01-ACF-and-PACF

October 19, 2022

---

1 ACF and PACF

2 Autocorrelation Function / Partial Autocorrelation Function

Before we can investigate autoregression as a modeling tool, we need to look at covariance and correlation as they relate to lagged (shifted) samples of a time series.

2.0.1 Goals

- Be able to create ACF and PACF charts
- Create these charts for multiple times series, one with seasonality and another without
- Be able to calculate Orders PQD terms for ARIMA off these charts (highlight where they cross the x axis)

Related Functions:

stattools.acovf(x[, unbiased, demean, fft, …]) Autocovariance for 1D stattools.acf(x[, unbiased, nlags, qstat, …]) Autocorrelation function for 1d arrays stattools.pacf(x[, nlags, method, alpha]) Partial autocorrelation estimated stattools.pacf_yw(x[, nlags, method]) Partial autocorrelation estimated with non-recursive yule_walker stattools.pacf_ols(x[, nlags]) Calculate partial autocorrelations

Related Plot Methods:

tsaplots.plot_acf(x) Plot the autocorrelation function tsaplots.plot_pacf(x) Plot the partial autocorrelation function

For Further Reading:

Wikipedia: Autocovariance Forecasting: Principles and Practice Autocorrelation NIST Statistics Handbook Partial Autocorrelation Plot

2.1 Perform standard imports and load datasets

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import statsmodels.api as sm

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

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

[2]:
```
# Import the models we'll be using in this section
from statsmodels.tsa.stattools import acovf,acf,pacf,pacf_yw,pacf_ols
```

2.1.1 Ignore harmless warnings

A quick note before we get started. Many of the models used in this and upcoming sections are likely to raise harmless warnings.For instance, the unbiased partial autocorrelation pacf_yw() performed below may raise a RuntimeWarning: invalid value encountered in sqrt. We don't really need to be concerned with this, and we can avoid it with the following code:

[3]:
```
import warnings
warnings.filterwarnings("ignore")
```

2.2 Autocovariance for 1D

In a deterministic process, like $y = sin(x)$, we always know the value of y for a given value of x. However, in a stochastic process there is always some randomness that prevents us from knowing the value of y. Instead, we analyze the past (or lagged) behavior of the system to derive a probabilistic estimate for $\hat{y}$.

One useful descriptor is covariance. When talking about dependent and independent x and y variables, covariance describes how the variance in x relates to the variance in y. Here the size of the covariance isn't really important, as x and y may have very different scales. However, if the covariance is positive it means that x and y are changing in the same direction, and may be related.

With a time series, x is a fixed interval. Here we want to look at the variance of y_t against lagged or shifted values of y_{t+k}

For a stationary time series, the autocovariance function for γ (gamma) is given as:

$\gamma_{XX}(t_1, t_2) = \mathrm{Cov}\left[X_{t_1}, X_{t_2}\right] = \mathrm{E}[(X_{t_1} - \mu_{t_1})(X_{t_2} - \mu_{t_2})]$

We can calculate a specific γ_k with:

$$\gamma_k = \frac{1}{n}\sum_{t=1}^{n-k}(y_t - \bar{y})(y_{t+k} - \bar{y})$$

A NOTE ON FORMULA CONVENTIONS: Different texts employ different symbol conventions. For example, in the above autocovariance formula we use k to represent the amount of lag or shift. Some texts use h instead.

2.2.1 Autocovariance Example:

Say we have a time series with five observations: $\{13, 5, 11, 12, 9\}$. We can quickly see that $n = 5$, the mean $\bar{y} = 10$, and we'll see that the variance $\sigma^2 = 8$. The following calculations give us our covariance values: $\gamma_0 = \frac{(13-10)(13-10)+(5-10)(5-10)+(11-10)(11-10)+(12-10)(12-10)+(9-10)(9-10)}{5} = \frac{40}{5} = 8.0$
$\gamma_1 = \frac{(13-10)(5-10)+(5-10)(11-10)+(11-10)(12-10)+(12-10)(9-10)}{5} = \frac{-20}{5} = -4.0$
$\gamma_2 = \frac{(13-10)(11-10)+(5-10)(12-10)+(11-10)(9-10)}{5} = \frac{-8}{5} = -1.6$
$\gamma_3 = \frac{(13-10)(12-10)+(5-10)(9-10)}{5} = \frac{11}{5} = 2.2$
$\gamma_4 = \frac{(13-10)(9-10)}{5} = \frac{-3}{5} = -0.6$ Note that γ_0 is just the population variance σ^2

