DATA SOCIETY®

Time series day 2

“One should look for what is and not what he thinks should be.”
-Albert Einstein.

Module completion checklist

Objective	Complete
Recall key features of time series	
Integrate the concepts and implications of a white noise series	
Asses the forecasted models based on comparison criteria	
Explain the concept of a random walk model and stationarity	
Predict using random walk model and measure error	

DATA SOCIETY © 2019

Loading packages

- Load the packages we will be using

```
import os
import pickle
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from random import gauss
from random import seed
from random import random
from pandas import Series
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.stats.diagnostic import acorr_ljungbox
```

Working directory

- Set working directory to **`data_dir`**

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

Load the dataset

- We cleaned and then pickled the dataset in our last class, so we do not have to do it today
- We will now load the pickled dataset and save it as **`passenger_miles`**

```
passenger_miles = pickle.load(open("passenger_miles.sav","rb"))
```

```
print(passenger_miles.head())
```

```
            revenue_passenger_miles  available_seat_miles  unused_seat_miles
date
1979-01-01                    15.50                 26.64              11.15
1979-02-01                    16.58                 27.20              10.62
1979-03-01                    18.85                 27.87               9.02
1979-04-01                    17.23                 23.22               5.99
1979-05-01                    16.04                 23.27               7.23
```

Data: use case in civil aviation

- Air passenger mobility can be studied by analyzing seat and passenger miles
- We are working with **passenger mile data** that contains time series of three variables **from January 1992 to April 2002**:
  - `Revenue passenger mile` is a measure of the volume of air passenger transportation; a revenue passenger-mile is equal to one paying passenger carried one mile
  - `Available seat mile` is the number of seats multiplied by the distance traveled per flight and can represent the overall capacity of the aircraft
  - `Unused mile` is the difference between the above two variables and is used to measure the airline capacity utilization

Understand the data

- Let's take a look at the data we just loaded

```
# Dataframe information.
passenger_miles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 280 entries, 1979-01-01 to 2002-04-01
Data columns (total 3 columns):
revenue_passenger_miles    280 non-null float64
available_seat_miles       280 non-null float64
unused_seat_miles          280 non-null float64
dtypes: float64(3)
memory usage: 8.8 KB
```

```
# Shape of the data
passenger_miles.shape
```

Recap: key features

- In our first time series class, we reviewed some core concepts including key features of time series:
  - Observations arrive according to some **order**: *starting on Jan 01, 1979 and ending on Apr 01, 2002*
  - Measurements are made in discrete **equally spaced time intervals**: *every month*

date	revenue_passenger_miles	available_seat_miles	unused_seat_miles
1979-01-01	15.50	26.64	11.15
1979-02-01	16.58	27.20	10.62
1979-03-01	18.85	27.87	9.02
1979-04-01	17.23	23.22	5.99

…

	revenue_passenger_miles	available_seat_miles	unused_seat_miles
2002-02-01	36.01	53.40	17.39
2002-03-01	41.21	54.85	13.64
2002-04-01	39.50	55.54	16.04

Recap: key features (cont'd)

- Measurements can be **discrete** or **continuous**: *passenger miles are assumed as* **continuous**

```
print(passenger_miles.head(20))
```

