The `tsfeatures` package developed by Nixtla allows us to calculate various features from time series data. It is the Python equivalent of the R package `tsfeatures`.

We have already seen some time series features. For example, the autocorrelations discussed in Section 2.8 can be considered features of a time series — they are numerical summaries computed from the series. Another feature we saw in the last chapter was the Guerrero estimate of the Box-Cox transformation parameter — again, this is a number computed from a time series.

We can compute many different features on many different time series, and use them to explore the properties of the series. In this chapter we will look at some features that have been found useful in time series exploration, and how they can be used to uncover interesting information about your data. We will use Australian quarterly tourism as a running example (previously discussed in Section 2.5).

4.1 Some simple statistics

Any numerical summary computed from a time series is a feature of that time series — the mean, minimum or maximum, for example. These can be computed using native functions from `pandas`. For example, let's compute the means of all the series in the Australian tourism data.

```
aus_tourism = pd.read_csv("../data/aus_tourism.csv", parse_dates=["ds"])
mean_df = aus_tourism.groupby("unique_id", as_index=False)["y"].mean()
mean_df.sort_values(by="y").head(10)
```

	unique_id	y
158	Kangaroo Island-South Australia-Other	0.340
182	MacDonnell-Northern Territory-Other	0.449
294	Wilderness West-Tasmania-Other	0.478
34	Barkly-Northern Territory-Other	0.632
86	Clare Valley-South Australia-Other	0.898
38	Barossa-South Australia-Other	1.022
154	Kakadu Arnhem-Northern Territory-Other	1.043
170	Lasseter-Northern Territory-Other	1.136
298	Wimmera-Victoria-Other	1.146
183	MacDonnell-Northern Territory-Visiting	1.175

Here we see that the series with least average number of visits was "Other" visits to Kangaroo Island in South Australia.

Rather than compute one feature at a time, it is convenient to compute many features at once. A common short summary of a data set is to compute five summary statistics: the minimum, first quartile, median, third quartile and maximum. These divide the data into four equal-size sections, each containing 25% of the data. The `statistics` can be used to compute them.

```
summary_stats = tsfeatures(aus_tourism, freq=4, features=[statistics],
    scale=False)
summary_stats[["unique_id", "min", "p25", "median", "p75", "max"]].head(10)
```

	unique_id	min	p25	median	p75	max
0	Adelaide Hills-South Australia-Business	0.000	0.000	1.255	3.920	28.602
1	Adelaide Hills-South Australia-Holiday	0.000	5.768	8.516	14.060	35.751
2	Adelaide Hills-South Australia-Other	0.000	0.000	0.908	2.093	8.953
3	Adelaide Hills-South Australia-Visiting	0.778	8.908	12.207	16.806	81.102
4	Adelaide-South Australia-Business	68.725	133.893	152.577	176.936	242.494
5	Adelaide-South Australia-Holiday	108.033	134.627	153.945	172.257	223.557
6	Adelaide-South Australia-Other	25.902	43.866	53.809	62.523	107.495
7	Adelaide-South Australia-Visiting	136.611	178.916	205.582	229.299	269.536
8	Alice Springs-Northern Territory-Business	1.008	9.133	13.324	18.456	34.077
9	Alice Springs-Northern Territory-Holiday	2.809	16.851	31.524	44.784	76.541

4.2 ACF features

Autocorrelations were discussed in Section 2.8. All the autocorrelations of a series can be considered features of that series. We can also summarise the autocorrelations to produce new features; for example, the sum of the first ten squared autocorrelation coefficients is a useful summary of how much autocorrelation there is in a series, regardless of lag.

We can also compute autocorrelations of the changes in the series between periods. That is, we “difference” the data and create a new time series consisting of the differences between consecutive observations. Then we can compute the autocorrelations of this new differenced series. Occasionally it is useful to apply the same differencing operation again, so we compute the differences of the differences. The autocorrelations of this double differenced series may provide useful information.

Another related approach is to compute seasonal differences of a series. If we had monthly data, for example, we would compute the difference between consecutive Januaries, consecutive Februaries, and so on. This enables us to look at how the series is changing between years, rather than between months. Again, the autocorrelations of the seasonally differenced series may provide useful information.

We discuss differencing of time series in more detail in Section 9.1.

Setting `features=[acf_features]` computes a selection of the autocorrelations discussed here. It will return six or seven features:

- the first autocorrelation coefficient from the original data;
- the sum of squares of the first ten autocorrelation coefficients from the original data;
- the first autocorrelation coefficient from the differenced data;
- the sum of squares of the first ten autocorrelation coefficients from the differenced data;
- the first autocorrelation coefficient from the twice differenced data;
- the sum of squares of the first ten autocorrelation coefficients from the twice differenced data;
- For seasonal data, the autocorrelation coefficient at the first seasonal lag is also returned.

