In this chapter we discuss regression models. The basic concept is that we forecast the time series of interest y assuming that it has a linear relationship with other time series x.

For example, we might wish to forecast monthly sales y using total advertising spend x as a predictor. Or we might forecast daily electricity demand y using temperature x_1 and the day of week x_2 as predictors.

The **forecast variable** y is sometimes also called the regressand, dependent or explained variable. The **predictor variables** x are sometimes also called the regressors, independent or explanatory variables. In this book we will always refer to them as the "forecast" variable and "predictor" variables.

7.1 The linear model

Simple linear regression

In the simplest case, the regression model allows for a linear relationship between the forecast variable y and a single predictor variable x: y_t = \beta_0 + \beta_1 x_t + \varepsilon_t. An artificial example of data from such a model is shown in Figure 7.1. The coefficients \beta_0 and \beta_1 denote the intercept and the slope of the line respectively. The intercept \beta_0 represents the predicted value of y when x=0. The slope \beta_1 represents the average predicted change in y resulting from a one unit increase in x.

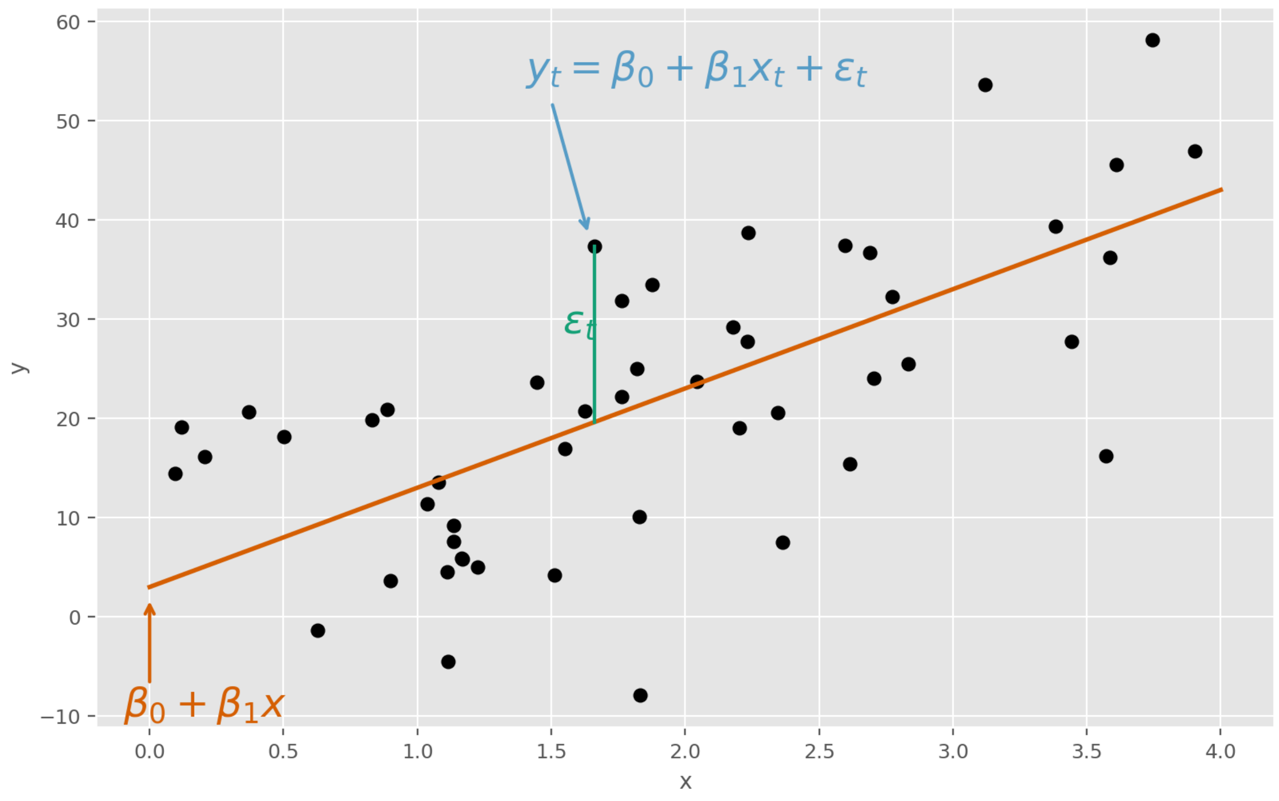

Figure 7.1: An example of data from a simple linear regression model.

Notice that the observations do not lie on the straight line but are scattered around it. We can think of each observation y_t as consisting of the systematic or explained part of the model, \beta_0+\beta_1x_t, and the random "error", \varepsilon_t. The "error" term does not imply a mistake, but a deviation from the underlying straight line model. It captures anything that may affect y_t other than x_t.

Example: US consumption expenditure

Figure 7.2 shows time series of quarterly percentage changes (growth rates) of real personal consumption expenditure, y, and real personal disposable income, x, for the US.

```python
# Read data
us_change = pd.read_csv("../data/US_change.csv", parse_dates=["ds"])
# Plot
fig, ax = plt.subplots()
ax.plot(us_change["ds"], us_change["Income"], label="Income")
ax.plot(us_change["ds"], us_change["y"], label="Consumption")
ax.set_xlabel("Quarter [1Q]")
ax.set_ylabel("% change")
ax.legend()
plt.show()
```

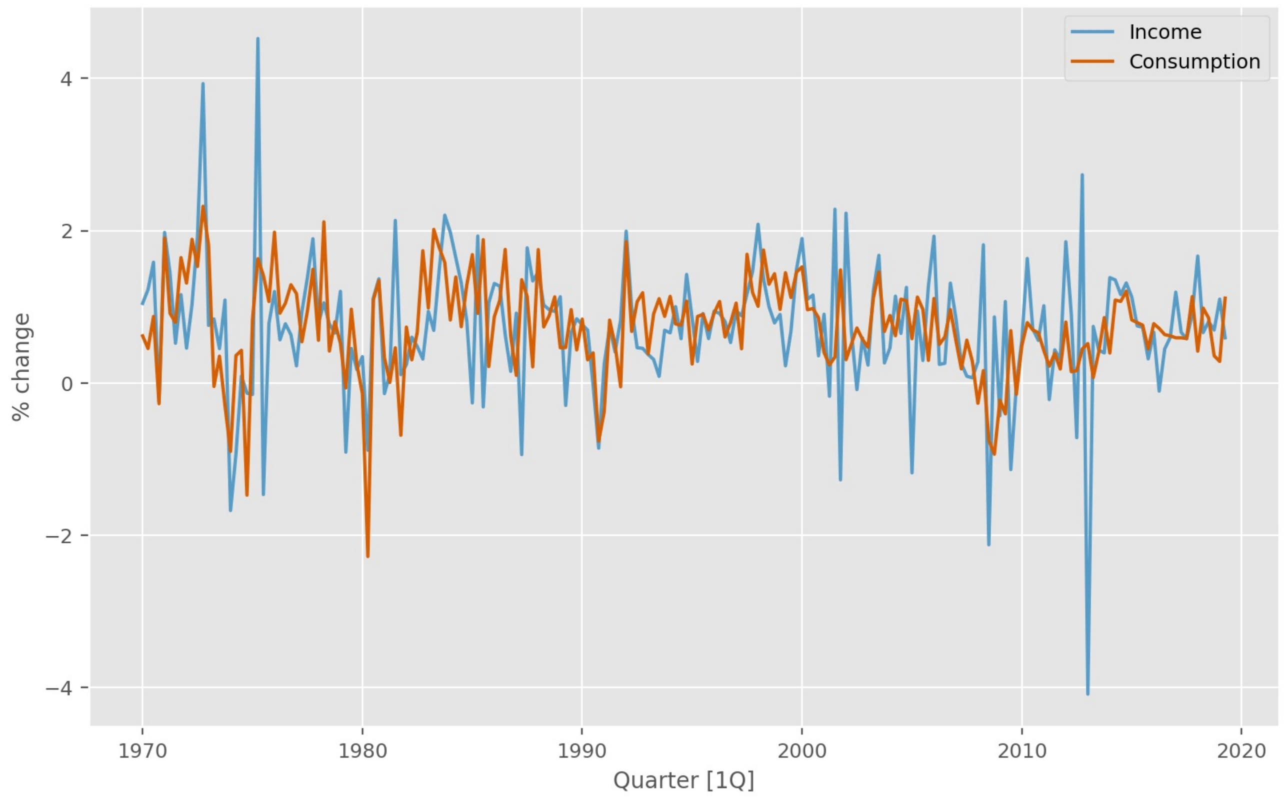

Figure 7.2: Percentage changes in personal consumption expenditure and personal income for the US.

A scatter plot of consumption changes against income changes is shown in Figure 7.3 along with the estimated regression line \hat{y}_t= 0.54 + 0.27 x_t. (We put a “hat” above y to indicate that this is the value of y predicted by the model.)

```python
# Plot
fig, ax = plt.subplots()
ax.scatter(us_change["Income"], us_change["y"], c="k")
ax.plot(us_change["Income"],
  mf.forecast_fitted_values()["LinearRegression"])

# Customize the plot
ax.set_xlabel("Income (quarterly % change)")
ax.set_ylabel("Consumption (quarterly % change)")
plt.show()
```

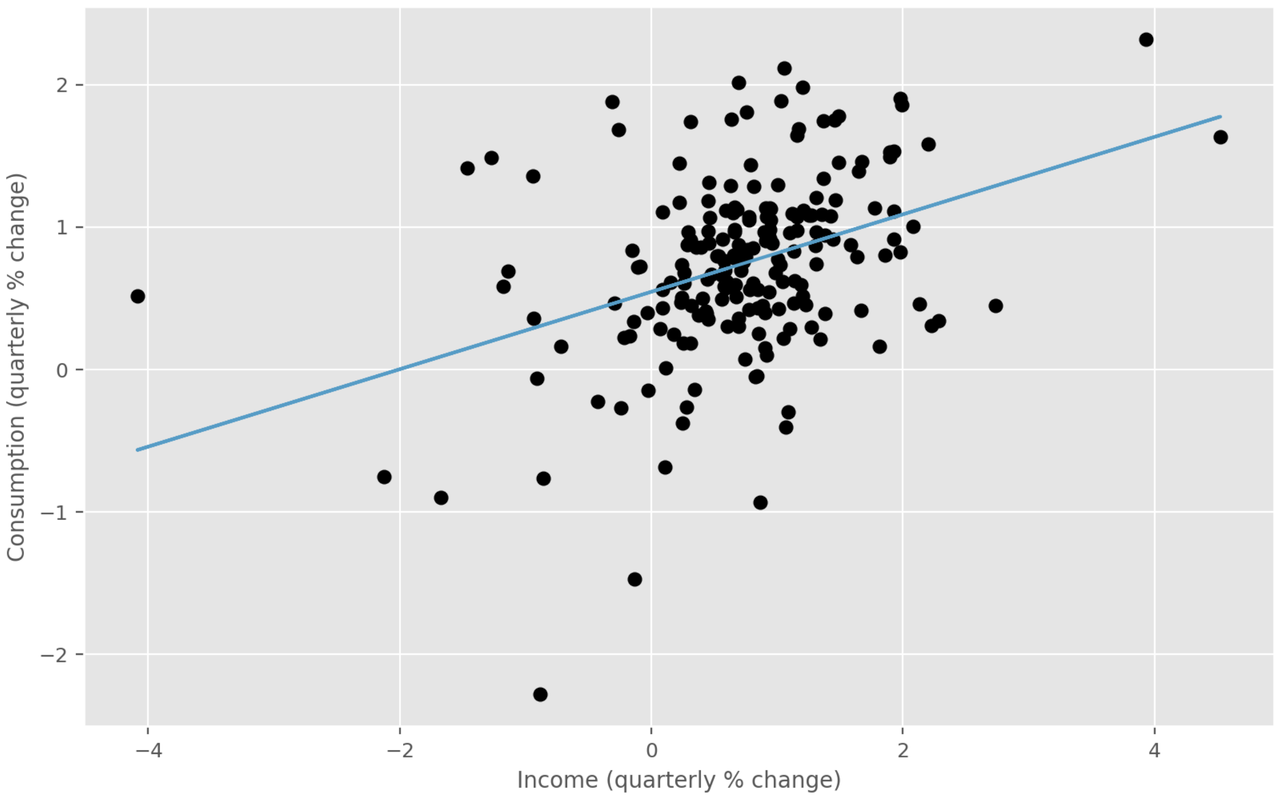

Figure 7.3: catterplot of quarterly changes in consumption expenditure versus quarterly changes in personal income and the fitted regression line.

The equation is estimated using the `LinearRegression()` class from `scikit-learn`:

```python
def print_regression_summary_from_model(model):
    X = model._X.values.astype(float)
    y = model._y
    residuals = model._residuals

    n, p = X.shape
    X_design = np.hstack([np.ones((n, 1)), X])
    df = n - p - 1

    res_summary = np.percentile(residuals, [0, 25, 50, 75, 100])
    print("#> Residuals:")
    print(f"#>     Min      1Q  Median      3Q     Max ")
    print(f"#> {res_summary[0]:7.4f} {res_summary[1]:7.4f} "
          f"{res_summary[2]:7.4f} {res_summary[3]:7.4f} "
          f"{res_summary[4]:7.4f}\n")

    coef = np.insert(model.coef_, 0, model.intercept_)
    rss = np.sum(residuals ** 2)
    mse = rss / df
    var_beta = mse * np.linalg.inv(X_design.T @ X_design).diagonal()
    se_beta = np.sqrt(var_beta)
    t_stats = coef / se_beta
    p_values = 2 * (1 - stats.t.cdf(np.abs(t_stats), df))

    print("#> Coefficients:")
    print(f"#> {'':>13} {'Estimate':>9} {'Std. Error':>11} {'t value':>9} "
          f"{'Pr(>|t|)':>9}")
    names = ['(Intercept)'] + model._var_names
    for name, est, se, t, p in zip(names, coef, se_beta, t_stats, p_values):
        stars = (
            '***' if p < 0.001 else
            '**' if p < 0.01 else
            '*' if p < 0.05 else
            '.' if p < 0.1 else
            ''
        )
        print(f"#> {name:>13} {est:9.4f} {se:11.4f} {t:9.2f} {p:9.3g} "
              f"{stars}")
    print("---")
    print("Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1")

    r_squared = model.score(X, y)
    adj_r_squared = 1 - (1 - r_squared) * (n - 1) / df
    f_stat = (r_squared / p) / ((1 - r_squared) / df)
    f_pval = 1 - stats.f.cdf(f_stat, p, df)

    print(f"\nResidual standard error: {np.sqrt(mse):.3f} "
          f"on {df} degrees of freedom")
    print(f"Multiple R-squared: {r_squared:.3f},   "
          f"Adjusted R-squared: {adj_r_squared:.3f}")
    print(f"F-statistic: {f_stat:.1f} on {p} and {df} DF, "
          f"p-value: {f_pval:.3g}")

print_regression_summary_from_model(mf.models_['LinearRegression'])
```

```
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.5824 -0.2778  0.0186  0.3233  1.4223

#> Coefficients:
#>              Estimate  Std. Error   t value  Pr(>|t|)
#>   (Intercept)   0.5445      0.0540     10.08          0 ***
#>        Income   0.2718      0.0467      5.82   2.4e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.591 on 196 degrees of freedom
Multiple R-squared: 0.147,   Adjusted R-squared: 0.143
F-statistic: 1408757523.9 on 2.402169752002692e-08 and 196 DF, p-value: 1.11e-16
```

We will discuss how `LinearRegression()` computes the coefficients in Section 7.2.

The fitted line has a positive slope, reflecting the positive relationship between income and consumption. The slope coefficient shows that a one unit increase in x (a 1 percentage point increase in personal disposable income) results on average in 0.27 units increase in y (an average increase of 0.27 percentage points in personal consumption expenditure). Alternatively the estimated equation shows that a value of 1 for x (the percentage increase in personal disposable income) will result in a forecast value of $0.54 + 0.27 \times 1 = 0.81$ for y (the percentage increase in personal consumption expenditure).

The interpretation of the intercept requires that a value of x=0 makes sense. In this case when x=0 (i.e., when there is no change in personal disposable income since the last quarter) the predicted value of y is 0.54 (i.e., an average increase in personal consumption expenditure of 0.54%). Even when x=0 does not make sense, the intercept is an important part of the model. Without it, the slope coefficient can be distorted unnecessarily. The intercept should always be included unless the requirement is to force the regression line "through the origin". In what follows we assume that an intercept is always included in the model.

Multiple linear regression

When there are two or more predictor variables, the model is called a **multiple regression model**. The general form of a multiple regression model is $y_t = \beta_{0} + \beta_{1} x_{1,t} + \beta_{2} x_{2,t} + \cdots + \beta_{k} x_{k,t} + \varepsilon_t, \tag{7.1}$ where y is the variable to be forecast and $x_{1},\dots,x_{k}$ are the k predictor variables. Each of the predictor variables must be numerical. The coefficients $\beta_{1},\dots,\beta_{k}$ measure the effect of each predictor after taking into account the effects of all the other predictors in the model. Thus, the coefficients measure the *marginal effects* of the predictor variables.

