S+P_Week_1_Lesson_2

December 6, 2020

[ ]:
```python
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

1 Lesson 2

In the screencast for this lesson I go through a few scenarios for time series. This notebook contains the code for that with a few little extras! :)

2 Setup

[ ]:
```python
!pip install -U tf-nightly-2.0-preview
```

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
```

[ ]:
```python
def plot_series(time, series, format="-", start=0, end=None, label=None):
    plt.plot(time[start:end], series[start:end], format, label=label)
    plt.xlabel("Time")
    plt.ylabel("Value")
    if label:
        plt.legend(fontsize=14)
    plt.grid(True)
```

3 Trend and Seasonality

[ ]:
```
def trend(time, slope=0):
    return slope * time
```

Let's create a time series that just trends upward:

[ ]:
```
time = np.arange(4 * 365 + 1)
baseline = 10
series = trend(time, 0.1)

plt.figure(figsize=(10, 6))
plot_series(time, series)
plt.show()
```

Now let's generate a time series with a seasonal pattern:

[ ]:
```
def seasonal_pattern(season_time):
    """Just an arbitrary pattern, you can change it if you wish"""
    return np.where(season_time < 0.4,
                    np.cos(season_time * 2 * np.pi),
                    1 / np.exp(3 * season_time))

def seasonality(time, period, amplitude=1, phase=0):
    """Repeats the same pattern at each period"""
    season_time = ((time + phase) % period) / period
    return amplitude * seasonal_pattern(season_time)
```

[ ]:
```
baseline = 10
amplitude = 40
series = seasonality(time, period=365, amplitude=amplitude)

plt.figure(figsize=(10, 6))
plot_series(time, series)
plt.show()
```

Now let's create a time series with both trend and seasonality:

[ ]:
```
slope = 0.05
series = baseline + trend(time, slope) + seasonality(time, period=365,␣
 ↪amplitude=amplitude)

plt.figure(figsize=(10, 6))
plot_series(time, series)
plt.show()
```

4 Noise

In practice few real-life time series have such a smooth signal. They usually have some noise, and the signal-to-noise ratio can sometimes be very low. Let's generate some white noise:

[ ]:
```
def white_noise(time, noise_level=1, seed=None):
    rnd = np.random.RandomState(seed)
    return rnd.randn(len(time)) * noise_level
```

[ ]:
```
noise_level = 5
noise = white_noise(time, noise_level, seed=42)

plt.figure(figsize=(10, 6))
plot_series(time, noise)
plt.show()
```

Now let's add this white noise to the time series:

[ ]:
```
series += noise

plt.figure(figsize=(10, 6))
plot_series(time, series)
plt.show()
```

All right, this looks realistic enough for now. Let's try to forecast it. We will split it into two periods: the training period and the validation period (in many cases, you would also want to have a test period). The split will be at time step 1000.

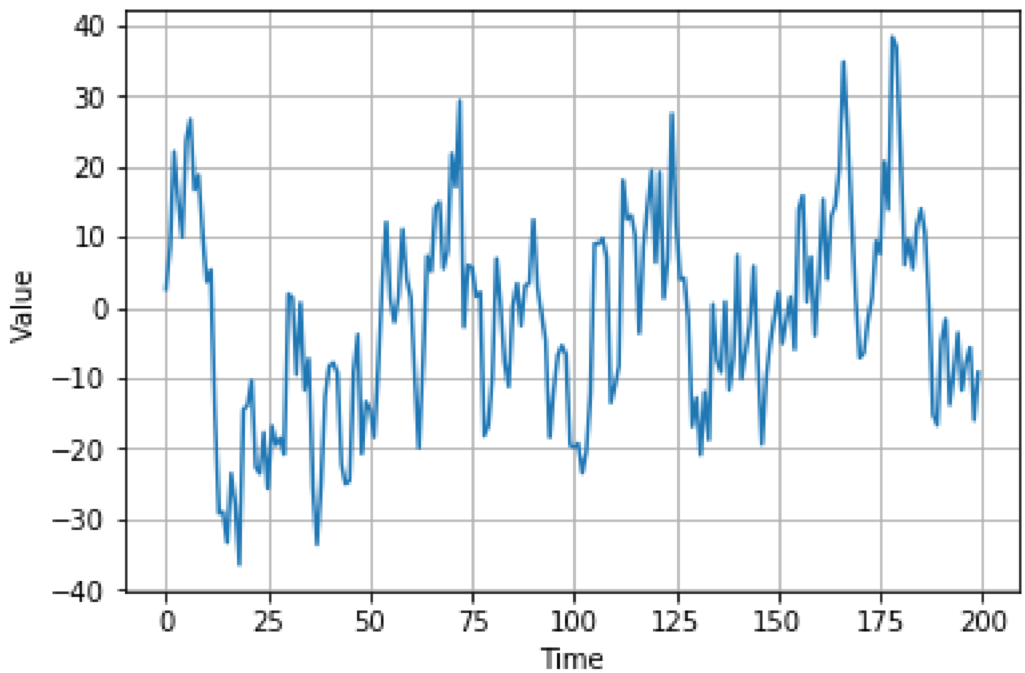
[ ]:
```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]
```

