Time Series Analysis

Road Map

- Why is prediction supervised learning, how it differs from classification.
- Beer dataset time series
- Electricity prediction example

What is Prediction?

A **prediction** or forecast is a statement about the way things will happen in the future, often but not always based on experience or knowledge.

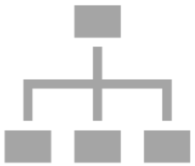
Why is Prediction Supervised Learning

We use labeled training data to learn and make prediction.

Supervised learning means we have an already marked dataset giving us what the learning process should give you.

Forms of data analysis to predict future data trends

There are two forms of data analysis that can be used for extracting models describing important classes or to predict future data trends. These two forms are as follows



Classification

Regression

Also, called prediction in earlier literature

Classification vs Regression

Classification	Regression/Prediction
It predicts categorical class labels	It predicts continuous valued functions
Example: A bank loan officer wants to analyze the data in order to know which customer (loan applicant) are risky or which are safe. A model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data	**Example:** Suppose the marketing manager needs to predict how much a given customer will spend during a sale at his company. In this example we are bothered to predict a numeric value.

Reference: https://www.tutorialspoint.com/

What is Time Series?

A **time series** is simply a series of data points taken at specified **times** usually at equal intervals.

It is used to **predict** the future values based on the **previous** observed values.

Importance of Time Series Analysis

- Business Forecasting
- Understand past behavior
- Plan future
- Evaluate current accomplishment

Decompose One Time Series into Multiple Series

Time series decomposition is a mathematical procedure which transforms a time series into multiple different time series.

The original time series is often split into 3 component series:

- Seasonal
- Trend
- Random/noise/remainder

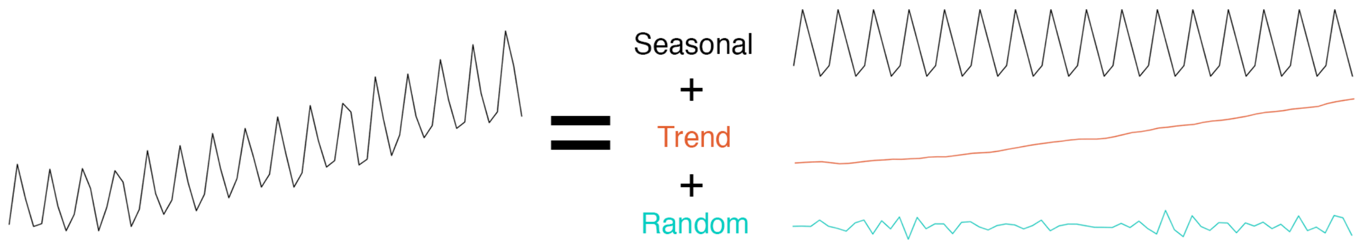


Image Source: https://anomaly.io/seasonal-trend-decomposition-in-r

Seasonal

- Patterns that repeat with a fixed period of time.
- **Example:** A website might receive more visits during weekends so this would produce data with a seasonality of 7 days.

Trend

- The underlying trend of the metrics.
- **Example:** A website increasing in popularity should show a general trend that goes up.

Random/ Noise/ Remainder

- This is the residuals of the original time series after the seasonal and trend series are removed.

Let’s take one example

Data: beer.csv

Monthly beer consumption in Australia from 1956 to 1995

	A	B	C
1	beer		
2	93	2	
3	96		
4	95	2	
5	77	1	
6	70	9	
7	64	8	
8	70	1	
9	77	3	
10	79	5	
11	100	6	
12	100	7	
13	107	1	
14	95	9	
15	82	8	
16	83	3	
17	80		
18	80	4	
19	67	5	
20	75	7	
21	71	1	
22	89	3	
23	101	1	
24	105	2	
25	114	1	
26	96	3	
27	84	4	
28	91	2	
29	81	9	
30	80	5	
31	70	4	
32	74	8	
33	75	9	