When applied to the Australian tourism data, we get the following output.

```
acf_feat = tsfeatures(aus_tourism, freq=4, features=[acf_features])
acf_feat.head(10)
```

	unique_id	x_acf1	x_acf10	diff1_acf1	diff1_acf10	diff2_acf1	diff2_acf10	seas_acf1
0	Adelaide Hills-South Australia-Business	0.071	0.134	-0.580	0.415	-0.750	0.746	-0.063
1	Adelaide Hills-South Australia-Holiday	0.131	0.313	-0.536	0.500	-0.716	0.906	0.208
2	Adelaide Hills-South Australia-Other	0.261	0.330	-0.253	0.317	-0.457	0.392	0.075
3	Adelaide Hills-South Australia-Visiting	0.139	0.117	-0.472	0.239	-0.626	0.408	0.170
4	Adelaide-South Australia-Business	0.033	0.131	-0.520	0.463	-0.676	0.741	0.201
5	Adelaide-South Australia-Holiday	0.046	0.372	-0.343	0.614	-0.487	0.558	0.351
6	Adelaide-South Australia-Other	0.517	1.154	-0.409	0.383	-0.675	0.792	0.342
7	Adelaide-South Australia-Visiting	0.068	0.294	-0.394	0.452	-0.518	0.447	0.345
8	Alice Springs-Northern Territory-Business	0.217	0.367	-0.500	0.381	-0.658	0.587	0.315
9	Alice Springs-Northern Territory-Holiday	-0.007	2.113	-0.153	2.113	-0.274	1.551	0.729

4.3 STL Features

The STL decomposition discussed in Chapter 3 is the basis for several more features.

A time series decomposition can be used to measure the strength of trend and seasonality in a time series. Recall that the decomposition is written as $y_t = T_t + S_{t} + R_t$, where T_t is the smoothed trend component, S_{t} is the seasonal component and R_t is a remainder component. For strongly trended data, the seasonally adjusted data should have much more variation than the remainder component. Therefore $Var(R_t)/Var(T_t+R_t)$ should be relatively small. But for data with little or no trend, the two variances should be approximately the same. So we define the strength of trend as: $F_T = \max\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(T_t+R_t)}\right)$. This will give a measure of the strength of the trend between 0 and 1. Because the variance of the remainder might occasionally be even larger than the variance of the seasonally adjusted data, we set the minimal possible value of F_T equal to zero.

The strength of seasonality is defined similarly, but with respect to the detrended data rather than the seasonally adjusted data: $F_S = \max\left(0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(S_{t}+R_t)}\right)$. A series with seasonal strength F_S close to 0 exhibits almost no seasonality, while a series with strong seasonality will have F_S close to 1 because $Var(R_t)$ will be much smaller than $Var(S_{t}+R_t)$.

These measures can be useful, for example, when you have a large collection of time series, and you need to find the series with the most trend or the most seasonality. These and other STL-based features are computed using `features=[stl_features]`.

```
stl_feat = tsfeatures(aus_tourism, freq=4, features=[stl_features])
stl_feat.head(10)
```

	unique_id	nperiods	seasonal_period	trend	spike	linearity	curvature	e_acf1	e_acf10	seasonal_s
0	Adelaide Hills-South Australia-Business	1	4	0.460	3.223e-04	0.278	-0.627	-0.594	0.502	0.168
1	Adelaide Hills-South Australia-Holiday	1	4	0.531	1.048e-04	1.669	3.925	-0.456	0.342	0.295
2	Adelaide Hills-South Australia-Other	1	4	0.590	5.346e-05	2.486	1.898	-0.295	0.273	0.407
3	Adelaide Hills-South Australia-Visiting	1	4	0.487	5.798e-04	3.231	-0.125	-0.474	0.438	0.248
4	Adelaide-South Australia-Business	1	4	0.462	9.712e-05	-0.106	2.049	-0.538	0.577	0.391
5	Adelaide-South Australia-Holiday	1	4	0.578	1.717e-05	1.850	3.028	-0.525	0.600	0.638
6	Adelaide-South Australia-Other	1	4	0.746	2.364e-05	5.548	2.493	-0.368	0.412	0.209
7	Adelaide-South Australia-Visiting	1	4	0.449	4.824e-05	1.033	2.267	-0.504	1.000	0.476
8	Alice Springs-Northern Territory-Business	1	4	0.552	5.246e-05	3.299	2.586	-0.481	0.519	0.302
9	Alice Springs-Northern Territory-Holiday	1	4	0.379	6.707e-06	-1.076	0.634	-0.529	0.709	0.832

We can then use these features in plots to identify what type of series are heavily trended and what are most seasonal.

```python
stl_feat[["region", "state", "purpose"]] = stl_feat["unique_id"].str.split(
    "-", expand=True
)

fig, axs = plt.subplots(3, 3, figsize=(8, 8))
axs = axs.flatten()
unique_states = stl_feat["state"].unique()
all_handles = []
all_labels = []
for i, state in enumerate(unique_states):
    state_df = stl_feat.query("state == @state")
    ax = axs[i]
    for purpose in state_df["purpose"].unique():
        purpose_df = state_df[state_df["purpose"] == purpose]
        handle = ax.scatter(
            purpose_df["trend"], purpose_df["seasonal_strength"],
            label=purpose
        )
        if purpose not in all_labels:
            all_handles.append(handle)
            all_labels.append(purpose)
    ax.set_title(state)
    ax.set_xlim(0,1)
    ax.set_ylim(0,1)
fig.legend(
    all_handles, all_labels, loc="center left", bbox_to_anchor=(1.02, .5),
    frameon=False, borderaxespad=0,
)
fig.supxlabel('Trend')
fig.supylabel('Seasonal Strength')

# Remove any empty subplots
for j in range(len(unique_states), len(axs)):
    fig.delaxes(axs[j])

plt.show()
```