Let's see if statsmodels gives us the same results! For this we'll create a fake dataset:

[4]:
```
df = pd.DataFrame({'a':[13, 5, 11, 12, 9]})
arr = acovf(df['a'])
arr
```

[4]:
```
array([ 8. , -4. , -1.6,  2.2, -0.6])
```

2.2.2 Unbiased Autocovariance

Note that the number of terms in the calculations above are decreasing.Statsmodels can return an "unbiased" autocovariance where instead of dividing by n we divide by $n - k$.

$\gamma_0 = \frac{(13-10)(13-10)+(5-10)(5-10)+(11-10)(11-10)+(12-10)(12-10)+(9-10)(9-10)}{5-0} = \frac{40}{5} = 8.0$
$\gamma_1 = \frac{(13-10)(5-10)+(5-10)(11-10)+(11-10)(12-10)+(12-10)(9-10)}{5-1} = \frac{-20}{4} = -5.0$
$\gamma_2 = \frac{(13-10)(11-10)+(5-10)(12-10)+(11-10)(9-10)}{5-2} = \frac{-8}{3} = -2.67$
$\gamma_3 = \frac{(13-10)(12-10)+(5-10)(9-10)}{5-3} = \frac{11}{2} = 5.5$
$\gamma_4 = \frac{(13-10)(9-10)}{5-4} = \frac{-3}{1} = -3.0$

[5]:
```
arr2 = acovf(df['a'],unbiased=True)
arr2
```

[5]:
```
array([ 8.        , -5.        , -2.66666667,  5.5       , -3.        ])
```

2.3 Autocorrelation for 1D

The correlation ρ (rho) between two variables y_1, y_2 is given as:

2.3.1 $\rho = \frac{E[(y_1-\mu_1)(y_2-\mu_2)]}{\sigma_1\sigma_2} = \frac{Cov(y_1,y_2)}{\sigma_1\sigma_2}$,

where E is the expectation operator, μ_1, σ_1 and μ_2, σ_2 are the means and standard deviations of y_1 and y_2.

When working with a single variable (i.e. autocorrelation) we would consider y_1 to be the original series and y_2 a lagged version of it. Note that with autocorrelation we work with $\bar{y}$, that is, the full population mean, and not the means of the reduced set of lagged factors (see note below).

Thus, the formula for ρ_k for a time series at lag k is:

$$\rho_k = \frac{\sum\limits_{t=1}^{n-k} (y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum\limits_{t=1}^{n} (y_t - \bar{y})^2}$$

This can be written in terms of the covariance constant γ_k as:

$$\rho_k = \frac{\gamma_k n}{\gamma_0 n} = \frac{\gamma_k}{\sigma^2}$$

For example, $\rho_4 = \frac{\gamma_4}{\sigma^2} = \frac{-0.6}{8} = -0.075$

Note that ACF values are bound by -1 and 1. That is, $-1 \le \rho_k \le 1$

[6]:
```
arr3 = acf(df['a'])
arr3
```

[6]:
```
array([ 1.   , -0.5  , -0.2  ,  0.275, -0.075])
```

2.4 Partial Autocorrelation

Partial autocorrelations measure the linear dependence of one variable after removing the effect of other variable(s) that affect both variables. That is, the partial autocorrelation at lag k is the autocorrelation between y_t and y_{t+k} that is not accounted for by lags 1 through $k-1$.

