index

April 28, 2022

1 Basic Time Series Models - Lab

1.1 Introduction

Now that you have some basic understanding of the white noise and random walk models, its time for you to implement them!

1.2 Objectives

In this lab you will:

- Generate and analyze a white noise model
- Generate and analyze a random walk model
- Implement differencing in a random walk model

1.3 A White Noise model

To get a good sense of how a model works, it is always a good idea to generate a process. Let's consider the following example: - Every day in August, September, and October of 2018, Nina takes the subway to work. Let's ignore weekends for now and assume that Nina works every day - We know that on average, it takes her 25 minutes, and the standard deviation is 4 minutes
- Create and visualize a time series that reflects this information

Let's import `pandas`, `numpy`, and `matplotlib.pyplot` using their standard alias.

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

# Do not change this seed
np.random.seed(12)
```

Create the dates. You can do this using the `date_range()` function Pandas. More info here.

[27]:
```
# Create dates
dates = pd.date_range(start = "08/01/2018", end = "10/31/2018", freq = "B")
len(dates)
```

[27]:
```
66
```

Generate the values for the white noise process representing Nina's commute in August and September.

[28]:
```
# Generate values for white noise
commute = np.random.normal(loc = 25, scale = 4, size = len(dates))
```

Create a time series with the dates and the commute times.

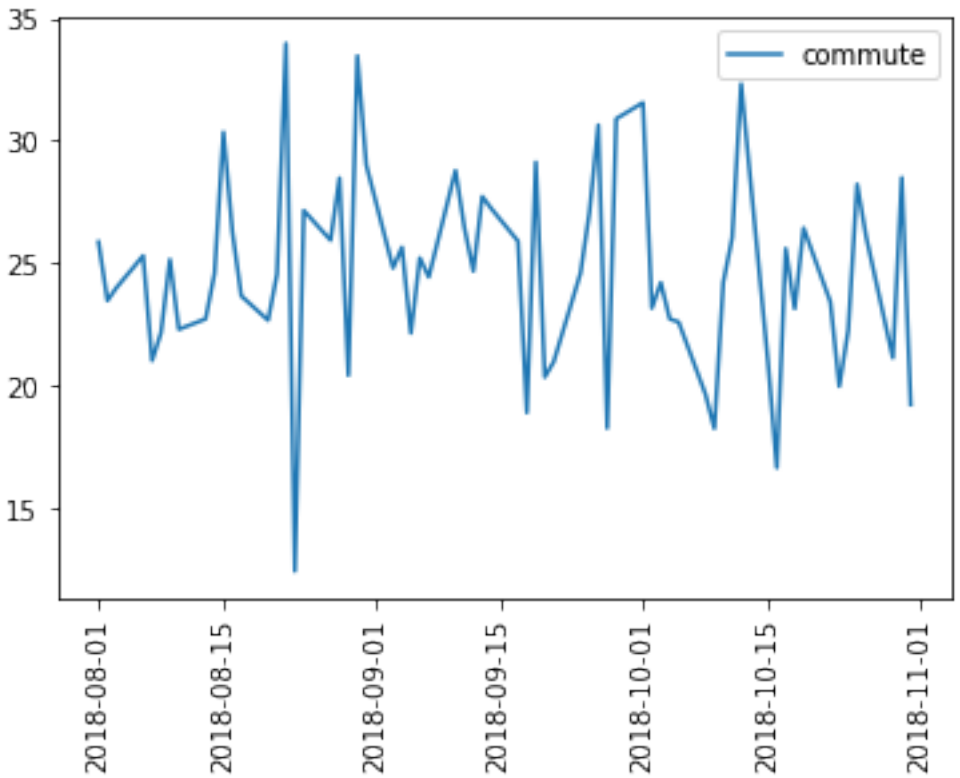
[29]:
```
# Create a time series
commute_series = pd.DataFrame(commute, columns = ["commute"],index = dates)
commute_series.head()
```

[29]:

	commute
2018-08-01	25.859904
2018-08-02	23.462565
