is important to realise that the prediction intervals from regression models (with or without ARIMA errors) do not take into account the uncertainty in the forecasts of the predictors. So they should be interpreted as being conditional on the assumed (or estimated) future values of the predictor variables.

Example: Forecasting electricity demand

Daily electricity demand can be modelled as a function of temperature. As can be observed on an electricity bill, more electricity is used on cold days due to heating and hot days due to air conditioning. The higher demand on cold and hot days is reflected in the U-shape of Figure 10.5, where daily demand is plotted versus daily maximum temperature.

```
vic_elec = pd.read_csv('../data/vic_elec.csv', parse_dates=['ds'])

vic_elec_df = vic_elec.pivot(index = 'ds', columns = 'unique_id', values = 'y').reset_index()
vic_elec_df = vic_elec_df.merge(vic_elec[['ds', 'Holiday']].drop_duplicates(), on = 'ds')
vic_elec_df.set_index('ds', inplace=True)

vic_elec_daily = vic_elec_df.resample('D').agg({
    'Demand': 'sum',
    'Temperature': 'max',
    'Holiday': 'first'
}).reset_index()

vic_elec_daily['Demand'] = vic_elec_daily['Demand']/1e3
vic_elec_daily.insert(0, 'unique_id', 'VIC')
vic_elec_daily = vic_elec_daily.query('ds >= "2014-01-01"')
vic_elec_daily['Day_Week'] = vic_elec_daily['ds'].dt.dayofweek

conditions = [
    vic_elec_daily['Holiday'] == True,
    (vic_elec_daily['Holiday'] == False) & (vic_elec_daily['Day_Week'] < 5),
    (vic_elec_daily['Holiday'] == False) & (vic_elec_daily['Day_Week'] >= 5)
]

vic_elec_daily['Day_Type'] = np.select(conditions, ['Holiday', 'Weekday', 'Weekend'])
vic_elec_daily.drop(columns=['Day_Week'], inplace=True)

plt.figure()
sns.scatterplot(data=vic_elec_daily, x='Temperature', y='Demand', hue='Day_Type')
plt.xlabel('Maximum daily temperature')
plt.ylabel('Electricity demand (GW)')
plt.show()
```

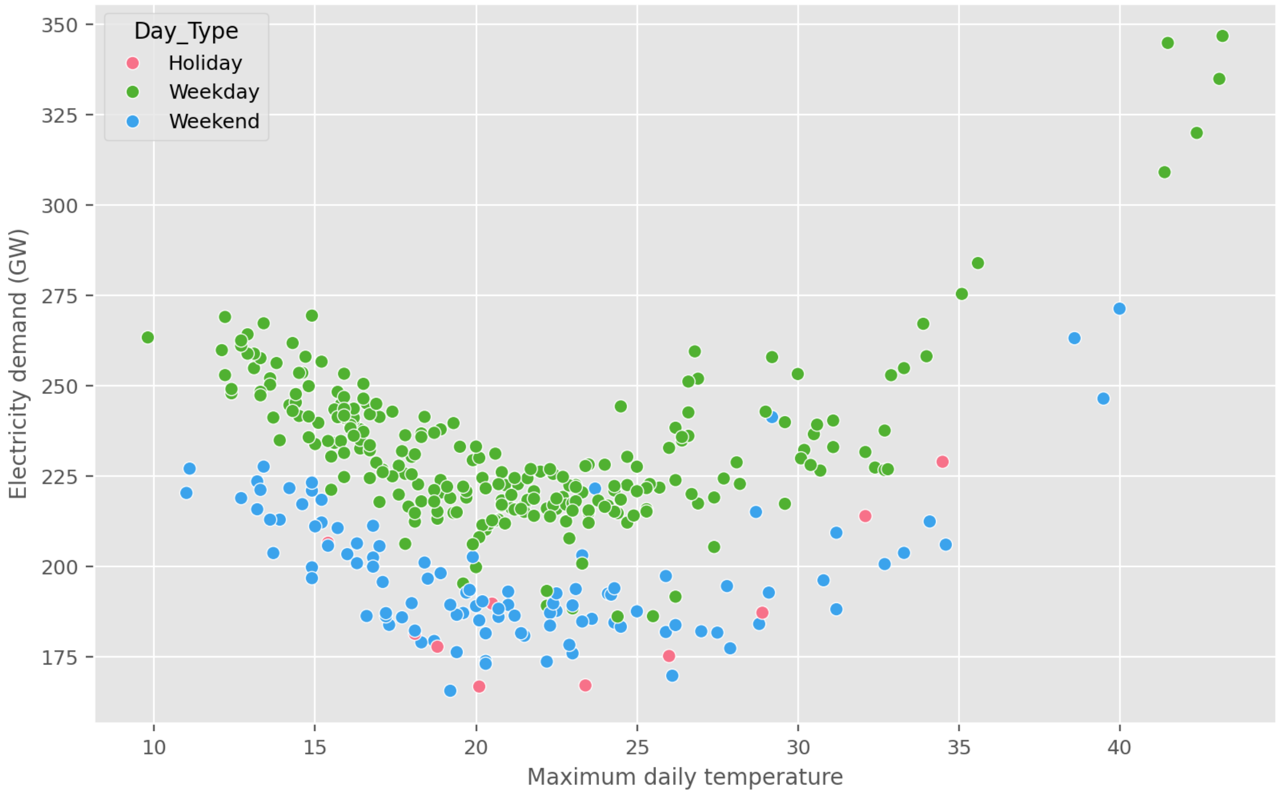

Figure 10.5: Daily electricity demand versus maximum daily temperature for the state of Victoria in Australia for 2014.

The data stored as `vic_elec_daily` includes total daily demand, daily maximum temperatures, and an indicator variable for if that day is a public holiday. Figure 10.6 shows the time series of both daily demand and daily maximum temperatures. The plots highlight the need for both a non-linear and a dynamic model.

```
fig, axes = plt.subplots(nrows=2, ncols=1)
sns.lineplot(data=vic_elec_daily, x='ds', y='Demand', ax=axes[0], color='black', label='Demand')
sns.lineplot(data=vic_elec_daily, x='ds', y='Temperature', ax=axes[1], color='black', label='Tempe

plt.show()
```

