Time series can often be naturally disaggregated by various attributes of interest. For example, the total number of bicycles sold by a cycling manufacturer can be disaggregated by product type such as road bikes, mountain bikes and hybrids. Each of these can be disaggregated into finer categories. For example hybrid bikes can be divided into city, commuting, comfort, and trekking bikes; and so on. These categories are nested within the larger group categories, and so the collection of time series follows a hierarchical aggregation structure. Therefore we refer to these as “hierarchical time series”.

Hierarchical time series often arise due to geographic divisions. For example, the total bicycle sales can be disaggregated by country, then within each country by state, within each state by region, and so on down to the outlet level.

Alternative aggregation structures arise when attributes of interest are crossed rather than nested. For example, the bicycle manufacturer may be interested in attributes such as frame size, gender, price range, etc. Such attributes do not naturally disaggregate in a unique hierarchical manner as the attributes are not nested. We refer to the resulting time series of crossed attributes as “grouped time series”.

More complex structures arise when attributes of interest are both nested and crossed. For example, it would be natural for the bicycle manufacturer to be interested in sales by product type and also by geographic division. Then both the product groupings and the geographic hierarchy are mixed together. We introduce alternative aggregation structures in Section 11.1.

Forecasts are often required for all disaggregate and aggregate series, and it is natural to want the forecasts to add up in the same way as the data. For example, forecasts of regional sales should add up to forecasts of state sales, which should in turn add up to give a forecast for national sales.

In this chapter we discuss forecasting large collections of time series that aggregate in some way. The challenge is that we require forecasts that are **coherent** across the entire aggregation structure. That is, we require forecasts to add up in a manner that is consistent with the aggregation structure of the hierarchy or group that defines the collection of time series. In this chapter we will relly on StatsForecast (Garza et al. (2022)) HierarchicalForecast(Olivares et al. (2024)).

11.1 Hierarchical and grouped time series

Hierarchical time series

Figure 11.1 shows a simple hierarchical structure. At the top of the hierarchy is the “Total”, the most aggregate level of the data. The tth observation of the Total series is denoted by y_t for t=1,\dots,T. The Total is disaggregated into two series, which in turn are divided into three and two series respectively at the bottom level of the hierarchy. Below the top level, we use y_{j,t} to denote the tth observation of the series corresponding to node j. For example, y_{\text{A},t} denotes the tth observation of the series corresponding to node A, y_{\text{AB},t} denotes the tth observation of the series corresponding to node AB, and so on.

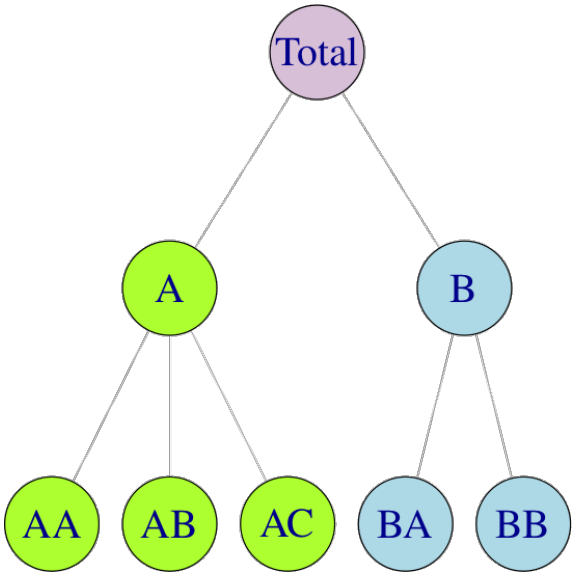

Figure 11.1: A two level hierarchical tree diagram.

In this small example, the total number of series in the hierarchy is n=1+2+5=8, while the number of series at the bottom level is m=5. Note that n>m in all hierarchies.

For any time t, the observations at the bottom level of the hierarchy will sum to the observations of the series above. For example,
y_{t}=y_{\text{AA},t}+y_{\text{AB},t}+y_{\text{AC},t}+y_{\text{BA},t}+y_{\text{BB},t}, \tag{11.1} y_{\text{A},t}=y_{\text{AA},t}+y_{\text{AB},t}+y_{\text{AC},t}\qquad \text{and} \qquad
y_{\text{B},t}=y_{\text{BA},t}+y_{\text{BB},t}. \tag{11.2} Substituting Equation 11.2 into Equation 11.1, we also get y_{t}=y_{\text{A},t}+y_{\text{B},t}.

Example: Australian tourism hierarchy

Australia is divided into six states and two territories, with each one having its own government and some economic and administrative autonomy. For simplicity, we refer to both states and territories as “states”. Each of these states can be further subdivided into regions as shown in Figure 11.2 and Table 11.1. In total there are 76 such regions. Business planners and tourism authorities are interested in forecasts for the whole of Australia, for each of the states and territories, and also for the regions.

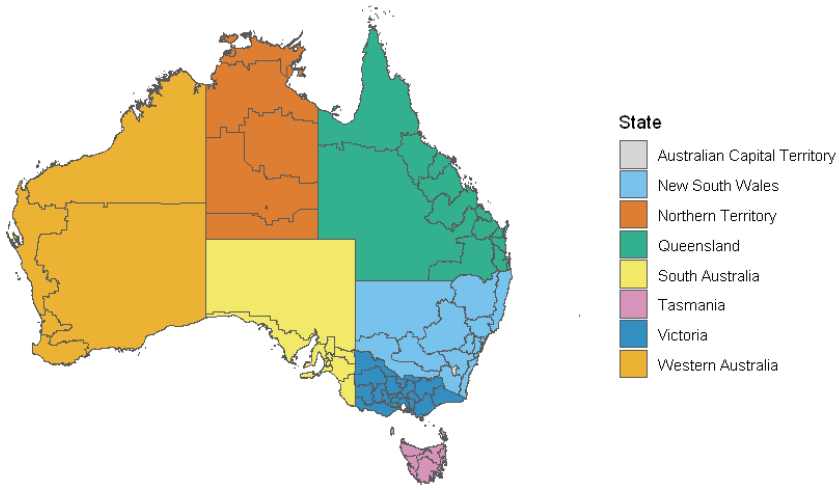

Figure 11.2: Australian states and tourism regions.

Table 11.1: Australian tourism regions.

State	Region
Australian Capital Territory	Canberra
New South Wales	Blue Mountains, Capital Country, Central Coast, Central NSW, Hunter, New England North West, North Coast NSW, Outback NSW, Riverina, Snowy Mountains, South Coast, Sydney, The Murray.
Northern Territory	Alice Springs, Barkly, Darwin, Kakadu Arnhem, Katherine Daly, Lasseter, MacDonnell.
Queensland	Brisbane, Bundaberg, Central Queensland, Darling Downs, Fraser Coast, Gold Coast, Mackay, Northern Outback, Sunshine Coast, Tropical North Queensland, Whitsundays.
South Australia	Adelaide, Adelaide Hills, Barossa, Clare Valley, Eyre Peninsula, Fleurieu Peninsula, Flinders Ranges and Outback, Kangaroo Island, Limestone Coast, Murraylands, Riverland, Yorke Peninsula.
Tasmania	East Coast, Hobart and the South, Launceston Tamar and the North, North West, Wilderness West.
Victoria	Ballarat, Bendigo Loddon, Central Highlands, Central Murray, Geelong and the Bellarine, Gippsland, Goulburn, Great Ocean Road, High Country, Lakes, Macedon, Mallee, Melbourne, Melbourne East, Murray East, Peninsula, Phillip Island, Spa Country, Upper Yarra, Western Grampians, Wimmera.
Western Australia	Australia's Coral Coast, Australia's Golden Outback, Australia's North West, Australia's South West, Experience Perth.

The `tourism` dataset contains data on quarterly domestic tourism demand, measured as the number of overnight trips Australians spend away from home. The key variables `State` and `Region` denote the geographical areas, while a further key `Purpose` describes the purpose of travel. For now, we will ignore the purpose of travel and just consider the geographic hierarchy. To make the graphs and tables simpler, we will recode `State` to use abbreviations.

```
aus_tourism["State"] = aus_tourism["State"].replace(
    {
        "New South Wales": "NSW",
        "Northern Territory": "NT",
        "Queensland": "QLD",
        "South Australia": "SA",
        "Tasmania": "TAS",
        "Victoria": "VIC",
        "Western Australia": "WA",
    }
)
```

Using the `aggregate()` function, we can create the hierarchical time series with overnight trips in regions at the bottom level of the hierarchy, aggregated to states, which are aggregated to the national total. A hierarchical time series corresponding to the nested structure is created using a `parent/child` specification.

```
spec = [
    ["Country"],
    ["Country", "State"],
    ["Country", "State", "Region"],
]
Y_df, S_df, tags = aggregate(df=aus_tourism.drop(columns=["unique_id"]), spec=spec)
Y_df.round(3)
```

	unique_id	ds	y
0	Australia	1998-03-31	23182.197
1	Australia	1998-06-30	20323.380
2	Australia	1998-09-30	19826.641
3	Australia	1998-12-31	20830.130
4	Australia	1999-03-31	22087.353
...	...	...	...
6795	Australia/WA/Experience Perth	2016-12-31	1058.296
6796	Australia/WA/Experience Perth	2017-03-31	956.574
6797	Australia/WA/Experience Perth	2017-06-30	908.335
6798	Australia/WA/Experience Perth	2017-09-30	1006.269
6799	Australia/WA/Experience Perth	2017-12-31	1102.557

6800 rows × 3 columns

The new `Y_df` now has some additional rows corresponding to state and national aggregations for each quarter. Figure 11.3 shows the aggregate total overnight trips for the whole of Australia as well as the states, revealing diverse and rich dynamics. For example, there is noticeable national growth since 2010 and for some states such as the ACT, New South Wales, Queensland, South Australia, and Victoria. There seems to be a significant jump for Western Australia in 2014.

```
states = list(tags["Country/State"])


_, axes = plt.subplots(nrows=4, ncols=2, figsize=(8, 8))
fig = plot_series_utils(Y_df[Y_df['unique_id'].isin(states)].reset_index(), plot_random=False, max_ids=9, ax=axes)
for ax in fig.axes:
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.tick_params(axis='both')
    if ax.get_legend():
        ax.get_legend().remove()
    for label in ax.get_xticklabels():
        label.set_rotation(0)
fig.legends = []
fig.suptitle("Australian tourism")
fig.supylabel("Trips ('000)")
fig.supxlabel("Quarter [1Q]")
fig.subplots_adjust(top=0.93)
fig
```

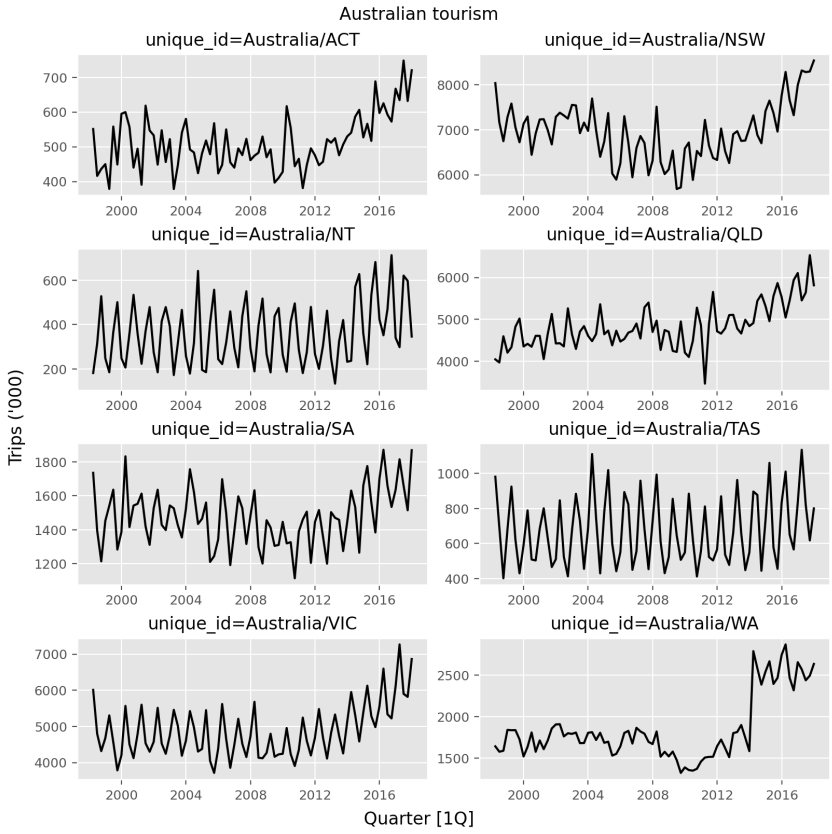

Figure 11.3: Domestic overnight trips from 1998 Q1 to 2017 Q4 aggregated by state.

```
states = ["Australia/QLD", "Australia/VIC", "Australia/NT", "Australia/TAS"]
tourism_sq = Y_df[Y_df['unique_id'].isin(states)]
tourism_sq["year"] = tourism_sq.ds.dt.year
tourism_sq["quarter"] = tourism_sq.ds.dt.quarter
years = tourism_sq["year"].unique()

fig, axs = plt.subplots(nrows=2, ncols=2)
for i, ax in enumerate(axs.flatten()):
    state_data = tourism_sq[tourism_sq['unique_id'] == states[i]]
    sns.lineplot(
        data=state_data,
        x="quarter",
        y="y",
        hue="year",
        palette="husl",
        ax=ax
    )
    ax.set_title(states[i])
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.legend(title="Year", bbox_to_anchor=(1.02, 0.5), loc='center left', borderaxespad=0, frameon=False)
fig.supylabel("Trips ('000)")
fig.supxlabel("Quarter [1Q]")
plt.show()
```

