03-Descriptive-Statistics-and-Tests

October 30, 2021

1 Descriptive Statistics and Tests

In upcoming sections we'll talk about different forecasting models like ARMA, ARIMA, Seasonal ARIMA and others. Each model addresses a different type of time series. For this reason, in order to select an appropriate model we need to know something about the data.

In this section we'll learn how to determine if a time series is stationary, if it's independent, and if two series demonstrate correlation and/or causality.

Related Functions:

stattools.ccovf(x, y[, unbiased, demean]) crosscovariance for 1D stattools.ccf(x, y[, unbiased]) cross-correlation function for 1d stattools.periodogram(X) Returns the periodogram for the natural frequency of X

stattools.adfuller(x[, maxlag, regression, …]) Augmented Dickey-Fuller unit root test stattools.kpss(x[, regression, lags, store]) Kwiatkowski-Phillips-Schmidt-Shin test for stationarity. stattools.coint(y0, y1[, trend, method, …]) Test for no-cointegration of a univariate equation stattools.bds(x[, max_dim, epsilon, distance]) Calculate the BDS test statistic for independence of a time series stattools.q_stat(x, nobs[, type]) Returns Ljung-Box Q Statistic stattools.grangercausalitytests(x, maxlag[, …]) Four tests for granger non-causality of 2 timeseries stattools.levinson_durbin(s[, nlags, isacov]) Levinson-Durbin recursion for autoregressive processes

stattools.eval_measures.mse(x1, x2, axis=0) mean squared error stattools.eval_measures.rmse(x1, x2, axis=0) root mean squared error stattools.eval_measures.meanabs(x1, x2, axis=0) mean absolute error

For Further Reading:

Wikipedia: Augmented Dickey–Fuller test Wikipedia: Kwiatkowski-Phillips-Schmidt-Shin test Wikipedia: Granger causality test Forecasting: Principles and Practice: Evaluating forecast accuracy

2 Tests for Stationarity

A time series is stationary if the mean and variance are fixed between any two equidistant points. That is, no matter where you take your observations, the results should be the same. A times series that shows seasonality is not stationary.

A test for stationarity usually involves a unit root hypothesis test, where the null hypothesis H_0 is that the series is nonstationary, and contains a unit root. The alternate hypothesis H_1

supports stationarity. The augmented Dickey-Fuller and Kwiatkowski-Phillips-Schmidt-Shin tests are stationarity tests.

2.1 Augmented Dickey-Fuller Test

To determine whether a series is stationary we can use the augmented Dickey-Fuller Test. In this test the null hypothesis states that $\phi = 1$ (this is also called a unit test). The test returns several statistics we'll see in a moment. Our focus is on the p-value. A small p-value ($p < 0.05$) indicates strong evidence against the null hypothesis.

To demonstrate, we'll use a dataset we know is not stationary, the airline_passenger dataset. First, let's plot the data along with a 12-month rolling mean and standard deviation:

[2]:
```
import pandas as pd
import numpy as np
%matplotlib inline
from statsmodels.tsa.stattools import adfuller,grangercausalitytests
from statsmodels.tools.eval_measures import mse, rmse, meanabs
from statsmodels.graphics.tsaplots import month_plot,quarter_plot
import warnings
warnings.filterwarnings('ignore')
```

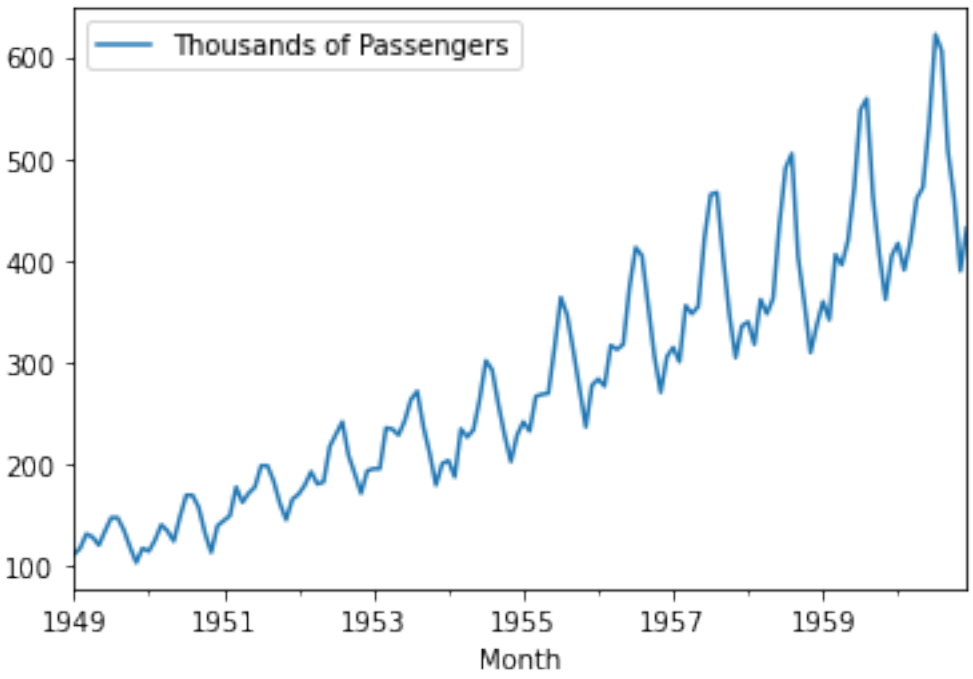
[26]:
```
df1= pd.read_csv("../data/airline_passengers.
 ↪csv",index_col="Month",parse_dates=True)
df1.index.freq="MS"
df1.head(5)
```

