Independent Study of Time Series Analysis
Name: Euijun Kim
UID: 705788156
With Professor Mahtash Esfandiari

- **Introduction**

Time series analysis is a field of study that deals with the analysis of data collected over time. It is an important tool for understanding patterns and trends in data, and has a wide range of applications in fields such as economics, finance, and engineering. This report aims to provide an overview of the key concepts and methods covered in an introductory level textbook on time series analysis. The textbook used as the basis for this report covers the basics of time series data, trend analysis, seasonality, stationarity, autocorrelation, and moving average, among other topics. Through this report, the reader will gain an understanding of the significance of time series analysis and its potential applications in real-world scenarios.

- **Background Information**

Time series data is a sequence of observations taken at regular intervals over time, such as daily temperature readings or monthly sales figures. Time series analysis involves the examination of these observations to uncover patterns and trends that can help predict future outcomes.

One of the main challenges of time series analysis is accounting for the temporal dependencies in the data. For example, the temperature on one day may be influenced by the temperature on the previous day. By understanding these dependencies, time series analysis can be used to make accurate forecasts and gain insights into complex systems.

Time series analysis has a wide range of applications, including forecasting, signal processing, and control. In finance, time series analysis is used to predict stock prices, while in economics it is used to forecast GDP and inflation. In engineering, time series analysis is used in control systems to improve the efficiency and stability of processes.

This report will provide an overview of the key concepts and methods in time series analysis, including trend analysis, seasonality, stationarity, autocorrelation, and moving average. By understanding these concepts, readers will gain a deeper appreciation for the importance of time series analysis and its potential applications in real-world scenarios.

- **Key conecepts**

1. Trend Analysis: A trend is the underlying pattern in the data that persists over time. Time series analysis uses various methods to identify trends, such as linear regression and exponential smoothing.
2. Seasonality: Seasonality refers to regular patterns in the data that occur at specific times of the year, such as increased sales during the holiday season. Time series analysis

uses methods such as decomposition and spectral analysis to identify seasonality in the data.

3. Stationarity: Stationarity is an important property of time series data, which means that the statistical properties of the data remain constant over time. Non-stationary time series data can make it difficult to forecast future outcomes and must be transformed into stationary data before analysis.
4. Autocorrelation: Autocorrelation is the relationship between the values of a time series data at different times. Time series analysis uses methods such as autocorrelation plots and partial autocorrelation plots to identify autocorrelation in the data and make predictions.
5. Moving Average: Moving average is a method used in time series analysis to smooth out the fluctuations in the data and reveal the underlying trend. A moving average is calculated by taking the average of a set of consecutive data points and using the average value as the forecast for the next time step.

- **Applications**

1. Forecasting: One of the most common applications of time series analysis is forecasting, which involves making predictions about future events based on historical data. Time series analysis uses various methods, such as exponential smoothing and ARIMA, to make accurate forecasts.
2. Financial Markets: Time series analysis is widely used in finance to analyze stock prices, interest rates, and other financial data. It can be used to make predictions about future market trends and to identify profitable investment opportunities.
3. Economics: Time series analysis is used in economics to forecast economic indicators, such as GDP and inflation. By understanding the patterns and trends in economic data, policymakers can make informed decisions and respond to economic challenges.
4. Engineering: Time series analysis is used in control systems to improve the stability and efficiency of processes. By analyzing sensor data, engineers can identify patterns and anomalies, and make changes to improve performance.
5. Healthcare: Time series analysis is used in healthcare to analyze patient data, such as vital signs and lab results. By understanding the patterns in patient data, healthcare providers can make more informed decisions about patient care and improve outcomes.
6. Weather prediction: Meteorologists use time series analysis to make predictions about future weather conditions based on historical data, such as temperature, pressure, and wind speed. By analyzing patterns and trends in this data, meteorologists can make more accurate forecasts and provide early warning of severe weather events.

    - Weather prediction

- For example, time series analysis can be used to identify the presence of periodic patterns in the data, such as the daily cycle of temperature fluctuations. By understanding these patterns, meteorologists can make predictions about future temperature trends and the likelihood of specific weather events.

- In addition, time series analysis can also be used to identify the relationship between different weather variables, such as the relationship between temperature and wind speed. By understanding these relationships, meteorologists can make more accurate predictions about future weather conditions and improve the reliability of their forecasts.
- Overall, weather prediction is just one example of the many applications of time series analysis in real-world scenarios. By understanding the concepts and methods covered in an introductory level textbook, readers will be well-equipped to analyze time series data and make informed decisions in a variety of fields, including weather prediction.

- **Methodology**

Currently, the most popular method of time series analysis is ARIMA (AutoRegressive Integrated Moving Average) and its variants, such as SARIMA (Seasonal ARIMA) and SARIMAX (Seasonal ARIMA with exogenous variables). ARIMA is a linear model that is widely used for univariate time series forecasting, especially in finance and economics. The popularity of ARIMA is due to its simplicity and interpretability, as well as its ability to model a wide range of time series patterns.

In recent years, Deep Learning models such as LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units) have become increasingly popular for time series forecasting, especially for high-dimensional and multivariate time series. These models are capable of learning complex non-linear relationships and capturing long-term dependencies in time series data.

Ultimately, the choice of method depends on the specific problem being addressed and the characteristics of the time series data. A practitioner might consider a combination of methods, such as ARIMA for initial exploratory analysis, followed by a Deep Learning model for more accurate predictions.

1. ARIMA(p, d, q): Here, "p" is the number of autoregressive (AR) terms, "d" is the number of differences needed for stationarity, and "q" is the number of moving average (MA) terms. This is the most commonly used type of ARIMA model.
2. SARIMA (Seasonal ARIMA): This is a variant of ARIMA that considers the seasonal patterns in the data. It adds two additional parameters, "P" and "Q", which are the seasonal autoregressive and seasonal moving average terms, respectively.
3. ARIMAX (Regressor ARIMA): This is a variant of ARIMA that considers the effect of exogenous variables (also called regressors) on the time series data. ARIMAX adds an additional "X" component to the ARIMA model to account for the effect of the exogenous variables.

- [**Reference**] https://otexts.com/fpp3/

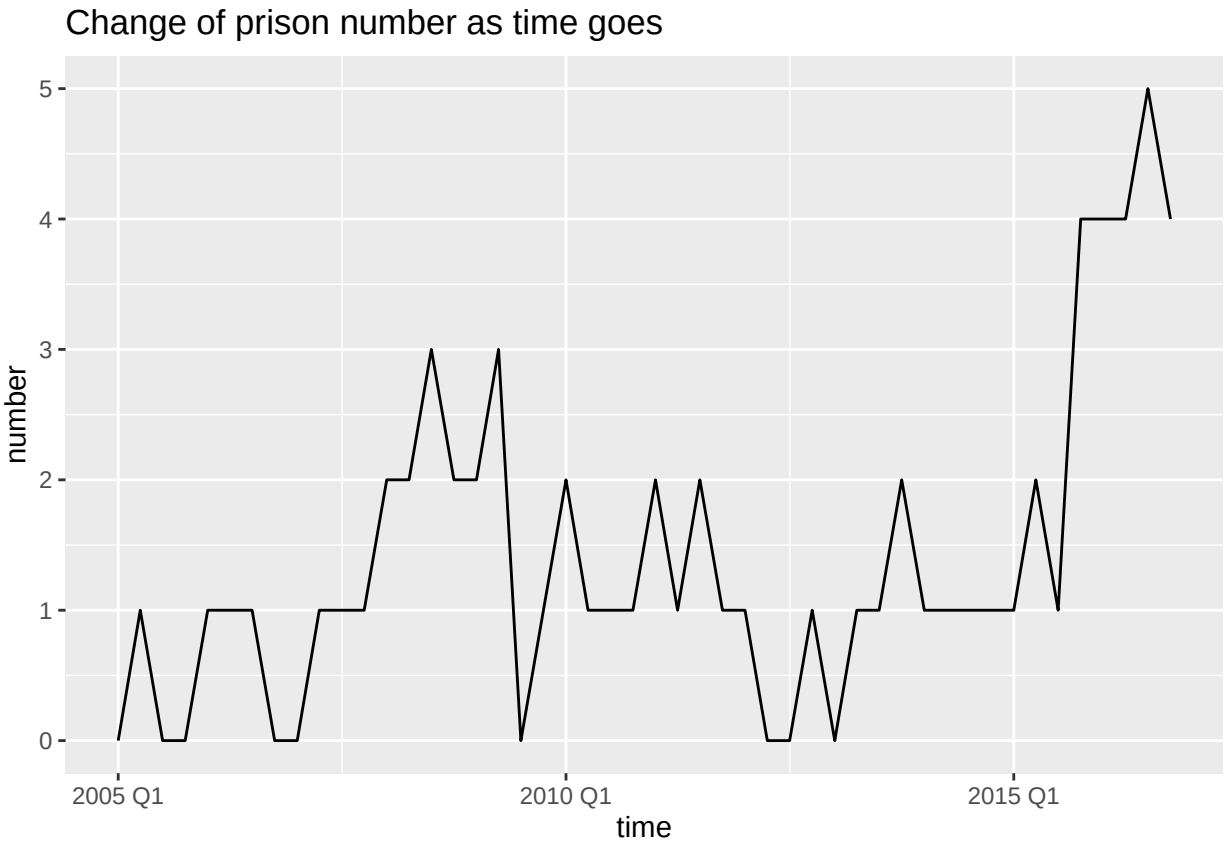
199 research study with textbook

Euijun Kim

Winter 2023

1.1 Time plots

This is the most basic graph in time series analysis. It plots the values of the response variable we want to see over time. Let's draw the changes in the number of prisoners over time for a specific group using the prison time series data defined earlier.



1.2 Time Series Analysis Components

Trend: The presence or absence of an overall direction or orientation

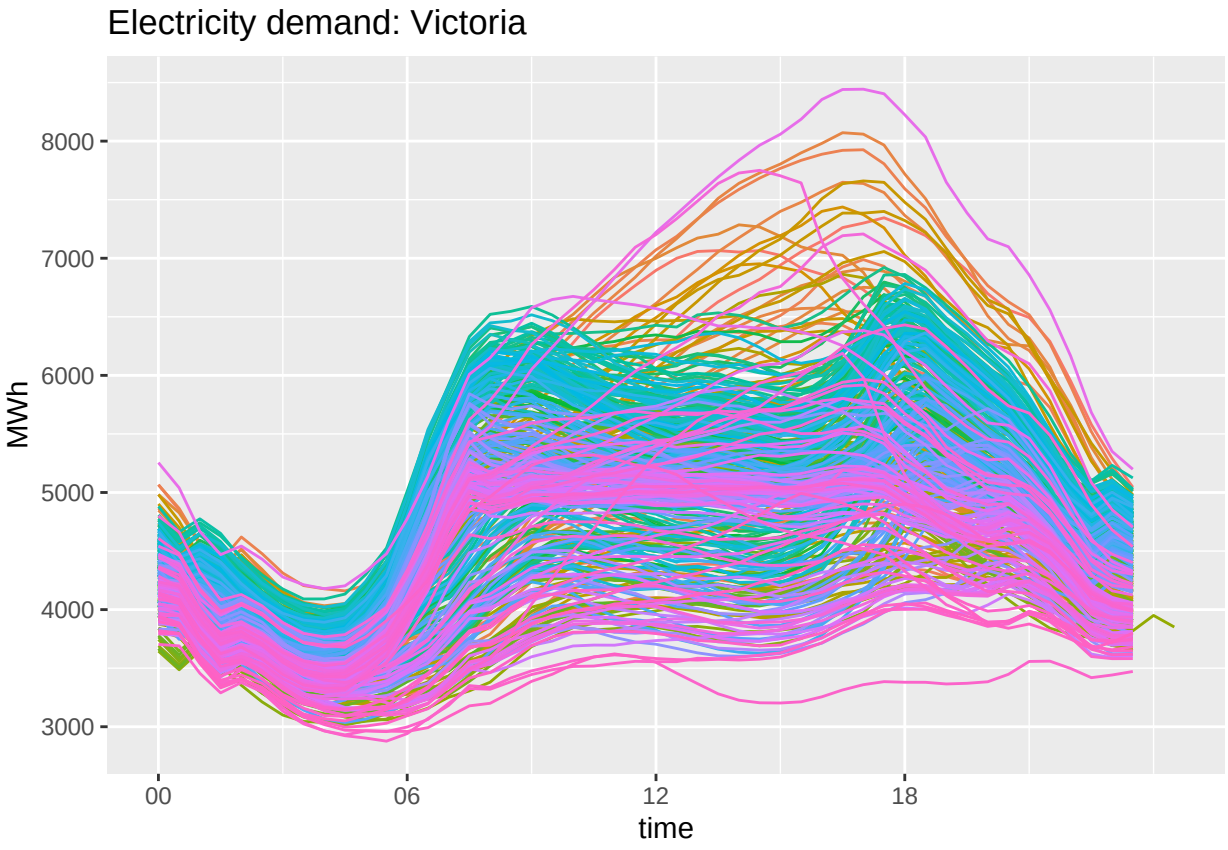
Seasonal: Repetition presence according to the season (fixed frequency)