Example: US consumption expenditure

Figure 7.4 shows additional predictors that may be useful for forecasting US consumption expenditure. These are quarterly percentage changes in industrial production and personal savings, and quarterly changes in the unemployment rate (as this is already a percentage). Building a multiple linear regression model can potentially generate more accurate forecasts as we expect consumption expenditure to not only depend on personal income but on other predictors as well.

```python
fig, axes = plt.subplots(nrows=3, ncols=1, sharex=True)
sns.lineplot(data=us_change, x="ds", y="Production", ax=axes[0],
  color="#D55F03", label="Production")
sns.lineplot(data=us_change, x="ds", y="Savings", ax=axes[1],
  color="#569CC6", label="Savings")
sns.lineplot(data=us_change, x="ds", y="Unemployment", ax=axes[2],
  color="#13A076", label="Unemployment")
for ax in axes:
    ax.set_ylabel("")
    ax.set_xlabel("")
fig.supxlabel("% change")
fig.supylabel("Quarter")
plt.show()
```

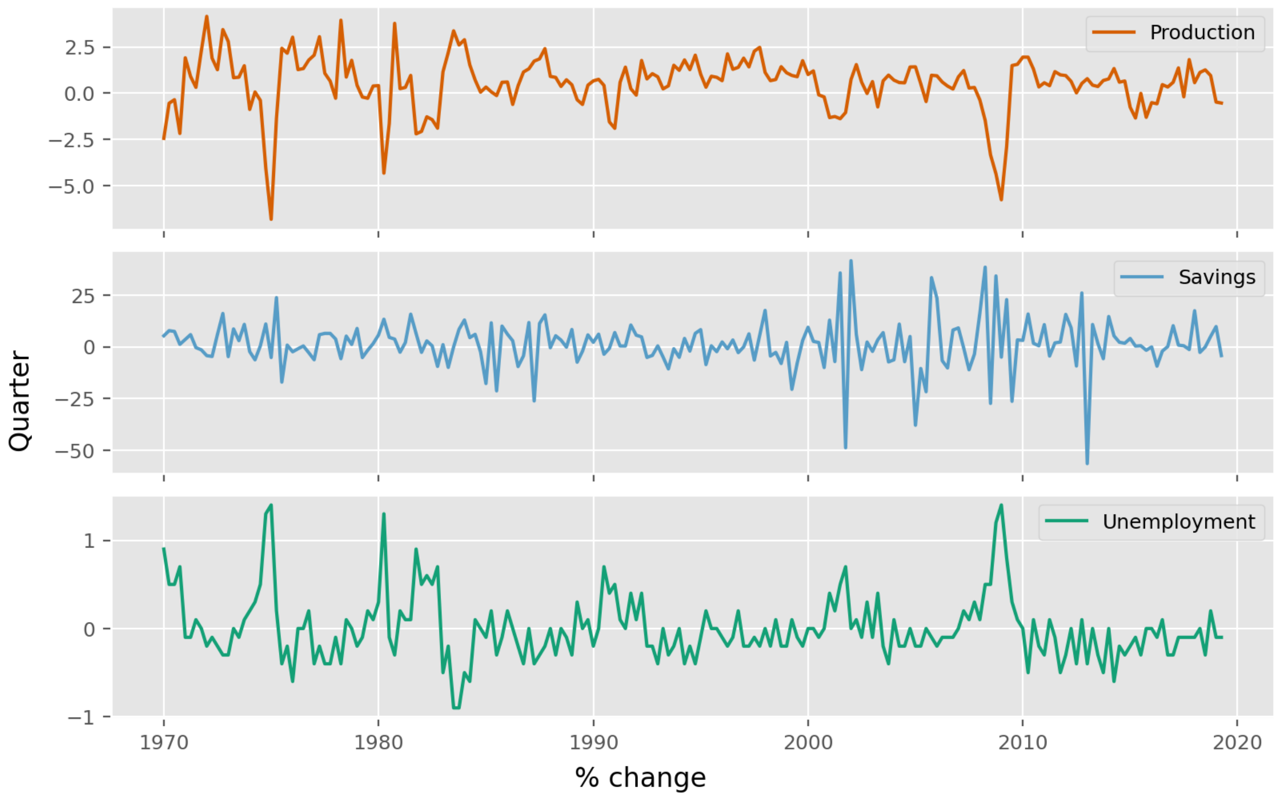

Figure 7.4: Quarterly percentage changes in industrial production and personal savings and quarterly changes in the unemployment rate for the US over the period 1970Q1-2019Q2.

Figure 7.5 is a scatterplot matrix of five variables. The first column shows the relationships between the forecast variable (consumption) and each of the predictors. The scatterplots show positive relationships with income and industrial production, and negative relationships with savings and unemployment. The strength of these relationships are shown by the correlation coefficients across the first row. The remaining scatterplots and correlation coefficients show the relationships between the predictors.

```python
plot_df = us_change[["y", "Income", "Production", "Savings",
  "Unemployment"]].copy()
plot_df = plot_df.rename(columns={'y': 'Consumption'})

def corrfunc(x, y, **kws):
    r, pvalue = pearsonr(x, y)
    ax = plt.gca()
    ax.annotate(
        f"Corr: {r:.3f}{'***' if pvalue < 0.05 else ''}",
        xy=(0.5, 0.5),
        xycoords="axes fraction",
        ha="center",
        va="center",
    )

g = sns.PairGrid(plot_df)
g.map_diag(sns.histplot)
g.map_upper(corrfunc)
g.map_lower(sns.scatterplot)
g.add_legend()
# Remove default axis labels
g.set(xlabel="")

# Move y-axis labels to the top
for i, col in enumerate(plot_df.columns):
    g.axes[0, i].set_title(col)
plt.show()
```

```
/home/hyndman/git/Books/fpppy/.venv/lib/python3.10/site-packages/seaborn/axisgrid.py:123: UserWarning: The figu
re layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

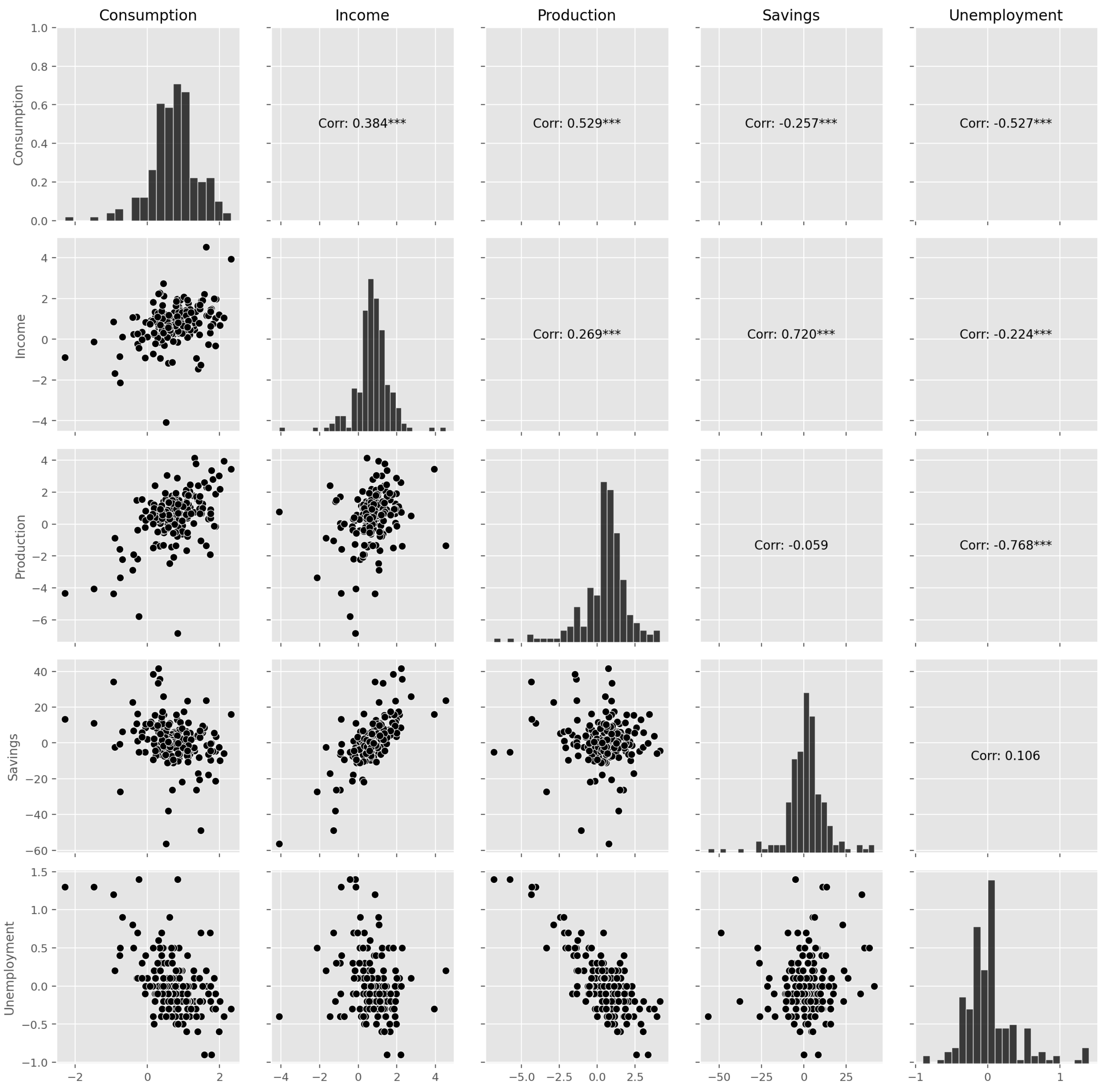

Figure 7.5: A scatterplot matrix of US consumption expenditure and the four predictors.

Assumptions

When we use a linear regression model, we are implicitly making some assumptions about the variables in Equation 7.1.

First, we assume that the model is a reasonable approximation to reality; that is, the relationship between the forecast variable and the predictor variables satisfies this linear equation.

Second, we make the following assumptions about the errors (\varepsilon_{1},\dots,\varepsilon_{T}):

- they have mean zero; otherwise the forecasts will be systematically biased.
- they are not autocorrelated; otherwise the forecasts will be inefficient, as there is more information in the data that can be exploited.
- they are unrelated to the predictor variables; otherwise there would be more information that should be included in the systematic part of the model.

It is also useful to have the errors being normally distributed with a constant variance \sigma^2 in order to easily produce prediction intervals.

Another important assumption in the linear regression model is that each predictor x is not a random variable. If we were performing a controlled experiment in a laboratory, we could control the values of each x (so they would not be random) and observe the resulting values of y. With observational data (including most data in business and economics), it is not possible to control the value of x, we simply observe it. Hence we make this an assumption.

7.2 Least squares estimation

In practice, of course, we have a collection of observations but we do not know the values of the coefficients $\beta_0,\beta_1, \dots, \beta_k$. These need to be estimated from the data.

The least squares principle provides a way of choosing the coefficients effectively by minimising the sum of the squared errors. That is, we choose the values of $\beta_0, \beta_1, \dots, \beta_k$ that minimise $\sum_{t=1}^T \varepsilon_t^2 = \sum_{t=1}^T (y_t - \beta_{0} - \beta_{1} x_{1,t} - \beta_{2} x_{2,t} - \cdots - \beta_{k} x_{k,t})^2$.

This is called **least squares** estimation because it gives the least value for the sum of squared errors. Finding the best estimates of the coefficients is often called “fitting” the model to the data, or sometimes “learning” or “training” the model. The line shown in Figure 7.3 was obtained in this way.

When we refer to the *estimated* coefficients, we will use the notation $\hat\beta_0, \dots, \hat\beta_k$. The equations for these will be given in Section 7.9.

The `LinearRegression()` class fits a linear regression model to time series data.

Example: US consumption expenditure

A multiple linear regression model for US consumption is

$$y_t=\beta_0 + \beta_1 x_{1,t}+ \beta_2 x_{2,t}+ \beta_3 x_{3,t}+ \beta_4 x_{4,t}+\varepsilon_t,$$

where y is the percentage change in real personal consumption expenditure, x_1 is the percentage change in real personal disposable income, x_2 is the percentage change in industrial production, x_3 is the percentage change in personal savings and x_4 is the change in the unemployment rate.

The following output provides information about the fitted model. `Coefficients` give an estimate of each β coefficient followed by its standard error (i.e., the standard deviation which would be obtained from repeatedly estimating the β coefficients on similar data sets). The standard error gives a measure of the uncertainty in the estimated β coefficient.

```python
mf = MLForecast(models=LinearRegression(), freq="QS-OCT")

# Fit model
mf.fit(us_change, fitted=True, static_features=[])

print_regression_summary_from_model(mf.models_['LinearRegression'])
```

```
#> Residuals:
#>      Min      1Q  Median      3Q     Max
#> -0.9055 -0.1582 -0.0361  0.1362  1.1547

#> Coefficients:
#>                Estimate  Std. Error   t value  Pr(>|t|)
#>   (Intercept)    0.2531      0.0345      7.34  5.71e-12 ***
#>        Income    0.7406      0.0401     18.46         0 ***
#>    Production    0.0472      0.0231      2.04    0.0429 *
#>       Savings   -0.0529      0.0029    -18.09         0 ***
#>  Unemployment   -0.1747      0.0955     -1.83    0.0689 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.310 on 193 degrees of freedom
Multiple R-squared: 0.768,   Adjusted R-squared: 0.763
F-statistic: 9281.0 on 0.06894899795561527 and 193 DF, p-value: 1.11e-16
```

Fitted values

Predictions of y can be obtained by using the estimated coefficients in the regression equation and setting the error term to zero. In general we write, \hat{y}_t = \hat\beta_{0} + \hat\beta_{1} x_{1,t} + \hat\beta_{2} x_{2,t} + \cdots + \hat\beta_{k} x_{k,t}. \tag{7.2} Plugging in the values of x_{1,t},\dots,x_{k,t} for t=1,\dots,T returns predictions of y_t within the training set, referred to as *fitted values*. Note that these are predictions of the data used to estimate the model, not genuine forecasts of future values of y.

The following plots show the actual values compared to the fitted values for the percentage change in the US consumption expenditure series. The time plot in Figure 7.6 shows that the fitted values follow the actual data fairly closely. This is verified by the strong positive relationship shown by the scatterplot in Figure 7.7.

```python
plot_series(us_change, mf.forecast_fitted_values().drop(columns="y"),
            xlabel="Quarter",
            ylabel="",
            title="Percent change is US consumption expenditure",
            rm_legend=False)
```

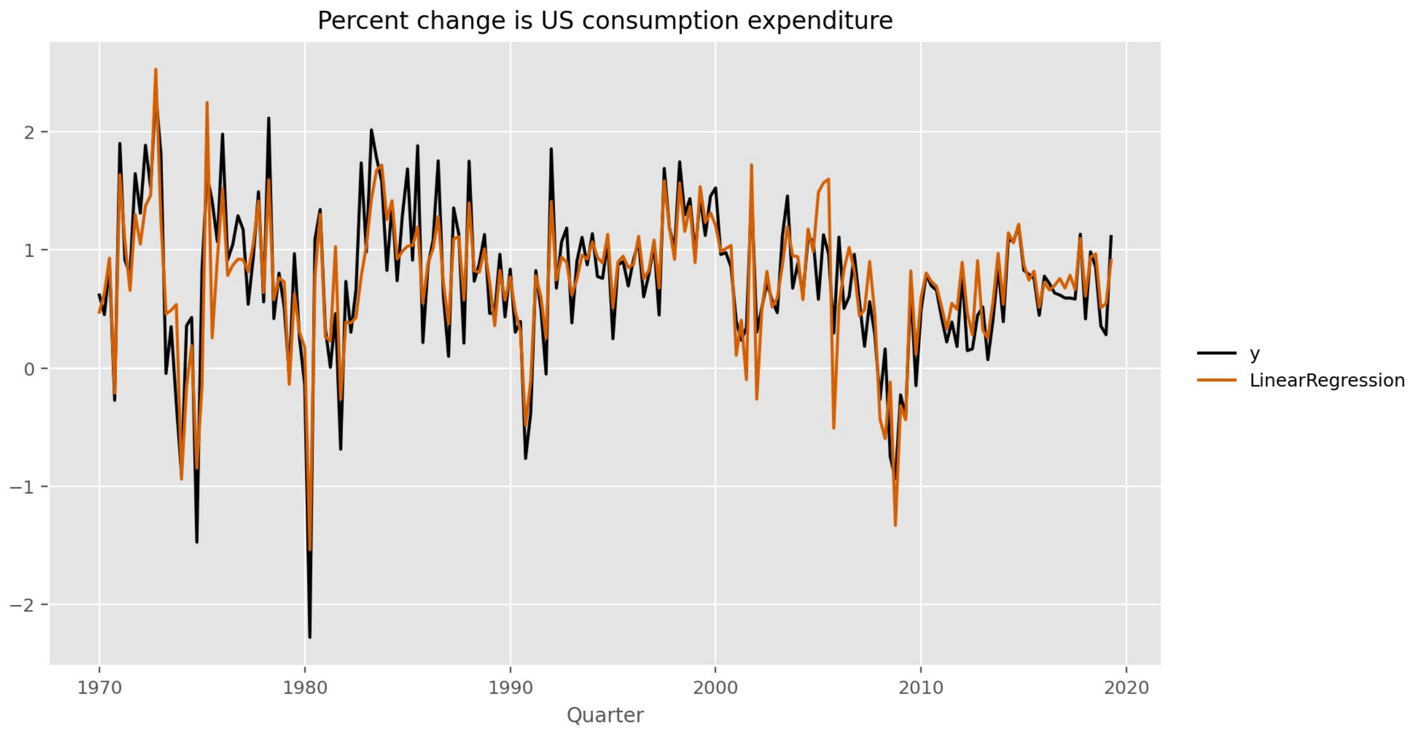

Figure 7.6: Time plot of actual US consumption expenditure and predicted US consumption expenditure.

