ARMA Models in StatsModels

Introduction

In this lesson, you'll use your knowledge of the autoregressive (AR) and moving average (MA) models, along with the `statsmodels` library to model time series data.

Objectives

You will be able to:

- Fit an AR model using `statsmodels`
- Fit an MA model using `statsmodels`

Generate a first order AR model

Recall that the AR model has the following formula:

$$Y_t = \mu + \phi * Y_{t-1} + \epsilon_t$$

This means that:

$$Y_1 = \mu + \phi * Y_0 + \epsilon_1$$
$$Y_2 = \mu + \phi * (\text{mean-centered version of } Y_1) + \epsilon_2$$

and so on.

Let's assume a mean-zero white noise with a standard deviation of 2. We'll also create a daily datetime index ranging from January 2017 until the end of March 2018.

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

np.random.seed(11225)

# Create a series with the specified dates
dates = pd.date_range('2017-01-01', '2018-03-31')
```

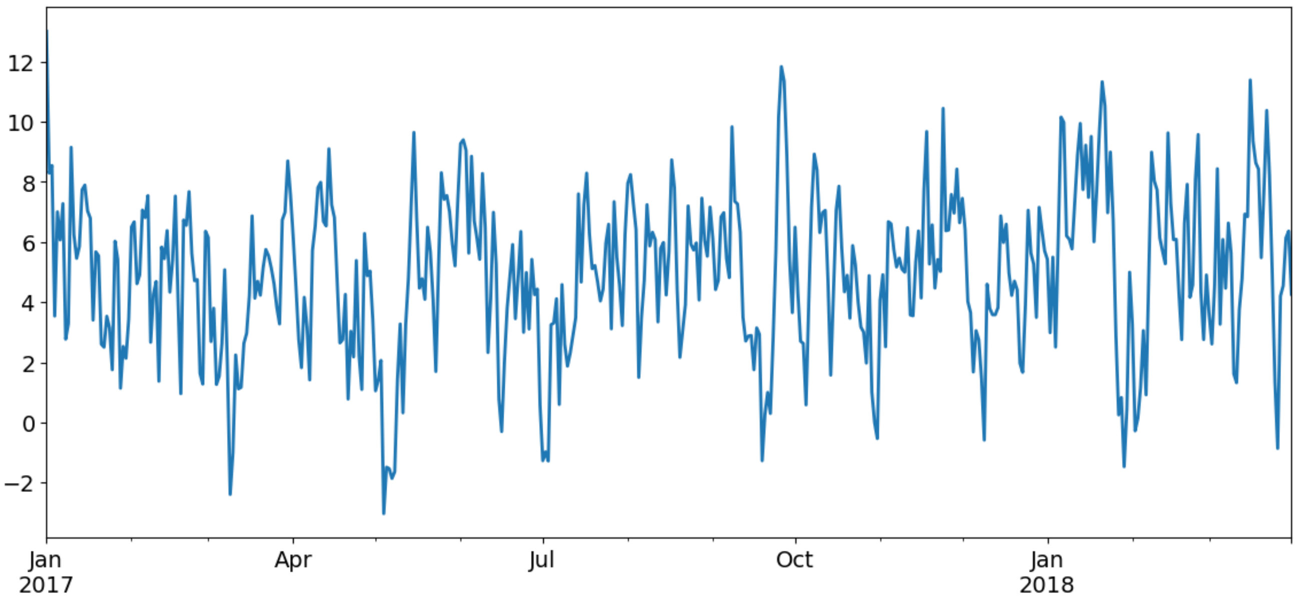
We will generate a first order AR model with $\phi = 0.7$, $\mu = 5$, and $Y_0 = 8$.

In [2]:
```
error = np.random.normal(0, 2, len(dates))
Y_0 = 8
mu = 5
phi = 0.7
```

In [3]:
```
TS = [None] * len(dates)
y = Y_0
for i, row in enumerate(dates):
    TS[i] = mu + y * phi + error[i]
    y = TS[i] - mu
```

Let's plot the time series to verify:

In [4]:
```
series = pd.Series(TS, index=dates)