[26]:

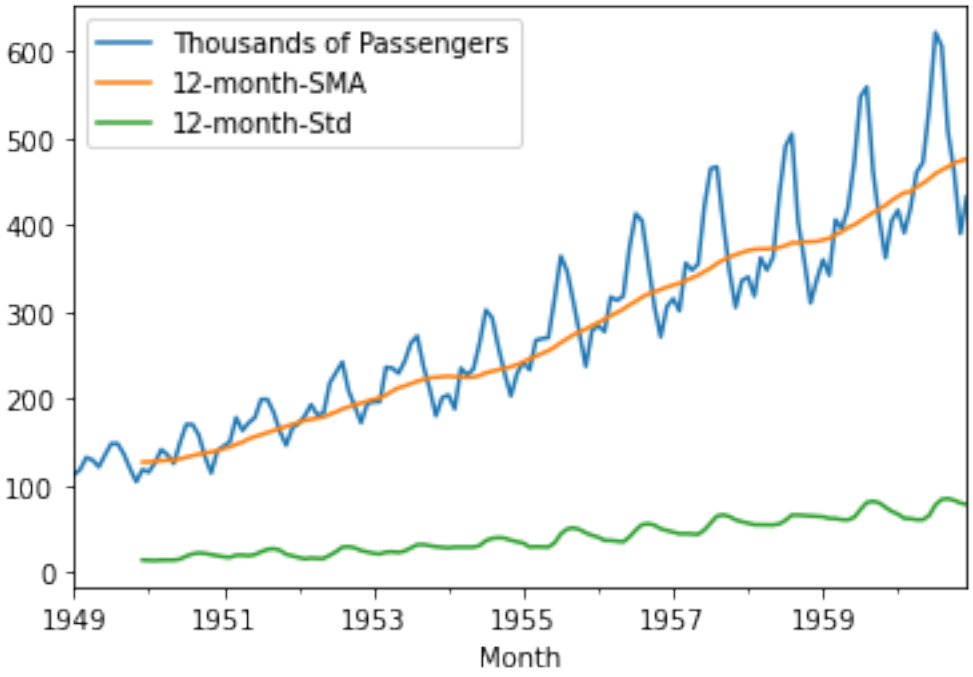
Month	Thousands of Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

[6]:
```
df1.plot()
```

[6]:
```
<AxesSubplot:xlabel='Month'>
```



[7]:
```
'''
What we're going to do is we're going to actually try to use the augmented␣
 ↪Dickie Fuller test in order
to test the null hypothesis that states that this is stationary or not.␣
 ↪basically can be quantitatively
configured, that this is going to be a non stationary data set.


'''
df1['12-month-SMA'] = df1['Thousands of Passengers'].rolling(window=12).mean()
df1['12-month-Std'] = df1['Thousands of Passengers'].rolling(window=12).std()

df1[['Thousands of Passengers','12-month-SMA','12-month-Std']].plot();
```

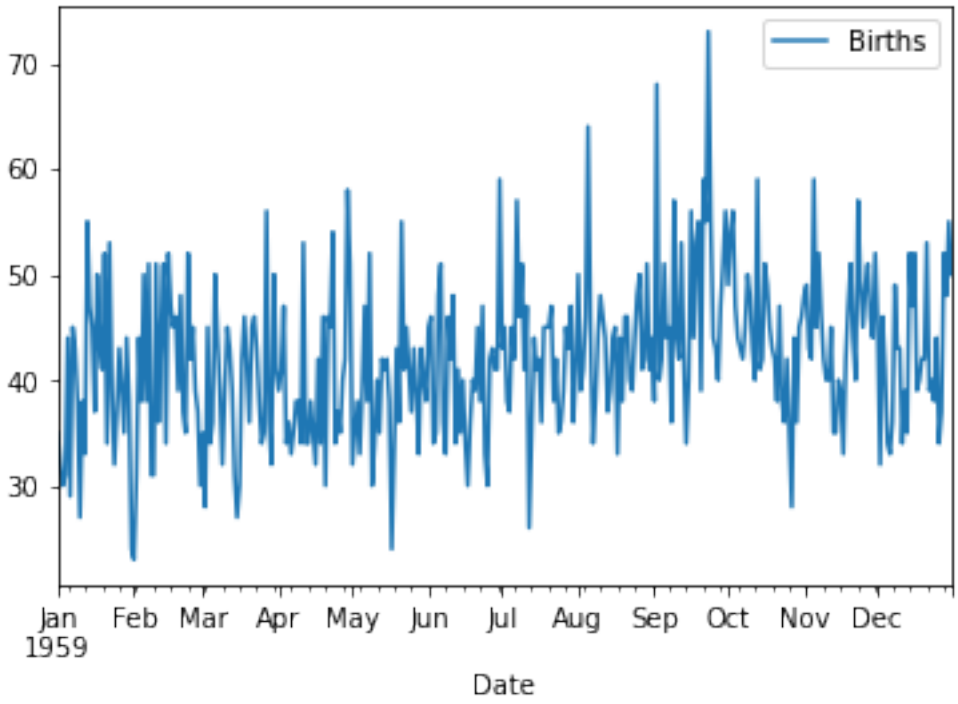


[28]:
```
df2= pd.read_csv("../data/DailyTotalFemaleBirths.
 ↪csv",index_col="Date",parse_dates=True)
df2.index.freq="D"
df2.head(5)
```

[28]:
```
            Births
Date
1959-01-01      35
1959-01-02      32
1959-01-03      30
1959-01-04      31
1959-01-05      44
```

