**1. Intro to ACF and PACF**

00:00 - 00:07

Now you know how to fit ARIMA models and make forecasts, but how do we choose which ARIMA model to fit?

**2. Motivation**

00:07 - 00:25

The model order is very important to the quality of forecasts. Here we have fit different models to the same dataset and then made forecasts using each. The mean predictions of the forecasts are shown as orange lines and you can see that they are very different.

**3. ACF and PACF**

00:25 - 00:37

One of the main ways to identify the correct model order is by using the autocorrelation function, the ACF, and the partial autocorrelation function the PACF.

**4. What is the ACF**

00:37 - 01:03

The autocorrelation function at lag-1 is the correlation between a time series and the same time series offset by one step. The autocorrelation at lag-2 is the correlation between a time series and itself offset by two steps. And so on. When we talk about the autocorrelation function we mean the set of correlation values for different lags

**5. What is the ACF**

01:03 - 01:19

We can plot the autocorrelation function to get an overview of the data. The bars show and ACF values at increasing lags. If these values are small and lie inside the blue shaded region, then they are not statistically significant.

**6. What is the PACF**

01:19 - 01:44

The partial autocorrelation is the correlation between a time series and the lagged version of itself after we subtract the effect of correlation at smaller lags. So it is the correlation associated with just that particular lag. The partial autocorrelation function is this series of values and we can plot it to get another view of the data.

**7. Using ACF and PACF to choose model order**

01:44 - 02:05

By comparing the ACF and PACF for a time series we can deduce the model order. If the amplitude of the ACF tails off with increasing lag and the PACF cuts off after some lag p, then we have a AR(p) model. This plot is an AR(2) model

**8. Using ACF and PACF to choose model order**

02:05 - 02:18

If the amplitude of the ACF cuts off after some lag q and the amplitude of the PACF tails off then we have a MA(q) model. This is an MA(2) model

**9. Using ACF and PACF to choose model order**

02:18 - 02:29

If both the ACF and PACF tail off then we have an ARMA model. In this case we can't deduce the model orders of p and q from the plot.

**10. Using ACF and PACF to choose model order**

02:29 - 02:35

You can refer to the following table when analyzing the ACF and PACF

**11. Implementation in Python**

02:35 - 03:21

In the statsmodel package there are two functions to make plots of the ACF and the PACF. These are plot-underscore-acf and plot-underscore-pacf functions. We import them like this. To use them, we start by creating a figure with two subplots. Into each function we pass the time series DataFrame and the maximum number of lags we would like to see. We also tell it whether to show the autocorrelation at lag-0. The ACF and PACF at lag-0 will always have a value of one so we'll set this argument to false to simplify the plot. Finally we pass it the axis to plot on. The plot-pacf function works in the same way.

**12. Implementation in Python**

03:21 - 03:25

Here are the plots we generated.

**13. Over/under differencing and ACF and PACF**

03:25 - 03:39

The time series must be made stationary before making these plots. If the ACF values are high and tail off very very slowly this is a sign that the data is non-stationarity, so it needs to be differenced.

**14. Over/under differencing and ACF and PACF**

03:39 - 03:47

If the autocorrelation at lag-1 is very negative this is a sign that we have taken the difference too many times.

**15. Let's practice!**

03:47 - 03:52

Now it's time to get down to some data. Let's practice!

## Exercise

# AR or MA

In this exercise you will use the ACF and PACF to decide whether some data is best suited to an MA model or an AR model. Remember that selecting the right model order is of great importance to our predictions.

Remember that for different types of models we expect the following behavior in the ACF and PACF:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **AR(p)** | **MA(q)** | **ARMA(p,q)** |
| ACF | Tails off | Cuts off after lag q | Tails off |
| PACF | Cuts off after lag p | Tails off | Tails off |

A time series with unknown properties, df is available for you in your environment.

## Instructions 1/2

* Import the plot\_acf and plot\_pacf functions from statsmodels.
* Plot the ACF and the PACF for the series df for the first 10 lags but not the zeroth lag.

**Questions:**

# Import

from statsmodels.graphics.tsaplots import \_\_\_\_, \_\_\_\_

# Create figure

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))

# Plot the ACF of df

\_\_\_\_(\_\_\_\_, lags=\_\_\_\_, zero=False, ax=ax1)

# Plot the PACF of df

\_\_\_\_(\_\_\_\_, lags=\_\_\_\_, zero=\_\_\_\_, ax=ax2)

plt.show()

# Import from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf # Create figure fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8)) # Plot the ACF of df plot\_acf(df, lags=10, zero=False, ax=ax1) # Plot the PACF of df plot\_pacf(df, lags=10, zero=False, ax=ax2) plt.show()

# Import

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# Create figure

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))

# Plot the ACF of df

plot\_acf(df, lags=10, zero=False, ax=ax1)

# Plot the PACF of df

plot\_pacf(df, lags=10, zero=False, ax=ax2)

plt.show()

## Question

Based on the ACF and PACF plots, what kind of model is this?

### Possible answers

AR(3)

**MA(3)**

ARMA(3)

Perfect! The ACF cuts off after 3 lags and the PACF tails off.

## Exercise

# Order of earthquakes

In this exercise you will use the ACF and PACF plots to decide on the most appropriate order to forecast the earthquakes time series.

|  | **AR(p)** | **MA(q)** | **ARMA(p,q)** |
| --- | --- | --- | --- |
| ACF | Tails off | Cuts off after lag q | Tails off |
| PACF | Cuts off after lag p | Tails off | Tails off |

The earthquakes time series earthquake, the plot\_acf(), and plot\_pacf() functions, and the ARIMA model class are available in your environment.

## Instructions 1/3

Plot the ACF and the PACF of the earthquakes time series earthquake up to a lag of 15 steps and don't plot the zeroth lag

# Create figure

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))

# Plot ACF and PACF

\_\_\_\_(\_\_\_\_, \_\_\_\_=\_\_\_\_, \_\_\_\_=\_\_\_\_, ax=ax1)

\_\_\_\_(\_\_\_\_, \_\_\_\_=\_\_\_\_, \_\_\_\_=\_\_\_\_, ax=ax2)

# Show plot

plt.show()

# Create figure fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8)) # Plot ACF and PACF plot\_acf(earthquake, lags=15, zero=False, ax=ax1) plot\_pacf(earthquake, lags=15, zero=False, ax=ax2) # Show plot plt.show()

**# Create figure**

**fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))**

**# Plot ACF and PACF**

**plot\_acf(earthquake, lags=15, zero=False, ax=ax1)**

**plot\_pacf(earthquake, lags=15, zero=False, ax=ax2)**

**# Show plot**

**plt.show()**

## Question

Look at the ACF/PACF plots and the table above.

What is the most appropriate model for the earthquake data?

### Possible answers

ARMA(1,4)

AR(4)

MA(1)

ARMA(1,1)

**AR(1) The PACF cuts off after lag-1 and the ACF tails off.**

**Create an appropriate model and train it on the earthquakes time series.**

# Show plot

plt.show()

# Instantiate model

model = \_\_\_\_

# Train model

results = \_\_\_\_

**# Create figure**

**fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))**

**# Plot ACF and PACF**

**plot\_acf(earthquake, lags=10, zero=False, ax=ax1)**

**plot\_pacf(earthquake, lags=10, zero=False, ax=ax2)**

**# Show plot**

**plt.show()**

**# Instantiate model**

**model = ARIMA(earthquake, order=(1,0,0))**

**# Train model**

**results = model.fit()**

**In this exercise you just went the whole way from raw data to the optimal trained model! Amazing!**

**1. AIC and BIC**

00:00 - 00:17

In the last lesson, we mentioned how ACF and PACF can't be used to choose the order of a model, when both of the orders p and q are non-zero. However there are more tools we can use, the AIC and the BIC.