```python
fig, ax = plt.subplots()
insample_forecasts = mf.forecast_fitted_values()["LinearRegression"]
ax.scatter(us_change["y"], insample_forecasts, c="k")
ax.axline((0, 0), (1, 1))
ax.set_xlabel("Data (actual values)")
ax.set_ylabel("Fitted (predictd values)")
ax.set_title("Percent change in US consumption expenditure")
plt.show()
```

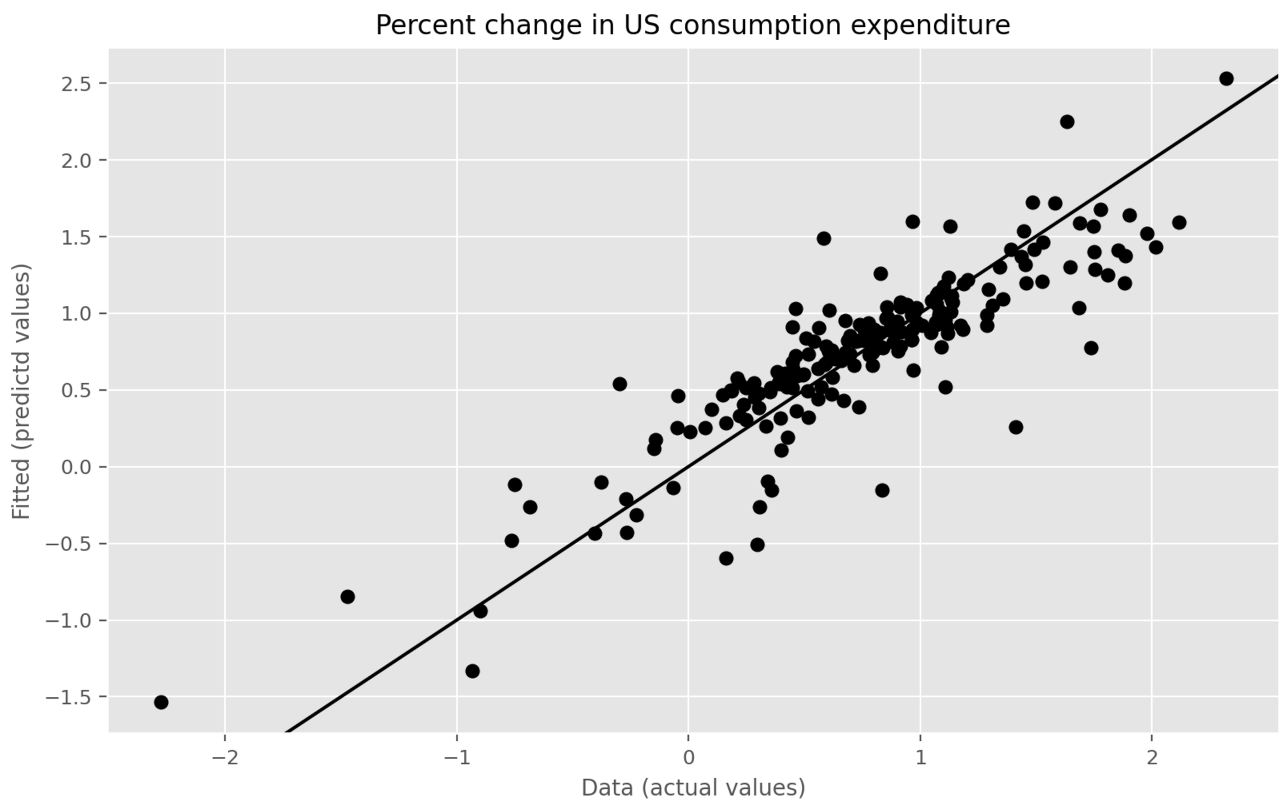

Figure 7.7: Actual US consumption expenditure plotted against predicted US consumption expenditure.

Goodness-of-fit

A common way to summarise how well a linear regression model fits the data is via the coefficient of determination, or R^2. This can be calculated as the square of the correlation between the observed y values and the predicted $\hat{y}$ values. Alternatively, it can also be calculated as, $R^2 = \frac{\sum(\hat{y}_{t} - \bar{y})^2}{\sum(y_{t}-\bar{y})^2}$, where the summations are over all observations. Thus, it reflects the proportion of variation in the forecast variable that is accounted for (or explained) by the regression model.

In simple linear regression, the value of R^2 is also equal to the square of the correlation between y and x (provided an intercept has been included).

If the predictions are close to the actual values, we would expect R^2 to be close to 1. On the other hand, if the predictions are unrelated to the actual values, then $R^2=0$ (again, assuming there is an intercept). In all cases, R^2 lies between 0 and 1.

The R^2 value is used frequently, though often incorrectly, in forecasting. The value of R^2 will never decrease when adding an extra predictor to the model and this can lead to over-fitting. There are no set rules for what is a good R^2 value, and typical values of R^2 depend on the type of data used. Validating a model’s forecasting performance on the test data is much better than measuring the R^2 value on the training data.

Example: US consumption expenditure

Figure 7.7 plots the actual consumption expenditure values versus the fitted values. The correlation between these variables is $r=0.768$ hence $R^2= 0.768$ (shown in the output above). In this case, the model does an excellent job as it explains 76.8% of the variation in the consumption data. Compare that to the R^2 value of 0.147 obtained from the simple regression with the same data set in Section 7.1. Adding the three extra predictors has allowed a lot more of the variation in the consumption data to be explained.

Standard error of the regression

Another measure of how well the model has fitted the data is the standard deviation of the residuals, which is often known as the “residual standard error”. This is shown in the above output with the value 0.31.

It is calculated using $\hat{\sigma}_e=\sqrt{\frac{1}{T-k-1}\sum_{t=1}^{T}{e_t^2}}, \tag{7.3}$ where k is the number of predictors in the model. Notice that we divide by $T-k-1$ because we have estimated $k+1$ parameters (the intercept and a coefficient for each predictor variable) in computing the residuals.

The standard error is related to the size of the average error that the model produces. We can compare this error to the sample mean of y or with the standard deviation of y to gain some perspective on the accuracy of the model.

The standard error will be used when generating prediction intervals, discussed in Section 7.6.

7.3 Evaluating the regression model

The differences between the observed y values and the corresponding fitted \hat{y} values are the training-set errors or “residuals” defined as, \begin{align*} e_t &= y_t - \hat{y}_t \\ &= y_t - \hat\beta_{0} - \hat\beta_{1} x_{1,t} - \hat\beta_{2} x_{2,t} - \cdots - \hat\beta_{k} x_{k,t} \end{align*} for t=1,\dots,T. Each residual is the unpredictable component of the associated observation.

The residuals have some useful properties including the following two: \sum_{t=1}^{T}{e_t}=0 \quad\text{and}\quad \sum_{t=1}^{T}{x_{k,t}e_t}=0\qquad\text{for all k}. As a result of these properties, it is clear that the average of the residuals is zero, and that the correlation between the residuals and the observations for the predictor variable is also zero. (This is not necessarily true when the intercept is omitted from the model.)

After selecting the regression variables and fitting a regression model, it is necessary to plot the residuals to check that the assumptions of the model have been satisfied. There are a series of plots that should be produced in order to check different aspects of the fitted model and the underlying assumptions. We will now discuss each of them in turn.

ACF plot of residuals

With time series data, it is highly likely that the value of a variable observed in the current time period will be similar to its value in the previous period, or even the period before that, and so on. Therefore when fitting a regression model to time series data, it is common to find autocorrelation in the residuals. In this case, the estimated model violates the assumption of no autocorrelation in the errors, and our forecasts may be inefficient — there is some information left over which should be accounted for in the model in order to obtain better forecasts. The forecasts from a model with autocorrelated errors are still unbiased, and so they are not “wrong”, but they will usually have larger prediction intervals than they need to. Therefore we should always look at an ACF plot of the residuals.

Histogram of residuals

It is always a good idea to check whether the residuals are normally distributed. As we explained earlier, this is not essential for forecasting, but it does make the calculation of prediction intervals much easier.

Example

Using the `plot_diagnostics()` function introduced in Section 5.3, we can obtain all the useful residual diagnostics mentioned above.

```
plot_diagnostics(forecaster=mf, n_lags=22)
```

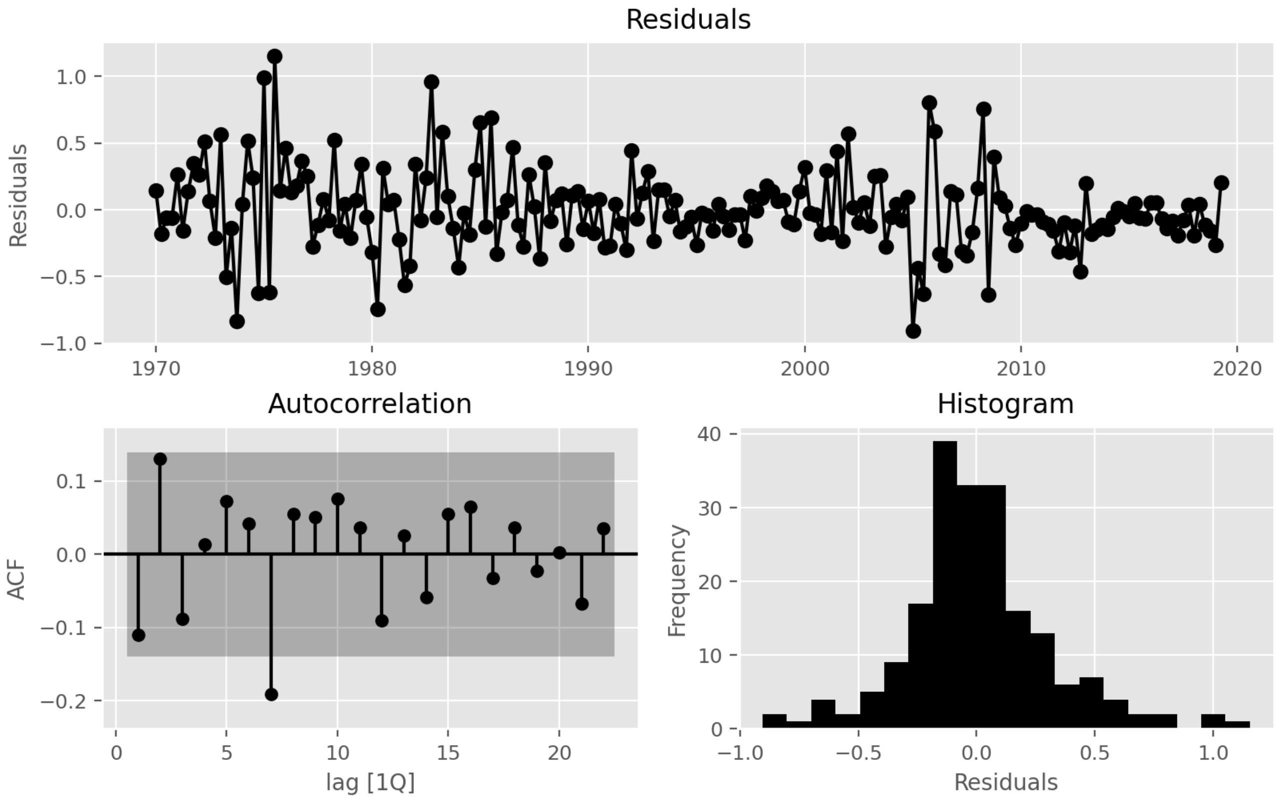

Figure 7.8: Analysing the residuals from a regression model for US quarterly consumption.

```
residuals = mf.models_["LinearRegression"]._residuals
acorr_ljungbox(residuals, lags=[10])
```

	lb_stat	lb_pvalue
10	18.865322	0.042007

The time plot shows some changing variation over time, but is otherwise relatively unremarkable. This heteroscedasticity will potentially make the prediction interval coverage inaccurate.

The histogram shows that the residuals seem to be slightly skewed, which may also affect the coverage probability of the prediction intervals.

The autocorrelation plot shows a significant spike at lag 7, and a significant Ljung-Box test at the 5% level. However, the autocorrelation is not particularly large, and at lag 7 it is unlikely to have any noticeable impact on the forecasts or the prediction intervals. In Chapter 10 we discuss dynamic regression models used for better capturing information left in the residuals.

Residual plots against predictors

We would expect the residuals to be randomly scattered without showing any systematic patterns. A simple and quick way to check this is to examine scatterplots of the residuals against each of the predictor variables. If these scatterplots show a pattern, then the relationship may be nonlinear and the model will need to be modified accordingly. See Section 7.7 for a discussion of nonlinear regression.

It is also necessary to plot the residuals against any predictors that are *not* in the model. If any of these show a pattern, then the corresponding predictor may need to be added to the model (possibly in a nonlinear form).

Example

The residuals from the multiple regression model for forecasting US consumption plotted against each predictor in Figure 7.9 seem to be randomly scattered. Therefore we are satisfied with these in this case.

```
fig, ax = plt.subplots(nrows=2, ncols=2)

ax[0, 0].scatter(us_change["Income"], residuals)
ax[0, 0].set_title("Income")

ax[0, 1].scatter(us_change["Production"], residuals)
ax[0, 1].set_title("Production")

ax[1, 0].scatter(us_change["Savings"], residuals)
ax[1, 0].set_title("Savings")

ax[1, 1].scatter(us_change["Unemployment"], residuals)
ax[1, 1].set_title("Unemployment")

fig.supylabel("Residuals")
plt.show()
```

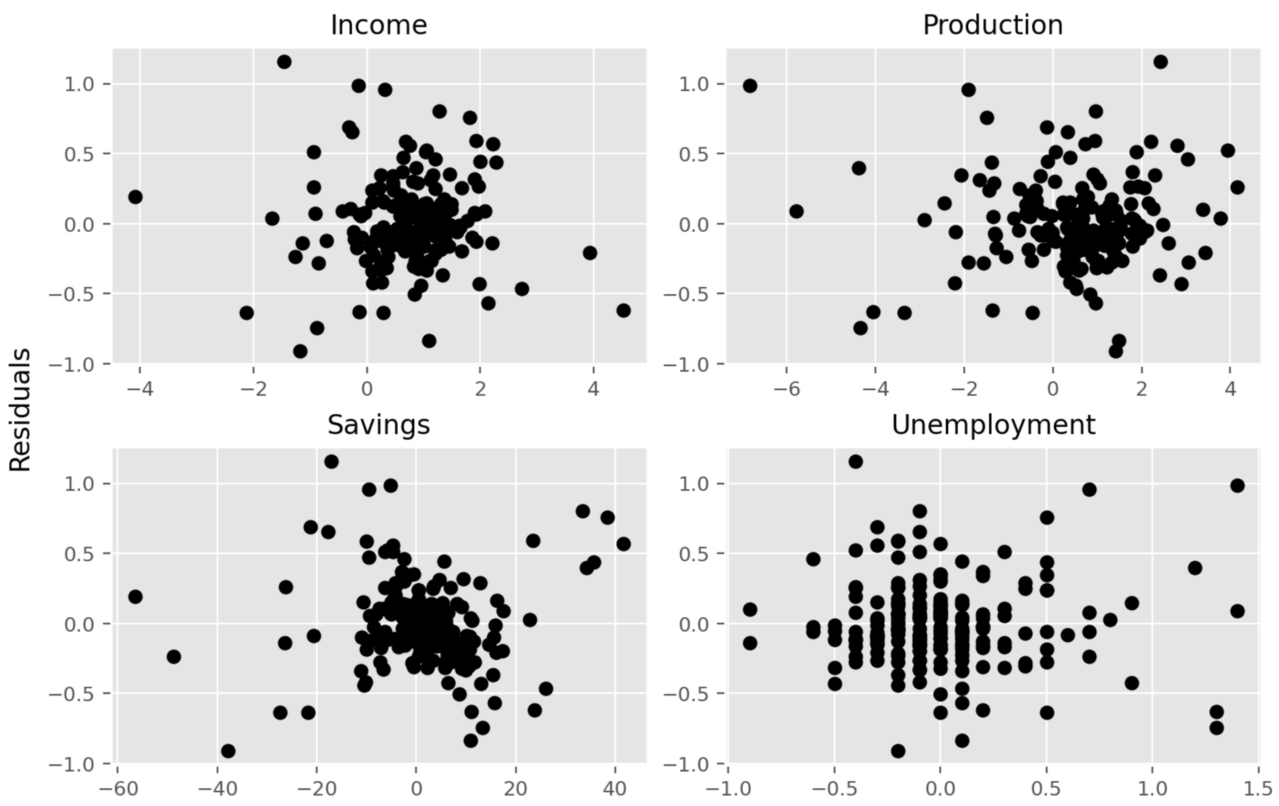

Figure 7.9: Scatterplots of residuals versus each predictor.

Residual plots against fitted values

A plot of the residuals against the fitted values should also show no pattern. If a pattern is observed, there may be “heteroscedasticity” in the errors which means that the variance of the residuals may not be constant. If this problem occurs, a transformation of the forecast variable such as a logarithm or square root may be required (see Section 3.1).

Example

Continuing the previous example, Figure 7.10 shows the residuals plotted against the fitted values. The random scatter suggests the errors are homoscedastic.