2018-08-03	23.984384
2018-08-06	25.293008
2018-08-07	21.011185

Visualize the time series and set appropriate axis labels.

[30]:
```
# Visualize the time series
sns.lineplot(data = commute_series);
plt.xticks(rotation = 90);
```



Print Nina's shortest and longest commute.

[31]:
```
# Shortest commute
shortest_commute = commute_series.commute.min()
shortest_commute
```

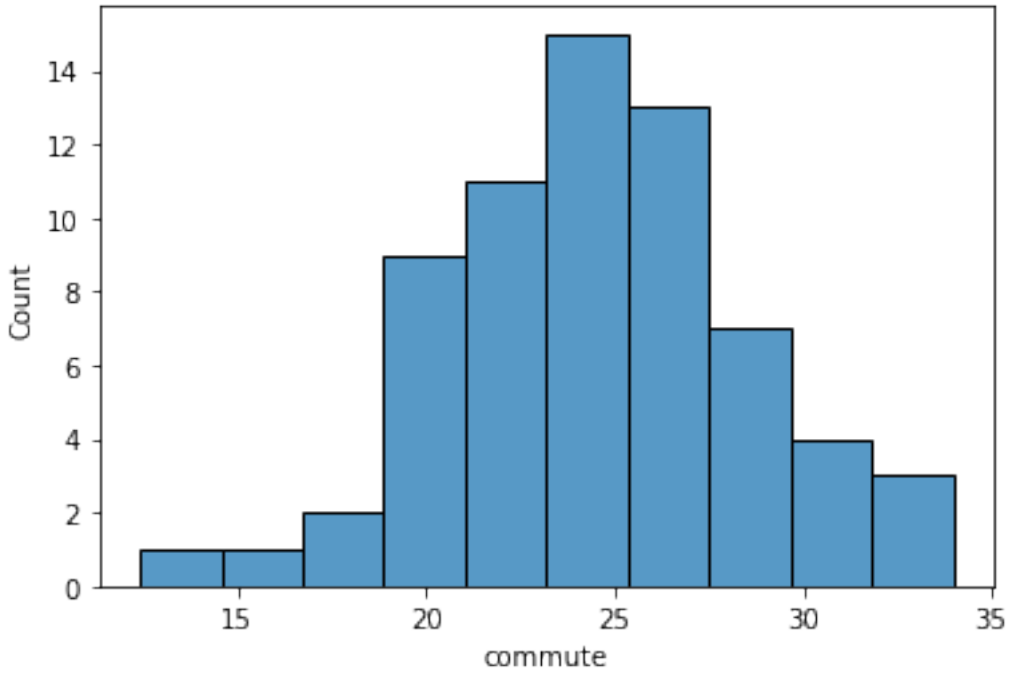
[31]:
```
12.41033391382408
```

[32]:
```
# Longest commute
longest_commute = commute_series.commute.max()
longest_commute
```

[32]:
```
33.96727117758093
```

Look at the distribution of commute times.

[34]:
```
# Distribution of commute times
sns.histplot(data = commute_series.commute);
```



Compute the mean and standard deviation of `commute_series`. The fact that the mean and standard error are constant over time is crucial!

[35]:
```
# Mean of commute_series
commute_mean = commute_series.commute.mean()
commute_mean
```

[35]: 24.612979372186242

[36]:
```
# Standard deviation of commute_series
commute_std = commute_series.commute.std()
commute_std
```

[36]: 4.053980257899386

Now, let's look at the mean and standard deviation for August and October.

[47]:
```
# Mean and standard deviation for August and October

august_mean = commute_series["08-01-2018":"09-01-2018"]["commute"].mean()
august_std = commute_series["08-01-2018":"09-01-2018"]["commute"].std()

# august_mean = commute_series["08-2018"]["commute"].mean()
# august_std = commute_series["08-2018"]["commute"].std()

october_mean = commute_series["10-01-2018":]["commute"].mean()
october_std = commute_series["10-01-2018":]["commute"].std()


print("August:","\t", august_mean, "\t", august_std)
print("October:","\t", october_mean, "\t", october_std)
```