**2. AIC - Akaike information criterion**

00:17 - 00:44

The Akaike information criterion, or AIC, is a metric which tells us how good a model is. A model which makes better predictions is given a lower AIC score. The AIC also penalizes models which have lots of parameters. This means if we set the order too high compared to the data, we will get a high AIC value. This stops us overfitting to the training data.

**3. BIC - Bayesian information criterion**

00:44 - 00:58

The Bayesian information criterion, or BIC, is very similar to the AIC. Models which fit the data better have lower BICs and the BIC penalizes overly complex models.

**4. AIC vs BIC**

00:58 - 01:36

For both of these metrics a lower value suggests a better model. The difference between these two metrics is how much they penalize model complexity. The BIC penalizes additional model orders more than AIC and so the BIC will sometimes suggest a simpler model. The AIC and BIC will often choose the same model, but when they don't we will have to make a choice. If our goal is to identify good predictive models, we should use AIC. However if our goal is to identify a good explanatory model, we should use BIC.

**5. AIC and BIC in statsmodels**

01:36 - 01:49

After fitting a model in Python, we can find th AIC and BIC by using the summary of the fitted-models-results object. These are on the right of the table.

**6. AIC and BIC in statsmodels**

01:49 - 02:00

You can also access the AIC and BIC directly by using the dot-aic attribute and the dot-bic attribute of the fitted-model-results object.

**7. Searching over AIC and BIC**

02:00 - 02:26

Being able to access the AIC and BIC directly means we can write loops to fit multiple ARIMA models to a dataset, to find the best model order. Here we loop over AR and MA orders between zero and two, and fit each model. Then we print the model order along with the AIC and BIC scores

**8. Searching over AIC and BIC**

02:26 - 02:37

If we want to test a large number of model orders, we can append the model order and the AIC and BIC to a list, and later covert it to a DataFrame.

**9. Searching over AIC and BIC**

02:37 - 03:01

This means we can sort by the AIC score and not have to search through the orders by eye. We can do the same with the BIC score. In this case the AIC and BIC favor different models, but we want a good predictive model so we will choose the model with the lowest AIC. This is an ARMA(2,1) model.

**10. Non-stationary model orders**

03:01 - 03:24

Sometimes when searching over model orders you will attempt to fit an order that leads to an error. This ValueError tells us that we have tried to fit a model which would result in a non-stationary set of AR coefficients. This is just a bad model for this data, and when we loop over p and q we would like to skip this one.

**11. When certain orders don't work**

03:24 - 03:33

We can skip these orders in our loop by using a try and except block in python. Here's the code we had before.

**12. When certain orders don't work**

03:33 - 03:47

First we try to run the code in the try statement. We try to fit the model and print the scores. If this fails then the code in the except statement is run, where we print None for the scores

**13. Let's practice!**

03:47 - 03:54

You've been learning a lot about the AIC and BIC. Time for you to put it to practice!

## Exercise

# Searching over model order

In this exercise you are faced with a dataset which appears to be an ARMA model. You can see the ACF and PACF in the plot below. In order to choose the best order for this model you are going to have to do a search over lots of potential model orders to find the best set.

A picture containing text, screenshot, line, diagram

Description automatically generated

The ARIMA model class and the time series DataFrame df are available in your environment.

## Instructions

100 XP

* Loop over values of p from 0-2.
* Loop over values of q from 0-2.
* Train and fit an ARMA(p,q) model.
* Append a tuple of (p,q, AIC value, BIC value) to order\_aic\_bic.

# Create empty list to store search results

order\_aic\_bic=[]

# Loop over p values from 0-2

for p in range(\_\_\_\_):

  # Loop over q values from 0-2

    for q in range(\_\_\_\_):

        # create and fit ARMA(p,q) model

        model = ARIMA(df, order=\_\_\_\_)

        results = model.fit()

        # Append order and results tuple

        order\_aic\_bic.append((\_\_\_\_))

# Create empty list to store search results order\_aic\_bic=[] # Loop over p values from 0-2 for p in range(3): # Loop over q values from 0-2 for q in range(3): # create and fit ARMA(p,q) model model = ARIMA(df, order=(p,0,q)) results = model.fit() # Append order and results tuple order\_aic\_bic.append((p, q, results.aic, results.bic))

**# Create empty list to store search results**

**order\_aic\_bic=[]**

**# Loop over p values from 0-2**

**for p in range(3):**

**# Loop over q values from 0-2**

**for q in range(3):**

**# create and fit ARMA(p,q) model**

**model = ARIMA(df, order=(p,0,q))**

**results = model.fit()**

**# Append order and results tuple**

**order\_aic\_bic.append((p, q, results.aic, results.bic))**

**Fantastic! You built 9 models in just a few seconds! In the next exercise you will evaluate the results to choose the best model.**

# Choosing order with AIC and BIC

Now that you have performed a search over many model orders, you will evaluate your results to find the best model order.

The list of tuples of (p,q, AIC value, BIC value) that you created in the last exercise, order\_aic\_bic, is available in your environment. pandas has also been imported as pd.

## Instructions 1/2

* Create a DataFrame to hold the order search information in the order\_aic\_bic list. Give it the column names ['p', 'q', 'AIC', 'BIC'].
* Print the DataFrame in order of increasing AIC and then BIC.

# Construct DataFrame from order\_aic\_bic

order\_df = pd.DataFrame(\_\_\_\_,

                        columns=[\_\_\_\_])

# Print order\_df in order of increasing AIC

print(\_\_\_\_)

# Print order\_df in order of increasing BIC

print(\_\_\_\_)

**# Construct DataFrame from order\_aic\_bic**

**order\_df = pd.DataFrame(order\_aic\_bic,**

**columns=['p', 'q', 'aic', 'bic'])**

**# Print order\_df in order of increasing AIC**

**print(order\_df.sort\_values('aic', ascending= True))**

**# Print order\_df in order of increasing BIC**

**print(order\_df.sort\_values('bic', ascending=True))**

**# Construct DataFrame from order\_aic\_bic**

**order\_df = pd.DataFrame(order\_aic\_bic,**

**columns=['p', 'q', 'AIC', 'BIC'])**

**# Print order\_df in order of increasing AIC**

**print(order\_df.sort\_values('AIC', ascending= True))**

**# Print order\_df in order of increasing BIC**

**print(order\_df.sort\_values('BIC', ascending=True))**

**p q AIC BIC**

**7 2 1 1414.249 1431.107**

**8 2 2 1416.085 1437.158**

**5 1 2 1417.030 1433.888**

**6 2 0 1419.109 1431.753**

**2 0 2 1425.057 1437.701**

**4 1 1 1428.052 1440.696**

**1 0 1 1429.989 1438.418**

**3 1 0 1497.308 1505.737**

**0 0 0 1615.494 1619.708**

**p q aic bic**

**7 2 1 1414.249 1431.107**

**6 2 0 1419.109 1431.753**

**5 1 2 1417.030 1433.888**

**8 2 2 1416.085 1437.158**

**2 0 2 1425.057 1437.701**

**1 0 1 1429.989 1438.418**

**4 1 1 1428.052 1440.696**

**3 1 0 1497.308 1505.737**

**0 0 0 1615.494 1619.708**

## Question

Which of the following models is the best fit?