Cycle: Not a fixed frequency, but in the form of increase or decrease

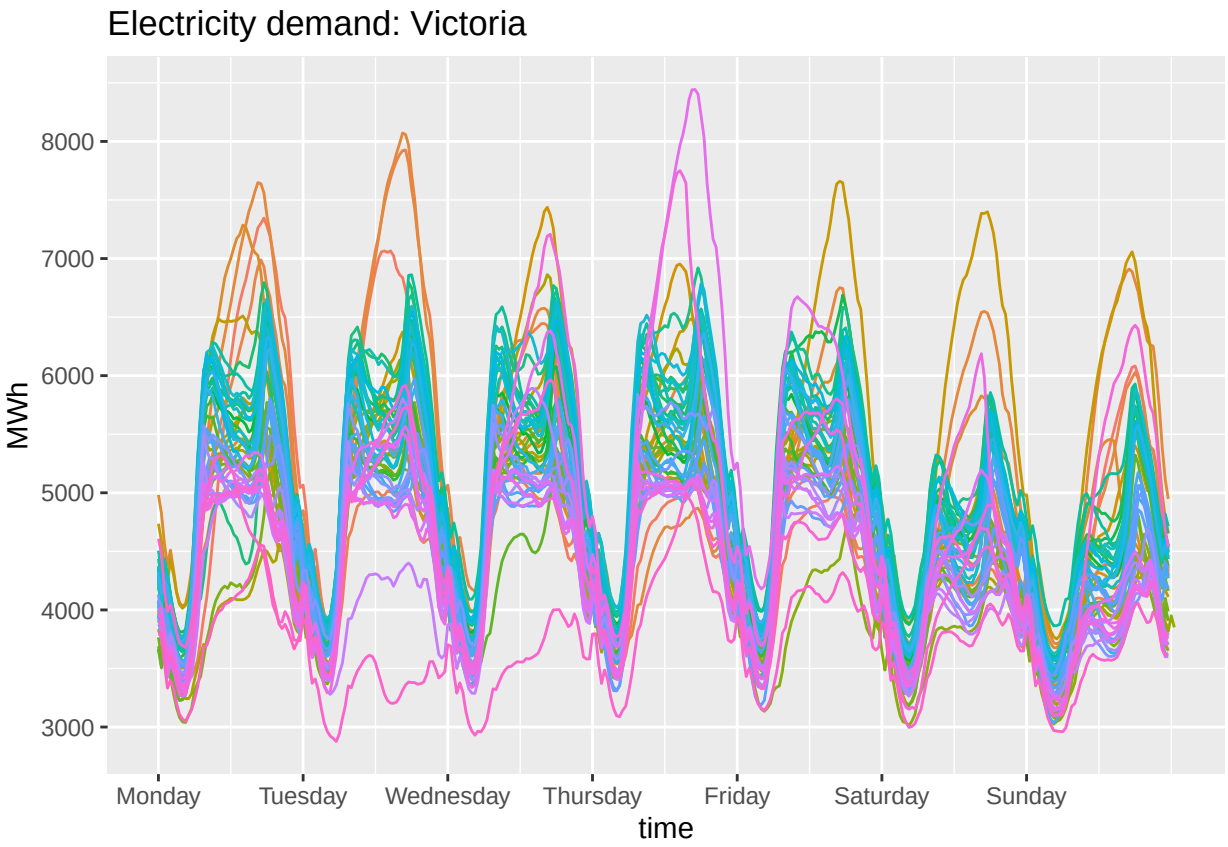
1.3 Seasonal plot

The "vic_elec" data is a dataset that records the electricity demand in Victoria, Australia at 30-minute intervals.

When we want to see how electricity demand moves in a pattern over a day in this data, a seasonal plot is useful. It shows the pattern of seasonality in the data over time within each day.

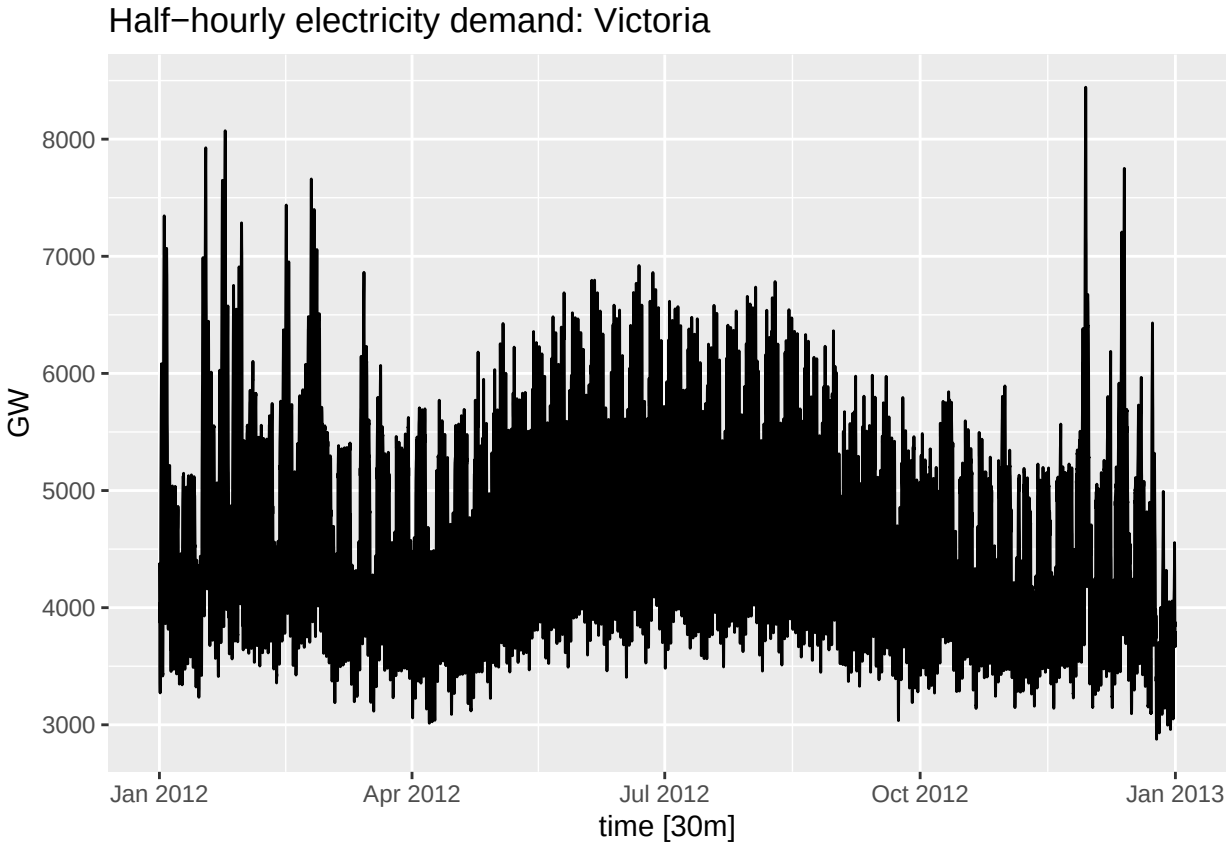


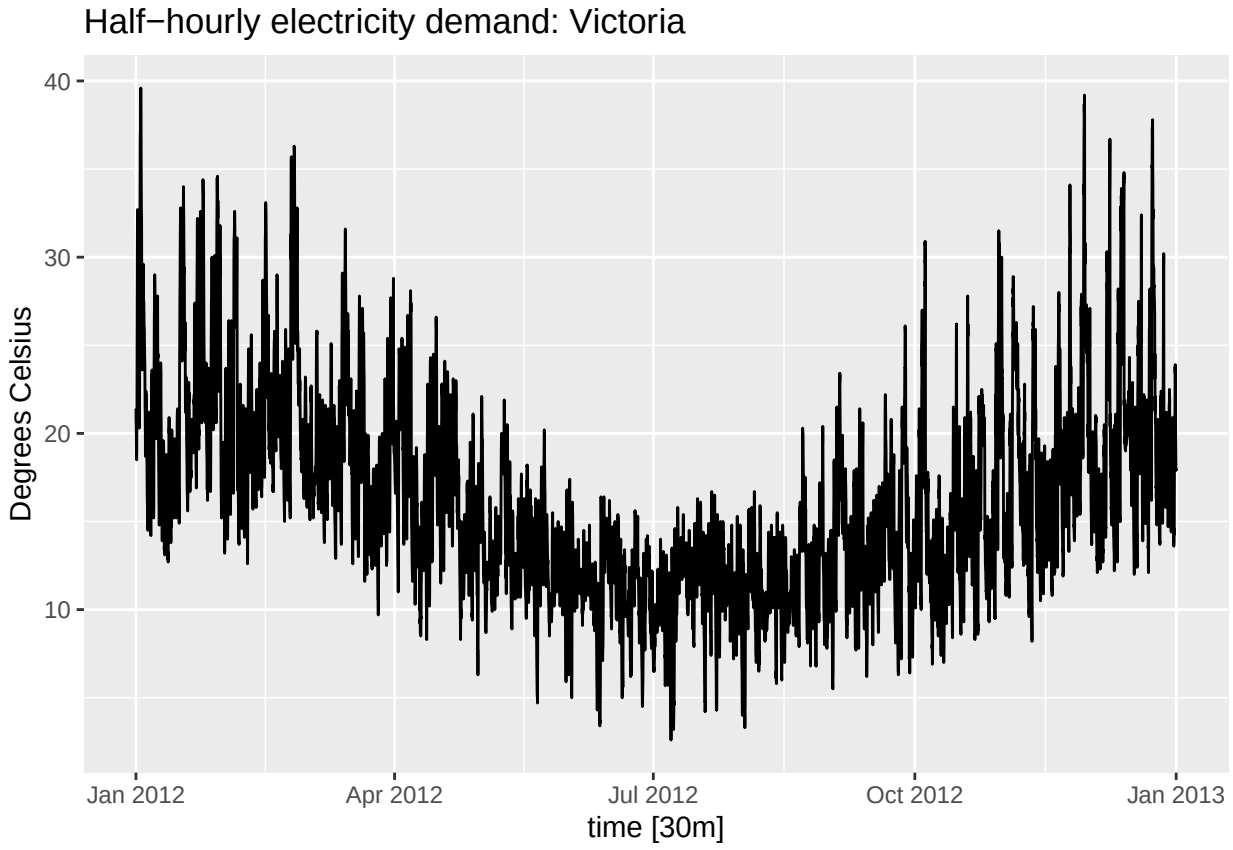
If we draw a seasonal plot for a week, we may discover a new pattern in the data.



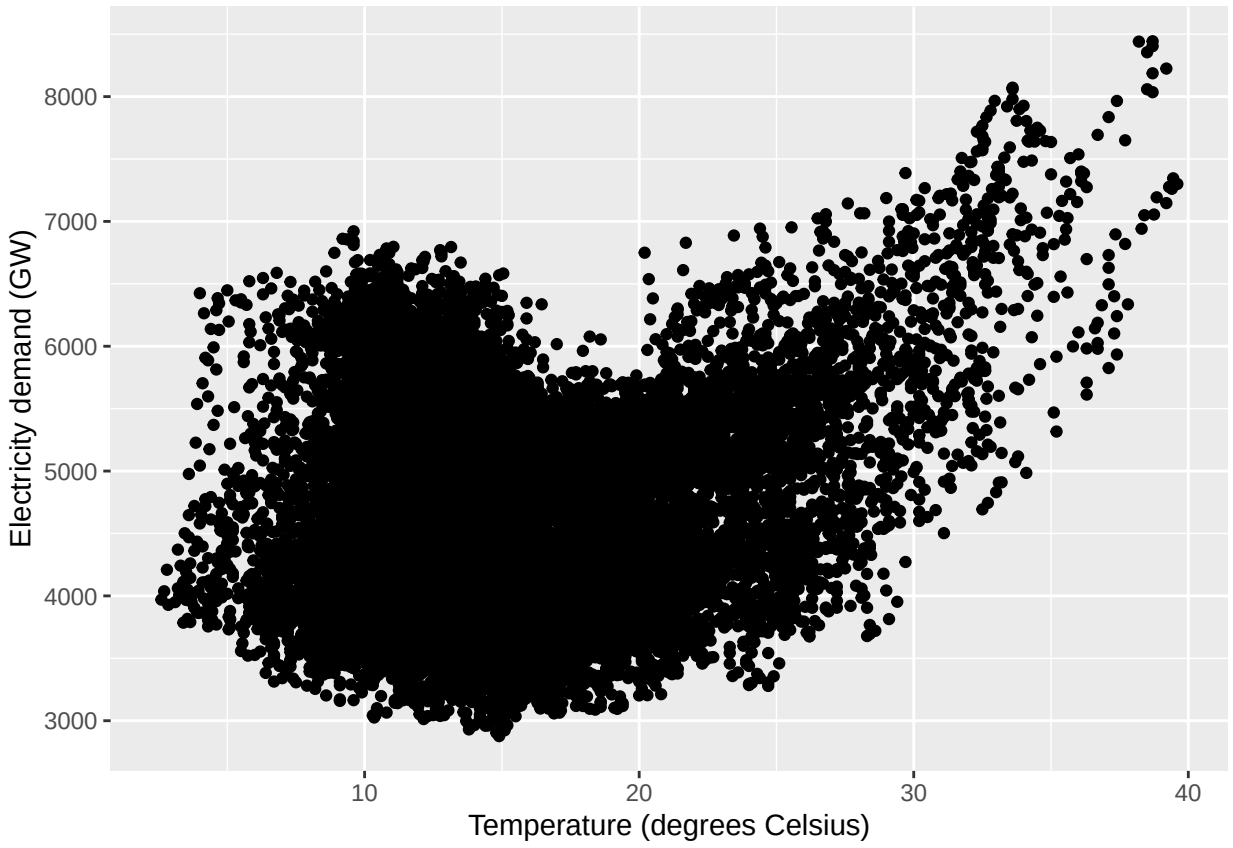
1.4 Scatter plots

In the electricity demand data, there is a "demand" variable that represents the electricity demand, and there is a corresponding "temperature" variable. If we consider these two variables as independent time series data and draw a time plot, it will look like the following.