A common method employs the non-recursive Yule-Walker Equations:

$\phi_0 = 1$
$\phi_1 = \rho_1 = -0.50$
$\phi_2 = \frac{\rho_2 - \rho_1{}^2}{1-\rho_1{}^2} = \frac{(-0.20)-(-0.50)^2}{1-(-0.50)^2} = \frac{-0.45}{0.75} = -0.60$

As k increases, we can solve for ϕ_k using matrix algebra and the Levinson–Durbin recursion algorithm which maps the sample autocorrelations ρ to a Toeplitz diagonal-constant matrix. The full solution is beyond the scope of this course, but the setup is as follows:

$$
\begin{pmatrix} \rho_0 & \rho_1 & \cdots & \rho_{k-1} \\ \rho_1 & \rho_0 & \cdots & \rho_{k-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{k-1} & \rho_{k-2} & \cdots & \rho_0 \end{pmatrix} \begin{pmatrix} \phi_{k1} \\ \phi_{k2} \\ \vdots \\ \phi_{kk} \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_k \end{pmatrix}
$$

[7]:
```
arr4 = pacf_yw(df['a'],nlags=4,method='mle')
arr4
```

[7]:
```
array([ 1.        , -0.5       , -0.6       , -0.38541667, -0.40563273])
```

NOTE: We passed in method='mle' above in order to use biased ACF coefficients. "mle" stands for "maximum likelihood estimation". Alternatively we can pass method='unbiased' (the statsmodels default):

[8]:
```
arr5 = pacf_yw(df['a'],nlags=4,method='unbiased')
arr5
```

[8]:
```
array([ 1.        , -0.625     , -1.18803419,  2.03764205,  0.8949589 ])
```

2.4.1 Partial Autocorrelation with OLS

This provides partial autocorrelations with ordinary least squares (OLS) estimates for each lag instead of Yule-Walker.

[9]:
```
arr6 = pacf_ols(df['a'],nlags=4)
arr6
```

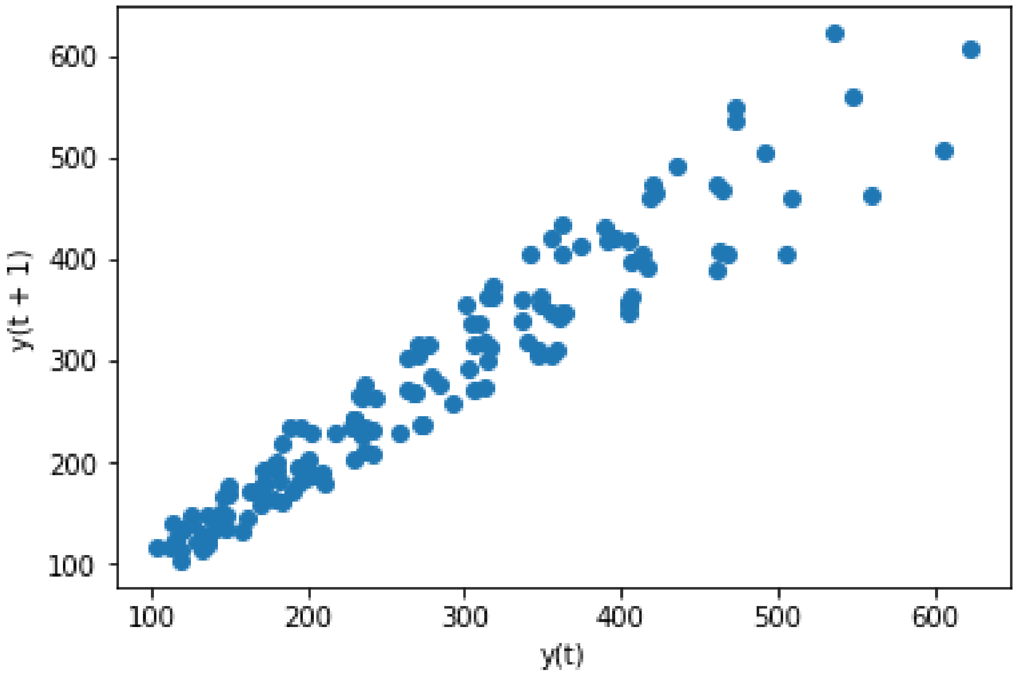
[9]:
```
array([ 1.        , -0.49677419, -0.43181818,  0.53082621,  0.25434783])
```

3 Plotting

The arrays returned by .acf(df) and .pacf_yw(df) show the magnitude of the autocorrelation for a given y at time t. Before we look at plotting arrays, let's look at the data itself for evidence of autocorrelation.

