QBUS6840: Tutorial 9 – MA models, ARMA models and fitting

Objectives

- Analyze MA and ARMA models
- Interpret ACF/PACF plots
- Fit ARMA models to observed data
- Develop Python skills

In this week, we will continue the topic of ARIMA model.

MA(q) process:

Rather than using past values of the forecast variable in a regression, a moving average process uses past model errors in a regression-like way. An MA(q) could be viewed as:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

Notice that each value of y_t can be thought of as a weighted moving average of the past few errors. **However, moving average process should not be confused with the moving average smoothing we discussed in time series decomposition. A moving average process defines how the time series process evolves and can be used for forecasting future values, while moving average smoothing is mainly used for revealing the trend-cycle of past values in our context.**

Invertibility:

It is possible to write any stationary AR(p) model as an MA(∞) model. For example, using repeated substitution, we can demonstrate this for an AR(1) model:

$$\begin{aligned}
y_t &= c + \phi_1 y_{t-1} + \varepsilon_t \\
&= c + \phi_1(\phi_1 y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\
&= c + {\phi_1}^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\
&\dots \\
&= c + {\phi_1}^k y_{t-k} + \cdots + {\phi_1}^2 \varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t
\end{aligned}$$

Provided $-1 < \phi_1 < 1$, the value of ${\phi_1}^k$ will get smaller as k gets larger. Therefore, eventually we have:

$$y_t = c + \varepsilon_t + \phi_1 \varepsilon_{t-1} + {\phi_1}^2 \varepsilon_{t-2} + {\phi_1}^3 \varepsilon_{t-3} + \cdots$$

which is an MA(∞) process.

Alternatively, we can write any invertible MA(q) process as an AR(∞) process.

For example, consider the MA(1) model:

$$\begin{aligned} y_t &= c + \theta_1 \varepsilon_{t-1} + \varepsilon_t \\ &= c + \theta_1 (y_{t-1} - c + \theta_1 \varepsilon_{t-2}) + \varepsilon_t \\ &= c(1 - \theta_1 + {\theta_1}^2) - [(-1)^1 \theta_1 y_{t-1} + (-1)^2 {\theta_1}^2 y_{t-2}] + \varepsilon_t \\ &\dots \\ &= c(1 - \theta_1 + {\theta_1}^2 - {\theta_1}^3 + \cdots) - \sum_{i=1}^{\infty} (-1)^i {\theta_1}^i y_{t-i} + \varepsilon_t \end{aligned}$$

Mathematically, only when $|\theta_1| < 1$, the above expansion is meaningful. In fact, $(1 - \theta_1 + {\theta_1}^2 - {\theta_1}^3 + \cdots)$ is 发散 divergent when $|\theta_1| > 1$. For example, when $\theta_1 = 1$, do you think the sum of $1 - \theta_1 + {\theta_1}^2 - {\theta_1}^3 + \cdots$ is 1 or 0? Hence we require $|\theta_1| < 1$, so the most recent observations have higher weight than observations from the more distant past. **Thus, the process is invertible when $|\theta_1| < 1$,** which is similar to the stationarity constraints. When $|\theta_1| > 1$, we have shown that y_t can only be expanded as the weighted sum of future observations in Lecture 8.

What are the invertibility constraints for MA(2) model?
For MA(2) model, we should have the following constraints: $|\theta_2| < 1$, $\theta_1 + \theta_2 < -1$ and $\theta_1 - \theta_2 < 1$

ARIMA(p,d,q) processes:

For a ARMA(p,q) process:

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

When p=q=0, the model is called the white noise process.
When q = 0, the model is called AR(p) process.
When p = 0, the model is called MA(q) process.

We are only interested in stationary ARMA processes. Obviously many observed time series do not look like a time series generated from stationary process, thus we will have difficulties to model it with ARMA processes. Therefore, we take d-th order difference transform to obtain a stationary input, and then model the resulting new time series with an ARMA process. In this case, the model becomes an ARIMA(p,d,q) process for the given observed time series (non-differencing) .

