time-series-analysis-arima-1

May 14, 2024

1 Time series analysis with ARIMA

1.1 Import libraries and get sample data

[ ]:
```
import warnings
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.rcParams.update({'font.size': 12})
plt.style.use('ggplot')
```

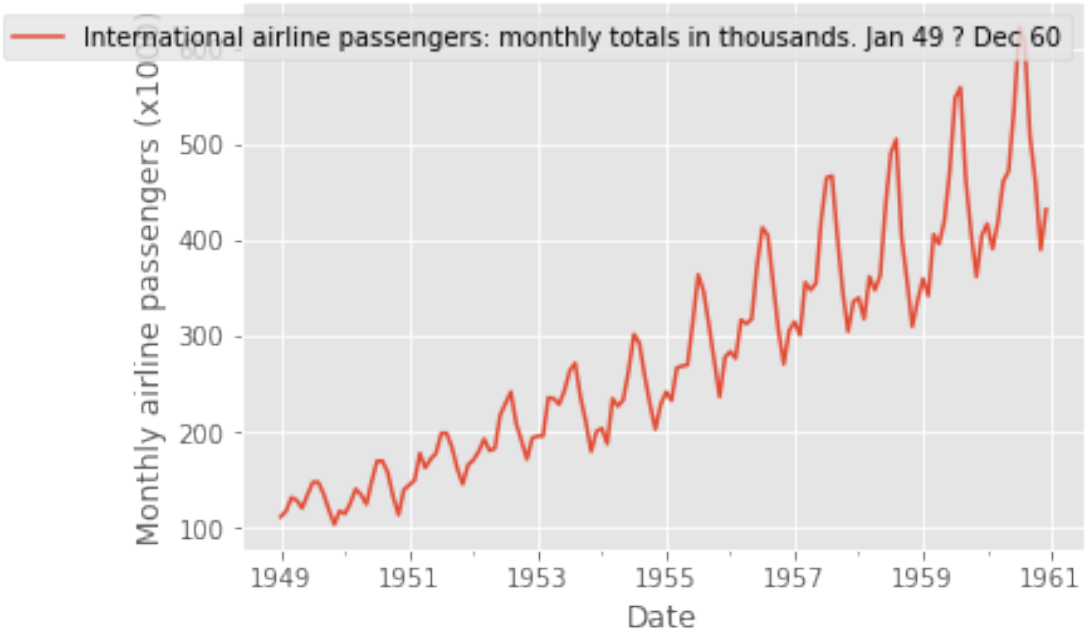
```
C:\Users\gmonaci\AppData\Local\Continuum\anaconda3\lib\site-
packages\statsmodels\compat\pandas.py:56: FutureWarning: The
pandas.core.datetools module is deprecated and will be removed in a future
version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

[ ]:
```
data = pd.read_csv('international-airline-passengers.csv', engine='python',␣
 ↪skipfooter=3)

data['Month']=pd.to_datetime(data['Month'], format='%Y-%m-%d')
data.set_index(['Month'], inplace=True)



data.plot()
plt.ylabel('Monthly airline passengers (x1000)')
plt.xlabel('Date')
plt.show()
```



[ ]:
```
q = d = range(0, 2)

p = range(0, 4)

pdq = list(itertools.product(p, d, q))

seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,␣
↪q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA…
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

We select a subset of the data series as training data, say the first 11 years. Our goal is to predict the last year of the series based on this input.

[ ]:
```
train_data = data['1949-01-01':'1959-12-01']
test_data = data['1960-01-01':'1960-12-01']
```

[ ]:
```
warnings.filterwarnings("ignore")

AIC = []
SARIMAX_model = []
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(train_data,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('SARIMAX{}x{} - AIC:{}'.format(param, param_seasonal, results.
 ↪aic), end='\r')
            AIC.append(results.aic)
            SARIMAX_model.append([param, param_seasonal])
        except:
            continue
```

```
SARIMAX(3, 1, 1)x(3, 1, 1, 12) - AIC:619.77849554151587
```