```
August:         24.988184282361516      4.491470224782213
October:        23.74539552332911       3.9937140705104497
```

Because you've generated this data, you know that the mean and standard deviation will be the same over time. However, comparing mean and standard deviation over time is useful practice for real data examples to check if a process is white noise!

1.4 A Random Walk model

Recall that a random walk model has:

- No specified mean or variance
- A strong dependence over time

Mathematically, this can be written as:

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

Because today's value depends on yesterday's, you need a starting value when you start off your time series. In practice, this is what the first few time series values look like:

$$Y_0 = \text{some specified starting value}$$

$$Y_1 = Y_0 + \epsilon_1$$

$$Y_2 = Y_1 + \epsilon_2 = Y_0 + \epsilon_1 + \epsilon_2$$

$$Y_3 = Y_2 + \epsilon_3 = Y_0 + \epsilon_1 + \epsilon_2 + \epsilon_3$$

...

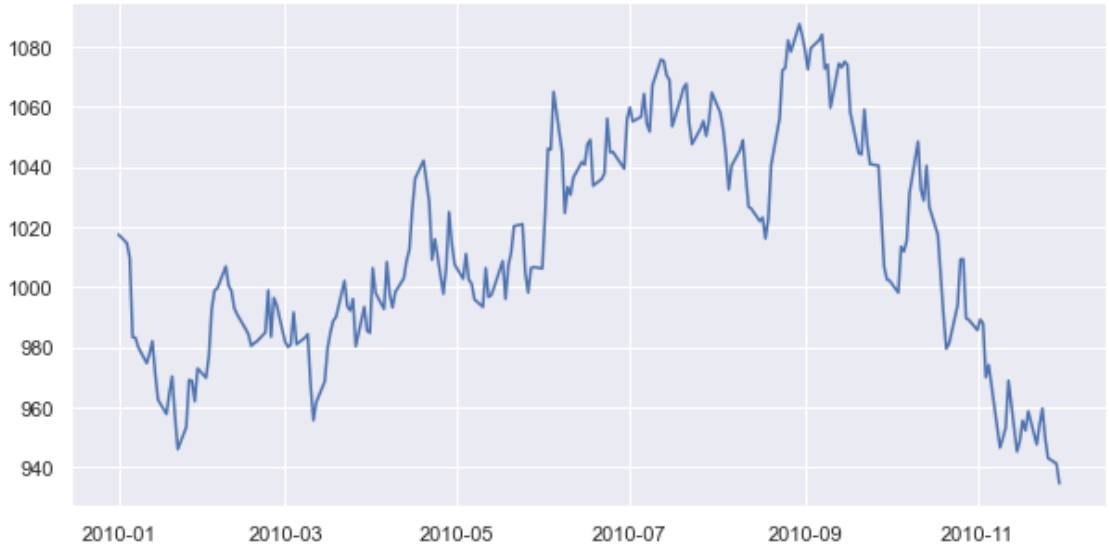
Keeping this in mind, let's create a random walk model:

Starting from a value of 1000 USD of a share value upon a company's first IPO (initial public offering) on January 1 2010 until end of November of the same year, generate a random walk model with a white noise error term, which has a standard deviation of 10.

[48]:
```
# Keep the random seed
np.random.seed(11)

# Create a series with the specified dates
dates = pd.date_range(start = "01/01/2010", end = "11/30/2010" , freq = "B")

# White noise error term
error = np.random.normal(loc = 0, scale = 10, size = len(dates))

# Define random walk
def random_walk(start, error):
    t_0 = start
    t = t_0 + np.cumsum(error)
    return t

shares_value = random_walk(1000, error)

shares_series = pd.Series(shares_value, index=dates)
```

Visualize the time series with correct axis labels.

[57]:
```
# Your code here
sns.set(rc={"figure.figsize":(10, 5)}) #width=3, #height=4
sns.lineplot(data = shares_series);