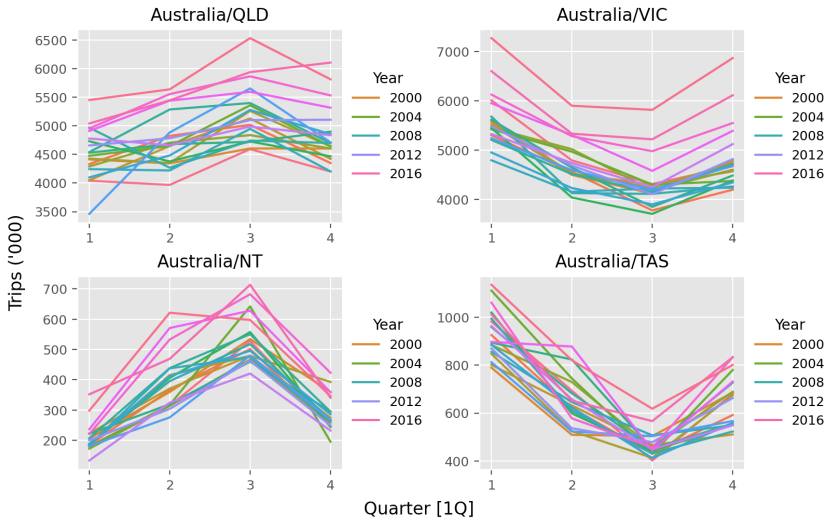

Figure 11.4: Seasonal plots for overnight trips for Queensland and the Northern Territory, and Victoria and Tasmania highlighting the contrast in seasonal patterns between northern and southern states in Australia.

The seasonal pattern of the northern states, such as Queensland and the Northern Territory, leads to peak visits in winter (corresponding to Q3) due to the tropical climate and rainy summer months. In contrast, the southern states tend to peak in summer (corresponding to Q1). This is highlighted in the seasonal plots shown in Figure 11.4 for Queensland and the Northern Territory (shown in the left column) versus the most southern states of Victoria and Tasmania (shown in the right column).

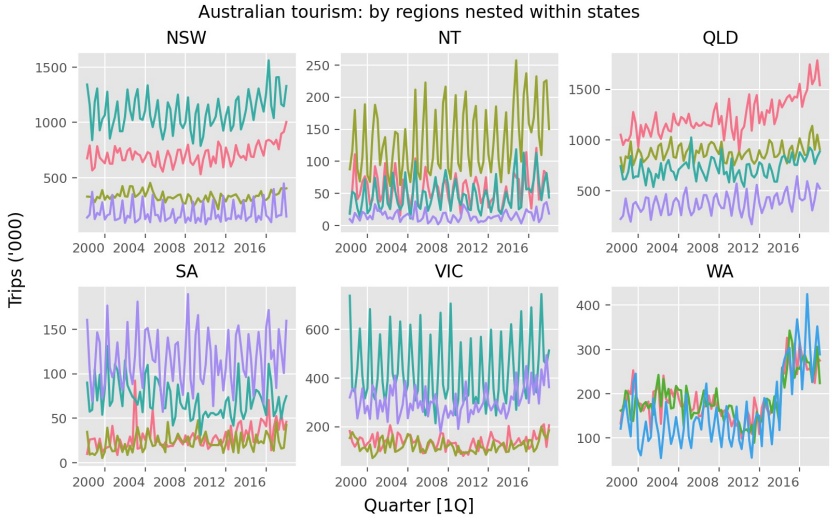

Figure 11.5: Domestic overnight trips from 1998 Q1 to 2017 Q4 for some selected regions.

The plots in Figure 11.5 shows data for some selected regions. These help us visualise the diverse regional dynamics within each state, with some series showing strong trends or seasonality, some showing contrasting seasonality, while some series appear to be just noise.

Grouped time series

With grouped time series, the data structure does not naturally disaggregate in a unique hierarchical manner. Figure 11.6 shows a simple grouped structure. At the top of the grouped structure is the Total, the most aggregate level of the data, again represented by y_t. The Total can be disaggregated by attributes (A, B) forming series y_{\text{A},t} and y_{\text{B},t}, or by attributes (X, Y) forming series y_{\text{X},t} and y_{\text{Y},t}. At the bottom level, the data are disaggregated by both attributes.

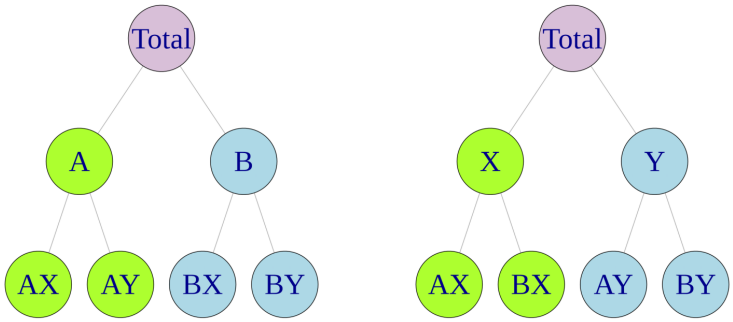

Figure 11.6: Alternative representations of a two level grouped structure.

This example shows that there are alternative aggregation paths for grouped structures. For any time t, as with the hierarchical structure, \begin{equation*} y_{t}=y_{\text{AX},t}+y_{\text{AY},t}+y_{\text{BX},t}+y_{\text{BY},t}. \end{equation*} However, for the first level of the grouped structure, y_{\text{A},t}=y_{\text{AX},t}+y_{\text{AY},t}\quad \quad y_{\text{B},t}=y_{\text{BX},t}+y_{\text{BY},t} \tag{11.3} but also y_{\text{X},t}=y_{\text{AX},t}+y_{\text{BX},t}\quad \quad y_{\text{Y},t}=y_{\text{AY},t}+y_{\text{BY},t}. \tag{11.4}

Grouped time series can sometimes be thought of as hierarchical time series that do not impose a unique hierarchical structure, in the sense that the order by which the series can be grouped is not unique.

Example: Australian prison population

In this example we consider the Australia prison population data introduced in Chapter 2. The top panel in Figure 11.7 shows the total number of prisoners in Australia over the period 2005Q1–2016Q4. This represents the top-level series in the grouping structure. The panels below show the prison population disaggregated or grouped by (a) state (b) legal status (whether prisoners have already been sentenced or are in remand waiting for a sentence), and (c) gender. The three factors are crossed, but none are nested within the others.

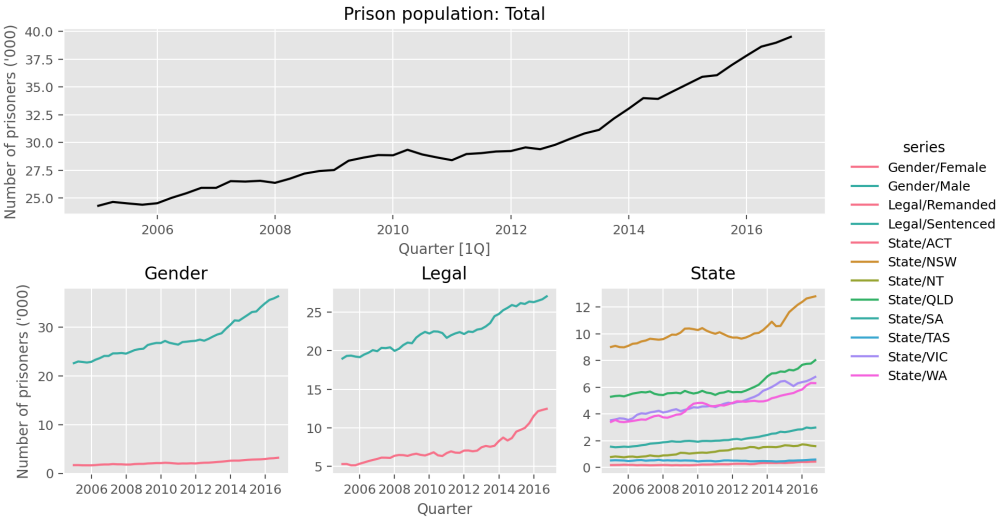

Figure 11.7: Total Australian quarterly adult prison population, disaggregated by state, by legal status, and by gender.

The following code builds a `Y_df` dataset for the prison data.

```
spec = [
    ['Country'],
    ['Country', 'State'],
    ['Country', 'Gender'],
    ['Country', 'Legal'],
    ['Country', 'Indigenous'],
    ['Country', 'State', 'Gender'],
    ['Country', 'State', 'Legal'],
    ['Country', 'Legal', 'Gender'],
    ['Country', 'State', 'Legal', 'Gender'],
    ['Country', 'State', 'Gender', 'Legal', 'Indigenous']
]

Y_df, S_df, tags = aggregate(df=prison, spec=spec)
```

We create a grouped time series using `aggregate()` with attributes or groupings of interest now being crossed using the `spec`.

Figure 11.8 shows the Australian prison population grouped by all possible combinations of two attributes at a time: state and gender, state and legal status, and legal status and gender. The following code will reproduce the first plot in Figure 11.8.

```
fig = plt.figure(figsize=(12, 8))
gs = fig.add_gridspec(3, 8)

states = list(tags["Country/State"])
genders = prison["Gender"].unique()
legals = prison["Legal"].unique()

for i, state in enumerate(states):
    ax = fig.add_subplot(gs[0, i])

    gender_data = pd.concat([Y_df_plot.loc[f"{state}/{label}"]
                             for label in genders], keys=genders)
    gender_data = gender_data.reset_index(level=0)

    sns.lineplot(
        data=gender_data,
        x="ds",
        y="y",
        hue="level_0",
        palette="husl",
        ax=ax
    )
    ax.set_title(f"{state}")
    if i == len(states) - 1:
        ax.legend(title="Gender", bbox_to_anchor=(1.02, 0.5), loc='center left', borderaxespad=0, frameon=False)
    else:
        ax.get_legend().remove()
    ax.set_xlabel("")
    ax.set_ylabel("")
    plt.setp(ax.get_xticklabels(), rotation=90, ha='right')

for i, state in enumerate(states):
    ax = fig.add_subplot(gs[1, i])

    # Prepare data
    legal_data = pd.concat([Y_df_plot.loc[f"{state}/{label}"]
                            for label in legals], keys=legals)
    legal_data = legal_data.reset_index(level=0)

    sns.lineplot(
        data=legal_data,
        x="ds",
        y="y",
        hue="level_0",
        palette="husl",
        ax=ax
    )
    ax.set_title(f"{state}")
    ax.set_xlabel("")
    ax.set_ylabel("")
    if i == len(states) - 1:
        ax.legend(title="Legal Status", bbox_to_anchor=(1.02, 0.5), loc='center left', borderaxespad=0, frameon=False)
    else:
        ax.get_legend().remove()
    plt.setp(ax.get_xticklabels(), rotation=90, ha='right')

ax = fig.add_subplot(gs[2, :4])
remand_data = pd.concat([Y_df_plot.loc[f"Australia/Remanded/{label}"]
                         for label in genders], keys=genders)
remand_data = remand_data.reset_index(level=0)
sns.lineplot(
    data=remand_data,
    x="ds",
    y="y",
    hue="level_0",
    palette="husl",
    ax=ax
)
ax.set_title("Australia/Remanded")
ax.get_legend().remove()
ax.set_xlabel("")
ax.set_ylabel("")
plt.setp(ax.get_xticklabels(), rotation=90, ha='right')

ax = fig.add_subplot(gs[2, 4:])
sentenced_data = pd.concat([Y_df_plot.loc[f"Australia/Sentenced/{label}"]
                            for label in genders], keys=genders)
sentenced_data = sentenced_data.reset_index(level=0)
sns.lineplot(
    data=sentenced_data,
    x="ds",
    y="y",
    hue="level_0",
    palette="husl",
    ax=ax,
)
ax.set_title("Australia/Sentenced")
ax.legend(title="Gender", bbox_to_anchor=(1.02, 0.5), loc='center left', borderaxespad=0, frameon=False)
ax.set_xlabel("")
ax.set_ylabel("")
plt.setp(ax.get_xticklabels(), rotation=90, ha='right')
fig.supylabel("Number of prisoners ('000)")
fig.supxlabel("Quarter")
plt.show()
```