[16]:
```
df2.plot()
```

[16]:
```
<AxesSubplot:xlabel='Date'>
```



[16]:
```
adfuller(df1["Thousands of Passengers"])
```

[16]:
```
(0.8153688792060512,
 0.991880243437641,
 13,
 130,
 {'1%': -3.4816817173418295,
  '5%': -2.8840418343195267,
  '10%': -2.578770059171598},
 996.692930839019)
```

[17]:
```
help(adfuller)
```

```
Help on function adfuller in module statsmodels.tsa.stattools:

adfuller(x, maxlag=None, regression='c', autolag='AIC', store=False,
regresults=False)
    Augmented Dickey-Fuller unit root test.

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

    Parameters
    ----------
```

```
x : array_like, 1d
    The data series to test.
maxlag : int
    Maximum lag which is included in test, default 12*(nobs/100)^{1/4}.
regression : {"c","ct","ctt","nc"}
    Constant and trend order to include in regression.

    * "c" : constant only (default).
    * "ct" : constant and trend.
    * "ctt" : constant, and linear and quadratic trend.
    * "nc" : no constant, no trend.

autolag : {"AIC", "BIC", "t-stat", None}
    Method to use when automatically determining the lag length among the
    values 0, 1, …, maxlag.

    * If "AIC" (default) or "BIC", then the number of lags is chosen
      to minimize the corresponding information criterion.
    * "t-stat" based choice of maxlag.  Starts with maxlag and drops a
      lag until the t-statistic on the last lag length is significant
      using a 5%-sized test.
    * If None, then the number of included lags is set to maxlag.
store : bool
    If True, then a result instance is returned additionally to
    the adf statistic. Default is False.
regresults : bool, optional
    If True, the full regression results are returned. Default is False.

Returns
-------
adf : float
    The test statistic.
pvalue : float
    MacKinnon"s approximate p-value based on MacKinnon (1994, 2010).
usedlag : int
    The number of lags used.
nobs : int
    The number of observations used for the ADF regression and calculation
    of the critical values.
critical values : dict
    Critical values for the test statistic at the 1 %, 5 %, and 10 %
    levels. Based on MacKinnon (2010).
icbest : float
    The maximized information criterion if autolag is not None.
resstore : ResultStore, optional
    A dummy class with results attached as attributes.

Notes
```

```
    -----
    The null hypothesis of the Augmented Dickey-Fuller is that there is a unit
    root, with the alternative that there is no unit root. If the pvalue is
    above a critical size, then we cannot reject that there is a unit root.

    The p-values are obtained through regression surface approximation from
    MacKinnon 1994, but using the updated 2010 tables. If the p-value is close
    to significant, then the critical values should be used to judge whether
    to reject the null.

    The autolag option and maxlag for it are described in Greene.

    References
    ----------
    .. [1] W. Green.  "Econometric Analysis," 5th ed., Pearson, 2003.

    .. [2] Hamilton, J.D.  "Time Series Analysis".  Princeton, 1994.

    .. [3] MacKinnon, J.G. 1994.  "Approximate asymptotic distribution functions
for
        unit-root and cointegration tests.  `Journal of Business and Economic
        Statistics` 12, 167-76.

    .. [4] MacKinnon, J.G. 2010. "Critical Values for Cointegration Tests."
Queen"s
        University, Dept of Economics, Working Papers.  Available at
        http://ideas.repec.org/p/qed/wpaper/1227.html

    Examples
    --------
    See example notebook
```

[9]:
```
'''
Basically, I ran that augmented Dicky Fuller test on the thousands of passengers

And then I formatted this DF test as well as this Dfout.

Probably the most important thing to realize here is the actual P value, zero␣
 ↪point nine nine.

And basically we are going to have to determine based off the P value, if we␣
 ↪should reject the null
hypothesis or fail to reject the null hypothesis.

'''
print('Augmented Dickey-Fuller Test on Airline Data')
```

```
dftest=adfuller(df1["Thousands of Passengers"])
dfout = pd.Series(dftest[0:4],index=['ADF test statistic','p-value','# lags␣
↪used','# observations'])

for key,val in dftest[4].items():
    dfout[f'critical value ({key})']=val
print(dfout)
```

```
Augmented Dickey-Fuller Test on Airline Data
ADF test statistic        0.815369
p-value                   0.991880
# lags used              13.000000
# observations          130.000000
critical value (1%)      -3.481682
critical value (5%)      -2.884042
critical value (10%)     -2.578770
dtype: float64
```

Here we have a very high p-value at 0.99, which provides weak evidence against the null hypothesis, and so we fail to reject the null hypothesis, and decide that our dataset is not stationary. Note: in statistics we don't "accept" a null hypothesis - nothing is ever truly proven - we just fail to reject it. Now let's apply the ADF test to stationary data with the Daily Total Female Births dataset.