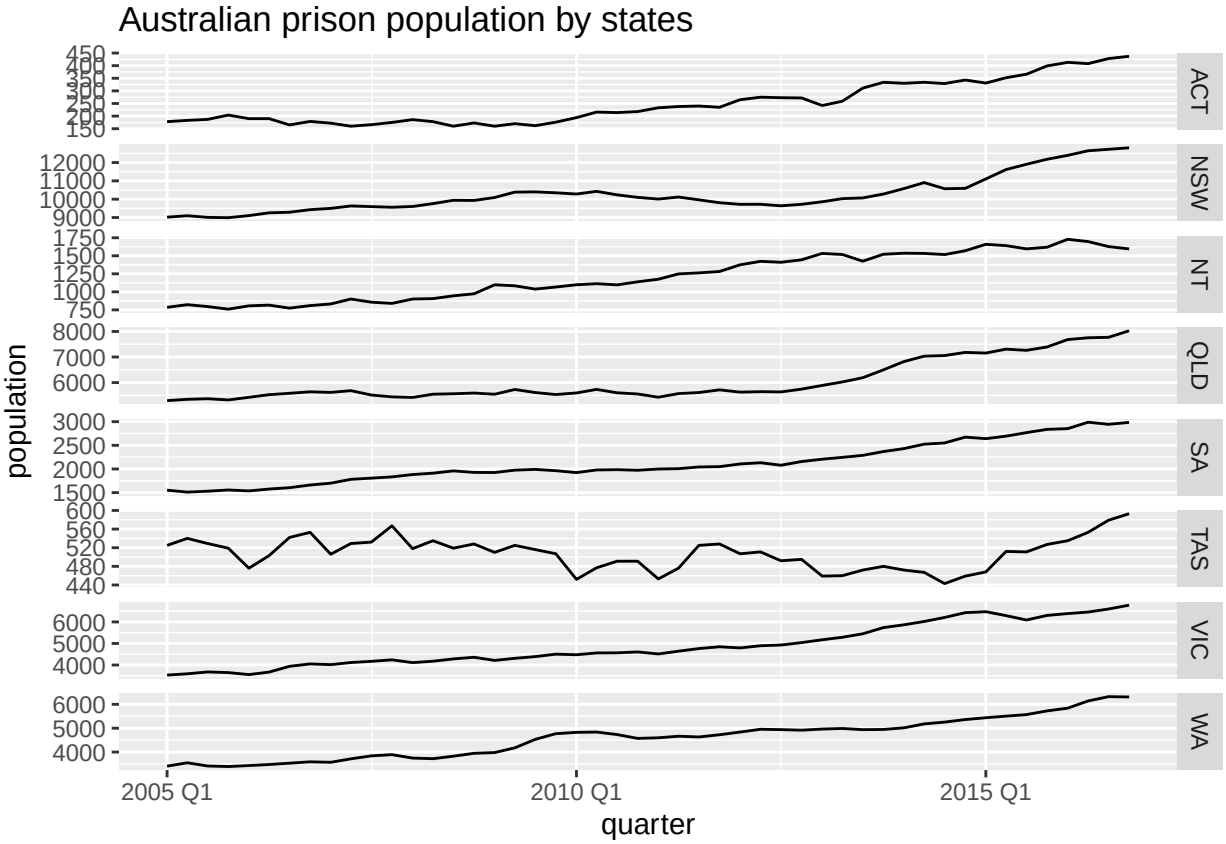



Half−hourly electricity demand: Victoria
Degrees Celsius
40
30
20
10
Jan 2012
Apr 2012
Jul 2012
Oct 2012
Jan 2013
time [30m]

However, what would happen if we draw a scatter plot of these two variables ignoring the time variable? By drawing a scatter plot as follows, we may discover a new pattern between temperature and electricity demand.



In the case of electricity demand, it shows a quadratic pattern that increases when the temperature is low or high. This seems to be due to the electricity demand for heating during winter and cooling during summer.



1.5 Lag plots

"Lag" means to be behind, and in the context of time series analysis, it refers to the number of time periods by which a time series is shifted or delayed. In other words, it can be considered as a time shift.

For example, suppose we have the following numbers:

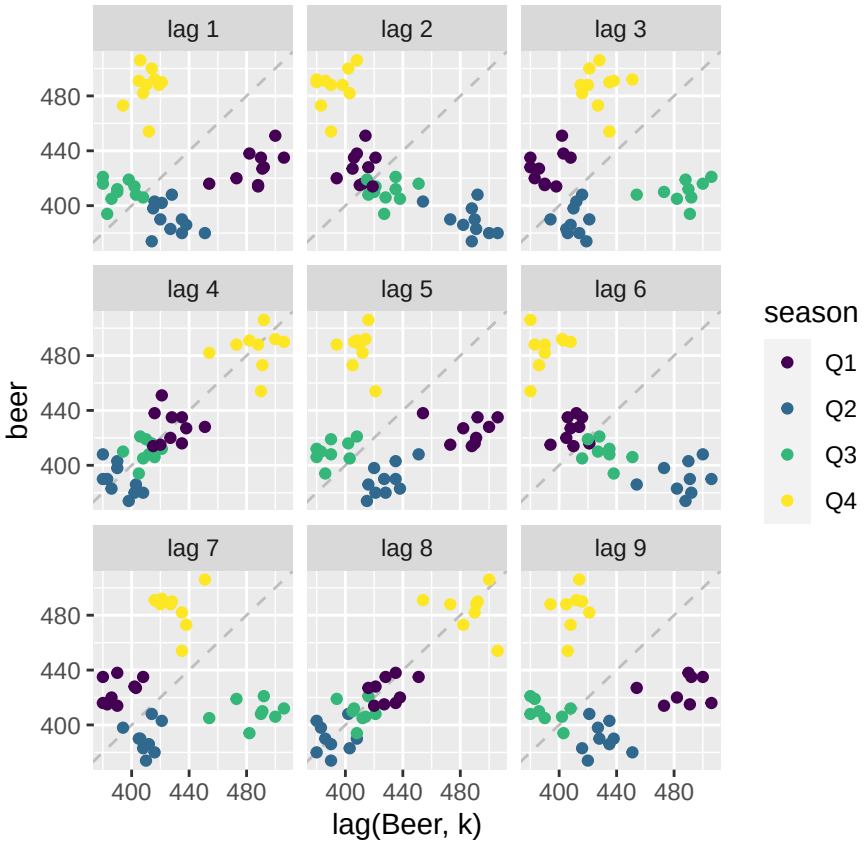
1, 3, 6, 10, 15, 21, 28, 36, 45, 55

Let's call this vector of numbers x. Then, the corresponding lag 1 and lag 2 vectors for x are as follows:

```
##        x lag_x_1 lag_x_2
##  [1,]  1      NA      NA
##  [2,]  3       1      NA
##  [3,]  6       3       1
##  [4,] 10       6       3
##  [5,] 15      10       6
##  [6,] 21      15      10
##  [7,] 28      21      15
##  [8,] 36      28      21
```

```
##  [9,] 45       36       28
## [10,] 55       45       36
```

A lag plot can be useful to check the seasonality of beer production data. When examining lag 4 and lag 8, we can see that the points are distributed along the diagonal. In contrast, for lag 2 and lag 4, the points are distributed in a diagonal shape that extends downward from left to right.



1.6 Autocorrelation

Let's take a look at the formula for calculating the correlation coefficient between two variables.

x={x1,x2,....xn}, y={y1,y2,...yn}

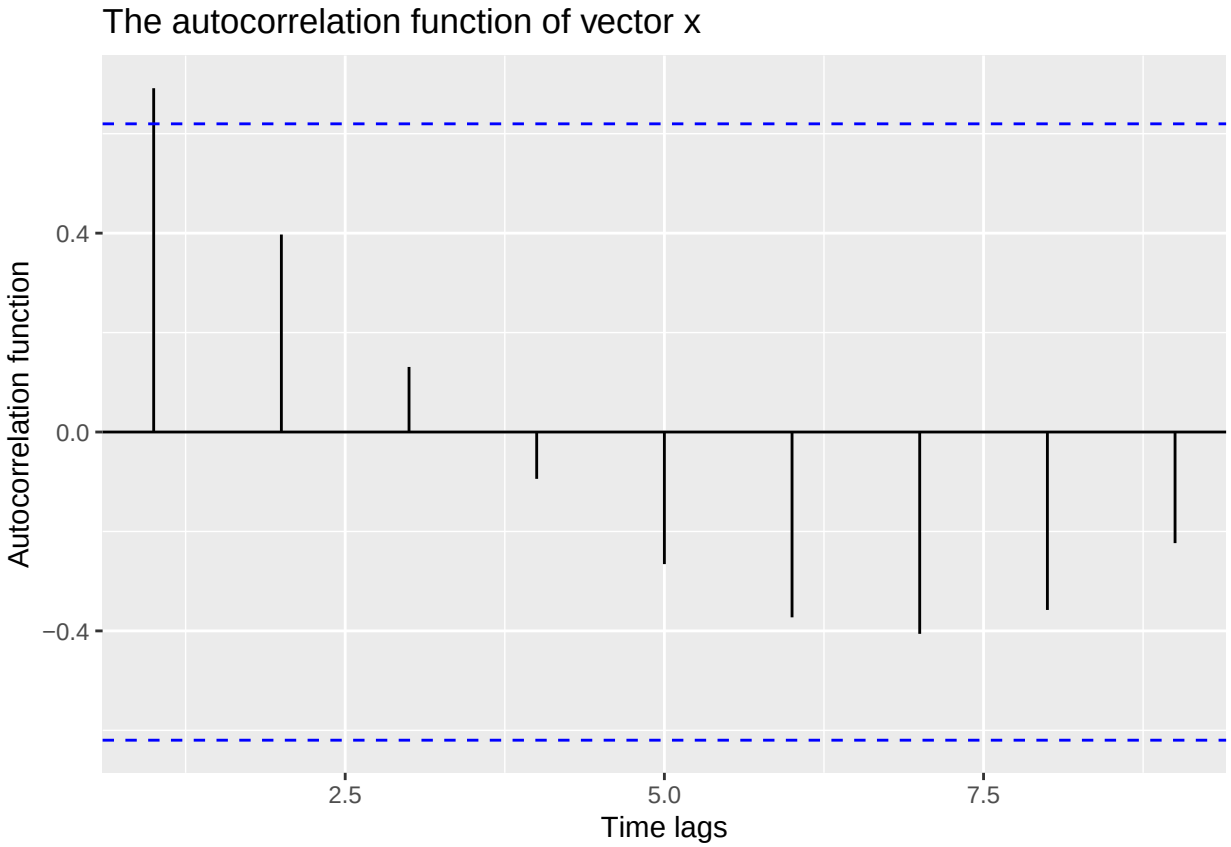
$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

Autocorrelation can be understood as the correlation coefficient between a vector corresponding to a specific lag value and itself. The (sample) autocorrelation coefficient r_k for a lag_k can be calculated as follows:

$$r_k = \frac{\sum_{t=k+1}^{T}(x_t-\bar{x})(x_{t-k}-\bar{x})}{\sum_{t=1}^{T}(x_t-\bar{x})^2}$$

```
## [1] 0.3971631
```

```
## # A tsibble: 9 x 2 [1]
##       lag     acf
##   <cf_lag>   <dbl>
## 1        1  0.691
## 2        2  0.397
## 3        3  0.131
## 4        4 -0.0941
## 5        5 -0.266
## 6        6 -0.373
## 7        7 -0.406
## 8        8 -0.358
## 9        9 -0.223
```



The autocorrelation function (ACF) graph plays an important role in identifying the characteristics of time series data. It is also used to classify time series data based on the shape of the ACF function because it captures these features well.

2.1 Steps in Time Series Forecasting

Step 1: Problem Definition

Defining the problem is the most challenging aspect of forecasting analysis. When defining a prediction problem, it is important to understand how the forecasting model will be used, who will require the forecasting model, and how well the prediction results align with user requirements. Forecasting analysts need to communicate with data collectors, database managers, and people who make future plans.

Step 2: Information Collection

The collected information includes not only statistical data, but also accumulated expertise in the field of data collection and prediction. Often, it is difficult to collect enough past data to establish a good statistical model. In such cases, judgmental forecasting is used (fpp3 Chapter 6). Older data is often less useful when the structure of the system being predicted changes. In such cases, only the most recent data is used. However, a good statistical model should be able to reflect changes in the system structure. Do not unnecessarily discard data.

Step 3: Basic Analysis

Always start by plotting the data. Is there a consistent pattern? Is there a significant trend? Is seasonality important? Is there evidence of periodicity? Are there outliers that need to be explained with professional knowledge? Are there strong correlations between variables?

Step 4: Model Selection and Fitting

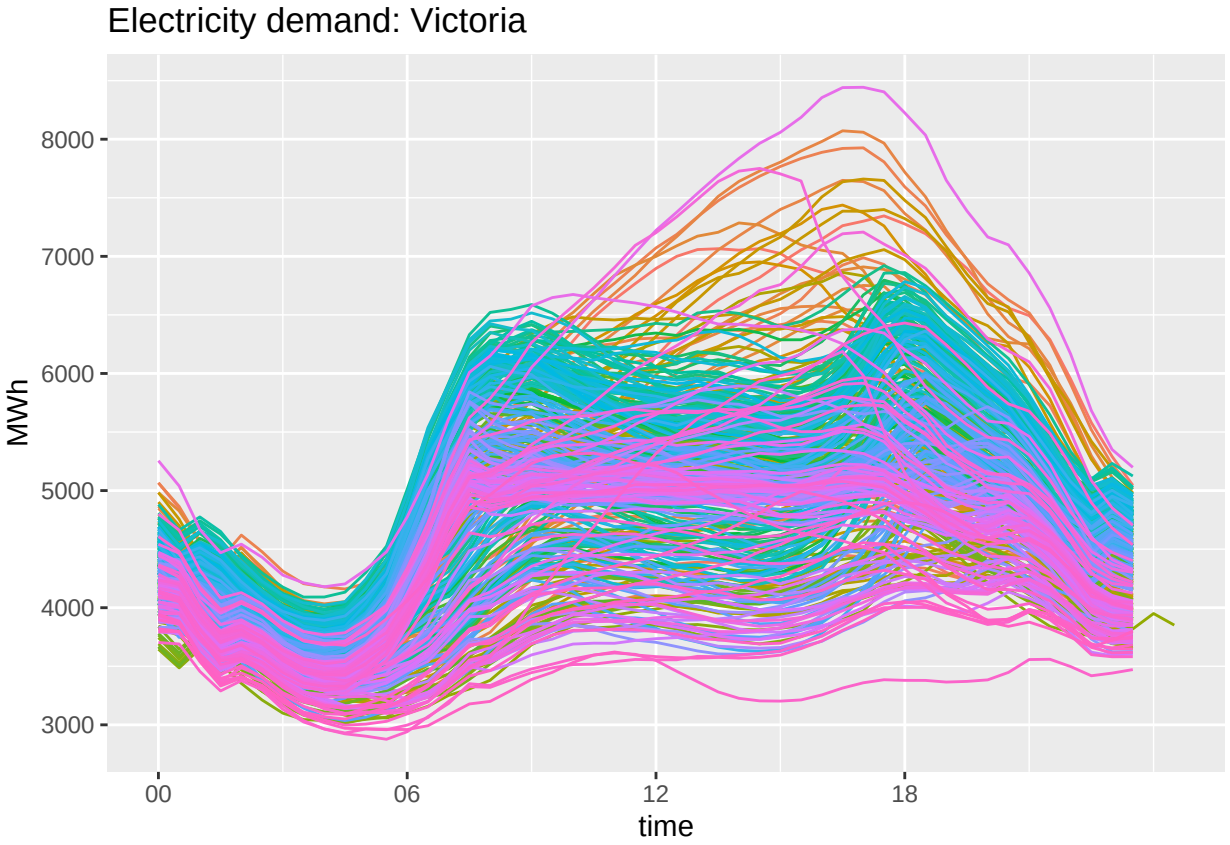
The best model is determined based on the use of past data, the degree of relationship between predictive variables, and the method of use for prediction. It is common to compare 2-3 models. Each model is an artificially created product based on several assumptions, and the parameters within the model are estimated through data samples.