## G

# ax = shares_series.plot(figsize=(14,6))
# ax.set_ylabel('Stock value', fontsize=16)
# ax.set_xlabel('Date', fontsize=16)
# plt.show()
```



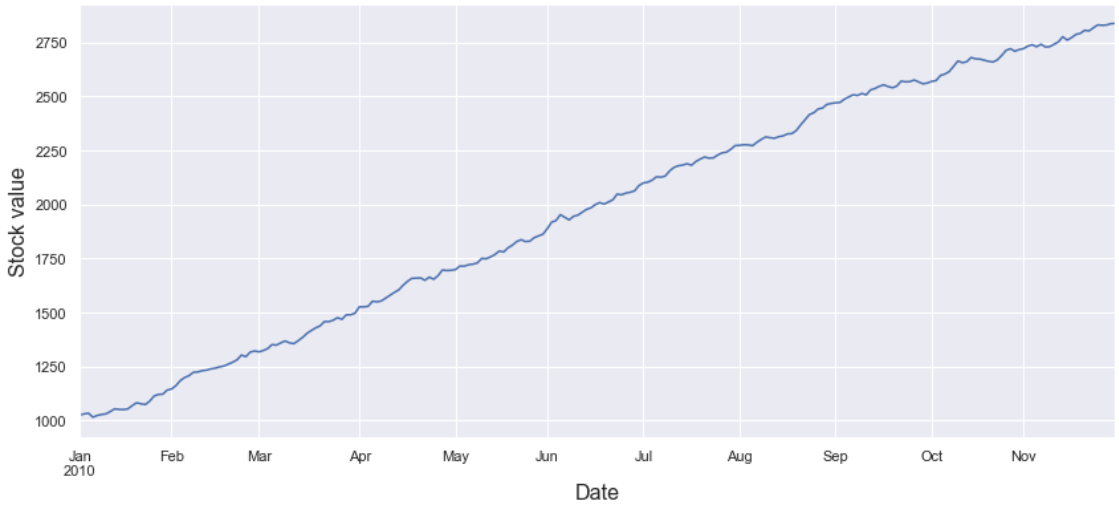
You can see how this very much looks like the exchange rate series you looked at in the lesson!

1.5 Random Walk with a Drift

Repeat the above, but include a drift parameter c of 8 now!

[59]:
```
# Keep the random seed
np.random.seed(11)

error_c = np.random.normal(loc = 8, scale = 10, size = len(dates))

# Define random walk
def random_walk_c(start, error):
    t_0 = start
    t = t_0 + np.cumsum(error_c)
    return t

shares_value_c = random_walk_c(1000, error_c)

