03-Descriptive-Statistics-and-Tests

October 19, 2022

For more information, visit us at www.pieriandata.com

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

1.1 Perform standard imports and load datasets

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline

# Ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")

# Load a seasonal dataset
df1 = pd.read_csv('../Data/airline_passengers.
 ↪csv',index_col='Month',parse_dates=True)
df1.index.freq = 'MS'

# Load a nonseasonal dataset
df2 = pd.read_csv('../Data/DailyTotalFemaleBirths.
 ↪csv',index_col='Date',parse_dates=True)
df2.index.freq = 'D'
```

[2]:
```
from statsmodels.tsa.stattools import ccovf,ccf,periodogram
```

[3]:
```
from statsmodels.tsa.stattools import␣
 ↪adfuller,kpss,coint,bds,q_stat,grangercausalitytests,levinson_durbin
```

[4]:
```
from statsmodels.tools.eval_measures import mse, rmse, meanabs

# Alternative:
# from sklearn.metrics import mean_squared_error
```

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.

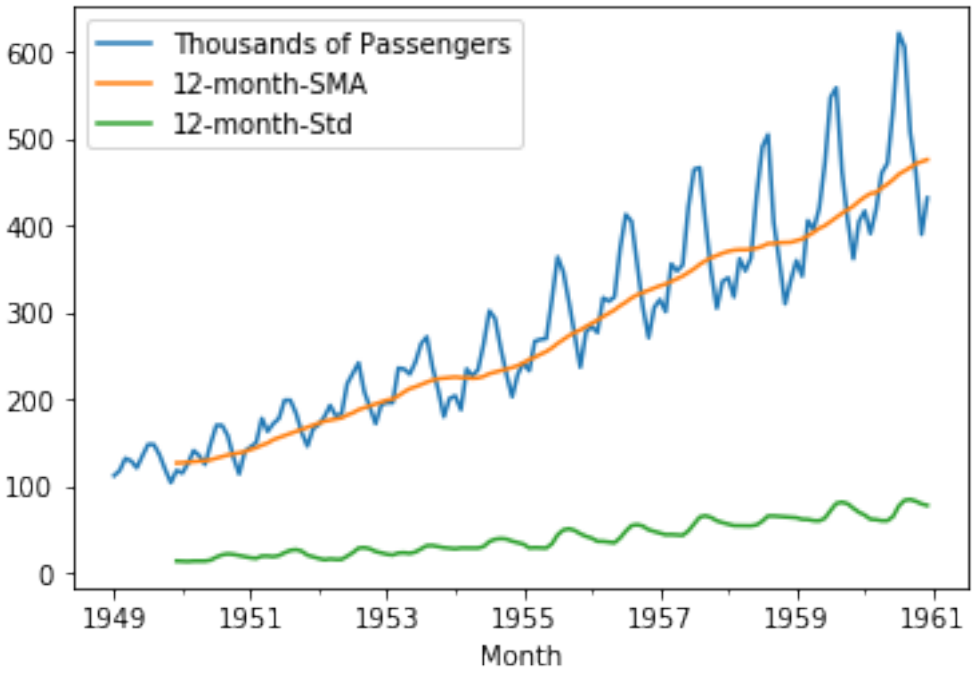
2.1 Augmented Dickey-Fuller Test

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:

[5]:
```
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();
```



Not only is this dataset seasonal with a clear upward trend, the standard deviation increases over time as well.

[6]:
```
print('Augmented Dickey-Fuller Test on Airline Data')
dftest = adfuller(df1['Thousands of Passengers'],autolag='AIC')
dftest
```

```
Augmented Dickey-Fuller Test on Airline Data
```

[6]:
```
(0.8153688792060423,
 0.9918802434376409,
 13,
 130,
 {'1%': -3.4816817173418295,
  '5%': -2.8840418343195267,
```