Step 5: Model Use and Evaluation

Once the model is selected and the parameters are estimated, the model can perform predictions. The performance of the model can be evaluated once the data for the prediction period becomes available. Many methods have been developed to evaluate the accuracy of predictions. Issues that are important for practical use of prediction models include handling missing values and outliers, or dealing with short time series.

2.2 Distribution of Time Series Data

The graph below shows the daily electricity consumption of the Australian state of Victoria recorded in 30-minute intervals in the vic_elec dataset for the year 2012.



2.3 Foundations for analyzing time series models

2.3.1 Stochastic Proces

$$\{Y_t : t = 0, \pm 1, \pm 2, \pm 3, \ldots\}$$

A stochastic process is a sequence of random variables used as a model for analyzing observed time series data. Analysis of the stochastic process is based on information obtained from mean, variance, and covariance.

2.3.2 Mean Function

$$\mu_t = E[Y_t] \quad \text{for } t = 0, \pm 1, \pm 2, \ldots$$

2.3.3 Autocovariance function

$$\gamma_{t,s} = Cov(Y_t, Y_s) \quad \text{for } t, s = 0, \pm 1, \pm 2, \ldots$$

$$Cov(Y_t, Y_s) = E[(Y_t - \mu_t)(Y_s - \mu_s)]$$

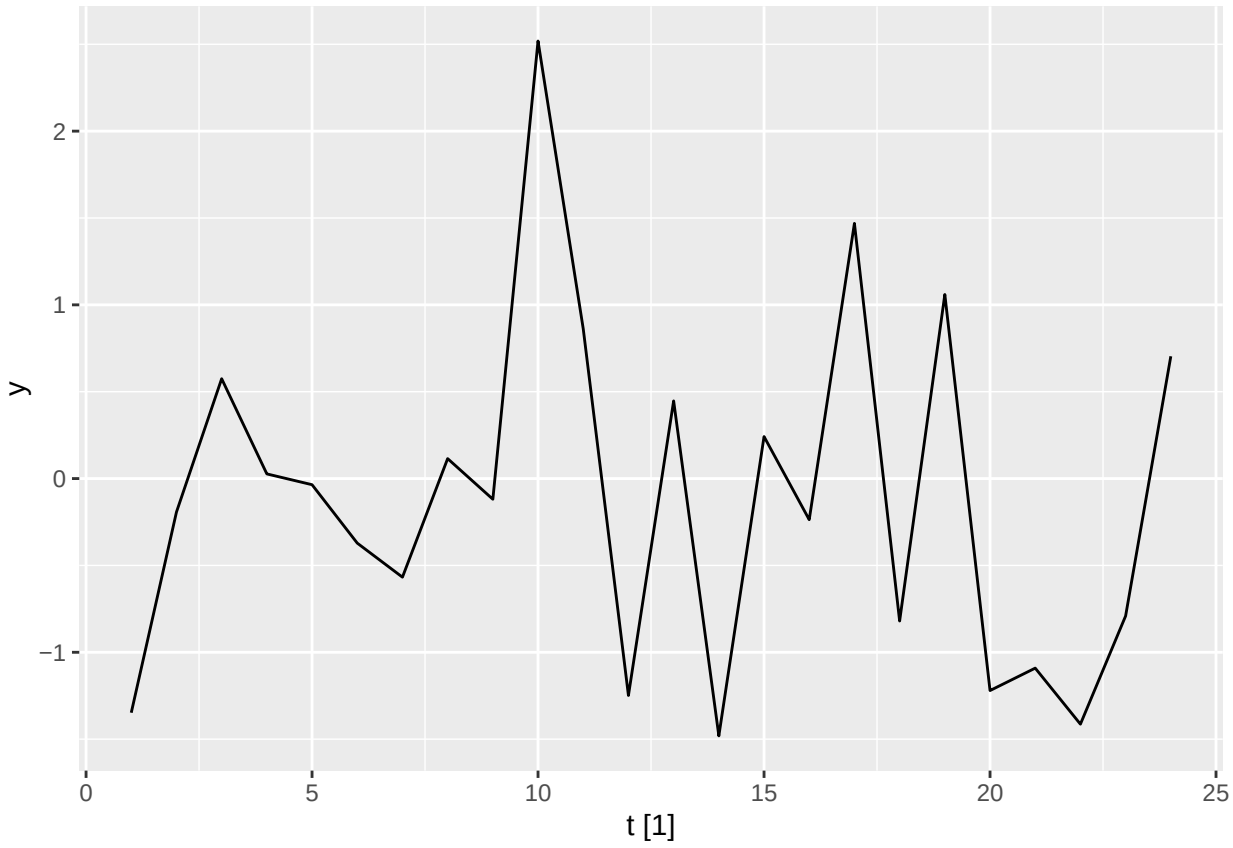
2.3.4 Autocorrelation function

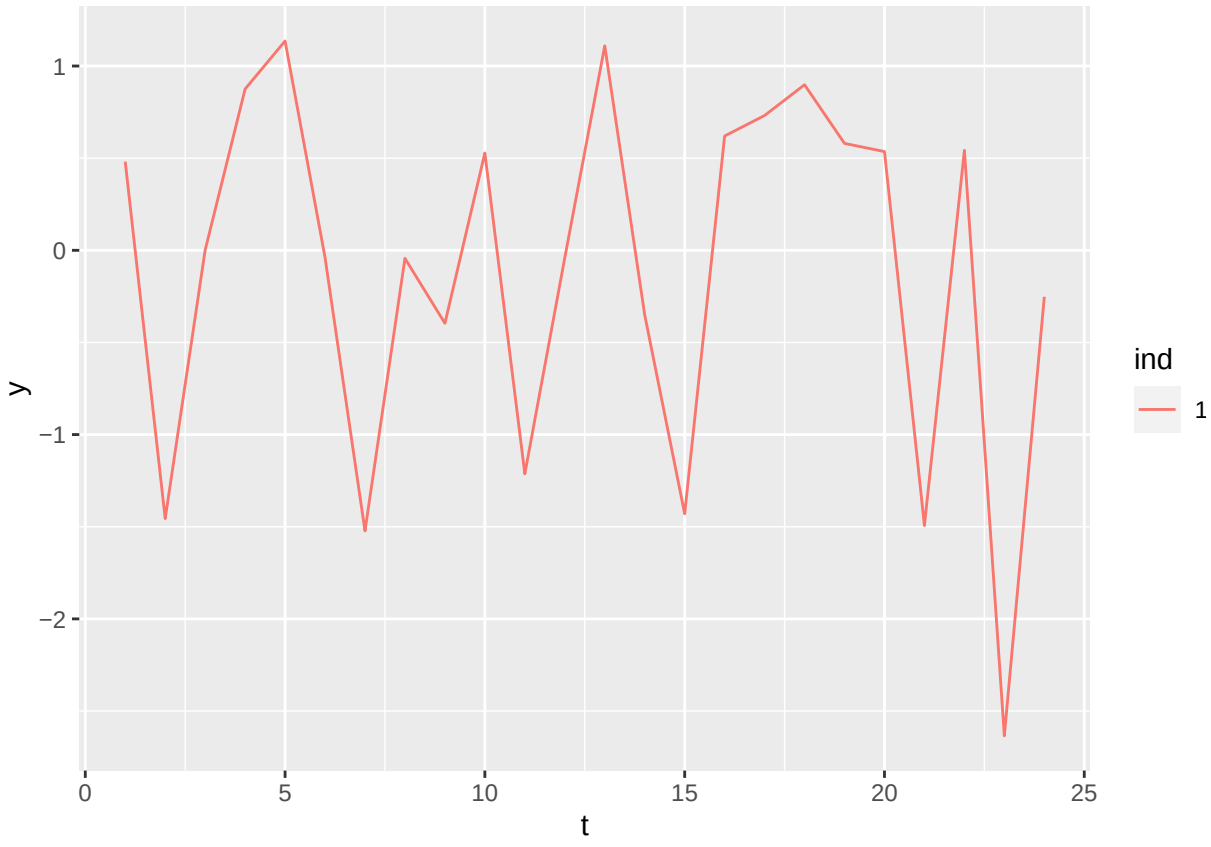
$$\rho_{t,s} = Corr(Y_t, Y_s) \quad \text{for } t, s = 0, \pm 1, \pm 2, \ldots$$
$$Corr(Y_t, Y_s) = \frac{Cov(Y_t, Y_s)}{\sqrt{Var(Y_t)}\sqrt{Var(Y_s)}}$$

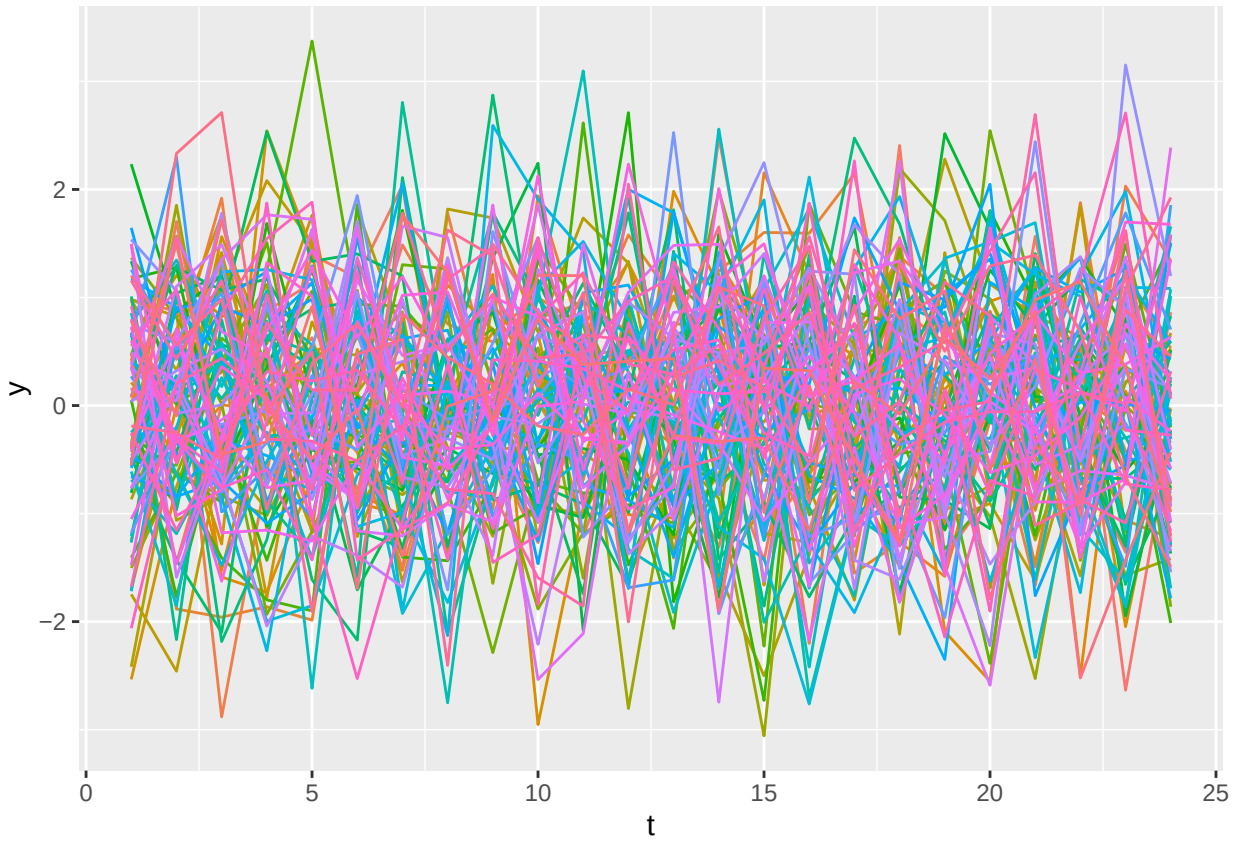
2.4 White Noise

Random variables $\{e_t = e_1, e_2, \ldots\}$ are assumed to be independent and identically distributed with a mean of 0 and a variance of σ_e^2. This is called white noise.

$$Y_t = e_t : t = 1, 2, 3, \ldots$$
$$\mu_t = E[Y_t] = E[e_t] = 0$$
$$Var(Y_t) = Var(e_t) = \sigma_e^2$$
$$Cov(Y_t, Y_s) = Cov(e_t, e_s) = 0 \quad \text{for } t \neq s$$
$$Corr(Y_t, Y_s) = Corr(e_t, e_s) = 0 \quad \text{for } t \neq s$$


2
1
0
−1
y
0
5
10
15
20
25
t [1]