[ ]:
```
def autocorrelation(time, amplitude, seed=None):
    rnd = np.random.RandomState(seed)
     1 = 0.5
     2 = -0.1
    ar = rnd.randn(len(time) + 50)
    ar[:50] = 100
    for step in range(50, len(time) + 50):
        ar[step] +=  1 * ar[step - 50]
        ar[step] +=  2 * ar[step - 33]
    return ar[50:] * amplitude
```

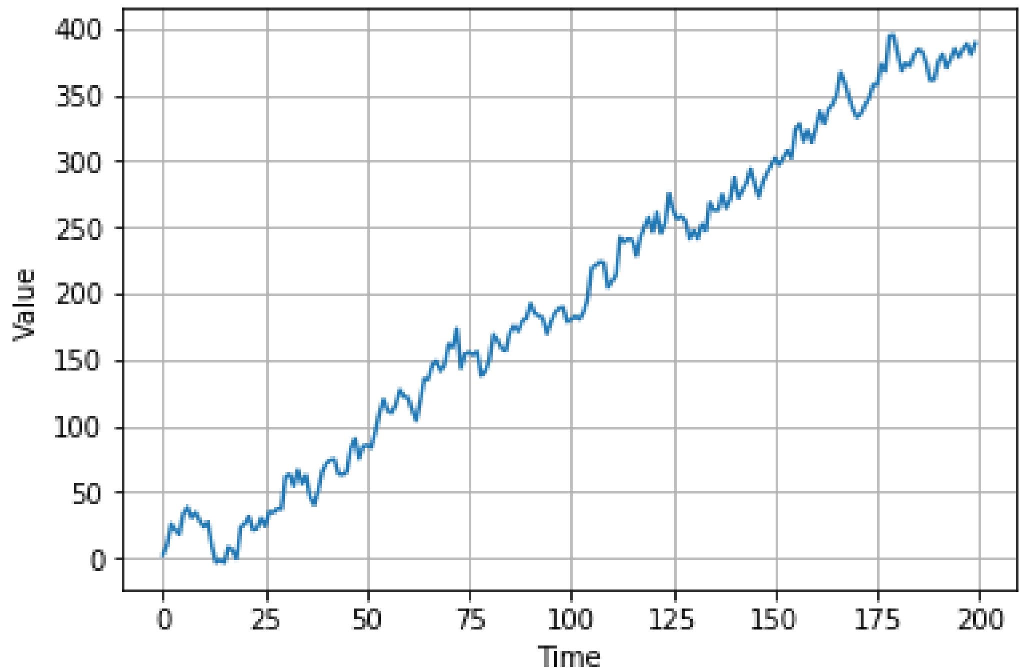
[ ]:
```
def autocorrelation(time, amplitude, seed=None):
    rnd = np.random.RandomState(seed)
      = 0.8
    ar = rnd.randn(len(time) + 1)
```

```
    for step in range(1, len(time) + 1):
        ar[step] +=   * ar[step - 1]
    return ar[1:] * amplitude
```