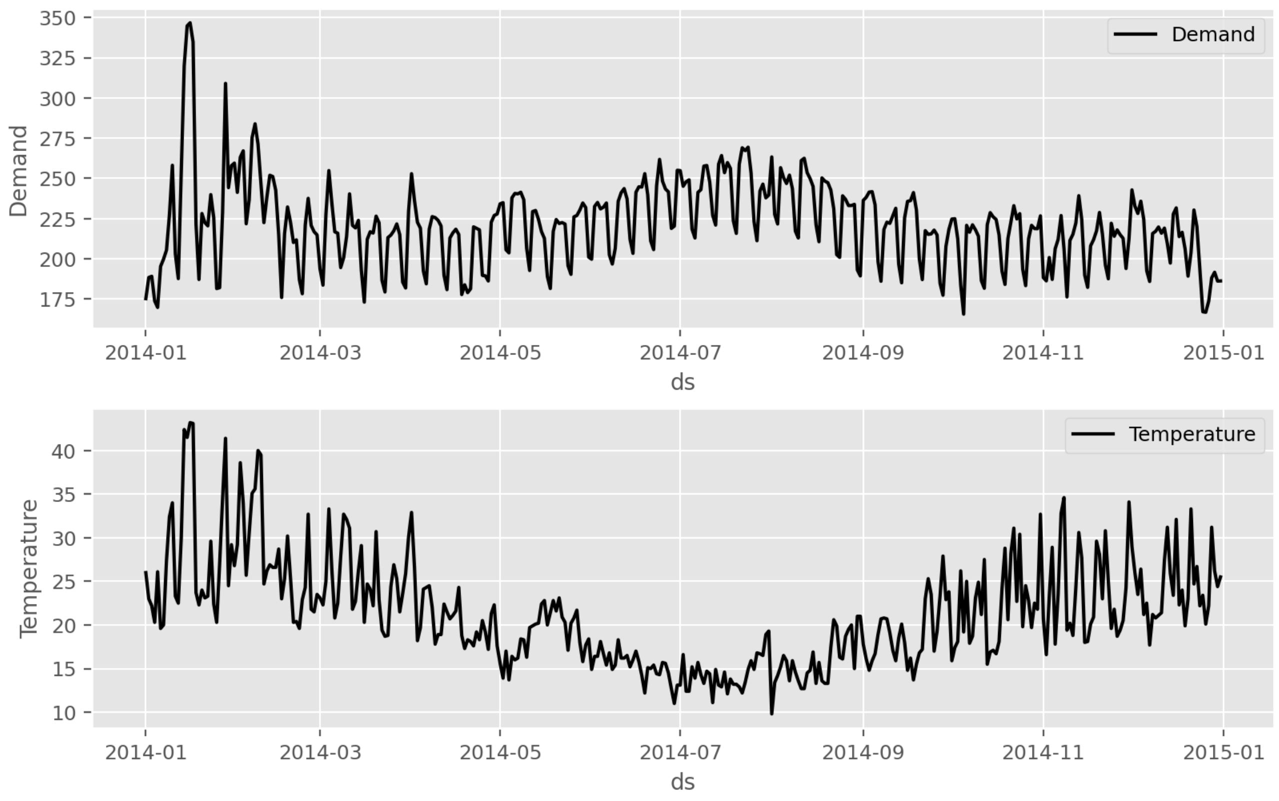

Figure 10.6: Daily electricity demand and maximum daily temperature for the state of Victoria in Australia for 2014.

In this example, we fit a quadratic regression model with ARMA errors using the `ARIMA()` function. The model also includes an indicator variable for if the day was a working day or not.

```
sf = StatsForecast(
    models=[AutoARIMA(season_length=7)],
    freq="D",
)
vic_elec_daily_ex = vic_elec_daily[["unique_id", "ds", "Demand", "Temperature"]]
vic_elec_daily_ex["Weekday"] = vic_elec_daily["Day_Type"] == "Weekday"
vic_elec_daily_ex["Temperature**2"] = vic_elec_daily_ex["Temperature"] ** 2
sf = sf.fit(
    df=vic_elec_daily_ex,
    target_col="Demand",
)
resids = sf.fitted_[0, 0].model_["residuals"]
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(vic_elec_daily['ds'], resids, marker="o")
ax1.set_ylabel('Innovation residuals')
ax1.set_xlabel('Date')

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

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

plt.show()
```

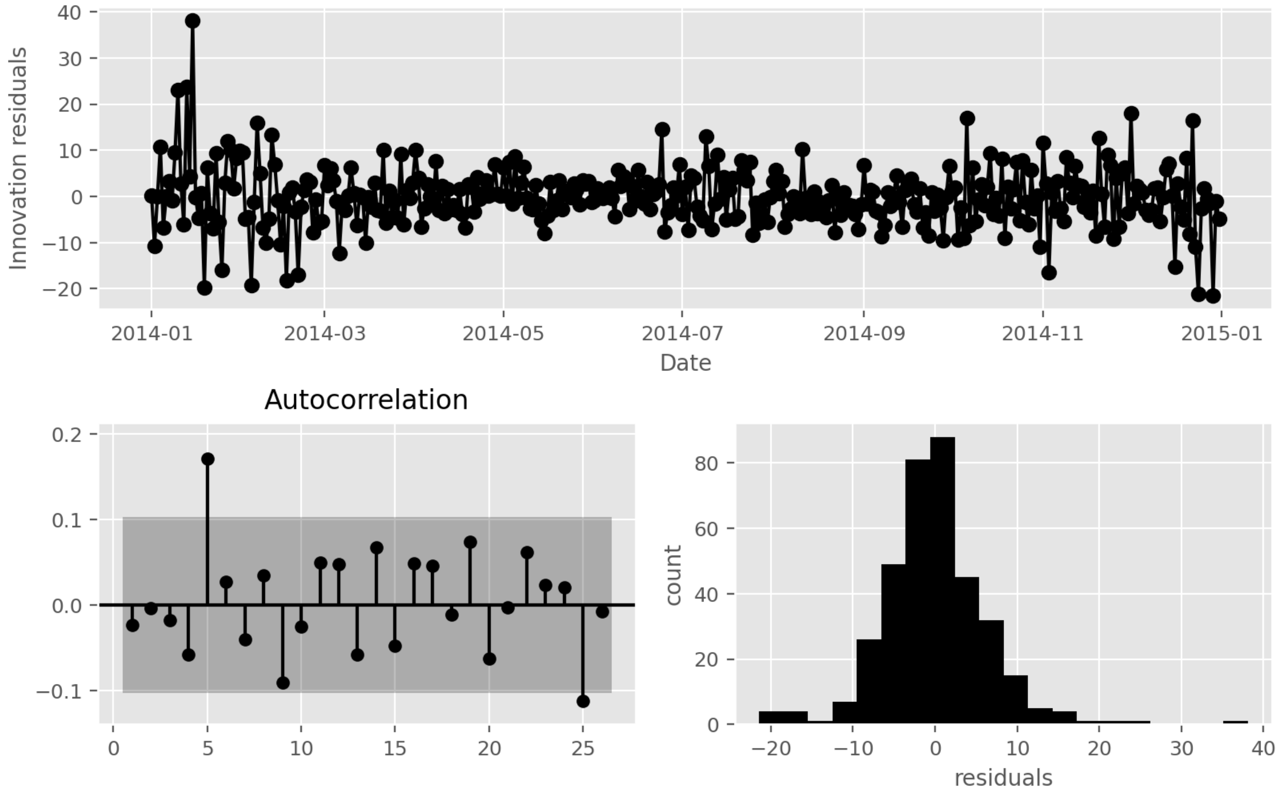

Figure 10.7: Residuals diagnostics for a dynamic regression model for daily electricity demand with workday and temperature effects.

```python
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}")
```

```
Regression with ARIMA(2,1,2)(2,0,2)[7] errors
Coefficients: {'ar1': -0.08712338557522067, 'ar2': 0.7162606447771099, 'ma1': -0.023671459995323885, 'ma2': -0.
9137732769094125, 'sar1': 0.30861541915253937, 'sar2': 0.5556108874362441, 'sma1': -0.16373062386128245, 'sma2'
: -0.3857762792040939, 'ex_1': -7.501428513654931, 'ex_2': 29.3253191232545, 'ex_3': 0.17767801803069874}
sigma^2      : 43.06
loglik       : -1197.38
aic          : 2418.77
aicc         : 2419.66
bic          : 2465.53
```

The fitted model has an ARIMA(2,1,2)(2,0,0)[7] error, so there are 6 AR and MA coefficients.

```python
from statsmodels.stats.diagnostic import acorr_ljungbox
ljung_box = acorr_ljungbox(resids, lags=[14], model_df=6)

ljung_box
```

	lb_stat	lb_pvalue
14	21.870567	0.005161

There is clear heteroscedasticity in the residuals, with higher variance in January and February, and lower variance in May. The model also has some significant autocorrelation in the residuals, and the histogram of the residuals shows long tails. All of these issues with the residuals may affect the coverage of the prediction intervals, but the point forecasts should still be ok.