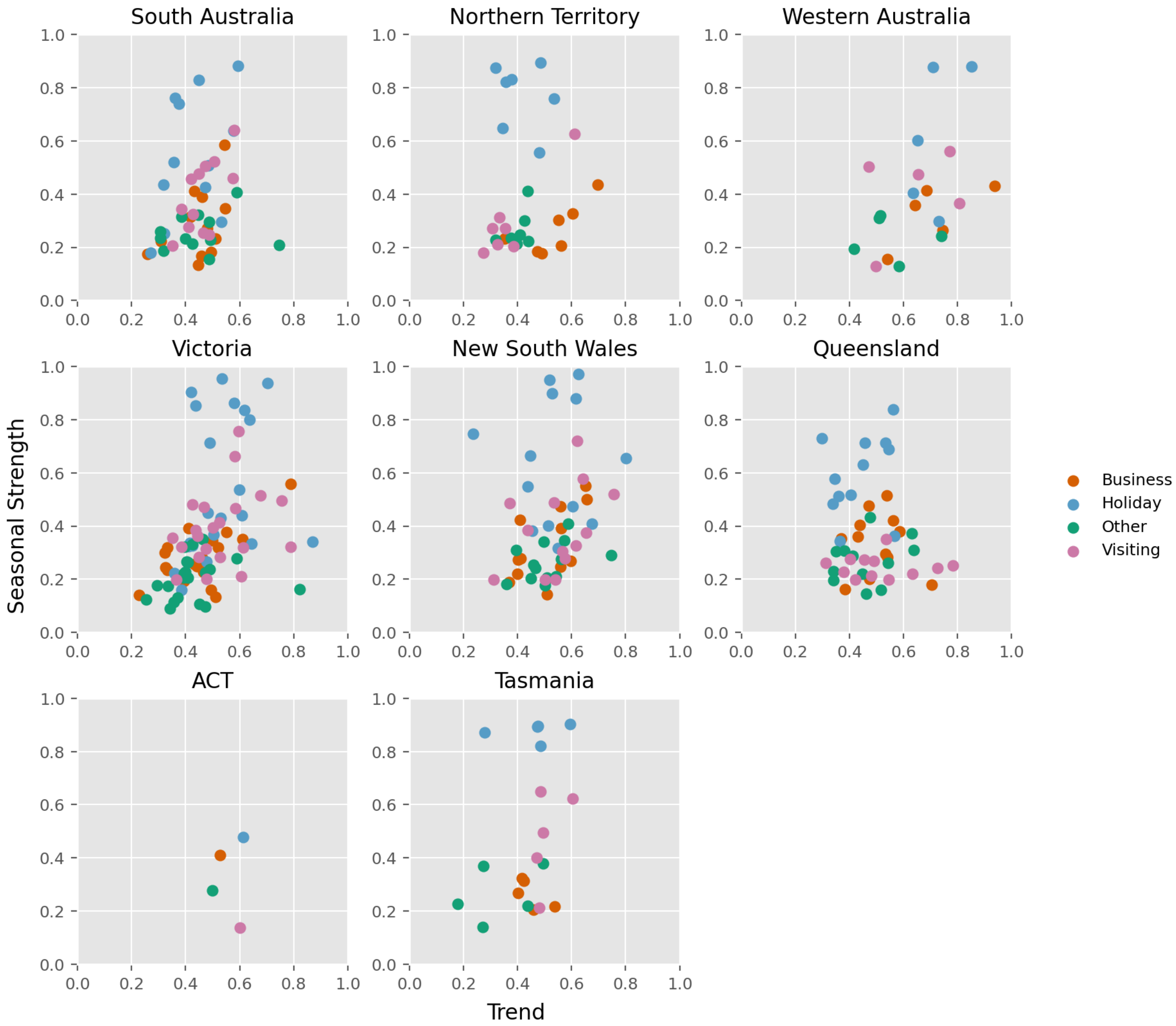

Figure 4.1: Seasonal strength vs trend strength for all tourism series.