Explore Data

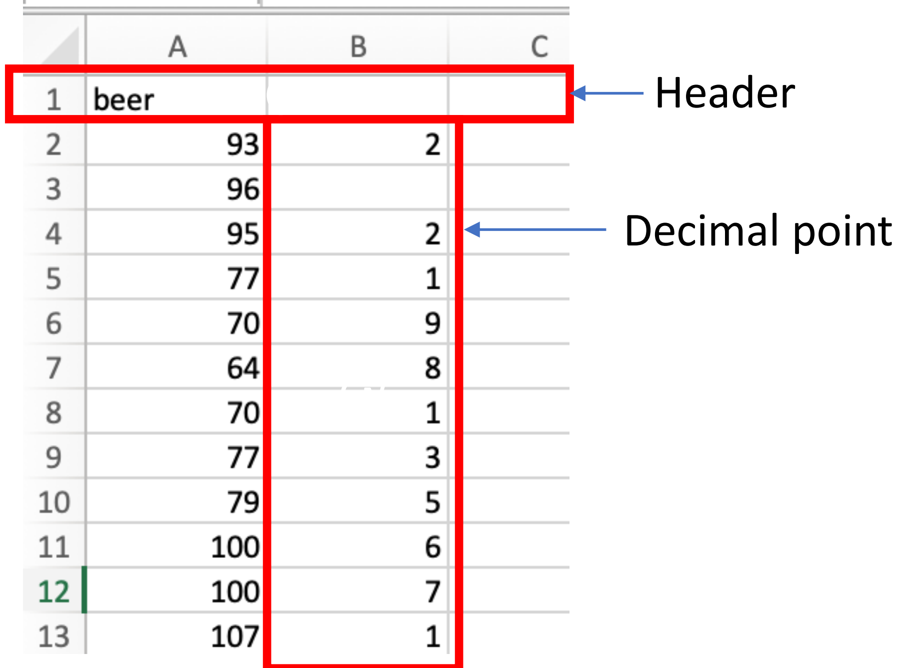
```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer

beer<-ts(beer[,1],start=1956,freq=12);

beer

plot(beer)

plot(stl(log(beer),s.window="periodic"))

lbeer<-log(beer);

plot(lbeer)
```

read.csv() is used to read csv file.

header: This CSV file has **header** so it is set to **true.**

dec: the character used in the file for decimal points.

sep: the field separator character. Values on each line of the file are separated by this character.



Explore Data

```r

beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer

beer<-ts(beer[,1],start=1956,freq=12);

beer

plot(beer)

plot(stl(log(beer),s.window="periodic"))

lbeer<-log(beer);

plot(lbeer)
```

Output

```
> beer
    beer
1   93.2
2   96.0
3   95.2
4   77.1
5   70.9
6   64.8
7   70.1
8   77.3
9   79.5
10 100.6
11 100.7
12 107.1
```

Explore Data

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");
beer
beer<-ts(beer[,1],start=1956,freq=12);
beer
plot(beer)
plot(stl(log(beer),s.window="periodic"))
lbeer<-log(beer);
plot(lbeer)
```

The function ts is used to create time-series objects.

freq = 12 when the data are sampled monthly and the natural time period is a year.

freq = 7 when the data are sampled daily, and the natural time period is a week.

Explore Data

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer

beer<-ts(beer[,1],start=1956,freq=12);

beer

plot(beer)

plot(stl(log(beer),s.window="periodic"))

lbeer<-log(beer);

plot(lbeer)
```