Using the estimated model we forecast 14 days ahead starting from Thursday 1 January 2015 (a non-work-day being a public holiday for New Years Day). In this case, we could obtain weather forecasts from the weather bureau for the next 14 days. But for the sake of illustration, we will use scenario based forecasting (as introduced in Section 7.6) where we set the temperature for the next 14 days to a constant 26 degrees.

```
future_predictors = make_future_dataframe(
    vic_elec_daily_ex["unique_id"].unique(),
    pd.to_datetime([vic_elec_daily_ex["ds"].max()]),
    freq="D",
    h=14,
)
future_predictors['Temperature'] = 26
future_predictors["Weekday"] = ~future_predictors["ds"].dt.dayofweek.isin([5, 6])
future_predictors['Temperature**2'] = 26 ** 2

fc = sf.predict(
    h=14, X_df=future_predictors,
    level=[80, 95]
)
plot_series(vic_elec_daily_ex, fc, target_col="Demand", level=[80, 95],
            xlabel="Date", ylabel="GW",
            title="Daily electricity demand: Victoria",
            rm_legend=False,)
```

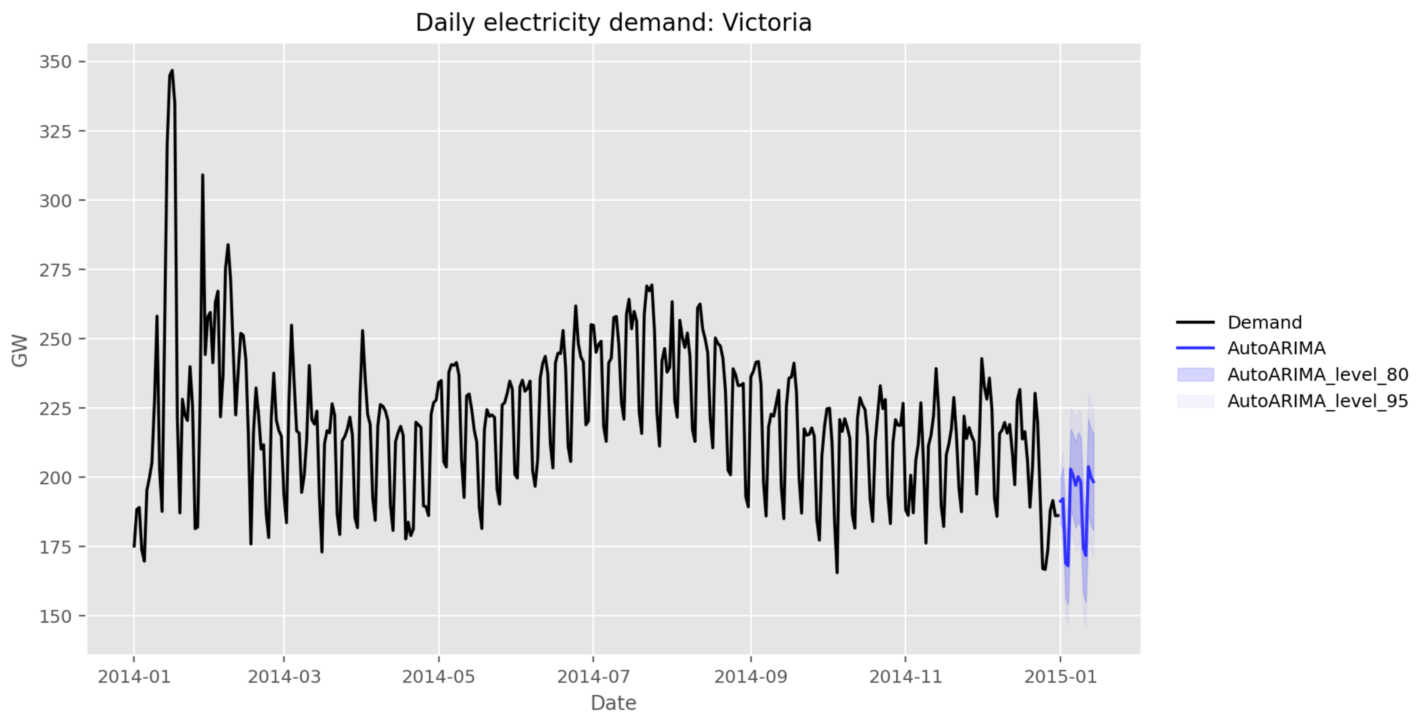

Figure 10.8: Forecasts from the dynamic regression model for daily electricity demand. All future temperatures have been set to 26 degrees, and the working day dummy variable has been set to known future values.

The point forecasts look reasonable for the first two weeks of 2015. The slow down in electricity demand at the end of 2014 (due to many people taking summer vacations) has caused the forecasts for the next two weeks to show similarly low demand values.

10.4 Stochastic and deterministic trends

There are two different ways of modelling a linear trend. A *deterministic trend* is obtained using the regression model y_t = \beta_0 + \beta_1 t + \eta_t, where \eta_t is an ARMA process. A *stochastic trend* is obtained using the model y_t = \beta_0 + \beta_1 t + \eta_t, where \eta_t is an ARIMA process with d=1. In the latter case, we can difference both sides so that y_t' = \beta_1 + \eta_t', where \eta_t' is an ARMA process. In other words, y_t = y_{t-1} + \beta_1 + \eta_t'. This is similar to a random walk with drift (introduced in Section 9.1), but here the error term is an ARMA process rather than simply white noise.

Although these models appear quite similar (they only differ in the number of differences that need to be applied to \eta_t), their forecasting characteristics are quite different.

Example: Air transport passengers Australia

```
aus_airpassengers = pd.read_csv('../data/aus_airpassengers.csv', parse_dates=['Year'])[['Year', 'P

aus_airpassengers.insert(0, 'unique_id', 'aus_airpassengers')
aus_airpassengers.rename(columns={'Year':'ds', 'Passengers':'y'}, inplace=True)

plot_series(aus_airpassengers,
            xlabel="Year [1Y]",
            ylabel="Passengers (millions)",
            title="Total annual air passengers")
```

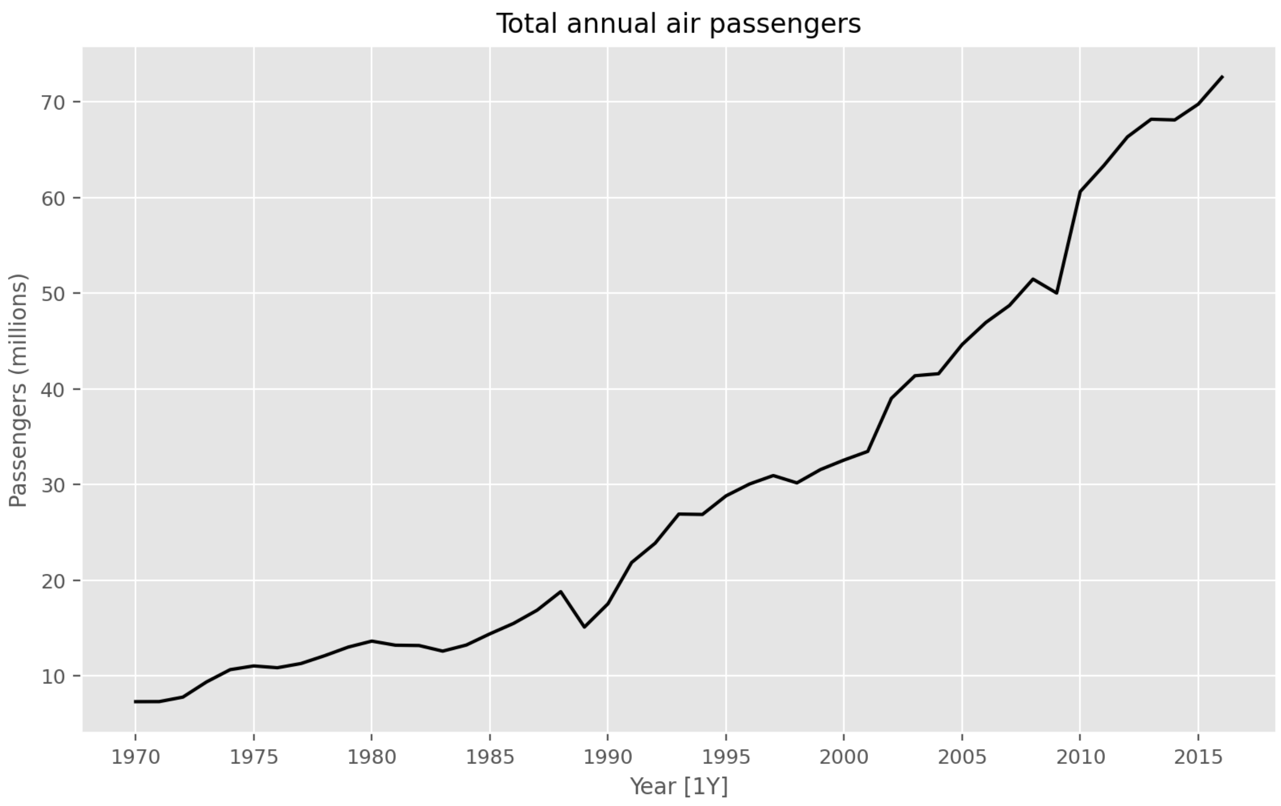

Figure 10.9: Total annual passengers (in millions) for Australian air carriers, 1970–2016.