1. Define MA(q) or ARMA(p,q) in `statsmodels`

Step 1. Set parameters for MA or ARMA processes

Import required libraries

```
import matplotlib.pyplot as plt
import statsmodels as sm
import statsmodels.api as smt
import numpy as np
```

Set up the parameters

```
np.random.seed(1)

arparams = np.array([0.9])
zero_lag = np.array([1])
maparams = np.array([0.6, -0.5])

ar = np.r_[1, -arparams]  # add zero-lag (coefficient 1) 
and negate
ma = np.r_[1, maparams]   # add zero-lag

c = 0

sigma2 = 1
```

Question: Why do we use `ar = np.r_[1, -arparams]` to define AR coefficient?

Note that we add a negative sign in front of `arparams`. The reason for this is, we write a general ARMA process in the following way:

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

But **`statsmodels`** defines an ARMA process as

$$y_t = c - \phi_1 y_{t-1} - \cdots - \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

Given the definition in the python code, we have $\phi_1 = 0.9$ with p = 1, $\theta_1 = 0.6$, $\theta_2 = -0.5$ with q = 2.

It is **`statsmodels`**‘ requirement that we must add 1 for both ar and ma coefficients.

Step 2. Define an MA process with the given coefficients

With the `statsmodels` libraries, an ARMA model is given by an ARMA model object (Object is python term). The following code shows you how:

```
ma_model = sm.tsa.arima_process.ArmaProcess(ar = zero_lag, 
ma = ma)

# Check if it is stationary
print("MA Model is{0}stationary".format(" " if 
ma_model.isstationary else " not " ))

# Check if it is invertible
print("MA Model is{0}invertible".format(" " if
ma_model.isinvertible else " not " ))

# Plot ACF
plt.figure()
plt.stem(ma_model.acf())

# Plot PACF
plt.figure()
plt.stem(ma_model.pacf())

# Generate samples
n_samples = 250

#Yt = c + e_t + 0.6e_{t-1} - 0.5 e_{t-2}
y2 = c + ma_model.generate_sample(n_samples)

plt.figure()
plt.plot(y2)
plt.title("Moving Average (MA) Model")
```

Note 1: The first statement creates an ARMA model. As we set `ar = zero_lag` which is 1, so we end up with an MA model. Actually `sm.tsa.arima_process.ArmaProcess` produces for theoretical analysis. It is unfortunately it only produces a process without the intercept c (i.e., c = 0).

Note 2: We can produce the **theoretical** ACF and PACF in statements 5 and 6 **BEFORE** we generate any samples!

Note 3: After we have created an ARMA model object (in this example, an MA(2) model object), we can produce time series sample, as shown in the statement for y2.

Step 3. Test the Sample ACF and PACF

Now we show the sample ACF and PACF from the time series y2.

```
smt.graphics.tsa.plot_acf(y2, lags=30, alpha = 0.05)
smt.graphics.tsa.plot_pacf(y2, lags=30, alpha = 0.05)
```

You may compare these sample ACF and PACF with the theoretical ACF and PACF.

Calculate the unconditional mean and variance.

```
# Mean
# Since epsilon_t ~ N(0, sigma^2)
# then E(epsilon_t) = 0

y2_mean = c
```

```
# Variance
# c is constant so var(c) = 0
# var(epsilon_t) = sigma^2
# then var(y2) = 0 + sigma^2 + theta_1 * sigma^2 +
theta_2 * sigma^2

y2_variance = sigma2 * (1 + maparams[0]**2 +
maparams[1]**2)
```

Step 4. Define an ARMA process

Now combine both AR and MA parameters to produce an ARMA process

```
arma_model = sm.tsa.arima_process.ArmaProcess(ar = ar, ma
= ma)

n_samples = 1000

#Yt = 0.9Y_{t-1} + 0.6e_{t-1} - 0.5 e_{t-2} + e_t
y3 = arma_model.generate_sample(n_samples)

plt.figure()
plt.plot(y3)
plt.title("ARMA Model")

smt.graphics.tsa.plot_acf(y3, lags=30, alpha = 0.05)
smt.graphics.tsa.plot_pacf(y3, lags=30, alpha = 0.05)
```

Question: How to produce theoretical ACF and PACF for this process?