Output

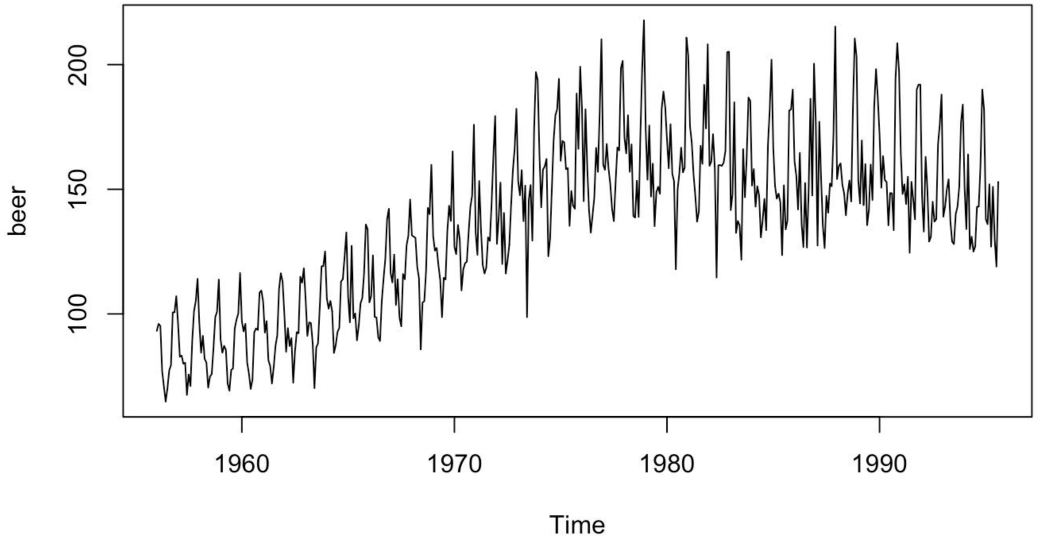
```
> beer
```

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1956	93.2	96.0	95.2	77.1	70.9	64.8	70.1	77.3	79.5	100.6	100.7	107.1
1957	95.9	82.8	83.3	80.0	80.4	67.5	75.7	71.1	89.3	101.1	105.2	114.1
1958	96.3	84.4	91.2	81.9	80.5	70.4	74.8	75.9	86.3	98.7	100.9	113.8
1959	89.8	84.4	87.2	85.6	72.0	69.2	77.5	78.1	94.3	97.7	100.2	116.4
1960	97.1	93.0	96.0	80.5	76.1	69.9	73.6	92.6	94.2	93.5	108.5	109.4
1961	105.1	92.5	97.1	81.4	79.1	72.1	78.7	87.1	91.4	109.9	116.3	113.0
1962	100.0	84.8	94.3	87.1	90.3	72.4	84.9	92.7	92.2	114.9	112.5	118.3
1963	106.0	91.2	96.6	96.3	88.2	70.2	86.5	88.2	102.8	119.1	119.2	125.1
1964	106.1	102.1	105.2	101.0	84.3	87.5	92.7	94.4	113.0	113.9	122.9	132.7
1965	106.9	96.6	127.3	98.2	100.2	89.4	95.3	104.2	106.4	116.2	135.9	134.0
1966	104.6	107.1	123.5	98.8	98.6	90.6	89.1	105.2	114.0	122.1	138.0	142.2
1967	116.4	112.6	123.8	103.6	113.9	98.6	95.0	116.0	113.9	127.5	131.4	145.9
1968	131.5	131.0	130.5	118.9	114.3	85.7	104.6	105.1	117.3	142.5	140.0	159.8
1969	131.2	125.4	126.5	119.4	113.5	98.7	114.5	113.8	133.1	143.4	137.3	165.2
1970	126.9	124.0	135.7	130.0	109.4	117.8	120.3	121.0	132.3	142.9	147.4	175.9
1971	132.6	123.7	153.3	134.0	119.6	116.2	118.6	130.7	129.3	144.4	163.2	179.4
1972	128.1	138.4	152.7	120.0	140.5	116.2	121.4	127.8	143.6	157.6	166.2	182.3
1973	153.1	147.6	157.7	137.2	151.5	98.7	145.8	151.7	129.4	174.1	197.0	193.9
1974	164.1	142.8	157.9	159.2	162.2	123.1	130.0	150.1	169.4	179.7	182.1	194.3
1975	161.4	169.4	168.8	158.1	158.5	135.3	149.3	143.4	142.2	188.4	166.2	199.2

Explore Data

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer

beer<-ts(beer[,1],start=1956,freq=12);

beer

plot(beer)

plot(stl(log(beer),s.window="periodic"))

lbeer<-log(beer);