[20]:
```
def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles␣
↪differenced data

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype:␣
↪float64"

    '''
    And so basically the way augmented Dickie Fuller test works, if this result␣
↪is less than or equal to

    0.05, then we have strong evidence against the null hypothesis.
```

```
    And so we reject the null hypothesis and we say the data has no unit root␣
 ↪and is stationary.

    Otherwise there's weak evidence against the null hypothesis.

    So we fail to reject the null hypothesis and we're going to assume that the␣
 ↪data has a unit root and

    is non stationary.
    '''
    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```

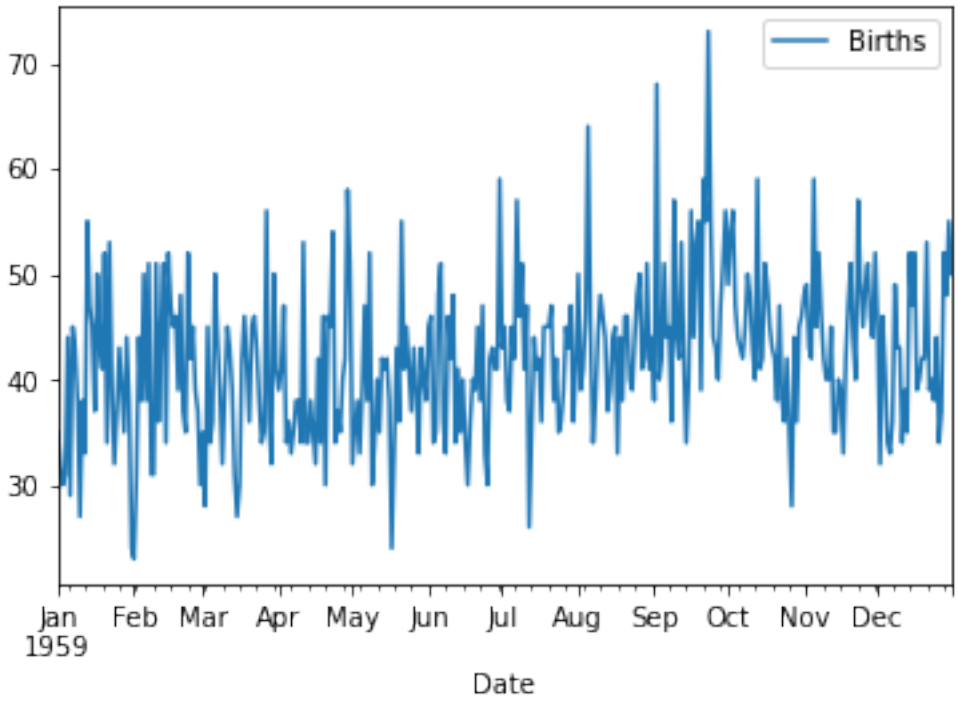
[11]:
```
'''
However, what's nice about this is it gives back this little report here that's␣
 ↪easy to read.

And basically the most important line is here at the bottom where it says data␣
 ↪has a unit root and is
non stationary.

So what does that actually mean?
It means that we're going to assume that thousands of passengers when we plot␣
 ↪this out is non stationary.

Well, clearly, it definitely is, because there's obvious seasonality and an␣
 ↪obvious trend.
So even just with the naked eye, we could see that this is definitely a non␣
 ↪stationary data set.

'''
adf_test(df1["Thousands of Passengers"])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic        0.815369
p-value                   0.991880
# lags used              13.000000
# observations          130.000000
critical value (1%)      -3.481682
critical value (5%)      -2.884042
critical value (10%)     -2.578770
```

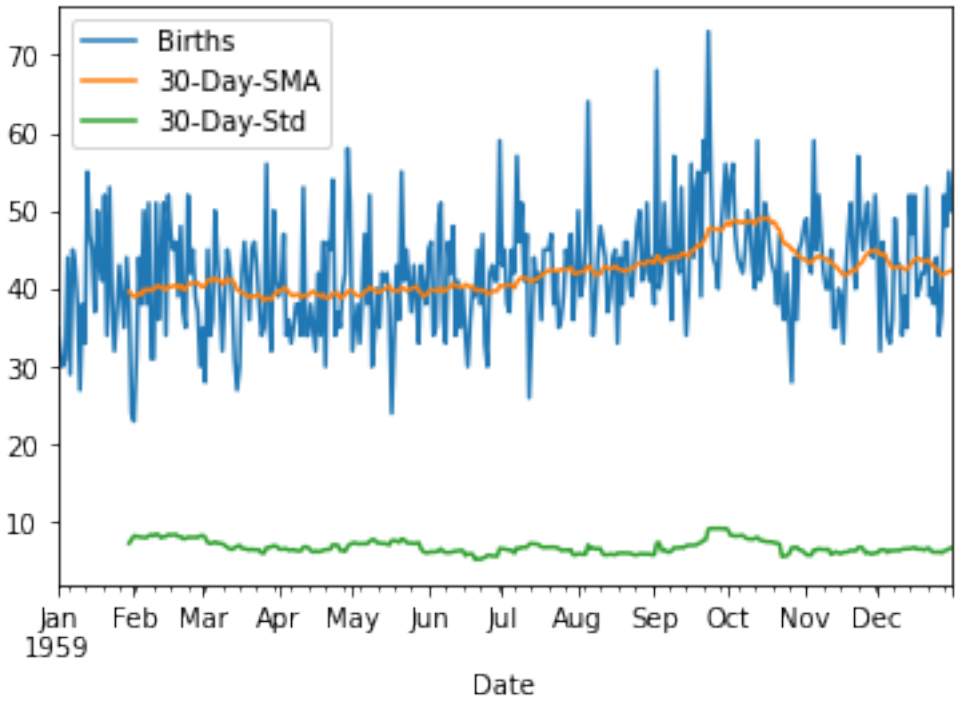
```
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

[24]:
```
df2.plot()
```

[24]:
```
<AxesSubplot:xlabel='Date'>
```



[17]:
```
df2['30-Day-SMA'] = df2['Births'].rolling(window=30).mean()
df2['30-Day-Std'] = df2['Births'].rolling(window=30).std()

df2[['Births','30-Day-SMA','30-Day-Std']].plot();
```