### Possible answers

ARMA(0,0)

ARMA(2,0)

ARMA(0,2)

**ARMA(2,1) Perfect! This time AIC and BIC favored the same model, but this won't always be the case.**

ARMA(2,2)

## Exercise

# AIC and BIC vs ACF and PACF

In this exercise you will apply an AIC-BIC order search for the earthquakes time series. In the last lesson you decided that this dataset looked like an AR(1) process. You will do a grid search over parameters to see if you get the same results. The ACF and PACF plots for this dataset are shown below.

A picture containing text, screenshot, line, diagram

Description automatically generated

The ARIMA model class and the time series DataFrame earthquake are available in your environment.

## Instructions

100 XP

* Loop over orders of p and q between 0 and 2.
* Inside the loop try to fit an ARMA(p,q) to earthquake on each loop.
* Print p and q alongside AIC and BIC in each loop.
* If the model fitting procedure fails print p, q, None, None.
* **# Loop over p values from 0-2**
* **for p in range(3):**
* **# Loop over q values from 0-2**
* **for q in range(3):**
* **try:**
* **# create and fit ARMA(p,q) model**
* **model = ARIMA(earthquake, order=(p,0,q))**
* **results = model.fit()**
* **# Print order and results**
* **print(p, q, results.aic, results.bic)**
* **except:**
* **print(p, q, None, None)**
* **0 0 676.5443594984636 681.7345991987328**
* **0 1 654.8468586200252 662.6322181704289**
* **0 2 651.733227051897 662.1137064524353**
* **1 0 643.9676770992027 651.7530366496064**
* **1 1 640.4562835911981 650.8367629917365**
* **1 2 642.3808673565673 655.3564666072402**
* **2 0 642.7338539385339 653.1143333390722**
* **2 1 647.4406643641798 660.4162636148528**
* **2 2 642.4685896288877 658.0393087296952**

# Loop over p values from 0-2

for p in range(3):

    # Loop over q values from 0-2

    for q in range(3):

        try:

            # create and fit ARMA(p,q) model

            model = ARIMA(earthquake, order=(p,0,q))

            results = model.fit()

            # Print order and results

            print(p, q, results.aic, results.bic)

        except:

            print(p, q, None, None)

**Super! If you look at your printed results you will see that the AIC and BIC both actually favor an ARMA(1,1) model. This isn't what you predicted from the ACF and PACF but notice that the lag 2-3 PACF values are very close to significant, so the ACF/PACF are close to those of an ARMA(p,q) model.**

**1. Model diagnostics**

00:00 - 00:12

You've come a long way, but our work isn't finished once we have built the model. The next step is using common model diagnostics to confirm our model is behaving well.

**2. Introduction to model diagnostics**

00:12 - 00:22

After we have picked a final model or a final few models we should ask how good they are. This is a key part of the model building life cycle.

**3. Residuals**

00:22 - 00:35

To diagnose our model we focus on the residuals to the training data. The residuals are the difference between the our model's one-step-ahead predictions and the real values of the time series.

**4. Residuals**

00:35 - 00:48

In statsmodels the residuals over the training period can be accessed using the dot-resid attribute of the results object. These are stored as a pandas series.

**5. Mean absolute error**

00:48 - 01:06

We might like to know, on average, how large the residuals are and so how far our predictions are from the true values. To answer this we can calculate the mean absolute error of the residuals. We can do this in Python using the numpy-dot-abs and the numpy-dot-mean functions.

**6. Plot diagnostics**

01:06 - 01:30

For an ideal model the residuals should be uncorrelated white Gaussian noise centered on zero. The rest of our diagnostics will help us to see if this is true. We can use the results object's dot-plot-underscore-diagnostics method to generate four common plots for evaluating this. These are shown on the right.

**7. Residuals plot**

01:30 - 01:41

One of the four plots shows the one-step-ahead standardized residuals. If our model is working correctly, there should be no obvious structure in the residuals.

**8. Residuals plot**

01:41 - 01:47

Here the plot on the left has no obvious pattern, but the plot on the right does.

**9. Histogram plus estimated density**

01:47 - 02:14

Another of the four plots, shows us the distribution of the residuals. The histogram shows us the measured distribution; the orange line shows a smoothed version of this histogram; and the green line, shows a normal distribution. If our model is good these two lines should be almost the same. Here, the plot on the left looks fine, but the plot on the right doesn't.

**10. Normal Q-Q**

02:14 - 02:32

The normal Q-Q plot is another way to show how the distribution of the model residuals compares to a normal distribution. If our residuals are normally distributed then all the points should lie along the red line, except perhaps some values at either end.

**11. Correlogram**

02:32 - 02:52

The last plot is the correlogram, which is just an ACF plot of the residuals rather than the data. 95% of the correlations for lag greater than zero should not be significant. If there is significant correlation in the residuals, it means that there is information in the data that our model hasn't captured.

**12. Summary statistics**

02:52 - 03:24

Some of these plots also have accompanying test statistics in results dot-summary tables. Prob(Q) is the p-value associated with the null hypothesis that the residuals have no correlation structure. Prob(JB) is the p-value associated with the null hypothesis that the residuals are Gaussian normally distributed. If either p-value is less than 0.05 we reject that hypothesis.

**13. Let's practice!**

03:24 - 03:29

Now let's use some of these new tools in practice!

## Exercise

# Mean absolute error

Obviously, before you use the model to predict, you want to know how accurate your predictions are. The mean absolute error (MAE) is a good statistic for this. It is the mean difference between your predictions and the true values.

In this exercise you will calculate the MAE for an ARMA(1,1) model fit to the earthquakes time series

numpy has been imported into your environment as np and the earthquakes time series is available for you as earthquake.

## Instructions

100 XP

* Use np functions to calculate the Mean Absolute Error (MAE) of the .resid attribute of the results object.
* Print the MAE.
* Use the DataFrame's .plot() method with no arguments to plot the earthquake time series.

# Fit model

model = ARIMA(earthquake, order=(1,0,1))

results = model.fit()

# Calculate the mean absolute error from residuals

mae = \_\_\_\_

# Print mean absolute error

print(\_\_\_\_)

# Make plot of time series for comparison

\_\_\_\_

plt.show()

# Fit model

model = ARIMA(earthquake, order=(1,0,1))

results = model.fit()

# Calculate the mean absolute error from residuals

mae = np.mean(np.abs(results.resid))

# Print mean absolute error

print(mae)

# Make plot of time series for comparison

earthquake.plot()

plt.show()

**# Fit model**

**model = ARIMA(earthquake, order=(1,0,1))**

**results = model.fit()**

**# Calculate the mean absolute error from residuals**

**mae = np.mean(np.abs(results.resid))**

**# Print mean absolute error**

**print(mae)**

**# Make plot of time series for comparison**

**earthquake.plot()**

**plt.show()**

**4.568988294963051**

**Great! Your mean error is about 4-5 earthquakes per year. You have plotted the time series so that you can see how the MAE compares to the spread of the time series. Considering that there are about 20 earthquakes per year that is not too bad.**

# Diagnostic summary statistics

It is important to know when you need to go back to the drawing board in model design. In this exercise you will use the residual test statistics in the results summary to decide whether a model is a good fit to a time series.

Here is a reminder of the tests in the model summary:

|  |  |  |
| --- | --- | --- |
| **Test** | **Null hypothesis** | **P-value name** |
| Ljung-Box | There are no correlations in the residual | Prob(Q) |
| Jarque-Bera | The residuals are normally distributed | Prob(JB) |

An unknown time series df and the ARIMA model class are available for you in your environment.

