← Go Back to Practical Data Science

Course Content

New Package Introduction - Time Series

Now, Let's go through some of the common functions used in this LVC implemented through the Statsmodels library

Statistical functions:

The Stats model contains classes and functions for time series analysis. Autoregressive models (AR), vector autoregressive models (VAR), and autoregressive moving average models (ARMA)are examples of basic models. Markov switching dynamic regression and autoregression are examples of non-linear models. It also includes time series descriptive statistics such as autocorrelation, partial autocorrelation function, and periodogram, as well as the theoretical properties of ARMA or related processes. Methods to work with autoregressive and moving average lag-polynomials are also included.

Statsmodels can be installed more easily as a part of the cross-platform Anaconda distribution, which is designed for scientific computing and data analysis.

Installation:

```
pip install statsmodels
```

After the installation of the statsmodels library, restart the jupyter kernel once and import the library into the working environment.

To import a statsmodels library, run the following command in a Jupyter notebook.

```
import statsmodels
```

i) ACF and PACF Plotting

An ACF is used to indicate how similar a value is to the previous value within a given time series. (Or) it helps to measure the degree of similarity between a given time series and the lagged version of that time series observed at various intervals.

Importing an ACF method

```
from statsmodels.graphics .tsaplots import plot_acf
```

```
statsmodels.graphics.tsaplots.plot_acf(x, ax=None, lags=None, *, alpha=0.05, use_vlines=True, adjusted=False, f
```

Parameters:

x = array_like

Array of time-series

lag= {int, array_like}, optional

An int or array of lag values, used on the horizontal axis. uses np.arange(lags) when lags is an int. If not provided, lags=np.arange(len(corr)) is used.

You can refer to the documentation for a better understanding of the parameters and attributes link

PACF

It always displays the sequence's correlation with itself with some number of time units per sequence order in which only the direct effect is displayed and all other intermediary effects are removed from the given time series.

Importing a PACF

```
from statsmodels.graphics .tsaplots import  plot_pacf
```

```
statsmodels.graphics.tsaplots.plot_pacf(x, ax=None, lags=None, alpha=0.05, method='ywm', use_vlines=True, title=
```

Parameters:

x=array_like

Array of time-series

lag= {int, array_like}, optional

You can refer to the documentation for a better understanding of the parameters and attributes link

Below we can find the function implementation in the Bitcoin_Price_Prediction case study:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plt.figure(figsize = (16, 8))

plot_acf(df_shift, lags = 12)

plt.show()

plot_pacf(df_shift, lags = 12)

plt.show()
```

ii) Augmented Dickey-Fuller (ADF):

The Augmented Dickey-Fuller test can be used to test for a unit root in a univariate process in the presence of serial correlation.

To import the required method, run the following command in a Jupyter notebook.

```
from statsmodels.tsa.stattools import adfuller
```

```
statsmodels.tsa.stattools.adfuller(x, maxlag=None, regression='c', autolag='AIC', store=False, regresults=False
```

Parameters:

x =array_like,1d

The data series to test.

Maxlag: {None,int}

Maximum lag which is included in the test, default value 12*(nobs/100)^{¼ } is used when None

Below we can find the function implementation in the Bitcoin_Price_Prediction case study:

```
# Define a function to use adfuller test
def adfuller(data):

  #Importing adfuller using statsmodels
    from statsmodels.tsa.stattools import adfuller

    print('Dickey-Fuller Test: ')

    adftest = adfuller(data['Close'])

    adfoutput = pd.Series(adftest[0:4], index = ['Test Statistic', 'p-value', 'Lags Used', 'No. of Observations'])

    for key, value in adftest[4].items():
        adfoutput['Critical Value (%s)'%key] = value

    print(adfoutput)

    adfuller(df_train)
```

You can refer to the documentation for a better understanding of the parameters and attributes link

iii) Decomposing the time-series components into a trend, seasonality, and residual

To import a necessary method, run the following command in a Jupyter notebook.

```
form statsmodels.tsa.seasonal import seasonal_decompose
```

```
statsmodels.tsa.seasonal.seasonal_decompose(x, model='additive', filt=None, period=None, two_sided=True, extrapo
```

Parameters:

x: array_like,

Time series, Individual series are in columns if 2d. x must have two complete cycles.

Period: int,optional

Period of the series.Must be used if x is not a Pandas object or if the index of x does not have a frequency. Overrides default periodicity of x if x is pandas object with a time series index.

Below we can find the function implementation in the Bitcoin_Price_Prediction case study:

```
# Importing the seasonal_decompose to decompose the time series
from statsmodels.tsa.seasonal import seasonal_decompose

decomp = seasonal_decompose(df_train)

trend = decomp.trend

seasonal = decomp.seasonal

residual = decomp.resid
```

You can refer to the documentation for a better understanding of the parameters and attributes link

iv) Auto Regressive Model:

The Autoregressive model is a type of random process. The model's output is linearly dependent on its own previous value, which is some number of lagged data points or past observations.

To import the required method, run the following command in a Jupyter notebook.

```
from statsmodels.tsa.ar_model. AutoReg
```

```
statsmodels.tsa.ar_model. AutoReg(endog, lags, trend='c', seasonal=False, exog=None, hold_back=None, period=None
```

Parameters:

endog: array_like

The dependent variable

lags: {None,int,list[int]}

The number of lags to include in the model is an integer or the list of lag indices to include. For example, [1, 4] will only include lags 1 and 4, while lags = 4 will include lags 1, 2, 3, and 4. None excludes all AR lags and behaves identically to 0.

Below we can find the function implementation in the Bitcoin_Price_Prediction case study:

```
# Importing AutoReg function to apply AR model
from statsmodels.tsa.ar_model import AutoReg

plt.figure(figsize = (16, 8))

# Using number of lags as 7
model_AR = AutoReg(df_shift, lags = 7)

results_AR = model_AR.fit()

plt.plot(df_shift)

predict = results_AR.predict(start = 0, end = len(df_shift) - 1)
```

You can refer to the documentation for a better understanding of the parameters and attributes link

V) ARIMA

An Auto regressive integrated moving average model is a generalization of a simple ARMA model. Both of these models are used to forecast or predict future time-series data points.

The ARIMA model is generally denoted as ARIMA(p, d, q) and the parameters p, d, q are defined as follows:

p: the lag order or the number of time lag of autoregressive model AR(p)

d: degree of differencing or the number of times the data have had subtracted with past value

q: the order of moving average model MA(q)

- **autoregressive models:** AR(p)
- **moving average models:** MA(q)
- **mixed autoregressive moving average models:** ARMA(p, q)
- **integration models:** ARIMA(p, d, q)
- **regression with errors that follow one of the above ARIMA-type models**

To import an ARIMA model, run the following command in a Jupyter notebook

```
from statsmodels.tsa.arima.model import ARMIA
```

```
statsmodels.tsa.arima.model. ARIMA(endog, exog=None, order=(0, 0, 0), seasonal_order=(0, 0, 0, 0), trend=None, e
```

Parameters:

endog: array_like, optional

The observed time-series process y

Exong: array_like, optional

Array of exogenous regressors

Below we can find the function implementation in the Bitcoin_Price_Prediction case study:

```
from statsmodels.tsa.arima_model import ARIMA

plt.figure(figsize = (16, 8))

# Using p = 7, d = 1, q = 7
model_ARIMA = ARIMA(df_shift, order = (7, 1, 7))

results_ARIMA = model_ARIMA.fit()

plt.plot(df_shift)

plt.plot(results_ARIMA.fittedvalues, color = 'red')
```