[18]:
```
print('Augmented Dickey-Fuller Test on Daily Female Births')
dftest = adfuller(df2['Births'],autolag='AIC')
dfout = pd.Series(dftest[0:4],index=['ADF test statistic','p-value','# lags␣
 ↪used','# observations'])

for key,val in dftest[4].items():
    dfout[f'critical value ({key})']=val
print(dfout)
```

```
Augmented Dickey-Fuller Test on Daily Female Births
ADF test statistic       -4.808291
p-value                   0.000052
# lags used                6.000000
# observations           358.000000
critical value (1%)      -3.448749
critical value (5%)      -2.869647
critical value (10%)     -2.571089
dtype: float64
```

[19]:
```
adf_test(df2["Births"])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic       -4.808291
p-value                   0.000052
```

```
# lags used               6.000000
# observations          358.000000
critical value (1%)      -3.448749
critical value (5%)      -2.869647
critical value (10%)     -2.571089
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

3 Granger Causality Tests

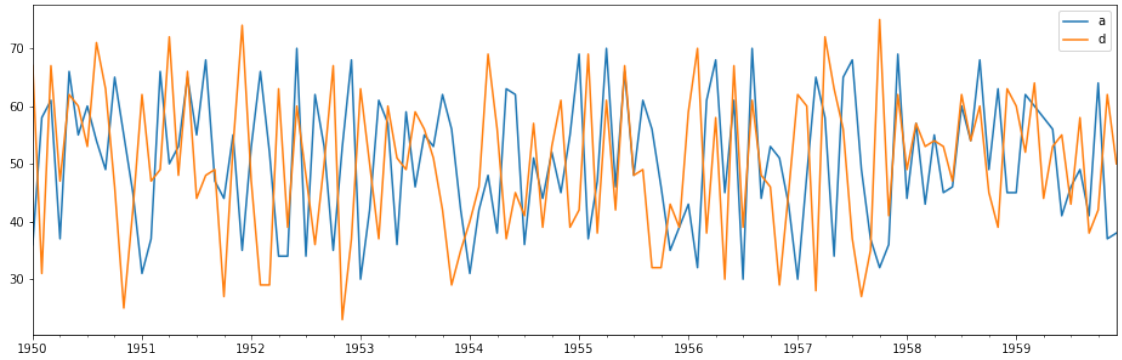
The Granger causality test is a a hypothesis test to determine if one time series is useful in forecasting another. While it is fairly easy to measure correlations between series - when one goes up the other goes up, and vice versa - it's another thing to observe changes in one series correlated to changes in another after a consistent amount of time. This may indicate the presence of causality, that changes in the first series influenced the behavior of the second. However, it may also be that both series are affected by some third factor, just at different rates. Still, it can be useful if changes in one series can predict upcoming changes in another, whether there is causality or not. In this case we say that one series "Granger-causes" another.

In the case of two series, y and x, the null hypothesis is that lagged values of x do not explain variations in y. In other words, it assumes that x_t doesn't Granger-cause y_t.

The stattools grangercausalitytests function offers four tests for granger non-causality of 2 timeseries

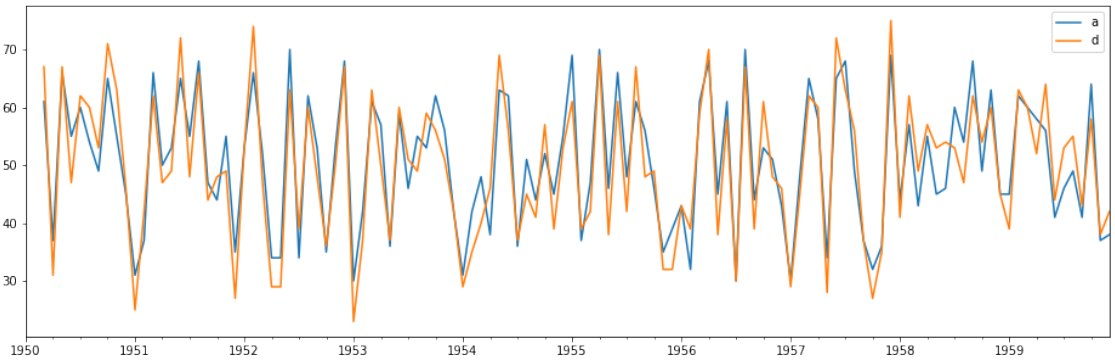
For this example we'll use the samples.csv file, where columns 'a' and 'd' are stationary datasets.

[22]:
```
df3 = pd.read_csv('../Data/samples.csv',index_col=0,parse_dates=True)
df3.index.freq = 'MS'
df3[['a','d']].plot(figsize=(16,5));
```



It's hard to tell from this overlay but df['d'] almost perfectly predicts the behavior of df['a']. To see this more clearly (spoiler alert!), we will shift df['d'] two periods forward.

[23]:
```
'''
So just using this or just looking at this with the naked eye, it's really hard␣
 ↪to tell if there's

any sort of causality between A and however, I'm going to show you that if we␣
 ↪were to actually shift

one of these by just a little bit, there is actually a really strong indication␣
 ↪that there is causality,

even if you just look at this from the naked eye.


we almost get kind of perfect alignment here, there is a little bit of noise,␣
 ↪but clearly

there are some sort of relationship here, as you can see, that they are lining

up on top of each other pretty well.

And essentially what that is saying is two days after there's some change and␣
 ↪it looks like a is reflecting

that.
'''
df3['a'].iloc[2:].plot(figsize=(16,5),legend=True);
df3['d'].shift(2).plot(legend=True);
```



[24]:
```
'''
However, it's extremely unlikely that you yourself are going to be able to␣
 ↪guess what the shift is.