```
 '10%': -2.578770059171598},
996.6929308390189)
```

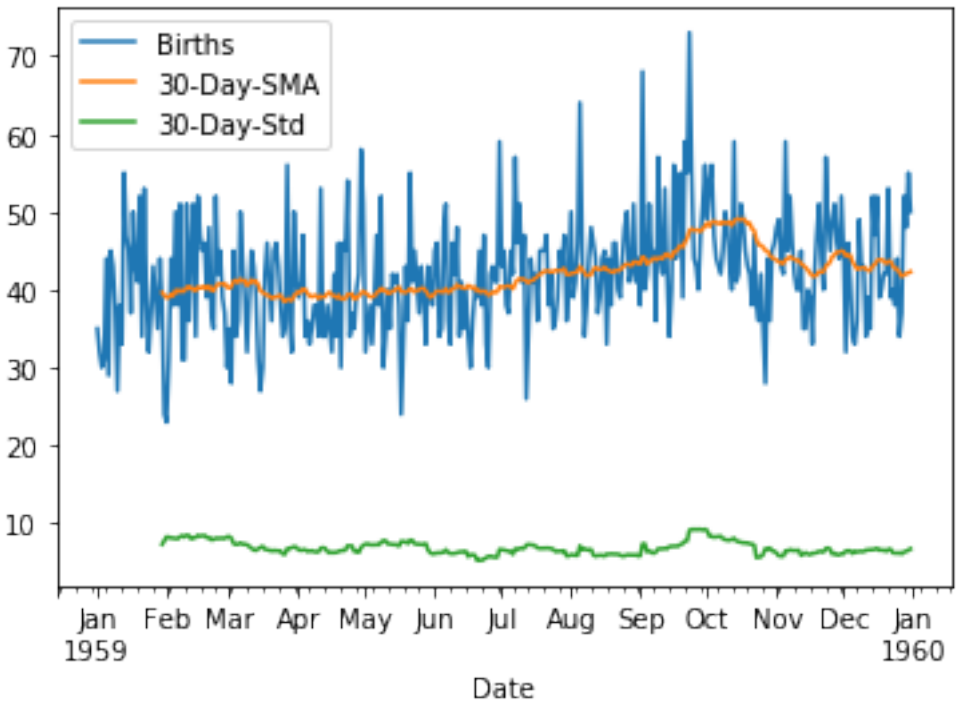
To find out what these values represent we can run help(adfuller). Then we can add our own labels:

[7]:
```
print('Augmented Dickey-Fuller Test on Airline Data')

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.

[8]:
```
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();
```



[9]:
```
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
```

In this case our p-value is very low at 0.000052, and we do reject the null hypothesis. This dataset appears to have no unit root, and is stationary.

2.1.1 Function for running the augmented Dickey-Fuller test

Since we'll use it frequently in the upcoming forecasts, let's define a function we can copy into future notebooks for running the augmented Dickey-Fuller test. Remember that we'll still have to import adfuller at the top of our notebook.

[10]:
```
from statsmodels.tsa.stattools import adfuller

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"

    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")
```

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.

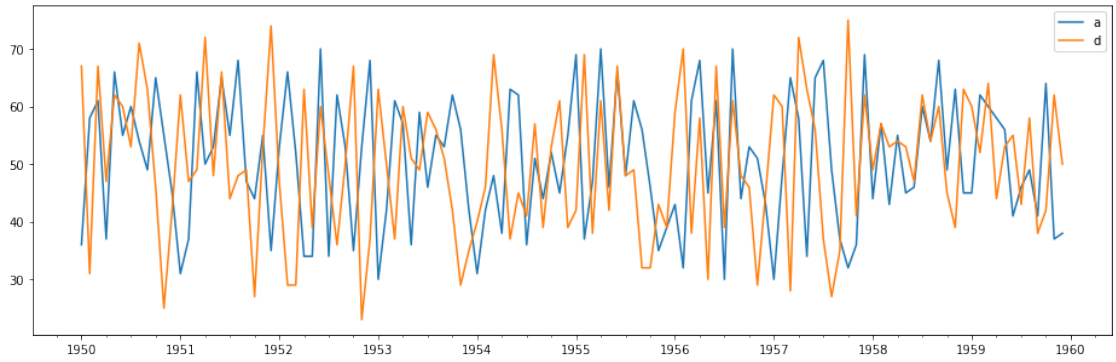
In the case of two series, y and x, the null hypothesis is that lagged values of x do not explain