```
fig, ax = plt.subplots()
ax.scatter(insample_forecasts, residuals)
ax.set_xlabel("Fitted")
ax.set_ylabel("Residuals")
plt.show()
```

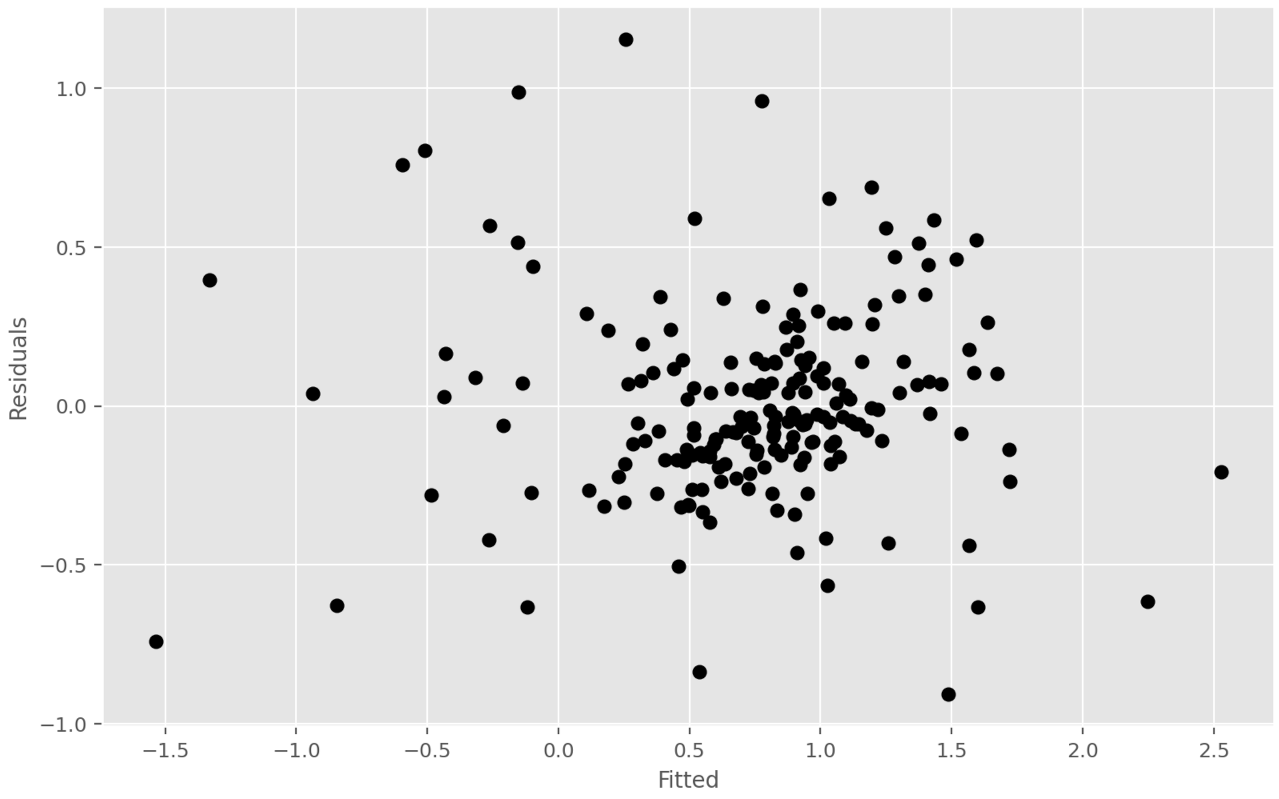

Figure 7.10: Scatterplots of residuals versus fitted values.

Outliers and influential observations

Observations that take extreme values compared to the majority of the data are called **outliers**. Observations that have a large influence on the estimated coefficients of a regression model are called **influential observations**. Usually, influential observations are also outliers that are extreme in the x direction.

There are formal methods for detecting outliers and influential observations that are beyond the scope of this textbook. As we suggested at the beginning of Chapter 2, becoming familiar with your data prior to performing any analysis is of vital importance. A scatter plot of y against each x is always a useful starting point in regression analysis, and often helps to identify unusual observations.

One source of outliers is incorrect data entry. Simple descriptive statistics of your data can identify minima and maxima that are not sensible. If such an observation is identified, and it has been recorded incorrectly, it should be corrected or removed from the sample immediately.

Outliers also occur when some observations are simply different. In this case it may not be wise for these observations to be removed. If an observation has been identified as a likely outlier, it is important to study it and analyse the possible reasons behind it. The decision to remove or retain an observation can be a challenging one (especially when outliers are influential observations). It is wise to report results both with and without the removal of such observations.

Example

Figure 7.11 highlights the effect of a single outlier when regressing US consumption on income (the example introduced in Section 7.1). In the left panel the outlier is only extreme in the direction of y, as the percentage change in consumption has been incorrectly recorded as -4%. The orange line is the regression line fitted to the data which includes the outlier, compared to the black line which is the line fitted to the data without the outlier. In the right panel the outlier now is also extreme in the direction of x with the 4% decrease in consumption corresponding to a 6% increase in income. In this case the outlier is extremely influential as the orange line now deviates substantially from the black line.

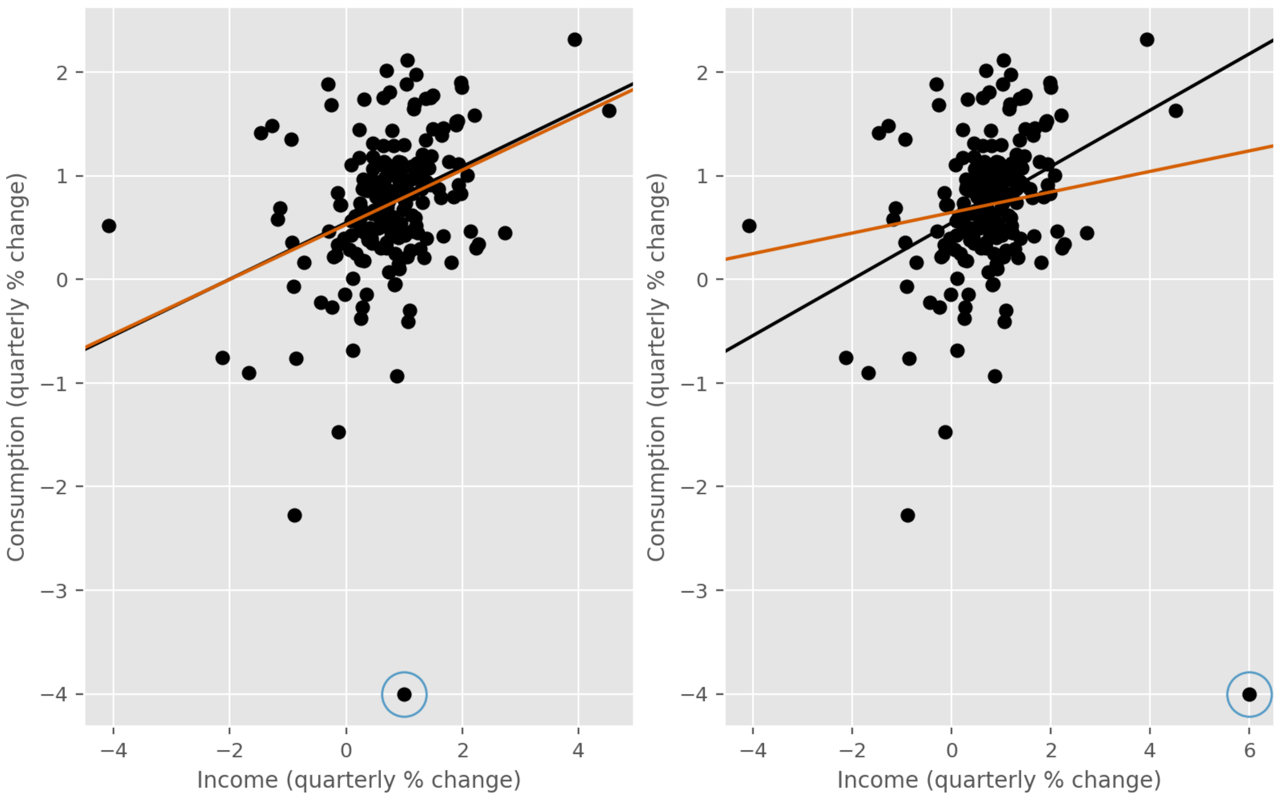

Figure 7.11: The effect of outliers and influential observations on regression.

Spurious regression

More often than not, time series data are "non-stationary"; that is, the values of the time series do not fluctuate around a constant mean or with a constant variance. We will deal with time series stationarity in more detail in Chapter 9, but here we need to address the effect that non-stationary data can have on regression models.

For example, consider the two variables plotted in Figure 7.12. These appear to be related simply because they both trend upwards in the same manner. However, air passenger traffic in Australia has nothing to do with rice production in Guinea.

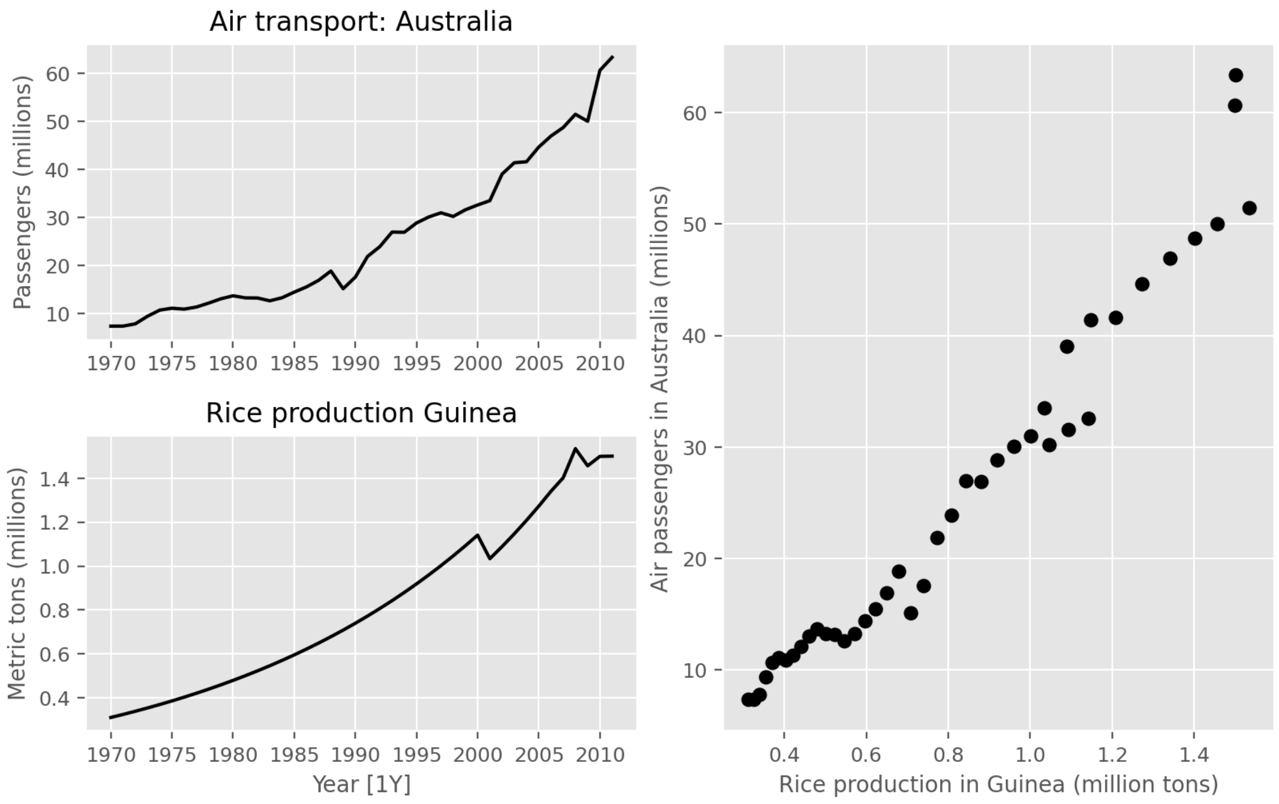

Figure 7.12: Trending time series data can appear to be related, as shown in this example where air passengers in Australia are regressed against rice production in Guinea.

Regressing non-stationary time series can lead to spurious regressions. The output of regressing Australian air passengers on rice production in Guinea is shown in Figure 7.13. High R^2 and high residual autocorrelation can be signs of spurious regression. Notice these features in the output below. Cases of spurious regression might appear to give reasonable short-term forecasts, but they will generally not continue to work into the future.

```
mf = MLForecast(models=LinearRegression(), freq="QS-OCT")

aus_airpassengers["unique_id"] = "AirPassengers"
mf.fit(
    aus_airpassengers,
    fitted=True,
    time_col="Year",
    target_col="Passengers",
    static_features=[],
)

insample_forecasts = mf.forecast_fitted_values()["LinearRegression"]
residuals = (
    mf.forecast_fitted_values()["Passengers"]
    - mf.forecast_fitted_values()["LinearRegression"]
)
r2 = r2_score(mf.forecast_fitted_values()["Passengers"], insample_forecasts)

print_regression_summary_from_model(mf.models_['LinearRegression'])
```

```
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -5.9448 -1.8917 -0.3272  1.8620 10.4210

#> Coefficients:
#>                Estimate  Std. Error   t value  Pr(>|t|)
#>   (Intercept)   -7.4925      1.2029     -6.23  2.25e-07 ***
#>    Production   40.2879      1.3369     30.13         0 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.239 on 40 degrees of freedom
Multiple R-squared: 0.958,   Adjusted R-squared: 0.957
F-statistic: inf on 0.0 and 40 DF, p-value: nan
```

```
plot_diagnostics(forecaster=mf, target_col="Passengers", time_col="Year",
  n_lags=20)
```

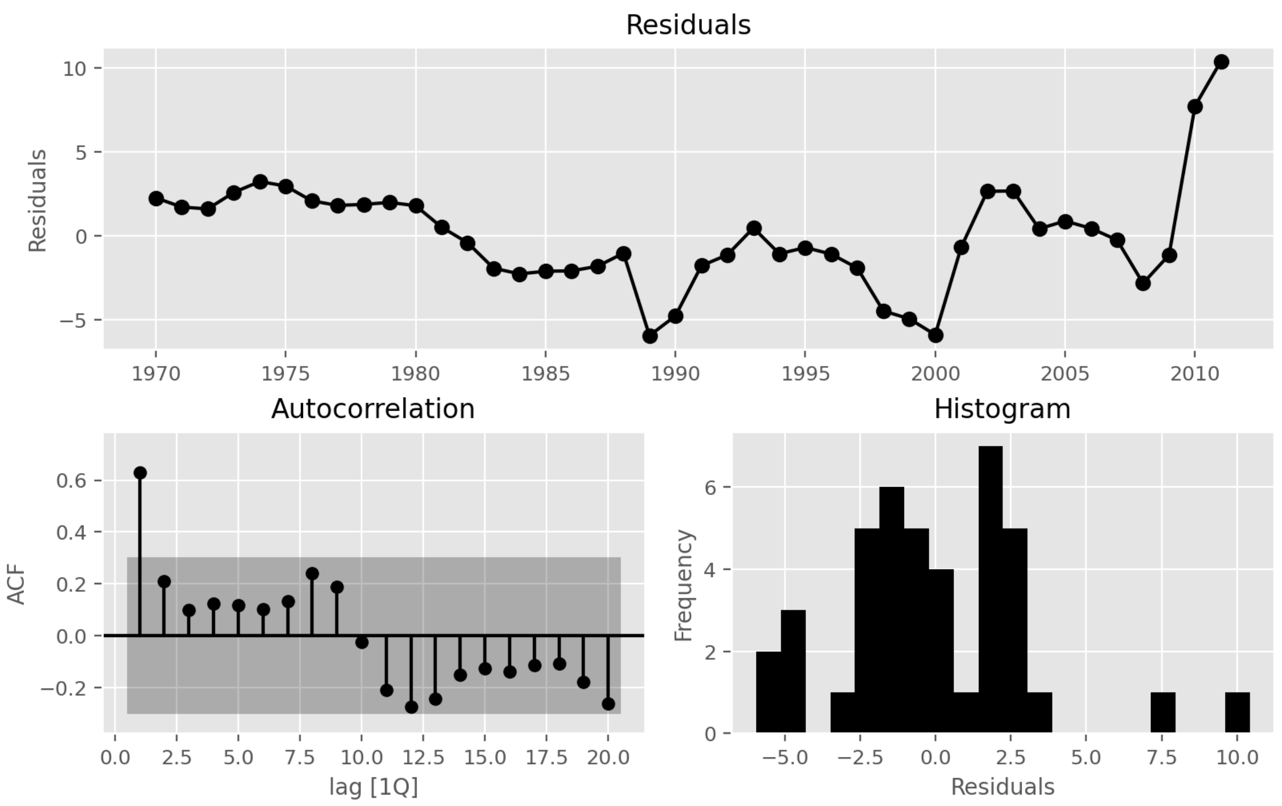

Figure 7.13: Residuals from a spurious regression.

7.4 Some useful predictors

There are several useful predictors that occur frequently when using regression for time series data.

Trend

It is common for time series data to be trending. A linear trend can be modelled by simply using x_{1,t}=t as a predictor, y_{t}= \beta_0+\beta_1t+\varepsilon_t, where t=1,\dots,T. A trend variable can be specified with the `trend` function from `utilsforecast`. In Section 7.7 we discuss how we can also model nonlinear trends.

Dummy variables

So far, we have assumed that each predictor takes numerical values. But what about when a predictor is a categorical variable taking only two values (e.g., “yes” and “no”)? Such a variable might arise, for example, when forecasting daily sales and you want to take account of whether the day is a **public holiday** or not. So the predictor takes value “yes” on a public holiday, and “no” otherwise.

This situation can still be handled within the framework of multiple regression models by creating a “dummy variable” which takes value 1 corresponding to “yes” and 0 corresponding to “no”. A dummy variable is also known as an “indicator variable”. This process can be referred to as “one-hot encoding”.

A dummy variable can also be used to account for an **outlier** in the data. Rather than omit the outlier, a dummy variable removes its effect. In this case, the dummy variable takes value 1 for that observation and 0 everywhere else. An example is the case where a special event has occurred. For example when forecasting tourist arrivals to Brazil, we will need to account for the effect of the Rio de Janeiro summer Olympics in 2016.

If there are more than two categories, then the variable can be coded using several dummy variables (one fewer than the total number of categories). We show below how to perform dummy variable encoding using the `get_dummies` function from `pandas`.

Seasonal dummy variables

Suppose that we are forecasting daily data and we want to account for the day of the week as a predictor. Then the following dummy variables can be created.

Notice that only six dummy variables are needed to code seven categories. That is because the seventh category (in this case Sunday) is captured by the intercept, and is specified when the dummy variables are all set to zero.

Many beginners will try to add a seventh dummy variable for the seventh category. This is known as the “dummy variable trap”, because it will cause the regression to fail. There will be one too many parameters to estimate when an intercept is also included. The general rule is to use one fewer dummy variables than categories. So for quarterly data, use three dummy variables; for monthly data, use 11 dummy variables; and for daily data, use six dummy variables, and so on. In `pandas`, we can enforce dropping one dummy variable by setting `drop_first=True`, as we show in the example further on.

The interpretation of each of the coefficients associated with the dummy variables is that it is *a measure of the effect of that category relative to the omitted category*. In the above example, the coefficient of $d_{1,t}$ associated with Monday will measure the effect of Monday on the forecast variable compared to the effect of Sunday. An example of interpreting estimated dummy variable coefficients capturing the quarterly seasonality of Australian beer production follows.

Example: Australian quarterly beer production

Recall the Australian quarterly beer production data shown again in Figure 7.14.

```python
# Read data
aus_production = pd.read_csv("../data/aus_production_formatted.csv",
  parse_dates=[1])
recent_production = aus_production.query(
    "unique_id == 'Beer' and ds >= '1992-01-01'"
).reset_index(drop=True)

plot_series(recent_production,
            xlabel="Quarter [1Q]",
            ylabel="Megalitres",
            title="Australian quarterly beer production")
```

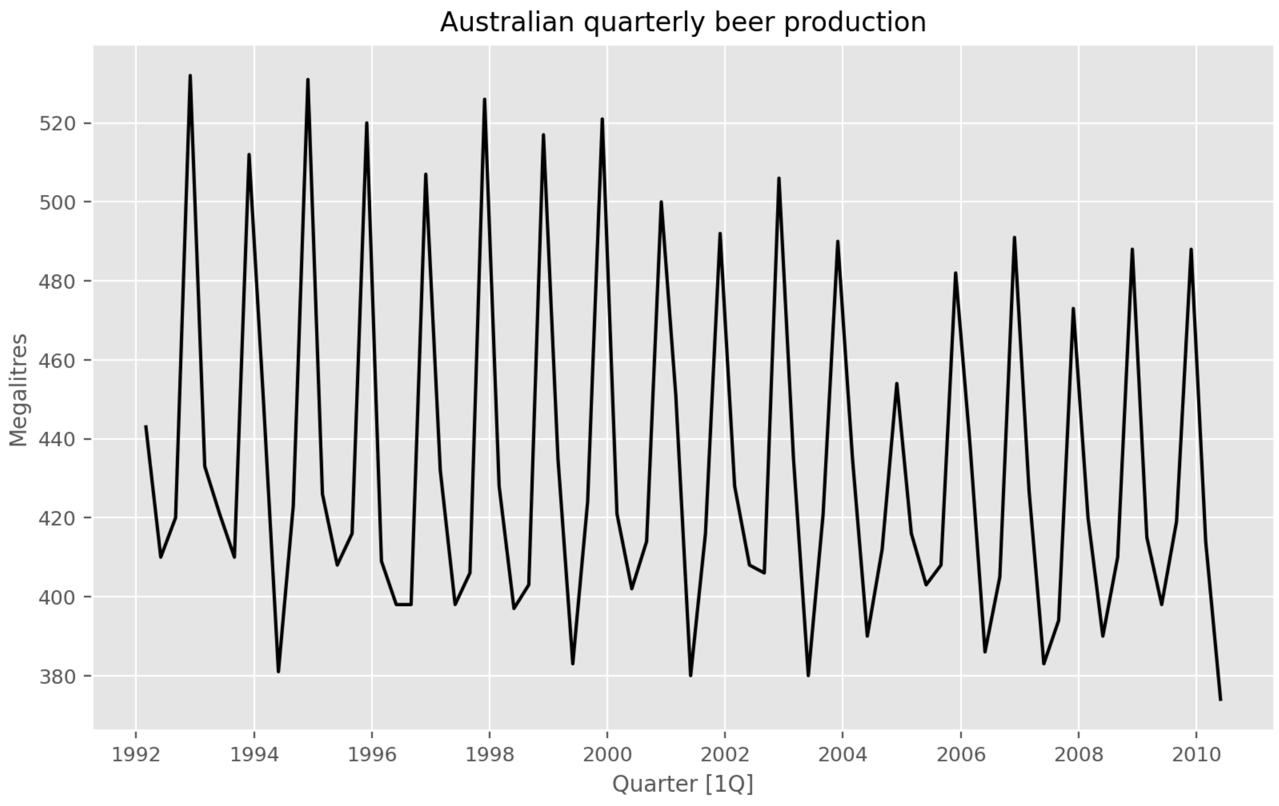

Figure 7.14: Australian quarterly beer production.

We want to forecast the value of future beer production. We can model this data using a regression model with a linear trend and quarterly dummy variables, $y_{t} = \beta_{0} + \beta_{1} t + \beta_{2}d_{2,t} + \beta_3 d_{3,t} + \beta_4 d_{4,t} + \varepsilon_{t}$, where $d_{i,t} = 1$ if t is in quarter i and 0 otherwise. The first quarter variable has been omitted, so the coefficients associated with the other quarters are measures of the difference between those quarters and the first quarter.