Clearly, holiday series are most seasonal which is unsurprising. The strongest trends tend to be in Western Australia and Victoria. The most seasonal series can also be easily identified and plotted.

```
max_seasonal_strength_row = \
    stl_feat.loc[stl_feat["seasonal_strength"].idxmax()]
max_unique_id = max_seasonal_strength_row["unique_id"]
aus_tourism["ds"] = pd.to_datetime(aus_tourism["ds"])
plot_series(aus_tourism, ids=[max_unique_id],
            xlabel="Quarter", ylabel="Trips",
            title="New South Wales-Snowy Mountains-Holiday")
```

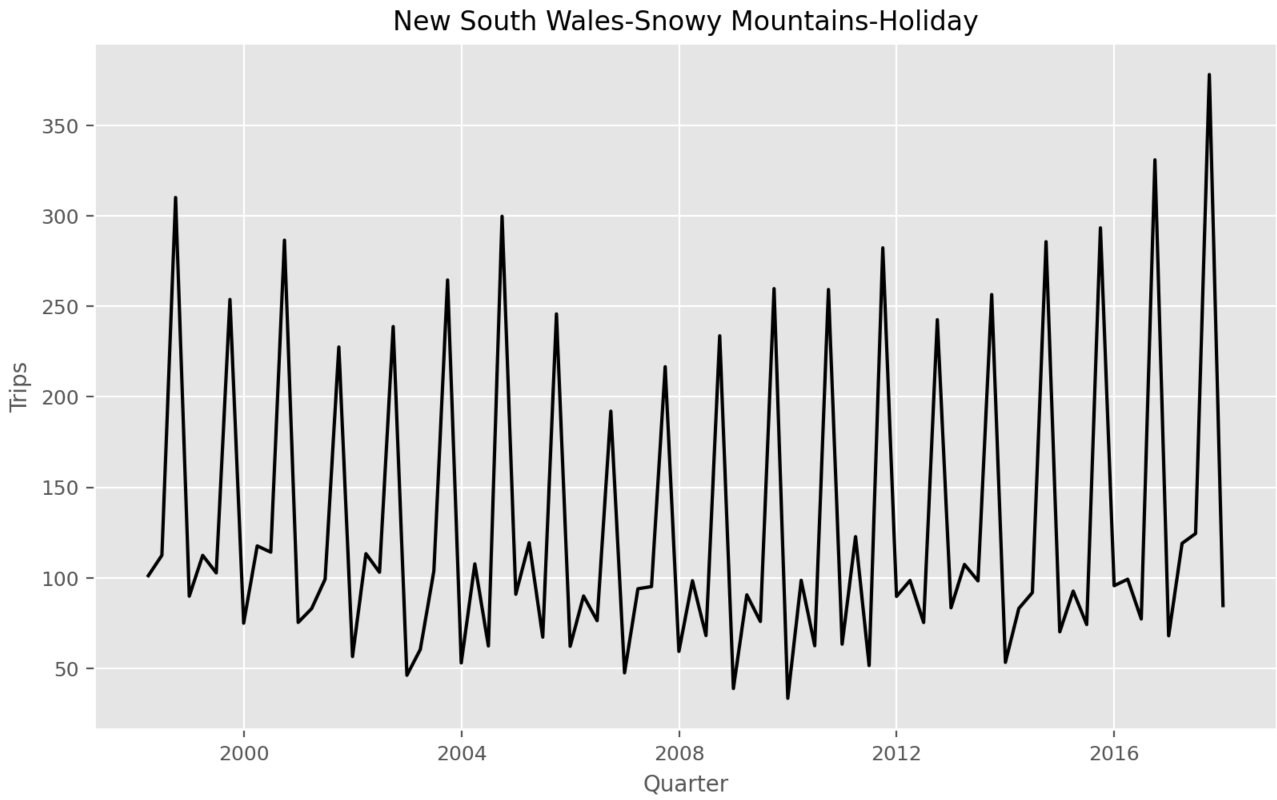

Figure 4.2: The most seasonal series in the Australian tourism data.

This shows holiday trips to the most popular ski region of Australia.

The `stl_features` function returns several more features other than those discussed above.

- `peak` indicates the timing of the peaks — which month or quarter contains the largest seasonal component. This tells us something about the nature of the seasonality. In the Australian tourism data, if Quarter 3 is the peak seasonal period, then people are travelling to the region in winter, whereas a peak in Quarter 1 suggests that the region is more popular in summer.
- `trough` indicates the timing of the troughs — which month or quarter contains the smallest seasonal component.
- `spike` measures the prevalence of spikes in the remainder component R_t of the STL decomposition. It is the variance of the leave-one-out variances of R_t.
- `linearity` measures the linearity of the trend component of the STL decomposition. It is based on the coefficient of a linear regression applied to the trend component.
- `curvature` measures the curvature of the trend component of the STL decomposition. It is based on the coefficient from an orthogonal quadratic regression applied to the trend component.
- `e_acf1` is the first autocorrelation coefficient of the remainder series.
- `e_acf10` is the sum of squares of the first ten autocorrelation coefficients of the remainder series.

4.4 Other features

Many more features are possible, and the `tsfeatures` library computes only a few dozen features that have proven useful in time series analysis. It is also easy to add your own features by writing a Python function that takes a univariate time series input and returns a numerical vector containing the feature values.

The remaining features in the `tsfeatures` library, not previously discussed, are listed here for reference. The details of some of them are discussed later in the book.