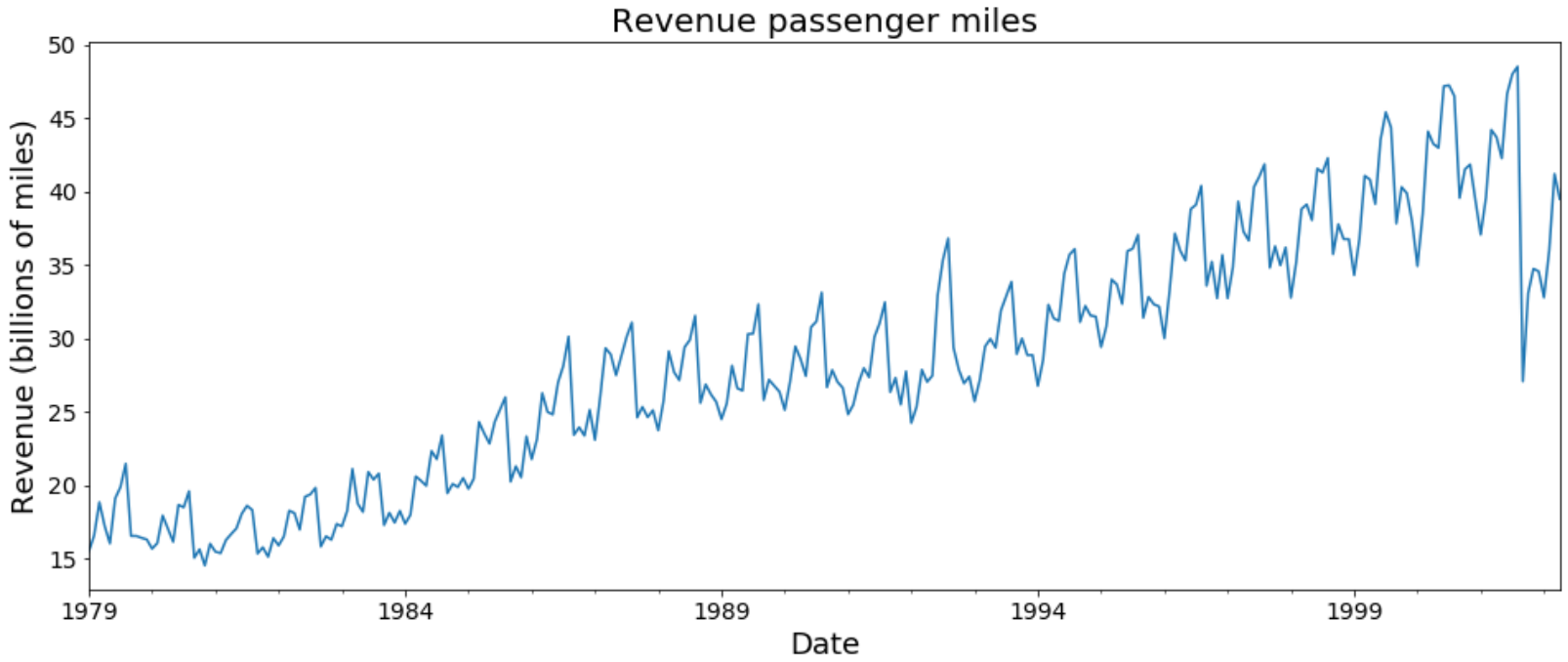
```
            revenue_passenger_miles  available_seat_miles  unused_seat_miles
date
1979-01-01                    15.50                 26.64              11.15
1979-02-01                    16.58                 27.20              10.62
1979-03-01                    18.85                 27.87               9.02
1979-04-01                    17.23                 23.22               5.99
1979-05-01                    16.04                 23.27               7.23
1979-06-01                    19.11                 27.30               8.19
1979-07-01                    19.88                 29.28               9.40
1979-08-01                    21.47                 31.23               9.76
1979-09-01                    16.55                 28.55              12.00
1979-10-01                    16.55                 27.84              11.28
1979-11-01                    16.42                 27.56              11.14
1979-12-01                    16.31                 28.87              12.56
1980-01-01                    15.68                 28.70              13.02
1980-02-01                    16.07                 28.38              12.31
1980-03-01                    17.94                 28.66              10.72
1980-04-01                    17.05                 28.76              11.71
1980-05-01                    16.15                 27.96              11.82
1980-06-01                    18.67                 29.92              11.25
1980-07-01                    18.50                 30.24              11.74
1980-08-01                    19.60                 30.21              10.61
```

Recap: key features (cont'd)

- Observations are **not necessarily independent**
- They **may be correlated** and the degree of correlation may depend on their positions in the sequence
  - Closer ones might be more related to each other!
- Only **stable series can be analyzed**