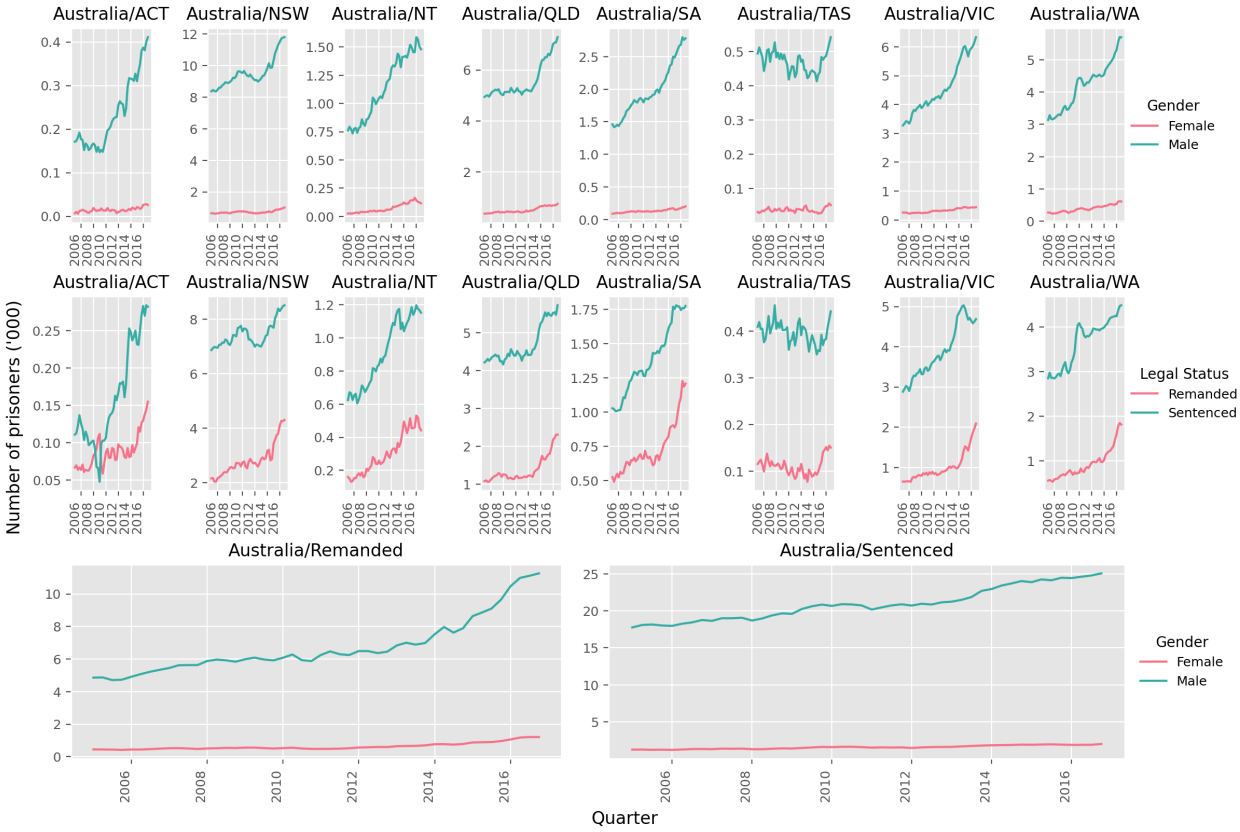

Figure 11.8: Australian adult prison population disaggregated by pairs of attributes.

Figure 11.8 shows the Australian prison population grouped by all possible combinations of two attributes at a time: state and gender, state and legal status, and legal status and gender. These form the bottom-level series of the grouped structure.

```
fig = plt.figure(figsize=(12, 5))
gs = fig.add_gridspec(1, 8)

states = list(tags["Country/State"])
genders = prison["Gender"].unique()
legals = prison["Legal"].unique()

for i, state in enumerate(states):
    ax = fig.add_subplot(gs[0, i])

    series_list = []
    for gender in genders:
        for legal in legals:
            label = f"{state}/{legal}/{gender}"
            series = pd.DataFrame({
                'ds': Y_df_plot.loc[label, "ds"],
                'y': Y_df_plot.loc[label, "y"],
                'category': f"{legal}/{gender}"
            })
            series_list.append(series)

    state_data = pd.concat(series_list)

    sns.lineplot(
        data=state_data,
        x="ds",
        y="y",
        hue="category",
        palette="husl",
        ax=ax
    )
    if i == 0:
        handles, labels = ax.get_legend_handles_labels()

    ax.legend_.remove()  # Remove subplot legend

    #ax.legend(title="Legal/Gender", fontsize=15, title_fontsize=15)
    if i == 0:
        ax.set_ylabel("Number of prisoners ('000)")
    else:
        ax.set_ylabel("")
    ax.set_title(state)
    ax.set_xlabel("")
    plt.setp(ax.get_xticklabels(), rotation=90, ha='right')
fig.legend(handles, labels,
           title="Legal/Gender",
           bbox_to_anchor=(1.02, 0.5),
           loc='center left',
           borderaxespad=0,
           frameon=False)
fig.suptitle("Australian prison population: bottom-level series")
fig.supxlabel("Quarter")
plt.show()
```

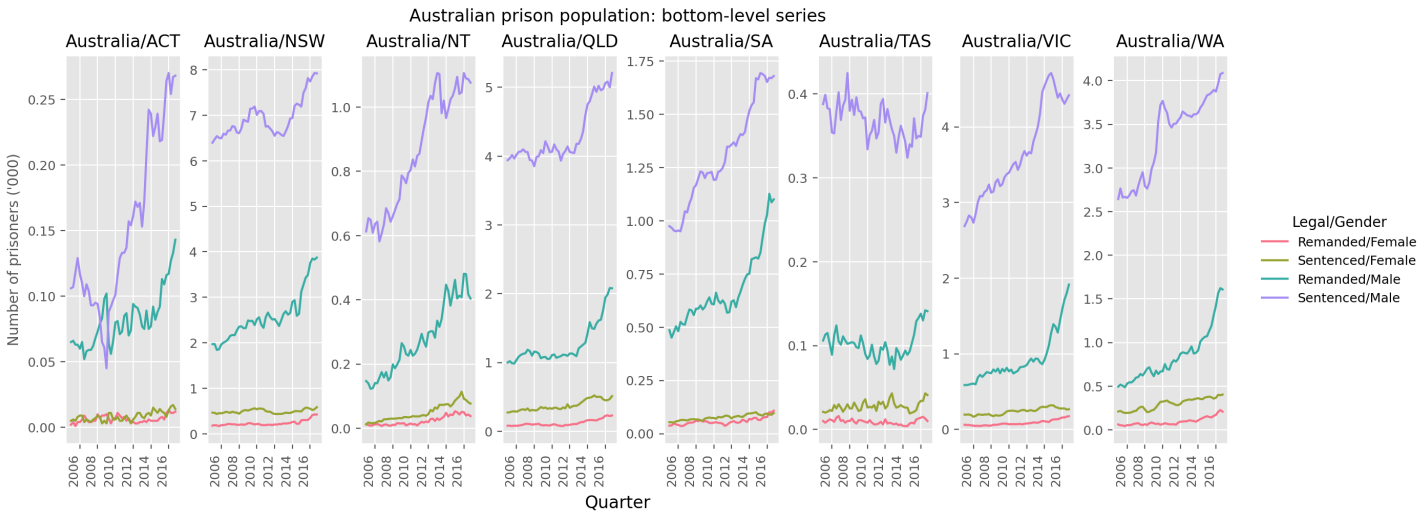

Figure 11.9: Bottom-level time series for the Australian adult prison population, grouped by state, legal status and gender.

Mixed hierarchical and grouped structure

Often disaggregating factors are both nested and crossed. For example, the Australian tourism data can also be disaggregated by the four purposes of travel: holiday, business, visiting friends and relatives, and other. This grouping variable does not nest within any of the geographical variables. In fact, we could consider overnight trips split by purpose of travel for the whole of Australia, and for each state, and for each region. We describe such a structure as a "nested" geographic hierarchy "crossed" with the purpose of travel. Using `aggregate()` this can be specified by simply combining the factors.

The `aus_tourism` dataset contains 425 series, including the 85 series from the hierarchical structure, as well as another 340 series obtained when each series of the hierarchical structure is crossed with the purpose of travel.

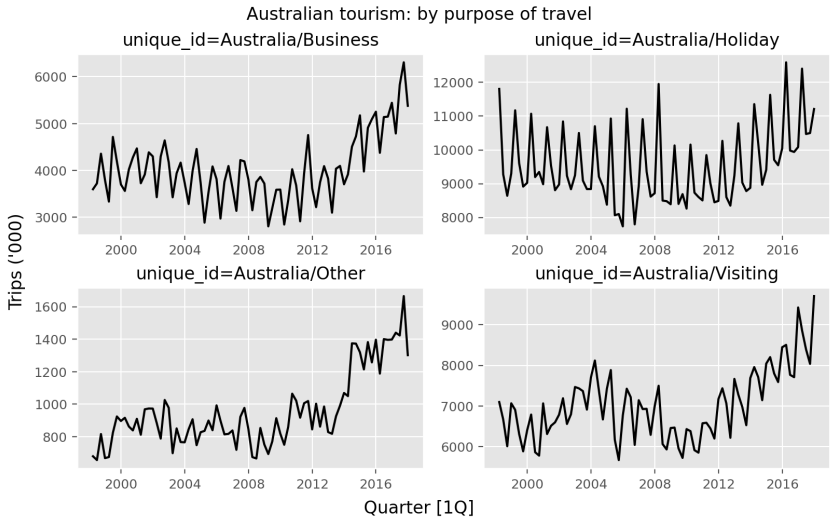

Figure 11.10: Australian domestic overnight trips from 1998 Q1 to 2017 Q4 disaggregated by purpose of travel.

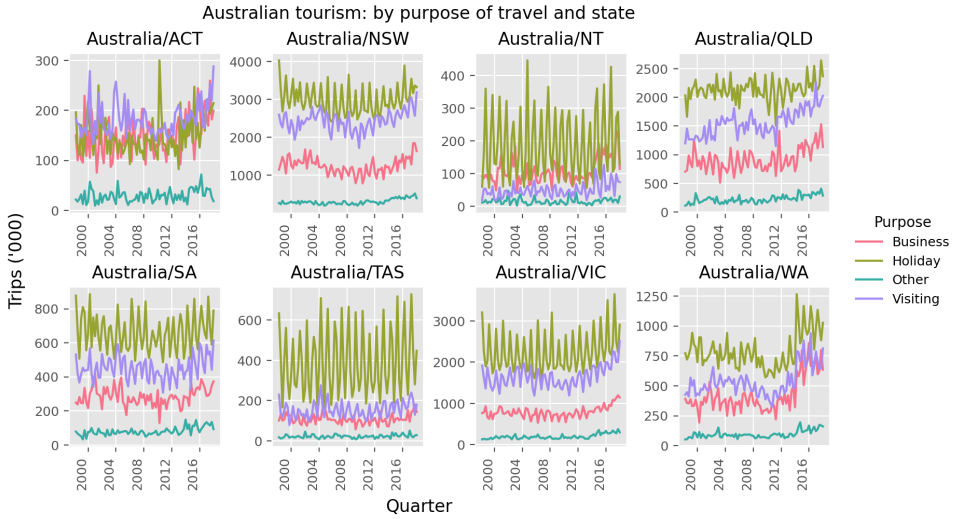

Figure 11.11: Australian domestic overnight trips over the period 1998 Q1 to 2017 Q4 disaggregated by purpose of travel and by state.

Figures 11.10 and 11.11 show the aggregate series grouped by purpose of travel, and the series grouped by purpose of travel and state, revealing further rich and diverse dynamics across these series.

11.2 Single level approaches

Traditionally, forecasts of hierarchical or grouped time series involved selecting one level of aggregation and generating forecasts for that level. These are then either aggregated for higher levels, or disaggregated for lower levels, to obtain a set of coherent forecasts for the rest of the structure.

The bottom-up approach

A simple method for generating coherent forecasts is the “bottom-up” approach. This approach involves first generating forecasts for each series at the bottom level, and then summing these to produce forecasts for all the series in the structure.

For example, for the hierarchy of Figure 11.1, we first generate h-step-ahead forecasts for each of the bottom-level series: \hat{y}_{\text{AA},h},~~\hat{y}_{\text{AB},h},~~\hat{y}_{\text{AC},h},~~ \hat{y}_{\text{BA},h}~~\text{and}~~\hat{y}_{\text{BB},h}. (We have simplified the previously used notation of \hat{y}_{T+h|T} for brevity.)