1
0
−1
−2
y
0
5
10
15
20
25
t
ind
1

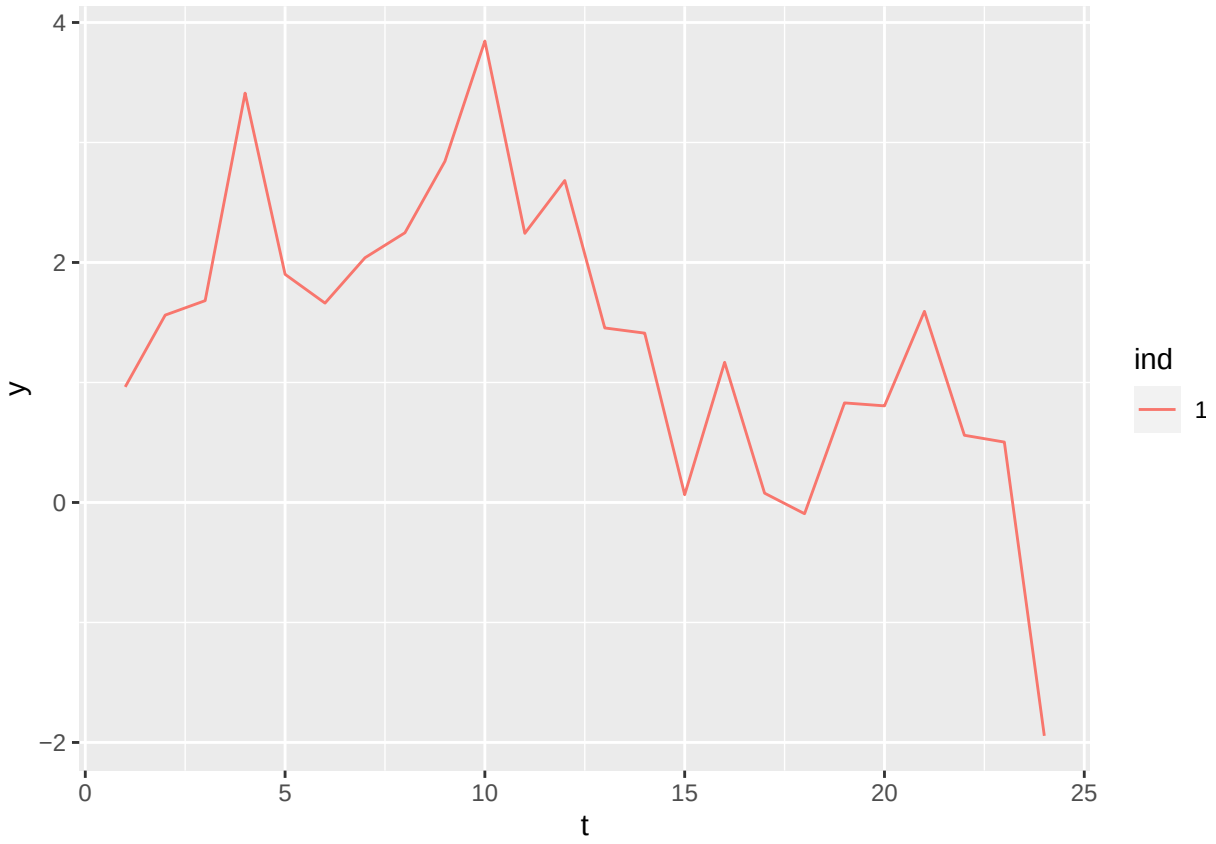


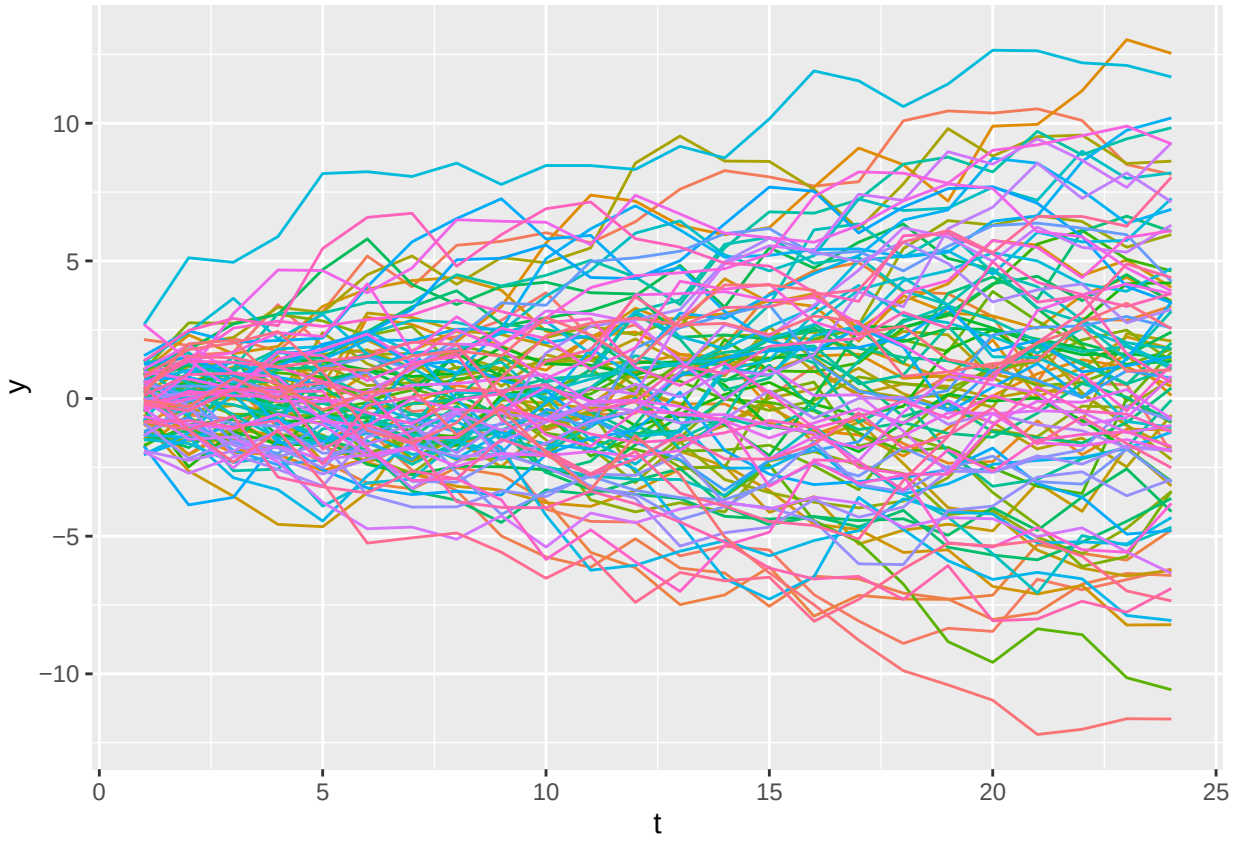
Since the random variables are independent of each other, the covariance becomes 0.

The distribution of white noise model data.

2.5 Random Walk

$$
\begin{aligned}
\mu_t &= E[Y_t] = E[Y_{t-1}] + E[e_t] = \mu_{t-1} + 0 = \cdots = \mu_1 = 0 \\
Var(Y_t) &= Var(Y_{t-1}) + Var(e_t) = Var(Y_{t-1}) + \sigma_e^2 = \cdots = t\sigma_e^2 \\
Cov(Y_t, Y_s) &= Cov(Y_{s+e_{s+1}} + \cdots + e_t, Y_s) = Cov(Y_s, Y_s) = Var(Y_s) = s\sigma_e^2 \text{ for } t > s \\
Corr(Y_t, Y_s) &= \frac{Cov(Y_t, Y_s)}{\sqrt{Var(Y_t)}\sqrt{Var(Y_s)}} = \frac{s\sigma_e^2}{\sqrt{t\sigma_e^2}\sqrt{s\sigma_e^2}} = \sqrt{\frac{s}{t}} \text{ for } t > s
\end{aligned}
\tag{1}
$$


4
2
0
−2
y
0
5
10
15
20
25
t
ind
1


10
5
0
−5
−10
y
0
5
10
15
20
25
t

In the above, we can see that the random walk model has a mean of 0 and its variance shows a pattern of increasing as time t increases.

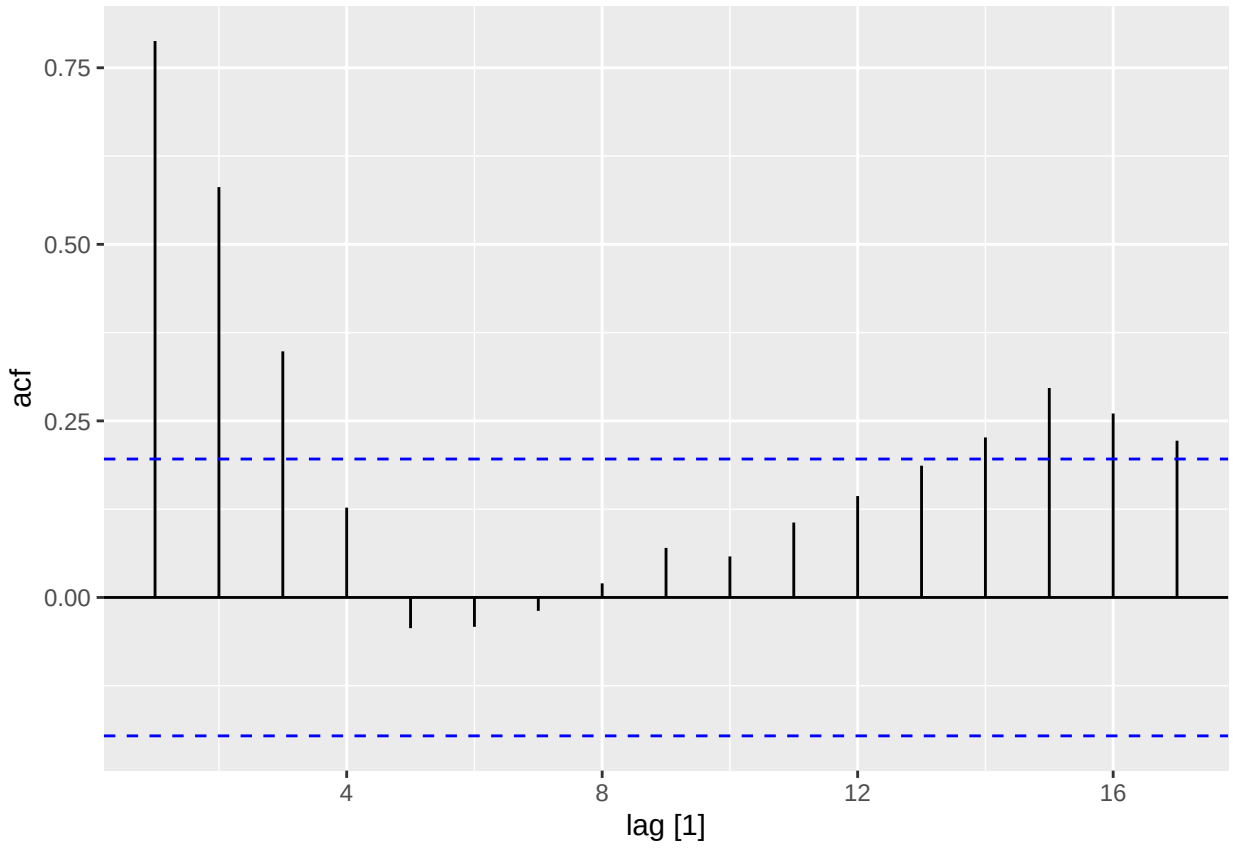
2.6 Moving Average

The moving average model is defined as follows:

$$
\begin{aligned}
\{Y_t = \frac{1}{2}(e_t + e_{t-1}) : t = 2, 3, \ldots\} \\
\mu_t = E[Y_t] = \frac{1}{2}(E[e_t] + E[e_{t-1}]) = 0 \\
Var(Y_t) = \frac{1}{4}Var(e_t) + \frac{1}{4}Var(e_{t-1}) = \frac{1}{2}Var(e_t) \\
Cov(Y_t, Y_{t-1}) = Cov\left(\frac{1}{2}(e_t + e_{t-1}), \frac{1}{2}(e_{t-1} + e_{t-2})\right) \\
= \frac{1}{4}Cov(e_t, e_{t-2}) + \frac{1}{4}Cov(e_t, e_{t-1}) + \frac{1}{4}Cov(e_{t-1}, e_{t-2}) \\
= \frac{1}{4}Var(e_t) = \frac{1}{4}\sigma_e^2 \\
Cov(Y_t, Y_s) = Cov\left(\frac{1}{2}(e_t + e_{t-1}), \frac{1}{2}(e_s + e_{s-1})\right) = 0 \quad \text{for } |t-s| > 1 \\
Corr(Y_t, Y_{t-1}) = \frac{Cov(Y_t, Y_{t-1})}{\sqrt{Var(Y_t)}\sqrt{Var(Y_{t-1})}} = \frac{1}{2} \\
Corr(Y_t, Y_s) = 0 \quad \text{for } |t-s| > 1
\end{aligned}
\tag{2}
$$

\end{equation}

```
## # A tsibble: 100 x 3 [1]
##        t        y   yavg
##    <int>    <dbl>  <dbl>
##  1     1 -2.58    NA
##  2     2  0.00558 NA
##  3     3 -0.628   NA
##  4     4 -1.06    NA
##  5     5 -1.55    -1.16
##  6     6  0.198   -0.607
##  7     7 -1.47    -0.902
##  8     8 -1.35    -1.05
##  9     9 -1.15    -1.06
## 10    10 -1.70    -1.10
## # ... with 90 more rows
```