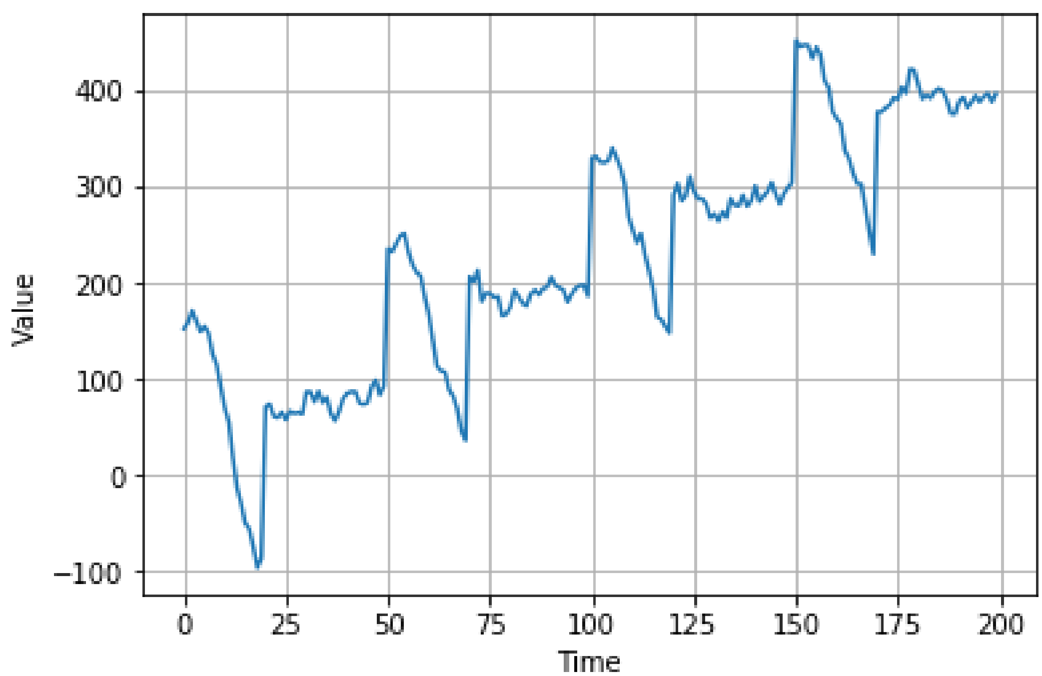
[ ]:
```
series = autocorrelation(time, 10, seed=42)
plot_series(time[:200], series[:200])
plt.show()
```



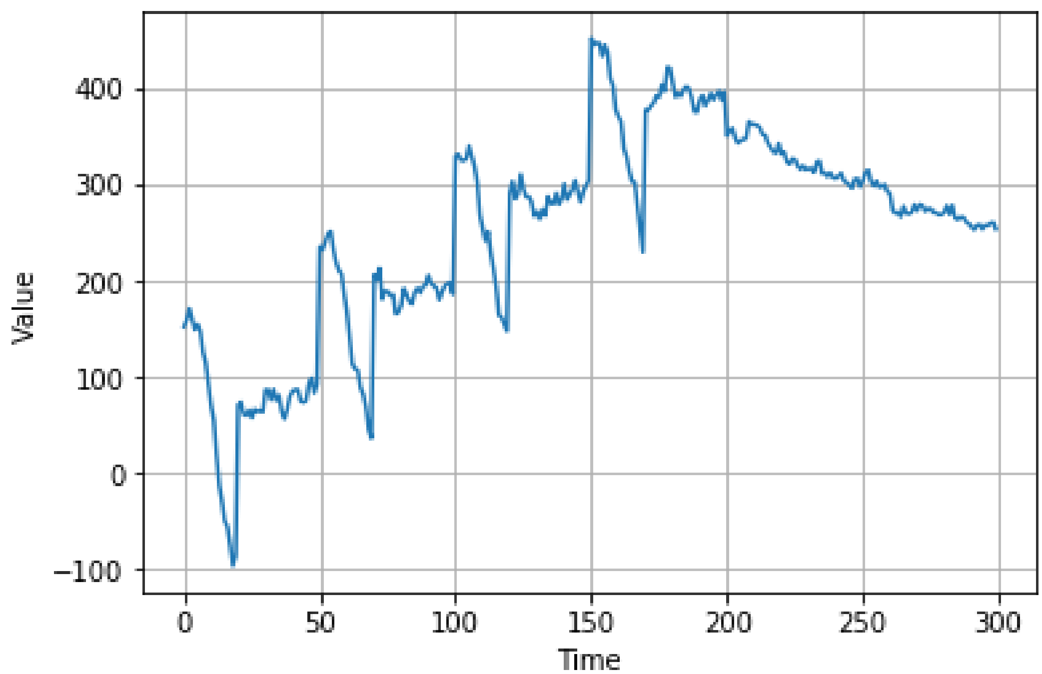
[ ]:
```
series = autocorrelation(time, 10, seed=42) + trend(time, 2)
plot_series(time[:200], series[:200])
plt.show()
```