Recap: visualize time series

```
fig, ax = plt.subplots(figsize = (16, 6))
passenger_miles['revenue_passenger_miles'].plot()
plt.title('Revenue passenger miles', fontsize = 20)
plt.xlabel('Date', fontsize = 18)
plt.ylabel('Revenue (billions of miles)', fontsize = 18)
ax.tick_params(labelsize = 14)
plt.show()
```

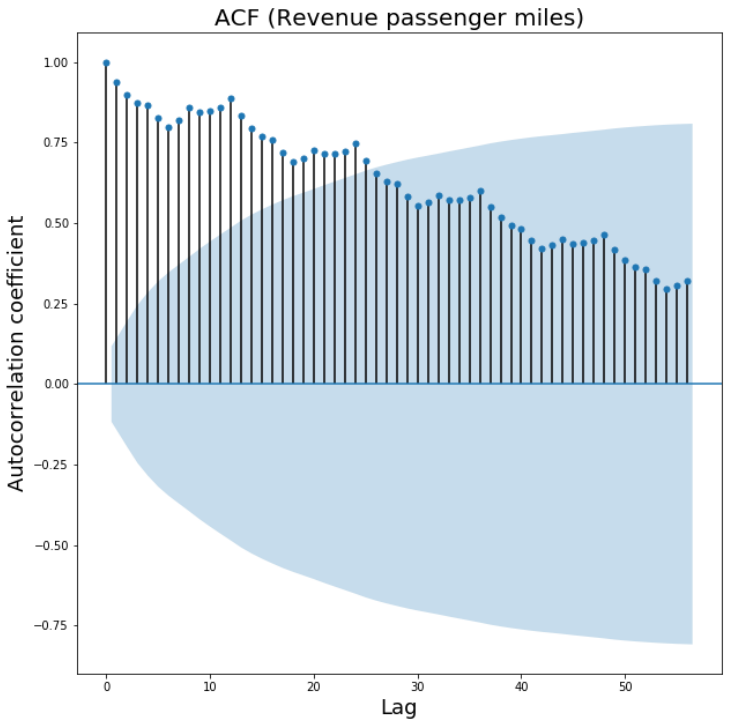


Recap: interpret autocorrelation function

Interpretation:

- The lag value k is displayed on the `x-axis`
- The autocorrelation coefficient r_k is displayed on the `y-axis` (notice it is a value between $[-1, 1]$)
  - Vertical lines and points help us get a better perception of the magnitude of the autocorrelation
- The **shaded area represents confidence intervals** (95% by default), which suggest that correlation values outside of this cone are very likely to occur in real life and are not just a statistical artifact
- This plot is sometimes called a *correlogram* or an *autocorrelation plot*

```
56
```



Module completion checklist

Objective	Complete
Recall key features of time series	✔
Integrate the concepts and implications of a white noise series	
Asses the forecasted models based on comparison criteria	
Explain the concept of a random walk model and stationarity	
Predict using random walk model and measure error	

DATA SOCIETY © 2019

Back to the noise

- If we re-define our **conditional mean** through trend and seasonal components, we get the following:

$$E(Y_t|H_{t-1}) = \mu_t = T_t + S_t$$

- This quantity is our **forecast** for Y_t at time $t-1$
- The error term ϵ_t, which is also known as the **noise** term is a simple difference between the **forecast** and the actual value of Y_t

$$\epsilon_t = Y_t - \mu_t = Y_t - (T_t + S_t) = Y_t - T_t - S_t$$

- When we remove the trend and the seasonality components of the time series, all we have left is the error $Y_t = \epsilon_t$

But what is white noise then?

- Today, we are going to introduce one of the most simplistic models, a **white noise** model
- **A time series may be white noise if:**
  - The variables are independent and identically distributed with a mean of zero
  - All variables have the same variance (σ^2) and each value has a zero correlation with all other values in the series
- Like we learned yesterday, the goal of all time series analysis is to reduce the series to white noise
- **Sometimes, the series will start off as white noise - how to determine that is what we will cover now**

Why do we want white noise?