## Instructions 1/4

* **Fit an ARMA(3,1) model** to the time series df.
* Print the model summary.

# Create and fit model

model1 = ARIMA(df, order=\_\_\_\_)

results1 = model1.fit()

# Print summary

print(\_\_\_\_)

**# Create and fit model**

**model1 = ARIMA(df, order=(3,0,1))**

**results1 = model1.fit()**

**# Print summary**

**print(results1.summary())**

**# Create and fit model**

**model1 = ARIMA(df, order=(3,0,1))**

**results1 = model1.fit()**

**# Print summary**

**print(results1.summary())**

**SARIMAX Results**

**==============================================================================**

**Dep. Variable: y No. Observations: 400**

**Model: ARIMA(3, 0, 1) Log Likelihood -555.394**

**Date: Fri, 30 Jun 2023 AIC 1122.787**

**Time: 09:20:13 BIC 1146.736**

**Sample: 01-01-2013 HQIC 1132.272**

**- 02-04-2014**

**Covariance Type: opg**

**==============================================================================**

**coef std err z P>|z| [0.025 0.975]**

**------------------------------------------------------------------------------**

**const 0.0314 0.030 1.060 0.289 -0.027 0.090**

**ar.L1 0.0163 0.109 0.150 0.881 -0.197 0.230**

**ar.L2 0.2159 0.052 4.179 0.000 0.115 0.317**

**ar.L3 -0.4551 0.050 -9.050 0.000 -0.554 -0.357**

**ma.L1 -0.2650 0.114 -2.326 0.020 -0.488 -0.042**

**sigma2 0.9388 0.065 14.460 0.000 0.812 1.066**

**===================================================================================**

**Ljung-Box (L1) (Q): 0.00 Jarque-Bera (JB): 0.75**

**Prob(Q): 0.97 Prob(JB): 0.69**

**Heteroskedasticity (H): 1.22 Skew: -0.07**

**Prob(H) (two-sided): 0.26 Kurtosis: 3.15**

**===================================================================================**

**Warnings:**

**[1] Covariance matrix calculated using the outer product of gradients (complex-step).**

## Question

Based on the outcomes of the tests in the summary, which of the following is correct about the residuals of results1?

### Possible answers

**they are not correlated and are normally distributed**

the outcome of the test are not conclusive.

They are correlated and are normally distributed.

They are not correlated and are not normally distributed

**Fit an AR(2) model** to the time series df.

Print the model summary.

# Create and fit model

model2 = ARIMA(df, order=\_\_\_\_)

results2 = model2.fit()

# Print summary

print(\_\_\_\_)

# Create and fit model

model2 = ARIMA(df, order=(2,0,0))

results2 = model2.fit()

# Print summary

print(results2.summary())

**# Print summary**

**print(results2.summary())**

**SARIMAX Results**

**==============================================================================**

**Dep. Variable: y No. Observations: 400**

**Model: ARIMA(2, 0, 0) Log Likelihood -590.964**

**Date: Fri, 30 Jun 2023 AIC 1189.928**

**Time: 09:28:44 BIC 1205.894**

**Sample: 01-01-2013 HQIC 1196.251**

**- 02-04-2014**

**Covariance Type: opg**

**==============================================================================**

**coef std err z P>|z| [0.025 0.975]**

**------------------------------------------------------------------------------**

**const 0.0341 0.051 0.662 0.508 -0.067 0.135**

**ar.L1 -0.3082 0.047 -6.554 0.000 -0.400 -0.216**

**ar.L2 0.2664 0.045 5.981 0.000 0.179 0.354**

**sigma2 1.1231 0.082 13.740 0.000 0.963 1.283**

**===================================================================================**

**Ljung-Box (L1) (Q): 4.40 Jarque-Bera (JB): 0.24**

**Prob(Q): 0.04 Prob(JB): 0.89**

**Heteroskedasticity (H): 1.18 Skew: -0.02**

**Prob(H) (two-sided): 0.34 Kurtosis: 2.89**

**===================================================================================**

**Warnings:**

**[1] Covariance matrix calculated using the outer product of gradients (complex-step).**

## Question

Based on the outcomes of the tests in the summary, which of the following is correct about the residuals of results2?

### Possible answers

The outcome of tests are not conclusive.

They are not correlated and are normally distributed.

They are correlated and are normally distributed.

They are correlated and are not normally distributed.

**Great. Our model didn't pull out all the correlations in the data. This suggests we could make it better. Perhaps by increasing the model order.**

## Exercise

# Plot diagnostics

It is important to know when you need to go back to the drawing board in model design. In this exercise you will use 4 common plots to decide whether a model is a good fit to some data.

Here is a reminder of what you would like to see in each of the plots for a model that fits well:

|  |  |
| --- | --- |
| **Test** | **Good fit** |
| Standardized residual | There are no obvious patterns in the residuals |
| Histogram plus kde estimate | The KDE curve should be very similar to the normal distribution |
| Normal Q-Q | Most of the data points should lie on the straight line |
| Correlogram | 95% of correlations for lag greater than zero should not be significant |

An unknown time series df and the ARIMA model class are available for you in your environment.-

## Instructions 1/3

Fit an ARIMA(1,1,1) model to the time series df.

Create the 4 diagnostic plots.

# Create and fit model

model = ARIMA(df, order=(\_\_\_\_))

results = model.fit()

# Create the 4 diagnostics plots

\_\_\_\_

plt.show()

# Create and fit model

model = ARIMA(df, order=(1,1,1))

results = model.fit()

# Create the 4 diagnostics plots

results.plot\_diagnostics()

plt.show()

# Create and fit model model = ARIMA(df, order=(1,1,1)) results = model.fit() # Create the 4 diagnostics plots results.plot\_diagnostics() plt.show()

## Question

Do these plots suggest that any of these are true about the model fit.

### Possible answers

The residuals are not normally distributed. You should try increasing d

The residuals are correlated. You should increase p or q.

**None of the above.**

**Based on the Q-Q plot and the histogram plot, the residuals do seem normally distributed.**

**The amount of correlation at each lag in the correlogram is not significant. The residuals seem uncorrelated.**

## Question

Below are 4 different diagnostic plots, each of the 4 plots comes from a different fitted model.

A picture containing text, diagram, screenshot, plot

Description automatically generated

Which of the plots above suggest that the fitted model could be improved?

### Possible answers

Standardized residuals It can be hard to tell from looking at a plot of residuals, but these ones look okay. There aren't any obvious patterns.

Histogram plus estimated density The distribution of residuals don't seem to be significantly different from normally distributed.

Normal Q-Q Great! The Q-Q plot deviates significantly from a straight line! This suggests the model could be improved.

Correlogram

None

**1. Box-Jenkins method**

00:00 - 00:11

You've learned lots of tools and methods for working with and modeling time series. In this lesson you will learn about the best practices framework for using these tools.

**2. The Box-Jenkins method**

00:11 - 00:45

Building time series models can represent a lot of work for the modeler and so we want to maximize our ability to carry out these projects fast, efficiently and rigorously. This is where the Box-Jenkins method comes in. The Box-Jenkins method is a kind of checklist for you to go from raw data to a model ready for production. The three main steps that stand between you and a production-ready model are identification, estimation and model diagnostics.

**3. Identification**

00:45 - 01:10

In the identification step we explore and characterize the data to find some form of it which is appropriate to ARIMA modeling. We need to know whether the time series is stationary and find which transformations, such as differencing or taking the log of the data, will make it stationary. Once we have found a stationary form, we must identify which orders p and q are the most promising.