Summing these, we get h-step-ahead coherent forecasts for the rest of the series: \begin{align*} \tilde{y}_{h} & =\hat{y}_{\text{AA},h}+\hat{y}_{\text{AB},h}+\hat{y}_{\text{AC},h}+\hat{y}_{\text{BA},h}+\hat{y}_{\text{BB},h}, \\ \tilde{y}_{\text{A},h} & = \hat{y}_{\text{AA},h}+\hat{y}_{\text{AB},h}+\hat{y}_{\text{AC},h}, \\ \text{and}\quad \tilde{y}_{\text{B},h} &= \hat{y}_{\text{BA},h}+\hat{y}_{\text{BB},h}. \end{align*} (In this chapter, we will use the “tilde” notation to indicate coherent forecasts.)

An advantage of this approach is that we are forecasting at the bottom level of a structure, and therefore no information is lost due to aggregation. On the other hand, bottom-level data can be quite noisy and more challenging to model and forecast.

Example: Generating bottom-up forecasts

Suppose we want national and state forecasts for the Australian tourism data, but we aren’t interested in disaggregations using regions or the purpose of travel. So we first create a simple dataset containing only state and national trip totals for each quarter.

```
spec = [
    ["Country"],
    ["Country", "State"],
]

Y_df, S_df, tags = aggregate(df=aus_tourism.drop(columns=["unique_id"]), spec=spec)
```

We could generate the bottom-level state forecasts first, and then sum them to obtain the national forecasts.

```
sf = StatsForecast(models=[AutoETS(season_length=4)], freq="Q", n_jobs=-1)
Y_hat_df = sf.forecast(h=4, df=Y_df.query("unique_id in @tags['Country/State']"))
Y_hat_country = Y_hat_df.groupby(["ds"])["AutoETS"].sum()
Y_hat_country.to_frame().round(3)
```

ds	AutoETS
2018-03-31	28962.014
2018-06-30	26955.710
2018-09-30	26297.198
2018-12-31	27195.351

However, we want a more general approach that will work with all the forecasting methods discussed in this chapter. So we will use the `reconcilers` object to specify how we want to compute coherent forecasts.

```
reconcilers = [BottomUp()]
hrec = HierarchicalReconciliation(reconcilers=reconcilers)
sf = StatsForecast(models=[AutoETS(season_length=4)], freq="Q", n_jobs=-1)

Y_hat_df = sf.forecast(h=4, df=Y_df)
Y_rec_df = hrec.reconcile(Y_hat_df=Y_hat_df, Y_df=Y_df, S=S_df, tags=tags)
Y_rec_df.round(3)
```

	unique_id	ds	AutoETS	AutoETS/BottomUp
0	Australia	2018-03-31	29079.439	28962.014
1	Australia	2018-06-30	27765.901	26955.710
2	Australia	2018-09-30	27616.980	26297.198
3	Australia	2018-12-31	28628.884	27195.351
4	Australia/ACT	2018-03-31	707.237	707.237
5	Australia/ACT	2018-06-30	723.998	723.998
6	Australia/ACT	2018-09-30	740.758	740.758
7	Australia/ACT	2018-12-31	757.518	757.518
8	Australia/NSW	2018-03-31	8905.409	8905.409
9	Australia/NSW	2018-06-30	8314.136	8314.136
10	Australia/NSW	2018-09-30	8116.859	8116.859
11	Australia/NSW	2018-12-31	8495.645	8495.645
12	Australia/NT	2018-03-31	262.025	262.025
13	Australia/NT	2018-06-30	509.417	509.417
14	Australia/NT	2018-09-30	673.229	673.229
15	Australia/NT	2018-12-31	367.963	367.963
16	Australia/QLD	2018-03-31	5572.309	5572.309
17	Australia/QLD	2018-06-30	5792.866	5792.866
18	Australia/QLD	2018-09-30	6227.062	6227.062
19	Australia/QLD	2018-12-31	5829.284	5829.284
20	Australia/SA	2018-03-31	1857.898	1857.898
21	Australia/SA	2018-06-30	1705.532	1705.532
22	Australia/SA	2018-09-30	1548.757	1548.757
23	Australia/SA	2018-12-31	1739.847	1739.847
24	Australia/TAS	2018-03-31	1120.978	1120.978
25	Australia/TAS	2018-06-30	796.083	796.083
26	Australia/TAS	2018-09-30	559.931	559.931
27	Australia/TAS	2018-12-31	786.337	786.337
28	Australia/VIC	2018-03-31	7798.503	7798.503
29	Australia/VIC	2018-06-30	6530.873	6530.873
30	Australia/VIC	2018-09-30	5937.754	5937.754
31	Australia/VIC	2018-12-31	6625.065	6625.065
32	Australia/WA	2018-03-31	2737.654	2737.654
33	Australia/WA	2018-06-30	2582.805	2582.805
34	Australia/WA	2018-09-30	2492.850	2492.850
35	Australia/WA	2018-12-31	2593.692	2593.692

The `reconcilers` object can be used in the `HierarchicalReconciliation` class to produce a set of hierarchically coherent forecasts. The `Y_df` dataframe contains the `AutoETS` forecasts as well as the coherent `AutoETS/BottomUp` forecasts, for the 8 states and the national aggregate. At the state level, these forecasts are identical, but the national `AutoETS` forecasts will be different from the national `AutoETS/BottomUp` forecasts.

For bottom-up forecasting, this is rather inefficient as we are not interested in the ETS model for the national total, and the resulting `Y_df` contains a lot of duplicates. But later we will introduce more advanced methods where we will need models for all levels of aggregation, and where the coherent forecasts are different from any of the original forecasts.

Workflow for forecasting aggregation structures

The below code illustrates the general workflow for hierarchical and grouped forecasts.

```
# Load data
data = ...

# Define aggregation structure and build Y_df object
spec = []
Y_df, S_df, tags = aggregate(df=data, spec=spec)

# Choose a set of models and create the base forecasts
sf = StatsForecast(models=[.....],
                   freq=...,
                   n_jobs=-1)

Y_hat_df = sf.forecast(h=...,
                       df=Y_df)

# Choose a set of reconcilers and reconcile the base forecasts
reconcilers = [BottomUp()]

hrec = HierarchicalReconciliation(reconcilers=reconcilers)

Y_rec_df = hrec.reconcile(Y_hat_df=Y_hat_df,
                          Y_df=Y_df,
                          S=S_df,
                          tags=tags)
```

1. Begin with a DataFrame (here labelled `data`) containing the individual bottom-level series.
2. Define in `spec` the aggregation structure and build a `Y_df` object that also contains the aggregate series.
3. Specify a set of models using for each series, at all levels of aggregation.
4. Specify in `reconcilers` how the coherent forecasts are to be generated from the selected models.
5. Use the `HierarchicalReconciliation` class with the `reconcile()` function to generate forecasts for the whole aggregation structure.

Top-down approaches

Top-down approaches involve first generating forecasts for the Total series y_t, and then disaggregating these down the hierarchy.

Let p_1,\dots,p_{m} denote a set of disaggregation proportions which determine how the forecasts of the Total series are to be distributed to obtain forecasts for each series at the bottom level of the structure. For example, for the hierarchy of Figure 11.1, using proportions p_1,\dots,p_{5} we get \tilde{y}_{\text{AA},t}=p_1\hat{y}_t,~~~\tilde{y}_{\text{AB},t}=p_2\hat{y}_t,~~~\tilde{y}_{\text{AC},t}=p_3\hat{y}_t,~~~\tilde{y}_{\text{BA},t}=p_4\hat{y}_t~~~\text{and}~~~~~~\tilde{y}_{\text{BB},t}=p_5\hat{y}_t. Once the bottom-level h-step-ahead forecasts have been generated, these are aggregated to generate coherent forecasts for the rest of the series.

Top-down forecasts can be generated using `TopDown()` within the `reconcilers` object.

There are several possible top-down methods that can be specified. The two most common top-down approaches specify disaggregation proportions based on the historical proportions of the data. These performed well in the study of Gross and Sohl (1990).

Average historical proportions

p_j=\frac{1}{T}\sum_{t=1}^{T}\frac{y_{j,t}}{{y_t}} for j=1,\dots,m. Each proportion p_j reflects the average of the historical proportions of the bottom-level series y_{j,t} over the period t=1,\dots,T relative to the total aggregate y_t.

This approach is implemented in the `TopDown()` class by setting `method = "average_proportions"`.

Proportions of the historical averages

p_j={\sum_{t=1}^{T}\frac{y_{j,t}}{T}}\Big/{\sum_{t=1}^{T}\frac{y_t}{T}} for j=1,\dots,m. Each proportion p_j captures the average historical value of the bottom-level series y_{j,t} relative to the average value of the total aggregate y_t.

This approach is implemented in the `TopDown()` class by setting `method = "proportion_averages"`.

A convenient attribute of such top-down approaches is their simplicity. One only needs to model and generate forecasts for the most aggregated top-level series. In general, these approaches seem to produce quite reliable forecasts for the aggregate levels and they are useful with low count data. On the other hand, one disadvantage is the loss of information due to aggregation. Using such top-down approaches, we are unable to capture and take advantage of individual series characteristics such as time dynamics, special events, different seasonal patterns, etc.

Forecast proportions

Because historical proportions used for disaggregation do not take account of how those proportions may change over time, top-down approaches based on historical proportions tend to produce less accurate forecasts at lower levels of the hierarchy than bottom-up approaches. To address this issue, proportions based on forecasts rather than historical data can be used (Athanasopoulos, Ahmed, and Hyndman 2009).

Consider a one level hierarchy. We first generate h-step-ahead forecasts for all of the series. We don't use these forecasts directly, and they are not coherent (they don't add up correctly). Let's call these "initial" forecasts. We calculate the proportion of each h-step-ahead initial forecast at the bottom level, to the aggregate of all the h-step-ahead initial forecasts at this level. We refer to these as the forecast proportions, and we use them to disaggregate the top-level h-step-ahead initial forecast in order to generate coherent forecasts for the whole of the hierarchy.

For a K-level hierarchy, this process is repeated for each node, going from the top to the bottom level. Applying this process leads to the following general rule for obtaining the forecast proportions: p_j=\prod^{K-1}_{\ell=0}\frac{\hat{y}_{j,h}^{(\ell)}}{\hat{S}_{j,h}^{(\ell+1)}} where j=1,2,\dots,m, \hat{y}_{j,h}^{(\ell)} is the h-step-ahead initial forecast of the series that corresponds to the node which is \ell levels above j, and \hat{S}_{j,h}^{(\ell)} is the sum of the h-step-ahead initial forecasts below the node that is \ell levels above node j and are directly connected to that node. These forecast proportions disaggregate the h-step-ahead initial forecast of the Total series to get h-step-ahead coherent forecasts of the bottom-level series.

We will use the hierarchy of Figure 11.1 to explain this notation and to demonstrate how this general rule is reached. Assume we have generated initial forecasts for each series in the hierarchy. Recall that for the top-level "Total" series, \tilde{y}_{h}=\hat{y}_{h}, for any top-down approach. Here are some examples using the above notation:

- \hat{y}_{\text{A},h}^{(1)}=\hat{y}_{\text{B},h}^{(1)}=\hat{y}_{h}= \tilde{y}_{h};
- \hat{y}_{\text{AA},h}^{(1)}=\hat{y}_{\text{AB},h}^{(1)}=\hat{y}_{\text{AC},h}^{(1)}= \hat{y}_{\text{A},h};
- \hat{y}_{\text{AA},h}^{(2)}=\hat{y}_{\text{AB},h}^{(2)}= \hat{y}_{\text{AC},h}^{(2)}=\hat{y}_{\text{BA},h}^{(2)}= \hat{y}_{\text{BB},h}^{(2)}=\hat{y}_{h}= \tilde{y}_{h};
- \hat{S}_{\text{AA},h}^{(1)} = \hat{S}_{\text{AB},h}^{(1)}= \hat{S}_{\text{AC},h}^{(1)}= \hat{y}_{\text{AA},h}+\hat{y}_{\text{AB},h}+\hat{y}_{\text{AC},h};
- \hat{S}_{\text{AA},h}^{(2)} = \hat{S}_{\text{AB},h}^{(2)}= \hat{S}_{\text{AC},h}^{(2)}= \hat{S}_{\text{A},h}^{(1)} = \hat{S}_{\text{B},h}^{(1)}= \hat{S}_{h}= \hat{y}_{\text{A},h}+\hat{y}_{\text{B},h}.

Moving down the farthest left branch of the hierarchy, coherent forecasts are given by \tilde{y}_{\text{A},h} = \Bigg(\frac{\hat{y}_{\text{A},h}}{\hat{S}_{\text{A},h}^{(1)}}\Bigg) \tilde{y}_{h} = \Bigg(\frac{\hat{y}_{\text{AA},h}^{(1)}}{\hat{S}_{\text{AA},h}^{(2)}}\Bigg) \tilde{y}_{h} and \tilde{y}_{\text{AA},h} = \Bigg(\frac{\hat{y}_{\text{AA},h}}{\hat{S}_{\text{AA},h}^{(1)}}\Bigg) \tilde{y}_{\text{A},h} =\Bigg(\frac{\hat{y}_{\text{AA},h}}{\hat{S}_{\text{AA},h}^{(1)}}\Bigg) \Bigg(\frac{\hat{y}_{\text{AA},h}^{(1)}}{\hat{S}_{\text{AA},h}^{(2)}}\Bigg)\tilde{y}_{h}. Consequently, p_1=\Bigg(\frac{\hat{y}_{\text{AA},h}}{\hat{S}_{\text{AA},h}^{(1)}}\Bigg) \Bigg(\frac{\hat{y}_{\text{AA},h}^{(1)}}{\hat{S}_{\text{AA},h}^{(2)}}\Bigg). The other proportions can be obtained similarly.