plot(lbeer)
```

Output

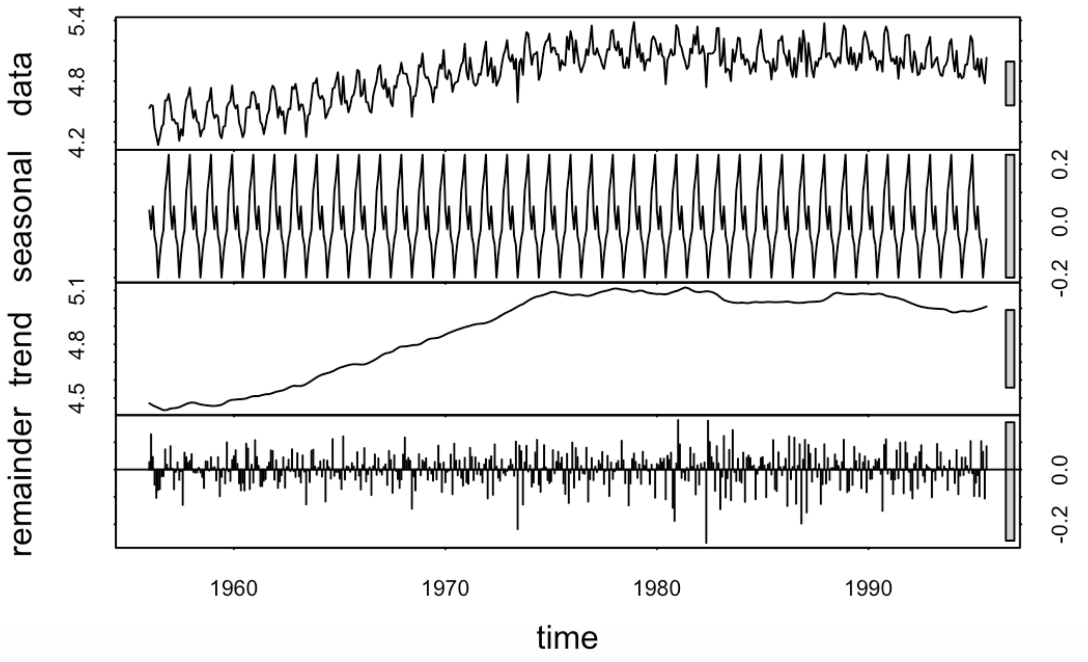


Explore Data

```r

beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer

beer<-ts(beer[,1],start=1956,freq=12);

beer

plot(beer)

plot(stl(log(beer),s.window="periodic"))

lbeer<-log(beer);

plot(lbeer)
```

Seasonal decomposition using stl( )

Output



Explore Data

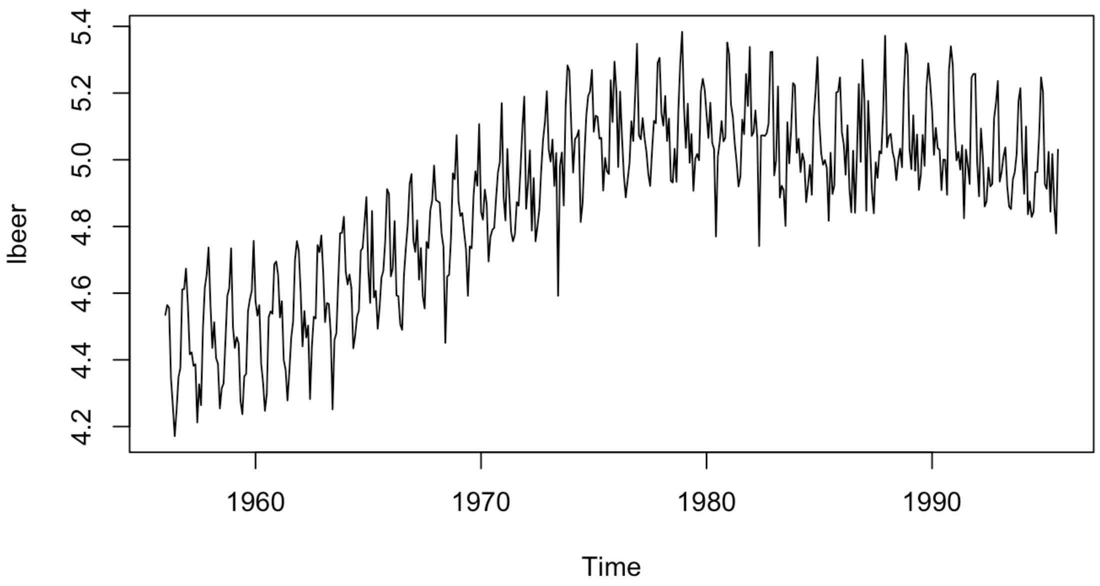
```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");
beer
beer<-ts(beer[,1],start=1956,freq=12);
beer
plot(beer)
plot(stl(log(beer),s.window="periodic"))
lbeer<-log(beer);
plot(lbeer)
```

We are doing little bit transformation and explore more. So we are using logarithm.

Explore Data

```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");
beer
beer<-ts(beer[,1],start=1956,freq=12);
beer
plot(beer)
plot(stl(log(beer),s.window="periodic"))
lbeer<-log(beer);
plot(lbeer)
```

Output



An imaginative solution by studying the pattern

```
t<-seq(1956,1995.2,length=length(beer))

t

t2<-t^2
sin.t<-sin(2*pi*t)
cos.t<-cos(2*pi*t)

sinCosSquareLM=lm(lbeer~t+t2+sin.t+cos.t)

mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
```

seq function generate regular sequences

seq(Start Number, End number, Length)

Here Sequence is starting from **1956**
Last number in the sequence will be **1995.2**

Length of sequence will be length of beer dataset which is **476**

So there will be 476 numbers in **t** variable

An imaginative solution by studying the pattern