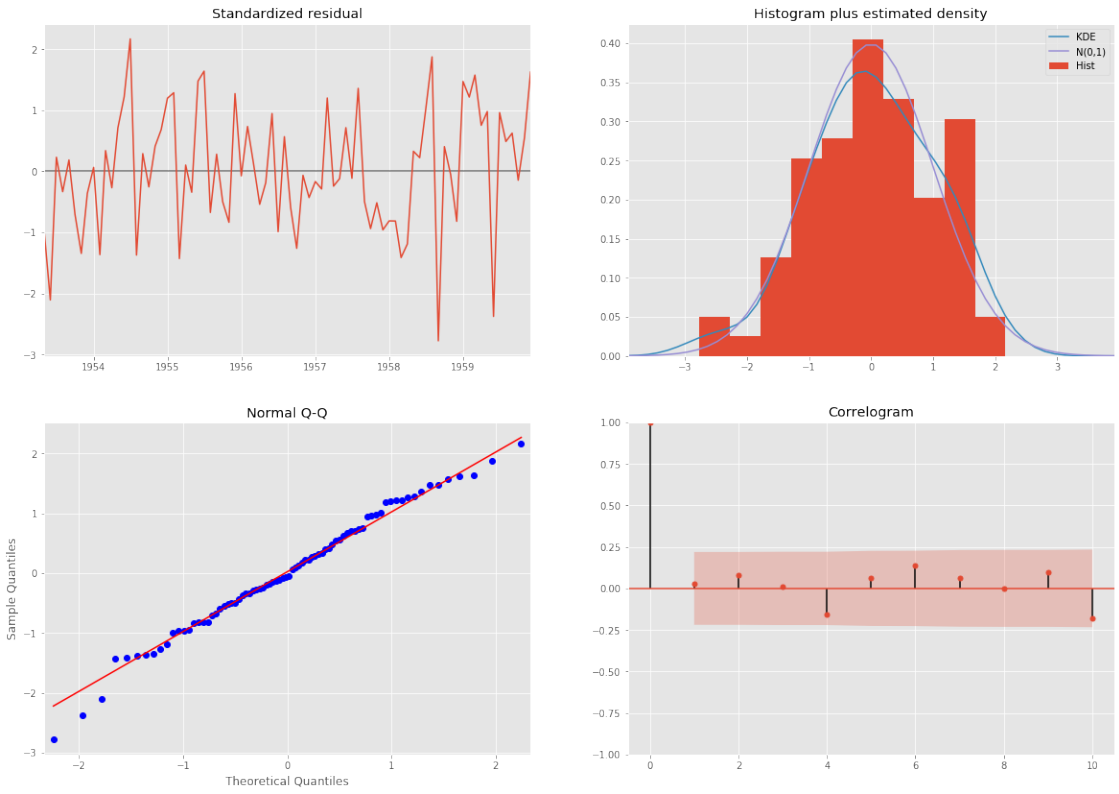
[ ]:
```
print('The smallest AIC is {} for model SARIMAX{}x{}'.format(min(AIC),␣
 ↪SARIMAX_model[AIC.index(min(AIC))][0],SARIMAX_model[AIC.index(min(AIC))][1]))
```

```
The smallest AIC is 618.2055110262379 for model SARIMAX(3, 1, 0)x(3, 1, 1, 12)
```

[ ]:
```
mod = sm.tsa.statespace.SARIMAX(train_data,
                                order=SARIMAX_model[AIC.index(min(AIC))][0],
                                seasonal_order=SARIMAX_model[AIC.
 ↪index(min(AIC))][1],
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()
```

[ ]:
```
results.plot_diagnostics(figsize=(20, 14))
plt.show()
```



[ ]:
```
pred0 = results.get_prediction(start='1958-01-01', dynamic=False)
pred0_ci = pred0.conf_int()
```