- `hurst` will calculate the Hurst coefficient of a time series which is a measure of “long memory”. A series with long memory will have significant autocorrelations for many lags.
- `feat_spectral` will compute the (Shannon) spectral entropy of a time series, which is a measure of how easy the series is to forecast. A series which has strong trend and seasonality (and so is easy to forecast) will have entropy close to 0. A series that is very noisy (and so is difficult to forecast) will have entropy close to 1.
- `box_pierce` gives the Box-Pierce statistic for testing if a time series is white noise, and the corresponding p-value. This test is discussed in Section 5.4.
- `ljung_box` gives the Ljung-Box statistic for testing if a time series is white noise, and the corresponding p-value. This test is discussed in Section 5.4.
- The kth partial autocorrelation measures the relationship between observations k periods apart after removing the effects of

observations between them. So the first partial autocorrelation (k=1) is identical to the first autocorrelation, because there is nothing between consecutive observations to remove. Partial autocorrelations are discussed in Section 9.5. The `feat_pacf` function contains several features involving partial autocorrelations including the sum of squares of the first five partial autocorrelations for the original series, the first-differenced series and the second-differenced series. For seasonal data, it also includes the partial autocorrelation at the first seasonal lag.

- `n_crossing_points` computes the number of times a time series crosses the median.
- `stat_arch_lm` returns the statistic based on the Lagrange Multiplier (LM) test of Engle (1982) for autoregressive conditional heteroscedasticity (ARCH).
- `guerrero` computes the optimal \lambda value for a Box-Cox transformation using the Guerrero method (discussed in Section 3.1).

4.5 Exploring Australian tourism data

All of the features included in the `tsfeatures` package can be computed in one line like this.

```
all_features = [
    acf_features,
    arch_stat,
    crossing_points,
    entropy,
    flat_spots,
    heterogeneity,
    holt_parameters,
    lumpiness,
    nonlinearity,
    pacf_features,
    stl_features,
    stability,
    hw_parameters,
    unitroot_kpss,
    unitroot_pp,
    series_length,
    hurst,
]
all_feat = tsfeatures(aus_tourism, freq=4, features=all_features)
all_feat.head(10)
```

	unique_id	hurst	series_length	unitroot_pp	unitroot_kpss	hw_alpha	hw_beta	hw_gamma	stability
0	Adelaide Hills-South Australia-Business	NaN	80	-80.527	0.060	1.491e-08	1.520e-09	0.000e+00	0.294
1	Adelaide Hills-South Australia-Holiday	0.823	80	-74.302	0.476	9.899e-02	9.899e-02	1.122e-10	0.409
2	Adelaide Hills-South Australia-Other	0.599	80	-55.385	0.605	1.490e-08	2.417e-09	1.799e-01	0.274
3	Adelaide Hills-South Australia-Visiting	0.757	80	-71.076	0.749	1.649e-08	4.162e-09	9.433e-10	0.333
4	Adelaide-South Australia-Business	0.840	80	-78.764	0.221	1.306e-01	1.306e-01	4.671e-13	0.318
5	Adelaide-South Australia-Holiday	0.709	80	-72.567	0.449	1.758e-01	1.758e-01	1.040e-15	0.312
6	Adelaide-South Australia-Other	0.776	80	-36.405	1.406	1.068e-01	7.719e-02	3.203e-12	0.574
7	Adelaide-South Australia-Visiting	0.737	80	-74.137	0.264	1.630e-01	0.000e+00	0.000e+00	0.267
8	Alice Springs-Northern Territory-Business	0.673	80	-65.845	0.807	1.456e-01	0.000e+00	6.254e-13	0.439
9	Alice Springs-Northern Territory-Holiday	0.579	80	-54.382	0.387	1.490e-08	0.000e+00	0.000e+00	0.084

10 rows × 43 columns

This gives 42 features for every combination of the three key variables (`Region`, `State` and `Purpose`). We can treat this dataframe like any data set and analyse it to find interesting observations or groups of observations.

We've already seen how we can plot one feature against another (Section 4.3). We can also do pairwise plots of groups of features. In Figure 4.3, for example, we show all features that involve seasonality, along with the `Purpose` variable.

```python
seasonal_feat = all_feat[
    ["unique_id", "seasonal_strength", "peak", "trough",
     "seas_acf1", "seas_pacf"]
]
seasonal_feat[["region", "state", "purpose"]] = \
    seasonal_feat["unique_id"].str.split("-", expand=True)
g = sns.pairplot(seasonal_feat, hue="purpose")
g.fig.set_size_inches(10, 10)
plt.show()
```