And then even then, it may not be as clear as this made up data.
```

```
So luckily, that's where the Granger causality test comes into play.

We can run the tests and then it will report back to us the hypothesis of␣
↪whether or not there's causality

in between these two time series.

So it's up to you to go ahead and decide how many legs should be checked for.


The tradeoff between choosing larger Max Lack's versus smaller Max Max is the␣
↪more max lag's you check,

the more opportunities you have for causality to pop up.

However, the more max lags you check, that means you're going to spend more␣
↪time computing these

causality tests.

And here we can see the number of lags on one, two and three.(number of lags␣
↪(no zero) 1)

And then we get a bunch of information back.

And essentially what we're looking for is extremely low P values, which we see␣
↪here at
lag-2(number of lags (no zero) 2).

So at lag-2, if we were to kind of check this out, we notice here that we have␣
↪extremely low P

values.

'''
grangercausalitytests(df3[["a","d"]],maxlag=3);
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.7051  , p=0.1942  , df_denom=116, df_num=1
ssr based chi2 test:   chi2=1.7492  , p=0.1860  , df=1
likelihood ratio test: chi2=1.7365  , p=0.1876  , df=1
parameter F test:         F=1.7051  , p=0.1942  , df_denom=116, df_num=1

Granger Causality
```

```
number of lags (no zero) 2
ssr based F test:         F=286.0339, p=0.0000  , df_denom=113, df_num=2
ssr based chi2 test:   chi2=597.3806, p=0.0000  , df=2
likelihood ratio test: chi2=212.6514, p=0.0000  , df=2
parameter F test:         F=286.0339, p=0.0000  , df_denom=113, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=188.7446, p=0.0000  , df_denom=110, df_num=3
ssr based chi2 test:   chi2=602.2669, p=0.0000  , df=3
likelihood ratio test: chi2=212.4789, p=0.0000  , df=3
parameter F test:         F=188.7446, p=0.0000  , df_denom=110, df_num=3
```

Essentially we're looking for extremely low p-values, which we see at lag 2. By comparison, let's compare two datasets that are not at all similar, 'b' and 'd'.

[33]:
```
'''
Now, what I want to do is compare this to a data set where we know there was no␣
 ↪sort of causality.

notice none of these P values never end up being less

than 0.05, which is basically our cutoff for any sort of hypothesis testing to␣
 ↪see

if there's causality there.


So the way to interpret this is if you take your data frame, passing the two␣
 ↪columns, choose some

sort of max lag value to check,since its going to check all those lags␣
 ↪eventually,

If one of these ends up popping with very low P values, something less than 0.
 ↪05,

you have stronger evidence and an indication that there is some sort of␣
 ↪causality between these two

data sets.

But Granger, causality is a great quantitative way of saying, hey, go ahead and␣
 ↪look here at this

particular max lag values and you'll see where there is no causality.
```

```
You basically get P values all greater than 0.05, indicating nothing to really␣
↪check

out here.

'''
grangercausalitytests(df3[["b","d"]],maxlag=3);
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.5225  , p=0.2197  , df_denom=116, df_num=1
ssr based chi2 test:   chi2=1.5619  , p=0.2114  , df=1
likelihood ratio test: chi2=1.5517  , p=0.2129  , df=1
parameter F test:         F=1.5225  , p=0.2197  , df_denom=116, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.4350  , p=0.6483  , df_denom=113, df_num=2
ssr based chi2 test:   chi2=0.9086  , p=0.6349  , df=2
likelihood ratio test: chi2=0.9051  , p=0.6360  , df=2
parameter F test:         F=0.4350  , p=0.6483  , df_denom=113, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.5333  , p=0.6604  , df_denom=110, df_num=3
ssr based chi2 test:   chi2=1.7018  , p=0.6365  , df=3
likelihood ratio test: chi2=1.6895  , p=0.6393  , df=3
parameter F test:         F=0.5333  , p=0.6604  , df_denom=110, df_num=3
```

4 Evaluating forecast accuracy

Two calculations related to linear regression are mean squared error (MSE) and root mean squared error (RMSE)

The formula for the mean squared error is $MSE = \frac{1}{L}\sum_{l=1}^{L}(y_{T+l} - \hat{y}_{T+l})^2$ where T is the last observation period and l is the lag point up to L number of test observations.

The formula for the root mean squared error is $RMSE = \sqrt{MSE} = \sqrt{\frac{1}{L}\sum_{l=1}^{L}(y_{T+l} - \hat{y}_{T+l})^2}$

The advantage of the RMSE is that it is expressed in the same units as the data.

A method similar to the RMSE is the mean absolute error (MAE) which is the mean of the magnitudes of the error, given as

$$MAE = \frac{1}{L}\sum_{l=1}^{L} \mid y_{T+l} - \hat{y}_{T+l} \mid$$

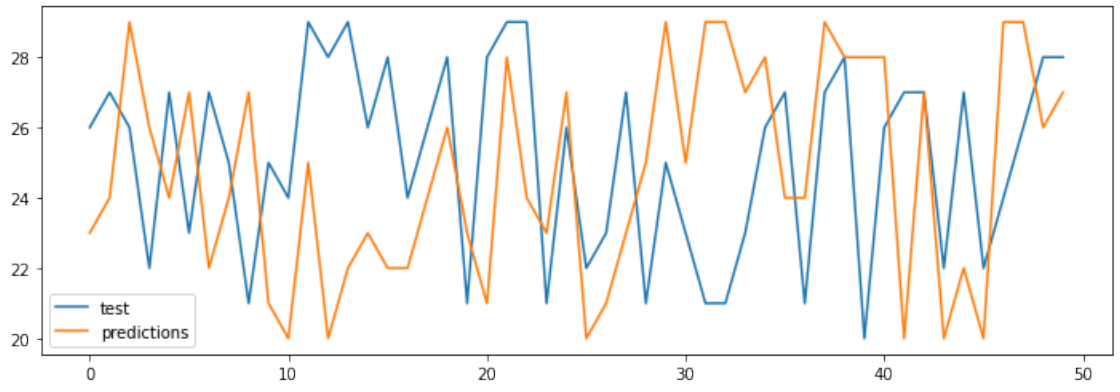
A forecast method that minimizes the MAE will lead to forecasts of the median, while minimizing the RMSE will lead to forecasts of the mean.

[29]:
```
'''
And then we'll say from range 20 to 30.