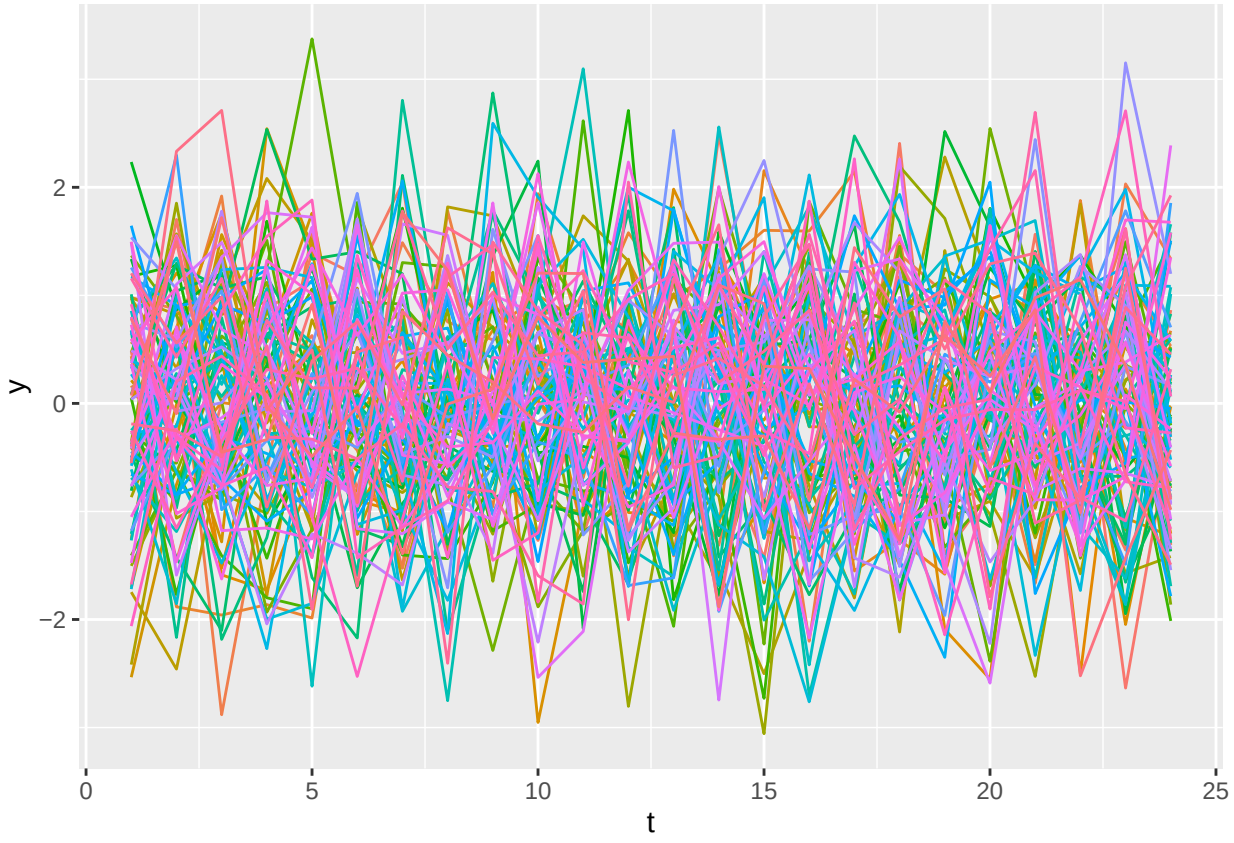

0.75
0.50
0.25
0.00
acf
4
8
12
16
lag [1]

2.7 Stationarity

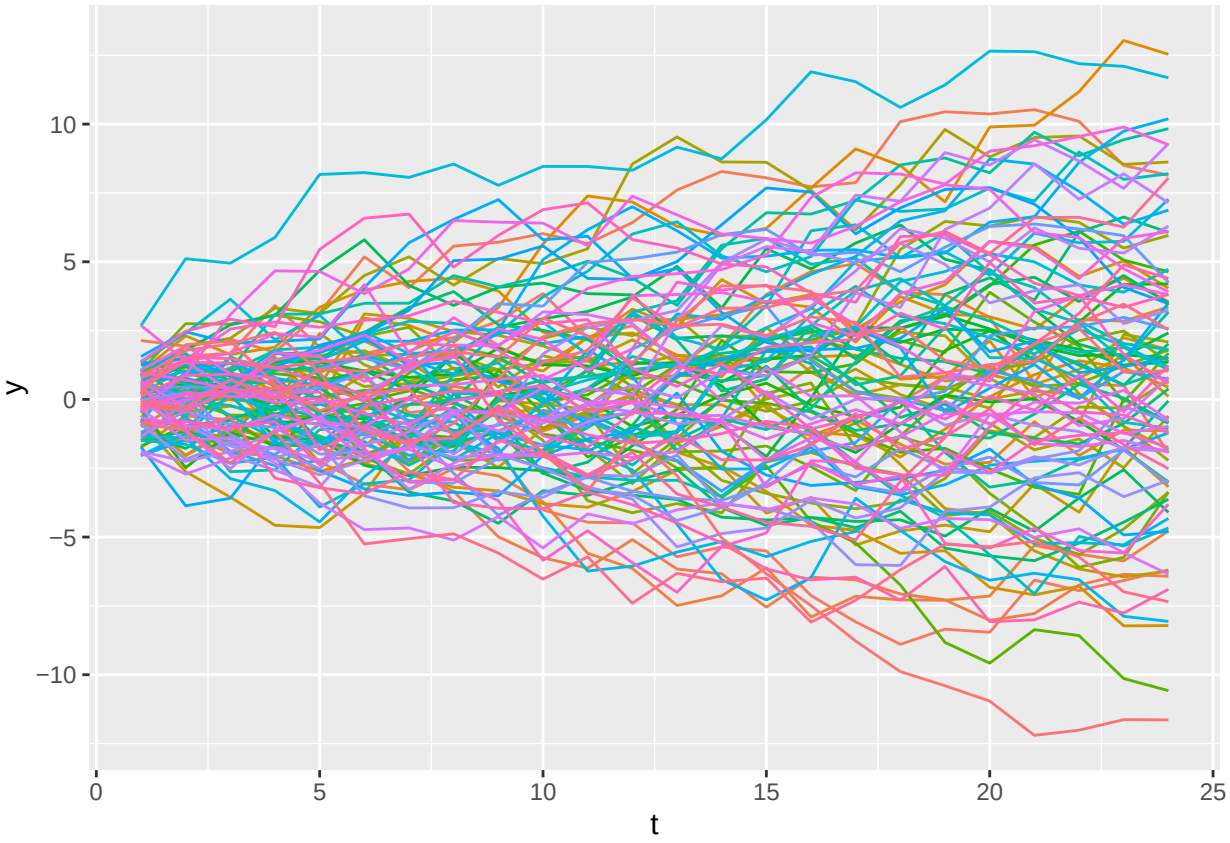
When a time series model satisfies the following conditions, it is said to exhibit stationarity.

Where μ_t is constant and independent of time t.

The autocovariance function $\gamma_{s,t}$ depends only on $|s-t|$, meaning it is independent of time t.



In the above example, we can say that the white noise model exhibits stationarity.



In the above example, we cannot say that the random walk model exhibits stationarity because it shows a pattern of increasing variance as time 't' increases.

199_research_Euijun_Kim

March 16, 2023

[1]:
```
import os
import sys
import warnings

# Redirect standard error to a null device
sys.stderr = open(os.devnull, 'w')

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # visualization library
import matplotlib.pyplot as plt # visualization library
from pandas import datetime
from plotly.offline import init_notebook_mode, iplot # plotly offline mode
init_notebook_mode(connected=True)
import plotly.graph_objs as go # plotly graphical object

import os
import warnings
warnings.filterwarnings("ignore") # if there is a warning after some codes,␣
 ↪this will avoid us to see them.
plt.style.use('ggplot') # style of plots. ggplot is one of the most used style,␣
 ↪I also like it.

sys.stderr = sys.__stderr__
```

[2]:
```
# Use the weather data from kaggle
```

[3]:
```
weather_station_location = pd.read_csv("Weather Station Locations.csv")
weather = pd.read_csv("Summary of Weather.csv")

weather_station_location = weather_station_location.loc[:,["WBAN","NAME","STATE/
 ↪COUNTRY ID","Latitude","Longitude"] ]
weather = weather.loc[:,["STA","Date","MeanTemp"] ]
```

[4]:
```
# There are two types of weather data: one is location data that displays␣
 ↪latitude, longitude, and other location information for each station, and␣
 ↪the other is actual temperature data for each station.
```

[5]:
```
weather_station_location = weather_station_location.loc[:,["WBAN","NAME","STATE/
↪COUNTRY ID","Latitude","Longitude"] ]
weather = weather.loc[:,["STA","Date","MeanTemp"] ]
```

[6]:
```
# data description
    #Weather station location:
        #WBAN: Weather station number
        #NAME: weather station name
        #STATE/COUNTRY ID: acronym of countries
        #Latitude: Latitude of weather station
        #Longitude: Longitude of weather station
    #Weather:
        #STA: eather station number (WBAN)
        #Date: Date of temperature measurement
        #MeanTemp: Mean temperature
```

[7]:
```
weather_station_location.head(5)
```

[7]:

	WBAN	NAME	STATE/COUNTRY ID	Latitude	Longitude
0	33013	AIN EL	AL	36.383333	6.650000
1	33031	LA SENIA	AL	35.616667	0.583333
2	33023	MAISON BLANCHE	AL	36.716667	3.216667
3	33044	TELERGMA	AL	36.116667	6.416667
4	12001	TINDOUF	AL	27.683333	-8.083333

[8]:
```
weather.head(5)
```

[8]:

	STA	Date	MeanTemp
0	10001	1942-7-1	23.888889
1	10001	1942-7-2	25.555556
2	10001	1942-7-3	24.444444
3	10001	1942-7-4	24.444444
4	10001	1942-7-5	24.444444

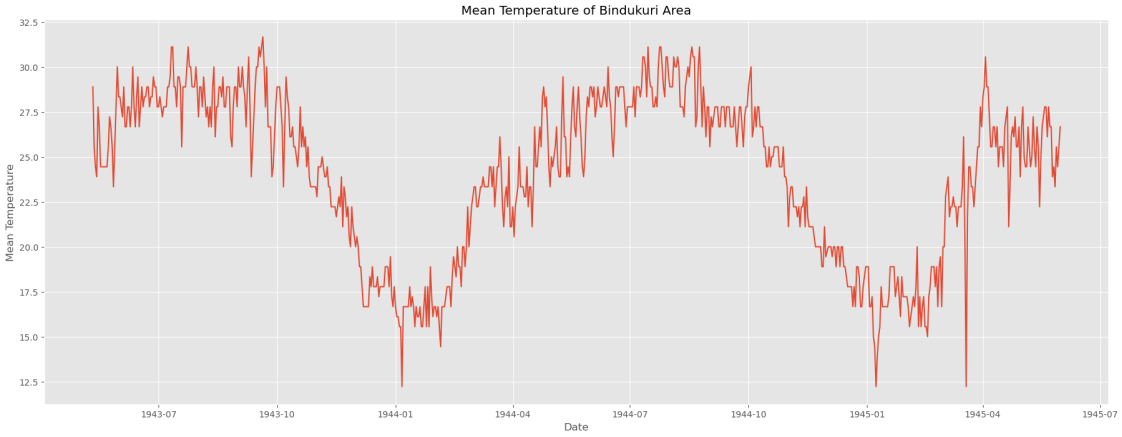
[9]:
```
weather_station_id = weather_station_location[weather_station_location.NAME ==␣
↪"BINDUKURI"].WBAN
weather_bin = weather[weather.STA == int(weather_station_id)]
weather_bin["Date"] = pd.to_datetime(weather_bin["Date"])
```

[10]:
```
# I will analyze the average daily temperature for the BINDUKURI region among␣
↪multiple regions.
```

weather_bin

[11]:
```
plt.figure(figsize=(22,8))
plt.plot(weather_bin.Date,weather_bin.MeanTemp)
plt.title("Mean Temperature of Bindukuri Area")
```

```
plt.xlabel("Date")
plt.ylabel("Mean Temperature")
plt.show()
```



[12]:
```
# Daily average temperature from May 11th, 1943 to May 31st, 1945.
```

[13]:
```
# lets create time series from weather
timeSeries = weather_bin.loc[:, ["Date","MeanTemp"]]
timeSeries.index = pd.DatetimeIndex(timeSeries.Date, freq='infer')
ts = timeSeries.drop("Date",axis=1)
```

[14]:
```
ts
```

[14]:
```
              MeanTemp
Date
1943-05-11  28.888889
1943-05-12  25.555556
1943-05-13  24.444444
1943-05-14  23.888889
1943-05-15  27.777778
…                …
1945-05-27  23.333333
1945-05-28  25.555556
1945-05-29  24.444444
1945-05-30  25.555556
1945-05-31  26.666667

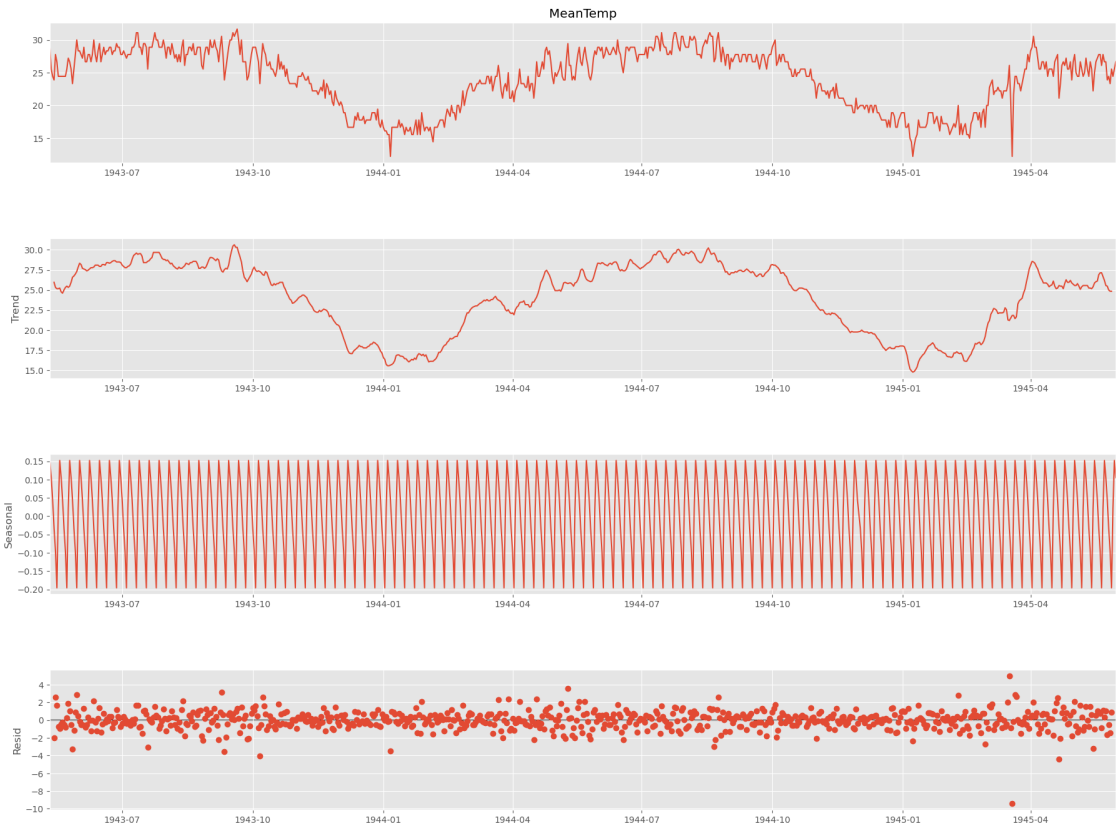
[751 rows x 1 columns]
```