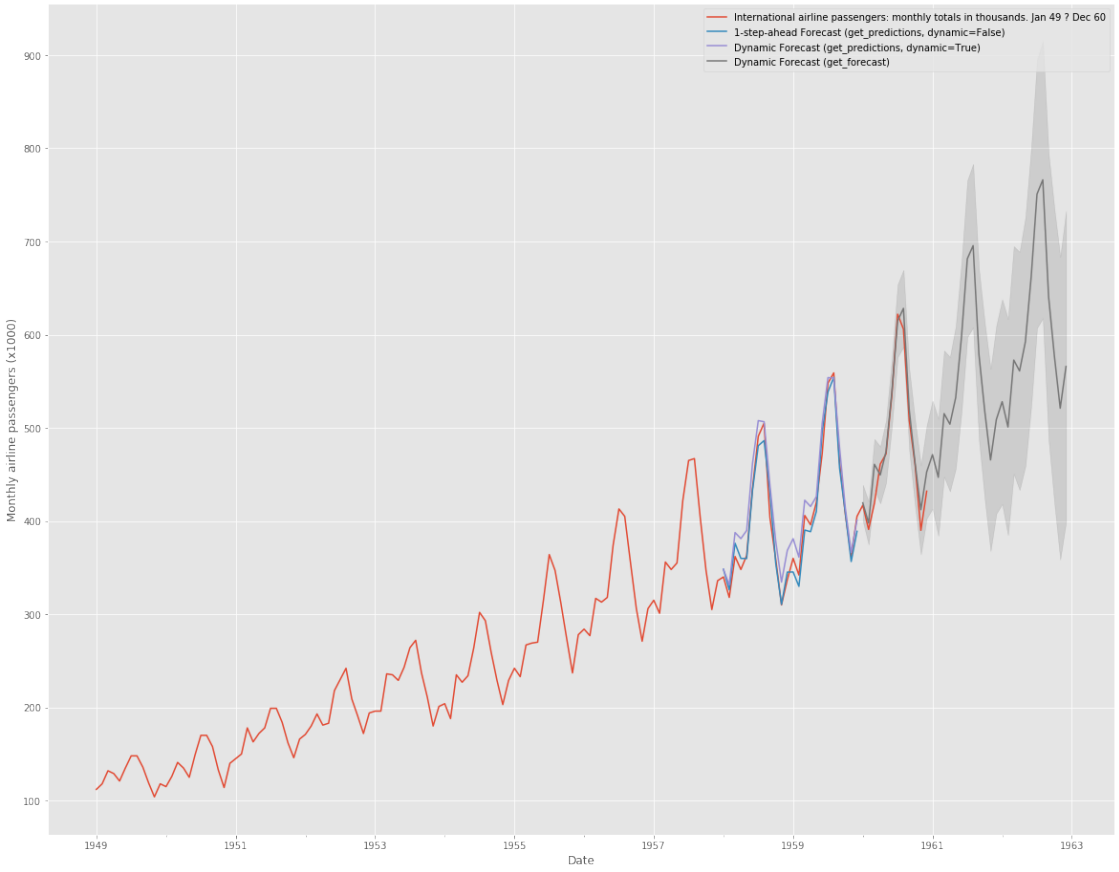
[ ]:
```
pred1 = results.get_prediction(start='1958-01-01', dynamic=True)
pred1_ci = pred1.conf_int()
```

[ ]:
```
pred2 = results.get_forecast('1962-12-01')
pred2_ci = pred2.conf_int()
print(pred2.predicted_mean['1960-01-01':'1960-12-01'])
```

```
1960-01-01    419.495085
1960-02-01    397.834142
1960-03-01    460.859052
1960-04-01    449.451900
1960-05-01    474.555739
1960-06-01    537.848954
1960-07-01    614.884907
1960-08-01    628.209240
1960-09-01    519.336551
1960-10-01    462.254691
1960-11-01    412.164222
1960-12-01    452.664872
```

```
Freq: MS, dtype: float64
```

[ ]:
```
ax = data.plot(figsize=(20, 16))
pred0.predicted_mean.plot(ax=ax, label='1-step-ahead Forecast (get_predictions,␣
 ↪dynamic=False)')
pred1.predicted_mean.plot(ax=ax, label='Dynamic Forecast (get_predictions,␣
 ↪dynamic=True)')
pred2.predicted_mean.plot(ax=ax, label='Dynamic Forecast (get_forecast)')
ax.fill_between(pred2_ci.index, pred2_ci.iloc[:, 0], pred2_ci.iloc[:, 1],␣
 ↪color='k', alpha=.1)
plt.ylabel('Monthly airline passengers (x1000)')
plt.xlabel('Date')
plt.legend()
plt.show()
```



[ ]:
```
prediction = pred2.predicted_mean['1960-01-01':'1960-12-01'].values

truth = list(itertools.chain.from_iterable(test_data.values))

MAPE = np.mean(np.abs((truth - prediction) / truth)) * 100
```

```
print('The Mean Absolute Percentage Error for the forecast of year 1960 is {:.
  ↪2f}%'.format(MAPE))
```

```
The Mean Absolute Percentage Error for the forecast of year 1960 is 2.81%
```