**4. Identification tools**

01:10 - 01:34

Our tools to test for stationarity include plotting the time series and using the augmented Dicky-Fuller test. Then we can take the difference or apply transformations until we find the simplest set of transformations that make the time series stationary. Finally we use the ACF and PACF to identify promising model orders.

**5. Estimation**

01:34 - 01:56

The next step is estimation, which involves using numerical methods to estimate the AR and MA coefficients of the data. Thankfully, this is automatically done for us when we call the model's dot-fit method. At this stage we might fit many models and use the AIC and BIC to narrow down to more promising candidates.

**6. Model diagnostics**

01:56 - 02:09

In the model diagnostics step, we evaluate the quality of the best fitting model. Here is where we use our test statistics and diagnostic plots to make sure the residuals are well behaved.

**7. Decision**

02:09 - 02:22

Using the information gathered from statistical tests and plots during the diagnostic step, we need to make a decision. Is the model good enough or do we need to go back and rework it.

**8. Repeat**

02:22 - 02:29

If the residuals aren't as they should be we will go back and rethink our choices in the earlier steps.

**9. Production**

02:29 - 02:34

If the residuals are okay then we can go ahead and make forecasts!

**10. Box-Jenkins**

02:34 - 02:49

This should be your general project workflow when developing time series models. You may have to repeat the process a few times in order to build a model that fits well. But as they say, no pain, no gain.

**11. Let's practice!**

02:49 - 03:00

In the following exercise you will go through these steps to take an unknown time series and make a model ready for forecasting. Let's go!

## Exercise

# Identification

In the following exercises you will apply to the Box-Jenkins methodology to go from an unknown dataset to a model which is ready to make forecasts.

You will be using a new time series. This is the personal savings as % of disposable income 1955-1979 in the US.

The first step of the Box-Jenkins methodology is Identification. In this exercise you will use the tools at your disposal to test whether this new time series is stationary.

The time series has been loaded in as a DataFrame savings and the adfuller() function has been imported.

## Instructions

100 XP

* Plot the time series using the DataFrame's .plot() method.
* Apply the Dicky-Fuller test to the 'savings' column of the savings DataFrame and assign the test outcome to result.
* Print the Dicky-Fuller test statistics and the associated p-value.

# Plot time series

\_\_\_\_

plt.show()

# Run Dicky-Fuller test

result = \_\_\_\_

# Print test statistic

\_\_\_\_

# Print p-value

\_\_\_\_

# Plot time series

savings.plot()

plt.show()

# Run Dicky-Fuller test

result = adfuller(savings['savings'])

# Print test statistic

print(result[0])

# Print p-value

print(result[1])

**# Plot time series**

**savings.plot()**

**plt.show()**

**# Run Dicky-Fuller test**

**result = adfuller(savings['savings'])**

**# Print test statistic**

**print(result[0])**

**# Print p-value**

**print(result[1])**

**-3.185899096242141**

**0.020815541644114092**

**Great! The Dicky-Fuller test says that the series is stationary. You can confirm this when you look at the plot. There is one fairly high value is 1976 which might be anomalous, but you will leave that for now.**

# Identification II

You learned that the savings time series is stationary without differencing. Now that you have this information you can try and identify what order of model will be the best fit.

The plot\_acf() and the plot\_pacf() functions have been imported and the time series has been loaded into the DataFrame savings.

## Instructions

100 XP

* Make a plot of the ACF, for lags 1-10 and plot it on axis ax1.
* Do the same for the PACF.

# Create figure

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))

# Plot the ACF of savings on ax1

\_\_\_\_

# Plot the PACF of savings on ax2

\_\_\_\_

plt.show()

# Create figure

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8))

# Plot the ACF of savings on ax1

plot\_acf(savings, lags=10, zero=False,ax=ax1)

# Plot the PACF of savings on ax2

plot\_pacf(savings,lags=10, zero=False,ax=ax2)

plt.show()

# Create figure fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,8)) # Plot the ACF of savings on ax1 plot\_acf(savings, lags=10, zero=False,ax=ax1) # Plot the PACF of savings on ax2 plot\_pacf(savings,lags=10, zero=False,ax=ax2) plt.show()zero=False

**Step one complete! The ACF and the PACF are a little inconclusive for this ones. The ACF tails off nicely but the PACF might be tailing off or it might be dropping off. So it could be an ARMA(p,q) model or a AR(p) model.**

## Exercise

# Estimation

In the last exercise, the ACF and PACF were a little inconclusive. The results suggest your data could be an ARMA(p,q) model or could be an imperfect AR(3) model. In this exercise you will search over models over some model orders to find the best one according to AIC.

The time series savings has been loaded and the ARIMA class has been imported into your environment.

## Instructions

100 XP

* Loop over values of p from 0 to 3 and values of q from 0 to 3.
* Inside the loop, create an ARMA(p,q) model.
* Then fit the model to the time series savings.
* At the end of each loop print the values of p and q and the AIC and BIC.

# Loop over p values from 0-3

for p in \_\_\_\_:

  # Loop over q values from 0-3

    for q in \_\_\_\_:

      try:

        # Create and fit ARMA(p,q) model

        model = \_\_\_\_(\_\_\_\_, order=\_\_\_\_)

        results = \_\_\_\_

        # Print p, q, AIC, BIC

        print(\_\_\_\_)

      except:

        print(p, q, None, None)

# Loop over p values from 0-3

for p in range(4):

  # Loop over q values from 0-3

    for q in range(4):

      try:

        # Create and fit ARMA(p,q) model

        model = ARIMA(savings, order=(p,0,q))

        results = model.fit()

        # Print p, q, AIC, BIC

        print(p,q,results.aic,results.bic)

      except:

        print(p, q, None, None)

**# Loop over p values from 0-3**

**for p in range(4):**

**# Loop over q values from 0-3**

**for q in range(4):**

**try:**

**# Create and fit ARMA(p,q) model**

**model = ARIMA(savings, order=(p,0,q))**

**results = model.fit()**

**# Print p, q, AIC, BIC**

**print(p,q,results.aic,results.bic)**

**except:**

**print(p, q, None, None)**

**0 0 313.6028657381061 318.85281136467466**

**0 1 267.06970980844704 274.94462824829986**

**0 2 232.16782677363798 242.66771802677508**

**0 3 217.59720509753365 230.72206916395498**

**1 0 216.203479563773 224.0783980036258**

**1 1 215.7003889529165 226.2002802060536**

**1 2 207.6529838444748 220.77784791089613**

**1 3 209.57498315813876 225.3248200378444**

**2 0 213.97232199692382 224.47221325006092**

**2 1 213.4303586787578 226.55522274517915**

**2 2 209.57903144324032 225.32886832294594**

**2 3 211.57498195187546 229.94979164486534**

**3 0 209.54492936717307 222.6697934335944**

**3 1 210.82147284903212 226.57130972873773**

**3 2 211.45759548497693 229.83240517796682**

**3 3 213.35650857579535 234.3562910820695**

**Step two complete! You didn't store and sort your results this time. But the AIC and BIC both picked the ARMA(1,2) model as the best and the AR(3) model as the second best.**

# Diagnostics

You have arrived at the model diagnostic stage. So far you have found that the initial time series was stationary, but may have one outlying point. You identified promising model orders using the ACF and PACF and confirmed these insights by training a lot of models and using the AIC and BIC.

You found that the ARMA(1,2) model was the best fit to our data and now you want to check over the predictions it makes before you would move it into production.

The time series savings has been loaded and the ARIMA class has been imported into your environment.