- White noise is an important concept in time series analysis and forecasting.
- It is important for two main reasons:
  - **Predictability:** if your time series is white noise, then, by definition, it is random
  - **Model diagnostics:** the series of errors from a time series forecast model should ideally be white noise
- Time series data are expected to contain some white noise on top of the signal generated by the underlying process

When forecast errors are white noise

- When forecast errors are white noise, it means that all of the signal information in the time series has been harnessed by the model in order to make predictions
- All that is left is the random fluctuations that cannot be modeled
- **A sign that model predictions are not white noise is an indication that further improvements to the forecast model may be possible**

For example:

- `y(t) = signal(t) + noise(t)`
  - Once predictions have been made by a time series forecast model, they can be collected and analyzed
  - **The series of forecast errors should ideally be white noise**

When a series is white noise

- What do we do once we have a white noise times series?
- First, we ensure that the time series is **white noise** by validating objectively
- We can do this a couple ways:
  - Visualizing the time series and looking to see if it looks random
  - Looking at the distribution of the series and checking if it is normal
  - Looking at the autocorrelation plot of the series and looking to see if all autocorrelations are 0
  - Running a **Ljung-Box test** to validate that all autocorrelations are 0 at a 95% confidence interval

Create a white noise series

- We want to understand what exactly a white noise series looks like, so let's create a randomized series and then check and validate if it is in fact white noise
- To do this, we:

1. Create a list of 1,000 random Gaussian variables
2. Draw variables from a Gaussian distribution with a mean of 0 and standard deviation of 1

```
# Seed random number generator.
seed(1)
# Create white noise series.
series = [gauss(0.0, 1.0) for i in range(1000)]
series = Series(series)
print(series.head())
```

```
0    1.288185
1    1.449446
2    0.066336
3   -0.764544
4   -1.092173
dtype: float64
```

Summary statistics of a white noise series

- Now we calculate and print some of the summary statistics of the newly created series
- We look at mean and standard deviation
- We know that for a white noise series, mean will be 0 and standard deviation will be 1

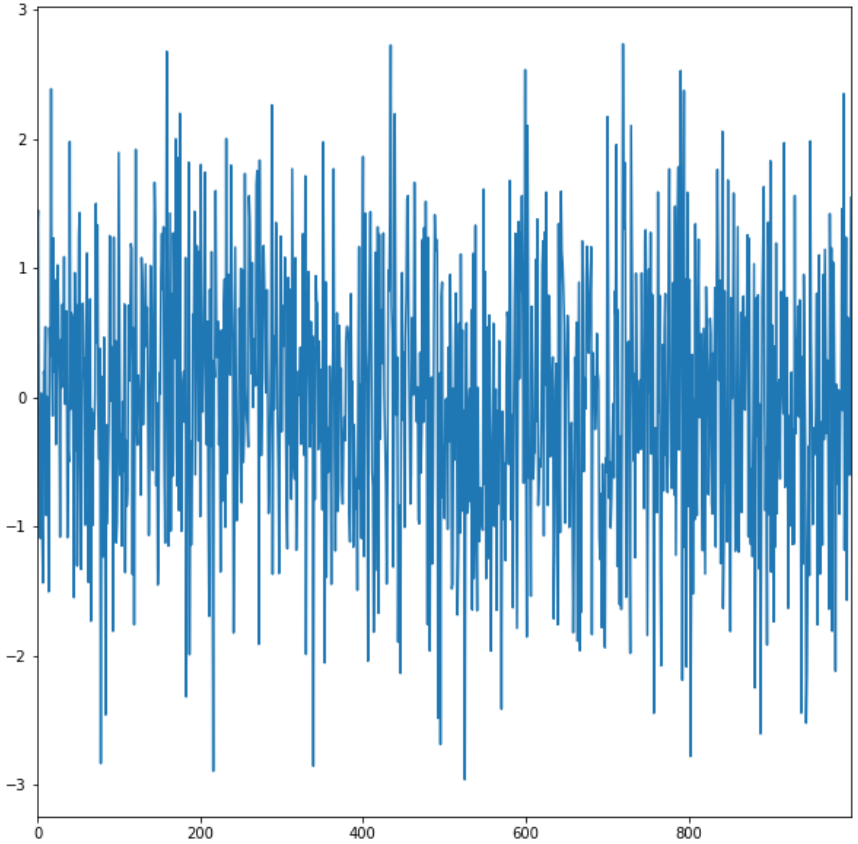
```
# Summary statistics.
print(series.describe())
```

```
count    1000.000000
mean       -0.013222
std         1.003685
min        -2.961214
25%        -0.684192
50%        -0.010934
75%         0.703915
max         2.737260
dtype: float64
```