series.plot(figsize=(14,6), linewidth=2, fontsize=14);
```

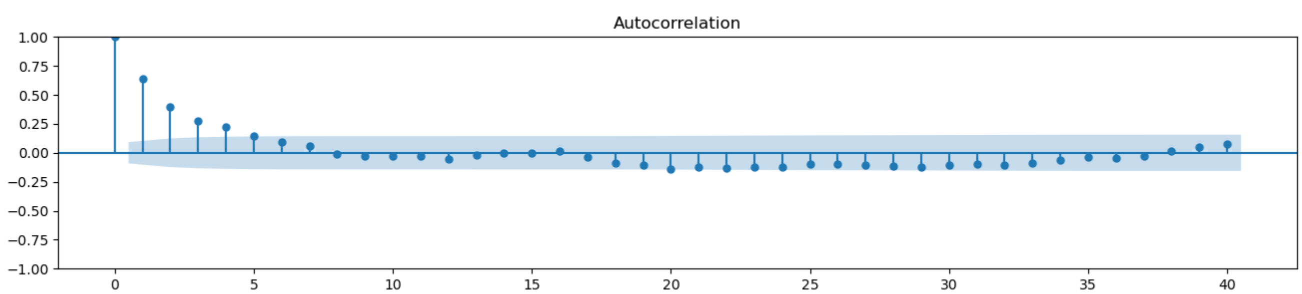


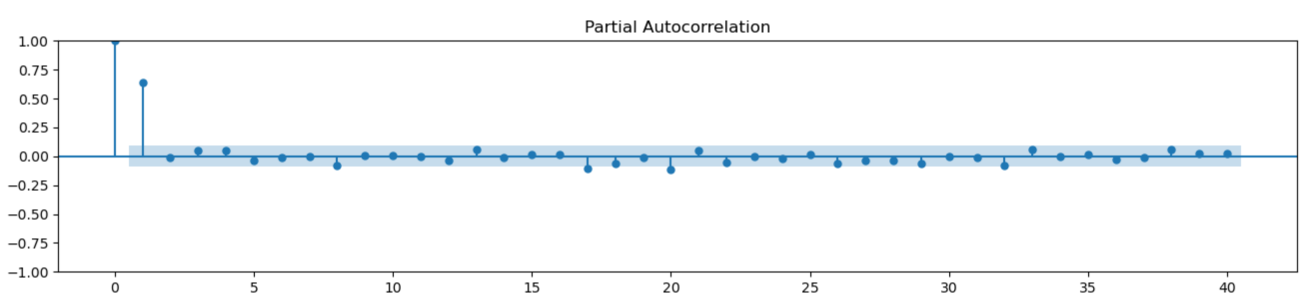
Look at the ACF and PACF of the model

Although we can use `pandas` to plot the ACF, we highly recommended that you use the `statsmodels` variant instead.

In [5]:
```
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

fig, ax = plt.subplots(figsize=(16,3))
plot_acf(series, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(series, ax=ax, lags=40);
```

```
/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(
```





Check the model with ARMA in `statsmodels`

`statsmodels` also has a tool that fits ARMA models to time series. The only thing we have to do is provide the number of orders for AR and MA. Have a look at the code below, and the output of the code.

The `ARIMA()` function requires two arguments:

1. The first is the time series to which the model is fit
2. The second is the `order` of the model in the form `(p, d, q)`

- `p` refers to the order of AR
- `d` refers to the order of I (which will be discussed in a future lesson -- for now just use 0)
- `q` refers to the order of MA

For example, a first order AR model would be represented as `(1,0,0)` .

In [6]:
```
# Import ARIMA
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm

# Instantiate an AR(1) model to the simulated data
mod_arma = ARIMA(series, order=(1,0,0))
```

Once you have instantiated the AR(1) model, you can call the `.fit()` method to the fit the model to the data.

In [7]:
```
# Fit the model to data
res_arma = mod_arma.fit()
```

Similar to other models, you can then call the `.summary()` method to print the information of the model.

In [8]:
```
# Print out summary information on the fit
print(res_arma.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  455
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -968.698
Date:                Sat, 19 Nov 2022   AIC                           1943.395
Time:                        12:13:39   BIC                           1955.756
Sample:                    01-01-2017   HQIC                          1948.265
                         - 03-31-2018
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.9664      0.262     18.937      0.000       4.452       5.480
ar.L1          0.6474      0.036     18.108      0.000       0.577       0.718
sigma2         4.1327      0.289     14.289      0.000       3.566       4.700
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 0.99
Prob(Q):                              0.99   Prob(JB):                         0.61
Heteroskedasticity (H):               1.02   Skew:                             0.08
Prob(H) (two-sided):                  0.89   Kurtosis:                         2.84
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Make sure that the output for the ϕ parameter and μ is as you'd expect. You can use the `.params` attribute to check these values.