```python
from utilsforecast.feature_engineering import trend

fit_beer = recent_production.copy()
# Add trend
fit_beer, _ = trend(fit_beer, freq="QS-DEC")
# Add quarterly indicators
quarter = pd.get_dummies(fit_beer.ds.dt.quarter, prefix="Quarter",
  drop_first=True)
fit_beer = fit_beer.join(quarter)

mf = MLForecast(
    models=LinearRegression(),
    freq="QS-DEC",
)

mf.fit(fit_beer, fitted=True, static_features=[])

insample_forecasts = mf.forecast_fitted_values()["LinearRegression"]
residuals = mf.models_["LinearRegression"]._residuals

print_regression_summary_from_model(mf.models_['LinearRegression'])
```

```
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -42.9030 -7.5995 -0.4594  7.9908 21.7895

#> Coefficients:
#>                Estimate  Std. Error   t value  Pr(>|t|)
#>   (Intercept)  441.8002      3.7335    118.33         0 ***
#>         trend   -0.3403      0.0666     -5.11  2.73e-06 ***
#>     Quarter_2  -34.6598      3.9683     -8.73   9.1e-13 ***
#>     Quarter_3  -17.8216      4.0225     -4.43  3.45e-05 ***
#>     Quarter_4   72.7964      4.0230     18.09         0 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.229 on 69 degrees of freedom
Multiple R-squared: 0.924,   Adjusted R-squared: 0.920
F-statistic: inf on 0.0 and 69 DF, p-value: nan
```

There is an average downward trend of -0.34 megalitres per quarter. On average, the second quarter has production of 34.7 megalitres lower than the first quarter, the third quarter has production of 17.8 megalitres lower than the first quarter, and the fourth quarter has production of 72.8 megalitres higher than the first quarter.

```python
plot_series(
  df = fit_beer,
  forecasts_df = mf.forecast_fitted_values().drop(columns="y"),
  xlabel = "Quarter",
  ylabel = "Megalitres",
  title = "Australian quarterly beer production",
  rm_legend = False
)
```

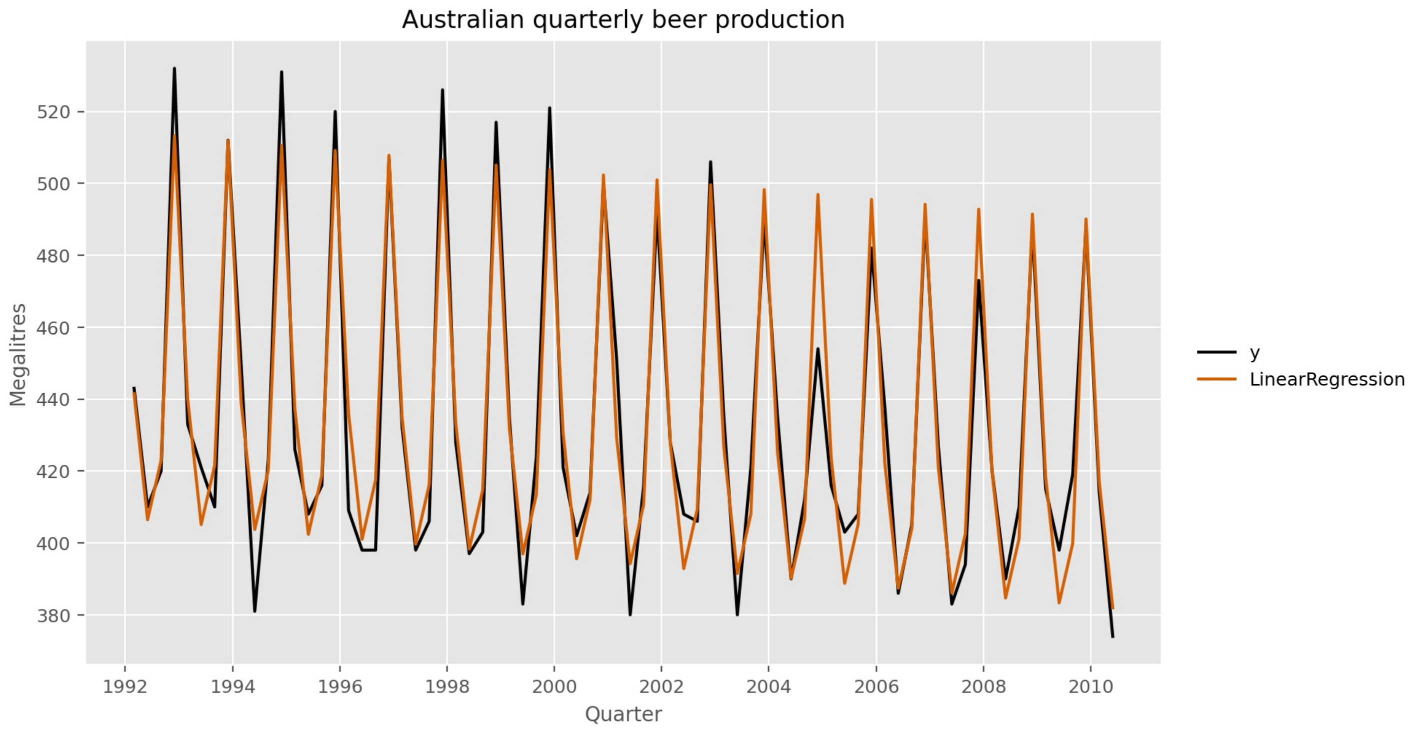

Figure 7.15: Time plot of beer production and predicted beer production.

```python
fig, ax = plt.subplots()
sc = ax.scatter(
    mf.forecast_fitted_values()["y"],
    insample_forecasts,
    label="Fitted",
    c=fit_beer["ds"].dt.quarter,
)
ax.axline((0, 0), slope=1)
ax.set_xlim(370, 550)
ax.set_ylim(370, 550)
ax.set_xlabel("Actual values")
ax.set_ylabel("Fitted")
ax.set_title("Australian quarterly beer production")
ax.legend(*sc.legend_elements())
plt.show()
```

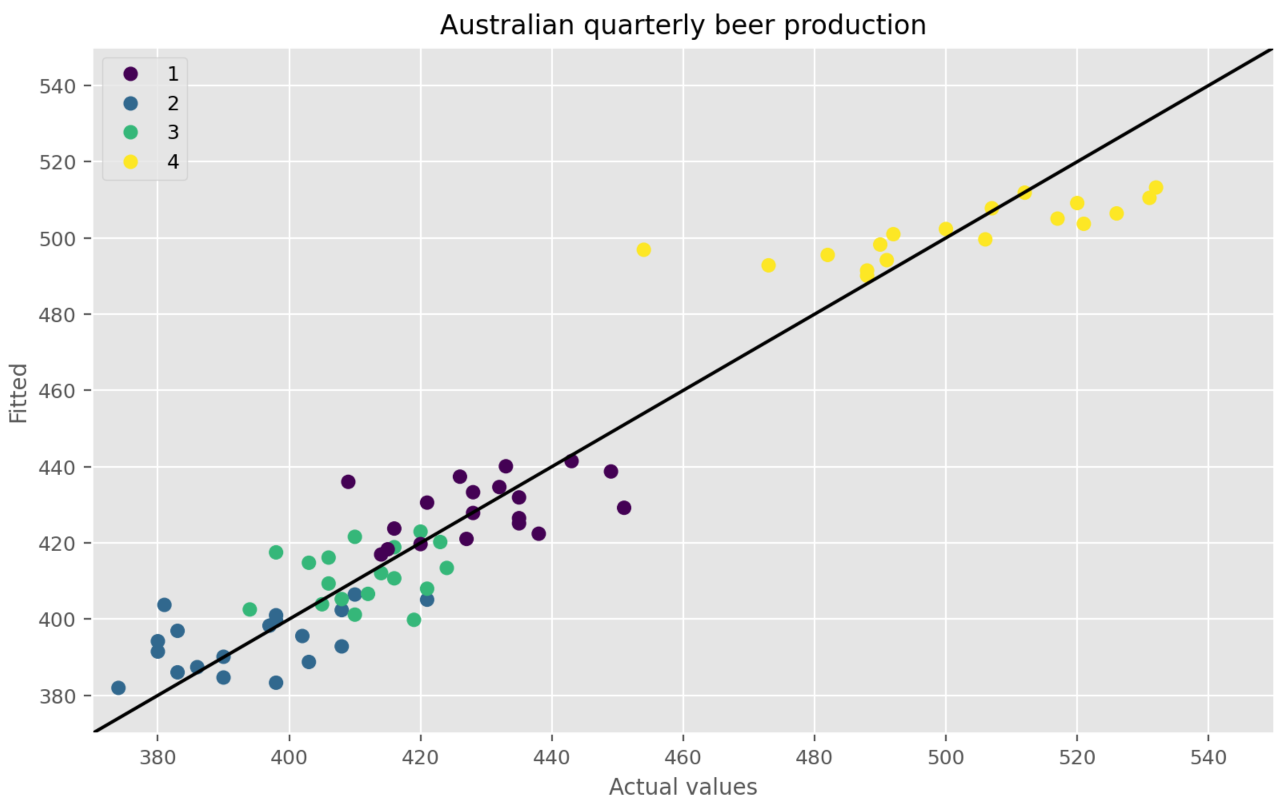

Figure 7.16: Actual beer production plotted against predicted beer production.

Intervention variables

It is often necessary to model interventions that may have affected the variable to be forecast. For example, competitor activity, advertising expenditure, industrial action, and so on, can all have an effect.

When the effect lasts only for one period, we use a “spike” variable. This is a dummy variable that takes value one in the period of the intervention and zero elsewhere. A spike variable is equivalent to a dummy variable for handling an outlier.

Other interventions have an immediate and permanent effect. If an intervention causes a level shift (i.e., the value of the series changes suddenly and permanently from the time of intervention), then we use a “step” variable. A step variable takes value zero before the intervention and one from the time of intervention onward.

Another form of permanent effect is a change of slope. Here the intervention is handled using a piecewise linear trend; a trend that bends at the time of intervention and hence is nonlinear. We will discuss this in Section 7.7.

Trading days

The number of trading days in a month can vary considerably and can have a substantial effect on sales data. To allow for this, the number of trading days in each month can be included as a predictor.

An alternative that allows for the effects of different days of the week has the following predictors: \begin{align*} x_{1} &= \text{number of Mondays in month;} \\ x_{2} &= \text{number of Tuesdays in month;} \\ & \vdots \\ x_{7} &= \text{number of Sundays in month.} \end{align*}

Distributed lags

It is often useful to include advertising expenditure as a predictor. However, since the effect of advertising can last beyond the actual campaign, we need to include lagged values of advertising expenditure. Thus, the following predictors may be used. \begin{align*} x_{1} &= \text{advertising for previous month;} \\ x_{2} &= \text{advertising for two months previously;} \\ & \vdots \\ x_{m} &= \text{advertising for m months previously.} \end{align*}

It is common to require the coefficients to decrease as the lag increases, although this is beyond the scope of this book.

Easter

Easter differs from most holidays because it is not held on the same date each year, and its effect can last for several days. In this case, a dummy variable can be used with value one where the holiday falls in the particular time period and zero otherwise.

With monthly data, if Easter falls in March then the dummy variable takes value 1 in March, and if it falls in April the dummy variable takes value 1 in April. When Easter starts in March and finishes in April, the dummy variable is split proportionally between months.

Fourier series

An alternative to using seasonal dummy variables, especially for long seasonal periods, is to use Fourier terms. Jean-Baptiste Fourier was a French mathematician, born in the 1700s, who showed that a series of sine and cosine terms of the right frequencies can approximate any periodic function. We can use them for seasonal patterns.

If m is the seasonal period, then the first few Fourier terms are given by x_{1,t} = \sin\left(\textstyle\frac{2\pi t}{m}\right), x_{2,t} = \cos\left(\textstyle\frac{2\pi t}{m}\right), x_{3,t} = \sin\left(\textstyle\frac{4\pi t}{m}\right), x_{4,t} = \cos\left(\textstyle\frac{4\pi t}{m}\right), x_{5,t} = \sin\left(\textstyle\frac{6\pi t}{m}\right), x_{6,t} = \cos\left(\textstyle\frac{6\pi t}{m}\right), and so on. If we have monthly seasonality, and we use the first 11 of these predictor variables, then we will get exactly the same forecasts as using 11 dummy variables.

With Fourier terms, we often need fewer predictors than with dummy variables, especially when m is large. This makes them useful for weekly data, for example, where m\approx 52. For short seasonal periods (e.g., quarterly data), there is little advantage in using Fourier terms over seasonal dummy variables.

These Fourier terms are produced using the `fourier()` function. For example, the Australian beer data can be modelled like this.

```python
fourier_beer = recent_production.copy()
# Add trend
fourier_beer, _ = trend(fourier_beer, freq="QS-DEC")
# Add fourier
fourier_beer, _ = fourier(fourier_beer, freq="QS-DEC", k=2, season_length=4)

mf = MLForecast(
    models=LinearRegression(),
    freq="QS-DEC",
)

mf.fit(fourier_beer, fitted=True, static_features=[])

insample_forecasts = mf.forecast_fitted_values()["LinearRegression"]
residuals = mf.models_["LinearRegression"]._residuals

print_regression_summary_from_model(mf.models_['LinearRegression'])
```

```
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -42.9030 -7.5996 -0.4594  7.9908 21.7894

#> Coefficients:
#>                Estimate  Std. Error   t value  Pr(>|t|)
#>   (Intercept)  446.8792      2.8955    154.33         0 ***
#>         trend   -0.3403      0.0671     -5.07  3.25e-06 ***
#>        sin1_4    8.9109      2.0371      4.37  4.29e-05 ***
#>        sin2_4    0.0001 389943.4328      0.00         1
#>        cos1_4   53.7281      2.0264     26.51         0 ***
#>        cos2_4   13.9896      1.9503      7.17  6.91e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.319 on 68 degrees of freedom
Multiple R-squared: 0.924,   Adjusted R-squared: 0.919
F-statistic: 1201061746667.8 on 6.914202543839565e-10 and 68 DF, p-value: 1.11e-16
```

The `K` argument to `fourier()` specifies how many pairs of sin and cos terms to include. The maximum allowed is K=m/2 where m is the seasonal period. Because we have used the maximum here, the results are identical to those obtained when using seasonal dummy variables.

If only the first two Fourier terms are used (x_{1,t} and x_{2,t}), the seasonal pattern will follow a simple sine wave. A regression model containing Fourier terms is often called a **harmonic regression** because the successive Fourier terms represent harmonics of the first two Fourier terms.

7.5 Selecting predictors

When there are many possible predictors, we need some strategy for selecting the best predictors to use in a regression model.

A common approach that is *not recommended* is to plot the forecast variable against a particular predictor and if there is no noticeable relationship, drop that predictor from the model. This is invalid because it is not always possible to see the relationship from a scatterplot, especially when the effects of other predictors have not been accounted for.

Another common approach which is also invalid is to do a multiple linear regression on all the predictors and disregard all variables whose p-values are greater than 0.05. To start with, statistical significance does not always indicate predictive value. Even if forecasting is not the goal, this is not a good strategy because the p-values can be misleading when two or more predictors are correlated with each other (see Section 7.8).

Instead, we will use a measure of predictive accuracy. Five such measures are introduced in this section and here applied to the model for US consumption:

```
# US Change example
us_change = pd.read_csv("../data/US_change.csv", parse_dates=[0])
mf = MLForecast(
    models=LinearRegression(),
    freq="QS-OCT",
)

mf.fit(us_change, fitted=True, static_features=[])
insample_forecasts = mf.forecast_fitted_values()["LinearRegression"]
insample_y = mf.forecast_fitted_values()["y"]
T = len(insample_y)
k = len(mf.models_["LinearRegression"].coef_)

print(f"Adjusted R^2 score: {adj_r2_score(insample_y, insample_forecasts, T, k):.3f}")
print(f"CV score          : {cv_score(mf, us_change, model='LinearRegression', target_col='y', n_w

print(f"AIC score         : {aic_score(insample_y, insample_forecasts, T, k):.3f}")
print(f"AICc score        : {aicc_score(insample_y, insample_forecasts, T, k):.3f}")
print(f"BIC score         : {bic_score(insample_y, insample_forecasts, T, k):.3f}")
```