Visualize the white noise series

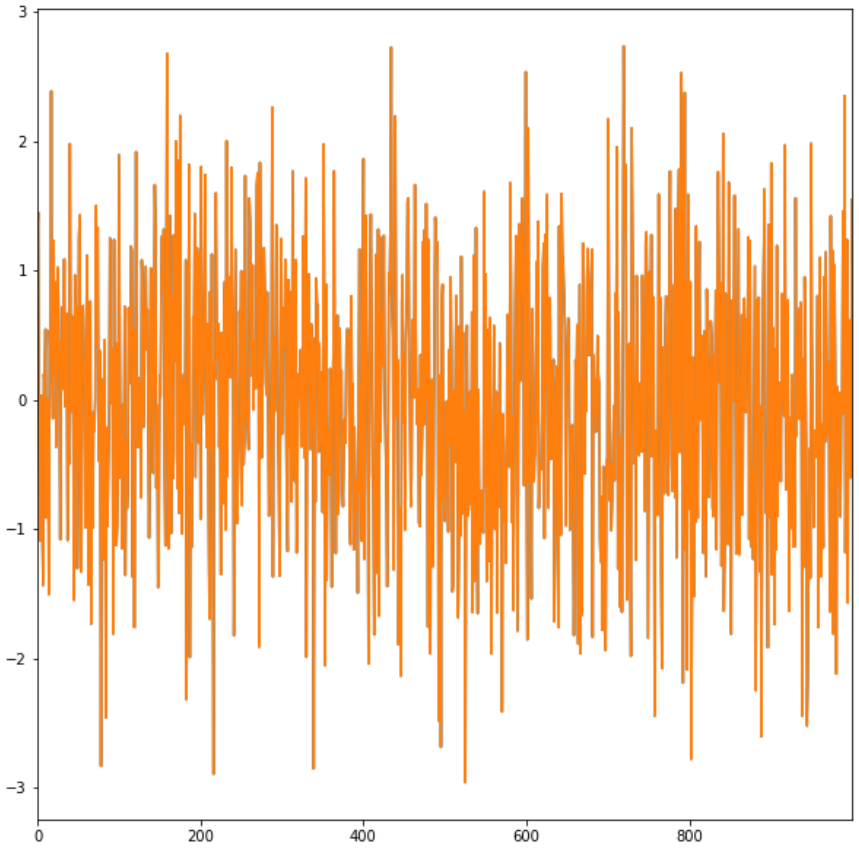
- Now, let's visualize the series
- This is our first check if the series may be white noise or not

```
# Let's create a lineplot of the series.
series.plot()
plt.show()
```