This approach is implemented in the `TopDown()` class by setting `method = "forecast_proportions"`. This approach tends to work better than other top-down methods.

One disadvantage of all top-down approaches, is that they may not produce unbiased coherent forecasts (Hyndman et al. 2011) even if the base forecasts are unbiased.

Middle-out approach

The middle-out approach combines bottom-up and top-down approaches. Again, it can only be used for strictly hierarchical aggregation structures.

First, a "middle" level is chosen and forecasts are generated for all the series at this level. For the series above the middle level, coherent forecasts are generated using the bottom-up approach by aggregating the "middle-level" forecasts upwards. For the series below the "middle level", coherent forecasts are generated using a top-down approach by disaggregating the "middle level" forecasts downwards.

This approach is implemented in the `MiddleOut()` class by specifying the appropriate middle level via the `middle_level` argument and selecting the top-down approach with the `top_down_method` argument.

11.3 Forecast reconciliation

Warning: the rest of this chapter is more advanced and assumes a knowledge of some basic matrix algebra.

Matrix notation

Recall that Equations 11.1 and 11.2 represent how data, that adhere to the hierarchical structure of Figure 11.1, aggregate. Similarly Equation 11.3 and Equation 11.4 represent how data, that adhere to the grouped structure of Figure 11.6, aggregate. These equations can be thought of as aggregation constraints or summing equalities, and can be more efficiently represented using matrix notation.

For any aggregation structure we construct an n\times m matrix \bm{S} (referred to as the "summing matrix") which dictates the way in which the bottom-level series aggregate.

For the hierarchical structure in Figure 11.1, we can write \begin{bmatrix} y_{t} \\ y_{\text{A},t} \\ y_{\text{B},t} \\ y_{\text{AA},t} \\ y_{\text{AB},t} \\ y_{\text{AC},t} \\ y_{\text{BA},t} \\ y_{\text{BB},t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{\text{AA},t} \\ y_{\text{AB},t} \\ y_{\text{AC},t} \\ y_{\text{BA},t} \\ y_{\text{BB},t} \end{bmatrix} or in more compact notation \bm{y}_t=\bm{S}\bm{b}_{t}, \tag{11.5} where \bm{y}_t is an n-dimensional vector of all the observations in the hierarchy at time t, \bm{S} is the summing matrix, and \bm{b}_{t} is an m-dimensional vector of all the observations in the bottom level of the hierarchy at time t. Note that the first row in the summing matrix \bm{S} represents Equation 11.1, the second and third rows represent Equation 11.2. The rows below these comprise an m-dimensional identity matrix \bm{I}_m so that each bottom-level observation on the right hand side of the equation is equal to itself on the left hand side.

Similarly for the grouped structure of Figure 11.6 we write \begin{bmatrix} y_{t} \\ y_{\text{A},t} \\ y_{\text{B},t} \\ y_{\text{X},t} \\ y_{\text{Y},t} \\ y_{\text{AX},t} \\ y_{\text{AY},t} \\ y_{\text{BX},t} \\ y_{\text{BY},t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{\text{AX},t} \\ y_{\text{AY},t} \\ y_{\text{BX},t} \\ y_{\text{BY},t} \end{bmatrix}, or \bm{y}_t=\bm{S}\bm{b}_{t}, \tag{11.6} where the second and third rows of \bm{S} represent Equation 11.3 and the fourth and fifth rows represent Equation 11.4.

Mapping matrices

This matrix notation allows us to represent all forecasting methods for hierarchical or grouped time series using a common notation.

Suppose we forecast all series ignoring any aggregation constraints. We call these the **base forecasts** and denote them by \hat{\bm{y}}_h where h is the forecast horizon. They are stacked in the same order as the data \bm{y}_t.

Then all coherent forecasting approaches for either hierarchical or grouped structures can be represented as[1] \tilde{\bm{y}}_h=\bm{S}\bm{G}\hat{\bm{y}}_h, \tag{11.7} where \bm{G} is a matrix that maps the base forecasts into the bottom level, and the summing matrix \bm{S} sums these up using the aggregation structure to produce a set of **coherent forecasts** \tilde{\bm{y}}_h.

The \bm{G} matrix is defined according to the approach implemented. For example if the bottom-up approach is used to forecast the hierarchy of Figure 11.1, then \bm{G}= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\ \end{bmatrix}. Notice that \bm{G} contains two partitions. The first three columns zero out the base forecasts of the series above the bottom level, while the m-dimensional identity matrix picks only the base forecasts of the bottom level. These are then summed by the \bm{S} matrix.

If any of the top-down approaches were used then \bm{G}= \begin{bmatrix} p_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ p_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ p_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ p_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ p_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ \end{bmatrix}. The first column includes the set of proportions that distribute the base forecasts of the top level to the bottom level. These are then summed up by the \bm{S} matrix. The rest of the columns zero out the base forecasts below the highest level of aggregation.

For a middle out approach, the \bm{G} matrix will be a combination of the above two. Using a set of proportions, the base forecasts of some pre-chosen level will be disaggregated to the bottom level, all other base forecasts will be zeroed out, and the bottom-level forecasts will then be summed up the hierarchy via the summing matrix.

Forecast reconciliation

Equation 11.7 shows that pre-multiplying any set of base forecasts with \bm{S}\bm{G} will return a set of coherent forecasts.

The traditional methods considered so far are limited in that they only use base forecasts from a single level of aggregation which have either been aggregated or disaggregated to obtain forecasts at all other levels. Hence, they use limited information. However, in general, we could use other \bm{G} matrices, and then \bm{S}\bm{G} combines and reconciles all the base forecasts in order to produce coherent forecasts.

In fact, we can find the optimal \bm{G} matrix to give the most accurate reconciled forecasts.

The MinT optimal reconciliation approach

Wickramasuriya, Athanasopoulos, and Hyndman (2019) found a \bm{G} matrix that minimises the total forecast variance of the set of coherent forecasts, leading to the MinT (Minimum Trace) optimal reconciliation approach.

Suppose we generate coherent forecasts using Equation 11.7. First we want to make sure we have unbiased forecasts. If the base forecasts \hat{\bm{y}}_h are unbiased, then the coherent forecasts \tilde{\bm{y}}_h will be unbiased provided[2] \bm{S}\bm{G}\bm{S}=\bm{S}. This provides a constraint on the matrix \bm{G}. Interestingly, no top-down method satisfies this constraint, so all top-down approaches result in biased coherent forecasts.

Next we need to find the errors in our forecasts. Wickramasuriya, Athanasopoulos, and Hyndman (2019) show that the variance-covariance matrix of the h-step-ahead coherent forecast errors is given by \begin{equation*} \bm{V}_h = \text{Var}[\bm{y}_{T+h}-\tilde{\bm{y}}_h]=\bm{S}\bm{G}\bm{W}_h\bm{G}'\bm{S}' \end{equation*} where \bm{W}_h=\text{Var}[(\bm{y}_{T+h}-\hat{\bm{y}}_h)] is the variance-covariance matrix of the corresponding base forecast errors.

The objective is to find a matrix \bm{G} that minimises the error variances of the coherent forecasts. These error variances are on the diagonal of the matrix \bm{V}_h, and so the sum of all the error variances is given by the trace of the matrix \bm{V}_h. Wickramasuriya, Athanasopoulos, and Hyndman (2019) show that the matrix \bm{G} which minimises the trace of \bm{V}_h such that \bm{S}\bm{G}\bm{S}=\bm{S}, is given by \bm{G}=(\bm{S}'\bm{W}_h^{-1}\bm{S})^{-1}\bm{S}'\bm{W}_h^{-1}. Therefore, the optimally reconciled forecasts are given by \tilde{\bm{y}}_h=\bm{S}(\bm{S}'\bm{W}_h^{-1}\bm{S})^{-1}\bm{S}'\bm{W}_h^{-1}\hat{\bm{y}}_h. \tag{11.8} We refer to this as the MinT (or Minimum Trace) optimal reconciliation approach. MinT is implemented by the `MinTrace()` class.

To use this in practice, we need to estimate \bm{W}_h, the forecast error variance of the h-step-ahead base forecasts. This can be difficult, and so we provide four simplifying approximations that have been shown to work well in both simulations and in practice.

1. Set \bm{W}_h=k_h\bm{I} for all h, where k_{h} > 0.[3] This is the most simplifying assumption to make, and means that \bm{G} is independent of the data, providing substantial computational savings. The disadvantage, however, is that this specification does not account for the differences in scale between the levels of the structure, or for relationships between series.

   Setting \bm{W}_h=k_h\bm{I} in Equation 11.8 gives the ordinary least squares (OLS) estimator we introduced in Section 7.9 with \bm{X}=\bm{S} and \bm{y}=\hat{\bm{y}}. Hence this approach is usually referred to as OLS reconciliation. It is implemented in `MinTrace()` by setting `method = "ols"`.

2. Set \bm{W}_{h} = k_{h}\text{diag}(\hat{\bm{W}}_{1}) for all h, where k_{h} > 0, \hat{\bm{W}}_{1} = \frac{1}{T}\sum_{t=1}^{T}\bm{e}_{t}\bm{e}_{t}', and \bm{e}_{t} is an n-dimensional vector of residuals of the models that generated the base forecasts stacked in the same order as the data.

   This specification scales the base forecasts using the variance of the residuals and it is therefore referred to as the WLS (weighted least squares) estimator using *variance scaling*. The approach is implemented in `MinTrace()` by setting `method = "wls_var"`.

3. Set \bm{W}_{h}=k_{h}\bm{\Lambda} for all h, where k_{h} > 0, \bm{\Lambda}=\text{diag}(\bm{S}\bm{1}), and \bm{1} is a unit vector of dimension m (the number of bottom-level series). This specification assumes that the bottom-level base forecast errors each have variance k_{h} and are uncorrelated between nodes. Hence each element of the diagonal \bm{\Lambda} matrix contains the number of forecast error variances contributing to each node. This estimator only depends on the structure of the aggregations, and not on the actual data. It is therefore referred to as *structural scaling*. Applying the structural scaling specification is particularly useful in cases where residuals are not available, and so variance scaling cannot be applied; for example, in cases where the base forecasts are generated by judgmental forecasting (Chapter 6). The approach is implemented in `MinTrace()` by setting `method = "wls_struct"`.

4. Set \bm{W}_h = k_h \bm{W}_1 for all h, where k_h>0. Here we only assume that the error covariance matrices are proportional to each other, and we directly estimate the full one-step covariance matrix \bm{W}_1. The most obvious and simple way would be to use the sample covariance. This is implemented in `MinTrace()` by setting `method = "mint_cov"`.

   However, for cases where the number of bottom-level series m is large compared to the length of the series T, this is not a good estimator. Instead we use a shrinkage estimator which shrinks the sample covariance to a diagonal matrix. This is implemented in `MinTrace()` by setting `method = "mint_shrink"`.

In summary, unlike any other existing approach, the optimal reconciliation forecasts are generated using all the information available within a hierarchical or a grouped structure. This is important, as particular aggregation levels or groupings may reveal features of the data that are of interest to the user and are important to be modelled. These features may be completely hidden or not easily identifiable at other levels.

For example, consider the Australian tourism data introduced in Section 11.1, where the hierarchical structure followed the geographic division of a country into states and regions. Some areas will be largely summer destinations, while others may be winter destinations. We saw in Figure 11.4 the contrasting seasonal patterns between the northern and the southern states. These differences will be smoothed at the country level due to aggregation.