[ ]:
```
series = autocorrelation(time, 10, seed=42) + seasonality(time, period=50,␣
 ↪amplitude=150) + trend(time, 2)
plot_series(time[:200], series[:200])
plt.show()
```

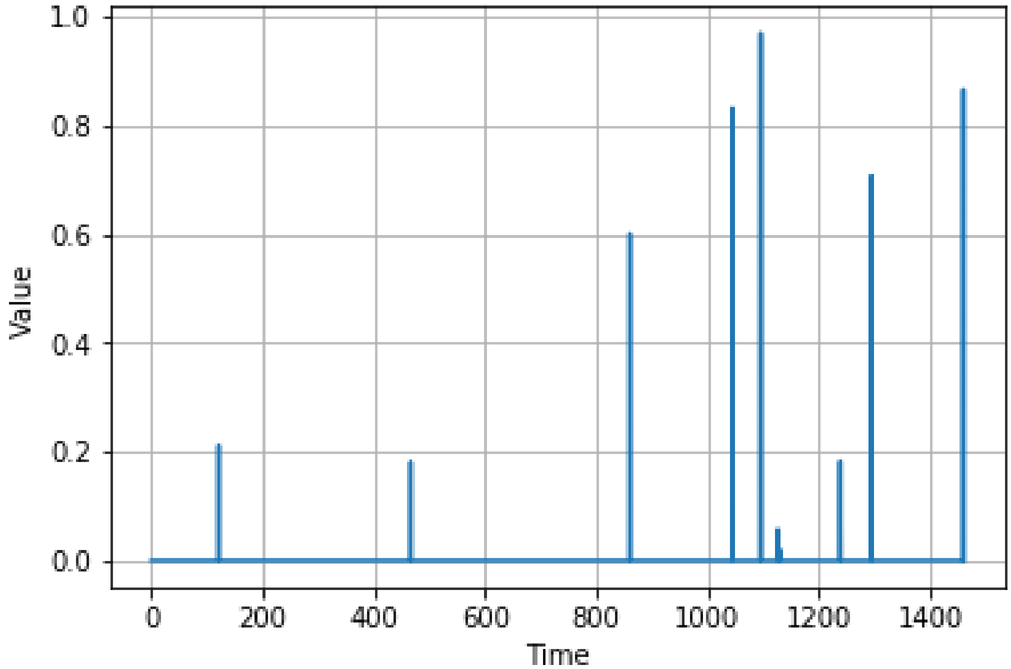


```
[ ]: series = autocorrelation(time, 10, seed=42) + seasonality(time, period=50,␣
     ↪amplitude=150) + trend(time, 2)
     series2 = autocorrelation(time, 5, seed=42) + seasonality(time, period=50,␣
     ↪amplitude=2) + trend(time, -1) + 550
     series[200:] = series2[200:]
     #series += noise(time, 30)
     plot_series(time[:300], series[:300])
     plt.show()
```



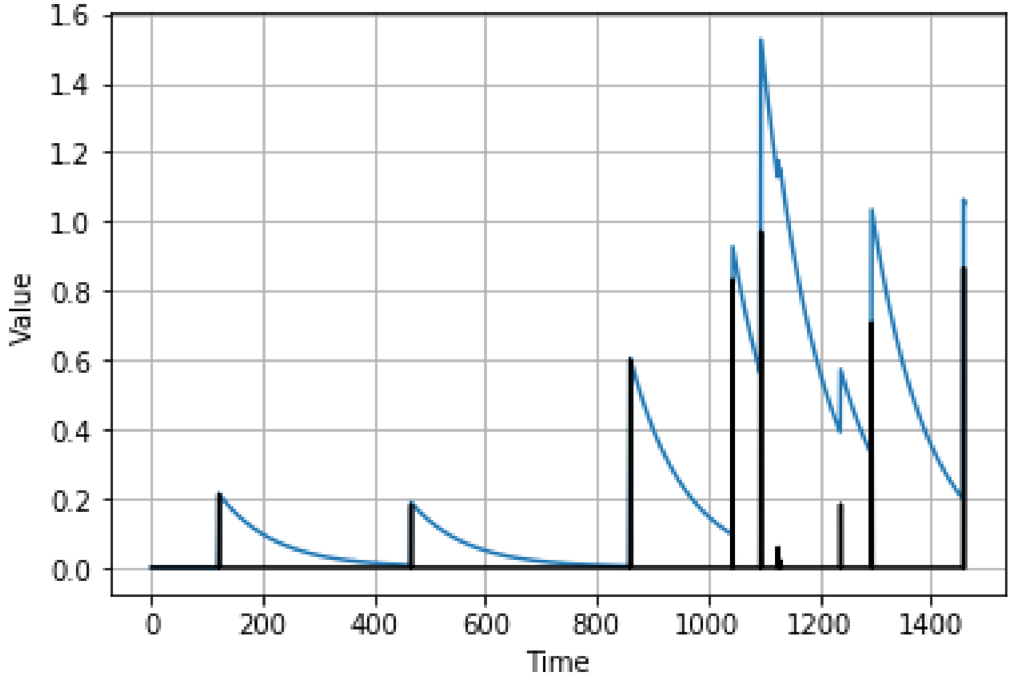

```
[ ]: def impulses(time, num_impulses, amplitude=1, seed=None):
         rnd = np.random.RandomState(seed)
         impulse_indices = rnd.randint(len(time), size=10)
         series = np.zeros(len(time))
         for index in impulse_indices:
             series[index] += rnd.rand() * amplitude
         return series
```

```
[ ]: series = impulses(time, 10, seed=42)
     plot_series(time, series)
     plt.show()
```