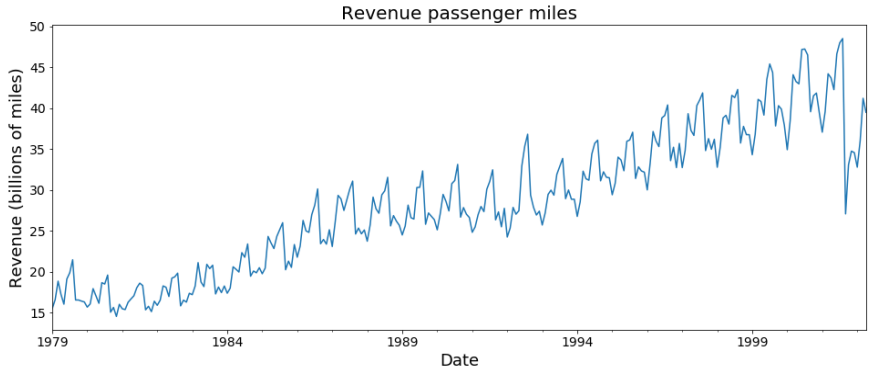


Visualize: white noise vs. not white noise

- To compare - we look at the potential white noise time series:



- And a time series that is definitely not white noise:



Autocorrelation of white noise series

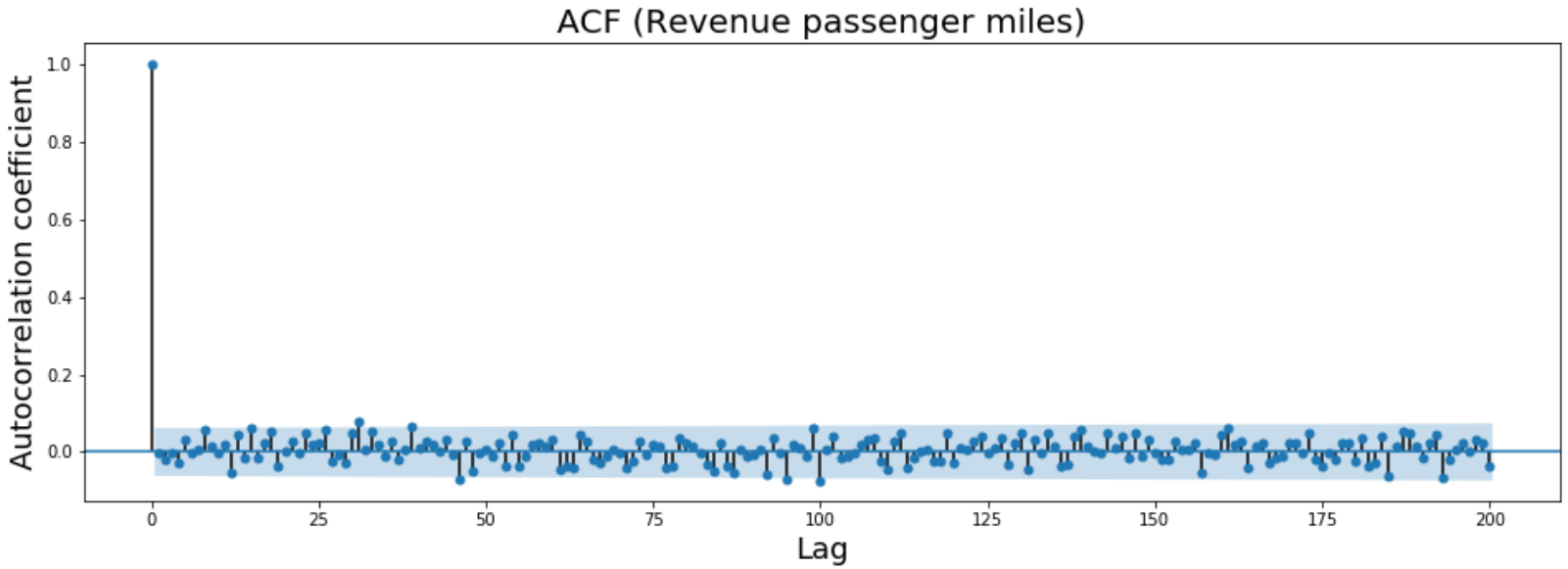
- Now that we have visualized the series, we look at the autocorrelation at all lags
- To be white noise, the autocorrelation should fall at 0 (within a 95% confidence interval)
- Let's find `max_k` as we did in our first class
- Remember this is based on the rule of thumb for an optimal value of k is about $1/5$ of the total number of observations

```
# Let's get a maximum lag based on the rule of thumb.
max_k = series.shape[0]//5
print(max_k)
```

```
200
```