shares_series_drift = pd.Series(shares_value_c, index=dates)
```

[60]:
```
ax = shares_series_drift.plot(figsize=(14,6))
ax.set_ylabel('Stock value', fontsize=16)
ax.set_xlabel('Date', fontsize=16)
plt.show()
```



Note that there is a very strong drift here!

1.6 Differencing in a Random Walk model

One important property of the random walk model is that a differenced random walk returns a white noise. This is a result of the mathematical formula:

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

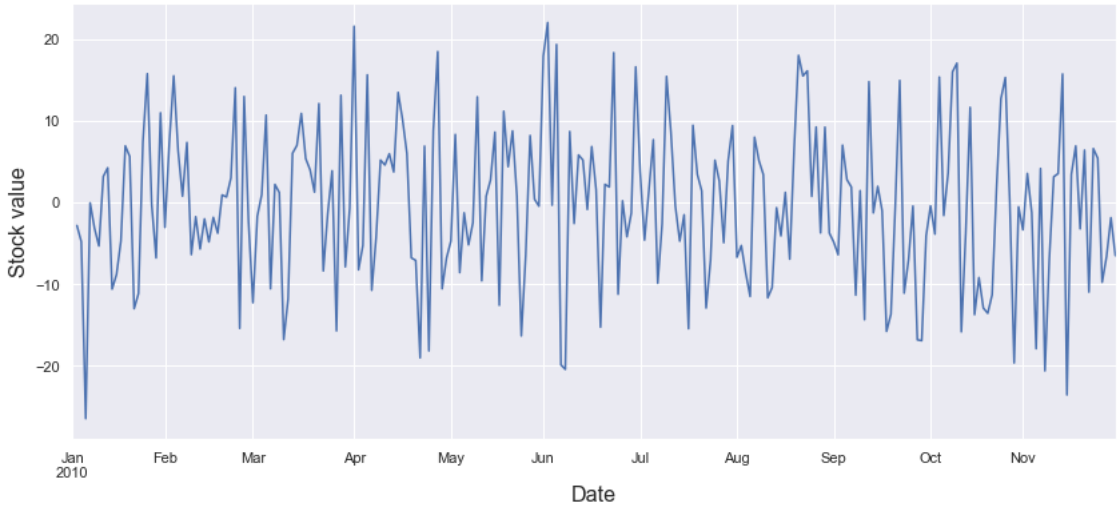
which is equivalent to

$$Y_t - Y_{t-1} = \epsilon_t$$

and we know that ϵ_t is a mean-zero white noise process!

Plot the differenced time series (time period of 1) for the shares time series (no drift).

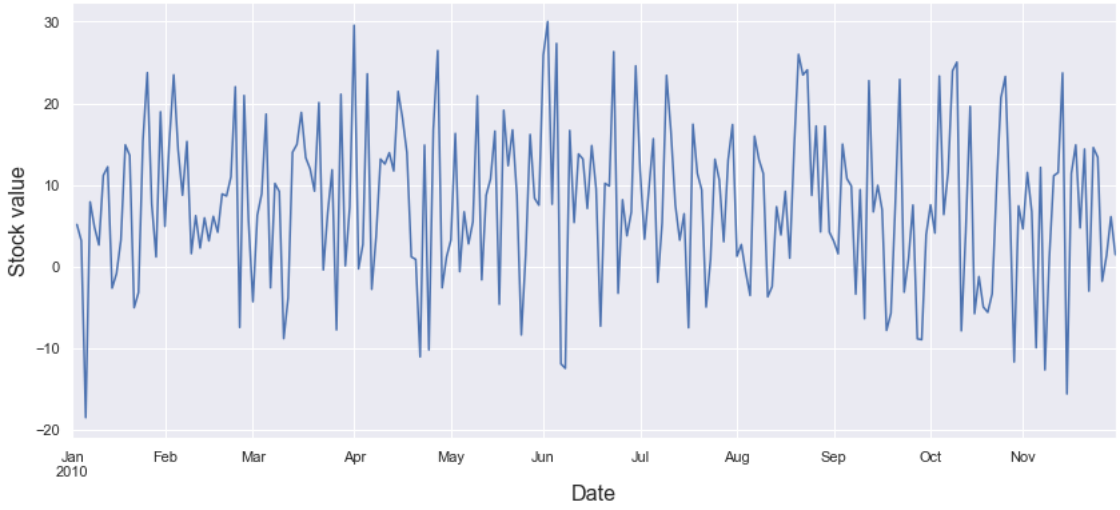
[63]:
```
# Your code here
shares_series_diff = shares_series.diff(periods = 1)

ax = shares_series_diff.plot(figsize=(14,6))
ax.set_ylabel('Stock value', fontsize=16)
ax.set_xlabel('Date', fontsize=16)
plt.show()
```



This does look a lot like a white noise series!

Plot the differenced time series for the shares time series (with a drift).

[64]:
```
# Your code here
shares_series_drift_diff = shares_series_drift.diff(periods = 1)

ax = shares_series_drift_diff.plot(figsize=(14,6))
ax.set_ylabel('Stock value', fontsize=16)
ax.set_xlabel('Date', fontsize=16)
plt.show()
```



This is also a white noise series, but what can you tell about the mean?

The mean is equal to the drift c, so 8 for this example!

1.7 Summary

Great, you now know how white noise and random walk models work!