```
t<-seq(1956,1995.2,length=length(beer))

t

t2<-t^2
sin.t<-sin(2*pi*t)
cos.t<-cos(2*pi*t)

sinCosSquareLM=lm(lbeer~t+t2+sin.t+cos.t)

mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
```

Output

```
> t
  [1] 1956.000 1956.083 1956.165 1956.248 1956.330 1956.413 1956.495
  [8] 1956.578 1956.660 1956.743 1956.825 1956.908 1956.990 1957.073
 [15] 1957.155 1957.238 1957.320 1957.403 1957.485 1957.568 1957.651
 [22] 1957.733 1957.816 1957.898 1957.981 1958.063 1958.146 1958.228
 [29] 1958.311 1958.393 1958.476 1958.558 1958.641 1958.723 1958.806
 [36] 1958.888 1958.971 1959.053 1959.136 1959.219 1959.301 1959.384
 [43] 1959.466 1959.549 1959.631 1959.714 1959.796 1959.879 1959.961
 [50] 1960.044 1960.126 1960.209 1960.291 1960.374 1960.456 1960.539
 [57] 1960.621 1960.704 1960.787 1960.869 1960.952 1961.034 1961.117
 [64] 1961.199 1961.282 1961.364 1961.447 1961.529 1961.612 1961.694
 [71] 1961.777 1961.859 1961.942 1962.024 1962.107 1962.189 1962.272
 [78] 1962.355 1962.437 1962.520 1962.602 1962.685 1962.767 1962.850
 [85] 1962.932 1963.015 1963.097 1963.180 1963.262 1963.345 1963.427
 [92] 1963.510 1963.592 1963.675 1963.757 1963.840 1963.923 1964.005
 [99] 1964.088 1964.170 1964.253 1964.335 1964.418 1964.500 1964.583
[106] 1964.665 1964.748 1964.830 1964.913 1964.995 1965.078 1965.160
[113] 1965.243 1965.325 1965.408 1965.491 1965.573 1965.656 1965.738
[120] 1965.821 1965.903 1965.986 1966.068 1966.151 1966.233 1966.316
[127] 1966.398 1966.481 1966.563 1966.646 1966.728 1966.811 1966.893
[134] 1966.976 1967.059 1967.141 1967.224 1967.306 1967.389 1967.471
[141] 1967.554 1967.636 1967.719 1967.801 1967.884 1967.966 1968.049
[148] 1968.131 1968.214 1968.296 1968.379 1968.461 1968.544 1968.627
```

An imaginative solution by studying the pattern

```r
t<-seq(1956,1995.2,length=length(beer))
t
t2<-t^2
sin.t<-sin(2*pi*t)
cos.t<-cos(2*pi*t)
sinCosSquareLM=lm(lbeer~t+t2+sin.t+cos.t)
mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
```

Generate other variables which we will use while creating Model

An imaginative solution by studying the pattern

```
t<-seq(1956,1995.2,length=length(beer))

t

t2<-t^2
sin.t<-sin(2*pi*t)
cos.t<-cos(2*pi*t)

sinCosSquareLM=lm(lbeer~t+t2+sin.t+cos.t)

mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
```

Here we are creating model.

lm() is used to fit linear models. It can be used to carry out regression.

An imaginative solution by studying the pattern

```
t<-seq(1956,1995.2,length=length(beer))
t
t2<-t^2
sin.t<-sin(2*pi*t)
cos.t<-cos(2*pi*t)
sinCosSquareLM=lm(lbeer~t+t2+sin.t+cos.t)
mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
```

1.972046% Accuracy

Output

```
> mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
[1] 1.972046
```

Not always possible to use such imagination

An imaginative solution by studying the pattern

```r
t<-seq(1956,1995.2,length=length(beer))

t

t2<-t^2
sin.t<-sin(2*pi*t)
cos.t<-cos(2*pi*t)

sinCosSquareLM=lm(lbeer~t+t2+sin.t+cos.t)

mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
```

1.972046% Accuracy

Output