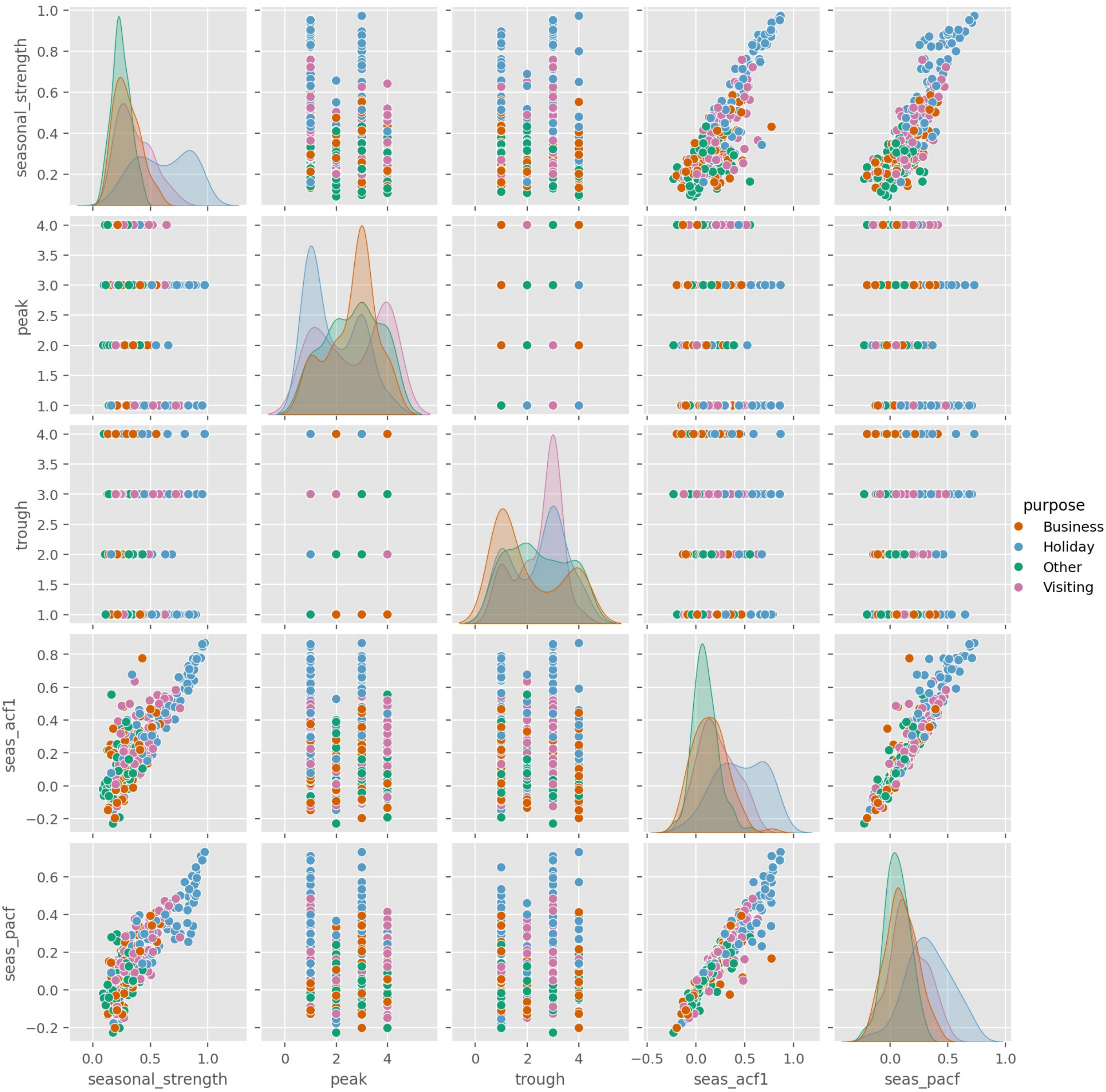

Figure 4.3: Pairwise plots of all the seasonal features for the Australian tourism data

Here, the `Purpose` variable is mapped to colour. There is a lot of information in this figure, and we will highlight just a few things we can learn.

- The three numerical measures related to seasonality (`seasonal_strength`, `season_acf1` and `season_pacf`) are all positively correlated.
- The bottom left panel and the top right panel both show that the most strongly seasonal series are related to holidays (as we saw previously).

- The bar plots in the bottom row of the `peak` and `trough` columns show that seasonal peaks in Business travel occur most often in Quarter 3, and least often in Quarter 1.

It is difficult to explore more than a handful of variables in this way. A useful way to handle many more variables is to use a dimension reduction technique such as principal components. This gives linear combinations of variables that explain the most variation in the original data. We can compute the principal components of the tourism features as follows.

```
all_feat['purpose'] = all_feat['unique_id'].str.split('-').str[-1]
all_feat = all_feat.dropna(axis=1)

features = all_feat.columns.drop(["unique_id", "purpose"])
X = all_feat[features]
y = all_feat["purpose"]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)

pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_features)

# Plotting the first two PCA components
pca_df = pd.DataFrame(data=principal_components, columns=["PC1", "PC2"])
pca_df["purpose"] = y

fig, ax = plt.subplots()
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="purpose", s=100, ax=ax)
ax.set_title("PCA of Features")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.get_legend().remove()
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, title="Purpose", loc="center left", bbox_to_anchor=(1.02, .5),
    frameon=False, borderaxespad=0)
plt.show()
```