[15]:
```
# When decomposed using time series decomposition, it looks like the following.␣
↪First, create a time series data in ts format.
```

[16]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(ts['MeanTemp'], model='additive', period=7)

fig = plt.figure()
fig = result.plot()
fig.set_size_inches(20, 15)
```

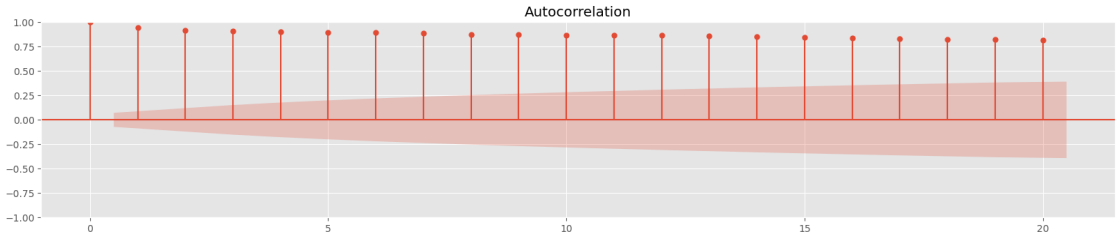
```
<Figure size 640x480 with 0 Axes>
```



[17]:
```
# Next, decompose using seasonal_decompose().
```

[18]:
```
import statsmodels.api as sm

fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts, lags=20, ax=ax1)
```



[19]:
```
# Since the data exhibits a pattern, stationarity is in doubt. To determine␣
 ↪this, an ACF plot was drawn.
```

[20]:
```
# It can be seen that the values slowly decrease. A very slow decrease in ACF␣
 ↪values indicates non-stationarity. To confirm stationarity using the␣
 ↪Augmented Dickey-Fuller test (ADF test), which is a unit root test.
# The hypothesis of this test is as follows:

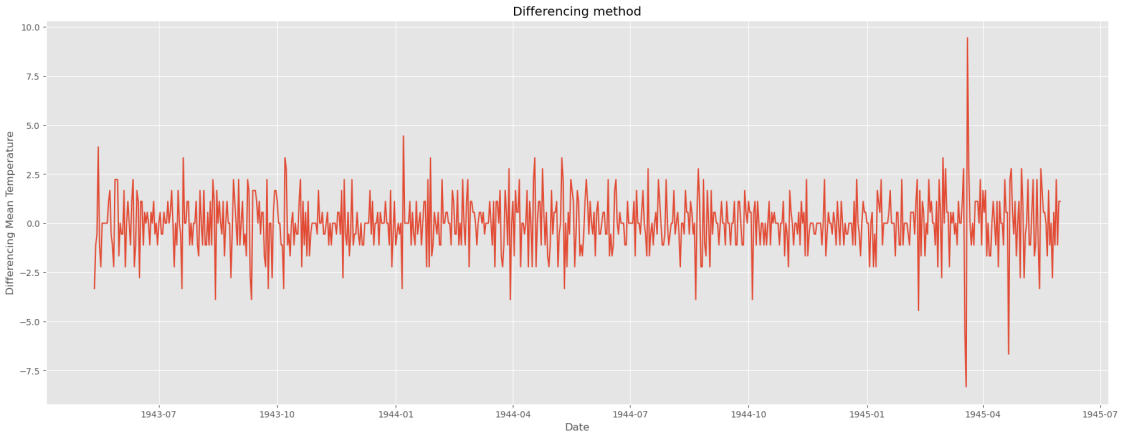
# H0 (null hypothesis): The data contains a unit root, meaning it is␣
 ↪non-stationary.
# H1 (alternative hypothesis): The data is stationary.
```

[21]:
```
from statsmodels.tsa.stattools import adfuller
result = adfuller(ts)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -1.409597
p-value: 0.577667
Critical Values:
        1%: -3.439
        5%: -2.865
        10%: -2.569
```

[22]:
```
# Since the p-value is greater than 0.05, we cannot reject the null hypothesis.␣
 ↪Therefore, the data does not satisfy the stationarity assumption.
```

[23]:
```
ts_diff = ts - ts.shift()
plt.figure(figsize=(22,8))
plt.plot(ts_diff)
plt.title("Differencing method")
plt.xlabel("Date")
plt.ylabel("Differencing Mean Temperature")
plt.show()
```



[24]:
```
# To address this, first-order differencing was performed.
```

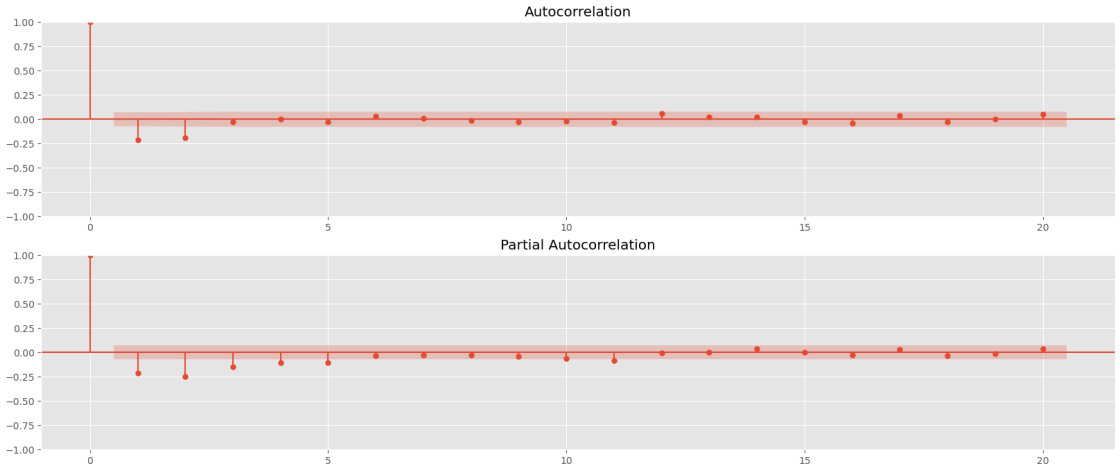
[25]:
```
result = adfuller(ts_diff[1:])
print('ADF Statistics: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values: ')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
ADF Statistics: -11.678956
p-value: 0.000000
Critical Values:
        1%: -3.439
        5%: -2.865
        10%: -2.569
```

[26]:
```
# As a result, no clear pattern was observed, and it appears to satisfy the␣
 ↪stationarity assumption.
# Since the p-value is less than 0.05, we reject the null hypothesis.
```

[27]:
```
import statsmodels.api as sm

fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts_diff[1:], lags=20, ax=ax1) #
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(ts_diff[1:], lags=20, ax=ax2)# , lags=40
```



[28]:
```
# Using the differenced data that satisfies stationarity,
# we will draw ACF and PACF plots to determine the p and q parameters for the␣
 ↪ARIMA model.

# Both ACF and PACF quickly converge to zero, and then converge to zero again␣
 ↪after the second lag.

# Therefore, we can try ARIMA models such as ARIMA(2,1,2) as a base model,
# and ARIMA(2,1,1), ARIMA(1,1,2), ARIMA(1,1,1), etc
```

[29]:
```
import os
import sys
import warnings

# Redirect standard error to a null device
sys.stderr = open(os.devnull, 'w')


from statsmodels.tsa.arima.model import ARIMA
from pandas import datetime

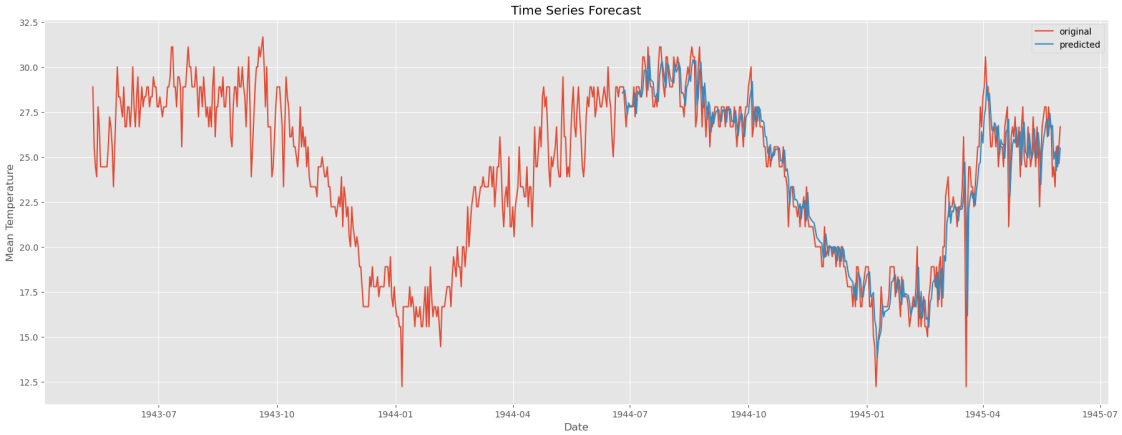
# fit model
model = ARIMA(ts, order=(2,1,2))
model_fit = model.fit()

# predict
start_index = datetime(1944, 6, 25)
end_index = datetime(1945, 5, 31)
forecast = model_fit.predict(start=start_index, end=end_index, typ='levels')

# visualization
```

```
plt.figure(figsize=(22,8))
plt.plot(weather_bin.Date,weather_bin.MeanTemp,label = "original")
plt.plot(forecast,label = "predicted")
plt.title("Time Series Forecast")
plt.xlabel("Date")
plt.ylabel("Mean Temperature")
plt.legend()
plt.show()


sys.stderr = sys.__stderr__
```



[30]:
```
# Here are the results of the ARIMA(2,1,2) model:

# We tested from June 25th, 1994 as the start of the test set. This corresponds␣
 ↪to 340 observations.

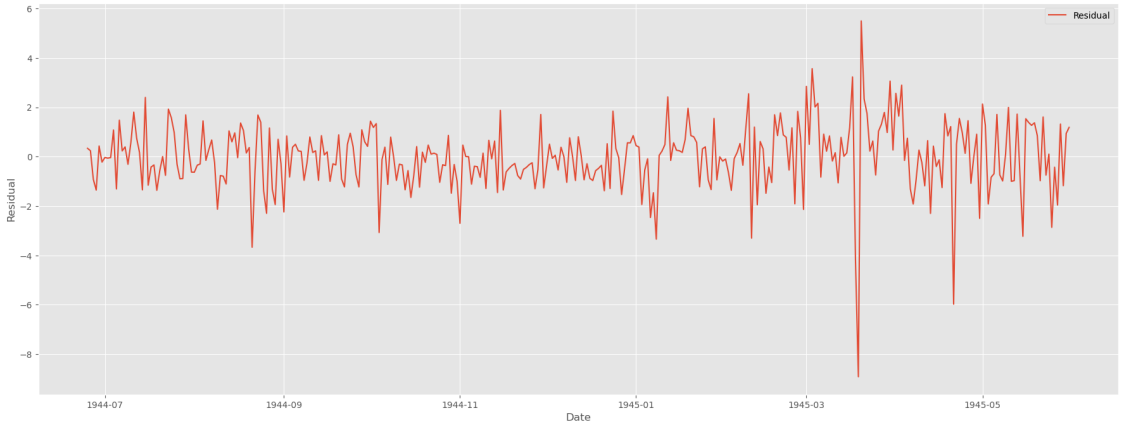
# When fitting the model, it is recommended to set the typ parameter to␣
 ↪'levels' for a differenced model.
# If typ is left as the default 'linear' value, the results will be given in␣
 ↪terms of the differenced values.
```

[31]:
```
# Visually, the result looks very good.

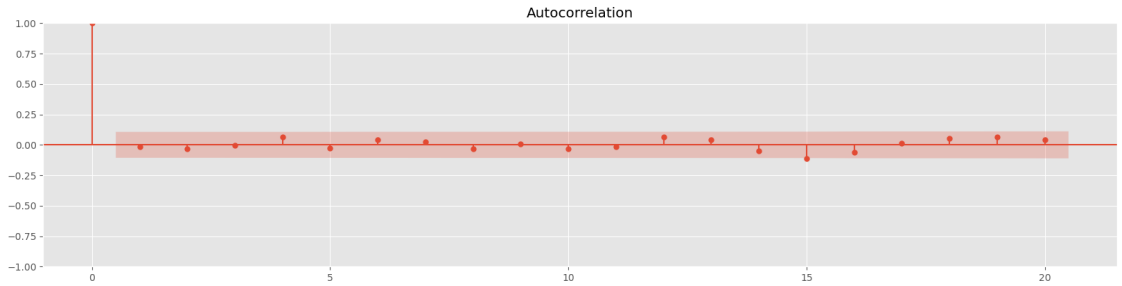
# Finally, residual analysis is used to check if there is anything missing or␣
 ↪any problems with the model.
#Residuals should not show any patterns or characteristics.
#If there is any pattern, it means that the model has been applied less and␣
 ↪less, which is not desirable.
```

[32]:
```
resi = np.array(weather_bin[weather_bin.Date>=start_index].MeanTemp) - np.
 ↪array(forecast)

plt.figure(figsize=(22,8))
plt.plot(weather_bin.Date[weather_bin.Date>=start_index],resi,label='Residual')
plt.xlabel("Date")
plt.ylabel("Residual")
plt.legend()
plt.show()
```



[33]:
```
fig = plt.figure(figsize=(20,10))
axl = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(resi,lags=20,ax=axl)
```



[34]:
```
result = adfuller(resi)
print('ADF Statistics: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values: ')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
ADF Statistics: -18.629906
p-value: 0.000000
Critical Values:
        1%: -3.450
        5%: -2.870
        10%: -2.571
```

[35]:
```
# ACF plot and ADF test are also used to determine stationarity.

