**AR**: autoregression **MA:** Moving Average

ar\_model = ARMA(timeseries, order=(p,0))

ma\_model = ARMA(timeseries, order=(0,q))

Have to make sure data is stationary:

* Take diff using .diff().dropna()
* Take log transform using np.log(series/series.diff()).dropna()

adfuller(data)

* Coefficients should be large negative numbers
* P value less than 0.05 rejects null hypothesis that data is not stationary, so small p value means data is stationary

ARIMA Models:

Typical workflow: Non-stationary data 🡪 transform to stationary data for modeling 🡪 fit model to transformed data 🡪 make transformed predictions 🡪 un-transform to get real predictions

ARIMA can automate all this.

From statsmodels.tsa.statespace.sarimax import SARIMAX

Model = SARIMAX(df, order=(p,d,q))

-p = # of autoregressive lags

- d = order of differencing (if 0, this becomes simply an ARMA model)

- q = moving average order

Still have to work out best d using adfuller test, want to difference data until it’s stationary but NO MORE

**ACF and PACF**

* Autocorrelation function
* Partial autocorrelation function

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 |

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

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

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

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

plt.show

**AIC** – the better the model is **the lower the AIC score**. AIC penalizes lots of parameters, which stops us from overfitting, **better at choosing predictive models**

**BIC** – Bayseian information criterion – very similar to AIC, **lower BIC indicates better model**, likes to choose simple models with lower order, penalizes complexity more than AIC, **better at choosing explanatory models**

results.aic

results.bic

Searching for model order:

# Create empty list to store search results

order\_aic\_bic=[]

# Loop over p values from 0-2

for p in range(0,3):

# Loop over q values from 0-2

for q in range(0,3):

# 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))

# 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'))

# Print order\_df in order of increasing BIC

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

Then choose model order with lowest AIC and/or BIC, depending on goals. May sometimes need to use Try:, Except: if all model orders tested don’t work.

**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.

# Create and fit model

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

results1 = model1.fit()

# Print summary

print(results1.summary())

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) |

**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 |

# Create and fit model

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

results=model.fit()

# Create the 4 diagostics plots

results.plot\_diagnostics()

plt.show()

**The Box-Jenkins method**

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**

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**

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**

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**

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**

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**

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

**9. Production**

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

**10. Box-Jenkins**

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.

# Seasonal decompose

You can think of a time series as being composed of trend, seasonal and residual components. This can be a good way to think about the data when you go about modeling it. If you know the period of the time series you can decompose it into these components.

# Import seasonal decompose

from statsmodels.tsa.seasonal import seasonal\_decompose

# Perform additive decomposition

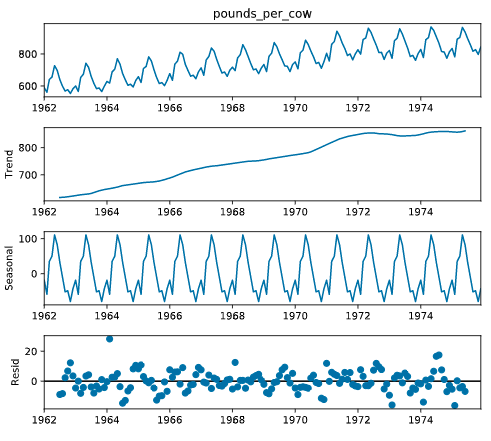
decomp = seasonal\_decompose(milk\_production['pounds\_per\_cow'],

period=12)

# Plot decomposition

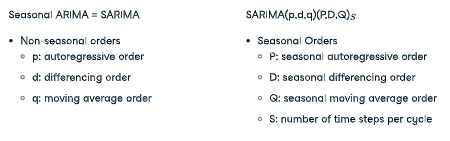
decomp.plot()

plt.show()



# Fitting SARIMA models seasonal ARIMA = SARIMA

Fitting SARIMA models is the beginning of the end of this journey into time series modeling.





# Machine Learning for Time Series Data in Python

[DataCamp](https://app.datacamp.com/learn/courses/machine-learning-for-time-series-data-in-python)