Step 5. Model Fitting

Now let’s try and fit a model to these samples and compare the parameters with those that we actually specified.

```
# Lets try and fit the model back
# (0, 2) refers to the number of parameters in the (AR,
MA) parts respectively

model_y2 = sm.tsa.arima_model.ARMA(y2, (0,
2)).fit(trend='nc')
print("Estimated Model Parameters: " +
str(model_y2.params))

model_y3 = sm.tsa.arima_model.ARMA(y3, (1,
2)).fit(trend='nc')
print("Estimated Model Parameters: " +
str(model_y3.params))
```

Note: Check how we produce an ARMA from a sample time series by comparing the first statement here with the first statement in Step 2? What is the difference?

2. Forecasting with ARIMA

In the previous task, we have learnt how to define a theoretical ARMA process for given model parameters and how to train an ARMA model on a given time series data by using `statsmodel` library. In this task, we will use ARIMA model to forecast the AirPassengers dataset.

Step 1. Load the dataset

Download the dataset from the Canvas website, and then copy the following code to your python editor.

```
#%% load the dataset
dateparse = lambda dates: pd.datetime.strptime(dates,
'%Y-%m')
airdata = pd.read_csv('AirPassengers.csv',
parse_dates=['Month'],
index_col='Month',date_parser=dateparse)
ts = airdata['Passengers']
plt.figure()
plt.plot(ts)
plt.title("Air passenger data")
```

Step 2. Log transform

Check the figures. We notice the peak within each period is increasing significantly. Recall in week04 tutorial, when we do the additive decomposition, we applied the log transform so that we could penalize the high peak and increase the low peak. Here, we do the same transform to stabilize the magnitude.

Run the following code and compare the log plot with the original plot.

```
#%% log the data
ts_log = np.log(ts)
plt.figure()
plt.plot(ts_log)
plt.title("Air passenger data (log)")
```

Step 3. 1st differencing

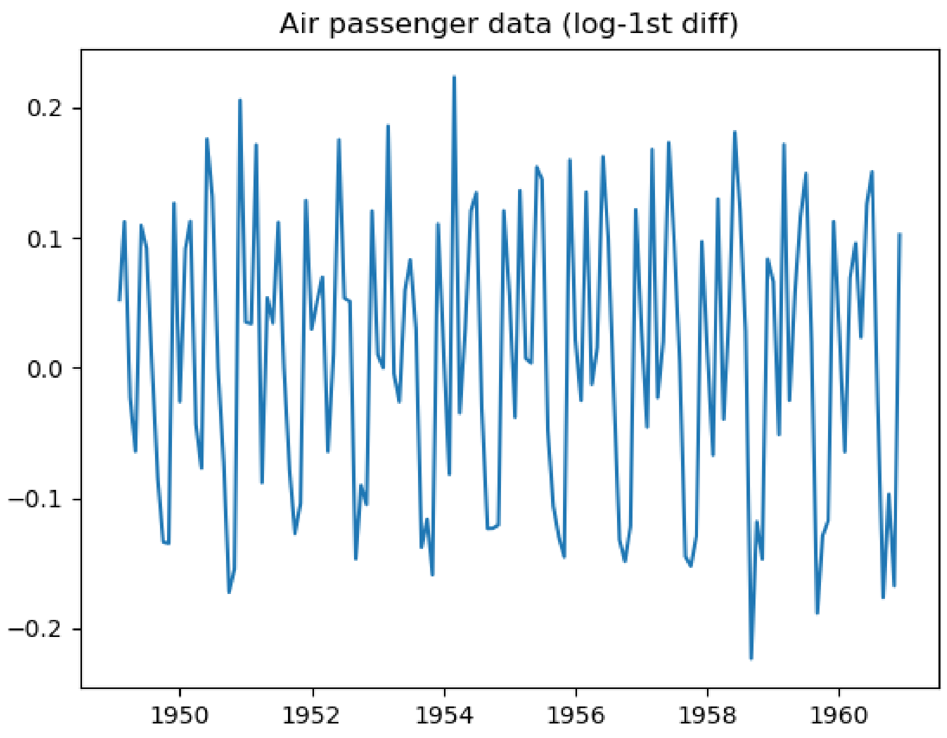
Clearly the transformed sequence is not stationary. Though `statmodel` library's ARIMA(p,d,q) does not require a stationary data as input, we still need to know the best fitting integrated value d. Therefore, we need to do the 1st order difference. If the 1st order difference is still not stationary, we then do 2nd diff, etc … until we decide an appropriate order. Here, we firstly start with trying the 1st order difference.

```
#%% take the 1st order diff
ts_log_diff = ts_log - ts_log.shift()
ts_log_diff.dropna(inplace=True)
plt.figure()
plt.plot(ts_log_diff)
plt.title("Air passenger data (log-1st diff)")
```