In [9]:
```
# Print out the estimate for the constant and for theta
print(res_arma.params)
```

```
const     4.966442
ar.L1     0.647429
sigma2    4.132665
dtype: float64
```

Generate a first order MA model

Recall that the MA model has the following formula:

$$Y_t = \mu + \epsilon_t + \theta * \epsilon_{t-1}$$

This means that:

$$Y_1 = \mu + \epsilon_1 + \theta * \epsilon_0$$

$$Y_2 = \mu + \epsilon_2 + \theta * \epsilon_1$$

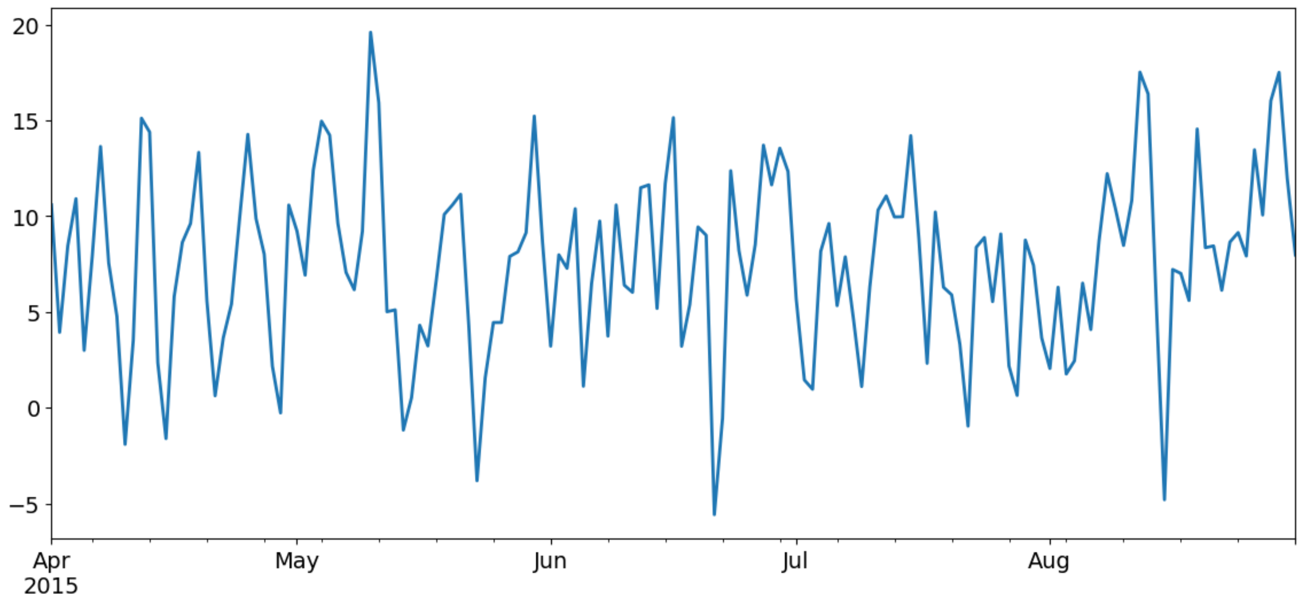
and so on.

Assume a mean-zero white noise with a standard deviation of 4. We'll also generate a daily datetime index ranging from April 2015 until the end of August 2015.

We will generate a first order MA model with $\theta = 0.9$ and $\mu = 7$.

In [10]:
```
np.random.seed(1234)

# Create a series with the specified dates
dates = pd.date_range('2015-04-01', '2015-08-31')

error = np.random.normal(0, 4, len(dates))
mu = 7
theta = 0.9

TS = [None] * len(dates)
error_prev = error[0]
for i, row in enumerate(dates):
    TS[i] = mu + theta * error_prev + error[i]
    error_prev = error[i]
```