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

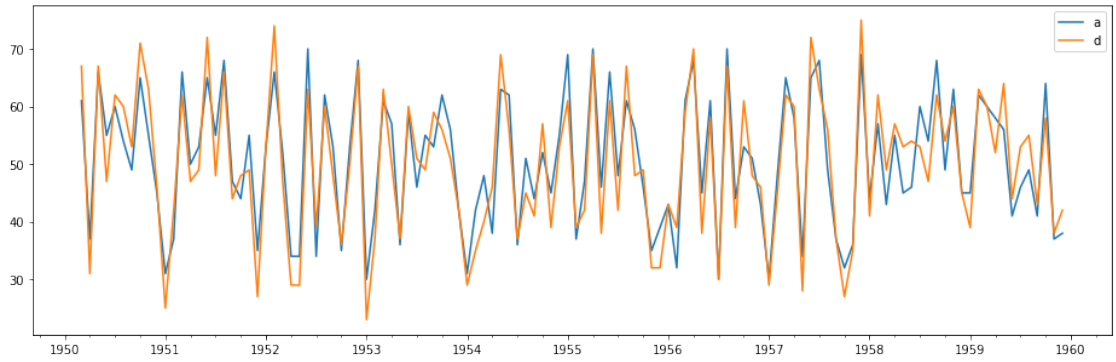
For this example we'll use the samples.csv file, where columns 'a' and 'd' are stationary datasets.

[11]:
```
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.

[12]:
```
df3['a'].iloc[2:].plot(figsize=(16,5),legend=True);
df3['d'].shift(2).plot(legend=True);
```



3.0.1 Run the test

The function takes in a 2D array [y,x] and a maximum number of lags to test on x. Here our y is column 'a' and x is column 'd'. We'll set maxlags to 3.

[13]:
```
# Add a semicolon at the end to avoid duplicate output
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'.

[14]:
```
# Add a semicolon at the end to avoid duplicate output
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
```

That's it!

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 $RMSE = \sqrt{MSE} = \sqrt{\frac{1}{L}\sum_{l=1}^{L}(y_{T+l} - \hat{y}_{T+l})^2}$

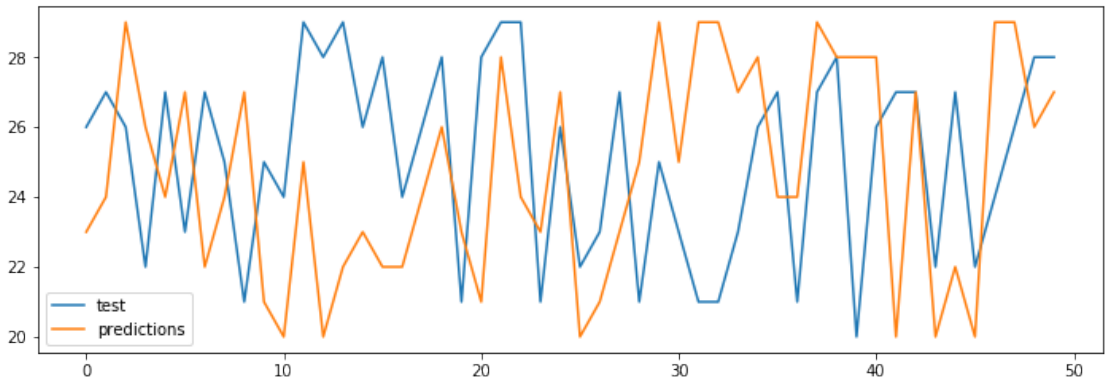
The advantage of the RMSE is that it is expressed in the same units as the data.

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.

[15]:
```
import numpy as np
import pandas as pd
%matplotlib inline
np.random.seed(42)
df = pd.DataFrame(np.random.
 ↪randint(20,30,(50,2)),columns=['test','predictions'])
df.plot(figsize=(12,4));
```



[16]:
```
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:

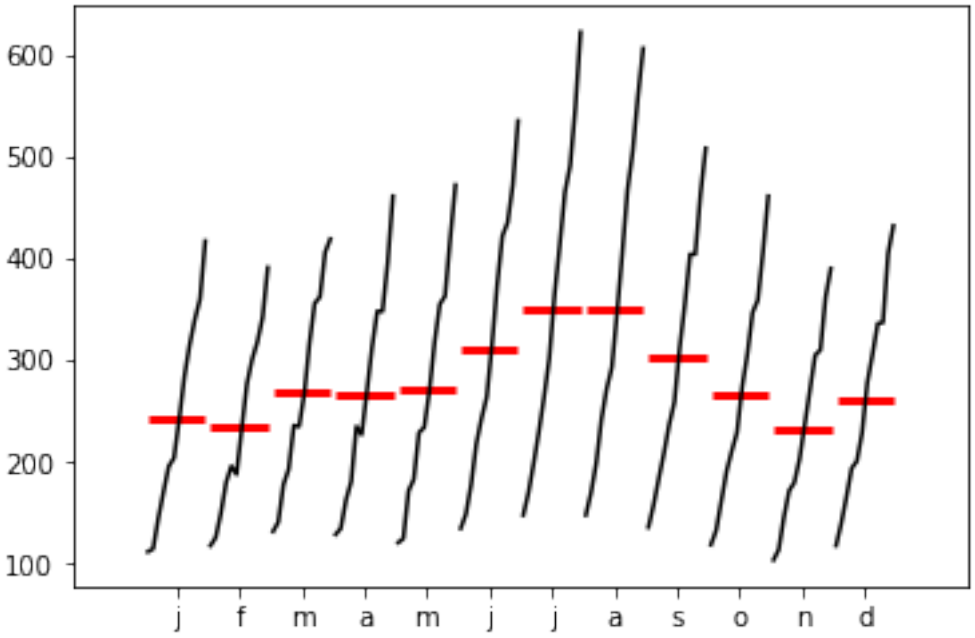
tsaplots.month_plot(x) Seasonal plot of monthly data tsaplots.quarter_plot(x) Seasonal plot of quarterly data

[17]:
```
import pandas as pd
import numpy as np
%matplotlib inline

df = pd.read_csv('../Data/airline_passengers.
 ↪csv',index_col='Month',parse_dates=True)
df.index.freq = 'MS'
df.head()
```

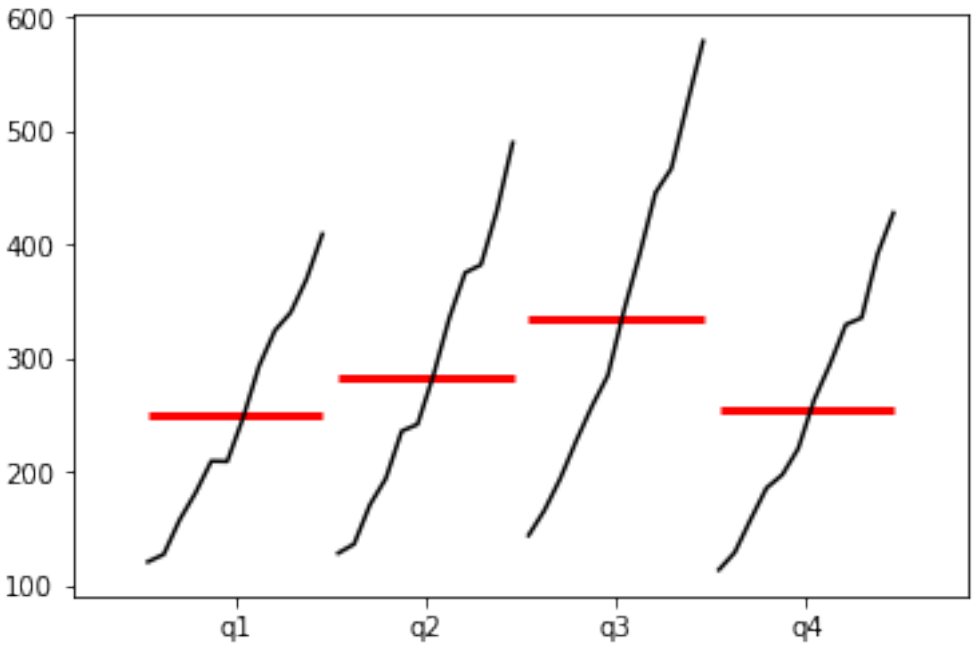
[17]:

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

[18]:
```
from statsmodels.graphics.tsaplots import month_plot,quarter_plot

# Note: add a semicolon to prevent two plots being displayed in jupyter
month_plot(df['Thousands of Passengers']);
```

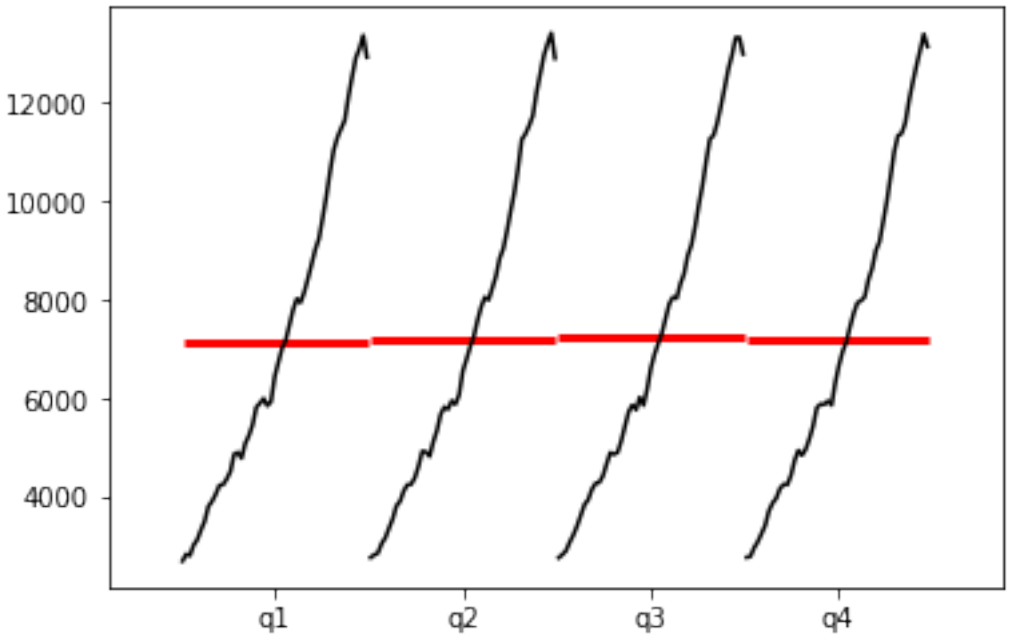


[19]:
```
dfq = df['Thousands of Passengers'].resample(rule='Q').mean()

quarter_plot(dfq);
```



Let's compare this to our nonseasonal macrodata.csv dataset:

[20]:
```
df3 = pd.read_csv('../Data/macrodata.csv',index_col=0,parse_dates=True)
quarter_plot(df3['realgdp']);
```



4.1.1 Great job!

Next up, we'll show how to determine the appropriate (p,d,q) orders for an ARIMA model.