Question: Why we need to call `dropna()` function in here?

Check the size of the `ts_log_diff` variable.

Check your 1st order difference figure and compare with this:



We can see the 1st difference plot remains the same mean value and the variance does not change significantly. In this case, we can call this data as stationary data. Then we could set the d=1 in ARIMA.

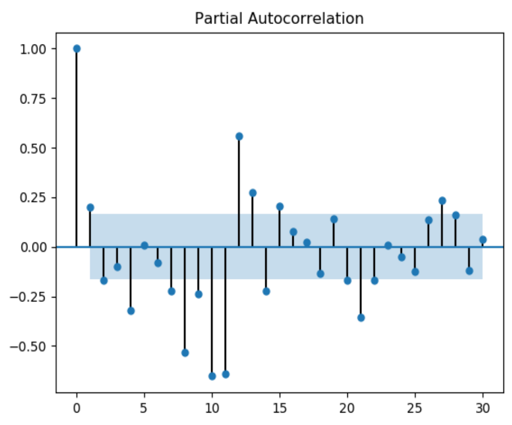
To ensure that the transformed series is stationary, we may carefully check its sample ACF. We do it in the following step.

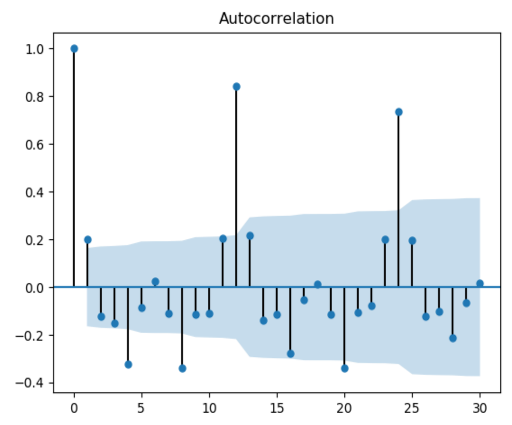
Step 4. ACF and PACF of the 1st differencing data

Then we examine the ACF and PACF plot to obtain a suitable order for AR(p) and MA(q) process.

Run the following code and answer the following question:

```
smt.graphics.tsa.plot_acf(ts_log_diff, lags=30, alpha =
0.05)
smt.graphics.tsa.plot_pacf(ts_log_diff, lags=30, alpha =
0.05)
```





Examine the PACF plot, what is the suitable order p for AR process?
Examine the ACF plot, what is the suitable order q for MA process?

As this time series contains seasonal component, we can see the big spikes at lag 12, 24 etc. We will handle this in the next tutorial. For our purpose of showing how to do ARIAM modeling, we decide to select p = 2 and q = 0. That is we are looking at ARIMA(2,1,0).