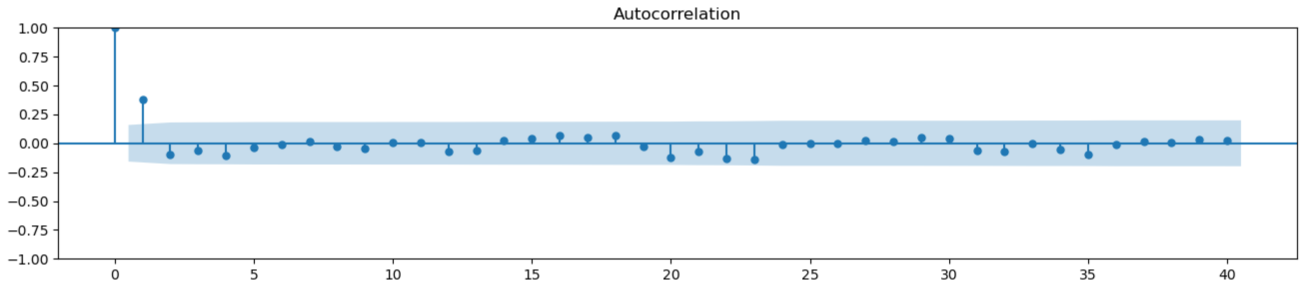
In [11]:
```
series = pd.Series(TS, index=dates)

series.plot(figsize=(14,6), linewidth=2, fontsize=14);
```

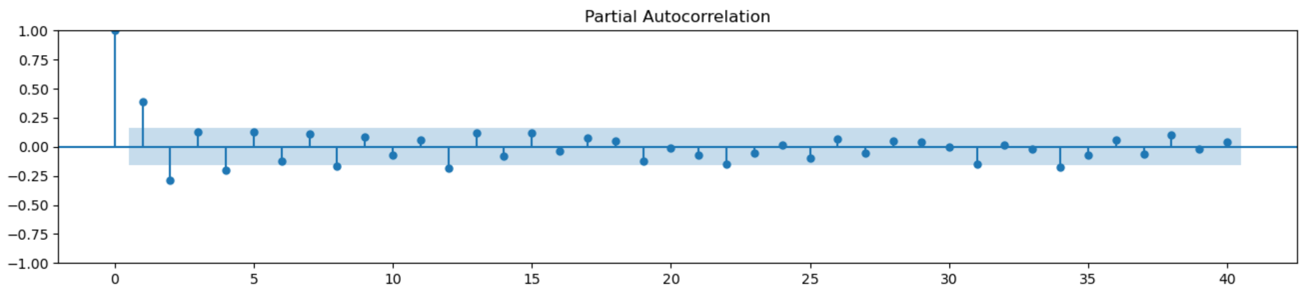


Look at the ACF and PACF of the model

In [12]:
```
fig, ax = plt.subplots(figsize=(16,3))
plot_acf(series, ax=ax, lags=40);

fig, ax = plt.subplots(figsize=(16,3))
plot_pacf(series, ax=ax, lags=40);
```

```
/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(
```





Check the model with ARMA in `statsmodels`

Let's fit an MA model to verify the parameters are estimated correctly. The first order MA model would be represented as `(0,1)` .

In [13]:
```
# Instantiate and fit an MA(1) model to the simulated data
mod_arma = ARIMA(series, order=(0,0,1))
res_arma = mod_arma.fit()

# Print out summary information on the fit
print(res_arma.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:                      y   No. Observations:                  153
Model:                 ARIMA(0, 0, 1)   Log Likelihood                -426.378
Date:                Sat, 19 Nov 2022   AIC                            858.757
Time:                        12:22:11   BIC                            867.848
Sample:                    04-01-2015   HQIC                           862.450
                         - 08-31-2015                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          7.5373      0.625     12.069      0.000       6.313       8.761
ma.L1          0.8727      0.047     18.489      0.000       0.780       0.965
sigma2        15.2765      1.592      9.597      0.000      12.156      18.397
===================================================================================
Ljung-Box (L1) (Q):                   5.72   Jarque-Bera (JB):                 9.11
Prob(Q):                              0.02   Prob(JB):                         0.01
Heteroskedasticity (H):               1.14   Skew:                            -0.48
Prob(H) (two-sided):                  0.65   Kurtosis:                         3.70
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [14]:
```
# Print out the estimate for the constant and for theta
print(res_arma.params)
```

```
const      7.537262
ma.L1      0.872686
sigma2    15.276484
dtype: float64
```

Summary

Great job! In this lesson, you saw how you can use the AR and MA models using the `ARIMA()` function from `statsmodels` by specifying the order in the form of `(p,q)` , where at least one of `p` or `q` was zero depending on the kind of model fit. You can use `ARIMA()` to fit a combined ARMA model as well -- which you will do in the next lab!