```
> mean(abs(lbeer-sinCosSquareLM$fit)/lbeer)*100
[1] 1.972046
```

Not always possible to use such imagination

Holt-Winters

- Exponential smoothing in its basic form (the term “exponential” comes from the fact that the weights decay exponentially) should only be used for time series with no systematic trend and/or seasonal components.

- It has been generalized to the “Holt–Winters”–procedure in order to deal with time series containg trend and seasonal variation.

A natural estimate for predicting the next value of a given time series x_t at th
period $t = \tau$ is to take weighted sums of past observations:

$$\hat{x}_{(t=\tau)}(1) = \lambda_0 \cdot x_\tau + \lambda_1 \cdot x_{\tau-1} + \ldots$$

It seems reasonable to weight recent observations more than observations fron
the past. Hence, one possibility is to use geometric weights

$$\lambda_i = \alpha(1-\alpha)^i \qquad ; \quad 0 < \alpha < 1$$

such that $\quad \hat{x}_{(t=\tau)}(1) = \alpha \cdot x_\tau + \alpha(1-\alpha) \cdot x_{\tau-1} + \alpha(1-\alpha)^2 \cdot x_{\tau-2} + \ldots$

Source: Walter Zucchini, Oleg Nenadi (Time Series Analysis with R - Part)

HoltWinters in R

- R contains the function HoltWinters(x,alpha,beta,gamma), which lets you perform the Holt–Winters procedure on a time series x.

- In case one does not specify smoothing parameters, these are determined “automatically” (i.e. by minimizing the mean squared prediction error from one–step forecasts).

Let’s do it in R with the same dataset

```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

hw<-HoltWinters(beer)

predict(hw,n.ahead=12)

plot(beer,xlim=c(1956,1996))

lines(predict(hw,n.ahead=12),col=2)

mean(abs(hw$fit[,1]-beer)/beer)*100
```

R Script

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

hw<-HoltWinters(beer)

predict(hw,n.ahead=12)

plot(beer,xlim=c(1956,1996))

lines(predict(hw,n.ahead=12),col=2)

mean(abs(hw$fit[,1]-beer)/beer)*100
```

This performs the Holt–Winters procedure on the beer – dataset.

R Script

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

hw<-HoltWinters(beer)

predict(hw,n.ahead=12)

plot(beer,xlim=c(1956,1996))

lines(predict(hw,n.ahead=12),col=2)

mean(abs(hw$fit[,1]-beer)/beer)*100
```

This gives us the predicted values for the next 12 periods (i.e. Sep. 1995 to Aug. 1996).

Output

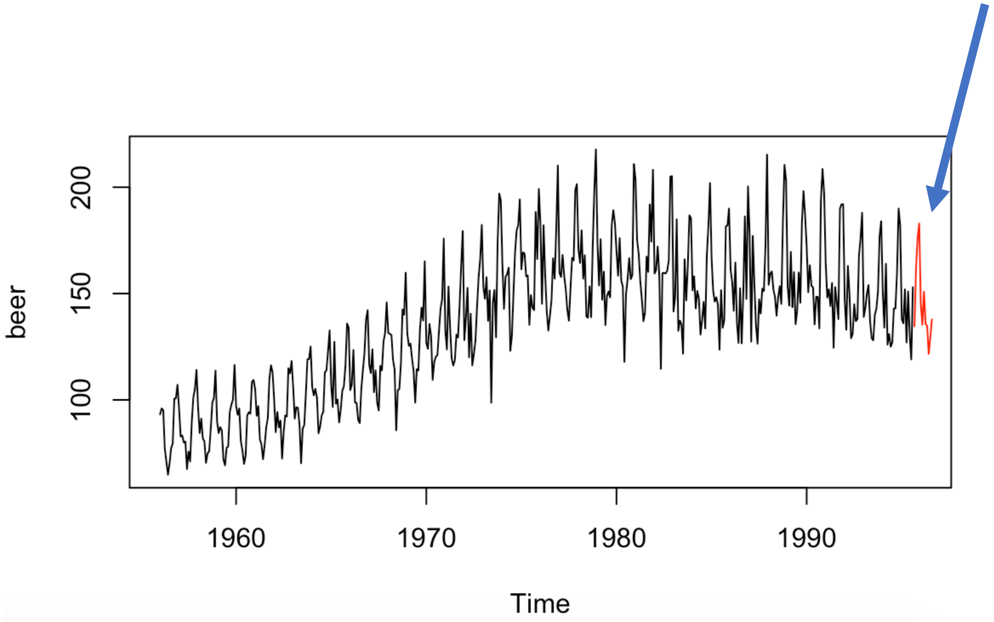
```
> predict(hw,n.ahead=12)
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep      Oct      Nov      Dec
1995                                                                            134.5864 162.1453 176.0870 182.9774
1996 145.0224 135.3375 150.8931 135.7861 134.8204 121.6526 129.1765 137.8240
```

R Script