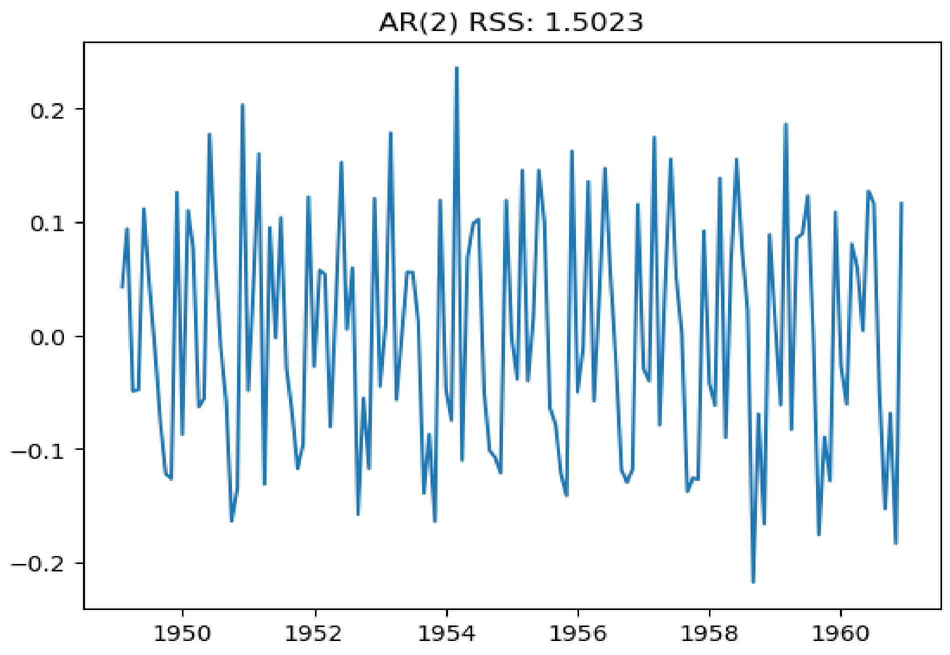
Step 5. Plot the AR(2) of the differenced data = ARIMA(2,1,0)

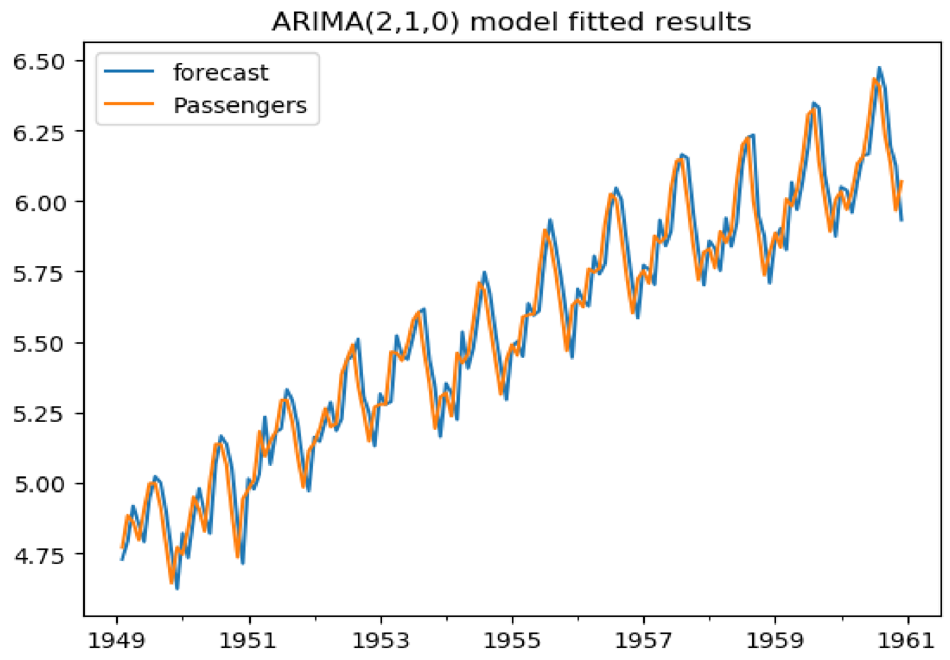
Run the following code and check the fitted results.

```
#%% AR(2) model
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(ts_log, order=(2, 1, 0))
results_AR = model.fit(disp=-1)
residuals = pd.DataFrame(results_AR.resid)
plt.figure()
plt.plot(residuals)
plt.title('AR(2) RSS: %.4f'%
sum((results_AR.resid.values)**2))

# Get Fitted Series
fitted = results_AR.predict(typ = 'levels', dynamic =
False)

# Actual vs Fitted
results_AR.plot_predict(dynamic=False)
```

```
plt.show()
```