# The ACF plot also converges quickly to 0 and the ADF test also shows a very␣
 ↪small p-value.
```

[36]:
```
from sklearn import metrics


def scoring(y_true, y_pred):
    r2 = round(metrics.r2_score(y_true, y_pred) * 100, 3)
    #     mae = round(metrics.mean_absolute_error(y_true, y_pred),3)
    corr = round(np.corrcoef(y_true, y_pred)[0, 1], 3)
    mape = round(
        metrics.mean_absolute_percentage_error(y_true, y_pred) * 100, 3)
    rmse = round(metrics.mean_squared_error(y_true, y_pred, squared=False), 3)

    df = pd.DataFrame({
        'R2': r2,
        "Corr": corr,
        "RMSE": rmse,
        "MAPE": mape
    },
                      index=[0])
    return df
```

[37]:
```
scoring(np.array(weather_bin[weather_bin.Date>=start_index].MeanTemp),np.
 ↪array(forecast))
```

[37]:

	R2	Corr	RMSE	MAPE
0	91.018	0.954	1.365	4.449

[38]:
```
# Lastly, we can check the performance of the model
```

- **Appendix**

Python code:

```
import os
import sys
import warnings

# Redirect standard error to a null device
sys.stderr = open(os.devnull, 'w')

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # visualization library
import matplotlib.pyplot as plt # visualization library
from pandas import datetime
from plotly.offline import init_notebook_mode, iplot # plotly offline mode
init_notebook_mode(connected=True)
import plotly.graph_objs as go # plotly graphical object

import os
import warnings
warnings.filterwarnings("ignore") # if there is a warning after some codes, this will avoid us to see them.
plt.style.use('ggplot') # style of plots. ggplot is one of the most used style, I also like it.

sys.stderr = sys.__stderr__

weather_station_location = pd.read_csv("Weather Station Locations.csv")
weather = pd.read_csv("Summary of Weather.csv")

weather_station_location = weather_station_location.loc[:,["WBAN","NAME","STATE/COUNTRY ID","Latitude","Longitude"] ]
weather = weather.loc[:,["STA","Date","MeanTemp"] ]

weather_station_location = weather_station_location.loc[:,["WBAN","NAME","STATE/COUNTRY ID","Latitude","Longitude"] ]
weather = weather.loc[:,["STA","Date","MeanTemp"] ]

weather_station_location.head(5)

weather.head(5)
```

```
weather_station_id = weather_station_location[weather_station_location.NAME ==
"BINDUKURI"].WBAN
weather_bin = weather[weather.STA == int(weather_station_id)]
weather_bin["Date"] = pd.to_datetime(weather_bin["Date"])

plt.figure(figsize=(22,8))
plt.plot(weather_bin.Date,weather_bin.MeanTemp)
plt.title("Mean Temperature of Bindukuri Area")
plt.xlabel("Date")
plt.ylabel("Mean Temperature")
plt.show()

# lets create time series from weather
timeSeries = weather_bin.loc[:, ["Date","MeanTemp"]]
timeSeries.index = pd.DatetimeIndex(timeSeries.Date, freq='infer')
ts = timeSeries.drop("Date",axis=1)

ts

from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(ts['MeanTemp'], model='additive', period=7)

fig = plt.figure()
fig = result.plot()
fig.set_size_inches(20, 15)

import statsmodels.api as sm

fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts, lags=20, ax=ax1)

from statsmodels.tsa.stattools import adfuller
result = adfuller(ts)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))


ts_diff = ts - ts.shift()
plt.figure(figsize=(22,8))
plt.plot(ts_diff)
```

```
plt.title("Differencing method")
plt.xlabel("Date")
plt.ylabel("Differencing Mean Temperature")
plt.show()

result = adfuller(ts_diff[1:])
print('ADF Statistics: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values: ')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

import statsmodels.api as sm

fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts_diff[1:], lags=20, ax=ax1) #
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(ts_diff[1:], lags=20, ax=ax2)# , lags=40

import os
import sys
import warnings

# Redirect standard error to a null device
sys.stderr = open(os.devnull, 'w')


from statsmodels.tsa.arima.model import ARIMA
from pandas import datetime

# fit model
model = ARIMA(ts, order=(2,1,2))
model_fit = model.fit()

# predict
start_index = datetime(1944, 6, 25)
end_index = datetime(1945, 5, 31)
forecast = model_fit.predict(start=start_index, end=end_index, typ='levels')

# visualization
plt.figure(figsize=(22,8))
plt.plot(weather_bin.Date,weather_bin.MeanTemp,label = "original")
plt.plot(forecast,label = "predicted")
```

```
plt.title("Time Series Forecast")
plt.xlabel("Date")
plt.ylabel("Mean Temperature")
plt.legend()
plt.show()


sys.stderr = sys.__stderr__

resi = np.array(weather_bin[weather_bin.Date>=start_index].MeanTemp) - np.array(forecast)

plt.figure(figsize=(22,8))
plt.plot(weather_bin.Date[weather_bin.Date>=start_index],resi,label='Residual')
plt.xlabel("Date")
plt.ylabel("Residual")
plt.legend()
plt.show()

fig = plt.figure(figsize=(20,10))
axl = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(resi,lags=20,ax=axl)

result = adfuller(resi)
print('ADF Statistics: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values: ')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))

from sklearn import metrics


def scoring(y_true, y_pred):
    r2 = round(metrics.r2_score(y_true, y_pred) * 100, 3)
    #     mae = round(metrics.mean_absolute_error(y_true, y_pred),3)
    corr = round(np.corrcoef(y_true, y_pred)[0, 1], 3)
    mape = round(
        metrics.mean_absolute_percentage_error(y_true, y_pred) * 100, 3)
    rmse = round(metrics.mean_squared_error(y_true, y_pred, squared=False), 3)

    df = pd.DataFrame({
        'R2': r2,
        "Corr": corr,
        "RMSE": rmse,
```

```
        "MAPE": mape
    },
                    index=[0])
    return df

scoring(np.array(weather_bin[weather_bin.Date>=start_index].MeanTemp),np.array(forecast))
```

R code:

```
```{r setup, include=FALSE}
  library(knitr)
  opts_chunk$set(echo = FALSE)
```



```{r, include=FALSE}
# install.packages("fpp3")
# install.packages("janitor")
library(fpp3)
```

```{r,include=FALSE}
prison <- readr::read_csv("https://OTexts.com/fpp3/extrafiles/prison_population.csv")
head(prison)
```



```{r,include=FALSE}
prison %<>%
  janitor::clean_names() %>%
  mutate(quarter = yearquarter(date),
         .keep = "unused") %>%
  as_tsibble(key = state:indigenous,
             index = quarter)
```



```{r}
prison %>%
  filter(state == "ACT", gender == "Female",
         legal =="Remanded", indigenous == "ATSI") %>%
  autoplot(count) +
  labs(title = "Change of prison number as time goes",
       y = "number",
```
```

```
      x = "time")
```

```{r}
vic_elec %>%
  janitor::clean_names() %>%
  filter(year(time) == 2012) %>%
  gg_season(demand, period = "day") +
  theme(legend.position = "none") +
  labs(y="MWh", title="Electricity demand: Victoria")
```

```{r}
vic_elec %>%
  janitor::clean_names() %>%
  filter(year(time) == 2012) %>%
  gg_season(demand, period = "week") +
  theme(legend.position = "none") +
  labs(y="MWh", title="Electricity demand: Victoria")
```

```{r}
vic_elec %>%
  janitor::clean_names() %>%
  filter(year(time) == 2012) %>%
   autoplot(demand) +
  labs(y = "GW",
       title = "Half-hourly electricity demand: Victoria")
```

```{r}
vic_elec %>%
  janitor::clean_names() %>%
  filter(year(time) == 2012) %>%
   autoplot(temperature) +
  labs(y = "Degrees Celsius",
       title = "Half-hourly electricity demand: Victoria")
```

```

```{r}
vic_elec %>%
  janitor::clean_names() %>%
  filter(year(time) == 2012) %>%
  ggplot(aes(x = temperature, y = demand)) +
  geom_point() +
  labs(x = "Temperature (degrees Celsius)",
       y = "Electricity demand (GW)")
```

```{r}
prison %>%
  group_by(state) %>%
  summarise(count = sum(count)) %>%
  ggplot(aes(x = quarter, y = count)) +
  geom_line() +
  facet_grid(vars(state), scales = "free_y") +
  labs(title = "Australian prison population by states",
       y= "population")
```

```{r}
x <- c(1,3,6,10,15,21,28,36,45,55)
lag_x_1 <- lag(x, n = 1)
lag_x_2 <- lag(x, n = 2)
cbind(x,lag_x_1,lag_x_2)
```

```{r}
aus_production %>%
  janitor::clean_names() %>%
  filter(year(quarter) >= 2000) %>%
  gg_lag(beer, geom = "point") +
  labs(x = "lag(Beer, k)")
```

```{r}

```
x <- c(1,3,6,10,15,21,28,36,45,55)
lag_x_1 <- lag(x, n = 1)
lag_x_2 <- lag(x, n = 2)
# ACF value: r_2
sum((x - mean(x)) * (lag(x, n = 2) - mean(x)),
    na.rm = TRUE) / (var(x) * (length(x) - 1))
```

```{r}
my_x <- tsibble(value = x,
                time = 1:length(x),
                index = time)
my_x %>%
  ACF(value, lag_max = 9)
```

```{r}
my_x %>%
  ACF(value, lag_max = 9) %>%
  autoplot() +
  labs(title = "The autocorrelation function of vector x",
       x = "Time lags",
       y = "Autocorrelation function")
```

```{r}
library(fpp3)
library(dplyr)
library(ggplot2)
```

```{r}
vic_elec %>%
  janitor::clean_names() %>%
  filter(year(time) == 2012) %>%
  gg_season(demand, period = "day") +
  theme(legend.position = "none") +
  labs(y="MWh", title="Electricity demand: Victoria")
```

```{r}
library(tsibble)
library(ggplot2)
white.noise <- tsibble(t = seq_len(24), y = rnorm(24), index = t)
white.noise %>% autoplot()
```

```{r}
N <- 100
df.wn <- data.frame(ind = integer(), t = integer(), y = double())
for (i in seq(N)){
  white.noise <- tsibble(t = seq_len(24), y = rnorm(24), index = t)
  white.noise <- white.noise %>%
    mutate(ind = i) %>%
    select(ind, t, y) %>%
    as.data.frame()
  df.wn <- df.wn %>% bind_rows(white.noise)
}

df.wn$ind <- df.wn$ind %>% as.factor()

# One sample
ggplot(df.wn %>% filter(ind == "1"), aes(x = t, y = y, col = ind)) + geom_line()
```

```{r}
# Distribution
ggplot(df.wn, aes(x = t, y = y, col = ind)) +
  geom_line() +
  theme(legend.position="none")
```

\
\begin{equation}
\begin{aligned}
\mu_t &= E[Y_t] = E[Y_{t-1}] + E[e_t] = \mu_{t-1} + 0 = \dots = \mu_1 = 0 \\
Var(Y_t) &= Var(Y_{t-1}) + Var(e_t) = Var(Y_{t-1}) + \sigma_e^2 = \dots = t \sigma_e^2 \\
Cov(Y_t, Y_s) &= Cov(Y_{s+e_{s+1}} + \dots + e_t, Y_s) = Cov(Y_s, Y_s) = Var(Y_s) =
s\sigma_e^2 \text{ for } t>s \\
Corr(Y_t, Y_s) &= \frac{Cov(Y_t, Y_s)}{\sqrt{Var(Y_t)}\sqrt{Var(Y_s)}} =
\frac{s\sigma_e^2}{\sqrt{t\sigma_e^2}\sqrt{s\sigma_e^2}} = \sqrt{\frac{s}{t}} \text{ for } t>s \\
\end{aligned}
\end{equation}

\

```{r}
N <- 100
df.rw <- data.frame(ind = integer(), t = integer(), y = double())
for (i in seq(N)){
  random.walk <- tsibble(t = seq_len(24), y = cumsum(rnorm(24)), index = t)
  random.walk <- random.walk %>%
    mutate(ind = i) %>%
    select(ind, t, y) %>%
    as.data.frame()
  df.rw <- df.rw %>% bind_rows(random.walk)
}

df.rw$ind <- df.rw$ind %>% as.factor()

# One sample
ggplot(df.rw %>% filter(ind == "1"), aes(x = t, y = y, col = ind)) + geom_line()
```

```{r}
# Distribution
ggplot(df.rw, aes(x = t, y = y, col = ind)) +
  geom_line() +
  theme(legend.position="none")
```

\
\begin{equation}
\begin{aligned}
\{Y_t &= \frac{1}{2} (e_t + e_{t-1}) : t=2,3,\ldots\}\\
\mu_t &= E[Y_t] = \frac{1}{2} (E[e_t] + E[e_{t-1}]) = 0\\
Var(Y_t) &= \frac{1}{4} Var(e_t) + \frac{1}{4} Var(e_{t-1}) = \frac{1}{2} Var(e_t)\\
Cov(Y_t, Y_{t-1}) &= Cov\left(\frac{1}{2}(e_t+e_{t-1}), \frac{1}{2}(e_{t-1}+e_{t-2})\right)\\
&= \frac{1}{4}Cov(e_t, e_{t-2}) + \frac{1}{4}Cov(e_t, e_{t-1}) + \frac{1}{4}Cov(e_{t-1}, e_{t-2})\\
&= \frac{1}{4}Var(e_t) = \frac{1}{4}\sigma_e^2\\
Cov(Y_t, Y_s) &= Cov\left(\frac{1}{2}(e_t+e_{t-1}), \frac{1}{2}(e_s+e_{s-1})\right) = 0 \quad
\text{for } |t-s| > 1\\
Corr(Y_t, Y_{t-1}) &= \frac{Cov(Y_t, Y_{t-1})}{\sqrt{Var(Y_t)}\sqrt{Var(Y_{t-1})}} = \frac{1}{2}\\
Corr(Y_t, Y_s) &= 0 \quad \text{for } |t-s| > 1\
\end{aligned}

\end{equation}\

```{r}
# install.packages("zoo")
library(zoo)
white.noise <- tsibble(t = seq_len(100), y = rnorm(100), index = t)

# moving average with rollmean
white.noise.r <- white.noise %>%
  mutate(yavg = rollmean(y, k = 5, align = "right", fill = NA ))

white.noise.r
```

```{r}
# Mean
white.noise <- white.noise %>%
  mutate(y1 = lag(y, n = 1)) %>%
  rowwise() %>%
  mutate(avg = mean(c_across(y:y1), na.rm=TRUE))

white.noise.r %>%
  ACF(yavg, lag_max = 17) %>%
  autoplot()
```

```{r}
# Distribution
ggplot(df.wn, aes(x = t, y = y, col = ind)) + geom_line() +
  theme(legend.position="none")
```

```{r}
# Distribution
ggplot(df.rw, aes(x = t, y = y, col = ind)) + geom_line() +
  theme(legend.position="none")
```