## Instructions

100 XP

* Retrain the ARMA(1,2) model on the time series, setting the trend to constant.
* Create the 4 standard diagnostics plots.
* Print the model residual summary statistics.

# Create and fit model

model = \_\_\_\_

results = \_\_\_\_

# Create the 4 diagostics plots

\_\_\_\_

plt.show()

# Print summary

\_\_\_\_

# Create and fit model

model = ARIMA(savings, order=(1,2,1))

results = model.fit()

# Create the 4 diagostics plots

results.plot\_diagnostics()

plt.show()

# Print summary

results.summary()

**# Create and fit model**

**model = ARIMA(savings, order=(1,0,2))**

**results = model.fit()**

**# Create the 4 diagostics plots**

**results.plot\_diagnostics()**

**plt.show()**

**# Print summary**

**print(results.summary())**

**SARIMAX Results**

**==============================================================================**

**Dep. Variable: savings No. Observations: 102**

**Model: ARIMA(1, 0, 2) Log Likelihood -98.826**

**Date: Fri, 30 Jun 2023 AIC 207.653**

**Time: 11:08:37 BIC 220.778**

**Sample: 01-01-1955 HQIC 212.968**

**- 04-01-1980**

**Covariance Type: opg**

**==============================================================================**

**coef std err z P>|z| [0.025 0.975]**

**------------------------------------------------------------------------------**

**const 6.1949 0.323 19.169 0.000 5.561 6.828**

**ar.L1 0.7284 0.111 6.534 0.000 0.510 0.947**

**ma.L1 -0.0538 0.145 -0.370 0.711 -0.338 0.231**

**ma.L2 0.3681 0.097 3.814 0.000 0.179 0.557**

**sigma2 0.4012 0.043 9.264 0.000 0.316 0.486**

**===================================================================================**

**Ljung-Box (L1) (Q): 0.02 Jarque-Bera (JB): 55.12**

**Prob(Q): 0.89 Prob(JB): 0.00**

**Heteroskedasticity (H): 2.61 Skew: 0.82**

**Prob(H) (two-sided): 0.01 Kurtosis: 6.20**

**===================================================================================**

**Warnings:**

**[1] Covariance matrix calculated using the outer product of gradients (complex-step).**

**Great! The JB p-value is zero, which means you should reject the null hypothesis that the residuals are normally distributed. However, the histogram and Q-Q plots show that the residuals look normal. This time the JB value was thrown off by the one outlying point in the time series. In this case, you could go back and apply some transformation to remove this outlier or you probably just continue to the production stage.**

# The Best of the Best Models

In this chapter, you will become a modeler of discerning taste. You'll learn how to identify promising model orders from the data itself, then, once the most promising models have been trained, you'll learn how to choose the best model from this fitted selection. You'll also learn a great framework for structuring your time series projects. This is the Summary of lecture "ARIMA Models in Python", via datacamp.

Jun 16, 2020 • Chanseok Kang • 12 min read

[Python](https://goodboychan.github.io/categories/#Python)   [Datacamp](https://goodboychan.github.io/categories/#Datacamp)   [Time\_Series\_Analysis](https://goodboychan.github.io/categories/#Time_Series_Analysis)

* [Intro to ACF and PACF](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Intro-to-ACF-and-PACF) 
  + [AR or MA](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#AR-or-MA)
  + [Order of earthquakes](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Order-of-earthquakes)
* [Intro to AIC and BIC](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Intro-to-AIC-and-BIC) 
  + [Searching over model order](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Searching-over-model-order)
  + [Choosing order with AIC and BIC](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Choosing-order-with-AIC-and-BIC)
  + [AIC and BIC vs ACF and PACF](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#AIC-and-BIC-vs-ACF-and-PACF)
* [Model diagnostics](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Model-diagnostics) 
  + [Mean absolute error](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Mean-absolute-error)
  + [Diagnostic summary statistics](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Diagnostic-summary-statistics)
  + [Plot diagnostics](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Plot-diagnostics)
* [Box-Jenkins method](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Box-Jenkins-method) 
  + [Identification](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Identification)
  + [Identification II](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Identification-II)
  + [Estimation](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Estimation)
  + [Diagnostics](https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/16/01-The-Best-of-the-Best-Models.html#Diagnostics)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

plt.rcParams['figure.figsize'] = (10, 5)

plt.style.use('fivethirtyeight')

## Intro to ACF and PACF

* ACF : AutoCorrelation Function \ : Correlation between time series and the same time series offset by n-step
  + lag-1 autocorrelation -> corr(yt,yt−1)corr(yt​,yt−1​)
  + lag-2 autocorrelation -> corr(yt,yt−2)corr(yt​,yt−2​)
  + ……
  + lag-n autocorrelation -> corr(yt,yt−n)corr(yt​,yt−n​)
  + If ACF values are small and lie inside the blue shaded region, then they are not statistically significant.
* PACF : Partial AutoCorrelation Function \ : Correlation between time series and lagged version of itself after we subtract the effect of correlation at smaller lags.
* Using ACF and PACF to choose model order

|  | **AR(p)** | **MA(q)** | **ARMA(p, q)** |
| --- | --- | --- | --- |
| ACF | Tails off | Cuts off after lag q | Tails off |
| PACF | Cuts off after lag p | Tails off | Tails off |

### AR or MA

In this exercise you will use the ACF and PACF to decide whether some data is best suited to an MA model or an AR model. Remember that selecting the right model order is of great importance to our predictions.

df = pd.read\_csv('./dataset/sample2.csv', index\_col=0, parse\_dates=True)

df = df.asfreq('D')

df.head()

|  | **y** |
| --- | --- |
| **2013-01-01** | 1.624345 |
| **2013-01-02** | -0.936625 |
| **2013-01-03** | 0.081483 |
| **2013-01-04** | -0.663558 |
| **2013-01-05** | 0.738023 |

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# Create figure

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12,8))

# Plot the ACF of df

plot\_acf(df, lags=10, zero=False, ax=ax1);

# Plot the PACF of df

plot\_pacf(df, lags=10, zero=False, ax=ax2);

**A picture containing text, screenshot, line, diagram

Description automatically generated**

Based on the ACF and PACF plots, This is MA(3) model.

### Order of earthquakes

In this exercise you will use the ACF and PACF plots to decide on the most appropriate order to forecast the earthquakes time series.

earthquake = pd.read\_csv('./dataset/earthquakes.csv', index\_col='date', parse\_dates=True)

earthquake.drop(['Year'], axis=1, inplace=True)

earthquake = earthquake.asfreq('AS-JAN')

earthquake.head()

|  | **earthquakes\_per\_year** |
| --- | --- |
| **date** |  |
| **1900-01-01** | 13.0 |
| **1901-01-01** | 14.0 |
| **1902-01-01** | 8.0 |
| **1903-01-01** | 10.0 |
| **1904-01-01** | 16.0 |

from statsmodels.tsa.statespace.sarimax import SARIMAX

# Create figure

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# Plot ACF and PACF

plot\_acf(earthquake, lags=15, zero=False, ax=ax1);

plot\_pacf(earthquake, lags=15, zero=False, ax=ax2);

**A picture containing text, diagram, line, screenshot

Description automatically generated**

Based on the ACF and PACF plots, This is AR(1) model.

model = SARIMAX(earthquake, order=(1, 0, 0))

# Train model

results = model.fit()