Pandas has a built-in plotting function that plots increasing y_t values on the horizontal axis against lagged versions of the values y_{t+1} on the vertical axis. If a dataset is non-stationary with an upward trend, then neighboring values should trend in the same way. Let's look at the Airline Passengers dataset first.

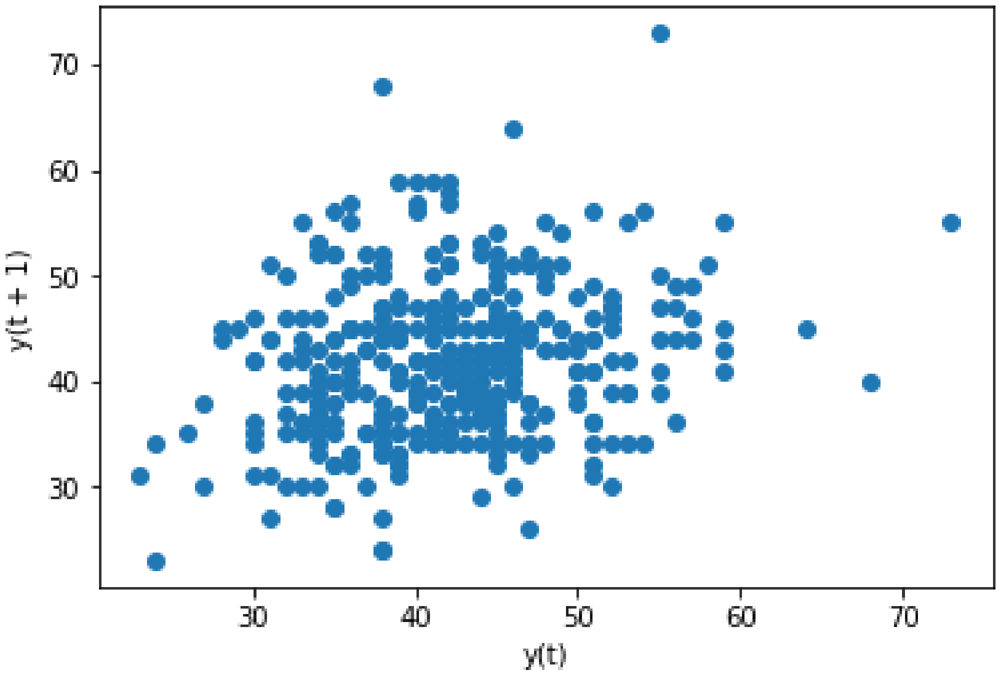
[10]:
```
from pandas.plotting import lag_plot

lag_plot(df1['Thousands of Passengers']);
```



Visually this shows evidence of a very strong autocorrelation; as y_t values increase, nearby (lagged) values also increase.

Now let's look at the stationary Daily Total Female Births dataset:

[11]:
```
lag_plot(df2['Births']);
```



As expected, there is little evidence of autocorrelation here.

3.1 ACF Plots

Plotting the magnitude of the autocorrelations over the first few (20-40) lags can say a lot about a time series.