11.4 Forecasting Australian domestic tourism

We will compute forecasts for the Australian tourism data that was described in Section 11.1. We use the data up to the end of 2015 as a training set, withholding the final two years (eight quarters, 2016Q1–2017Q4) as a test set for evaluation. The code below demonstrates the full workflow for generating coherent forecasts using the bottom-up, OLS and MinT methods. The workflow for this example is:

1. Begin with a DataFrame (here labelled `aus_tourism`) containing the individual bottom-level series.
2. Define in `spec` the aggregation structure and build a `Y_df` object that also contains the aggregate series.
3. Specify a set of models using for each series, at all levels of aggregation. In this example we use `AutoETS`.
4. Specify in `reconcilers` how the coherent forecasts are to be generated from the selected models. In this example, we use `BottumUp()` and two versions of `MinTrace()`.
5. Use the `HierarchicalReconciliation` class with the `reconcile()` function to generate forecasts for the whole aggregation structure.

```
# 2. Define the spec and aggregate the data
spec = [
    ["Country"],
    ["Country", "State"],
    ["Country", "Purpose"],
    ["Country", "State", "Region"],
    ["Country", "State", "Purpose"],
    ["Country", "State", "Region", "Purpose"],
]

Y_df, S_df, tags = aggregate(aus_tourism.drop(columns=["unique_id"]), spec)

Y_train_df = Y_df.query("ds.dt.year <= 2015")
Y_test_df = Y_df.query("ds.dt.year > 2015")

# 3. Specify a set of models and forecast
sf = StatsForecast(models=[AutoETS(season_length=4)], freq="Q", n_jobs=-1)

Y_hat_df = sf.forecast(h=8, df=Y_train_df, fitted=True)
Y_fitted_df = sf.forecast_fitted_values()

# 4. Specify the reconcilers
reconcilers = [BottomUp(), MinTrace(method="ols"), MinTrace(method="mint_shrink")]

# 5. Reconcile the forecasts
hrec = HierarchicalReconciliation(reconcilers=reconcilers)

Y_rec_df = hrec.reconcile(Y_hat_df=Y_hat_df, Y_df=Y_fitted_df, S=S_df, tags=tags)
```

Figures 11.12 and 11.13 plot the four point forecasts for the overnight trips for the Australian total, the states, and the purposes of travel, along with the actual observations of the test set.

```
plot_aggs = list(tags["Country"]) + list(tags["Country/State"])

_, axes = plt.subplots(nrows=5, ncols=2, figsize=(8, 9))
fig = plot_series_utils(
    df=Y_df.query("unique_id in @plot_aggs and ds.dt.year >= 2011"),
    forecasts_df=Y_rec_df.query(
        "unique_id in @plot_aggs and ds.dt.year >= 2011"
    ),
    plot_random=False,
    max_ids=9,
    ax=axes
)
for ax in fig.axes:
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.tick_params(axis='both')
    if ax.get_legend():
        ax.get_legend().remove()
    for label in ax.get_xticklabels():
        label.set_rotation(0)
fig.supylabel("Trips ('000)")
fig.supxlabel("Quarter [1Q]")
fig.legends = []
handles, labels = fig.axes[0].get_legend_handles_labels()
fig.legend(handles, labels,
           bbox_to_anchor=(1.02, 0.5),
           loc='center left',
           borderaxespad=0,
           frameon=False)
fig
```

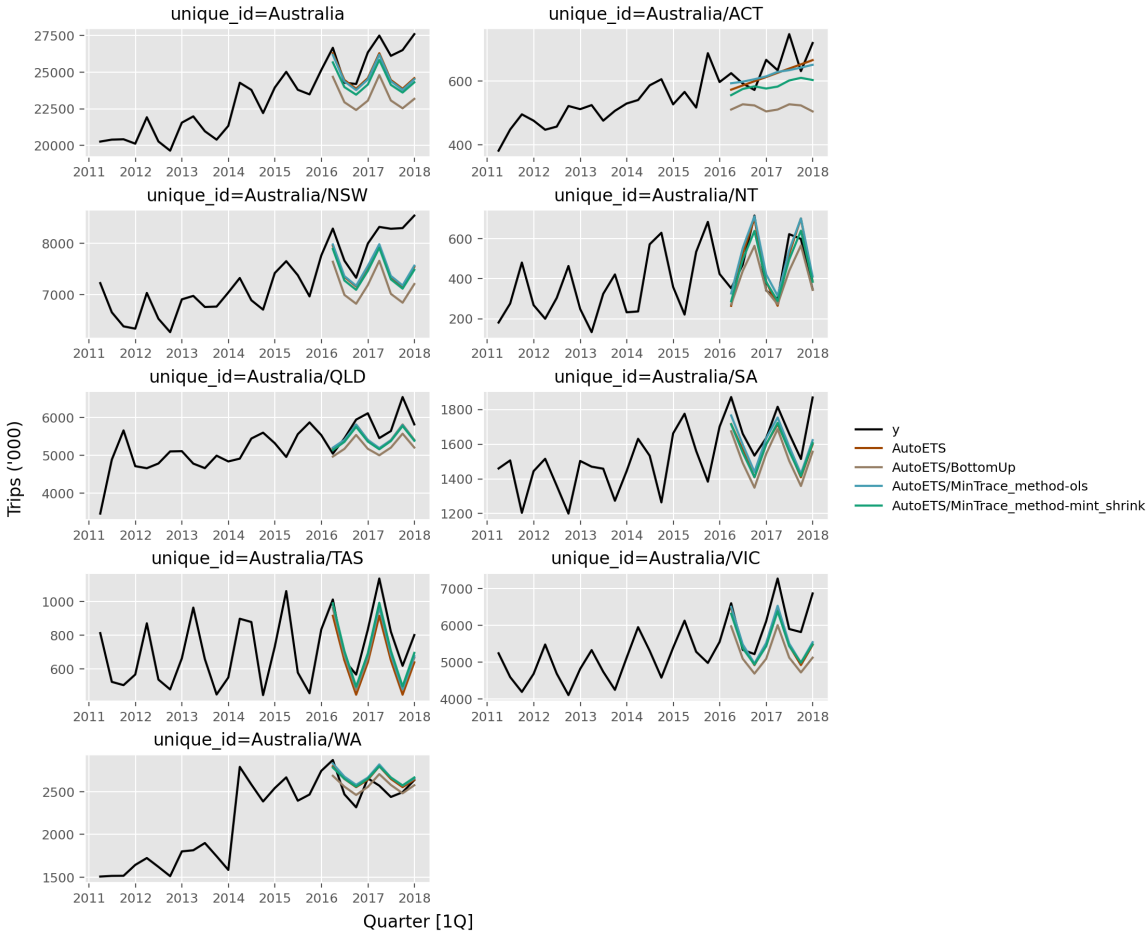

Figure 11.12: Forecasts of overnight trips for Australia and its states over the test period 2016Q1–2017Q4.

```
plot_aggs = list(tags["Country/Purpose"])
_, axes = plt.subplots(nrows=2, ncols=2)
fig = plot_series_utils(
    df=Y_df.query("unique_id in @plot_aggs and ds.dt.year >= 2011"),
    forecasts_df=Y_rec_df.query(
        "unique_id in @plot_aggs and ds.dt.year >= 2011"
    ),
    plot_random=False,
    ax=axes,
)
for ax in fig.axes:
    #ax.set_title(title, fontsize=18)
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.tick_params(axis='both')
    if ax.get_legend():
        ax.get_legend().remove()
    for label in ax.get_xticklabels():
        label.set_rotation(0)
fig.legends = []
handles, labels = fig.axes[0].get_legend_handles_labels()
fig.legend(handles, labels,
           bbox_to_anchor=(1.02, 0.5),
           loc='center left',
           borderaxespad=0,
           frameon=False)
fig.supylabel("Trips ('000)")
fig.supxlabel("Quarter [1Q]")
fig
```

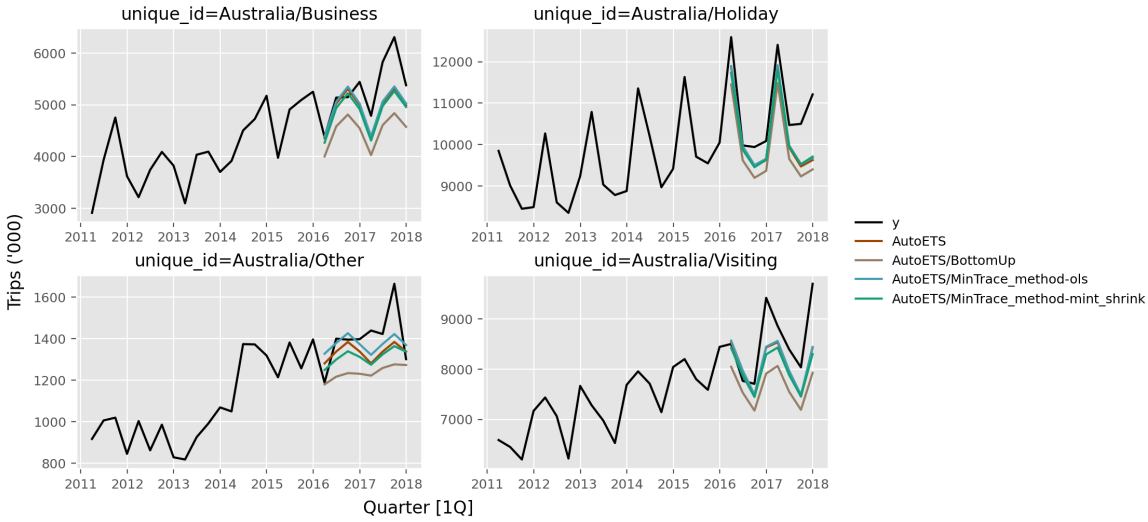

Figure 11.13: Forecasts of overnight trips by purpose of travel over the test period 2016Q1–2017Q4.

To make it easier to see the differences, we have included only the last five years of the training data, and have omitted the prediction intervals. In most panels, the increase in overnight trips, especially in the second half of the test set, is higher than what is predicted by the point forecasts. This is particularly noticeable for the mainland eastern states of ACT, New South Wales, Queensland and Victoria, and across all purposes of travel.

The accuracy of the forecasts over the test set can be evaluated using the `evaluate()` function. We summarise some results in Table 11.2 using RMSE and MASE.

The scales of the series at different levels of aggregation are quite different, due to aggregation. Hence, we need to be cautious when comparing or calculating scale dependent error measures, such as the RMSE, across levels as the aggregate series will dominate. Therefore, we compare error measures across each level of aggregation, before providing the error measures across all the series in the bottom-row. Notice, that the RMSE increases as we go from the bottom level to the aggregate levels above.

The following code generates the accuracy measures for the aggregate series shown in the first row of the table. Similar code is used to evaluate forecasts for other levels.

```
# We define a set of evaluation tags
eval_tags = {}
eval_tags["Total"] = tags["Country"]
eval_tags["Purpose"] = tags["Country/Purpose"]
eval_tags["State"] = tags["Country/State"]
eval_tags["Regions"] = tags["Country/State/Region"]
eval_tags["Bottom"] = tags["Country/State/Region/Purpose"]

# We run the evaluation
eval_df = Y_rec_df.merge(Y_test_df, on=["unique_id", "ds"])
evaluation = evaluate(
    df=eval_df,
    tags=eval_tags,
    train_df=Y_train_df,
    metrics = [rmse,
               partial(mase, seasonality=4)]
)

# We apply some formatting to the evaluation table
evaluation = evaluation.set_index(["level", "metric"])
evaluation = evaluation.rename(index={"Overall": "All series"})
evaluation = evaluation.map('{:.2f}'.format).astype(np.float64)
evaluation.columns = ["Base", "BottomUp", "OLS", "MinT"]
evaluation = evaluation[["Base", "BottomUp", "MinT", "OLS"]]
evaluation = evaluation.stack(sort=False).unstack(["metric", 2], sort=False)

evaluation
```

Table 11.2: Accuracy of forecasts for Australian overnight trips over the test set 2016Q1–2017Q4.

metric	rmse				mase			
	Base	BottomUp	MinT	OLS	Base	BottomUp	MinT	OLS
level								
Total	1713.15	2988.73	2008.69	1780.35	1.53	3.09	1.93	1.60
Purpose	524.21	784.32	552.68	501.59	1.30	2.17	1.41	1.22
State	298.42	407.30	305.92	284.18	1.31	1.85	1.32	1.19
Regions	50.84	54.31	45.53	45.91	1.11	1.18	0.99	0.99
Bottom	19.31	19.31	17.63	18.16	0.99	0.99	0.94	1.02
All series	40.54	49.32	39.41	38.35	1.02	1.06	0.96	1.02

Reconciling the base forecasts using OLS and MinT results in more accurate forecasts compared to the bottom-up approach. This result is commonly observed in applications as reconciliation approaches use information from all levels of the structure, resulting in more accurate coherent forecasts compared to the older traditional methods which use limited information. Furthermore, reconciliation usually improves the incoherent base forecasts for almost all levels.

11.5 Reconciled distributional forecasts

So far we have only discussed the reconciliation of point forecasts. However, we are usually also interested in the forecast distributions so that we can compute prediction intervals.

Panagiotelis et al. (2022) present several important results for generating reconciled probabilistic forecasts. We focus here on two fundamental results.

1. If the base forecasts are normally distributed, i.e., \hat{\bm{y}}_h\sim N(\hat{\bm\mu}_h,\hat{\bm\Sigma}_h), then the reconciled forecasts are also normally distributed, \tilde{\bm{y}}_h \sim

N(\bm{S}\bm{G}\hat{\bm{\mu}}_h,\bm{S}\bm{G}\hat{\bm{\Sigma}}_{h}\bm{G}'\bm{S}').

To generate prediction intervals in this way, we simply set `intervals_method = 'normality'` in the `reconcile()` function. This is the default choice.