Figure 10.9 shows the total number of passengers for Australian air carriers each year from 1970 to 2016. We will fit both a deterministic and a stochastic trend model to these data using the `trend` method from `utilsforecast`. The deterministic trend model is obtained as follows:

```python
aus_airpassengers, future_predictors = trend(aus_airpassengers, freq='AS', h=20)
aus_airpassengers["intercept"] = 1
future_predictors["intercept"] = 1

sf = StatsForecast(
    models=[AutoARIMA(d=0, alias="AutoARIMA_stoch")],
    freq="AS",
)
sf = sf.fit(
    df=aus_airpassengers,
)
fc_stoch = sf.predict(
    h=20, X_df=future_predictors,
    level=[80, 95]
)
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}")
```

```
Regression with ARIMA(1,0,0) errors
Coefficients: {'ar1': 0.9563979073613333, 'ex_1': 1.4150974130298302, 'ex_2': 0.9022964828100731}
sigma^2     : 4.34
loglik      : -100.88
aic         : 209.77
aicc        : 210.72
bic         : 217.17
```

This model can be written as \begin{align*} y_t &= 0.901 + 1.415 t + \eta_t \\ \eta_t &= 0.956 \eta_{t-1} + \varepsilon_t\\ \varepsilon_t &\sim \text{NID}(0, 4.34). \end{align*}

The estimated growth in visitor numbers is 1.42 million people per year.

Alternatively, the stochastic trend model can be estimated.

```python
sf = StatsForecast(
    models=[AutoARIMA(d=1, alias="AutoARIMA_det")],
    freq="AS",
)
sf = sf.fit(
    df=aus_airpassengers[["unique_id", "ds", "y"]],
)
fc_det = sf.predict(
    h=20,
    level=[80, 95]
)
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(0,1,0) with drift
Coefficients: {'drift': 1.4191087131739135}
sigma^2     : 4.27
loglik      : -98.16
aic         : 200.31
aicc        : 200.59
bic         : 203.97
```

This model can be written as $y_t-y_{t-1} = 1.419 + \varepsilon_t$, or equivalently

$$\begin{align*} y_t &= y_0 + 1.419 t + \eta_t \\ \eta_t &= \eta_{t-1} + \varepsilon_{t}\\ \varepsilon_t &\sim \text{NID}(0,0.4271). \end{align*}$$

In this case, the estimated growth in visitor numbers is also 1.42 million people per year. Although the growth estimates are similar, the prediction intervals are not, as Figure 10.10 shows. In particular, stochastic trends have much wider prediction intervals because the errors are non-stationary.

```python
res = fc_stoch.merge(fc_det, on=['unique_id', 'ds'])
plot_series(aus_airpassengers, res, level=[95],
            xlabel="Year [1Y]",
            ylabel="Air passengers (millions)",
            title="Forecasts from trend models",
            rm_legend=False)
```

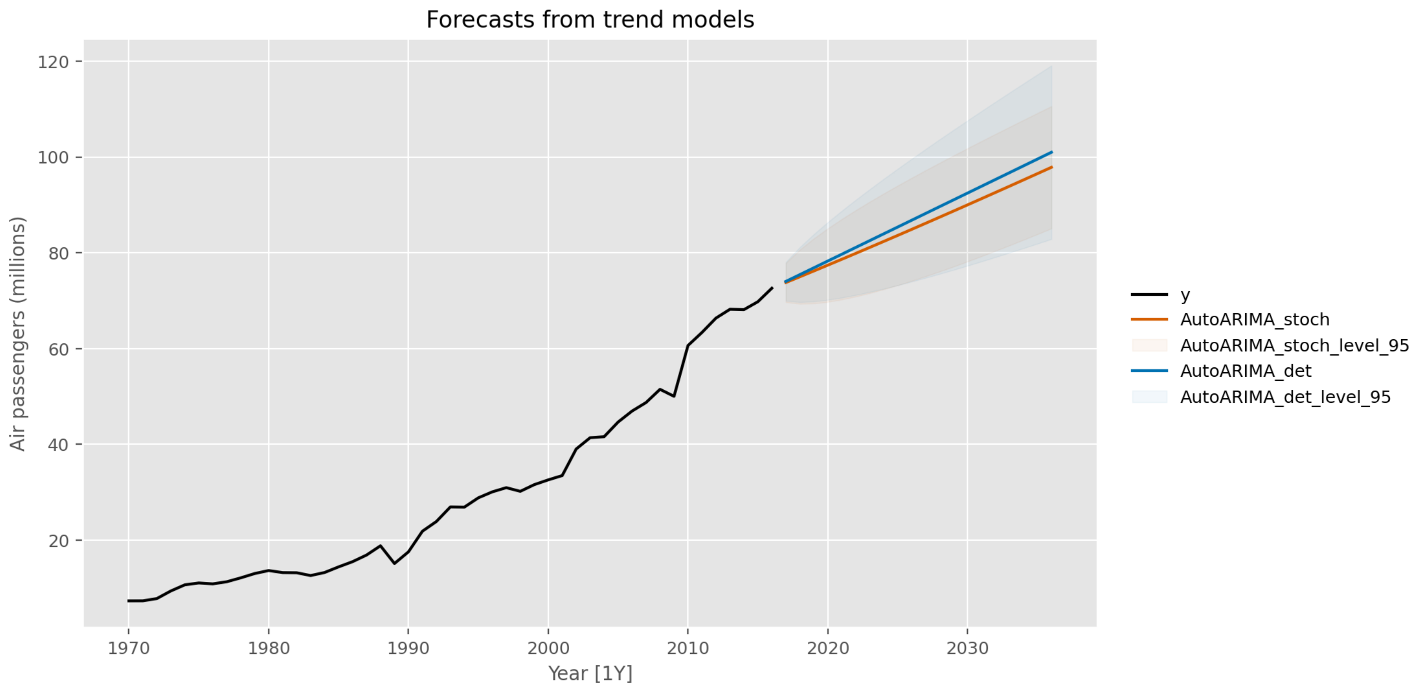

Figure 10.10: Forecasts of annual passengers for Australian air carriers using a deterministic trend model (yellow) and a stochastic trend model (pink).

There is an implicit assumption with deterministic trends that the slope of the trend is not going to change over time. On the other hand, stochastic trends can change, and the estimated growth is only assumed to be the average growth over the historical period, not necessarily the rate of growth that will be observed into the future. Consequently, it is safer to forecast with stochastic trends, especially for longer forecast horizons.

10.5 Dynamic harmonic regression

When there are long seasonal periods, a dynamic regression with Fourier terms is often better than other models we have considered in this book.[2]

For example, daily data can have annual seasonality of length 365, weekly data has seasonal period of approximately 52, while half-hourly data can have several seasonal periods, the shortest of which is the daily pattern of period 48.