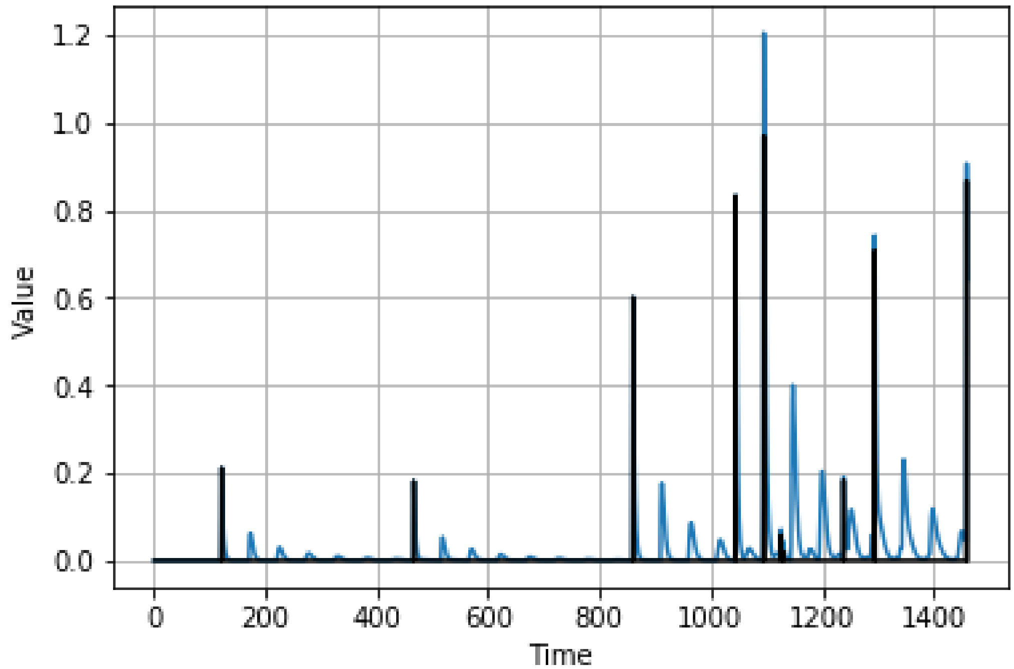


[ ]:
```
def autocorrelation(source, s):
    ar = source.copy()
    max_lag = len( s)
    for step, value in enumerate(source):
        for lag,   in s.items():
            if step - lag > 0:
              ar[step] +=   * ar[step - lag]
    return ar
```