```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

hw<-HoltWinters(beer)

predict(hw,n.ahead=12)

plot(beer,xlim=c(1956,1996))

lines(predict(hw,n.ahead=12),col=2)

mean(abs(hw$fit[,1]-beer)/beer)*100
```

These commands can be used to create a graph with the predictions for the next 1 year1 (i.e. 12 months):

Output



R Script

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

hw<-HoltWinters(beer)

predict(hw,n.ahead=12)

plot(beer,xlim=c(1956,1996))

lines(predict(hw,n.ahead=12),col=2)

mean(abs(hw$fit[,1]-beer)/beer)*100
```

Accuracy is 5.444085%

Output

```
> mean(abs(hw$fit[,1]-beer)/beer)*100
[1] 5.444085
```

ARIMA Model

- ARIMA stands for Auto Regressive Integrated Moving Average.
- It is specified by three order parameters: **p**, **d**, **q**

ARIMA(p,d,q)

ARIMA models are classified by three factors:

p = number of autoregressive terms (AR)

$$x_t = c_1 x_{t-1} + c_2 x_{t-2} + \ldots + c_p x_{t-p}$$

Auto Regressive (AR only) model is one where x_t depends only on its own lags.

d = how many non-seasonal differences are needed

$$x_t = c_1 x_{t-1} + c_2 (x_{t-2} - x_{t-1}) + c_3 (x_{t-3} - x_{t-2}) + \ldots + c_d (x_{t-d} - x_{t-d})$$

Moving Average (MA only) model is one where x_t depends only on the lagged forecast errors.

q = number of lagged forecast errors in the prediction equation (MA)

As can be seen, the second equation with d can be rewritten as the first equation with p.

Let’s do it in R with the same dataset

```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

ar<-arima(beer,order=c(4,0,0))

ar.predict <- predict(ar,n.ahead=12)

ar.predict
```

R Script

```r
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");

beer<-ts(beer[,1],start=1956,freq=12);

ar<-arima(beer,order=c(4,0,0))

ar.predict <- predict(ar,n.ahead=12)

ar.predict
```

This performs the ARIMA procedure on the beer – dataset.

Here $p = 4$
$d = 0$
$q = 0$

R Script

```
beer<-read.csv("~/Downloads/beer.csv",header=T,dec=",",sep=";");
beer<-ts(beer[,1],start=1956,freq=12);
ar<-arima(beer,order=c(4,0,0))
ar.predict <- predict(ar,n.ahead=12)
ar.predict
```

This gives us the predicted values for the next 12 periods (i.e. Sep. 1995 to Aug. 1996).

Output

```
> ar.predict
$pred
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep      Oct      Nov      Dec
1995                                                                             144.3779 140.4554 143.7611 141.9400
1996 140.8368 140.7840 140.1265 139.6032 139.2611 138.8772 138.5422 138.2600

$se
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep      Oct      Nov      Dec
1995                                                                             18.35513 22.28215 24.43782 26.80246
1996 28.35215 29.47108 30.39402 31.09260 31.63192 32.05982 32.39471 32.65809
```

Let’s try with Electricity Dataset

- Create our own autoregression
  - same hour of the last week
  - same hour yesterday
  - last four hours
- Use these six hours to predict the next hour

Let’s try with Electricity Dataset

Data: electricityPrediction.txt

	V1	V2	V3	V4	V5	V6	V7
1	71.77033	320.57416	401.91388	344.49761	440.19139	440.19139	425.83732
2	62.20096	320.57416	344.49761	440.19139	440.19139	425.83732	397.12919
3	71.77033	358.85167	440.19139	440.19139	425.83732	397.12919	344.49761
4	66.98565	296.65072	440.19139	425.83732	397.12919	344.49761	344.49761
5	66.98565	330.14354	425.83732	397.12919	344.49761	344.49761	311.00478
6	66.98565	287.08134	397.12919	344.49761	344.49761	311.00478	339.71292
7	66.98565	258.37321	344.49761	344.49761	311.00478	339.71292	306.22010
8	66.98565	210.52632	344.49761	311.00478	339.71292	306.22010	315.78947
9	71.77033	196.17225	311.00478	339.71292	306.22010	315.78947	334.92823
10	66.98565	177.03349	339.71292	306.22010	315.78947	334.92823	320.57416
11	71.77033	200.95694	306.22010	315.78947	334.92823	320.57416	315.78947
12	71.77033	162.67943	315.78947	334.92823	320.57416	315.78947	325.35885

Let’s try with Electricity

- Use these six hours to predict the next hour

```r
xy<-read.csv("~/Downloads/electricityPrediction.txt",header=F);

xy

x=xy[,1:6]

y=xy[,7]

rg=lm(V7~.,xy)

rg$coeff

mean(100*abs(y-rg$fit)/y)
```