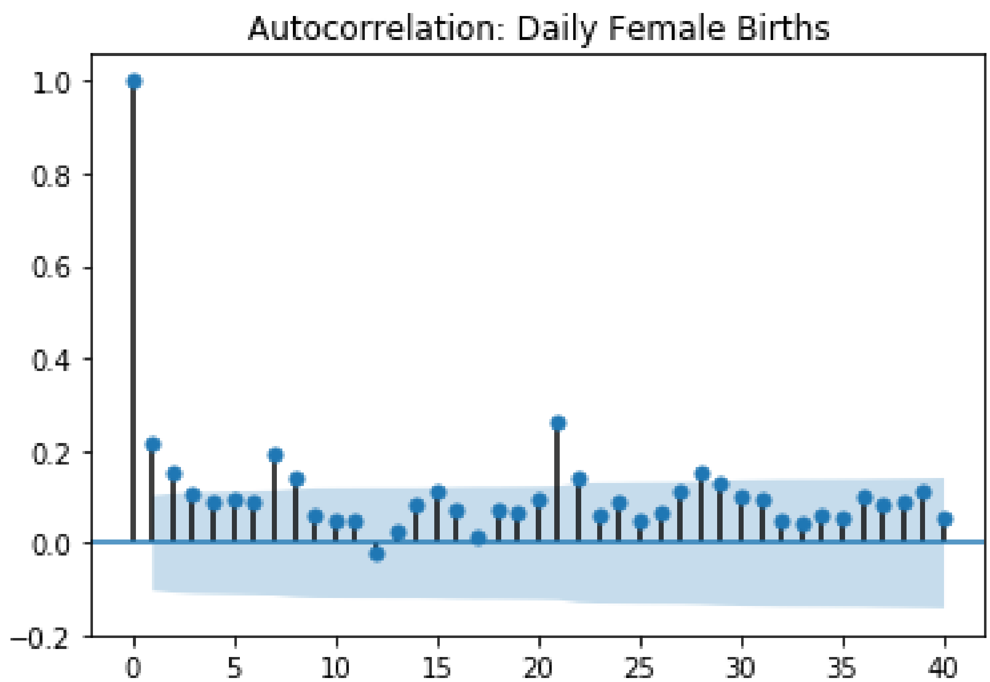
For example, consider the stationary Daily Total Female Births dataset:

[12]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

[13]:
```
# Let's look first at the ACF array. By default acf() returns 40 lags
acf(df2['Births'])
```

[13]:
```
array([ 1.        ,  0.21724118,  0.15287758,  0.10821254,  0.09066059,
        0.09595481,  0.09104012,  0.19508071,  0.14115295,  0.06117859,
        0.04781522,  0.04770662, -0.01964707,  0.02287422,  0.08112657,
        0.11185686,  0.07333732,  0.01501845,  0.07270333,  0.06859   ,
        0.09280107,  0.26386846,  0.14012147,  0.06070286,  0.08716232,
        0.05038825,  0.0650489 ,  0.11466565,  0.1552232 ,  0.12850638,
        0.10358981,  0.09734643,  0.04912286,  0.04022798,  0.05838555,
        0.05359812,  0.10151053,  0.08268663,  0.0912185 ,  0.11192192,
        0.05652846])
```

[14]:
```
# Now let's plot the autocorrelation at different lags
title = 'Autocorrelation: Daily Female Births'
lags = 40
plot_acf(df2,title=title,lags=lags);
```



This is a typical ACF plot for stationary data, with lags on the horizontal axis and correlations on the vertical axis. The first value y_0 is always 1. A sharp dropoff indicates that there is no AR component in the ARIMA model.

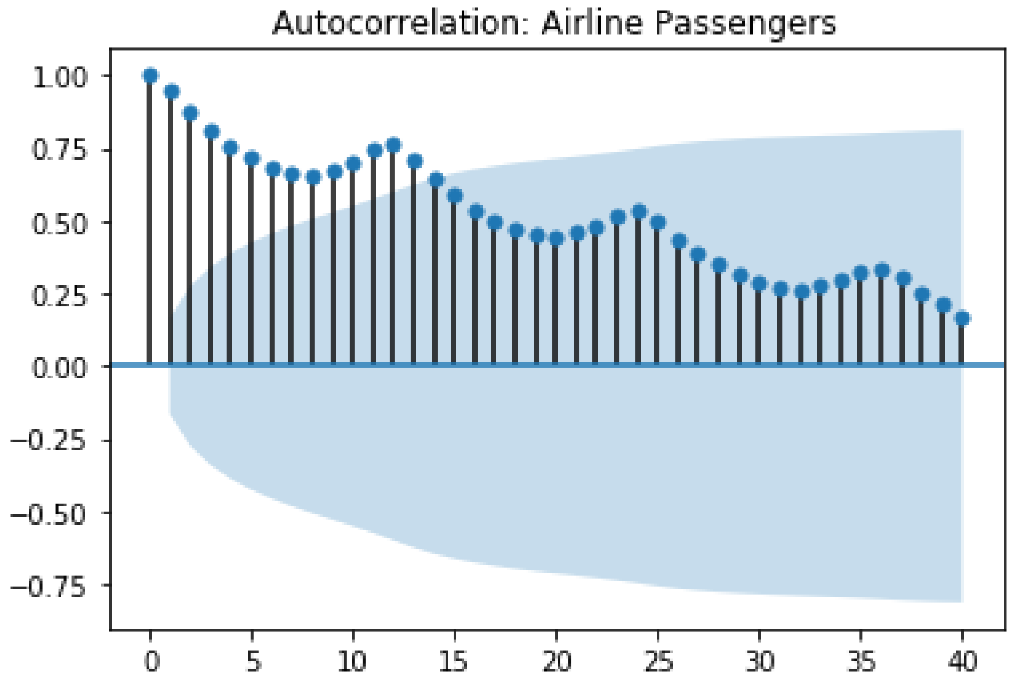
Next we'll look at non-stationary data with the Airline Passengers dataset:

[15]:
```
acf(df1['Thousands of Passengers'])
```

[15]:
```
array([1.        , 0.94804734, 0.87557484, 0.80668116, 0.75262542,
       0.71376997, 0.6817336 , 0.66290439, 0.65561048, 0.67094833,
       0.70271992, 0.74324019, 0.76039504, 0.71266087, 0.64634228,
       0.58592342, 0.53795519, 0.49974753, 0.46873401, 0.44987066,
       0.4416288 , 0.45722376, 0.48248203, 0.51712699, 0.53218983,
       0.49397569, 0.43772134, 0.3876029 , 0.34802503, 0.31498388,
       0.28849682, 0.27080187, 0.26429011, 0.27679934, 0.2985215 ,
       0.32558712, 0.3370236 , 0.30333486, 0.25397708, 0.21065534,
       0.17217092])
```

[16]:
```
title = 'Autocorrelation: Airline Passengers'
lags = 40
```

```
plot_acf(df1,title=title,lags=lags);
```

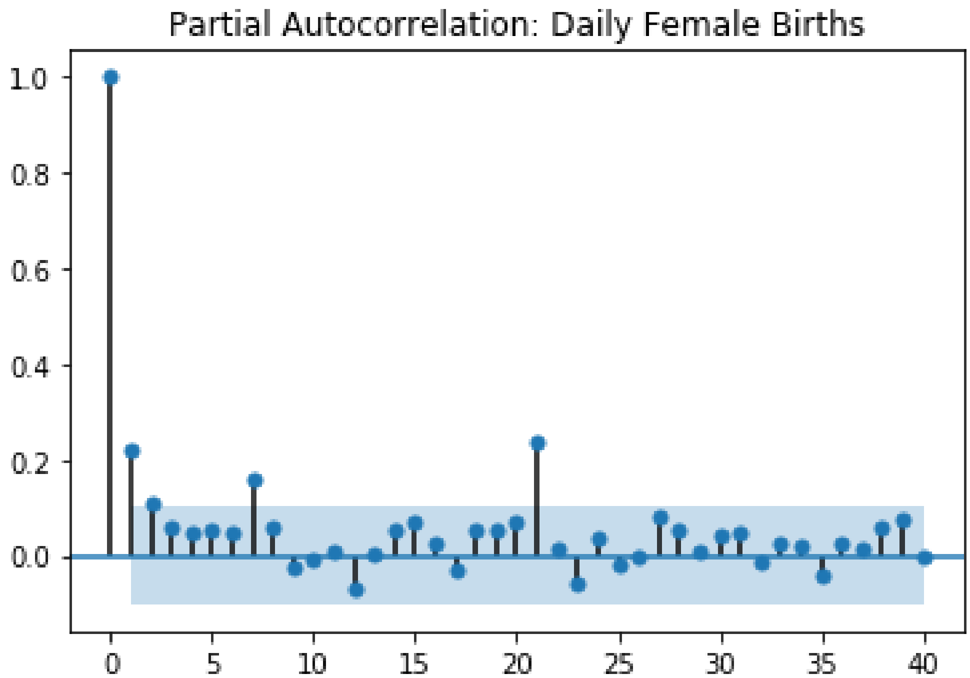


This plot indicates non-stationary data, as there are a large number of lags before ACF values drop off.

3.2 PACF Plots

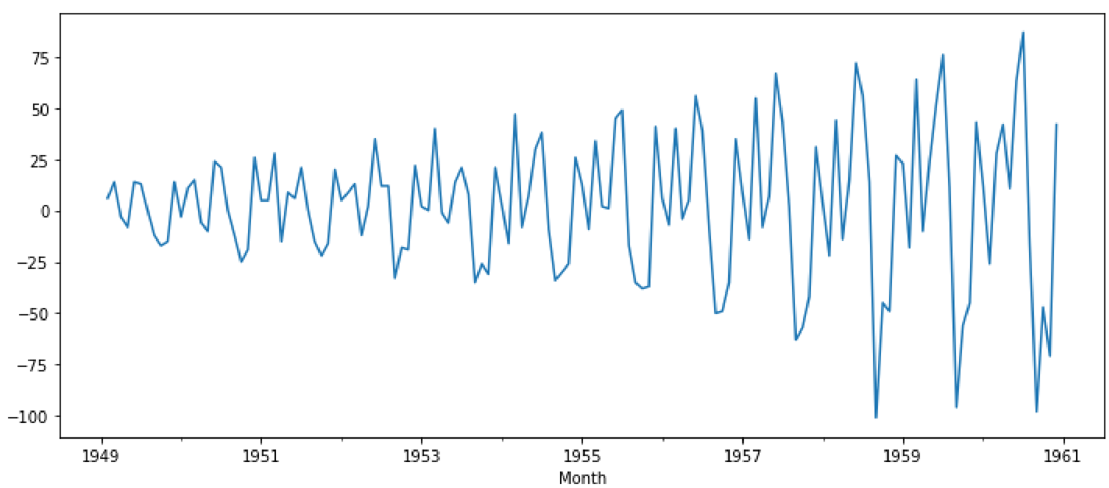
Partial autocorrelations work best with stationary data. Let's look first at Daily Total Female Births:

[17]:
```
title='Partial Autocorrelation: Daily Female Births'
lags=40
plot_pacf(df2,title=title,lags=lags);
```

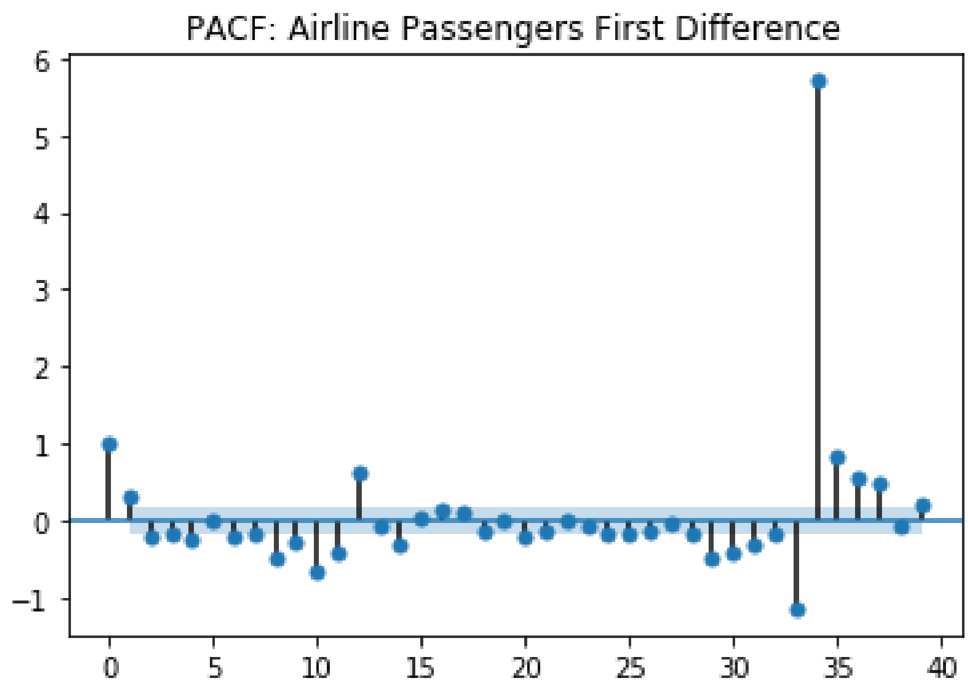


To make the Airline Passengers data stationary, we'll first apply differencing:

[18]:
```
from statsmodels.tsa.statespace.tools import diff

df1['d1'] = diff(df1['Thousands of Passengers'],k_diff=1)
df1['d1'].plot(figsize=(12,5));
```



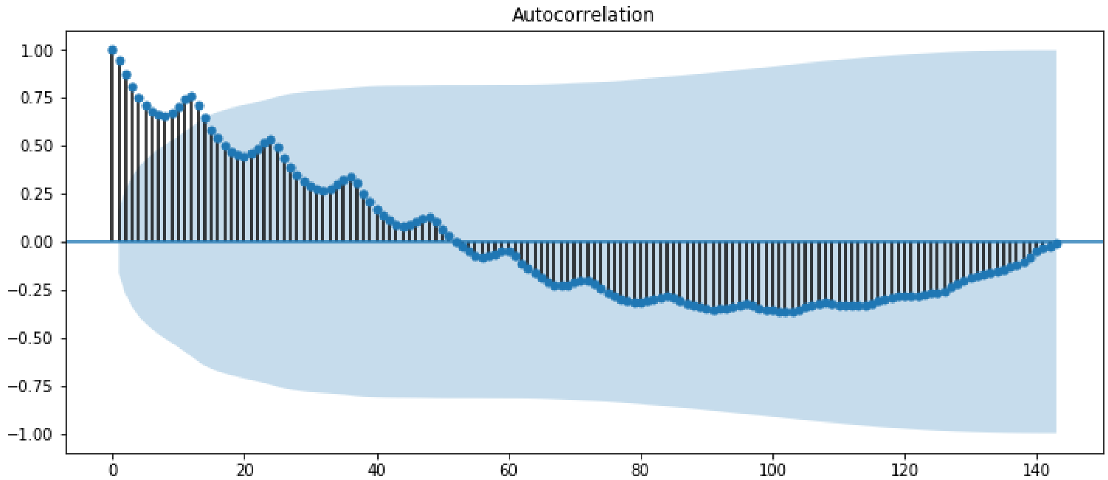
[19]:
```
title='PACF: Airline Passengers First Difference'
lags=40
plot_pacf(df1['d1'].dropna(),title=title,lags=np.arange(lags));  # be sure to␣
 ↪add .dropna() here!
```



3.2.1 Plot Resizing

In case you want to display the full autocorrelation plot, it helps to increase the figure size using matplotlib:

[21]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12,5))

plot_acf(df1['Thousands of Passengers'],ax=ax);
```



A NOTE ABOUT AUTOCORRELATION: Some texts compute lagged correlations using the Pearson Correlation Coefficient given by: $r_{xy} = \dfrac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$

These are easily calculated in numpy with numpy.corrcoef(x,y) and in Excel with =CORREL(x,y). Using our example, r_0 is still 1, but to solve for r_1:

$x_1 = [13, 5, 11, 12], \bar{x_1} = 10.25$ $y_1 = [5, 11, 12, 9]$, $\bar{y_1} = 9.25$ $r_{x_1 y_1} = \frac{(13-10.25)(5-9.25)+(5-10.25)(11-9.25)+(11-10.25)(12-9.25)+(12-10.25)(9-9.25)}{\sqrt{((13-10.25)^2+(5-10.25)^2+(11-10.25)^2+(12-10.25)^2)}\sqrt{((5-9.25)^2+(11-9.25)^2+(12-9.25)^2+(9-9.25)^2)}} = \frac{-19.25}{33.38} = -0.577$ However, there are some shortcomings. Using the Pearson method, the second-to-last term r_{k-1} will always be 1 and the last term r_k will always be undefined.