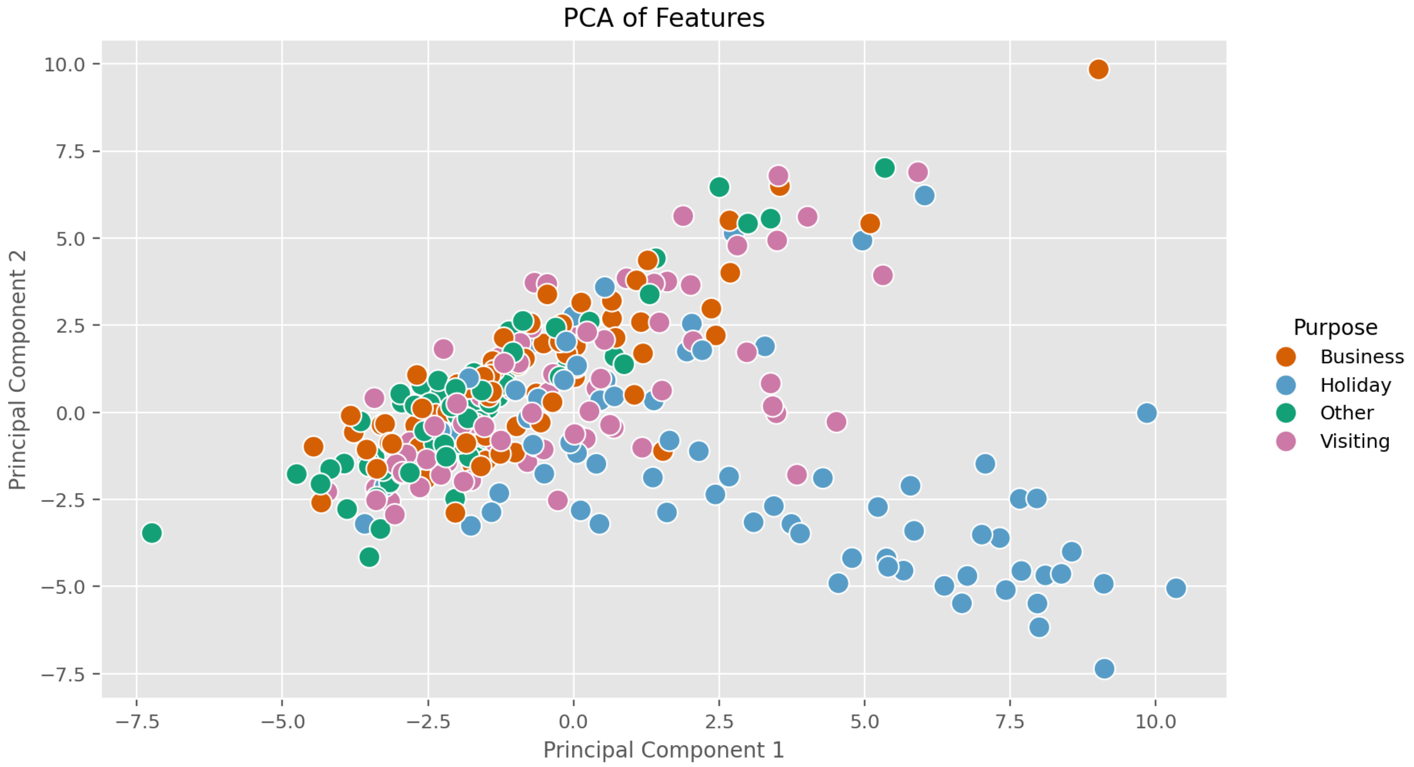

Figure 4.4: A plot of the first two principal components, calculated from the 37 features of the Australian quarterly tourism data.

Each point on Figure 4.4 represents one series and its location on the plot is based on all 37 features. The first principal component (`PC1`) is the linear combination of the features which explains the most variation in the data. The second principal component (`PC2`) is the linear combination which explains the next most variation in the data, while being uncorrelated with the first principal component. For more information about principal component dimension reduction, see Izenman (2008).

Figure 4.4 reveals a few things about the tourism data. First, the holiday series behave quite differently from the rest of the series. Almost all of the holiday series appear in the bottom half of the plot, while almost all of the remaining series appear in the top half of the plot. Clearly, the second principal component is distinguishing between holidays and other types of travel.

The plot also allows us to identify anomalous time series — series which have unusual feature combinations. These appear as points that are separate from the majority of series in Figure 4.4. There are five that stand out, and we can identify which series they correspond to as follows.

```
pca_df["unique_id"] = all_feat["unique_id"]
top_unique_ids = pca_df.nlargest(5, "PC1")["unique_id"]
plot_df = aus_tourism[aus_tourism["unique_id"].isin(top_unique_ids)]

fig, axs = plt.subplots(5, 1, sharex=True, figsize=(8, 8))

for i, unique_id in enumerate(top_unique_ids):
    subset = plot_df[plot_df["unique_id"] == unique_id]
    axs[i].plot(subset["ds"], subset["y"])
    axs[i].set_title(unique_id)
    axs[i].set_xlabel("")
    axs[i].set_ylabel("")
fig.suptitle("Outlying time series in PC space", x=0.525)
fig.supylabel("Trips")
fig.supxlabel("Quarter")
plt.show()
```