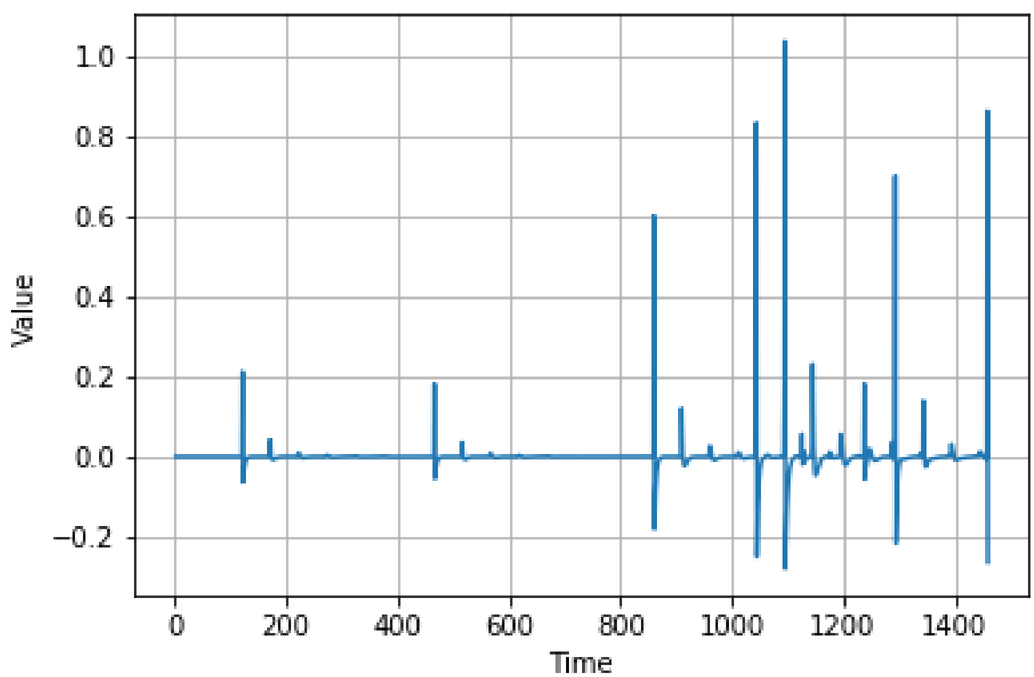
[ ]:
```
signal = impulses(time, 10, seed=42)
series = autocorrelation(signal, {1: 0.99})
plot_series(time, series)
plt.plot(time, signal, "k-")
plt.show()
```



[ ]:
```
signal = impulses(time, 10, seed=42)
series = autocorrelation(signal, {1: 0.70, 50: 0.2})
plot_series(time, series)
plt.plot(time, signal, "k-")
plt.show()
```



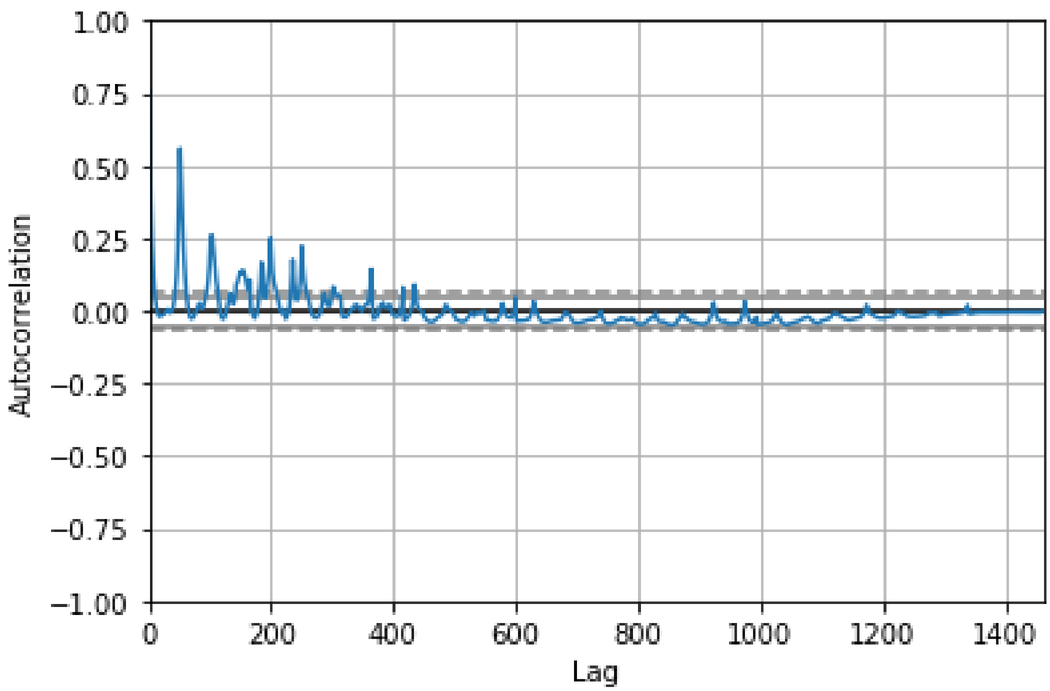
[ ]:
```
series_diff1 = series[1:] - series[:-1]
plot_series(time[1:], series_diff1)
```