Seasonal versions of ARIMA and ETS models are designed for shorter periods such as 12 for monthly data or 4 for quarterly data. The `ETS()` model restricts seasonality to be a maximum period of 24 to allow hourly data but not data with a larger seasonal period. The problem is that there are m-1 parameters to be estimated for the initial seasonal states where m is the seasonal period. So for large m, the estimation becomes almost impossible.

So for such time series, we prefer a harmonic regression approach where the seasonal pattern is modelled using Fourier terms with short-term time series dynamics handled by an ARMA error.

The advantages of this approach are:

- it allows any length seasonality;
- for data with more than one seasonal period, Fourier terms of different frequencies can be included;
- the smoothness of the seasonal pattern can be controlled by K, the number of Fourier sin and cos pairs – the seasonal pattern is smoother for smaller values of K;
- the short-term dynamics are easily handled with a simple ARMA error.

The only real disadvantage (compared to a seasonal ARIMA model) is that the seasonality is assumed to be fixed — the seasonal pattern is not allowed to change over time. But in practice, seasonality is usually remarkably constant so this is not a big disadvantage except for long time series.

Example: Australian eating out expenditure

In this example we demonstrate combining Fourier terms for capturing seasonality with ARIMA errors capturing other dynamics in the data. For simplicity, we will use an example with monthly data. The same modelling approach using weekly data is discussed in Section 13.1.

We use the total monthly expenditure on cafes, restaurants and takeaway food services in Australia (\$ billion) from 2004 up to 2018 and forecast 24 months ahead. We vary K, the number of Fourier sin and cos pairs, from K=1 to K=6 (which is equivalent to including seasonal dummies). Figure 10.11 shows the seasonal pattern projected forward as K increases. Notice that as K increases the Fourier terms capture and project a more “wiggly” seasonal pattern and simpler ARIMA models are required to capture other dynamics. The AICc value is minimised for K=6, with a significant jump going from K=4 to K=5, hence the forecasts generated from this model would be the ones used.

Figure 10.11 shows the seasonal pattern projected forward as K increases. Notice that as K increases the Fourier terms capture and project a more “wiggly” seasonal pattern and simpler ARIMA models are required to capture other dynamics.

```
aus_cafe_df = (
    pd.read_csv('../data/aus_retail.csv', parse_dates=['Month'])
    .query('Industry == "Cafes, restaurants and takeaway food services" & Month >= "2004-01-01" &

    .groupby('Month')
    .agg({'Turnover': 'sum'})
    .assign(unique_id = 'aus_cafe')
    .reset_index()
    .rename(columns={'Month':'ds', 'Turnover':'y'})
    .reindex(columns=['unique_id', 'ds', 'y'])
)

results = []
for i in range(1,7):
    aus_cafe = aus_cafe_df.copy()
    aus_cafe['y'] = np.log(aus_cafe['y'])

    features = [
        partial(fourier, season_length=12, k=i)
    ]

    aus_cafe, future_predictors = pipeline(
        aus_cafe,
        features=features,
        freq='MS',
        h=24
    )

    sf = StatsForecast(
        models=[AutoARIMA(season_length=12, seasonal=False)],
        freq='MS'
    )

    res = sf.forecast(
        df=aus_cafe, h=24,
        X_df=future_predictors,
        level=[80, 95],
    )
    res[res.select_dtypes(include=np.number).columns] = res.select_dtypes(include=np.number).apply

    res['unique_id'] = 'K='+str(i)
    results.append(res)

def plot_with_intervals(ax, result, color):
    sns.lineplot(data=aus_cafe_df, x='ds', y='y', ax=ax, color='black')
    sns.lineplot(data=result, x='ds', y='AutoARIMA', ax=ax, color=color)
    sns.lineplot(data=result, x='ds', y='AutoARIMA-lo-95', ax=ax, color=color, alpha=0.2)
    sns.lineplot(data=result, x='ds', y='AutoARIMA-hi-95', ax=ax, color=color, alpha=0.2)
    ax.fill_between(result['ds'], result['AutoARIMA-lo-95'], result['AutoARIMA-hi-95'], color=colo

    ax.set_ylabel("")
    ax.set_xlabel("")
    ax.set_title("K = "+str(i+1))

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(8, 8))
colors = ['#d6610a', '#207fb9', '#0aa076', '#d388ac', '#e9a32b', '#62b9e9']
for i, ax in enumerate(axes.flat):
    plot_with_intervals(ax, results[i], colors[i])
fig.suptitle("Total monthly eating-out expenditure")
fig.supylabel("$ billions")
fig.supxlabel("Month")
plt.show()
```

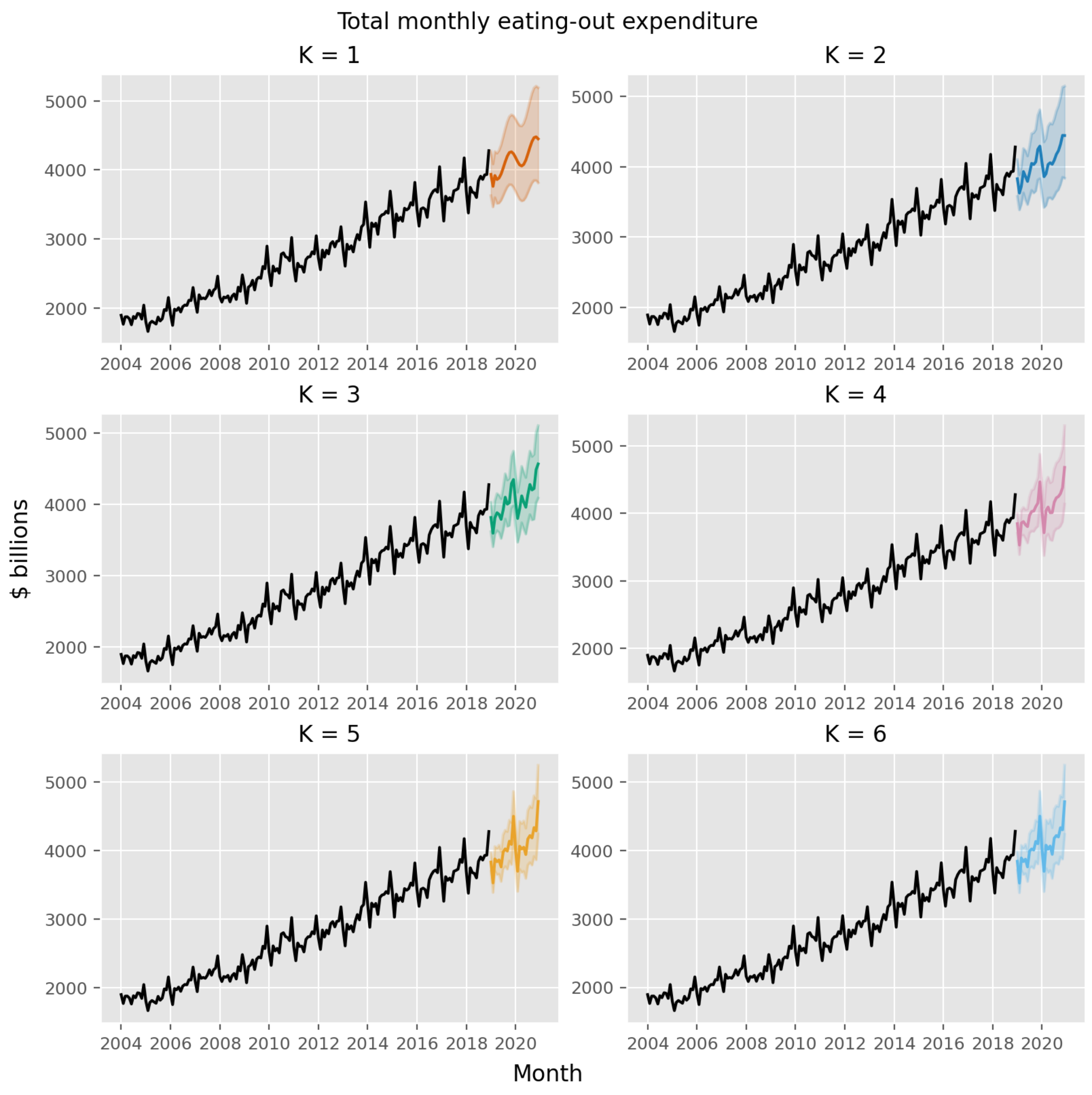

Figure 10.11: Using Fourier terms and ARIMA errors for forecasting monthly expenditure on eating out in Australia.