```
Adjusted R^2 score: 0.763
CV score          : 0.249
AIC score         : -456.580
AICc score        : -456.140
BIC score         : -436.850
```

We compare these values against the corresponding values from other models. For the CV, AIC, AICc and BIC measures, we want to find the model with the lowest value; for Adjusted R^2, we seek the model with the highest value.

Adjusted R^2

Computer output for a regression will always give the R^2 value, discussed in Section 7.2. However, it is not a good measure of the predictive ability of a model. It measures how well the model fits the historical data, but not how well the model will forecast future data.

In addition, R^2 does not allow for “degrees of freedom”. Adding *any* variable tends to increase the value of R^2, even if that variable is irrelevant. For these reasons, forecasters should not use R^2 to determine whether a model will give good predictions, as it will lead to overfitting.

An equivalent idea is to select the model which gives the minimum sum of squared errors (SSE), given by \text{SSE} = \sum_{t=1}^T e_{t}^2.

Minimising the SSE is equivalent to maximising R^2 and will always choose the model with the most variables, and so is not a valid way of selecting predictors.

An alternative which is designed to overcome these problems is the adjusted R^2 (also called “R-bar-squared”): \bar{R}^2 = 1-(1-R^2)\frac{T-1}{T-k-1}, where T is the number of observations and k is the number of predictors. This is an improvement on R^2, as it will no longer increase with each added predictor. Using this measure, the best model will be the one with the largest value of

$\bar{R}^2$. Maximising $\bar{R}^2$ is equivalent to minimising the standard error $\hat{\sigma}_e$ given in Equation 7.3.

Maximising $\bar{R}^2$ works quite well as a method of selecting predictors, although it does tend to err on the side of selecting too many predictors.

Cross-validation

Time series cross-validation was introduced in Section 5.8 as a general tool for determining the predictive ability of a model. For regression models, it is also possible to use classical leave-one-out cross-validation to select predictors (Bergmeir, Hyndman, and Koo 2018). This is faster and makes more efficient use of the data. The procedure uses the following steps:

1. Remove observation t from the data set, and fit the model using the remaining data. Then compute the error ($e_{t}^*=y_{t}-\hat{y}_{t}$) for the omitted observation. (This is not the same as the residual because the tth observation was not used in estimating the value of $\hat{y}_{t}$.)
2. Repeat step 1 for $t=1,\dots,T$.
3. Compute the MSE from $e_{1}^*,\dots,e_{T}^*$. We shall call this the **CV**.

Although this looks like a time-consuming procedure, there are fast methods of calculating CV, so that it takes no longer than fitting one model to the full data set. The equation for computing CV efficiently is given in Section 7.9. Under this criterion, the best model is the one with the smallest value of CV.

Akaike's Information Criterion

A closely-related method is Akaike's Information Criterion, which we define as $\text{AIC} = T\log\left(\frac{\text{SSE}}{T}\right) + 2(k+2)$, where T is the number of observations used for estimation and k is the number of predictors in the model. Different computer packages use slightly different definitions for the AIC, although they should all lead to the same model being selected. The k+2 part of the equation occurs because there are k+2 parameters in the model: the k coefficients for the predictors, the intercept and the variance of the residuals. The idea here is to penalise the fit of the model (SSE) with the number of parameters that need to be estimated.

The model with the minimum value of the AIC is often the best model for forecasting. For large values of T, minimising the AIC is equivalent to minimising the CV value.

Corrected Akaike's Information Criterion

For small values of T, the AIC tends to select too many predictors, and so a bias-corrected version of the AIC has been developed, $\text{AIC}_{\text{c}} = \text{AIC} + \frac{2(k+2)(k+3)}{T-k-3}$. As with the AIC, the AICc should be minimised.

Schwarz's Bayesian Information Criterion

A related measure is Schwarz's Bayesian Information Criterion (usually abbreviated to BIC, SBIC or SC): $\text{BIC} = T\log\left(\frac{\text{SSE}}{T}\right) + (k+2)\log(T)$. As with the AIC, minimising the BIC is intended to give the best model. The model chosen by the BIC is either the same as that chosen by the AIC, or one with fewer terms. This is because the BIC penalises the number of parameters more heavily than the AIC. For large values of T, minimising BIC is similar to leave-v-out cross-validation when $v = T[1-1/(\log(T)-1)]$.

Which measure should we use?

While $\bar{R}^2$ is widely used, and has been around longer than the other measures, its tendency to select too many predictor variables makes it less suitable for forecasting.

Many statisticians like to use the BIC because it has the feature that if there is a true underlying model, the BIC will select that model given enough data. However, in reality, there is rarely, if ever, a true underlying model, and even if there was a true underlying model, selecting that model will not necessarily give the best forecasts (because the parameter estimates may not be accurate).

Consequently, we recommend that one of the AICc, AIC, or CV statistics be used, each of which has forecasting as their objective. If the value of T is large enough, they will all lead to the same model. In most of the examples in this book, we use the AICc value to select the forecasting model.

Example: US consumption

In the multiple regression example for forecasting US consumption we considered four predictors. With four predictors, there are $2^4=16$ possible models. Now we can check if all four predictors are actually useful, or whether we can drop one or more of them. All 16 models were fitted and the results are summarised in Table 7.1. The column `predictor_variables` indicates which predictor variables were included in the model. Hence the first row shows the measures of predictive accuracy for a model including all four predictors.

The results have been sorted according to the AICc. Therefore the best models are given at the top of the table, and the worst at the bottom of the table.

Table 7.1: All 16 possible models for forecasting US consumption with 4 predictors.

	predictor_variables	adjr2_score	cv_score	aic_score	aicc_score	bic_score
0	[Income, Production, Savings, Unemployment]	0.76	0.25	-456.58	-456.14	-436.85
1	[Income, Production, Savings]	0.76	0.23	-455.18	-454.87	-438.74
2	[Income, Savings, Unemployment]	0.76	0.27	-454.36	-454.05	-437.92
3	[Income, Savings]	0.73	0.19	-435.72	-435.51	-422.56
4	[Income, Production, Unemployment]	0.37	1.20	-262.27	-261.96	-245.83
5	[Production, Savings, Unemployment]	0.35	0.60	-257.14	-256.83	-240.70
6	[Income, Unemployment]	0.34	1.40	-256.85	-256.64	-243.70
7	[Income, Production]	0.34	1.15	-254.16	-253.95	-241.01
8	[Production, Savings]	0.32	0.66	-250.68	-250.47	-237.53
9	[Savings, Unemployment]	0.31	0.74	-246.89	-246.68	-233.74
10	[Production, Unemployment]	0.31	0.92	-246.07	-245.87	-232.92
11	[Production]	0.28	0.95	-238.10	-237.98	-228.24
12	[Unemployment]	0.27	1.09	-237.39	-237.26	-227.52
13	[Income]	0.14	1.28	-204.59	-204.47	-194.73
14	[Savings]	0.06	0.56	-186.55	-186.42	-176.68
15	[]	0.00	0.90	-175.06	-175.00	-168.48

The best model contains all four predictors. However, a closer look at the results reveals some interesting features. There is clear separation between the models in the first four rows and the ones below. This indicates that `Income` and `Savings` are both more important variables than `Production` and `Unemployment`. Also, the first three rows have almost identical values of CV, AIC and AICc. So we could possibly drop either the `Production` variable, or the `Unemployment` variable, and get similar forecasts. Note that `Production` and `Unemployment` are highly (negatively) correlated, as shown in Figure 7.5, so most of the predictive information in `Production` is also contained in the `Unemployment` variable.

Best subset regression

Where possible, all potential regression models should be fitted (as was done in the example above) and the best model should be selected based on one of the measures discussed. This is known as "best subsets" regression or "all possible subsets" regression.

Stepwise regression

If there are a large number of predictors, it is not possible to fit all possible models. For example, 40 predictors leads to 2^{40} > 1 trillion possible models! Consequently, a strategy is required to limit the number of models to be explored.

An approach that works quite well is *backwards stepwise regression*:

- Start with the model containing all potential predictors.
- Remove one predictor at a time. Keep the model if it improves the measure of predictive accuracy.
- Iterate until no further improvement.

If the number of potential predictors is too large, then the backwards stepwise regression will not work and *forward stepwise regression* can be used instead. This procedure starts with a model that includes only the intercept. Predictors are added one at a time, and the one that most improves the measure of predictive accuracy is retained in the model. The procedure is repeated until no further improvement can be achieved.

Alternatively for either the backward or forward direction, a starting model can be one that includes a subset of potential predictors. In this case, an extra step needs to be included. For the backwards procedure we should also consider adding a predictor with each step, and for the forward procedure we should also consider dropping a predictor with each step. These are referred to as *hybrid* procedures.

It is important to realise that any stepwise approach is not guaranteed to lead to the best possible model, but it almost always leads to a good model. For further details see James et al. (2014).

Beware of inference after selecting predictors

We do not discuss statistical inference of the predictors in this book (e.g., looking at p-values associated with each predictor). If you do wish to look at the statistical significance of the predictors, beware that *any* procedure involving selecting predictors first will invalidate the assumptions behind the p-values. The procedures we recommend for selecting predictors are helpful when the model is used for forecasting; they are not helpful if you wish to study the effect of any predictor on the forecast variable.

7.6 Forecasting with regression

Recall that predictions of y can be obtained using $\hat{y_t} = \hat\beta_{0} + \hat\beta_{1} x_{1,t} + \hat\beta_{2} x_{2,t} + \cdots + \hat\beta_{k} x_{k,t}$, which comprises the estimated coefficients and ignores the error in the regression equation. Plugging in the values of the predictor variables $x_{1,t},\dots,x_{k,t}$ for $t=1,\dots,T$ returns the fitted (training set) values of y. What we are interested in here, however, is forecasting *future* values of y.

Ex-ante versus ex-post forecasts

When using regression models for time series data, we need to distinguish between the different types of forecasts that can be produced, depending on what is assumed to be known when the forecasts are computed.

Ex-ante forecasts are those that are made using only the information that is available in advance. For example, ex-ante forecasts for the percentage change in US consumption for quarters following the end of the sample, should only use information that was available *up to and including* 2019 Q2. These are genuine forecasts, made in advance using whatever information is available at the time. Therefore in order to generate ex-ante forecasts, the model requires forecasts of the predictors. To obtain these we can use one of the simple methods introduced in Section 5.2 or more sophisticated pure time series approaches that follow in Chapters 8 and 9. Alternatively, forecasts from some other source, such as a government agency, may be available and can be used.

Ex-post forecasts are those that are made using later information on the predictors. For example, ex-post forecasts of consumption may use the actual observations of the predictors, once these have been observed. These are not genuine forecasts, but are useful for studying the behaviour of forecasting models.

The model from which ex-post forecasts are produced should not be estimated using data from the forecast period. That is, ex-post forecasts can assume knowledge of the predictor variables (the x variables), but should not assume knowledge of the data that are to be forecast (the y variable).

A comparative evaluation of ex-ante forecasts and ex-post forecasts can help to separate out the sources of forecast uncertainty. This will show whether forecast errors have arisen due to poor forecasts of the predictor or due to a poor forecasting model.

Example: Australian quarterly beer production

Normally, we cannot use actual future values of the predictor variables when producing ex-ante forecasts because their values will not be known in advance. However, the special predictors introduced in Section 7.4 are all known in advance, as they are based on calendar variables (e.g., seasonal dummy variables or public holiday indicators) or deterministic functions of time (e.g. time trend). In such cases, there is no difference between ex-ante and ex-post forecasts.

```
fit_beer = recent_production.copy()

features = [
    trend,
    partial(fourier, season_length=4, k=1),
]

fit_beer_with_predictors, future_predictors = pipeline(
    fit_beer,
    features=features,
    freq="QS-DEC",
    h=8,
)

# Create and fit model
mf = MLForecast(
    models=LinearRegression(),
    freq="QS-DEC",
)
mf.fit(
    fit_beer_with_predictors,
    static_features=[],
    prediction_intervals=PredictionIntervals(n_windows=4, h=8),
)

forecasts = mf.predict(h=8, X_df=future_predictors, level=[80, 95])

plot_series(df=fit_beer, forecasts_df=forecasts, level=[80, 95],
            xlabel="Quarter",
            ylabel="megalitres",
            title="Forecasts of beer production using regression",
            rm_legend=False,)
```

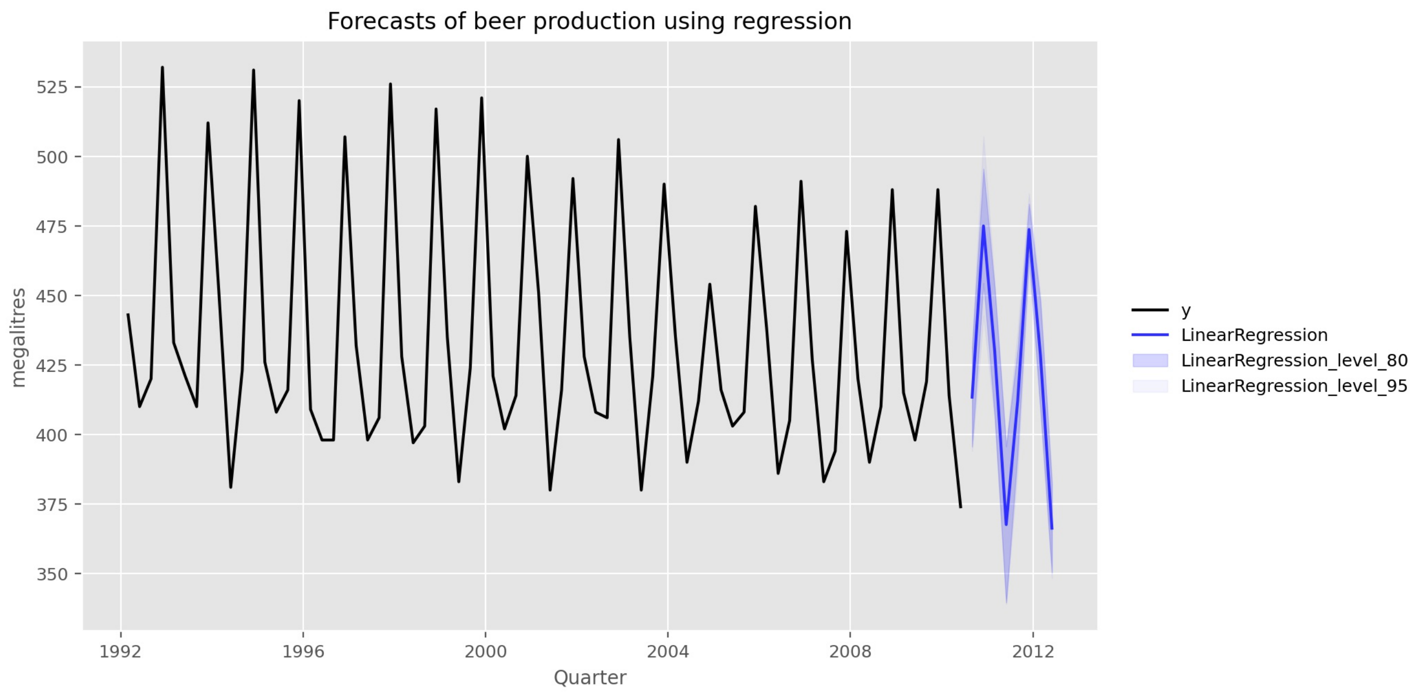

Figure 7.17: Forecasts from the regression model for beer production. The dark shaded region shows 80% prediction intervals and the light shaded region shows 95% prediction intervals.

Scenario based forecasting

In this setting, the forecaster assumes possible scenarios for the predictor variables that are of interest. For example, a US policy maker may be interested in comparing the predicted change in consumption when there is a constant growth of 1% and 0.5%

respectively for income and savings with no change in the employment rate, versus a respective decline of 1% and 0.5%, for each of the four quarters following the end of the sample. The resulting forecasts are calculated below and shown in Figure 7.18. We should note that prediction intervals for scenario based forecasts do not include the uncertainty associated with the future values of the predictor variables. They assume that the values of the predictors are known in advance.

```python
# Fit a model using the exogenous regressors Income + Savings + Unemployment
us_change_scenarios = us_change[
    ["ds", "unique_id", "y", "Income", "Savings", "Unemployment"]
].copy()
# Fit model
mf = MLForecast(
    models=LinearRegression(),
    freq="QS-OCT",
)
mf.fit(
    us_change_scenarios,
    static_features=[],
)

# Generate futr_df
futr_df = mf.make_future_dataframe(h=4)
```

```python
# Scenario 1 - Increase
futr_df["Income"] = 1
futr_df["Savings"] = 0.5
futr_df["Unemployment"] = 0

# Create forecasts
forecasts_scenario1 = mf.predict(h=4, X_df=futr_df)
forecasts_scenario1 = \
  mf.models_["LinearRegression"].add_prediction_intervals(forecasts_scenario1, futr_df)
forecasts_scenario1.columns = \
  forecasts_scenario1.columns.str.replace("LinearRegression", "increase")

# Scenario 2 - Decrease
futr_df["Income"] = -1
futr_df["Savings"] = -0.5
futr_df["Unemployment"] = 0

# Create forecasts
forecasts_scenario2 = mf.predict(h=4, X_df=futr_df)
forecasts_scenario2 = \
  mf.models_["LinearRegression"].add_prediction_intervals(forecasts_scenario2, futr_df)
forecasts_scenario2.columns = \
  forecasts_scenario2.columns.str.replace("LinearRegression", "decrease")

# Plot forecasts
#fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(12,9))
plot_series(
    df=us_change_scenarios,
    forecasts_df=forecasts_scenario1.merge(forecasts_scenario2),
    level=[80, 95], rm_legend=False,
    xlabel="Quarter [1Q]",
    ylabel="% change",
    title="US Consumption",
)
```

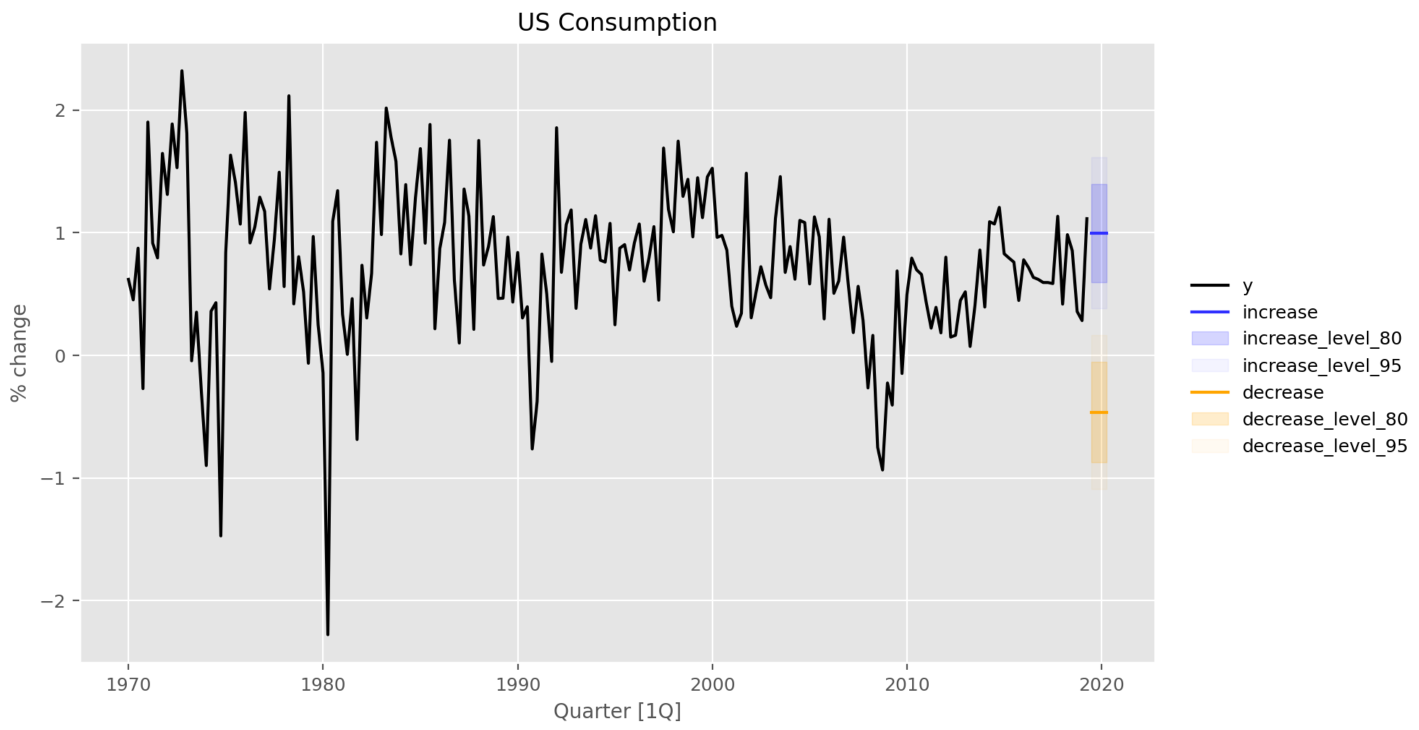

Figure 7.18: Forecasting percentage changes in personal consumption expenditure for the US under scenario based forecasting.

Building a predictive regression model

The great advantage of regression models is that they can be used to capture important relationships between the forecast variable of interest and the predictor variables. However, for ex ante forecasts, these models require future values of each predictor, which can be challenging. If forecasting each predictor is too difficult, we may use scenario-based forecasting instead, where we assume specific future values for all predictors.

An alternative formulation is to use as predictors their lagged values. Assuming that we are interested in generating a h-step ahead forecast we write $y_{t+h}=\beta_0+\beta_1x_{1,t}+\dots+\beta_kx_{k,t}+\varepsilon_{t+h}$ for $h=1,2\dots$. The predictor set is formed by values of the xs that are observed h time periods prior to observing y. Therefore when the estimated model is projected into the future, i.e., beyond the end of the sample T, all predictor values are available.

Including lagged values of the predictors does not only make the model operational for easily generating forecasts, it also makes it intuitively appealing. For example, the effect of a policy change with the aim of increasing production may not have an instantaneous effect on consumption expenditure. It is most likely that this will happen with a lagging effect. We touched upon this in Section 7.4 when briefly introducing distributed lags as predictors. Several directions for generalising regression models to better incorporate the rich dynamics observed in time series are discussed in Chapter 10.

Prediction intervals

With each forecast for the change in consumption in Figure 7.2, 95% and 80% prediction intervals are also included. The general formulation of how to calculate prediction intervals for multiple regression models is presented in Section 7.9. As this involves some advanced matrix algebra we present here the case for calculating prediction intervals for a simple regression, where a forecast can be generated using the equation, $\hat{y}=\hat{\beta}_0+\hat{\beta}_1x$. Assuming that the regression errors are normally distributed, an approximate 95% prediction interval associated with this forecast is given by $\hat{y} \pm 1.96 \hat{\sigma}_e\sqrt{1+\frac{1}{T}+\frac{(x-\bar{x})^2}{(T-1)s_x^2}}, \tag{7.4}$ where T is the total number of observations, $\bar{x}$ is the mean of the observed x values, s_x is the standard deviation of the observed x values and $\hat{\sigma}_e$ is the standard error of the regression given by Equation 7.3. Similarly, an 80% prediction interval can be obtained by replacing 1.96 by 1.28. Other prediction intervals can be obtained by replacing the 1.96 with the appropriate value given in Table 5.1.

Equation 7.4 shows that the prediction interval is wider when x is far from $\bar{x}$. That is, we are more certain about our forecasts when considering values of the predictor variable close to its sample mean.

Example

The estimated simple regression line in the US consumption example is $\hat{y}_t= 0.54 + 0.27x_t$.