Taking first 6 columns as x and last column as y

Output

```
> x=xy[,1:6]
> x
        V1        V2        V3        V4        V5        V6
1  71.77033 320.57416 401.91388 344.49761 440.19139 440.19139
2  62.20096 320.57416 344.49761 440.19139 440.19139 425.83732
3  71.77033 358.85167 440.19139 440.19139 425.83732 397.12919
4  66.98565 296.65072 440.19139 425.83732 397.12919 344.49761
5  66.98565 330.14354 425.83732 397.12919 344.49761 344.49761
6  66.98565 287.08134 397.12919 344.49761 344.49761 311.00478
7  66.98565 258.37321 344.49761 344.49761 311.00478 339.71292
8  66.98565 210.52632 344.49761 311.00478 339.71292 306.22010
9  71.77033 196.17225 311.00478 339.71292 306.22010 315.78947
```

Let’s try with Electricity

- Use these six hours to predict the next hour

```
xy<-read.csv("~/Downloads/electricityPrediction.txt",header=F);

xy

x=xy[,1:6]

y=xy[,7]

rg=lm(V7~.,xy)

rg$coeff

mean(100*abs(y-rg$fit)/y)
```

Using all columns except V7 for modeling

Let’s try with Electricity

- Use these six hours to predict the next hour

```r
xy<-read.csv("~/Downloads/electricityPrediction.txt",header=F);

xy

x=xy[,1:6]

y=xy[,7]

rg=lm(V7~.,xy)

rg$coeff

mean(100*abs(y-rg$fit)/y)
```

Output

Extracting Model coefficients

```
> rg$coeff
  (Intercept)            V1            V2            V3            V4            V5            V6 
-0.3804857237  0.2528230532  0.1679550277  0.0345118162  0.0852869403 -0.0006030161  0.4614112202 
```

Let's try with Electricity

- Use these six hours to predict the next hour

```
xy<-read.csv("~/Downloads/electricityPrediction.txt",header=F);

xy

x=xy[,1:6]

y=xy[,7]

rg=lm(V7~.,xy)

rg$coeff

mean(100*abs(y-rg$fit)/y)
```

Accuracy

Output

```
> mean(100*abs(y-rg$fit)/y)
[1] 11.87863
```

Let’s try with Random Forest

```
install.packages("randomForest")
library(randomForest)

rf=randomForest(x,y,nodesize=10000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)

rf=randomForest(x,y,nodesize=1000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)
```

Try with Random Forest and nodesize=10000

Output

Let’s try with Random Forest

```
install.packages("randomForest")
library(randomForest)

rf=randomForest(x,y,nodesize=10000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)

rf=randomForest(x,y,nodesize=1000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)
```

Accuracy: 12.26945

Let’s try with Random Forest

```r
install.packages("randomForest")
library(randomForest)

rf=randomForest(x,y,nodesize=10000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)

rf=randomForest(x,y,nodesize=1000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)
```

Change Node Size to 1000

Let’s try with Random Forest

```r
install.packages("randomForest")
library(randomForest)

rf=randomForest(x,y,nodesize=10000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)

rf=randomForest(x,y,nodesize=1000)

rfp=predict(rf,x)

mean(100*abs(y-rfp)/y)
```

Accuracy: 10.76181

Let us try with train and test set

```
> total=nrow(xy)
> train=0.8*total
> test = 0.2*total
> xyTrain=xy[sample(1:total,train),]
> xyTest=xy[sample(1:total,test),]
> xTrain=xyTrain[,1:6]
> xTest=xyTest[,1:6]
> yTrain=xyTrain[,7]
> yTest=xyTest[,7]
> rg=lm(V7~.,xyTrain)
> mean(100*abs(yTrain-rg$fit)/yTrain)
[1] 11.86982
> ytp=predict(rg,xTest)
> mean(100*abs(yTest-ytp)/yTest)
[1] 11.75274
```

80% for training

20% for testing

Similarly you can also perform using Random Forest

```
rf=randomForest(xTrain,yTrain,nodesize=10000)
rfp=predict(rf,xTrain)
mean(100*abs(yTrain-rfp)/yTrain)
# [1] 12.60685
rfpt=predict(rf,xTest)
mean(100*abs(yTest-rfpt)/yTest)
# [1] 12.81233
rf=randomForest(xTrain,yTrain,nodesize=1000)
mean(100*abs(yTrain-rfp)/yTrain)
# [1] 10.82798
rfpt=predict(rf,xTest)
mean(100*abs(yTest-rfpt)/yTest)
# [1] 10.99167
```

Thank You