[ ]:
```
from pandas.plotting import autocorrelation_plot

autocorrelation_plot(series)
```

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f53e49d9860>



[ ]:
```
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(series, order=(5, 1, 0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                 1460
Model:                 ARIMA(5, 1, 0)   Log Likelihood                2223.428
Method:                       css-mle   S.D. of innovations              0.053
Date:                Tue, 30 Jul 2019   AIC                          -4432.855
Time:                        17:40:55   BIC                          -4395.852
Sample:                             1   HQIC                         -4419.052

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
```

```
================================================================================
const          0.0003      0.001      0.384      0.701     -0.001      0.002
ar.L1.D.y     -0.1235      0.026     -4.714      0.000     -0.175     -0.072
ar.L2.D.y     -0.1254      0.029     -4.333      0.000     -0.182     -0.069
ar.L3.D.y     -0.1089      0.029     -3.759      0.000     -0.166     -0.052
ar.L4.D.y     -0.0914      0.029     -3.162      0.002     -0.148     -0.035
ar.L5.D.y     -0.0774      0.029     -2.675      0.008     -0.134     -0.021
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0145           -1.1311j            1.5194           -0.1336
AR.2            1.0145           +1.1311j            1.5194            0.1336
AR.3           -1.8173           -0.0000j            1.8173           -0.5000
AR.4           -0.6967           -1.6113j            1.7554           -0.3150
AR.5           -0.6967           +1.6113j            1.7554            0.3150
-----------------------------------------------------------------------------
```

[ ]:
```
df = pd.read_csv("sunspots.csv", parse_dates=["Date"], index_col="Date")
series = df["Monthly Mean Total Sunspot Number"].asfreq("1M")
series.head()
```

[ ]:
```
series.plot(figsize=(12, 5))
```

[ ]:
```
series["1995-01-01":].plot()
```

[ ]:
```
series.diff(1).plot()
plt.axis([0, 100, -50, 50])
```

[ ]:
```
from pandas.plotting import autocorrelation_plot

autocorrelation_plot(series)
```

[ ]:
```
autocorrelation_plot(series.diff(1)[1:])
```

[ ]:
```
autocorrelation_plot(series.diff(1)[1:].diff(11 * 12)[11*12+1:])
plt.axis([0, 500, -0.1, 0.1])
```

[ ]:
```
autocorrelation_plot(series.diff(1)[1:])
plt.axis([0, 50, -0.1, 0.1])
```

[ ]:
```
116.7 - 104.3
```

[ ]:
```
[series.autocorr(lag) for lag in range(1, 50)]
```

[ ]:
```

```

```
pd.read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer',␣
 ↪names=None, index_col=None, usecols=None, squeeze=False, prefix=None,␣
 ↪mangle_dupe_cols=True, dtype=None, engine=None, converters=None,␣
 ↪true_values=None, false_values=None, skipinitialspace=False, skiprows=None,␣
 ↪skipfooter=0, nrows=None, na_values=None, keep_default_na=True,␣
 ↪na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False,␣
 ↪infer_datetime_format=False, keep_date_col=False, date_parser=None,␣
 ↪dayfirst=False, iterator=False, chunksize=None, compression='infer',␣
 ↪thousands=None, decimal=b'.', lineterminator=None, quotechar='"', quoting=0,␣
 ↪doublequote=True, escapechar=None, comment=None, encoding=None,␣
 ↪dialect=None, tupleize_cols=None, error_bad_lines=True, warn_bad_lines=True,␣
 ↪delim_whitespace=False, low_memory=True, memory_map=False,␣
 ↪float_precision=None)
Read a comma-separated values (csv) file into DataFrame.
```

[ ]:
```
from pandas.plotting import autocorrelation_plot

series_diff = series
for lag in range(50):
  series_diff = series_diff[1:] - series_diff[:-1]

autocorrelation_plot(series_diff)
```

[ ]:
```
import pandas as pd

series_diff1 = pd.Series(series[1:] - series[:-1])
autocorrs = [series_diff1.autocorr(lag) for lag in range(1, 60)]
plt.plot(autocorrs)
plt.show()
```