```
# Scenario 1
futr_df["Income"] = us_change["Income"].mean()
# Create forecasts
forecasts_scenario1 = mf.predict(h=4, X_df=futr_df)
forecasts_scenario1 = \
  mf.models_["LinearRegression"].add_prediction_intervals(forecasts_scenario1, futr_df)
forecasts_scenario1.columns = \
  forecasts_scenario1.columns.str.replace("LinearRegression", "average_increase")
# Scenario 2
futr_df["Income"] = 12
# Create forecasts
forecasts_scenario2 = mf.predict(h=4, X_df=futr_df)
forecasts_scenario2 = \
  mf.models_["LinearRegression"].add_prediction_intervals(forecasts_scenario2, futr_df)
forecasts_scenario2.columns = \
  forecasts_scenario2.columns.str.replace("LinearRegression", "extreme_increase")
# Plot forecasts scenario 1
plot_series(
  df = us_change_scenarios,
  forecasts_df = forecasts_scenario1.merge(forecasts_scenario2),
  level = levels,
  xlabel = "Quarter [1Q]",
  ylabel = "% change",
  title = "US consumption",
            rm_legend=False)
```

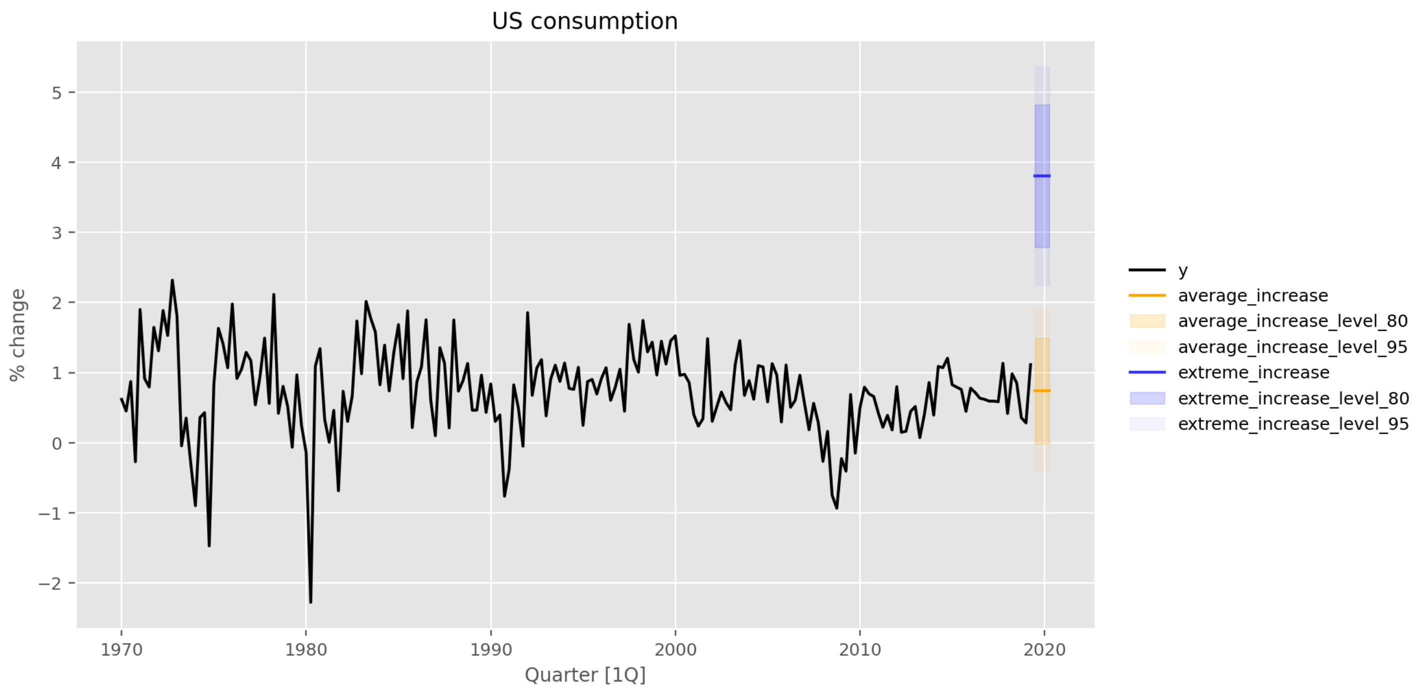

Figure 7.19: Prediction intervals if income is increased by its historical mean of 0.73% versus an extreme increase of 12%.

Assuming that for the next four quarters, personal income will increase by its historical mean value of \bar{x}=0.73%, consumption is forecast to increase by 0.74% and the corresponding 80% and 95% prediction intervals are [-0.02, 1.5] and [-0.42, 1.9] respectively (calculated using python). If we assume an extreme increase of 12% in income, then the prediction intervals are considerably wider as shown in Figure 7.19.

7.7 Nonlinear regression

Although the linear relationship assumed so far in this chapter is often adequate, there are many cases in which a nonlinear functional form is more suitable. To keep things simple in this section we assume that we only have one predictor x.

The simplest way of modelling a nonlinear relationship is to transform the forecast variable y and/or the predictor variable x before estimating a regression model. While this provides a non-linear functional form, the model is still linear in the parameters. The most commonly used transformation is the (natural) logarithm (see Section 3.1).

A **log-log** functional form is specified as \log y=\beta_0+\beta_1 \log x +\varepsilon. In this model, the slope \beta_1 can be interpreted as an elasticity: \beta_1 is the average percentage change in y resulting from a 1% increase in x. Other useful forms can also be specified. The **log-linear** form is specified by only transforming the forecast variable and the **linear-log** form is obtained by transforming the predictor.

Recall that in order to perform a logarithmic transformation to a variable, all of its observed values must be greater than zero. In the case that variable x contains zeros, we use the transformation \log(x+1); i.e., we add one to the value of the variable and then take logarithms. This has a similar effect to taking logarithms but avoids the problem of zeros. It also has the neat side-effect of zeros on the original scale remaining zeros on the transformed scale.

There are cases for which simply transforming the data will not be adequate and a more general specification may be required. Then the model we use is y=f(x) +\varepsilon where f is a nonlinear function. In standard (linear) regression, f(x)=\beta_{0} + \beta_{1} x. In the specification of nonlinear regression that follows, we allow f to be a more flexible nonlinear function of x, compared to simply a logarithmic or other transformation.

One of the simplest specifications is to make f **piecewise linear**. That is, we introduce points where the slope of f can change. These points are called **knots**. This can be achieved by letting x_{1}=x and introducing variable x_{2} such that \begin{align*} x_{2} = (x-c)_+ &= \left\{ \begin{array}{ll} 0 & \text{if } x < c\\ x-c & \text{if } x \ge c. \end{array}\right. \end{align*} The notation (x-c)_+ means the value x-c if it is positive and 0 otherwise. This forces the slope to bend at point c. Additional bends can be included in the relationship by adding further variables of the above form.

Piecewise linear relationships constructed in this way are a special case of **regression splines**. In general, a linear regression spline is obtained using x_{1}= x \quad x_{2} = (x-c_{1})_+ \quad\dots\quad x_{k} = (x-c_{k-1})_+ where c_{1},\dots,c_{k-1} are the knots (the points at which the line can bend). Selecting the number of knots (k-1) and where they should be positioned can be difficult and somewhat arbitrary. Some automatic knot selection algorithms are available, but are not widely used.

Forecasting with a nonlinear trend

In Section 7.4 fitting a linear trend to a time series by setting x=t was introduced. The simplest way of fitting a nonlinear trend is using quadratic or higher order trends obtained by specifying [x_{1,t} =t,x_{2,t}=t^2,.] However, it is not recommended that quadratic or higher order trends be used in forecasting. When they are extrapolated, the resulting forecasts are often unrealistic.

A better approach is to use the piecewise specification introduced above and fit a piecewise linear trend which bends at some point in time. We can think of this as a nonlinear trend constructed of linear pieces. If the trend bends at time \tau, then it can be specified by simply replacing x=t and c=\tau above such that we include the predictors, \begin{align*} x_{1,t} & = t \\ x_{2,t} &= (t-\tau)_+ = \left\{ \begin{array}{ll} 0 & \text{if } t < \tau\\ t-\tau & \text{if } t \ge \tau \end{array}\right. \end{align*} in the model. If the associated coefficients of x_{1,t} and x_{2,t} are \beta_1 and \beta_2, then \beta_1 gives the slope of the trend before time \tau, while the slope of the line after time \tau is given by \beta_1+\beta_2. Additional bends can be included in the relationship by adding further variables of the form (t-\tau)_+ where \tau is the “knot” or point in time at which the line should bend.

Example: Boston marathon winning times

We will fit some trend models to the Boston marathon winning times for men. First we extract the men’s data and convert the winning times to a numerical value. The course was lengthened (from 24.5 miles to 26.2 miles) in 1924, which led to a jump in the winning times, so we only consider data from that date onwards.

```
boston_marathon = pd.read_csv(
    "../data/boston_marathon.csv", index_col=[0], parse_dates=[2]
)
boston_men = boston_marathon.query(
    """ds >= 1924 and unique_id == "Men's open division" """
).reset_index(drop=True)
boston_men["y"] = boston_men["Time"] / 60
boston_men = boston_men[["ds", "y", "unique_id"]]
```

The top panel of Figure 7.20 shows the winning times since 1924. The time series shows a general downward trend as the winning times have been improving over the years. The bottom panel shows the residuals from fitting a linear trend to the data. The plot shows an obvious nonlinear pattern which has not been captured by the linear trend.

```python
# Create model
mf_linear = MLForecast(
    models=LinearRegression(),
    freq="YS",
)
# Add trend
boston_men, futr_df = trend(boston_men, freq="YS", h=10)

# Fit model
mf_linear.fit(
    boston_men,
    static_features=[],
    fitted=True,
    prediction_intervals=PredictionIntervals(n_windows=2, h=4),
)

insample_forecasts = mf_linear.forecast_fitted_values()["LinearRegression"]
slope = mf_linear.models_["LinearRegression"].coef_[0]
intercept = mf_linear.models_["LinearRegression"].intercept_
residuals = mf_linear.forecast_fitted_values()["y"] - insample_forecasts

# Plot
fig, ax = plt.subplots(nrows=2)

ax[0].plot(boston_men["ds"], boston_men["y"])
ax[0].plot(boston_men["ds"], insample_forecasts)
ax[0].set_title("Boston marathon winning times")
#ax[0].set_xlabel("Year")
ax[0].set_ylabel("Minutes")

ax[1].plot(boston_men["ds"], residuals)
ax[1].set_title("Residuals from a linear trend")
ax[1].set_xlabel("Year")
ax[1].set_ylabel("Minutes")
plt.show()
```

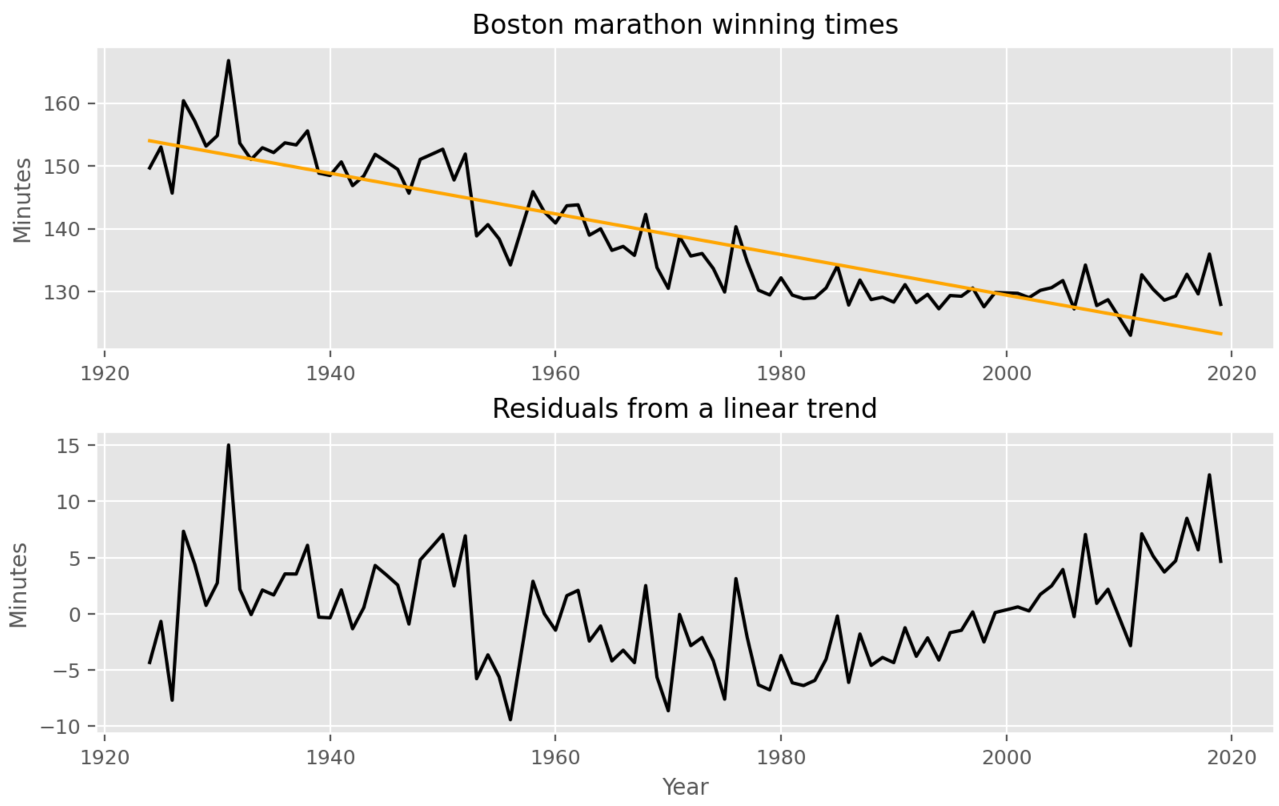

Figure 7.20: Fitting a linear trend to the Boston marathon winning times is inadequate.

Fitting an exponential trend (equivalent to a log-linear regression) to the data can be achieved by transforming the y variable so that the model to be fitted is, \log y_t=\beta_0+\beta_1 t +\varepsilon_t. The fitted exponential trend and forecasts are shown in Figure 7.21 and Figure 7.21. Although the exponential trend does not seem to fit the data much better than the linear trend, it perhaps gives a more sensible projection in that the winning times will decrease in the future but at a decaying rate rather than a fixed linear rate.

The plot of winning times reveals three different periods. There is a lot of volatility in the winning times up to about 1950, with the winning times barely declining. After 1950 there is a clear decrease in times, followed by a flattening out after the 1980s, with the suggestion of an upturn towards the end of the sample.