Note: Here `fitted` means fitting the `ts_log`. `dynamic = False` means that the in-sample lagged values are used for prediction. `typ = 'levels'` means predicting the levels of the original time series; while `typ = 'linear'` means linear prediction in terms of the differenced time series.





Why we need to input ts_log for ARIMA rather than ts_log_diff?

Step 6. Using AIC to select best fitting AR and MA order

AIC function is useful in selecting predictors for regression. It is also useful for determining the order of an ARIMA model.

From `statsmodel` library, you can call `arma_order_select_ic()` function to select the best fitting order of an ARIMA model:

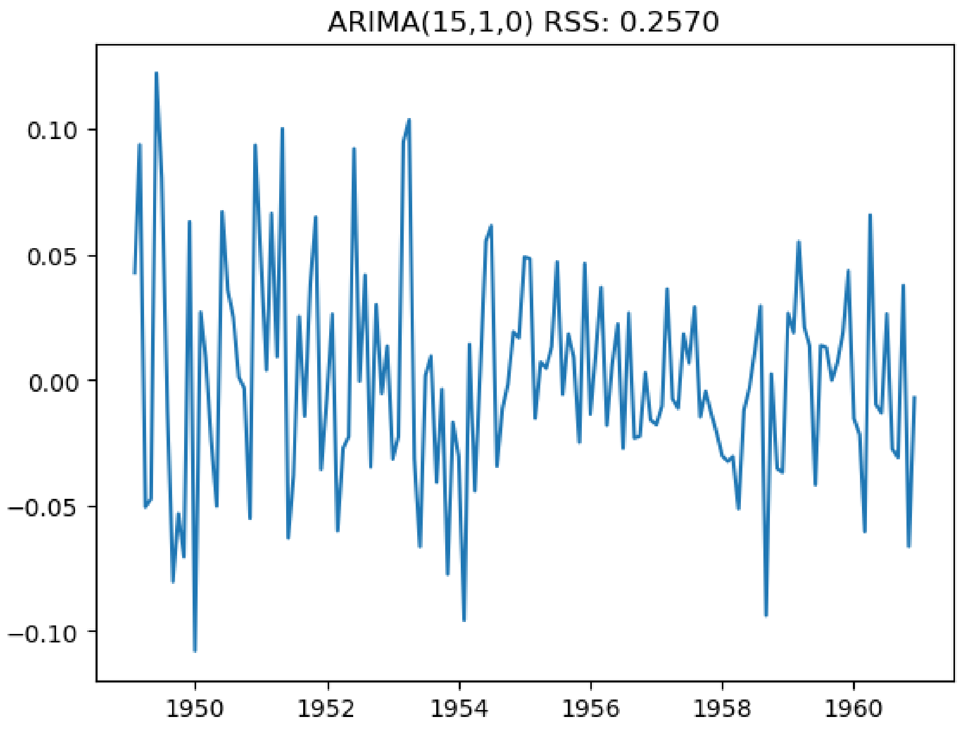
```
import statsmodels.tsa.stattools as st
order =
st.arma_order_select_ic(ts_log_diff,max_ar=15,max_ma=15,i
c=['aic'])
print(order.aic_min_order)
```