## Intro to AIC and BIC

* AIC (Akaike Information Criterion)
  + Lower AIC indicates a better model
  + AIC likes to choose simple models with lower order
* BIC (Bayesian Information Criterion)
  + Very similar to AIC
  + Lower BIC indicates a better model
  + BIC likes to choose simple models with lower order
* AIC vs BIC
  + The difference between two metrics is how much they penalize model complexity
  + BIC favors simpler models than AIC
  + AIC is better at choosing predictive models
  + BIC is better at choosing good explanatory model

### Searching over model order

In this exercise you are faced with a dataset which appears to be an ARMA model. You can see the ACF and PACF in the plot below. In order to choose the best order for this model you are going to have to do a search over lots of potential model orders to find the best set.

df = pd.read\_csv('./dataset/sample2.csv', index\_col=0, parse\_dates=True)

df = df.asfreq('D')

# Create figure

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12,8))

# Plot the ACF of df

plot\_acf(df, lags=10, zero=False, ax=ax1);

# Plot the PACF of df

plot\_pacf(df, lags=10, zero=False, ax=ax2);

**A picture containing text, screenshot, line, diagram

Description automatically generated**

order\_aic\_bic = []

# Loop over p values from 0-2

for p in range(3):

# Loop over q values from 0-2

for q in range(3):

# Create and fit ARMA(p, q) model

model = SARIMAX(df, order=(p, 0, q))

results = model.fit()

# Append order and results tuple

order\_aic\_bic.append((p, q, results.aic, results.bic))

/home/chanseok/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.py:963: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.

warn('Non-stationary starting autoregressive parameters'

/home/chanseok/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.py:975: UserWarning: Non-invertible starting MA parameters found. Using zeros as starting parameters.

warn('Non-invertible starting MA parameters found.'

### Choosing order with AIC and BIC

Now that you have performed a search over many model orders, you will evaluate your results to find the best model order.

order\_df = pd.DataFrame(order\_aic\_bic, columns=['p', 'q', 'AIC', 'BIC'])

# Print order\_df in order of increasing AIC

print(order\_df.sort\_values('AIC'))

# Print order\_df in order of increasing BIC

print(order\_df.sort\_values('BIC'))

p q AIC BIC

8 2 2 2808.309189 2832.847965

5 1 2 2817.292441 2836.923462

2 0 2 2872.205748 2886.929013

7 2 1 2889.542335 2909.173356

6 2 0 2930.299481 2945.022747

4 1 1 2960.351104 2975.074370

3 1 0 2969.236399 2979.051910

1 0 1 2978.726909 2988.542419

0 0 0 2996.526734 3001.434489

p q AIC BIC

8 2 2 2808.309189 2832.847965

5 1 2 2817.292441 2836.923462

2 0 2 2872.205748 2886.929013

7 2 1 2889.542335 2909.173356

6 2 0 2930.299481 2945.022747

4 1 1 2960.351104 2975.074370

3 1 0 2969.236399 2979.051910

1 0 1 2978.726909 2988.542419

0 0 0 2996.526734 3001.434489

Based on this result, ARMA(2,2) model is the best fit

### AIC and BIC vs ACF and PACF

In this exercise you will apply an AIC-BIC order search for the earthquakes time series. In the last lesson you decided that this dataset looked like an AR(1) process. You will do a grid search over parameters to see if you get the same results.

for p in range(3):

for q in range(3):

try:

# Create and fit ARMA(p, q) model

model = SARIMAX(earthquake, order=(p, 0, q))

results = model.fit()

# Print order and results

print(p, q, results.aic, results.bic)

except:

print(p, q, None, None)

0 0 888.4297722924081 891.0248921425426

0 1 799.6741727812074 804.8644124814766

0 2 761.0674787503889 768.8528383007927

1 0 666.6455255041611 671.8357652044303

1 1 647.1322999673836 654.9176595177873

1 2 648.7385664620616 659.1190458625999

2 0 656.0283744146394 663.8137339650432

2 1 648.8428399959623 659.2233193965006

2 2 648.850644343021 661.826243593694

/home/chanseok/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.py:963: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.

warn('Non-stationary starting autoregressive parameters'

## Model diagnostics

* Introduction to model diagnostics
  + How good is the final model?
    - Residual with mean absolute error
    - Normal Q-Q plot
    - Correlogram
    - Summary statistics

### Mean absolute error

Obviously, before you use the model to predict, you want to know how accurate your predictions are. The mean absolute error (MAE) is a good statistic for this. It is the mean difference between your predictions and the true values.

In this exercise you will calculate the MAE for an ARMA(1,1) model fit to the earthquakes time series

model = SARIMAX(earthquake, order=(1, 0, 1))

results = model.fit()

# Calculate the mean absolute error from residuals

mae = np.mean(np.abs(results.resid))

# Print mean absolute error

print(mae)

# Make plot of time series for comparison

earthquake.plot();

4.7556256718469045

**A picture containing text, font, plot, line

Description automatically generated**

### Diagnostic summary statistics

It is important to know when you need to go back to the drawing board in model design. In this exercise you will use the residual test statistics in the results summary to decide whether a model is a good fit to a time series.

Here is a reminder of the tests in the model summary:

| **Test** | **Null hypothesis** | **P-value** |
| --- | --- | --- |
| Ljung-Box | There are no correlations in the residual | Prob(Q) |
| Jarque-Bera | The residuals are normally distributed | Prob(JB) |

model1 = SARIMAX(df, order=(3, 0, 1))

results1 = model1.fit()

# Print summary

print(results1.summary())

SARIMAX Results

==============================================================================

Dep. Variable: y No. Observations: 1000

Model: SARIMAX(3, 0, 1) Log Likelihood -1421.678

Date: Tue, 16 Jun 2020 AIC 2853.356

Time: 10:58:31 BIC 2877.895

Sample: 01-01-2013 HQIC 2862.682

- 09-27-2015

Covariance Type: opg

==============================================================================

coef std err z P>|z| [0.025 0.975]

------------------------------------------------------------------------------

ar.L1 -0.0719 0.110 -0.653 0.514 -0.288 0.144

ar.L2 0.2542 0.034 7.573 0.000 0.188 0.320

ar.L3 0.2528 0.039 6.557 0.000 0.177 0.328

ma.L1 -0.1302 0.112 -1.158 0.247 -0.351 0.090

sigma2 1.0051 0.042 23.701 0.000 0.922 1.088

===================================================================================

Ljung-Box (Q): 65.16 Jarque-Bera (JB): 6.29

Prob(Q): 0.01 Prob(JB): 0.04

Heteroskedasticity (H): 1.01 Skew: -0.09

Prob(H) (two-sided): 0.94 Kurtosis: 3.34

===================================================================================

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

model2 = SARIMAX(df, order=(2, 0, 0))

results2 = model2.fit()

# Print summary

print(results2.summary())

SARIMAX Results

==============================================================================

Dep. Variable: y No. Observations: 1000

Model: SARIMAX(2, 0, 0) Log Likelihood -1462.150

Date: Tue, 16 Jun 2020 AIC 2930.299

Time: 10:58:31 BIC 2945.023

Sample: 01-01-2013 HQIC 2935.895

- 09-27-2015

Covariance Type: opg

==============================================================================

coef std err z P>|z| [0.025 0.975]

------------------------------------------------------------------------------

ar.L1 -0.1361 0.031 -4.400 0.000 -0.197 -0.075

ar.L2 0.2005 0.032 6.355 0.000 0.139 0.262

sigma2 1.0901 0.043 25.360 0.000 1.006 1.174

===================================================================================

Ljung-Box (Q): 126.14 Jarque-Bera (JB): 15.52

Prob(Q): 0.00 Prob(JB): 0.00

Heteroskedasticity (H): 1.02 Skew: -0.07

Prob(H) (two-sided): 0.85 Kurtosis: 3.59

===================================================================================

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

### Plot diagnostics

It is important to know when you need to go back to the drawing board in model design. In this exercise you will use 4 common plots to decide whether a model is a good fit to some data.