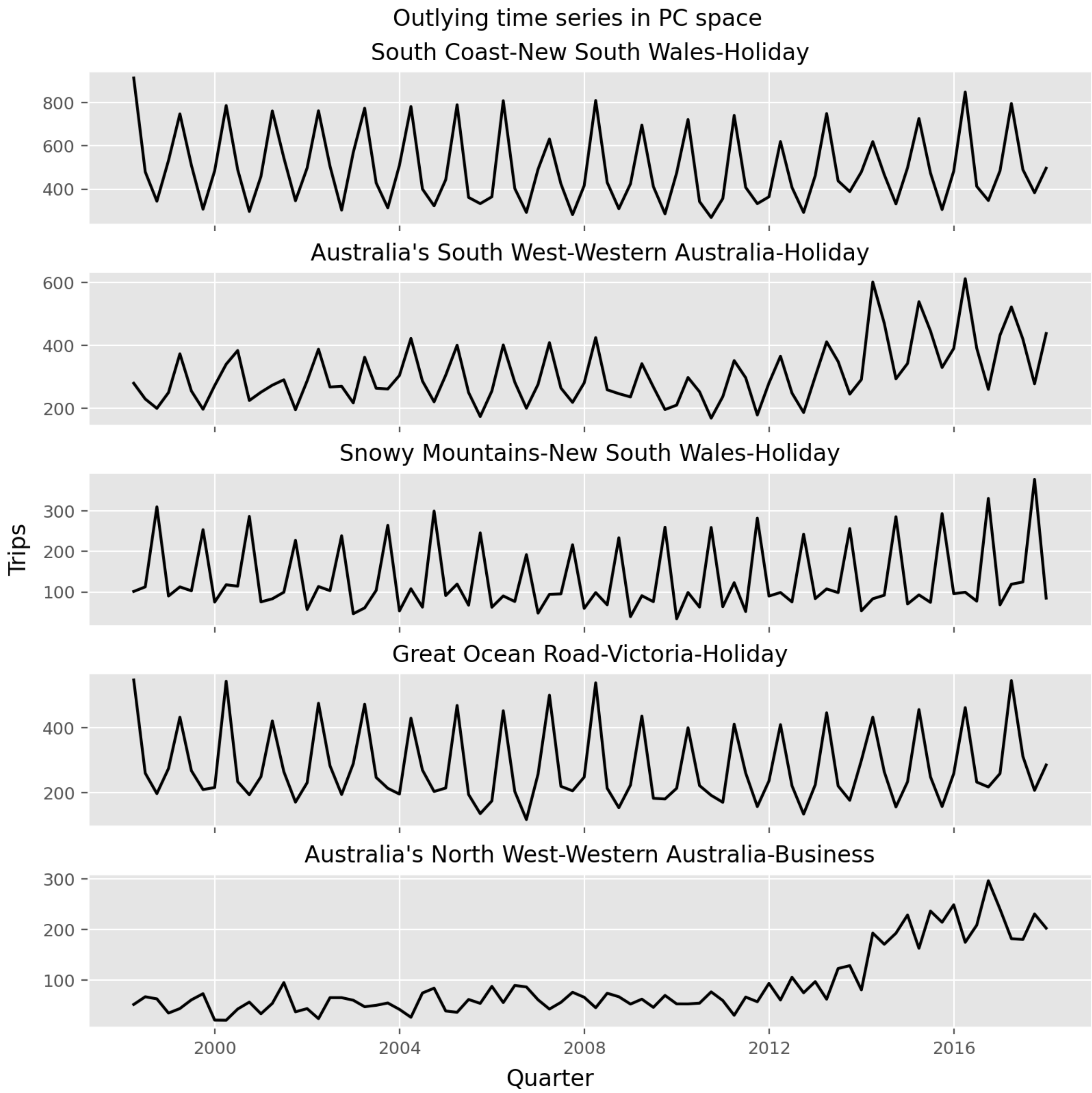

Figure 4.5: Four anomalous time series from the Australian tourism data.

We can speculate why these series are identified as unusual.

- Holiday visits to the south coast of NSW is highly seasonal but has almost no trend, whereas most holiday destinations in Australia show some trend over time.
- The south western corner of Western Australia is unusual because it shows both an increase in holiday tourism in the last few years of data and a high level of seasonality.
- The north western corner of Western Australia is unusual because it shows an increase in business tourism in the last few years of data, but little or no seasonality.

4.6 Exercises

1. Write a function to compute the mean and standard deviation of a time series, and apply it to the `PBS` data. Plot the series with the highest mean, and the series with the lowest standard deviation.

2. Use `sns.pairplot()` to look at the relationships between the STL-based features for the holiday series in the `tourism` data. Change `seasonal_peak_year` and `seasonal_trough_year` to factors, as shown in Figure 4.3. Which is the peak quarter for holidays in each state?

3. Use a feature-based approach to look for outlying series in the `PBS` data. What is unusual about the series you identify as "outliers".

4.7 Further reading

- The idea of using STL for features originated with Wang, Smith, and Hyndman (2006).
- The features provided by the `feasts` package were motivated by their use in Hyndman, Wang, and Laptev (2015) and Kang, Hyndman, and Smith-Miles (2017).
- The exploration of a set of time series using principal components on a large collection of features was proposed by Kang, Hyndman, and Smith-Miles (2017).

4.8 Used modules and classes

tsfeatures

- `tsfeatures` package - For computing time series features
- `acf_features`, `stl_features` functions - For autocorrelation and STL decomposition features

UtilsForecast

- `plot_series` utility - For creating time series visualizations

← Chapter 3 Time series decomposition

Chapter 5 The forecaster's toolbox →