10.6 Lagged predictors

Sometimes, the impact of a predictor that is included in a regression model will not be simple and immediate. For example, an advertising campaign may impact sales for some time beyond the end of the campaign, and sales in one month will depend on the advertising expenditure in each of the past few months. Similarly, a change in a company's safety policy may reduce accidents immediately, but have a diminishing effect over time as employees take less care when they become familiar with the new working conditions.

In these situations, we need to allow for lagged effects of the predictor. Suppose that we have only one predictor in our model. Then a model which allows for lagged effects can be written as y_t = \beta_0 + \gamma_0x_t + \gamma_1 x_{t-1} + \dots + \gamma_k x_{t-k} + \eta_t, where \eta_t is an ARIMA process. The value of k can be selected using the AICc, along with the values of p and q for the ARIMA error.

Example: TV advertising and insurance quotations

A US insurance company advertises on national television in an attempt to increase the number of insurance quotations provided (and consequently the number of new policies). Figure 10.12 shows the number of quotations and the expenditure on television advertising for the company each month from January 2002 to April 2005.

```
insurance = pd.read_csv('../data/insurance.csv', parse_dates=['ds'])

fig, axes = plt.subplots(nrows=2, ncols=1)
sns.lineplot(data=insurance, x='ds', y='Quotes', ax=axes[0], color='black', label='Quotes')
sns.lineplot(data=insurance, x='ds', y='TVadverts', ax=axes[1], color='black', label='TVadverts')

fig.suptitle("Insurance advertising and quotations")
plt.show()
```

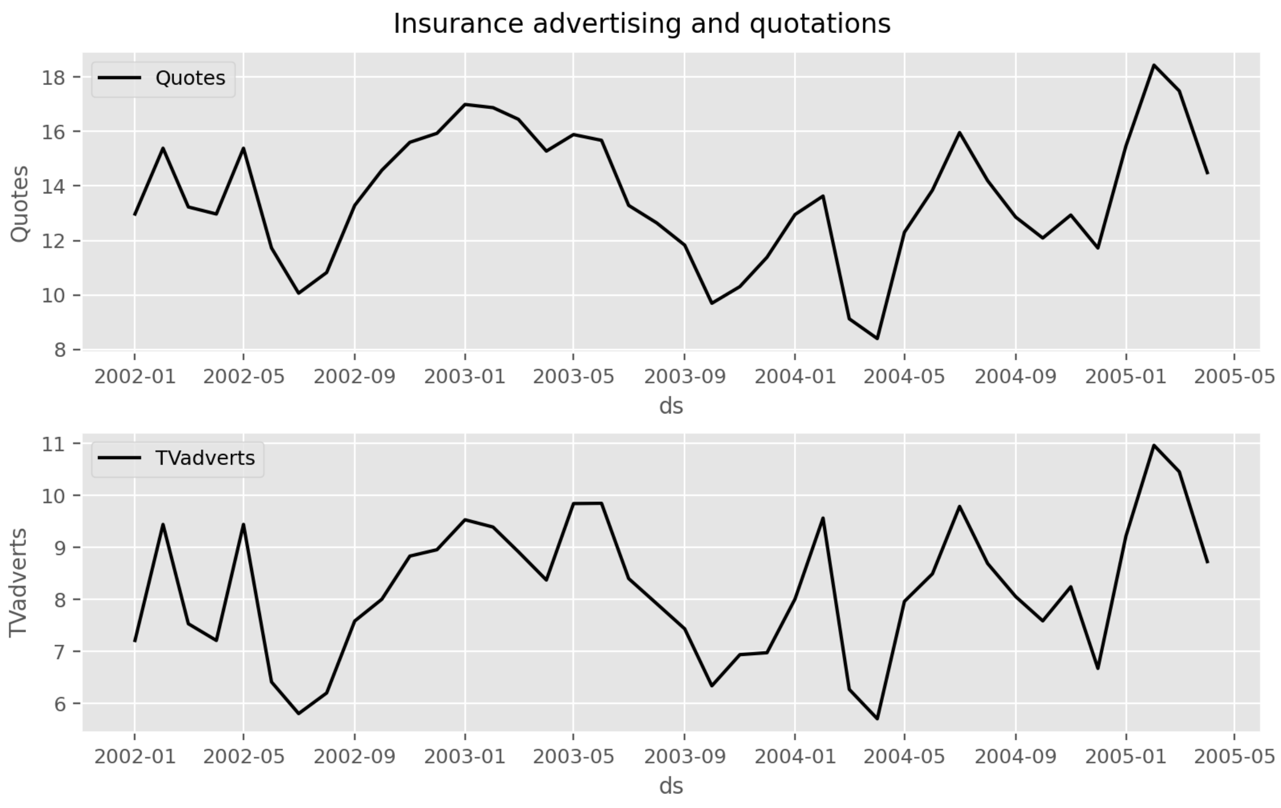

Figure 10.12: Numbers of insurance quotations provided per month and the expenditure on advertising per month.

We will consider including advertising expenditure for up to four months; that is, the model may include advertising expenditure in the current month, and the three months before that. When comparing models, it is important that they all use the same training set. In the following code, we exclude the first three months in order to make fair comparisons.

```python
insurance_model = insurance.copy(deep=True)
insurance_model["Quotes"][:4] = np.nan
for lag in [1, 2, 3]:
    insurance_model[f"TVadverts_{lag}"] = insurance_model["TVadverts"].shift(lag)

sf_1 = StatsForecast(
    models=[
        AutoARIMA(season_length=12, d=0),
    ],
    freq='MS'
)
sf_1.fit(
    df=insurance_model[["unique_id", "ds", "Quotes", "TVadverts"]],
    target_col="Quotes",
)

sf_2 = StatsForecast(
    models=[
        AutoARIMA(season_length=12, d=0),
    ],
    freq='MS'
)
sf_2.fit(
    df=insurance_model[["unique_id", "ds", "Quotes", "TVadverts", "TVadverts_1"]],
    target_col="Quotes",
)

sf_3 = StatsForecast(
    models=[
        AutoARIMA(season_length=12, d=0),
    ],
    freq='MS'
)
sf_3.fit(
    df=insurance_model[["unique_id", "ds", "Quotes", "TVadverts", "TVadverts_1", "TVadverts_2"]],

    target_col="Quotes",
)

sf_4 = StatsForecast(
    models=[
        AutoARIMA(season_length=12, d=0),
    ],
    freq='MS'
)
sf_4 = sf_4.fit(
    df=insurance_model[["unique_id", "ds", "Quotes", "TVadverts", "TVadverts_1", "TVadverts_2", "T

    target_col="Quotes",
)
```