Go ahead and give me 50 by 2.

So that's the shape and then I'm going to pass that on to my data frame and say␣
 ↪columns is equal to

test.

So all I'm doing is I'm choosing random integers between 20 and 30, given the␣
 ↪shape of 50 rows by 2

columns, I'll name the columns test and predictions.

'''
np.random.seed(42)
df = pd.DataFrame(np.random.
 ↪randint(20,30,(50,2)),columns=['test','predictions'])
df.head(5)
```

[29]:

	test	predictions
0	26	23
1	27	24
2	26	29
3	22	26
4	27	24

[30]:
```
df.plot(figsize=(12,4));
```



[32]:
```
from statsmodels.tools.eval_measures import mse, rmse, meanabs

MSE = mse(df['test'],df['predictions'])
RMSE = rmse(df['test'],df['predictions'])
MAE = meanabs(df['test'],df['predictions'])

print(f'Model  MSE: {MSE:.3f}')
print(f'Model RMSE: {RMSE:.3f}')
print(f'Model  MAE: {MAE:.3f}')
```

```
Model  MSE: 17.020
Model RMSE: 4.126
Model  MAE: 3.540
```

4.0.1 AIC / BIC

More sophisticated tests include the Akaike information criterion (AIC) and the Bayesian information criterion (BIC).

The AIC evaluates a collection of models and estimates the quality of each model relative to the others. Penalties are provided for the number of parameters used in an effort to thwart overfitting. The lower the AIC and BIC, the better the model should be at forecasting.

These functions are available as

from from statsmodels.tools.eval_measures import aic, bic

but we seldom compute them alone as they are built into many of the statsmodels tools we use.

4.1 Exposing Seasonality with Month and Quarter Plots

Statsmodels has two plotting functions that group data by month and by quarter. Note that if the data appears as months, you should employ resampling with an aggregate function before running a quarter plot. These plots return a matplotlib.Figure object.

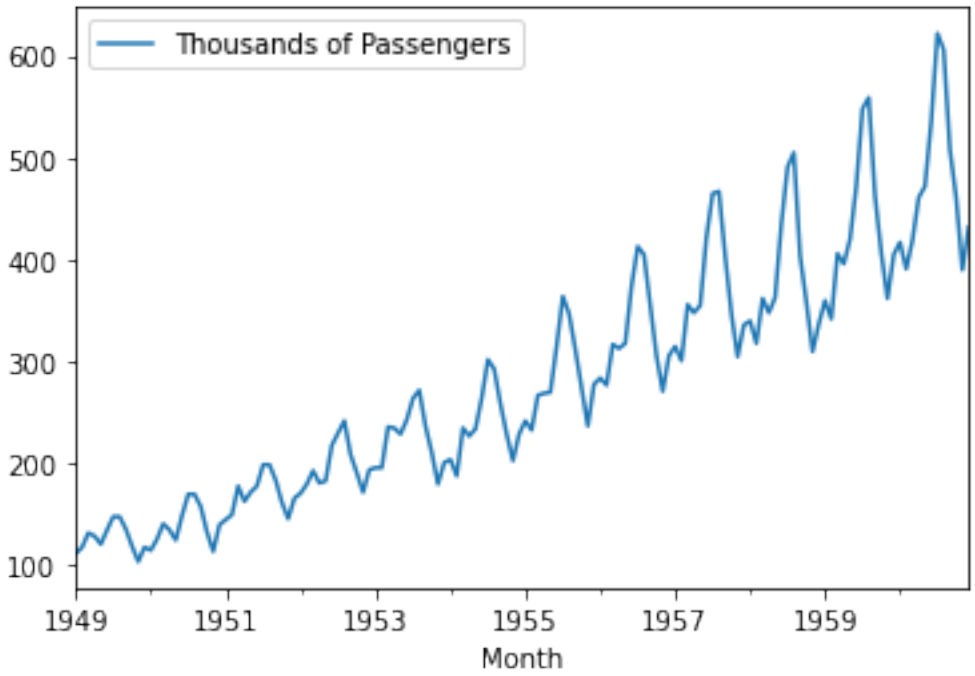
Related Plot Methods:

tsaplots.month_plot(x) Seasonal plot of monthly data tsaplots.quarter_plot(x) Seasonal plot of quarterly data

[33]:
```
'''
so if you just plot this out without looking at the data,

it may be a little hard to tell when the seasonality is actually happening.

We already know that it happens during the summers.

