Time Series Forecasting - II

Introduction to AR models

In an autoregressive model, the regression technique is used to formulate a time series problem. In order to implement autoregressive models, we forecast the future observations using a linear combination of past observations of the same variable i.e. to predict $y\hat{}_t$ which is the future forecast of a variable, we need one or more past observations of y_t that is y_{t-1}, y_{t-2}, etc. But to implement these techniques, we need to abide by some assumptions. There are two fundamental assumptions to build an autoregressive model. They are -

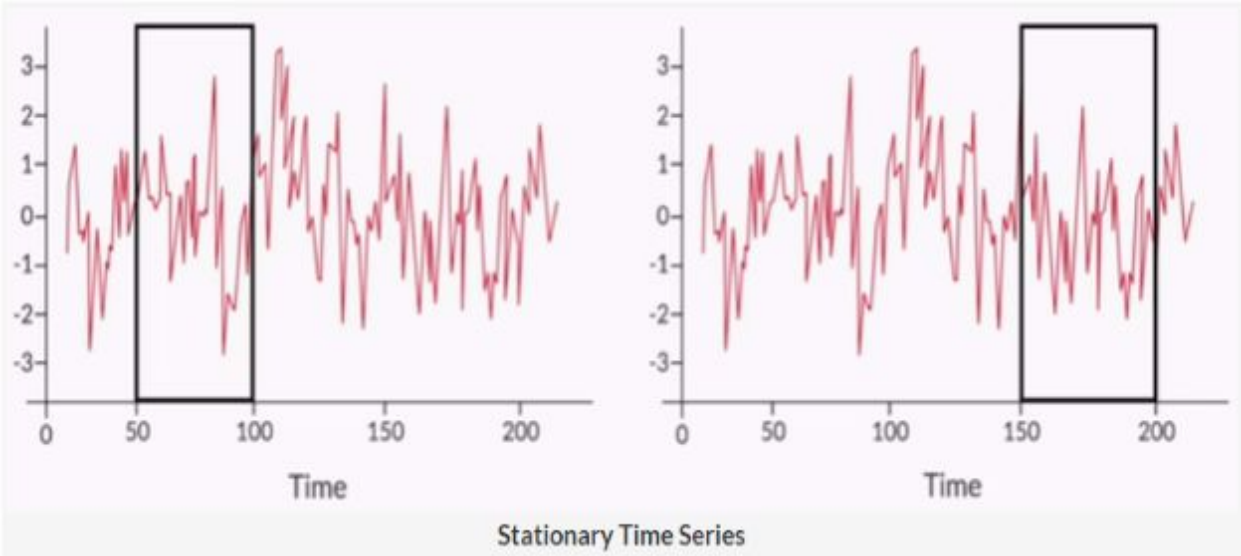
1. Stationarity
2. Autocorrelation

Let us understand both these assumptions one by one. We will also learn how to make a non-stationary time series into a stationary one so as to use the autoregressive models.

Stationarity

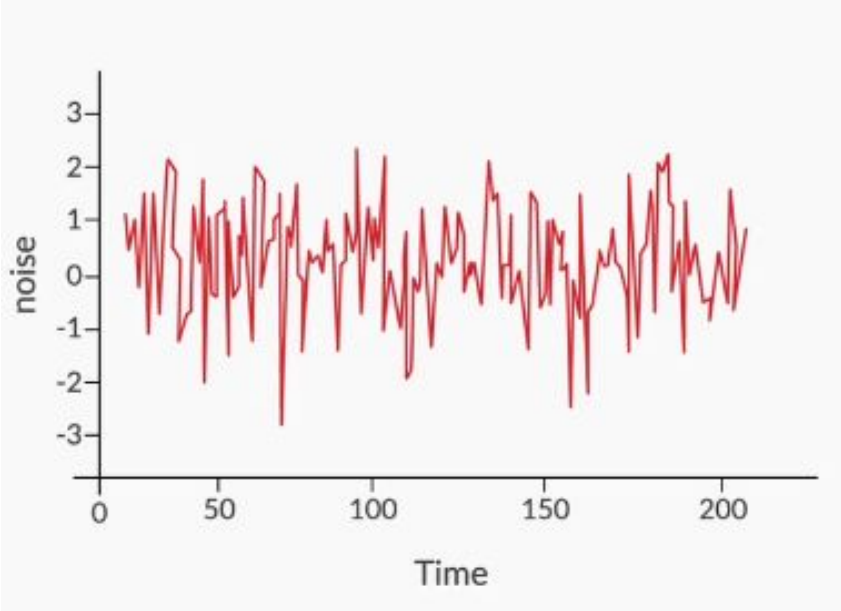
If a time series is stationary, its statistical properties like mean, variance, and covariance will be the same throughout the series, irrespective of the time at which you observe them. As the series will be stationary, the statistical properties such as mean, variance and covariance need to be the same and thus these need to be followed by a series with temporal data also.

The following image illustrates a stationary time series in which the properties such as mean, variance and covariance are the same for any two-time windows.



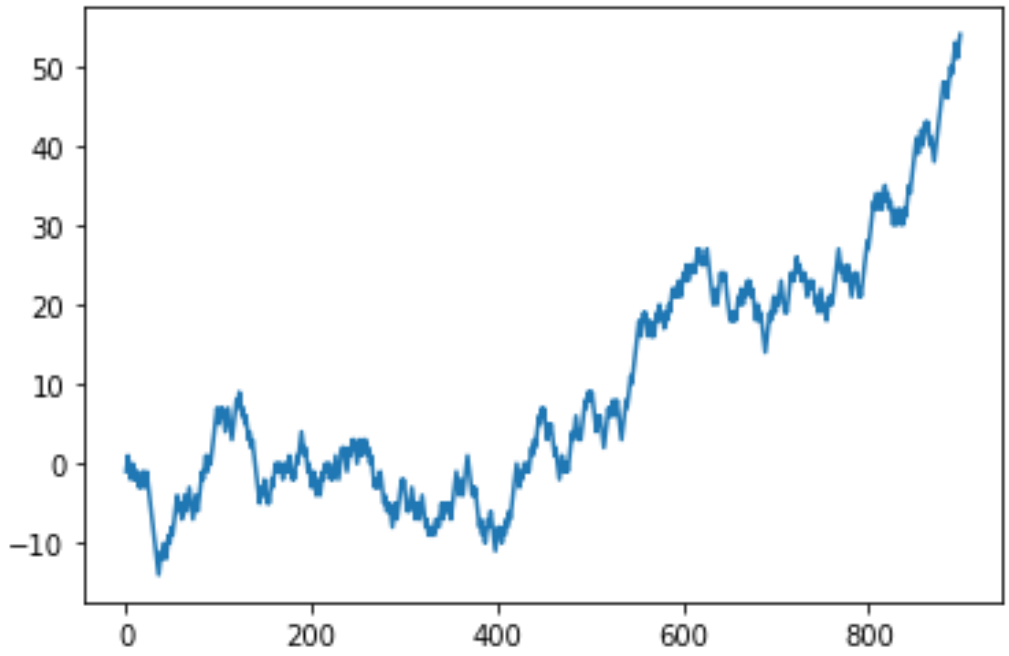
Stationary Time Series

Let us understand now what is white noise. White Noise is an example of a stationary time series with purely random, uncorrelated observations with no identifiable trend, seasonal or cyclical components. A white noise series can seen from the image below



Notice that there are no identifiable trends, seasonal or cyclical components. So a white noise series is basically an example of a stationary series.

The following example of a non-stationary series is called random walk.



A random walk is a time series where the current observation is equal to the previous observation plus a random change. Here, variance increases over time resulting in a non-stationary series.

In general, a stationary time series will have no long-term predictable patterns such as trends or seasonality. Time plots will show the series to roughly have a horizontal trend with constant variance.

Stationarity Tests

Most of the time, visually, you can see that the series has a clear trend or seasonal component to comment about its stationarity. However, you cannot say that for sure as you also have to test whether its statistical properties, such as mean, variance etc. remain constant. This is not possible to comment by inspecting the time series visually. There are some formal tests for checking this.

There are two formal tests for stationarity based on hypothesis testing. The unit root test is a common procedure to determine whether a time series is stationary or not. If the existence of a unit root for a series cannot be rejected, then the series is said to be non-stationary.

Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test

- Null Hypothesis (H_0): The series is stationary
    - $p-value > 0.05$
- Alternate Hypothesis (H_1): The series is not stationary
    - $p-value \leq 0.05$

Augmented Dickey-Fuller (ADF) Test

- Null Hypothesis (H_0): The series is not stationary
    - $p-value > 0.05$
- Alternate Hypothesis (H_1): The series is stationary
    - $p-value \leq 0.05$

Interpretation of p-value.

A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis, otherwise, a p-value above the threshold suggests we fail to reject the null hypothesis.

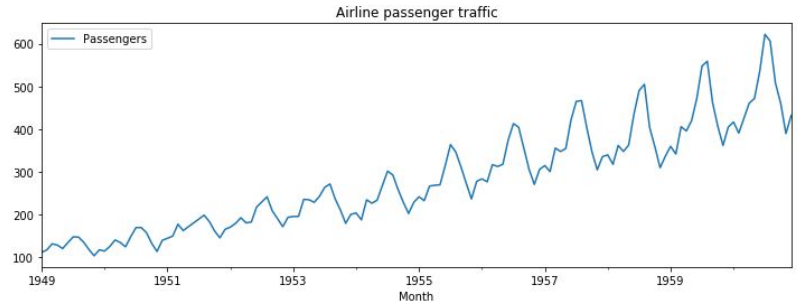
- $p-value > 0.05$: Fail to reject the null hypothesis (H_0).
- $p-value \leq 0.05$: Reject the null hypothesis (H_0).