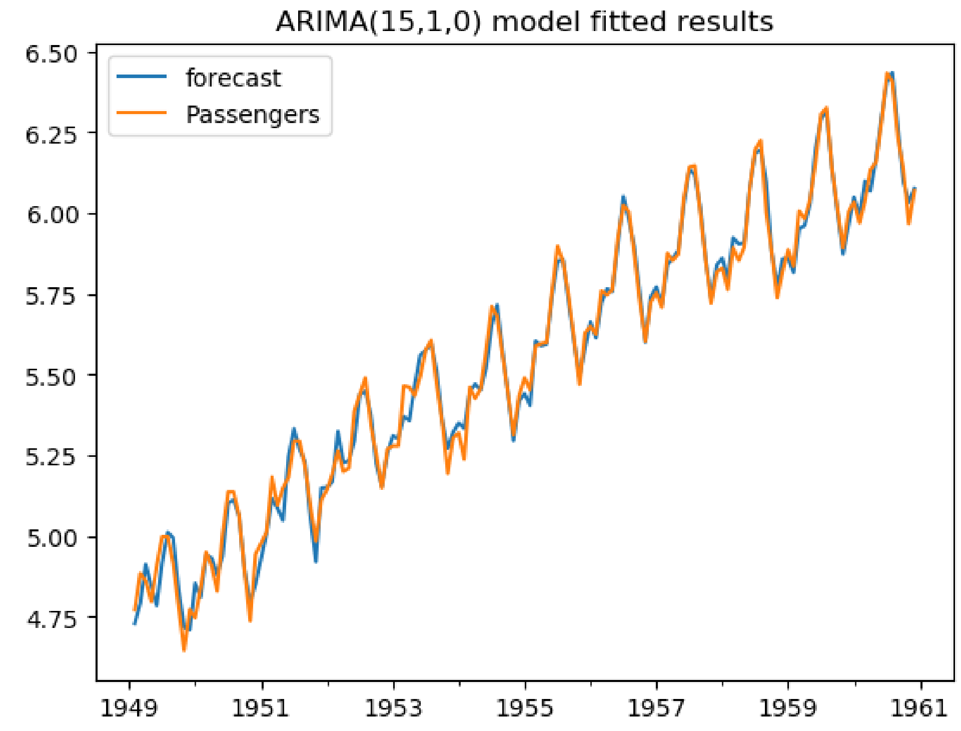
Alternatively, you can call BIC criteria by setting the `ic = [‘bic’]`

Note: calling the above function is time consuming. You don’t need to run the above code, as it will take around 30 minutes to 1 hour to finish training. Therefore, avoid setting a large value of `max_ar` or `max_ma` if you want to use AIC or BIC to select the order number.

According to the result, the best fitting order is p = 15, q = 0. We take this order in the ARIMA process and compare the fitted series and the Actual curve.

```
#%%
#p,q = order.aic_min_order[0], order.aic_min_order[1]
# or you can directly assign p,q with 15,0 without
training
p,q = 15,0

model_AIC = ARIMA(ts_log, order=(p, 1, q))
results_AIC_ARIMA = model_AIC.fit(disp=-1)

residuals_AIC = pd.DataFrame(results_AIC_ARIMA.resid)
plt.figure()
plt.plot(residuals_AIC)
plt.title('ARIMA(15,1,0) RSS: %.4f'%
sum((results_AIC_ARIMA.resid.values)**2))

# Get Fitted Series
fitted_AIC = results_AIC_ARIMA.predict(typ = 'levels',
dynamic = False)

# Actual vs Fitted
results_AIC_ARIMA.plot_predict(dynamic=False)
plt.show()
plt.title("ARIMA(15,1,0) model fitted results")
```


ARIMA(15,1,0) RSS: 0.2570
0.10
0.05
0.00
−0.05
−0.10
1950
1952
1954
1956
1958
1960


ARIMA(15,1,0) model fitted results
forecast
Passengers
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
1949
1951
1953
1955
1957
1959
1961