2. If it is unreasonable to assume normality for the base forecasts, we can use bootstrapping. Bootstrapped prediction intervals were introduced in Section 5.5. The same idea can be used here. We can simulate future sample paths from the model(s) that produce the base forecasts, and then reconcile these sample paths. Coherent prediction intervals can be computed from the reconciled sample paths.

   Suppose that (\hat{\bm{y}}_h^{[1]},\dots,\hat{\bm{y}}_h^{[B]}) are a set of B simulated sample paths, generated independently from the models used to produce the base forecasts. Then (\bm{S}\bm{G}\hat{\bm{y}}_h^{[1]},\dots,\bm{S}\bm{G}\hat{\bm{y}}_h^{[B]}) provides a set of reconciled sample paths, from which percentiles can be calculated in order to construct coherent prediction intervals.

   To generate bootstrapped prediction intervals in this way, we simply set `intervals_method = 'bootstrap'` in the `reconcile()` function.

11.6 Forecasting Australian prison population

Returning to the Australian prison population data (Section 11.1), we will compare the forecasts from bottom-up and MinT methods applied to base ETS models, using a test set comprising the final two years or eight quarters 2015Q1–2016Q4 of the available data. We will also forecast using the `Naive` model, to compare the results against later on.

```
spec = [
    ["Country"],
    ["Country", "State"],
    ["Country", "Gender"],
    ["Country", "Legal"],
    ["Country", "State", "Gender", "Legal"],
]
prison_scaled = prison.copy()
prison_scaled["y"] /= 1e3
Y_df, S_df, tags = aggregate(prison_scaled, spec)
#Y_df["y"] /= 1e3

Y_train_df = Y_df.query("ds.dt.year <= 2014")
Y_test_df = Y_df.query("ds.dt.year > 2014")

sf = StatsForecast(models=[AutoETS(season_length=4, model="MAM"),
                           Naive()],
                   freq="QS", n_jobs=-1)
levels = list(range(10, 100, 10))
sf.fit(df=Y_train_df)
Y_hat_df = sf.forecast(h=8, df=Y_train_df, fitted=True, level=levels)
Y_fitted_df = sf.forecast_fitted_values()

reconcilers = [BottomUp(), MinTrace(method="mint_shrink")]

hrec = HierarchicalReconciliation(reconcilers=reconcilers)

Y_rec_df = hrec.reconcile(
    Y_hat_df=Y_hat_df, Y_df=Y_fitted_df, S=S_df, tags=tags, level=levels
)

naive_cols = Y_rec_df.filter(like="Naive").columns
Y_rec_df_ets = Y_rec_df.drop(columns=naive_cols)
```

```
plot_aggs = list(tags["Country"])

fig = plot_series(
    df=Y_df.query("unique_id in @plot_aggs"),
    forecasts_df=Y_rec_df_ets.query("unique_id in @plot_aggs"),
    plot_random=False,
    level=[90],
    xlabel="Quarter",
    ylabel="Number of prisoners ('000)",
    title="Australian prison population (total)",
    rm_legend=False,
)
fig.legends = []
handles, labels = fig.axes[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='center left', bbox_to_anchor=(1.02, 0.5),
           borderaxespad=0,
           frameon=False)
fig
```

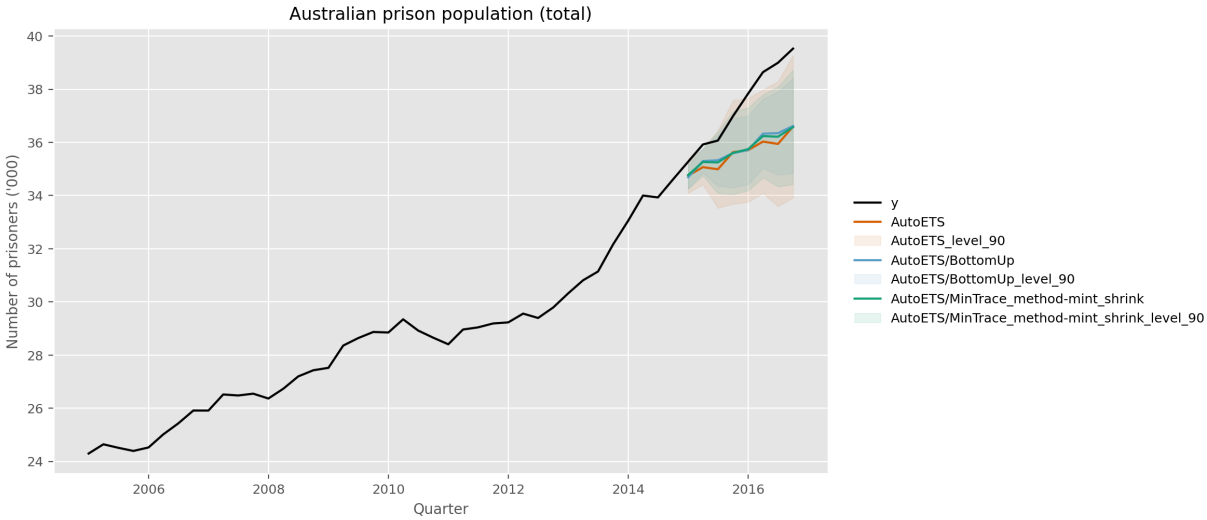

Figure 11.14: Forecasts for the total Australian quarterly adult prison population for the period 2015Q1–2016Q4.

Figure 11.14 shows the three sets of forecasts for the aggregate Australian prison population. The base and bottom-up forecasts from the ETS models seem to underestimate the trend over the test period. The MinT approach combines information from all the base forecasts in the aggregation structure; in this case, the base forecasts at the top level are adjusted upwards.

The MinT reconciled prediction intervals are much tighter than the base forecasts, due to MinT being based on an estimator that minimizes variances. The base forecast distributions are also incoherent, and therefore carry with them the extra uncertainty of the incoherency error.

We exclude the bottom-up forecasts from the remaining plots in order to simplify the visual exploration. However, we do revisit their accuracy in the evaluation results presented later.

Figures 11.15–11.18 show the MinT and base forecasts at various levels of aggregation. To make it easier to see the effect, we only show the last five years of training data. In general, MinT adjusts the base forecasts in the direction of the test set, hence improving the forecast accuracy. There is no guarantee that MinT reconciled forecasts will be more accurate than the base forecasts for every series, but they will be more accurate on average (see Panagiotelis et al. 2021).

```
plot_aggs = list(tags["Country/State"])
Y_rec_df_ets = Y_rec_df_ets.loc[:, ~Y_rec_df_ets.columns.str.contains("BottomUp")]
_, axes = plt.subplots(nrows=4, ncols=2, figsize=(8, 7))
fig = plot_series_utils(
    df=Y_df.query("unique_id in @plot_aggs"),
    forecasts_df=Y_rec_df_ets.query("unique_id in @plot_aggs"),
    plot_random=False,
    level=[90],
    ax=axes
)
for ax in fig.axes:
    ax.set_xlabel("")
    ax.set_ylabel("")
    ax.tick_params(axis='both')
    if ax.get_legend():
        ax.get_legend().remove()
    for label in ax.get_xticklabels():
        label.set_rotation(0)
#fig.legends = []
fig.supylabel("Number of prisoners ('000)")
fig.supxlabel("Quarter [1Q]")
fig.legends = []
handles, labels = fig.axes[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='center left', bbox_to_anchor=(1.02, 0.5),
           borderaxespad=0,
           frameon=False)
fig
```

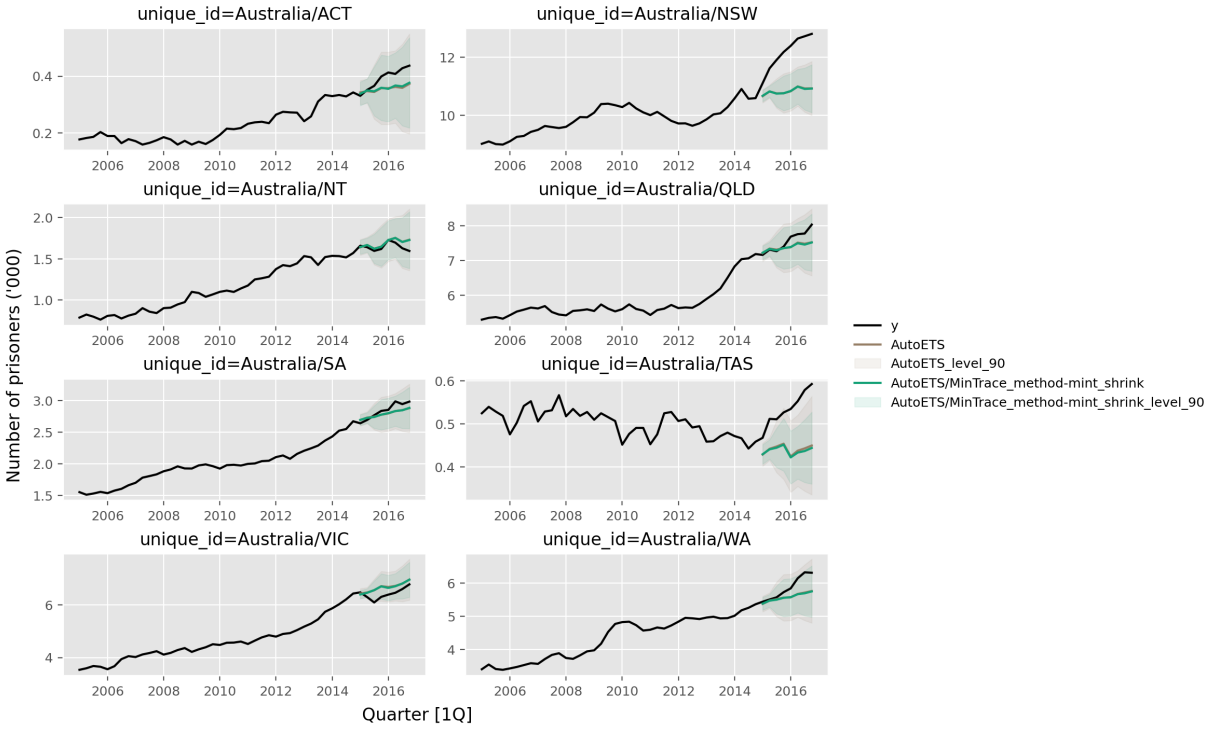

Figure 11.15: Forecasts for the Australian quarterly adult prison population, disaggregated by state.

Figure 11.15 shows forecasts for each of the eight states. There is a general upward trend during the test set period across all the states. However, there appears to be a relatively large and sudden surge in New South Wales and Tasmania, which means the test set observations are well outside the upper bound of the forecast intervals for both these states. Because New South Wales is the state with the largest prison population, this surge will have a substantial impact on the total. In contrast, Victoria shows a substantial dip in 2015Q2–2015Q3, before returning to an upward trend. This dip is not captured in any of the Victorian forecasts.

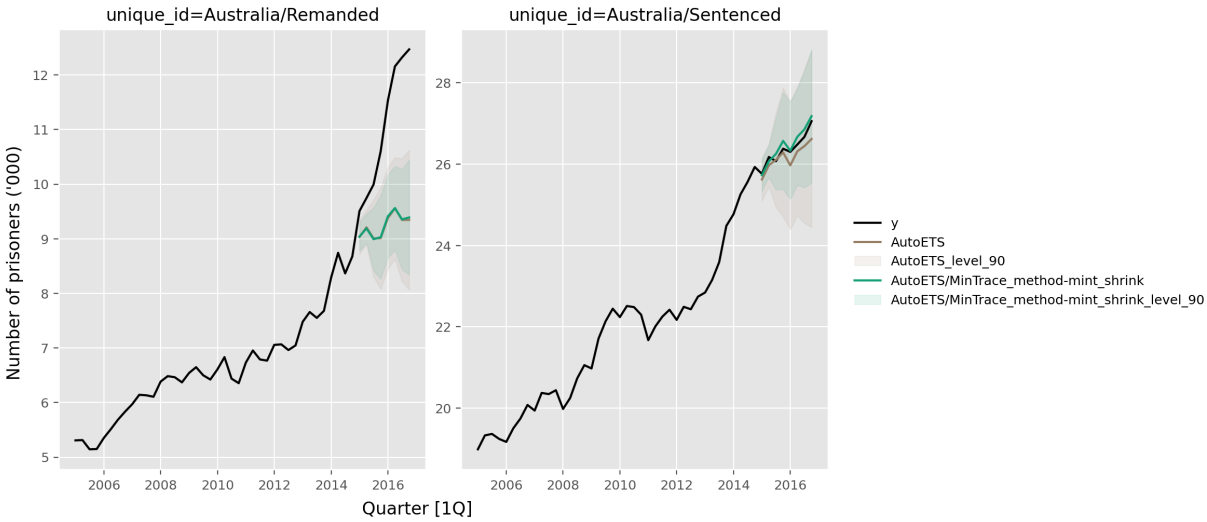

Figure 11.16: Forecasts for the Australian quarterly adult prison population, disaggregated by legal status.