For the airline passenger traffic example, we need to first check whether the series is stationary or not. We can do that visually inspecting and checking if we can see from the graph ourselves.

Auto Regressive methods

Stationarity vs non-stationary time series

```
In [33]: data['Passengers'].plot(figsize=(12, 4))
         plt.legend(loc='best')
         plt.title('Airline passenger traffic')
         plt.show(block=False)
```



Later, we apply the Augmented Dickey-Fuller (ADF) Test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test as shown in the images below.

Augmented Dickey-Fuller (ADF) test

```
In [34]: from statsmodels.tsa.stattools import adfuller
         adf_test = adfuller(data['Passengers'])

         print('ADF Statistic: %f' % adf_test[0])
         print('Critical Values @ 0.05: %.2f' % adf_test[4]['5%'])
         print('p-value: %f' % adf_test[1])
```

```
ADF Statistic: 0.894609
Critical Values @ 0.05: -2.88
p-value: 0.993020
```

Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

```
In [35]: from statsmodels.tsa.stattools import kpss
         kpss_test = kpss(data['Passengers'])

         print('KPSS Statistic: %f' % kpss_test[0])
         print('Critical Values @ 0.05: %.2f' % kpss_test[3]['5%'])
         print('p-value: %f' % kpss_test[1])
```

On inspecting the ADF test, we know that as per the null hypothesis, the series is not stationary When p−value>0.05. Thus, we know that the time series in the airline passenger data is not a stationary time series and we need to make it stationary first by making the variance and the mean constant.

This is done by the Box-Cox transformation and the Differencing over the time series and will be explained in the sections below.

Converting a Non-Stationary Time-Series to a Stationary Time-Series

For implementing the auto-regressivce models, we discussed that the time-series needs to be stationary. And thus, for a non-stationary time series, we need to first convert it into a stationary one. There are two tools to convert a non-stationary series into stationary series are as follows:

1. Differencing
2. Transformation

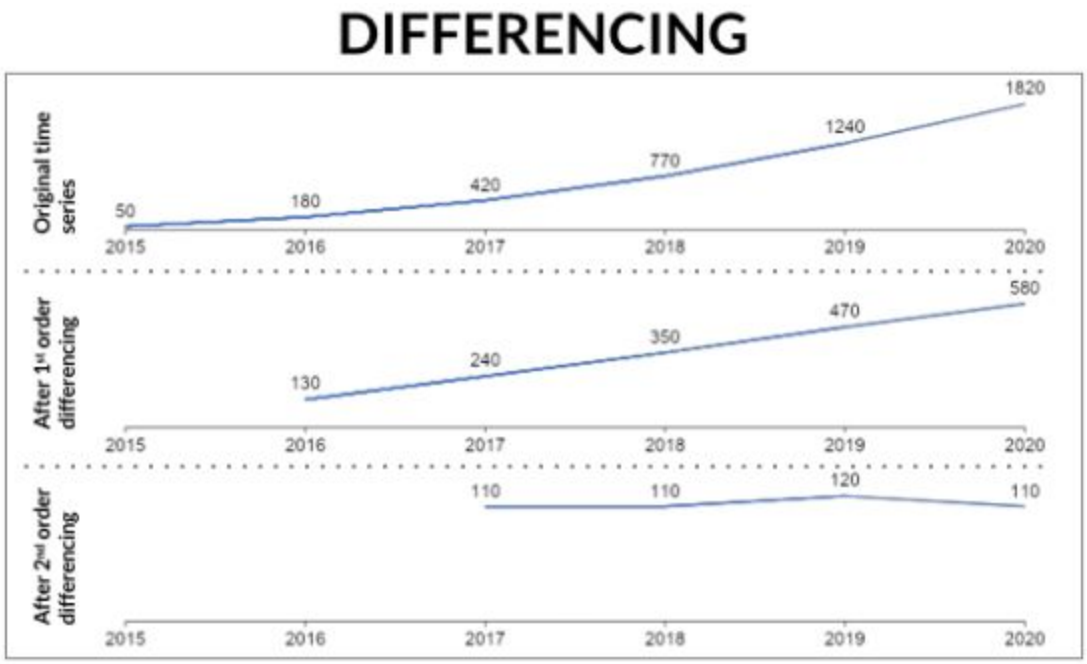
To remove the trend (to make the mean constant) in a time series you use the technique called **differencing**. As the name suggests, in differencing you compute the differences between consecutive observations. It's just like differencing a sloped line so as to make the slope zero. Differencing stabilises the mean of a time series by removing changes in the level of a time series and therefore eliminating (or reducing) trend and seasonality.

For Example:

Original time series	After 1st order differencing	After 2nd order differencing
50		
180	130	
420	240	110
770	350	110
1240	470	120
1820	580	110

Here 1st order differencing is calculated by the difference of two consecutive observations of the original time series. 2nd order differencing is calculated by the difference of two consecutive observations of the 1st order differenced series. This is similar to the difference in the y values

divided by the x values while calculating a slope of a line. You can observe the effect of differencing on an original time series shown in the first image below, and how after the second order differencing, the time series looks like.



At the first difference level, it still has a linear trend but we have successfully removed the higher-order trend observed in the original time series. After the 2nd difference level, the series obtained is mostly a horizontal line. There is no overall visible trend in this series and thus you can say the series obtained is a stationary series.

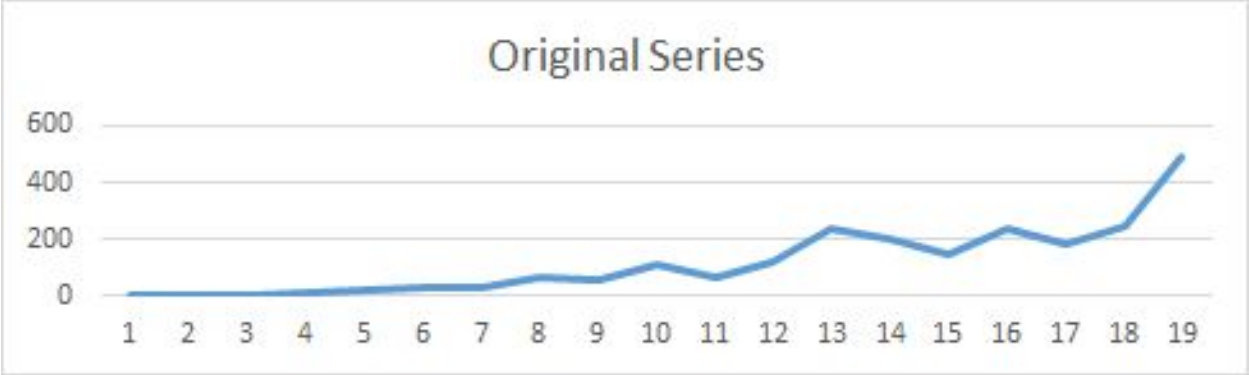
The other method to introduce the stationarity is making the variance constant. There can be many transformation methods used to make a non-stationary series stationary but here, we are discussing the Box-Cox transformation.

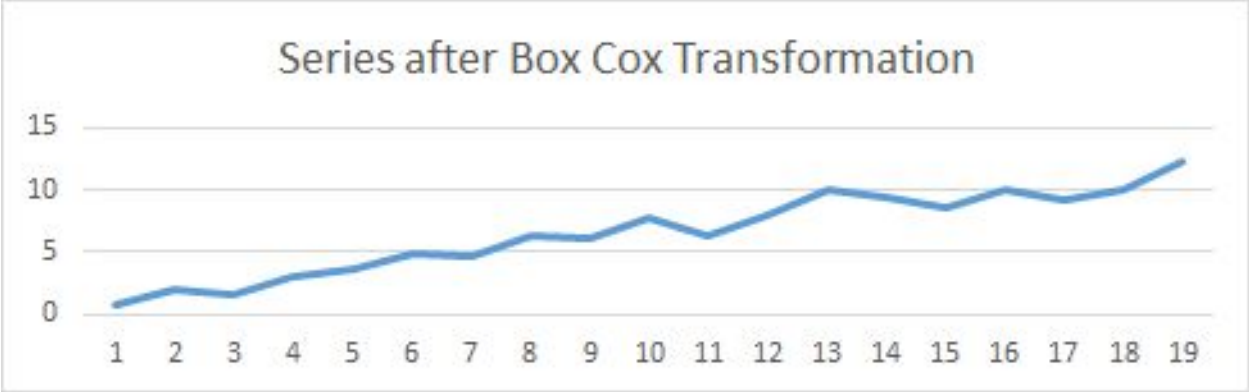
The mathematical formulae that Box-cox transformation is:

$$y(\lambda) = \{\frac{y^{\lambda} - 1}{\lambda}, \text{if} \lambda = 0$$

$$y(\lambda) = \{ \log y, \text{if} \lambda \neq 0$$

where y is the original time series and y(λ) is the transformed series. The procedure for the Box-Cox transformation is to find the optimal value of λ between -5 and 5 that minimizes the variance of the transformed data.

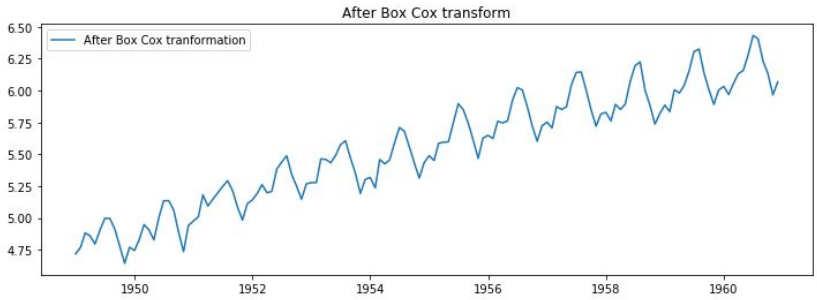




Now, applying the box-cox transformation and differencing the airline passenger dataset in python, we get the following:

Box Cox transformation to make variance constant

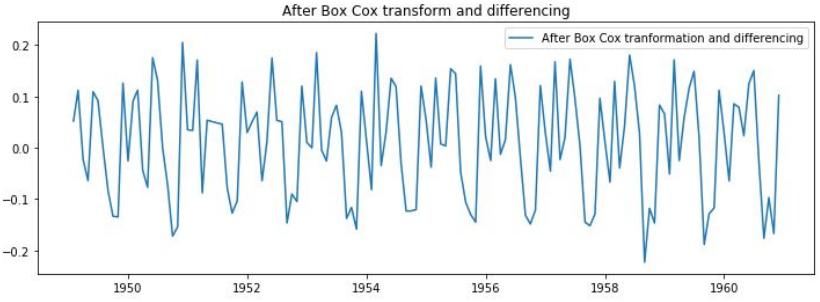
```
In [36]: from scipy.stats import boxcox
data_boxcox = pd.Series(boxcox(data['Passengers'], lmbda=0), index = data.index)

plt.figure(figsize=(12,4))
plt.plot(data_boxcox, label='After Box Cox tranformation')
plt.legend(loc='best')
plt.title('After Box Cox transform')
plt.show()
```



In the above image, you can clearly see that the variance in the magnitude of the time series values as the series progresses is fairly constant. Thus the box-cox transformation is applied successfully.

Differencing to remove trend

```
In [37]: data_boxcox_diff = pd.Series(data_boxcox - data_boxcox.shift(), data.index)
plt.figure(figsize=(12,4))
plt.plot(data_boxcox_diff, label='After Box Cox tranformation and differencing')
plt.legend(loc='best')
plt.title('After Box Cox transform and differencing')
plt.show()
```



```
In [38]: data_boxcox_diff.dropna(inplace=True)

In [39]: data_boxcox_diff.tail()

Out[39]: Month
         1960-08-01   -0.026060
         1960-09-01   -0.176399
         1960-10-01   -0.097083
         1960-11-01   -0.167251
         1960-12-01    0.102279
         dtype: float64
```

Also, the trend in the time series gets removed after the series is differenced using the above steps.

ACF and PACF

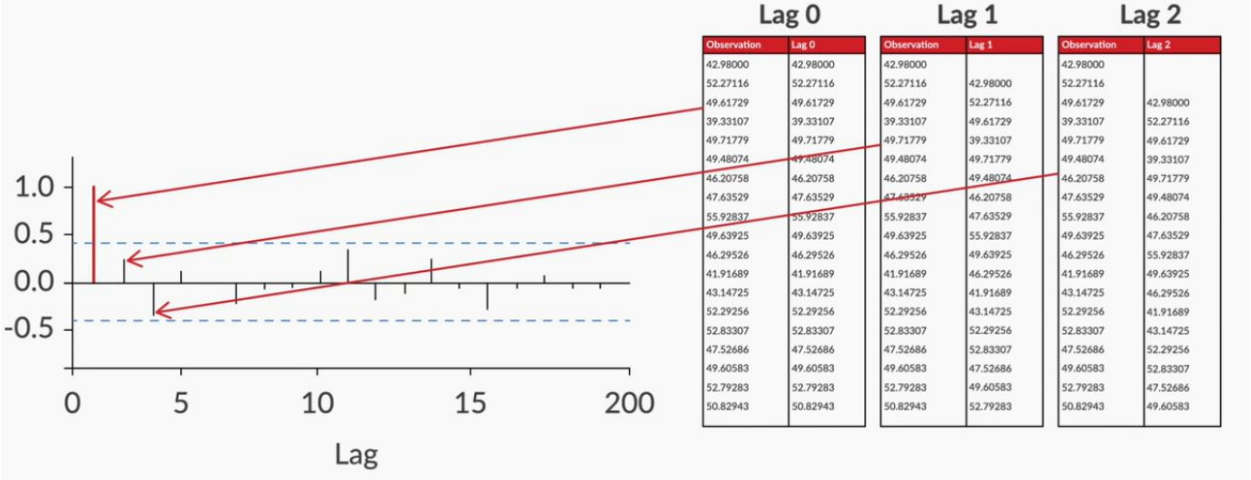
Autocorrelation is capturing the relationship between observations y_t at time t and y_{t-k} at time k time period before t. In simpler words, autocorrelation helps us to know how a variable is influenced by its own lagged values. We looked at two Autocorrelation measures here:

1. Autocorrelation function (ACF)
2. Partial autocorrelation function (PACF)

The autocorrelation function tells about the correlation between an observation with its lagged values. It helps you to determine which lag of the observation is influencing it the most.

Timestamp	Observation	Lag 1	Lag 2
T1	42.98000		
T2	52.27116	42.98000	
T3	49.61729	52.27116	42.98000
T4	39.33107	49.61729	52.27116
T5	49.71779	39.33107	49.61729
T6	49.48074	49.71779	39.33107
T7	46.20758	49.48074	49.71779
T8	47.63529	46.20758	49.48074
T9	55.92837	47.63529	46.20758
T10	49.63925	55.92837	47.63529
T11	46.29526	49.63925	55.92837
T12	41.91689	46.29526	49.63925
T13	43.14725	41.91689	46.29526
T14	52.29256	43.14725	41.91689
T15	52.83307	52.29256	43.14725
T16	47.52686	52.83307	52.29256
T17	49.60583	47.52686	52.83307
T18	52.79283	49.60583	47.52686
T19	50.82943	52.79283	49.60583

In the above example, we see the time series with lag 1 and lag 2 of its original time series. Now let's see the autocorrelation plot of the same data.



Lag 0

Observation	Lag 0
42.98000	42.98000
52.27116	52.27116
49.61729	49.61729
39.33107	39.33107
49.71779	49.71779
49.48074	49.48074
46.20758	46.20758
47.63529	47.63529
55.92837	55.92837
49.63925	49.63925
46.29526	46.29526
41.91689	41.91689
43.14725	43.14725
52.29256	52.29256
52.83307	52.83307
47.52686	47.52686
49.60583	49.60583
52.79283	52.79283
50.82943	50.82943

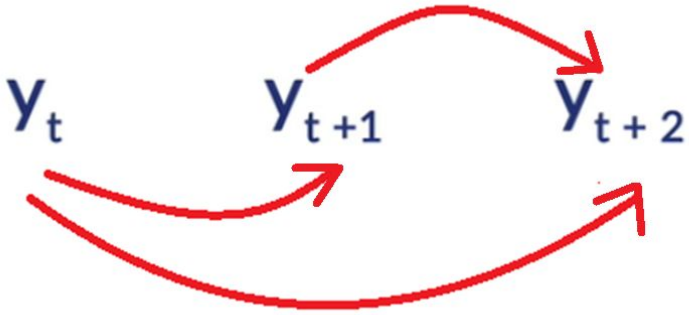
Lag 1

Observation	Lag 1
42.98000	
52.27116	42.98000
49.61729	52.27116
39.33107	49.61729
49.71779	39.33107
49.48074	49.71779
46.20758	49.48074
47.63529	46.20758
55.92837	47.63529
49.63925	55.92837
46.29526	49.63925
41.91689	46.29526
43.14725	41.91689
52.29256	43.14725
52.83307	52.29256
47.52686	52.83307
49.60583	47.52686
52.79283	49.60583
50.82943	52.79283

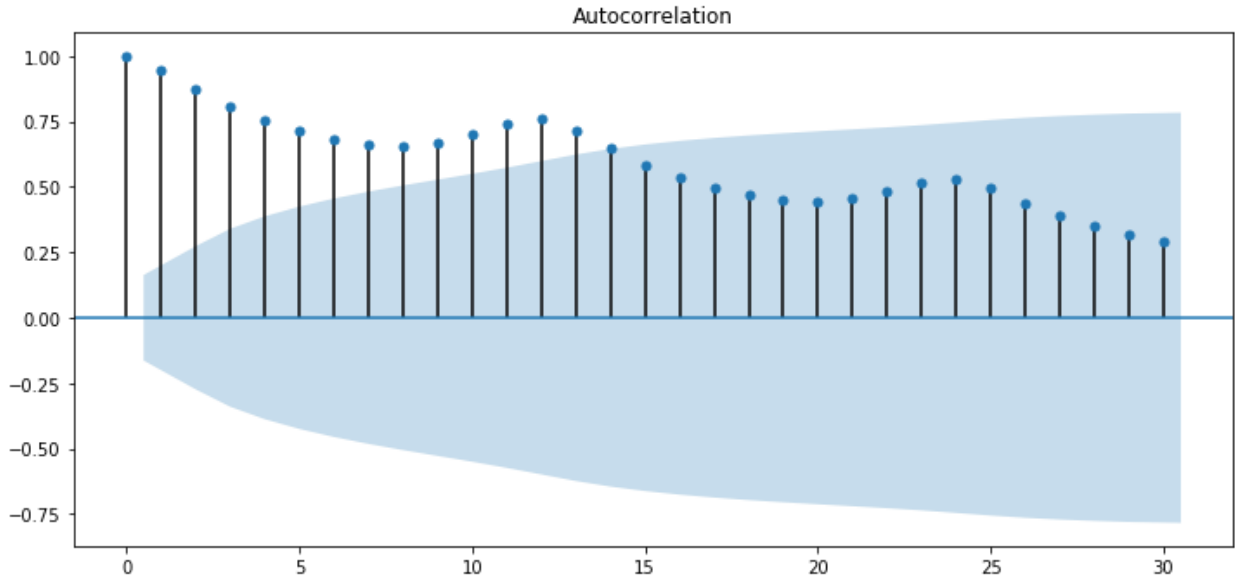
Lag 2

Observation	Lag 2
42.98000	
52.27116	
49.61729	42.98000
39.33107	52.27116
49.71779	49.61729
49.48074	39.33107
46.20758	49.71779
47.63529	49.48074
55.92837	46.20758
49.63925	47.63529
46.29526	55.92837
41.91689	49.63925
43.14725	46.29526
52.29256	41.91689
52.83307	43.14725
47.52686	52.29256
49.60583	52.83307
52.79283	47.52686
50.82943	49.60583

Here, we clearly see that the current observation is significantly correlated with lag 1 and lag 2. The other interesting thing to notice is that the autocorrelation function captures both direct and indirect relationships between the variables. For example:



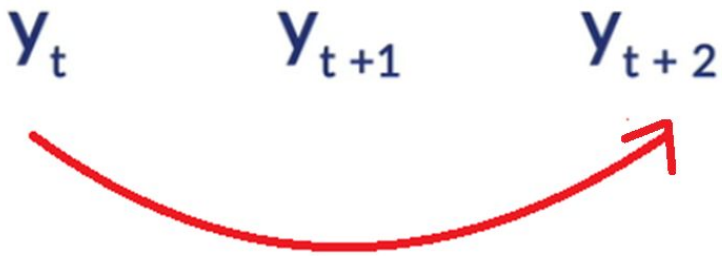
An ACF plot shown in an example taken by the SME is shown below



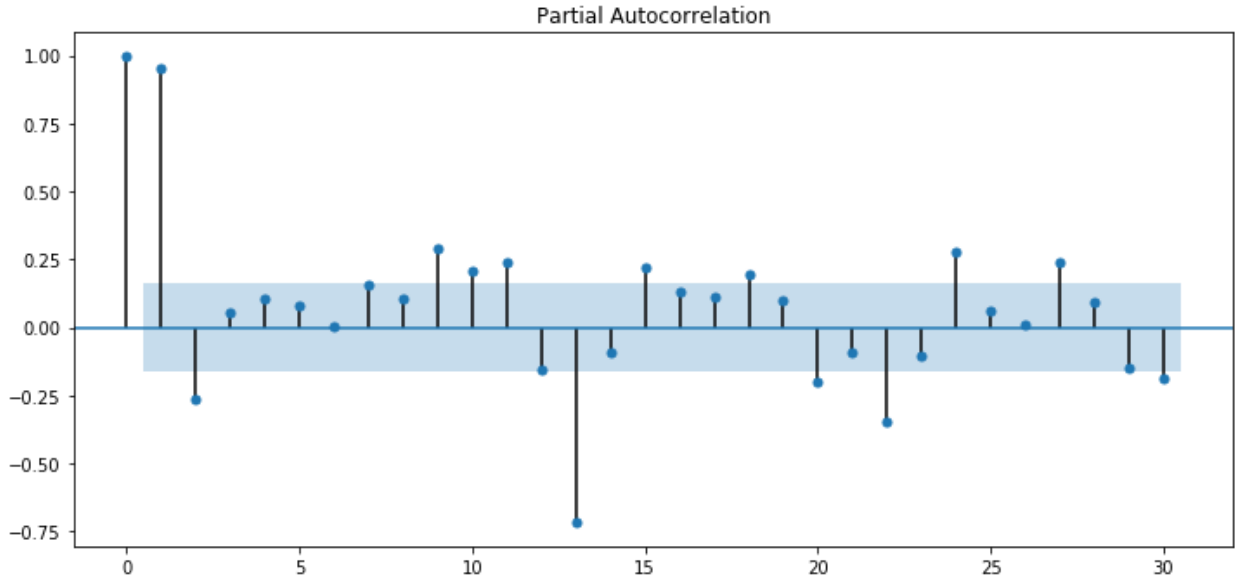
For y_t, y_{t+1}, y_{t+2}**:** Autocorrelation function captures both direct and indirect relationship with its lagged values. Here, the big arrow on the bottom indicates the direct relationship that is captured between y_t, y_{t+2}

The autocorrelation function also captures the indirect relationship between y_t and y_{t+2} through y_{t+1}. In simpler words, y_t will have some correlation with y_{t+1}, and y_{t+1} will also have a correlation with y_{t+2}. This transitive correlation that passes through y_{t+1} is the indirect relationship which is also captured by the Autocorrelation function.

Thus, you can't differentiate out only the direct relationship using ACF. To capture only direct relationships, you have another measure called Partial Autocorrelation Function or PACF.



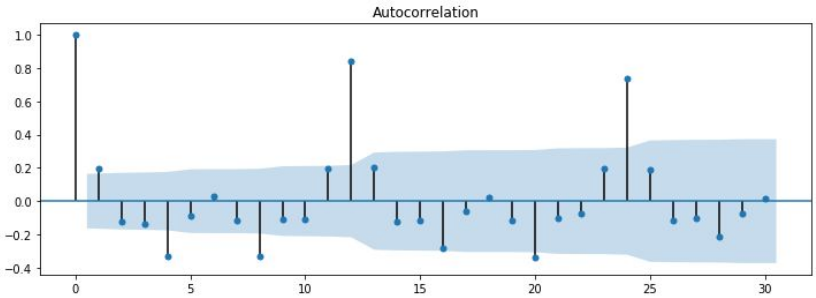
A PACF plot shown in an example taken by the SME is shown below



For the airline passenger traffic dataset, the ACF and the PACF plots can be plotted in python as shown below.

Autocorrelation function (ACF)

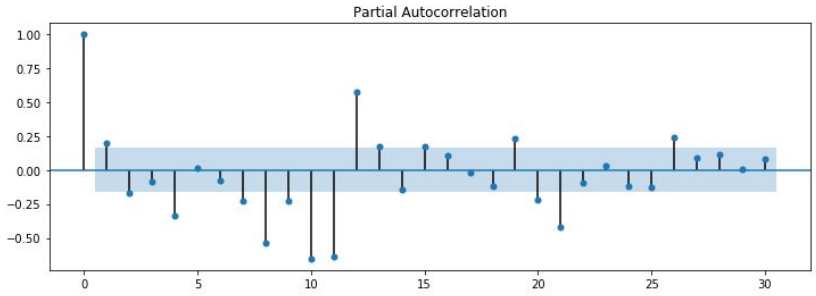
```
In [42]: from statsmodels.graphics.tsaplots import plot_acf
         plt.figure(figsize=(12,4))
         plot_acf(data_boxcox_diff, ax=plt.gca(), lags = 30)
         plt.show()
```



In the ACF plot above, you can see the plot for 30 lags in the dataset. We decide the window size, that is q using the above plot. As you can see, the highest lag after which the autocorrelation dies down is 1. Thus q =1 will be suitable for the airline passenger problem.

Partial autocorrelation function (PACF)

In [43]:
```
from statsmodels.graphics.tsaplots import plot_pacf
plt.figure(figsize=(12,4))
plot_pacf(data_boxcox_diff, ax=plt.gca(), lags = 30)
plt.show()
```



In the PACF plot above, the highest lag for which the partial autocorrelation is significantly high is the p = 1. Thus this value of p will be suitable while using the ARIMA techniques in the airline passenger problem.

We performed the box cox transformation on the entire dataset and not just the train data. Thus, before starting with the models, we need to split the box cox data into train and test and that is shown below.

In [44]:
```
train_data_boxcox = data_boxcox[:train_len]
test_data_boxcox = data_boxcox[train_len:]
train_data_boxcox_diff = data_boxcox_diff[:train_len-1]
test_data_boxcox_diff = data_boxcox_diff[train_len-1:]
```

In [45]:
```
train_data_boxcox_diff
```

Out[45]:
```
Month
1949-02-01    0.052186
1949-03-01    0.112117
1949-04-01   -0.022990
1949-05-01   -0.064022
1949-06-01    0.109484
                ...
1958-08-01    0.028114
1958-09-01   -0.223144
1958-10-01   -0.118092
1958-11-01   -0.146750
1958-12-01    0.083511
Length: 119, dtype: float64
```

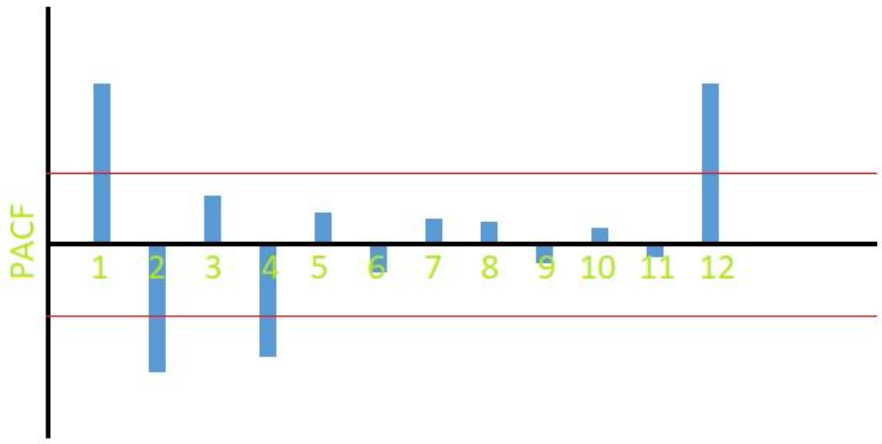
The Simple Auto Regressive Model (AR)

The Simple Auto Regressive model predicts the future observation as linear regression of one or more past observations. In simpler terms, the simple autoregressive model forecasts the dependent variable (future observation) when one or more independent variables are known (past observations). This model has a parameter **'p'** called **lag order**. Lag order is the maximum number of lags used to build 'p' number of past data points to predict future data points.

Example: Consider an example of forecasting monthly sales of ice cream for the year 2021 on the basis of the previous 3 years' monthly sales data of the ice cream. This can be one of the simple autoregressive models.

To determine the value of parameter 'p'.

- Plot partial autocorrelation function



- Select p as the highest lag where partial autocorrelation is significantly high

Here, the lag value of 1, 2, 4 and 12 has a significant level of confidence. i.e., significant level of influence on future observation (refer to the red line). Hence, the value of 'p' will be set to 12 since that is the highest lag where partial autocorrelation is significantly high.

- Build autoregression model equation:

$$\hat{y} = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \beta_4 y_{t-4} + \beta_{12} y_{t-12}$$

The past values which have a significant value are 1, 2, 4 and 12. Therefore, in the regression model the independent variables y_{t-1}, y_{t-2}, y_{t-4} and y_{t-12} which are the observations from the past have been taken to predict the dependent variable $\hat{y}_t$.

Now let us get back again to the airline passenger dataset and build an AR model on it. In order to build the AR model, the stationary series is divided into train and test data.

The following steps are performed on the airline passenger data to build the AR model. The forecast plot is then calculated along with the MAPE value and compared with the previously built models.

Auto regression method (AR)

In [46]:
```
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(train_data_boxcox_diff, order=(1, 0, 0))
model_fit = model.fit()
print(model_fit.params)
```

```
const      0.009477
ar.L1.y    0.183116
dtype: float64
```

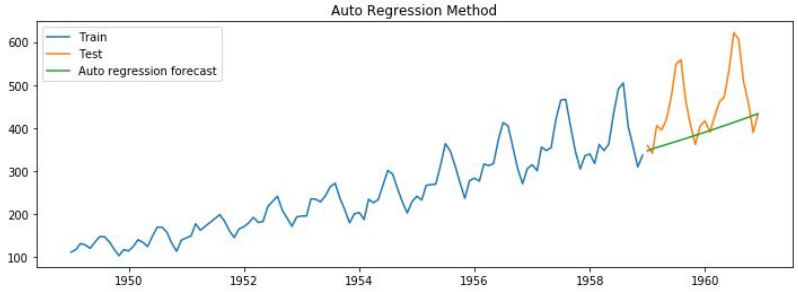
```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

Recover original time series

In [47]:
```
y_hat_ar = data_boxcox_diff.copy()
y_hat_ar['ar_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.index.min(), data_boxcox_diff.index.max())
y_hat_ar['ar_forecast_boxcox'] = y_hat_ar['ar_forecast_boxcox_diff'].cumsum()
y_hat_ar['ar_forecast_boxcox'] = y_hat_ar['ar_forecast_boxcox'].add(data_boxcox[0])
y_hat_ar['ar_forecast'] = np.exp(y_hat_ar['ar_forecast_boxcox'])
```

Plot train, test and forecast

In [48]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_ar['ar_forecast'][test.index.min():], label='Auto regression forecast')
plt.legend(loc='best')
plt.title('Auto Regression Method')
plt.show()
```



Calculate RMSE and MAPE

In [49]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_ar['ar_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_ar['ar_forecast'][test.index.min():])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Autoregressive (AR) method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[49]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91
0	Autoregressive (AR) method	93.39	13.77

Moving Average Model (MA)

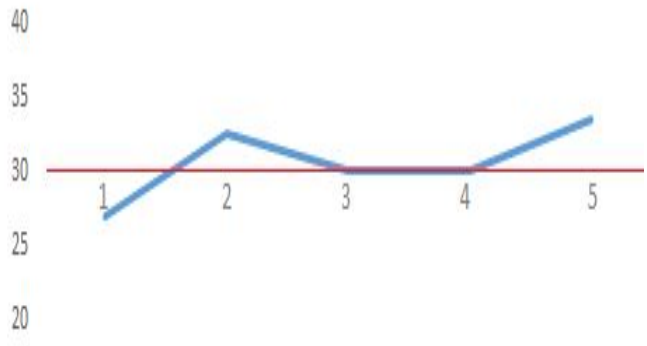
The Moving Average Model models the future forecasts using past forecast errors in a regression-like model. This model has a parameter 'q' called window size over which linear combination of errors are calculated.

Example: Forecast daily ice cream sales for the next 5 days when on an average daily ice cream sale is 30.

t	$\hat{y}_t$	ε_t	y_t
1	30	-3	27
2	28.5	4	32.5
3	32	-2	30
4	29	1	30
5	30.5	3	33.5

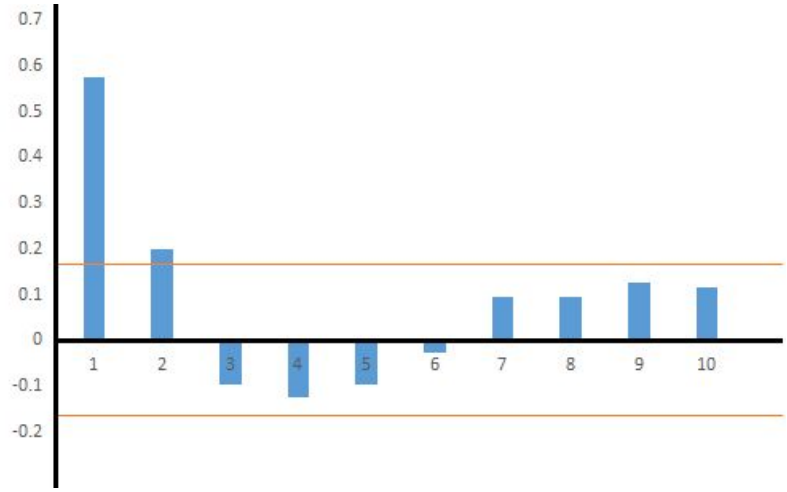
MA(1) forecast is given by $\hat{y}_t = \mu + \Phi_1 \varepsilon_{t-1}$ where $\mu = 10$, $\Phi_1 = 0.5$

Explanation: On the first day you predicted the sale to be the average sale that is 30 but actual demand came to be around 27. There is an error of -3. For the next day, you predicted the sale of ice cream to be mean along with a percentage of error of previous day prediction. i.e., 30+0.5(-3) = 28.5. Similarly, you calculate the forecast for the rest of the days. The prediction of sales of ice cream is moving around the overall mean 30. For this reason, the model is called the moving average (MA) model. We can look at the following plot to confirm this.



Now, in order to build the moving average model, you need to determine the value of parameter 'q'. Let's follow the steps below to do that.

- Plot the Autocorrelation function (ACF)



- Select q as the highest lag beyond which autocorrelation dies down:

Here, for lag 1 and lag 2 the autocorrelation is above significance level. Select q=2 as it is the highest lag beyond which autocorrelation dies down.

- Build Moving Average model equations as:

$$\hat{y}_t = \mu + \Phi_1 \varepsilon_{t-1} + \Phi_2 \varepsilon_{t-2}$$

The following steps are performed on the airline passenger data to build the MA model. The forecast plot is then calculated along with the MAPE value and compared with the previously built models.

Moving average method (MA)

In [50]:
```
model = ARIMA(train_data_boxcox_diff, order=(0, 0, 1))
model_fit = model.fit()
print(model_fit.params)
```

```
const      0.009538
ma.L1.y    0.266103
dtype: float64
```

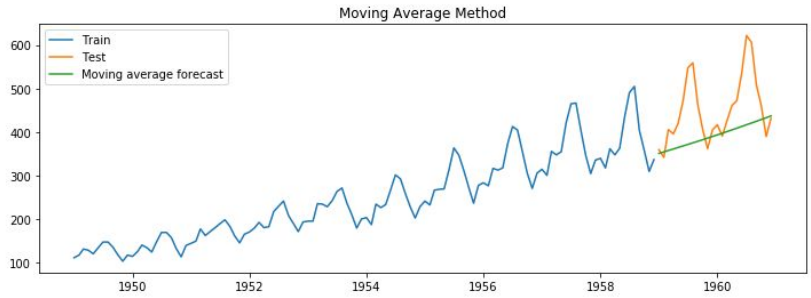
```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

Recover original time series

In [51]:
```
y_hat_ma = data_boxcox_diff.copy()
y_hat_ma['ma_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.index.min(), data_boxcox_diff.index.max())
y_hat_ma['ma_forecast_boxcox'] = y_hat_ma['ma_forecast_boxcox_diff'].cumsum()
y_hat_ma['ma_forecast_boxcox'] = y_hat_ma['ma_forecast_boxcox'].add(data_boxcox[0])
y_hat_ma['ma_forecast'] = np.exp(y_hat_ma['ma_forecast_boxcox'])
```

Plot train, test and forecast

In [52]:
```
plt.figure(figsize=(12,4))
plt.plot(data['Passengers'][:train_len], label='Train')
plt.plot(data['Passengers'][train_len:], label='Test')
plt.plot(y_hat_ma['ma_forecast'][test.index.min():], label='Moving average forecast')
plt.legend(loc='best')
plt.title('Moving Average Method')
plt.show()
```



Calculate RMSE and MAPE

In [53]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_ma['ma_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_ma['ma_forecast'][test.index.min():])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Moving Average (MA) method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[53]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91
0	Autoregressive (AR) method	93.39	13.77
0	Moving Average (MA) method	91.21	13.39

Auto Regressive Moving Average (ARMA)

A time series that exhibits the characteristics of an AR(p) and/or an MA(q) process can be modelled using an ARMA(p,q) model.

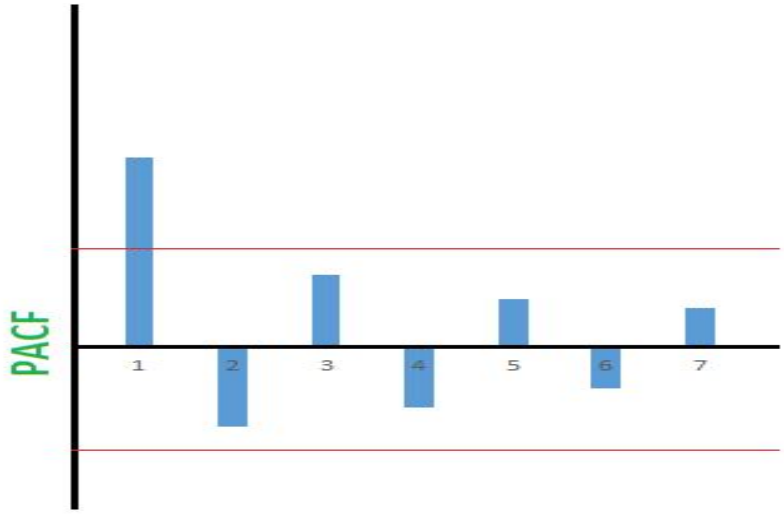
ARMA (1,1) equation: For p = 1 and q=1 —

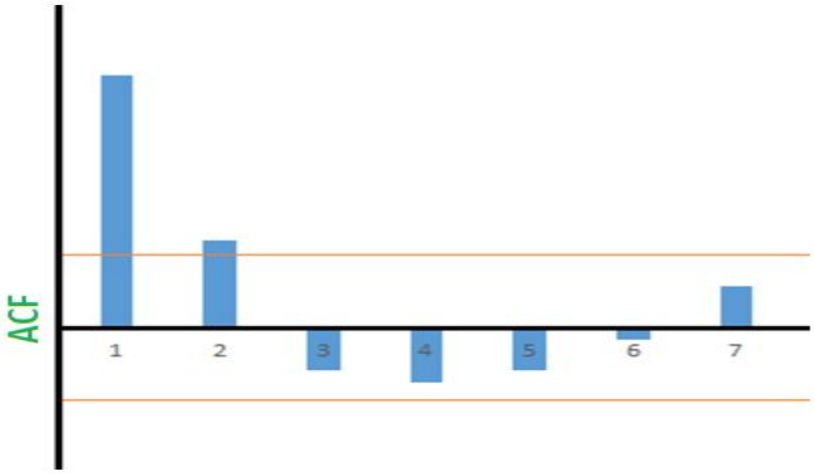
$$\hat{y} = \beta_0 + \beta_1 S_{t-1} + \phi_1 \varepsilon_{t-1}$$

Here, ^y is the forecasted value.

To determine the parameters ‘p’ and ‘q’ —

- Plot autocorrelation function (ACF) and partial autocorrelation function (PACF)





- If you check the plot for PACF, you will see that you need to select p = 1 as the highest lag where partial autocorrelation is significantly high.
- Similarly from the ACF plot, select q = 2 as the highest lag beyond which autocorrelation dies down.
- So, we would select an ARMA (1, 2) model in this example.

The following steps are performed on the airline passenger data to build the ARMA model. The forecast plot is then calculated along with the MAPE value and compared with the previously built models.

Auto regression moving average method (ARMA)

In [54]:
```
model = ARIMA(train_data_boxcox_diff, order=(1, 0, 1))
model_fit = model.fit()
print(model_fit.params)
```

```
const      0.009624
ar.L1.y   -0.527115
ma.L1.y    0.798281
dtype: float64
```

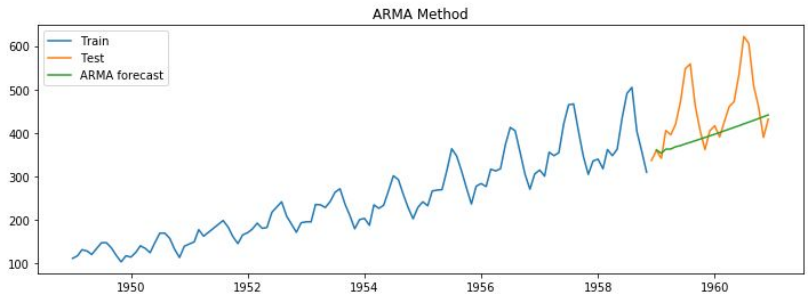
```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

Recover original time series

In [55]:
```
y_hat_arma = data_boxcox_diff.copy()
y_hat_arma['arma_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.index.min(), data_boxcox_diff.index.max())
y_hat_arma['arma_forecast_boxcox'] = y_hat_arma['arma_forecast_boxcox_diff'].cumsum()
y_hat_arma['arma_forecast_boxcox'] = y_hat_arma['arma_forecast_boxcox'].add(data_boxcox[0])
y_hat_arma['arma_forecast'] = np.exp(y_hat_arma['arma_forecast_boxcox'])
```

Plot train, test and forecast

In [56]:
```
plt.figure(figsize=(12,4))
plt.plot( data['Passengers'][:train_len-1], label='Train')
plt.plot(data['Passengers'][train_len-1:], label='Test')
plt.plot(y_hat_arma['arma_forecast'][test.index.min():], label='ARMA forecast')
plt.legend(loc='best')
plt.title('ARMA Method')
plt.show()
```



Calculate RMSE and MAPE

In [57]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_arma['arma_forecast'][train_len-1:])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_arma['arma_forecast'][train_len-1:])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Autoregressive moving average (ARMA) method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[57]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91
0	Autoregressive (AR) method	93.39	13.77
0	Moving Average (MA) method	91.21	13.39
0	Autoregressive moving average (ARMA) method	88.88	12.89

Auto Regressive Integrated Moving Average (ARIMA)

Steps of ARIMA model

- Original time series is differenced to make it stationary
- Differenced series is modeled as a linear regression of
    - One or more past observations
    - Past forecast errors
- ARIMA model has three parameters
    - **p:** Highest lag included in the regression model
    - **d:** Degree of differencing to make the series stationary
    - **q:** Number of past error terms included in the regression model
    - Here the new parameter introduced is the 'I' part called integrated. It removes the trend (non-stationarity) and later integrates the trend to the original series.

So if you think about it, ARIMA is nothing different from what you have done so far. Initially you applied both the box cox transformation and differencing in order to convert the data into a stationary time-series data. Here, you are just applying box cox before building the model and letting the model take care of the differencing i.e. the trend component itself.

Let's now quickly revisit the equations.

ARIMA(1,1,1) Equations:

$$Z_t = \phi_1 Z_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

where

$$Z_t = y_{t+1} - y_t$$

Here, Z_t is the first order differencing for time series.

To determine the parameters 'p', 'd' and 'q'

- For 'd': Select d as the order of difference required to make the original time series stationary. We can verify if this differenced series is stationary or not by using the stationarity tests: ADF or KPSS test.
- For 'p' and 'q': Plot ACF and PACF of the 1st order differenced time series. Find the value of 'p' and 'q' as discussed previously with the earlier Auto Regressive Models.
- The last step in the ARIMA model is to recover the original time series forecast.

The following steps are performed on the airline passenger data to build the ARIMA model. The forecast plot is then calculated along with the MAPE value and compared with the previously built models.

Auto regressive integrated moving average (ARIMA)

In [58]:
```
model = ARIMA(train_data_boxcox, order=(1, 1, 1))
model_fit = model.fit()
print(model_fit.params)
```

```
const        0.009624
ar.L1.D.y   -0.527115
ma.L1.D.y    0.798281
dtype: float64
```

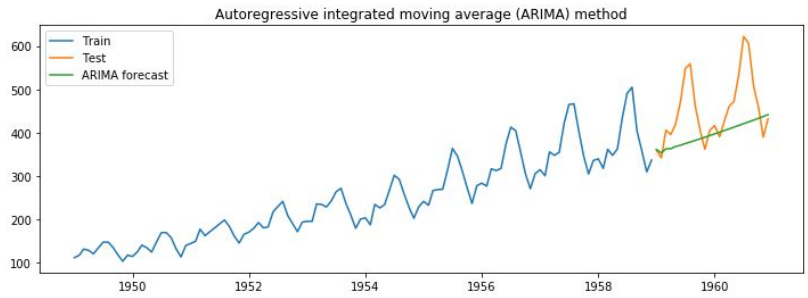
```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

Recover original time series forecast

In [59]:
```
y_hat_arima = data_boxcox_diff.copy()
y_hat_arima['arima_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.index.min(), data_boxcox_diff.index.max())
y_hat_arima['arima_forecast_boxcox'] = y_hat_arima['arima_forecast_boxcox_diff'].cumsum()
y_hat_arima['arima_forecast_boxcox'] = y_hat_arima['arima_forecast_boxcox'].add(data_boxcox[0])
y_hat_arima['arima_forecast'] = np.exp(y_hat_arima['arima_forecast_boxcox'])
```

Plot train, test and forecast

In [60]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_arima['arima_forecast'][test.index.min():], label='ARIMA forecast')
plt.legend(loc='best')
plt.title('Autoregressive integrated moving average (ARIMA) method')
plt.show()
```



Calculate RMSE and MAPE

In [61]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_arima['arima_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_arima['arima_forecast'][test.index.min():])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Autoregressive integrated moving average (ARIMA) method'], 'RMSE': [rmse],'MAPE': [mape] }
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[61]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91
0	Autoregressive (AR) method	93.39	13.77
0	Moving Average (MA) method	91.21	13.39
0	Autoregressive moving average (ARMA) method	88.88	12.89
0	Autoregressive integrated moving average (ARIM...	88.88	12.89

Seasonal Auto Regressive Integrated Moving Average (SARIMA)

SARIMA brings all the features of an ARIMA model with an extra feature - seasonality.

The non-seasonal elements of SARIMA

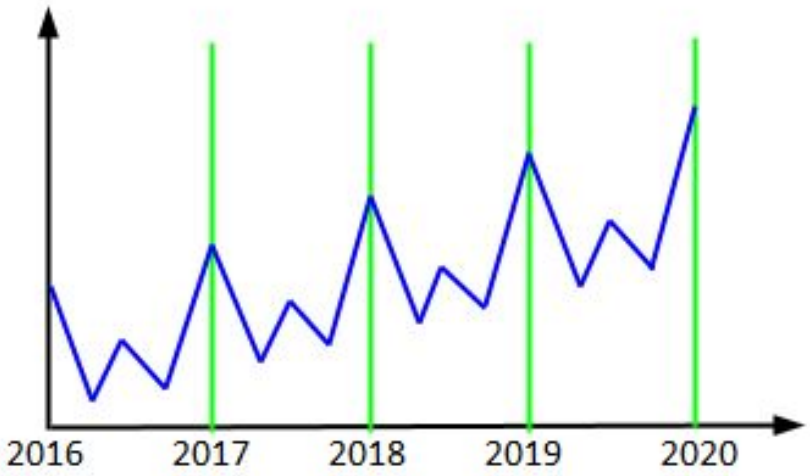
- Time series is differenced to make it stationary.
- Models future observation as linear regression of past observations and past forecast errors.

The seasonal elements of SARIMA

- Perform seasonal differencing on time series.
- Model future seasonality as linear regression of past observations of seasonality and past forecast errors of seasonality.

Example:

To forecast quarterly ice cream sales of 2020 using the Quarterly ice cream sales data for the last 4 years.



Explanation:

- Historical Sales follow a quarterly seasonality and in four years we get 16 data points.
- Future sales is related to past sales with lag = 4
- Sales over the last 4 years is steadily increasing

The above three points make clear that this example has a seasonal component.

The parameters 'p', 'd', 'q' and 'P', 'D', 'Q':

- Non-seasonal elements
  - **p:** Trend autoregression order
  - **d:** Trend difference order
  - **q:** Trend moving average order
- Seasonal elements
  - **m:** The number of time steps for a single seasonal period
  - **P:** Seasonal autoregressive order
  - **D:** Seasonal difference order
  - **Q:** Seasonal moving average order

Equation for SARIMA$(1,0,0)(0,1,1)_4$:

$$z_t = \beta z_{t-1} + \phi \varepsilon_{t-4} + \varepsilon_t \quad \text{where}$$

$$z_t = y_t - y_{t-4}$$

The following steps are performed on the airline passenger data to build the SARIMA model. The forecast plot is then calculated along with the MAPE value and compared with the previously built models.

Seasonal auto regressive integrated moving average (SARIMA)

In [62]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(train_data_boxcox, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
model_fit = model.fit()
print(model_fit.params)
```

```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was
provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

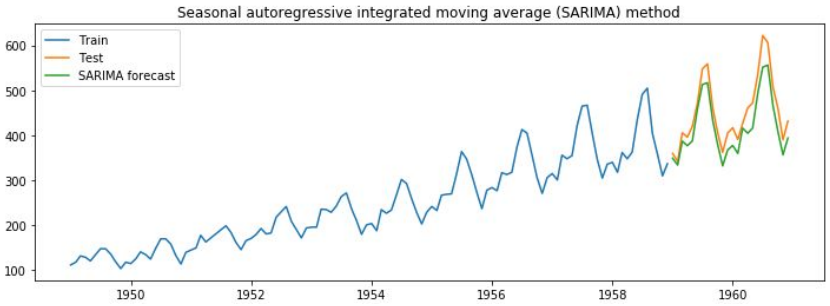
```
ar.L1       0.388460
ma.L1      -0.678612
ar.S.L12   -0.043682
ma.S.L12   -0.516362
sigma2      0.001405
dtype: float64
```

Recover original time series forecast

In [63]:
```
y_hat_sarima = data_boxcox_diff.copy()
y_hat_sarima['sarima_forecast_boxcox'] = model_fit.predict(data_boxcox_diff.index.min(), data_boxcox_diff.index.max())
y_hat_sarima['sarima_forecast'] = np.exp(y_hat_sarima['sarima_forecast_boxcox'])
```

Plot train, test and forecast

In [64]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_sarima['sarima_forecast'][test.index.min():], label='SARIMA forecast')
plt.legend(loc='best')
plt.title('Seasonal autoregressive integrated moving average (SARIMA) method')
plt.show()
```



Calculate RMSE and MAPE

In [65]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_sarima['sarima_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_sarima['sarima_forecast'][test.index.min():])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Seasonal autoregressive integrated moving average (SARIMA) method'], 'RMSE': [rmse],'MAPE
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[65]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91
0	Autoregressive (AR) method	93.39	13.77
0	Moving Average (MA) method	91.21	13.39
0	Autoregressive moving average (ARMA) method	88.88	12.89
0	Autoregressive integrated moving average (ARIM...	88.88	12.89
0	Seasonal autoregressive integrated moving aver...	37.54	7.37

SARIMAX

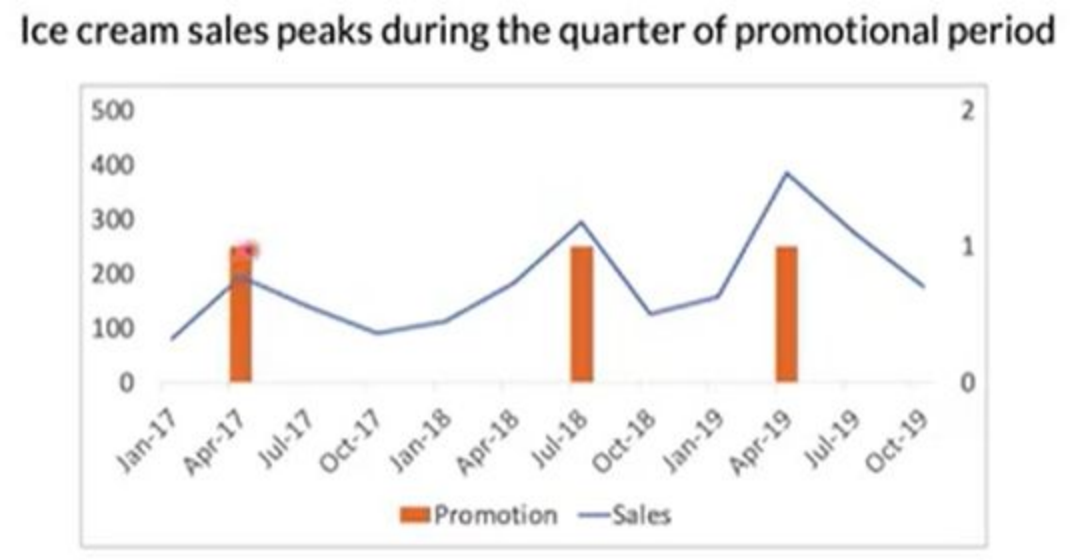
SARIMAX has three components:

- Non-seasonal elements
  - Models future observation as a linear regression of past observations and past forecast errors.
  - Performs differencing to make time-series stationary.
- Seasonal elements
  - Models seasonality as the linear regression of past observations and past forecast errors from previous seasons.
  - Perform seasonal differencing to make time-series stationary over seasons.
- Exogenous variable
  - Models future observations as linear regression of an external variable.

Example: Forecast quarterly ice cream sales for 2020 when you have the data of quarterly sales of the ice cream for the last 4 years.

In the above time plot, sales data displays trend and seasonality but peaks are not periodic. The peak in 2017 has appeared in the first quarter, while in 2018 it has appeared in the 2nd quarter and again it has appeared in the first quarter of 2018. Thus, it is evident that peaks, in spite of being close to summer, are not periodic.

This irregularity in the peak is due to an extra factor, maybe the promotions. The respective months when a promotion is done, there is an extra sale leading to a peak in the quarter. You can visually observe this here:



If we know this promotion period for the quarter, it will help us to forecast the sales more accurately.

Equations:

SARIMAX(1,0,0)(0,1,1)$_4$

$$\hat{z}_t = \beta z_{t-1} + \phi \varepsilon_{t-4} + \alpha x_t$$

where

$$z_t = y_t - y_{t-4}$$

The parameters **'p', 'd', 'q'** and **'P', 'D', 'Q'** will be the same as SARIMA(p,d,q)(P,D,Q)$_m$.

- Determining parameter values
  - PACF plots to determine non-seasonal 'p' value
  - ACF plots to identify non-seasonal 'q' value
- Use stationarity tests to determine the value 'd'
- Use grid search to choose optimal seasonal P, D and Q parameter values

The following steps are performed on a promotion dataset that has an exogenous variable to build the SARIMAX model. The forecast plot is then calculated along with the MAPE value and compared with the previously built models.

Seasonal auto regressive integrate moving average with exogenous variable (SARIMAX)

Import promotion data

In [66]:
```
promo = pd.read_csv('promotion.csv', header = None)
promo.columns = ['Month','Event']
promo['Month'] = pd.to_datetime(promo['Month'], format='%Y-%m')
promo = promo.set_index('Month')
promo
```

Split promotion data into train and test data sets

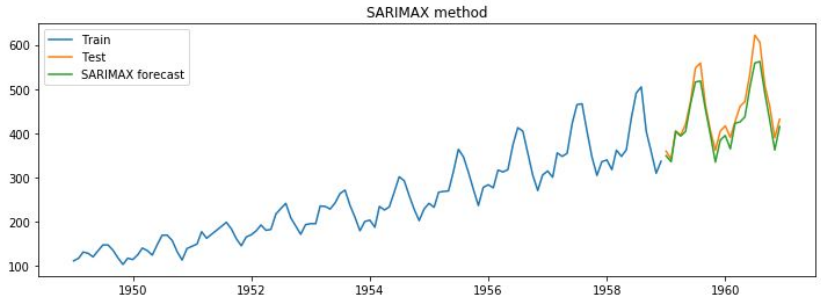
In [67]:
```
promo_train = promo[:train_len]
promo_test = promo[train_len:]
```

Recover original time series forecast

In [69]:
```
y_hat_sarimax = data_boxcox_diff.copy()
y_hat_sarimax['sarimax_forecast_boxcox'] = model_fit.predict(data_boxcox_diff.index.min(), data_boxcox_diff.index.max(), exog=pro
y_hat_sarimax['sarimax_forecast'] = np.exp(y_hat_sarimax['sarimax_forecast_boxcox'])
```

Plot train, test and forecast

In [70]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_sarimax['sarimax_forecast'][test.index.min():], label='SARIMAX forecast')
plt.legend(loc='best')
plt.title('SARIMAX method')
plt.show()
```