Next we choose the optimal lag length for advertising based on the AICc.

```
pd.DataFrame({
    "model": ["lag0", "lag1", "lag2", "lag3"],
    "aicc":[
        sf_1.fitted_[0][0].model_["aicc"],
        sf_2.fitted_[0][0].model_["aicc"],
        sf_3.fitted_[0][0].model_["aicc"],
        sf_4.fitted_[0][0].model_["aicc"],
    ],

}).round(2)
```

	model	aicc
0	lag0	66.67
1	lag1	58.43
2	lag2	61.53
3	lag3	64.71

The best model (with the smallest AICc value) is `lag1` with two predictors; that is, it includes advertising only in the current month and the previous month. So we now re-estimate that model, but using all the available data.

```
for lag in [1, 2, 3]:
    insurance[f"TVadverts_{lag}"] = insurance["TVadverts"].shift(lag)
sf = StatsForecast(
    models=[
        AutoARIMA(season_length=1, d=0),
    ],
    freq='MS'
)
sf.fit(
    df=insurance[["unique_id", "ds", "Quotes", "TVadverts", "TVadverts_1"]].dropna(),
    target_col="Quotes",
)
print(ARIMASummary(sf.fitted_[0, 0].model_))
sf.fitted_[0, 0].model_['coef']["mean"] = sf.fitted_[0, 0].model_['coef'].pop("intercept")
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}")
```

```
Regression with ARIMA(1,0,2) errors
Coefficients: {'ar1': 0.5123159565701185, 'ma1': 0.9169305818168708, 'ma2': 0.4591374664104491, 'ex_1': 1.25265
21058620168, 'ex_2': 0.1463678171900414, 'mean': 2.1554426362910095}
sigma^2     : 0.22
loglik      : -23.94
aic         : 61.88
aicc        : 65.49
bic         : 73.52
```

The chosen model has ARIMA(1,0,2) errors. The model can be written as y_t = 2.155 + 1.253 x_t + 0.146 x_{t-1} + \eta_t, where y_t is the number of quotations provided in month t, x_t is the advertising expenditure in month t,

\eta_t = 0.512 \eta_{t-1} + \varepsilon_t + 0.917 \varepsilon_{t-1} + 0.459 \varepsilon_{t-2}

and \varepsilon_t is white noise.

We can calculate forecasts using this model if we assume future values for the advertising variable. If we set the future monthly advertising to 8 units, we get the forecasts in Figure 10.13.

```
future_predictors = make_future_dataframe(
    insurance["unique_id"].unique(),
    pd.to_datetime([insurance["ds"].max()]),
    freq="MS",
    h=12,
)
future_predictors["TVadverts"] = 8
future_predictors["TVadverts_1"] = 8
sf.fit(
    df=insurance[["unique_id", "ds", "Quotes", "TVadverts", "TVadverts_1"]].dropna(),
    target_col="Quotes",
)
res = sf.predict(h=12, X_df=future_predictors, level=[80,95])

plot_series(insurance, res, level=[80,95],
            target_col="Quotes",
            xlabel="Month",
            ylabel="Quotes",
            title="Forecast quotes with future advertising set to 8",
            rm_legend=False,)
```

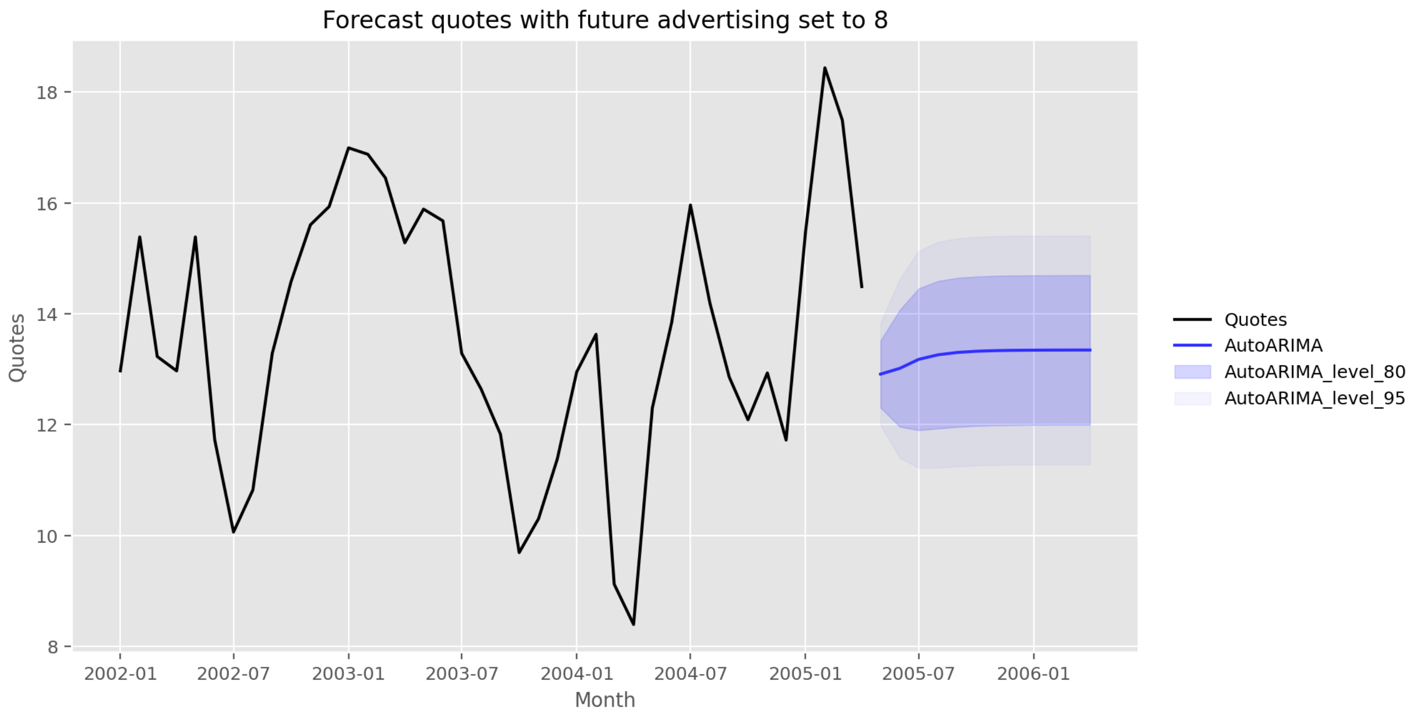

Figure 10.13: Forecasts of monthly insurance quotes, assuming that the future advertising expenditure is 8 units in each future month.

10.7 Exercises

1. This exercise uses data set `LakeHuron` giving the level of Lake Huron from 1875–1972.

   a. Load and plot the data.

   ```
   df = pd.read_csv('../data/lake_huron.csv', parse_dates=['ds'])
   ```

   b. Fit a linear trend model to the Lake Huron data with an ARMA error structure.
   c. Forecast the level for the next 30 years. Do you think the extrapolated linear trend is realistic?

2. Repeat Exercise 4 from Section 7.10, but this time adding in ARIMA errors to address the autocorrelations in the residuals.

   a. How much difference does the ARIMA error process make to the forecasts?
   b. Check the residuals of the ARIMA model to ensure it has adequately addressed the autocorrelations in the data.

3. Repeat the daily electricity example, but add a indicator variable called `Temperature2` with a value of 1 when the temperature is equal or greater than 25 degree Celsius, and zero otherwise.

```
vic_elec_daily['Temperature2'] = (vic_elec_daily['Temperature'] >= 25).astype(int)
```

4. This exercise concerns `aus_accommodation`: the total quarterly takings from accommodation and the room occupancy level for hotels, motels, and guest houses in Australia, between January 1998 and June 2016. Total quarterly takings are in millions of Australian dollars.

   a. Compute the CPI-adjusted takings and plot the result for each state
   b. For each state, fit a dynamic regression model of CPI-adjusted takings with seasonal dummy variables, a linear trend model and ARIMA errors.
   c. Check that the residuals of the ARIMA model look like white noise.
   d. Forecast the takings for each state to the end of 2017. (Hint: You will need to produce forecasts of the CPI first.)
   e. What sources of uncertainty have not been taken into account in the prediction intervals?

5. We fitted a harmonic regression model to part of the `us_gasoline` series in Exercise 5 in Section 7.10. We will now revisit this model, and extend it to include more data and ARMA errors.

   a. Using `MLForecast`, fit a harmonic regression with a trend to the full series. Select the appropriate number of Fourier terms empirically.
   b. Now include ARIMA errors, keeping the same predictor variables as you used before.
   c. Check the residuals of the ARIMA model. Do they look sufficiently like white noise to continue? If not, try modifying your model, or removing the first few years of data.
   d. Once you have a model with white noise residuals, produce forecasts for the next year.