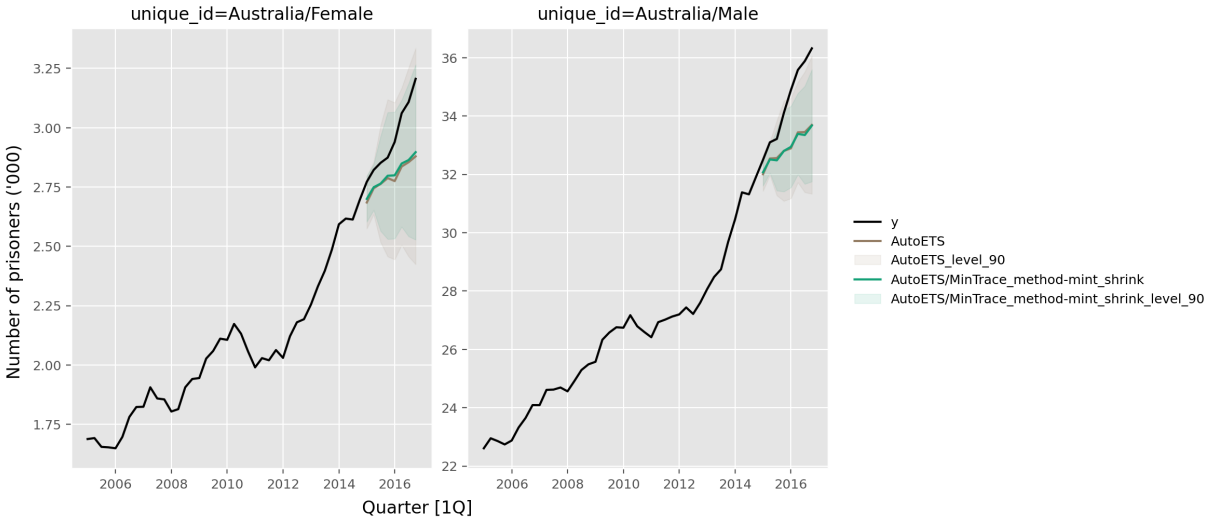

Figure 11.17: Forecasts for the Australian quarterly adult prison population, disaggregated by gender.

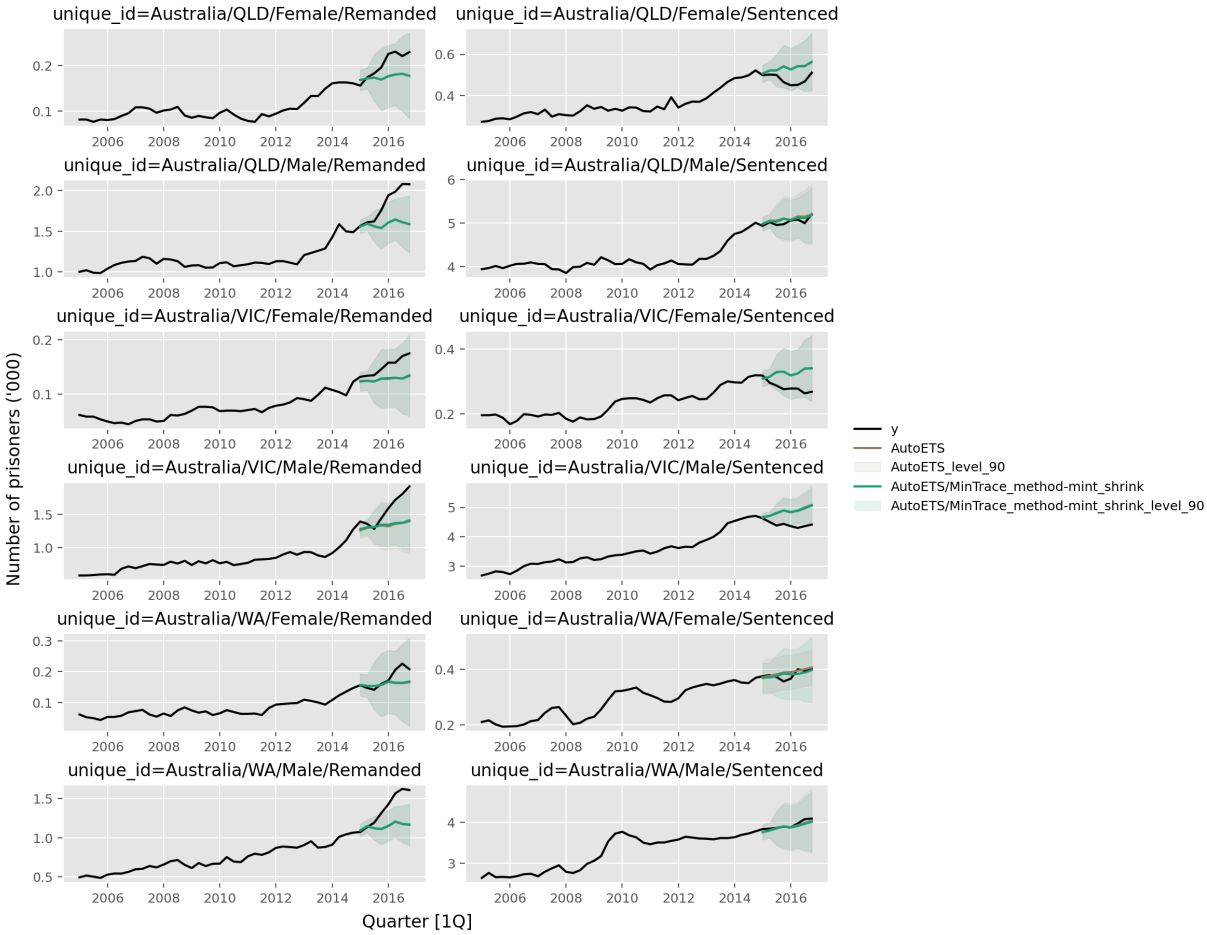

Figure 11.18: Forecasts for bottom-level series the Australian quarterly adult prison population, disaggregated by state, by legal status and by gender.

Figure 11.18 shows the forecasts for some selected bottom-level series of the Australian prison population. The four largest states are represented across the columns, with legal status and gender down the rows. These allow for some interesting analysis and observations that have policy implications. The large increase observed across the states during the 2015Q1–2016Q4 test period appears to be driven by large increases in the remand prison population. These increases seem to be generally missed by both forecasts. In contrast to the other states, for New South Wales there is also a substantial increase in the sentenced prison population. In particular, the increase in numbers of sentenced males in NSW contributes substantially to the rise in state and national prison numbers.

Using the `evaluate()` function, we evaluate the forecast accuracy across the grouped structure. The code below evaluates the forecast accuracy for only the top-level national aggregate of the Australian prison population time series. Similar code is used for the rest of the results shown in Table 11.3.

```
# We define a set of evaluation tags
eval_tags = {}
eval_tags["Total"] = tags["Country"]
eval_tags["State"] = tags["Country/State"]
eval_tags["Legal status"] = tags["Country/Legal"]
eval_tags["Gender"] = tags["Country/Gender"]
eval_tags["Bottom"] = tags["Country/State/Gender/Legal"]

# We run the evaluation
eval_df = Y_rec_df.merge(Y_test_df, on=["unique_id", "ds"])
evaluation = evaluate(
    df=eval_df,
    tags=eval_tags,
    train_df=Y_train_df,
    metrics = [partial(mase, seasonality=4),
               mqloss],
    level=levels,
)

# We apply some formatting to the evaluation table
evaluation = evaluation.set_index(["level", "metric"])
evaluation = evaluation.rename(index={"Overall": "All series",
                                      "mqloss": "skill score (crps)"})
evaluation = evaluation.map('{:.2f}'.format).astype(np.float64)
evaluation = evaluation.stack(sort=False).unstack(["metric", 2], sort=False)

# Calculate crps skill score using Naive's CRPS score as the base
crps_ets = evaluation.loc[:, "skill score (crps)"].filter(like="AutoETS")
crps_naive = evaluation.loc[:, "skill score (crps)"].filter(like="Naive")
crps_skill_score = np.round(100 * ((crps_naive.values - crps_ets.values) / crps_naive.values), 2)
evaluation.loc[:, ("skill score (crps)", crps_ets.columns)] = crps_skill_score

# Prettify table, we only keep AutoETS entries
evaluation = evaluation.filter(like="AutoETS")
evaluation = evaluation.stack(0, sort=False)
evaluation.columns = ["Base", "BottomUp", "MinT"]
evaluation = evaluation.stack().unstack(1).unstack(1).reindex(["Total", "State", "Legal status", "Gender", "Bottom", "All series"])
```

Table 11.3 summarises the accuracy of the base, bottom-up and the MinT reconciled forecasts over the 2015Q1–2016Q4 test period across each of the levels of the grouped aggregation structure as well as all the levels.

Table 11.3: Accuracy of Australian prison population forecasts for different groups of series.

metric	mase			skill score (crps)		
	Base	BottomUp	MinT	Base	BottomUp	MinT
level						
Total	1.64	1.51	1.54	44.07	48.03	47.15
State	1.67	1.64	1.67	25.00	31.25	25.00
Legal status	2.55	2.57	2.49	26.23	28.57	29.03
Gender	1.37	1.30	1.34	49.15	49.21	47.54
Bottom	2.11	2.11	2.10	20.00	20.00	20.00
All series	2.01	2.00	1.99	28.57	33.33	28.57

We use scaled measures because the numbers of prisoners vary substantially across the groups. The MASE gives a scaled measure of point-forecast accuracy (see Section 5.8), while the CRPS skill score gives a scaled measure of distributional forecast accuracy (see Section 5.9). A low value of MASE indicates a good forecast, while a high value of the skill score indicates a good forecast.

The results show that the MinT reconciled forecasts improve on the accuracy of the base forecasts and are also more accurate than the bottom-up forecasts. As the MinT optimal reconciliation approach uses information from all levels in the structure, it generates more accurate forecasts than the traditional approaches (such as bottom-up) which use limited information.

11.7 Exercises

1. Consider the `PBS` data which has aggregation structure `ATC1/ATC2 * Concession * Type`.
   a. Produce plots of the aggregated Scripts data by `Concession`, `Type` and `ATC1`.
   b. Forecast the PBS Scripts data using ETS, ARIMA and seasonal naive models, applied to all but the last three years of data.
   c. Reconcile each of the forecasts using MinT.
   d. Which type of model works best on the test set?
   e. Does the reconciliation improve the forecast accuracy?
   f. Why doesn't the reconciliation make any difference to the seasonal naive forecasts?
2. Repeat the `tourism` example from Section 11.4, but also evaluate the forecast distribution accuracy using CRPS scores. Which method does best on this measure?
3. Repeat the `prison` example from Section 11.6, but using a bootstrap to generate the forecast distributions rather than assuming normality. Does it make much difference to the CRPS scores?

11.8 Further reading

There are no other textbooks which cover hierarchical forecasting in any depth, so interested readers will need to tackle the original research papers for further information.

- Gross and Sohl (1990) provide a good introduction to the top-down approaches.
- A recent survey of forecast reconciliation is provided by Athanasopoulos et al. (2020).
- The reconciliation methods were developed in a series of papers. The later papers summarise previous results and present the most general theory: Wickramasuriya, Athanasopoulos, and Hyndman (2019), Panagiotelis et al. (2021), Panagiotelis et al. (2022).
- Athanasopoulos et al. (2017) extends the reconciliation approach to deal with temporal hierarchies.
- The tourism example is discussed in more detail in Athanasopoulos, Ahmed, and Hyndman (2009), Wickramasuriya, Athanasopoulos, and Hyndman (2019), and Kourentzes and Athanasopoulos (2019).

11.9 Used modules and classes

StatsForecast

- `StatsForecast` class- Core forecasting engine
- `AutoETS` model - For automatic exponential smoothing

HierarchicalForecast

- `HierarchicalReconciliation` class - For reconciling hierarchical forecasts
- `BottomUp`, `TopDown`, `MiddleOut`, `MinTrace` methods - Different reconciliation approaches
- `aggregate` utility - For creating hierarchical structures
- `HierarchicalPlot` utility - For visualizing hierarchical forecasts
- `evaluate` utility - For evaluating hierarchical forecasts

UtilsForecast

- `plot_series` utility - For creating time series visualizations
- `rmse`, `mae`, `smape`, `mase`, `scaled_crps` metrics - For forecast evaluation

. Actually, some recent nonlinear reconciliation methods require a slightly more complicated equation. This equation is for general linear reconciliation methods. □□

!. This "unbiasedness preserving" constraint was first introduced in Hyndman et al. (2011). Panagiotelis et al. (2021) show that this is equivalent to \bm{S}\bm{G} being a projection matrix onto the m-dimensional coherent subspace for which the aggregation constraints hold. □□

!. Note that k_{h} is a proportionality constant. It does not need to be estimated or specified here as it gets cancelled out in Equation 11.8. □□

← Chapter 10 Dynamic regression models

Chapter 12 Advanced forecasting methods →