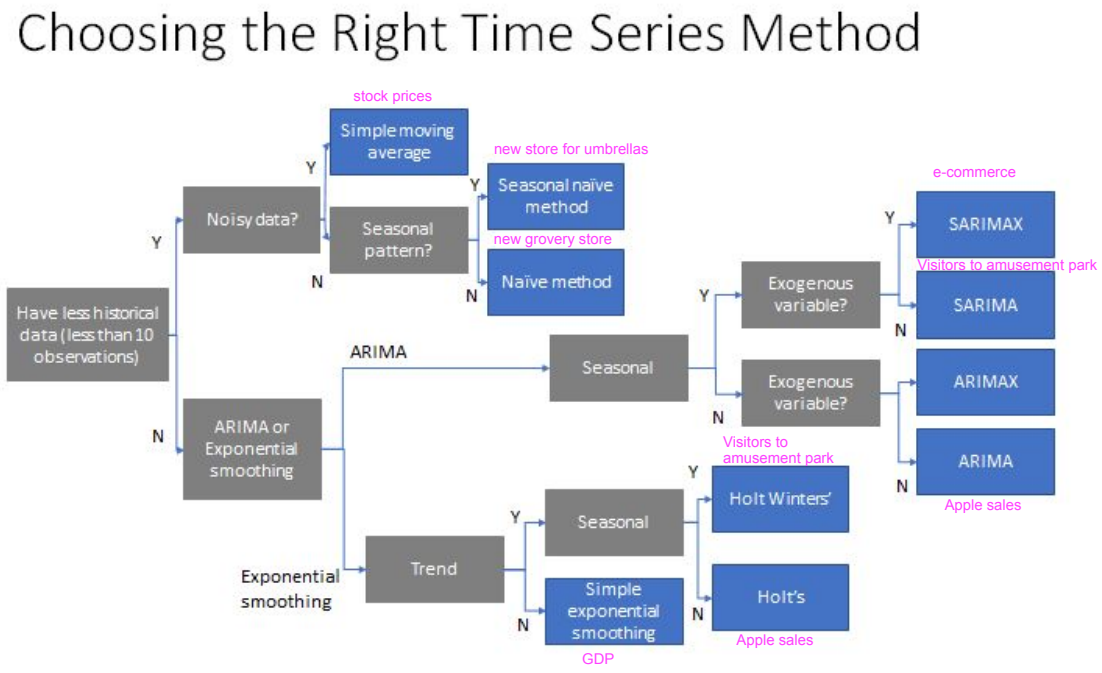
Calculate RMSE and MAPE

In [71]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_sarimax['sarimax_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_sarimax['sarimax_forecast'][test.index.min():])/test['Passengers'])*100,

tempResults = pd.DataFrame({'Method':['SARIMAX method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[71]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91
0	Autoregressive (AR) method	93.39	13.77
0	Moving Average (MA) method	91.21	13.39
0	Autoregressive moving average (ARMA) method	88.88	12.89
0	Autoregressive integrated moving average (ARIM...	88.88	12.89
0	Seasonal autoregressive integrated moving aver...	37.54	7.37
0	SARIMAX method	26.64	4.66

Choosing the Right Time Series Method



When you have time-series data of fewer than 10 observations that are noisy, you should use a simple moving average method because it helps cancel out the noise to some extent. This method does not work well if the data has a seasonal component or more than 10 observations.

An example in which a simple moving average method works well is the daily forecasting of stock prices. This is because the number of observations is fewer and the data is noisy. Thus, the simple moving average method is able to predict the forecast better since it takes the variation of very few data points.

If the data points are fewer than 10 but the data is neither noisy nor has any seasonal pattern, then the Naive method works well because it will forecast the next values based on the previous values of the train data. In the case of a higher number of observations(generally more than 10), the forecast tends to underpredict or overpredict the values.

Thus, the naive method works better when there are fewer than 10 observations. An example of the usage of the naive method would be in forecasting the sales of a grocery store that opened

recently. In this case, the sales are not much dependent on any seasonal component, and thus, the naive method can be used here

Also, in the case of non-noisy data and seasonality with data points fewer than 10, the seasonal naive method works well. An example in which the seasonal naive method works well is forecasting the sales of a newly opened store that sells umbrellas on a monthly basis.

In the case of more than 10 data points, for using a technique among the smoothing/ARIMA techniques, you need to keep the following points in mind:

1. To capture the level in time series data, the simple exponential smoothing technique is used. An example in which the simple exponential smoothing technique is used is the forecasting of the annual GDP of a developed country. In this case, there may or may not be an obvious trend in the data and the seasonal component is absent. Thus, using a simple exponential smoothing technique makes sense here.
2. Holt's exponential smoothing technique / ARIMA method works best in capturing both the level and the trend in the time series data. An example in which Holt's exponential smoothing technique/ARIMA method can be used is when there is an obvious trend in the data but no specific seasonality exists. For example, the sales data for iPhones for the last 10–12 years show an increasing trend but no specific seasonality; thus, the Holt's method / ARIMA method can be used to forecast the time series data.
3. Now, to capture the level, the trend and seasonality, the Holt-Winters' exponential smoothing technique / SARIMA works best. However, this method should be used when the dataset has no exogenous variables. An example in which the techniques mentioned here can be used is in determining the number of monthly visitors to an amusement park. With increased marketing, the number of visitors to the amusement park keeps increasing and thus, the data shows a trend. However, these numbers show a seasonal pattern. For example, during the summer and winter vacation months, visitors tend to flock to the amusement park even more, which implies that the dataset has a seasonality component.
4. The ARIMAX/SARIMAX method works best for capturing the level, the trend and the seasonality in time series data when some exogenous variables are present. The ARIMAX method may not capture the seasonality but the SARIMAX method does. An example of using these methods is in determining the monthly sales of an e-commerce website. Here, the sales are affected by numerous external factors and thus, the ARIMAX/SARIMAX method will work well here.

Disclaimer: All content and material on the UpGrad website is copyrighted material, either belonging to UpGrad or its bonafide contributors and is purely for the dissemination of education. You are permitted to access print and download extracts from this site purely for your own education only and on the following basis:

● You can download this document from the website for self-use only.
● Any copies of this document, in part or full, saved to disc or to any other storage medium may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
● Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of content for any other commercial/unauthorized purposes in any way which could infringe the intellectual property rights of UpGrad or its contributors, is strictly prohibited.
● No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
● No material in this document will be modified, adapted or altered in any way.
● No part of this document or UpGrad content may be reproduced or stored in any other web site or included in any public or private electronic retrieval system or service without UpGrad's prior written permission.
● Any rights not expressly granted in these terms are reserved.