But if you were just to be given this raw data set, then you would have to go␣
 ↪in and kind of stretch
```

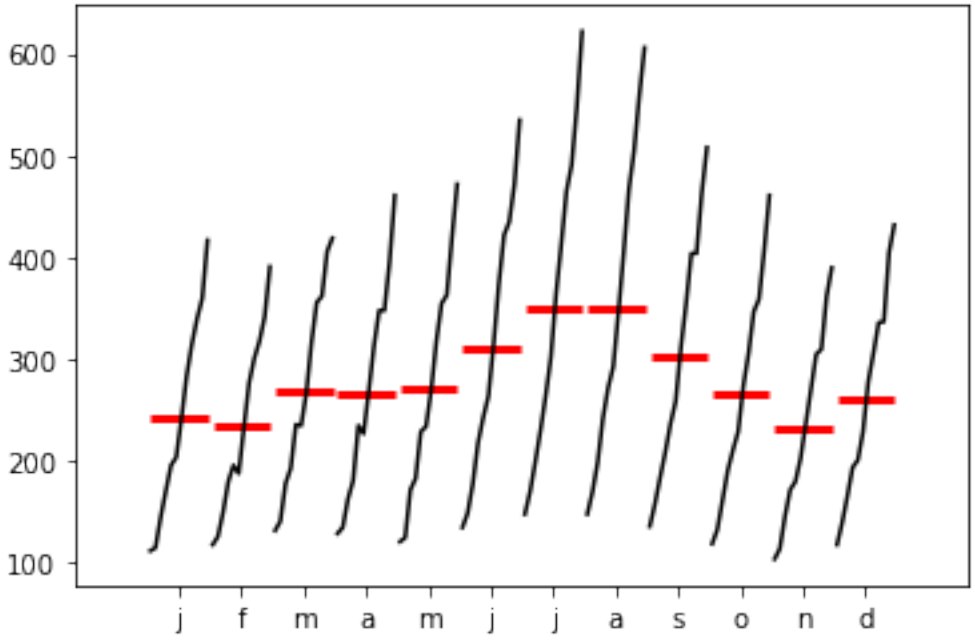
```
this out ,Look at the months, maybe edit your X-axis axis labels so you can␣
 ↪actually view the months, etc. in

order to determine the seasonality effects.
'''
df1.plot()
```

[33]: <AxesSubplot:xlabel='Month'>



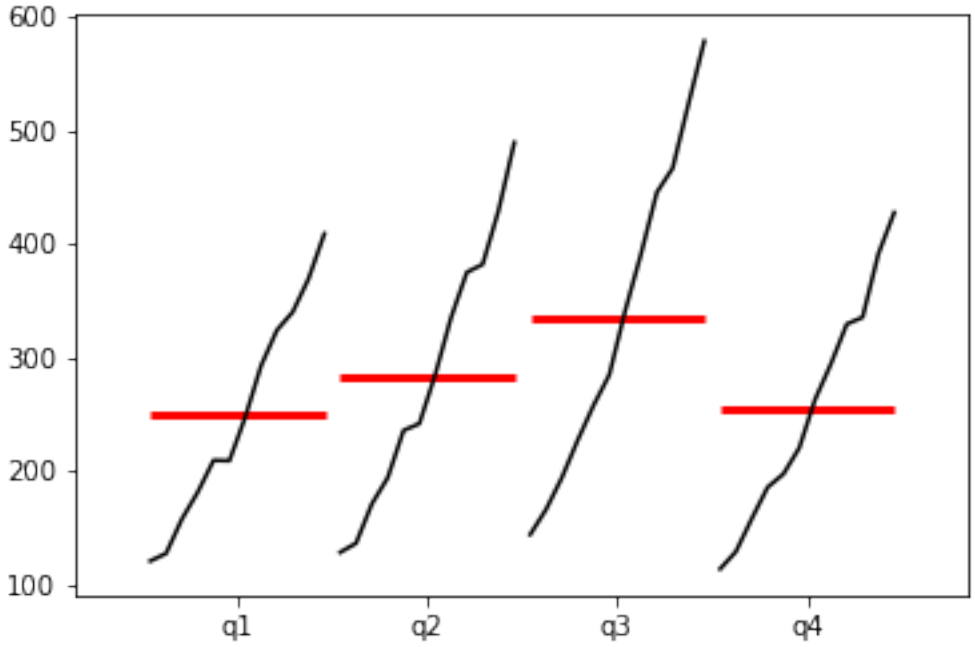
[34]:
```
'''
what this does is if you have timestamped index, just like we have here at the␣
 ↪index, that's models

can actually then separate this out, all these values per month and per quarter.

what this does is it essentially takes in is essentially a group buy and then
plots out the range of values as well as your average.

So you see January, February, March, April and so on.

And here we can then clearly see that there's an uptick during June and July in␣
 ↪the summer months,

which intuitively makes sense.
```

```
More people are traveling during the summers.

They have summer vacation, et cetera.

'''
from statsmodels.graphics.tsaplots import month_plot,quarter_plot
# Note: add a semicolon to prevent two plots being displayed in jupyter
month_plot(df1['Thousands of Passengers']);
```



[44]:
```
'''
So maybe you're dealing with business data and you want to check this out by␣
 ↪quarter.

f we haven't sampled our data to be quartered, we simply say using pandas,␣
 ↪sampled quarterly

resample(rule='Q') and then taken the mean values.


OK, so here we can see that in general, it looks like Q3 tends to be when␣
 ↪passengers go and travel

more, which basically lines up with those summer months that we saw earlier.
```

```
'''
dfq = df1['Thousands of Passengers'].resample(rule='Q').mean()

quarter_plot(dfq);
```



[ ]: