Correlation and Autocorrelation in Time Series

Introduction

In this lesson, we'll talk about correlation, autocorrelation, and partial autocorrelation in time series.

Objectives

You will be able to:

- Describe what role correlation plays in time series
- Plot and discuss the autocorrelation function (ACF) for a time series
- Plot and discuss the partial autocorrelation function (PACF) for a time series

Correlated Time Series

As you've seen before, correlation is a statistical technique that shows whether and how strongly pairs of variables are related to each other. For correlated variables, you've seen that the Pearson correlation coefficient can be used to summarize the correlation between the variables.

Knowing this, it's no surprise that time series can be correlated as well. To introduce the concept of correlated time series, we will use a dataset used in this blog post (https://www.datacamp.com/community/tutorials/time-series-analysis-tutorial). The dataset contains Google Trends data of three keywords: Diet, Gym, and Finance. Let's visualize this time series data.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

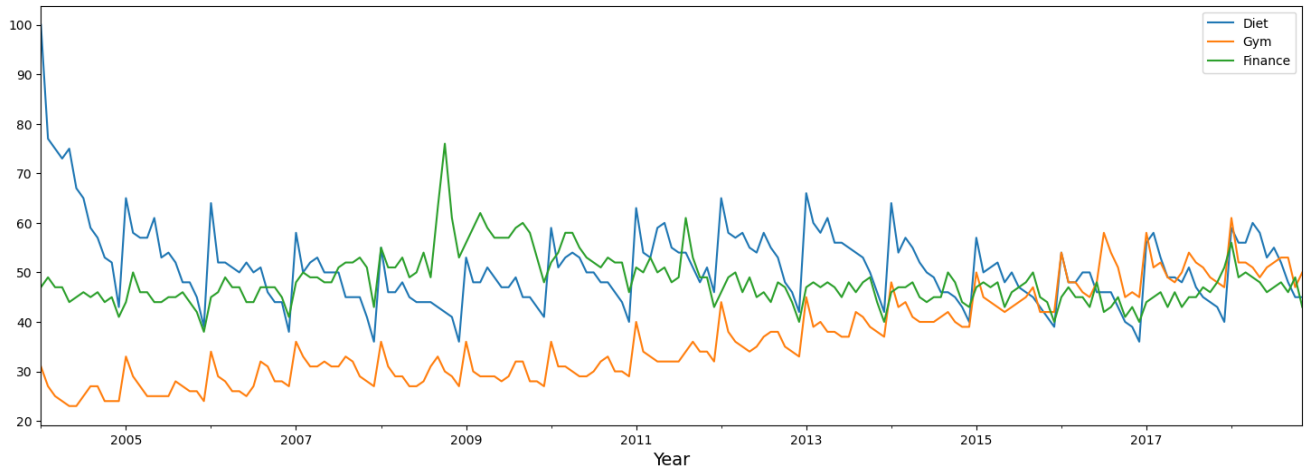
In [4]:
```
gtrends = pd.read_csv('google_trends.csv', skiprows=1)
gtrends.head()
```

Out[4]:

	Month	diet: (Worldwide)	gym: (Worldwide)	finance: (Worldwide)
0	2004-01	100	31	47
1	2004-02	77	27	49
2	2004-03	75	25	47
3	2004-04	73	24	47
4	2004-05	75	23	44

In [5]:
```
gtrends.columns = ['Month', 'Diet', 'Gym', 'Finance']
gtrends['Month'] = pd.to_datetime(gtrends['Month'])
gtrends.set_index('Month', inplace=True)
```

In [6]:
```
gtrends.plot(figsize=(18,6))
plt.xlabel('Year', fontsize=14);
```



These time series seem to exhibit some seasonality as well. Do you see what's happening? Especially for "Diet" and "Gym" there seems to be a peak in the beginning of each year. The famous New Year's Resolutions!

Not surprisingly, these two seem to move in similar directions at same times as well. We can use the `.corr()` method to formally find the correlation measure:

In [7]:
```
gtrends.corr()
```

Out[7]:

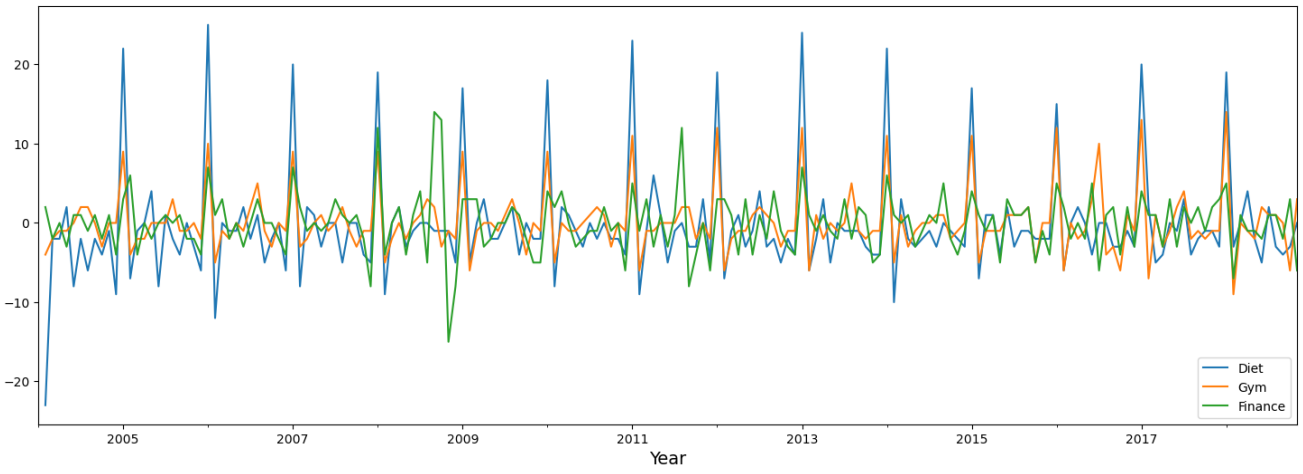
	Diet	Gym	Finance
Diet	1.000000	-0.050934	-0.026604
Gym	-0.050934	1.000000	-0.223186
Finance	-0.026604	-0.223186	1.000000

Interestingly, the correlations do not seem to be big, and have negative signs. But when we look at the plots, there are clearly some similar movements. What are we doing wrong?

Remember how we said that we want to make our time series **stationary**? This is where you can show off your detrending skills! Turns out you can easily find out if time series are correlated if you **detrend** them first. Let's use differencing to detrend these time series and then calculate the correlation again!

In [8]:
```
gtrends_diff = gtrends.diff(periods=1)
```

In [9]:
```
gtrends_diff.plot(figsize=(18,6))
plt.xlabel('Year', fontsize=14);
```



In [10]:
```
gtrends_diff.corr()
```

Out[10]:

	Diet	Gym	Finance
Diet	1.000000	0.793339	0.395105
Gym	0.793339	1.000000	0.341564
Finance	0.395105	0.341564	1.000000

So how did this happen? The spikes at the beginning of the year are a form of **seasonality**. By using 1-lag differencing you eliminated the simple trend, without getting rid of the seasonality. The trend "confused" the correlation coefficient, but after differencing, the correlation is very apparent.

Autocorrelation

Autocorrelation is a very powerful tool for time series analysis. It helps us study how each time series observation is related to its recent (or not so recent) past. Processes with greater autocorrelation are more predictable than those without any form of autocorrelation.

Let's start with comparing the time series of the keyword "Diet", with the time series with a lag of one. What you're essentially doing is comparing each value in the time series with it's previous value (in case of the "Diet" series, with the value in the previous month). This is called "lag 1 autocorrelation".

You can use the `.shift()` method in pandas to shift the index forward, or backward.

In [11]:
```
diet = gtrends[['Diet']]
```

In [12]:
```
diet_shift_1 = diet.shift(periods=1)
diet_shift_1.head()
```

Out[12]:

Month	Diet
2004-01-01	NaN
2004-02-01	100.0
2004-03-01	77.0
2004-04-01	75.0
2004-05-01	73.0

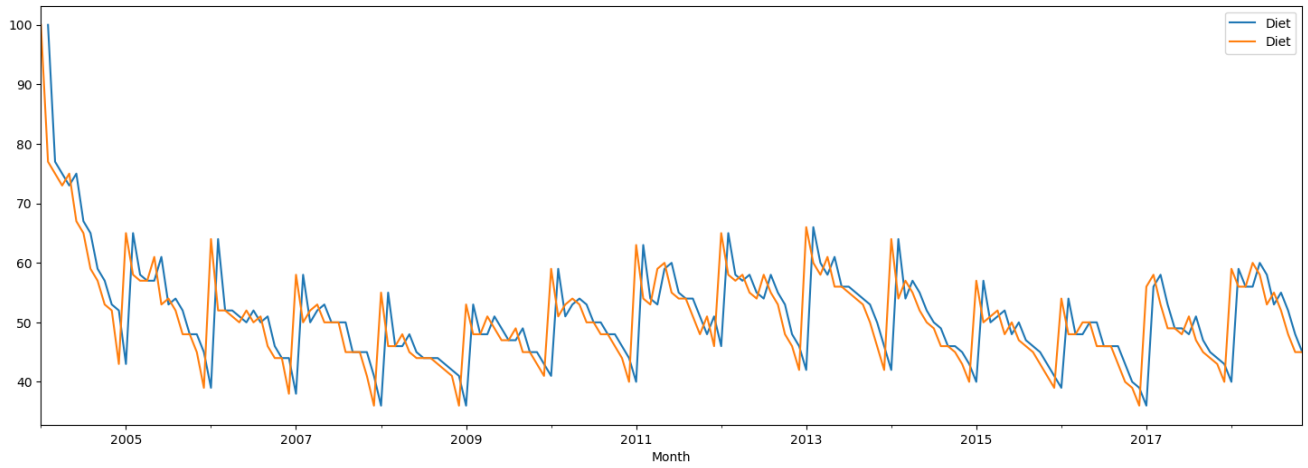
In [13]:
```
lag_1 = pd.concat([diet_shift_1, diet], axis=1)

lag_1.corr()
```

Out[13]:

	Diet	Diet
Diet	1.000000	0.624862
Diet	0.624862	1.000000

You can see that the "lag 1 autocorrelation" is 0.62. Let's plot them together to get a sense of what's happening:

In [14]:
```
lag_1.plot(figsize=(18,6));
```



Let's look at lag 2:

In [15]:
```
diet_shift_2 = diet.shift(periods=2)

lag_2 = pd.concat([diet_shift_2, diet], axis=1)

lag_2.corr()
```

Out[15]:

	Diet	Diet
Diet	1.000000	0.537913
Diet	0.537913	1.000000

The "lag 2 autocorrelation" is 0.54, so a little lower than the "lag 1 autocorrelation".

Now, how about a lag 12 autocorrelation?

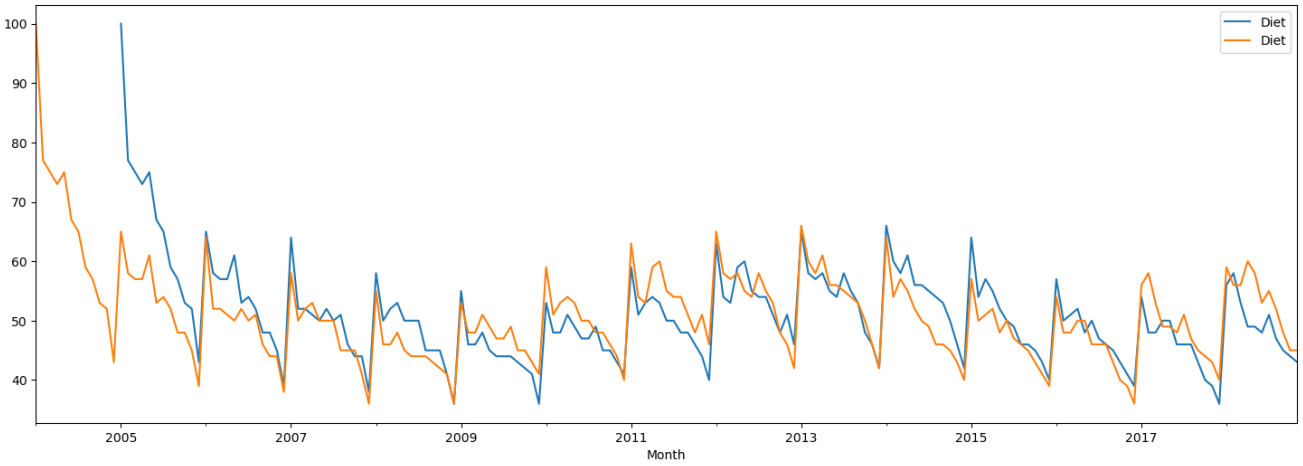
In [16]:
```
diet_shift_12 = diet.shift(periods=12)

lag_12 = pd.concat([diet_shift_12, diet], axis=1)

lag_12.corr()
```

Out[16]:

	Diet	Diet
Diet	1.000000	0.754955
Diet	0.754955	1.000000

Unsurprisingly, this autocorrelation is high! We're basically comparing the series by shifting our data by 1 year, so January 2004 is compared to January 2005, and so on. Let's visualize these series and the 12-lag shifted series as well.

In [17]:
```
lag_12.plot(figsize=(18,6));
```



The Autocorrelation Function

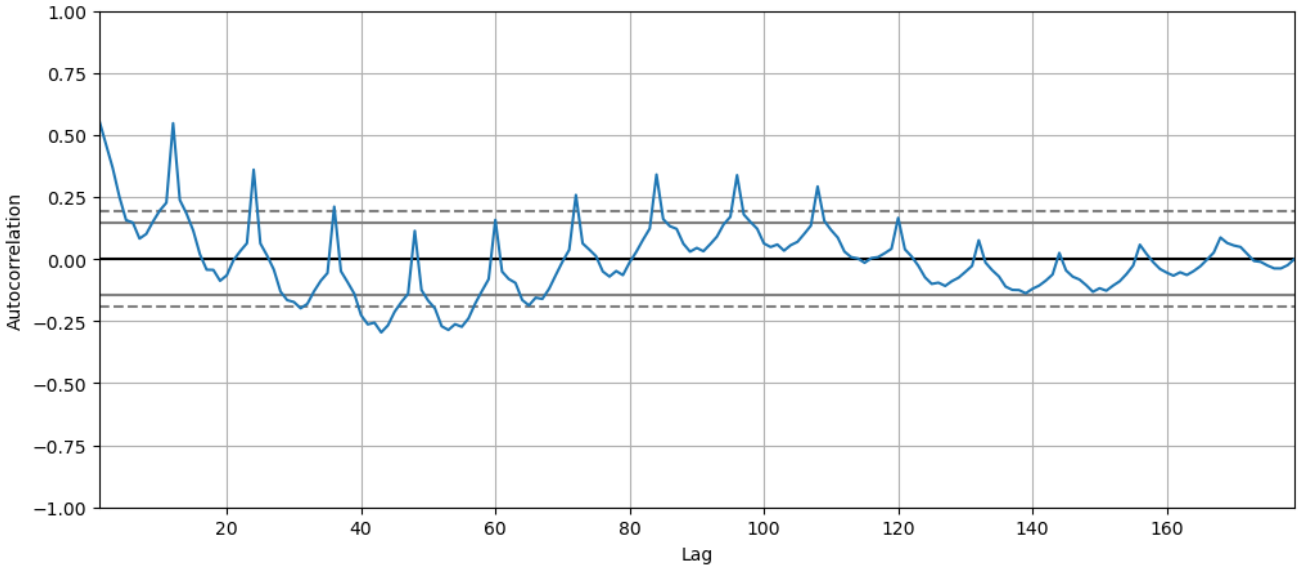
Great, but wouldn't it be nice to get a summary of the autocorrelations for each lag? Well, that's exactly what the **autocorrelation function** (often abbreviated to ACF) does. The autocorrelation function is a function that represents autocorrelation of a time series as a function of the time lag.

The autocorrelation function tells interesting stories about trends and seasonality. For example, if the original time series repeats itself every five days, you would expect to see a spike in the autocorrelation function at 5 days.

Creating an autocorrelation function for our "Diet" series, we have the lag on the x-axis and the correlation value for each respective lag value on the y-axis.

You can use the `autocorrelation_plot()` function in Pandas' `plotting` module.

In [20]:
```
plt.figure(figsize=(12,5))
pd.plotting.autocorrelation_plot(diet);
```

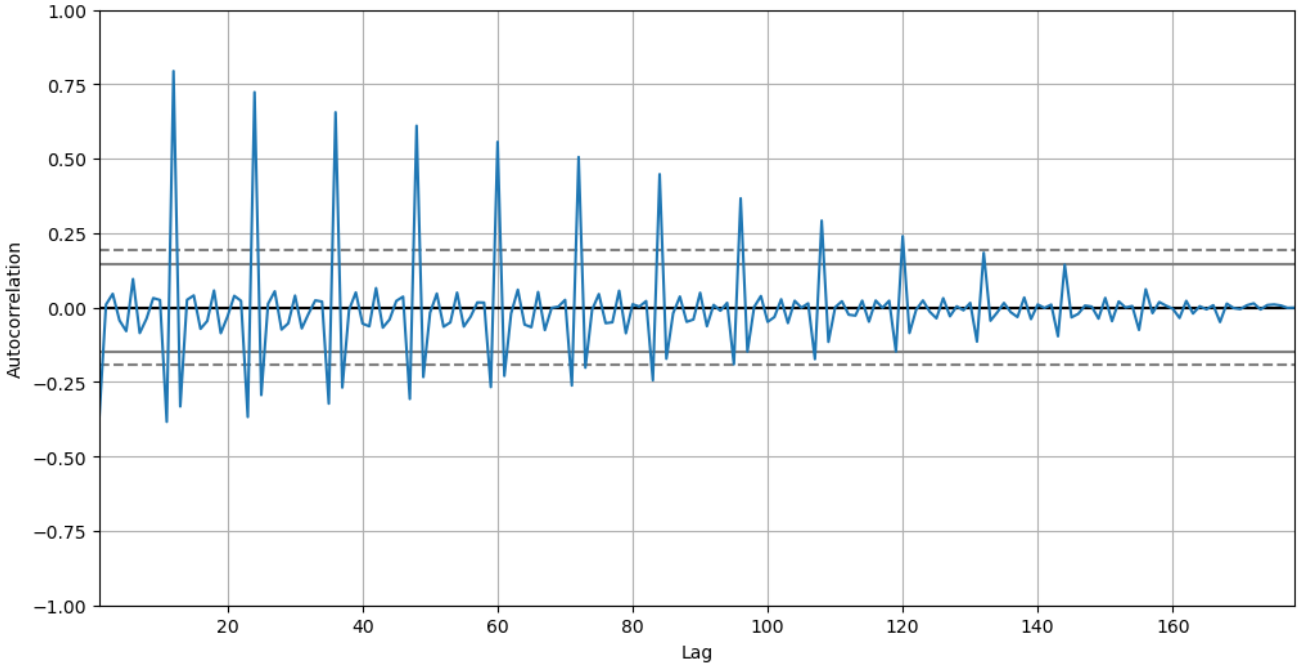


Look at that, you can clearly identify spikes for lags of multiples of 12. The dotted lines in the plot tell you about the statistical significance of the correlation. For this time series, you can say that "Diet" is definitely autocorrelated for lags of twelve months and 24 months, but for some later lags the result is not significant.

Like before, instead of plotting the autocorrelation function for the "Diet" series as is, we can also plot the autocorrelation function for the differenced series. Let's see how that changes our result.

In [21]:
```
diet_diff = gtrends_diff[['Diet']].dropna()
```

In [22]:
```
plt.figure(figsize=(12,6))
pd.plotting.autocorrelation_plot(diet_diff);
```



You can see that the ACF here seems a little more *stable*, revolving around 0, which is no surprise. Additionally, the autocorrelation for multiples of 12 seems consistently statistically significant, while it decays for longer time lags!

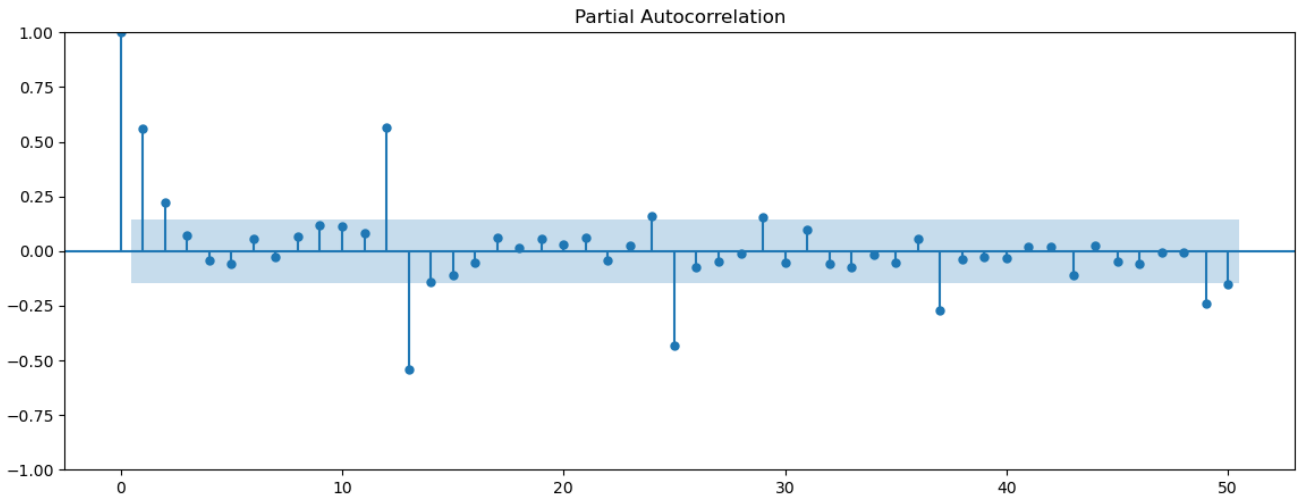
The Partial Autocorrelation Function

Similar to the autocorrelation function, the **Partial Autocorrelation Function** (or PACF) gives the partial correlation of a time series with its own lagged values, controlling for the values of the time series at all shorter lags (unlike the autocorrelation function, which does not control for other lags). PACF can be thought of as a summary of the relationship between a time series element with observations at a lag, *with the relationships of intervening observations removed*.

Let's plot the partial autocorrelation function of our "Diet" series. Although Pandas doesn't have a partial autocorrelation function, but luckily, `statsmodels` has one in its `tsaplots` module!

In [23]:
```
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 14, 5

plot_pacf(diet, lags=50);
```

```
/opt/saturncloud/envs/saturn/lib/python3.10/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default meth
od 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



The partial autocorrelation function can be interpreted as a regression of the series against its past lags. It helps you come up with a possible order for the auto regressive term. The terms can be interpreted the same way as a standard linear regression, that is the contribution of a change in that particular lag while holding others constant. The use of PACF will become more clear when we will be looking at some more "advanced" time series models!

> NOTE: There is also a function `plot_acf()` in `statsmodels`, which serves as an alternative to Pandas' `autocorrelation_plot()`.

In [ ]:
```
from statsmodels.graphics.tsaplots import plot_acf
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 14, 5

plot_acf(diet, lags=50);
```

Note that the plots (and especially the confidence bands) are slightly different. Feel free to have a look at this stackoverflow post (https://stackoverflow.com/questions/36038927/whats-the-difference-between-pandas-acf-and-statsmodel-acf) if you want to dig deeper.

Addional reading

This blogpost (https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/) gives a great overview on the concepts in this lesson, and we **strongly recommend** you read it!

Summary

Great, you've now been introduced to correlation, the ACF, and PACF. Let's practice in the next lab!