```python
from mlforecast.target_transforms import GlobalSklearnTransformer
from sklearn.preprocessing import FunctionTransformer
```

```python
# Create model
sk_log = FunctionTransformer(func=np.log, inverse_func=np.exp)

mf_log = MLForecast(
    models=LinearRegression(),
    freq="YS",
    target_transforms=[GlobalSklearnTransformer(sk_log)],
)
# Add trend and preprocess
boston_men, futr_df = trend(boston_men, freq="YS", h=10)
boston_men_log = mf_log.preprocess(boston_men, static_features=[])

# Fit model
mf_log.fit(
    boston_men_log,
    static_features=[],
    fitted=True,
    prediction_intervals=PredictionIntervals(n_windows=2, h=4),
)
levels = [95]
```

```
# Linear target
forecasts_linear = mf_linear.predict(h=4, X_df=futr_df, level=levels)
forecasts_linear.columns = \
  forecasts_linear.columns.str.replace("LinearRegression", "linear")

forecasts_log = mf_log.predict(h=4, X_df=futr_df, level=levels)
forecast_cols = \
  ["LinearRegression", "LinearRegression-lo-95", "LinearRegression-hi-95"]
forecasts_log[forecast_cols] = np.exp(forecasts_log[forecast_cols])
forecasts_log.columns = \
  forecasts_log.columns.str.replace("LinearRegression", "exponential")

plot_series(
  df = boston_men,
  forecasts_df = forecasts_linear.merge(forecasts_log),
  level = levels,
  rm_legend = False,
            xlabel="Year [1Y]",
            ylabel="Minutes",
            title="Boston marathon winning times"
           )
```

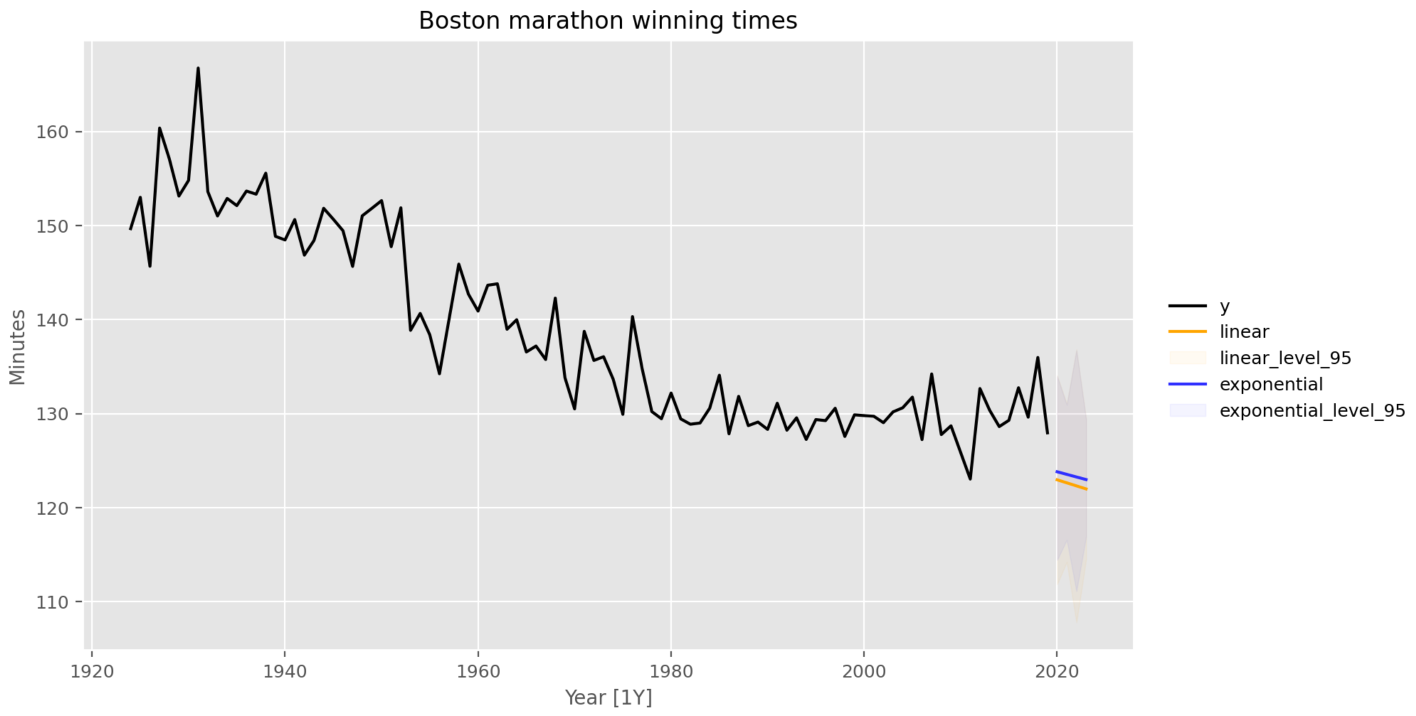

Figure 7.21: Projecting forecasts from linear trends for the Boston marathon winning times.

Figure 7.21 shows the fitted lines and forecasts from linear and exponential trends. The best forecasts appear to come from the exponential trend.

7.8 Correlation, causation and forecasting

Correlation is not causation

It is important not to confuse correlation with causation, or causation with forecasting. A variable x may be useful for forecasting a variable y, but that does not mean x is causing y. It is possible that x *is* causing y, but it may be that y is causing x, or that the relationship between them is more complicated than simple causality.

For example, it is possible to model the number of drownings at a beach resort each month with the number of ice-creams sold in the same period. The model can give reasonable forecasts, not because ice-creams cause drownings, but because people eat more ice-creams on hot days when they are also more likely to go swimming. So the two variables (ice-cream sales and drownings) are correlated, but one is not causing the other. They are both caused by a third variable (temperature). This is an example of “confounding” — where an omitted variable causes changes in both the response variable and at least one predictor variable.

We describe a variable that is not included in our forecasting model as a **confounder** when it influences both the response variable and at least one predictor variable. Confounding makes it difficult to determine what variables are *causing* changes in other variables, but it does not necessarily make forecasting more difficult.

Similarly, it is possible to forecast if it will rain in the afternoon by observing the number of cyclists on the road in the morning. When there are fewer cyclists than usual, it is more likely to rain later in the day. The model can give reasonable forecasts, not because cyclists prevent rain, but because people are more likely to cycle when the published weather forecast is for a dry day. In this case, there is a causal relationship, but in the opposite direction to our forecasting model. The number of cyclists falls because there is rain forecast. That is, y (rainfall) is affecting x (cyclists).

It is important to understand that correlations are useful for forecasting, even when there is no causal relationship between the two variables, or when the causality runs in the opposite direction to the model, or when there is confounding.

However, often a better model is possible if a causal mechanism can be determined. A better model for drownings will probably include temperatures and visitor numbers and exclude ice-cream sales. A good forecasting model for rainfall will not include cyclists, but it will include atmospheric observations from the previous few days.

Forecasting with correlated predictors

When two or more predictors are highly correlated it is always challenging to accurately separate their individual effects. Suppose we are forecasting monthly sales of a company for 2012, using data from 2000–2011. In January 2008, a new competitor came into the market and started taking some market share. At the same time, the economy began to decline. In your forecasting model, you include both competitor activity (measured using advertising time on a local television station) and the health of the economy (measured using GDP). It will not be possible to separate the effects of these two predictors because they are highly correlated.

Having correlated predictors is not really a problem for forecasting, as we can still compute forecasts without needing to separate out the effects of the predictors. However, it becomes a problem with scenario forecasting as the scenarios should take account of the relationships between predictors. It is also a problem if some historical analysis of the contributions of various predictors is required.

Multicollinearity and forecasting

A closely related issue is **multicollinearity**, which occurs when similar information is provided by two or more of the predictor variables in a multiple regression.

It can occur when two predictors are highly correlated with each other (that is, they have a correlation coefficient close to +1 or -1). In this case, knowing the value of one of the variables tells you a lot about the value of the other variable. Hence, they are providing similar information. For example, foot size can be used to predict height, but including the size of both left and right feet in the same model is not going to make the forecasts any better, although it won’t make them worse either.

Multicollinearity can also occur when a linear combination of predictors is highly correlated with another linear combination of predictors. In this case, knowing the value of the first group of predictors tells you a lot about the value of the second group of predictors. Hence, they are providing similar information.

An example of this problem is the dummy variable trap discussed in Section 7.4. Suppose you have quarterly data and use four dummy variables, d_1, d_2, d_3 and d_4. Then d_4=1-d_1-d_2-d_3, so there is perfect correlation between d_4 and d_1+d_2+d_3.

In the case of perfect correlation (i.e., a correlation of +1 or -1, such as in the dummy variable trap), it is not possible to estimate the regression model.

If there is high correlation (close to but not equal to +1 or -1), then the estimation of the regression coefficients is computationally difficult. In fact, some software (notably Microsoft Excel) may give highly inaccurate estimates of the coefficients. Most reputable statistical software will use algorithms to limit the effect of multicollinearity on the coefficient estimates, but you do need to be careful. The major software packages such as Python, R, SPSS, SAS and Stata all use estimation algorithms to avoid the problem as much as possible.

When multicollinearity is present, the uncertainty associated with individual regression coefficients will be large. This is because they are difficult to estimate. Consequently, statistical tests (e.g., t-tests) on regression coefficients are unreliable. (In forecasting we are rarely interested in such tests.) Also, it will not be possible to make accurate statements about the contribution of each separate predictor to the forecast.

Forecasts will be unreliable if the values of the future predictors are outside the range of the historical values of the predictors. For example, suppose you have fitted a regression model with predictors x_1 and x_2 which are highly correlated with each other, and suppose that the values of x_1 in the training data ranged between 0 and 100. Then forecasts based on x_1>100 or x_1<0 will be unreliable. It is always a little dangerous when future values of the predictors lie much outside the historical range, but it is especially problematic when multicollinearity is present.

Note that if you are using good statistical software, if you are not interested in the specific contributions of each predictor, and if the future values of your predictor variables are within their historical ranges, there is nothing to worry about — multicollinearity is not a problem except when there is perfect correlation.

7.9 Matrix formulation

Warning: this is a more advanced, optional section and assumes knowledge of matrix algebra.

Recall that multiple regression model can be written as $y_{t} = \beta_{0} + \beta_{1} x_{1,t} + \beta_{2} x_{2,t} + \cdots + \beta_{k} x_{k,t} + \varepsilon_{t}$ where ε_{t} has mean zero and variance σ^2. This expresses the relationship between a single value of the forecast variable and the predictors.

It can be convenient to write this in matrix form where all the values of the forecast variable are given in a single equation. Let $\bm{y} = (y_{1},\dots,y_{T})'$, $\bm{\varepsilon} = (\varepsilon_{1},\dots,\varepsilon_{T})'$, $\bm{\beta} = (\beta_{0},\dots,\beta_{k})'$ and $\bm{X} = \left[\begin{matrix} 1 & x_{1,1} & x_{2,1} & \dots & x_{k,1}\\ 1 & x_{1,2} & x_{2,2} & \dots & x_{k,2}\\ \vdots& \vdots& \vdots&& \vdots\\ 1 & x_{1,T}& x_{2,T}& \dots& x_{k,T} \end{matrix}\right]$. Then $\bm{y} = \bm{X}\bm{\beta} + \bm{\varepsilon}$ where $\bm{\varepsilon}$ has mean $\bm{0}$ and variance $\sigma^2\bm{I}$. Note that the $\bm{X}$ matrix has T rows reflecting the number of observations and $k+1$ columns reflecting the intercept which is represented by the column of ones plus the number of predictors.

Least squares estimation

Least squares estimation is performed by minimising the expression $\bm{\varepsilon}'\bm{\varepsilon} = (\bm{y} - \bm{X}\bm{\beta})'(\bm{y} - \bm{X}\bm{\beta})$. It can be shown that this is minimised when $\bm{\beta}$ takes the value $\hat{\bm{\beta}} = (\bm{X}'\bm{X})^{-1}\bm{X}'\bm{y}$. This is sometimes known as the “normal equation”. The estimated coefficients require the inversion of the matrix $\bm{X}'\bm{X}$. If $\bm{X}$ is not of full column rank then matrix $\bm{X}'\bm{X}$ is singular and the model cannot be estimated. This will occur, for example, if you fall for the “dummy variable trap”, i.e., having the same number of dummy variables as there are categories of a categorical predictor, as discussed in Section 7.4.

The residual variance is estimated using $\hat{\sigma}_e^2 = \frac{1}{T-k-1}(\bm{y} - \bm{X}\hat{\bm{\beta}})' (\bm{y} - \bm{X}\hat{\bm{\beta}})$.

Fitted values and cross-validation

The normal equation shows that the fitted values can be calculated using $\bm{\hat{y}} = \bm{X}\hat{\bm{\beta}} = \bm{X}(\bm{X}'\bm{X})^{-1}\bm{X}'\bm{y} = \bm{H}\bm{y}$, where $\bm{H} = \bm{X}(\bm{X}'\bm{X})^{-1}\bm{X}'$ is known as the “hat-matrix” because it is used to compute $\bm{\hat{y}}$ (“y-hat”).

If the diagonal values of $\bm{H}$ are denoted by $h_{1},\dots,h_{T}$, then the cross-validation statistic can be computed using $\text{CV} = \frac{1}{T}\sum_{t=1}^T [e_{t}/(1-h_{t})]^2$, where e_{t} is the residual obtained from fitting the model to all T observations. Thus, it is not necessary to actually fit T separate models when computing the CV statistic.

Forecasts and prediction intervals

Let $\bm{x}^*$ be a row vector containing the values of the predictors (in the same format as $\bm{X}$) for which we want to generate a forecast. Then the forecast is given by $\hat{y} = \bm{x}^*\hat{\bm{\beta}}=\bm{x}^*(\bm{X}'\bm{X})^{-1}\bm{X}'\bm{y}$ and the estimated forecast variance is given by $\hat\sigma_e^2 \left[1 + \bm{x}^* (\bm{X}'\bm{X})^{-1} (\bm{x}^*)'\right]$. A 95% prediction interval can be calculated (assuming normally distributed errors) as $\hat{y} \pm 1.96 \hat{\sigma}_e \sqrt{1 + \bm{x}^* (\bm{X}'\bm{X})^{-1} (\bm{x}^*)'}$. This takes into account the uncertainty due to the error term ε and the uncertainty in the coefficient estimates. However, it ignores any errors in $\bm{x}^*$. Thus, if the future values of the predictors are uncertain, then the prediction interval calculated using this expression will be too narrow.

7.10 Exercises

1. Half-hourly electricity demand for Victoria, Australia is contained in `vic_elec`. Extract the January 2014 electricity demand, and aggregate this data to daily with daily total demands and maximum temperatures.

   a. Plot the data and find the regression model for Demand with temperature as a predictor variable. Why is there a positive relationship?
   b. Produce a residual plot. Is the model adequate? Are there any outliers or influential observations?
   c. Use the model to forecast the electricity demand that you would expect for the next day if the maximum temperature was

15^\circ \text{C} and compare it with the forecast if the with maximum temperature was 35^\circ \text{C}. Do you believe these forecasts?

d. Give prediction intervals for your forecasts.
e. Plot Demand vs Temperature for all of the available data in `vic_elec` aggregated to daily total demand and maximum temperature. What does this say about your model?

2. Data set `olympic_running` contains the winning times (in seconds) in each Olympic Games sprint, middle-distance and long-distance track events from 1896 to 2016.

   a. Plot the winning time against the year for each event. Describe the main features of the plot.
   b. Fit a regression line to the data for each event. Obviously the winning times have been decreasing, but at what *average* rate per year?
   c. Plot the residuals against the year. What does this indicate about the suitability of the fitted lines?
   d. Predict the winning time for each race in the 2020 Olympics. Give a prediction interval for your forecasts. What assumptions have you made in these calculations?

3. An elasticity coefficient is the ratio of the percentage change in the forecast variable (y) to the percentage change in the predictor variable (x). Mathematically, the elasticity is defined as (dy/dx)\times(x/y). Consider the log-log model, \log y=\beta_0+\beta_1 \log x + \varepsilon. Express y as a function of x and show that the coefficient \beta_1 is the elasticity coefficient.

4. The data set `souvenirs` concerns the monthly sales figures of a shop which opened in January 1987 and sells gifts, souvenirs, and novelties. The shop is situated on the wharf at a beach resort town in Queensland, Australia. The sales volume varies with the seasonal population of tourists. There is a large influx of visitors to the town at Christmas and for the local surfing festival, held every March since 1988. Over time, the shop has expanded its premises, range of products, and staff.

   a. Produce a time plot of the data and describe the patterns in the graph. Identify any unusual or unexpected fluctuations in the time series.
   b. Explain why it is necessary to take logarithms of these data before fitting a model.
   c. Fit a regression model to the logarithms of these sales data with a linear trend, seasonal dummies and a “surfing festival” dummy variable.
   d. Plot the residuals against time and against the fitted values. Do these plots reveal any problems with the model?
   e. Do boxplots of the residuals for each month. Does this reveal any problems with the model?
   f. What do the values of the coefficients tell you about each variable?
   g. What does the Ljung-Box test tell you about your model?
   h. Regardless of your answers to the above questions, use your regression model to predict the monthly sales for 1994, 1995, and 1996. Produce prediction intervals for each of your forecasts.
   i. How could you improve these predictions by modifying the model?

5. The `us_gasoline` series consists of weekly data for supplies of US finished motor gasoline product, from 2 February 1991 to 20 January 2017. The units are in “million barrels per day”. Consider only the data to the end of 2004.

   a. Fit a harmonic regression with trend to the data. Experiment with changing the number Fourier terms. Plot the observed gasoline and fitted values and comment on what you see.
   b. Select the appropriate number of Fourier terms to include by minimising the AICc or CV value.
   c. Plot the residuals of the final model using the `plot_diagnostics()` function and comment on these. Use a Ljung-Box test to check for residual autocorrelation.
   d. Generate forecasts for the next year of data and plot these along with the actual data for 2005. Comment on the forecasts.

6. The annual population of Afghanistan is available in the `global_economy` data set.

   a. Plot the data and comment on its features. Can you observe the effect of the Soviet-Afghan war?
   b. Fit a linear trend model and compare this to a piecewise linear trend model with knots at 1980 and 1989.
   c. Generate forecasts from these two models for the five years after the end of the data, and comment on the results.

7. *(For advanced readers following on from* Section 7.9*)*.

   Using matrix notation it was shown that if \bm{y}=\bm{X}\bm{\beta}+\bm{\varepsilon}, where \bm{\varepsilon} has mean \bm{0} and variance matrix \sigma^2\bm{I}, the estimated coefficients are given by \hat{\bm{\beta}}=(\bm{X}'\bm{X})^{-1}\bm{X}'\bm{y} and a forecast is given by \hat{y}=\bm{x}^*\hat{\bm{\beta}}=\bm{x}^*(\bm{X}'\bm{X})^{-1}\bm{X}'\bm{y} where \bm{x}^* is a row vector containing the values of the predictors for the forecast (in the same format as \bm{X}), and the forecast variance is given by \text{Var}(\hat{y})=\sigma^2 \left[1+\bm{x}^*(\bm{X}'\bm{X})^{-1}(\bm{x}^*)'\right].

   Consider the simple time trend model where y_t = \beta_0 + \beta_1t. Using the following results, \sum^{T}_{t=1}{t}=\frac{1}{2}T(T+1),\quad \sum^{T}_{t=1}{t^2}=\frac{1}{6}T(T+1)(2T+1) derive the following expressions:

   a. \displaystyle\bm{X}'\bm{X}=\frac{1}{6}\left[\begin{array}{cc} 6T & 3T(T+1) \\ 3T(T+1) & T(T+1)(2T+1) \\ \end{array} \right]

   b. \displaystyle(\bm{X}'\bm{X})^{-1}=\frac{2}{T(T^2-1)}\left[\begin{array}{cc} (T+1)(2T+1) & -3(T+1) \\ -3(T+1) & 6 \\ \end{array} \right]

c. $\displaystyle\hat{\beta}_0=\frac{2}{T(T-1)}\left[(2T+1)\sum^T_{t=1}y_t-3\sum^T_{t=1}ty_t \right]$

   $\displaystyle\hat{\beta}_1=\frac{6}{T(T^2-1)}\left[2\sum^T_{t=1}ty_t-(T+1)\sum^T_{t=1}y_t \right]$

d. $\displaystyle\text{Var}(\hat{y}_{t})=\hat{\sigma}^2\left[1+\frac{2}{T(T-1)}\left(1-4T-6h+6\frac{(T+h)^2}{T+1}\right)\right]$

7.11 Further reading

There are countless books on regression analysis, but few with a focus on regression for time series and forecasting.

- A good general and modern book on regression is Sheather (2009).
- Another general regression text full of excellent practical advice is Harrell (2015).
- Ord, Fildes, and Kourentzes (2017) provides a practical coverage of regression models for time series in Chapters 7–9, with a strong emphasis on forecasting.

7.12 Used modules and classes

MLForecast

- `MLForecast` class - Core class for fitting machine learning models and generating forecasts
- `PredictionIntervals` class - For generating prediction intervals around forecasts

UtilsForecast

- `plot_series` function - For visualizing time series data and forecasts
- `pipeline` function - For preprocessing time series data and generating features

The code primarily uses MLForecast for regression-based forecasting with scikit-learn models, along with UtilsForecast for visualization and data preprocessing.

← Chapter 6 Judgmental forecasts

Chapter 8 Exponential smoothing →