Here is a reminder of what you would like to see in each of the plots for a model that fits well:

| **Test** | **Good fit** |
| --- | --- |
| Standardized residual | There are no obvious patterns in the residuals |
| Histogram plus kde estimate | The KDE curve should be very similar to the normal distribution |
| Normal Q-Q | Most of the data points should lie on the straight line |
| Correlogram | 95% of correlations for lag greater than one should not be significant |

model = SARIMAX(df, order=(1, 1, 1))

results=model.fit()

# Create the 4 diagostics plots

results.plot\_diagnostics(figsize=(20, 10));

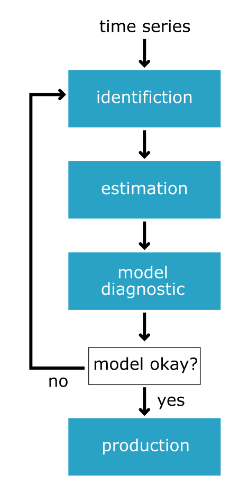
plt.tight\_layout();

plt.savefig('../images/plot\_diagnostics.png')

**A picture containing text, diagram, plot, line

Description automatically generated**

## Box-Jenkins method



* From raw data -> production model
  + identification
  + estimation
  + model diagnostics
* Identification
  + Is the time series stationary?
  + What differencing will make it stationary?
  + What transforms will make it stationary?
  + What values of p and q are most promising?
* Estimation
  + Use the data to train the model coefficient
  + Done for us using model.fit()
  + Choose between models using AIC and BIC
* Model Diagnostics
  + Are the residuals uncorrelated
  + Are residuals normally distributed

### Identification

In the following exercises you will apply to the Box-Jenkins methodology to go from an unknown dataset to a model which is ready to make forecasts.

You will be using a new time series. This is the personal savings as % of disposable income 1955-1979 in the US.

The first step of the Box-Jenkins methodology is Identification. In this exercise you will use the tools at your disposal to test whether this new time series is stationary.

savings = pd.read\_csv('./dataset/savings.csv', parse\_dates=True, index\_col='date')

savings = savings.asfreq('QS')

savings.head()

|  | **savings** |
| --- | --- |
| **date** |  |
| **1955-01-01** | 4.9 |
| **1955-04-01** | 5.2 |
| **1955-07-01** | 5.7 |
| **1955-10-01** | 5.7 |
| **1956-01-01** | 6.2 |

from statsmodels.tsa.stattools import adfuller

# Plot time series

savings.plot();

# Run Dicky-Fuller test

result = adfuller(savings['savings'])

# Print test statistics

print(result[0])

# Print p-value

print(result[1])

(-3.1858990962421405, 0.020815541644114133, 2, 99, {'1%': -3.498198082189098, '5%': -2.891208211860468, '10%': -2.5825959973472097}, 188.1686662239687)

0.020815541644114133

**A picture containing plot, text, line, font

Description automatically generated**

### Identification II

You learned that the savings time series is stationary without differencing. Now that you have this information you can try and identify what order of model will be the best fit.

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

# Plot the ACF of savings on ax1

plot\_acf(savings, lags=10, zero=False, ax=ax1);

# Plot the PACF of savings on ax2

plot\_pacf(savings, lags=10, zero=False, ax=ax2);

**A picture containing text, screenshot, line, diagram

Description automatically generated**

The ACF and the PACF are a little inconclusive for this ones. The ACF tails off nicely but the PACF might be tailing off or it might be dropping off. So it could be an ARMA(p,q) model or a AR(p) model.

### Estimation

In the last exercise, the ACF and PACF were a little inconclusive. The results suggest your data could be an ARMA(p,q) model or could be an imperfect AR(3) model. In this exercise you will search over models over some model orders to find the best one according to AIC.

for p in range(4):

# Loop over q values from 0-3

for q in range(4):

try:

# Create and fit ARMA(p, q) model

model = SARIMAX(savings, order=(p, 0, q), trend='c')

results = model.fit()

# Print p, q, AIC, BIC

print(p, q, results.aic, results.bic)

except:

print(p, q, None, None)

0 0 313.6028657326894 318.85281135925794

0 1 267.06970976886913 274.94462820872195

0 2 232.16782676455585 242.66771801769295

0 3 217.59720511188743 230.7220691783088

1 0 216.20348062499983 224.07839906485265

1 1 215.7003896386748 226.2002808918119

1 2 207.65298608433693 220.7778501507583

1 3 209.57498691600946 225.32482379571508

2 0 213.9723232754384 224.4722145285755

2 1 213.43035679044817 226.55522085686954

/home/chanseok/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/sarimax.py:963: UserWarning: Non-stationary starting autoregressive parameters found. Using zeros as starting parameters.

warn('Non-stationary starting autoregressive parameters'

2 2 209.57903436790426 225.32887124760987

2 3 211.57503208933585 229.94984178232573

3 0 209.5449310791239 222.66979514554527

3 1 210.8214763494127 226.57131322911835

3 2 211.45759881817608 229.83240851116597

3 3 213.54389994712193 234.5436824533961

You didn't store and sort your results this time. But the AIC and BIC both picked the ARMA(1,2) model as the best and the AR(3) model as the second best.

### Diagnostics

You have arrived at the model diagnostic stage. So far you have found that the initial time series was stationary, but may have one outlying point. You identified promising model orders using the ACF and PACF and confirmed these insights by training a lot of models and using the AIC and BIC.

You found that the ARMA(1,2) model was the best fit to our data and now you want to check over the predictions it makes before you would move it into production.

model = SARIMAX(savings, order=(1, 0, 2), trend='c')

results = model.fit()

# Create the 4 diagnostics plots

results.plot\_diagnostics(figsize=(20, 15));

plt.tight\_layout()

# Print summary

print(results.summary())

SARIMAX Results

==============================================================================

Dep. Variable: savings No. Observations: 102

Model: SARIMAX(1, 0, 2) Log Likelihood -98.826

Date: Tue, 16 Jun 2020 AIC 207.653

Time: 10:54:42 BIC 220.778

Sample: 01-01-1955 HQIC 212.968

- 04-01-1980

Covariance Type: opg

==============================================================================

coef std err z P>|z| [0.025 0.975]

------------------------------------------------------------------------------

intercept 1.6813 0.714 2.356 0.018 0.283 3.080

ar.L1 0.7286 0.111 6.538 0.000 0.510 0.947

ma.L1 -0.0539 0.145 -0.372 0.710 -0.338 0.230

ma.L2 0.3680 0.097 3.812 0.000 0.179 0.557

sigma2 0.4012 0.043 9.265 0.000 0.316 0.486

===================================================================================

Ljung-Box (Q): 33.59 Jarque-Bera (JB): 55.13

Prob(Q): 0.75 Prob(JB): 0.00

Heteroskedasticity (H): 2.61 Skew: 0.82

Prob(H) (two-sided): 0.01 Kurtosis: 6.20

===================================================================================

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

**A picture containing text, diagram, plot, line

Description automatically generated**

The JB p-value is zero, which means you should reject the null hypothesis that the residuals are normally distributed. However, the histogram and Q-Q plots show that the residuals look normal. This time the JB value was thrown off by the one outlying point in the time series. In this case, you could go back and apply some transformation to remove this outlier or you probably just continue to the production stage.