6. Electricity consumption is often modelled as a function of temperature. Temperature is measured by daily heating degrees and cooling degrees. Heating degrees is $18^\circ C$ minus the average daily temperature when the daily average is below $18^\circ C$; otherwise it is zero. This provides a measure of our need to heat ourselves as temperature falls. Cooling degrees measures our need to cool ourselves as the temperature rises. It is defined as the average daily temperature minus $18^\circ C$ when the daily average is above $18^\circ C$; otherwise it is zero. Let y_t denote the monthly total of kilowatt-hours of electricity used, let $x_{1,t}$ denote the monthly total of heating degrees, and let $x_{2,t}$ denote the monthly total of cooling degrees.

   An analyst fits the following model to a set of such data: $y^*_t = \beta_1x^*_{1,t} + \beta_2x^*_{2,t} + \eta_t$, where $(1-\Phi_{1}B^{12} - \Phi_{2}B^{24})(1-B)(1-B^{12})\eta_t = (1+\theta_1 B)\varepsilon_t$ and $y^*_t = \log(y_t)$, $x^*_{1,t} = \sqrt{x_{1,t}}$ and $x^*_{2,t}=\sqrt{x_{2,t}}$.

   a. What sort of ARIMA model is identified for η_t?

   b. The estimated coefficients are

Parameter	Estimate	s.e.	Z	P-value
β_1	0.0077	0.0015	4.98	0.000
β_2	0.0208	0.0023	9.23	0.000
θ_1	-0.5830	0.0720	8.10	0.000
Φ_{1}	-0.5373	0.0856	-6.27	0.000
Φ_{2}	-0.4667	0.0862	-5.41	0.000

   Explain what the estimates of β_1 and β_2 tell us about electricity consumption.

   c. Write the equation in a form more suitable for forecasting.
   d. Describe how this model could be used to forecast electricity demand for the next 12 months.
   e. Explain why the η_t term should be modelled with an ARIMA model rather than modelling the data using a standard regression package. In your discussion, comment on the properties of the estimates, the validity of the standard regression results, and the importance of the η_t model in producing forecasts.

7. For the retail time series considered in earlier chapters:

   a. Develop an appropriate dynamic regression model with Fourier terms for the seasonality. (You will probably need to use the same Box-Cox transformation you identified previously.)
   b. Check the residuals of the ARIMA model. Does the residual series look like white noise?
   c. Compare the forecasts with those you obtained earlier using alternative models.

10.8 Further reading

- A detailed discussion of dynamic regression models is provided in Pankratz (1991).
- A generalisation of dynamic regression models, known as “transfer function models”, is discussed in Box et al. (2015).

```
vic_elec_df = pd.read_csv("../data/vic_elec.csv")
vic_elec_df["ds"] = pd.to_datetime(vic_elec_df["ds"])
vic_elec_demand = vic_elec_df[vic_elec_df["unique_id"] == "Demand"].copy()
vic_elec_demand["hour-minute"] = vic_elec_demand["ds"].dt.strftime("%H:%M:%S")
vic_elec_demand["day"] = vic_elec_demand["ds"].dt.date

plt.figure()
sns.lineplot(
    data=vic_elec_demand,
    x="hour-minute",
    y="y",
    hue="day",
    palette="husl",
    legend=False,
)
unique_ticks = vic_elec_demand["hour-minute"].unique()
ticks_to_plot = unique_ticks[::2]
plt.xticks(ticks=range(0, len(unique_ticks), 2), labels=ticks_to_plot, rotation=45)
plt.title("Electricity Demand: Victoria")
plt.xlabel("Time")
plt.ylabel("MWh")
plt.show()
```

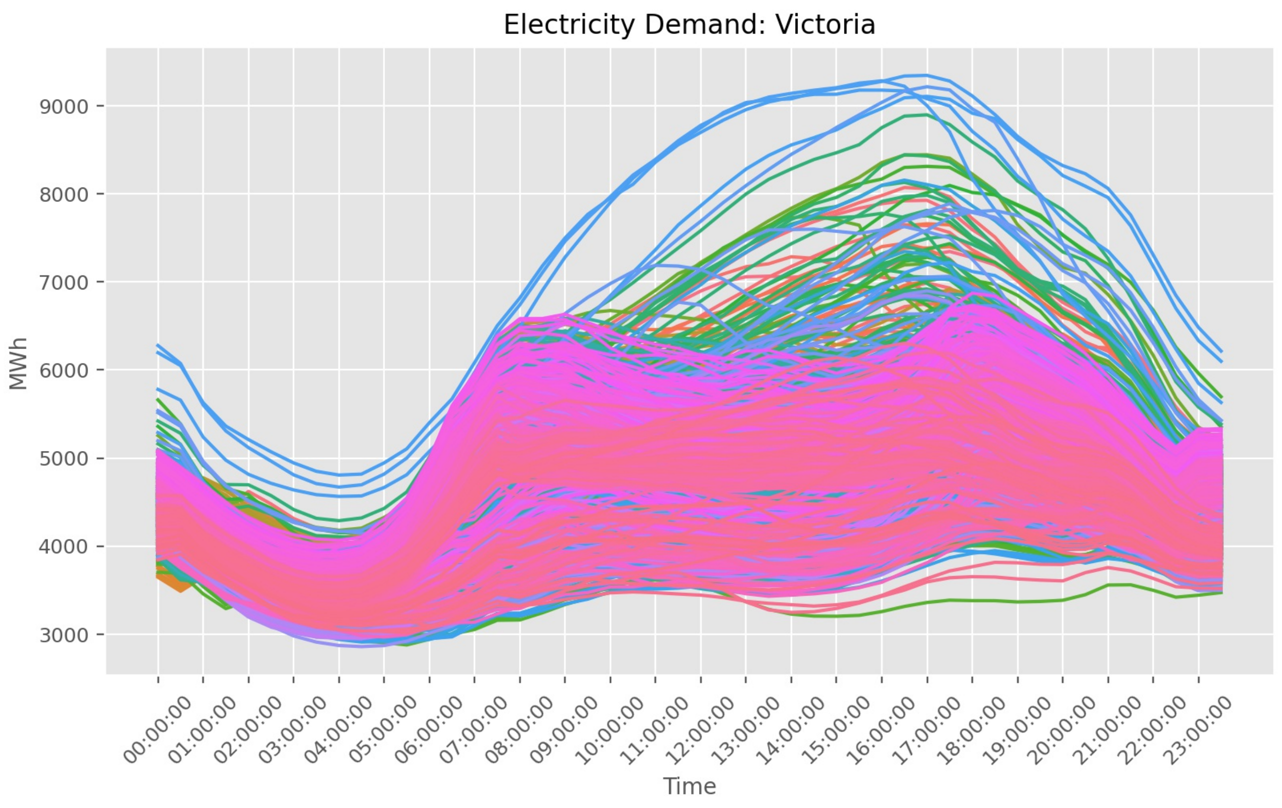

Figure 10.14: Seasonal plot showing daily seasonal patterns for Victorian electricity demand.

10.9 Used modules and classes

MLForecast

- `MLForecast` class - For machine learning based forecasting

StatsForecast

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

UtilsForecast

- `trend`, `fourier` functions - For feature engineering
- `plot_series` utility - For time series visualization

. Forecasting with cointegrated models is discussed by Harris and Sollis (2003). ↩︎

2. The term “dynamic harmonic regression” is also used for a harmonic regression with time-varying parameters (Young, Pedregal, and Tych 1999). ↩︎

← Chapter 9 ARIMA models

Chapter 11 Forecasting hierarchical and grouped time series →

Published by OTexts using Quarto. © 2025