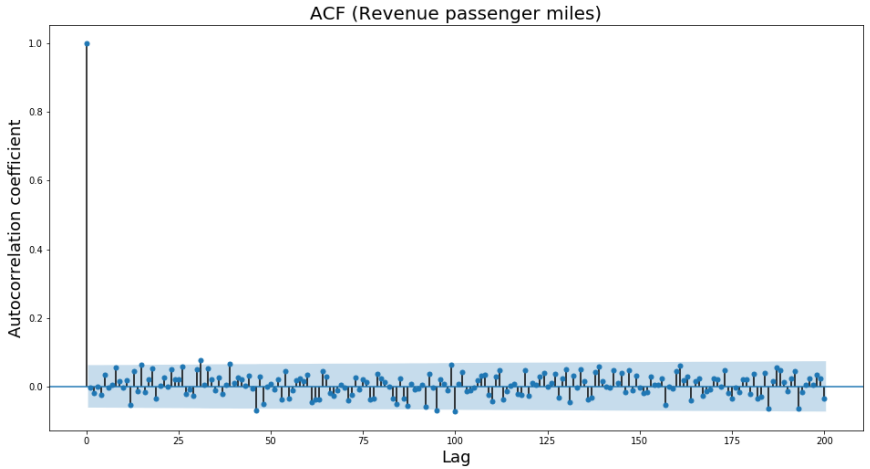
Autocorrelation of white noise series

```
# Autocorrelation.
fig, ax = plt.subplots(figsize = (16, 5))
plot_acf(series, lags = max_k) #<- adjust k
plt.title('ACF (Revenue passenger miles)', fontsize = 20)
plt.xlabel('Lag', fontsize = 18)
plt.ylabel('Autocorrelation coefficient', fontsize = 18)
ax.tick_params(labelsize = 14)
plt.show()
```

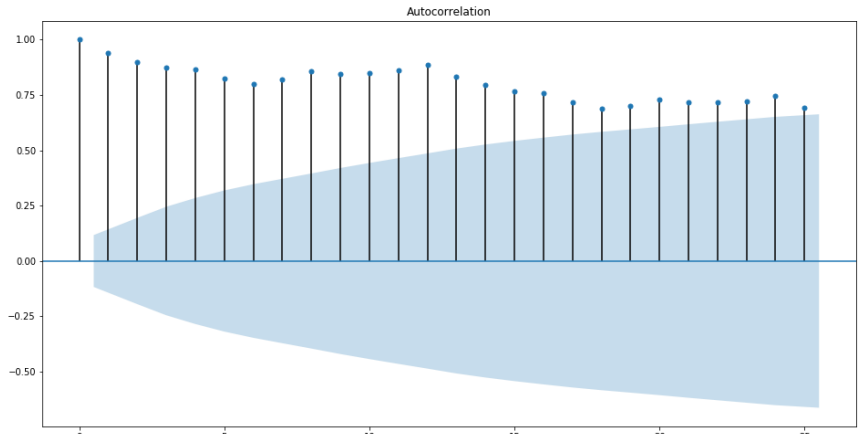


Autocorrelation: white noise vs. not white noise

- To compare, let's look at the autocorrelation in the potential white noise time series:



- Next to the autocorrelation of a time series that is definitely not white noise:



Introduce the Ljung-Box test

- Our final test for white noise is to use a specific test, the ***Ljung-Box test***
- It is a statistical test of whether any of a group of autocorrelations of a time series are different from zero
- Instead of testing randomness at each distinct lag, it tests "overall" randomness based on a number of lags
- The test states:

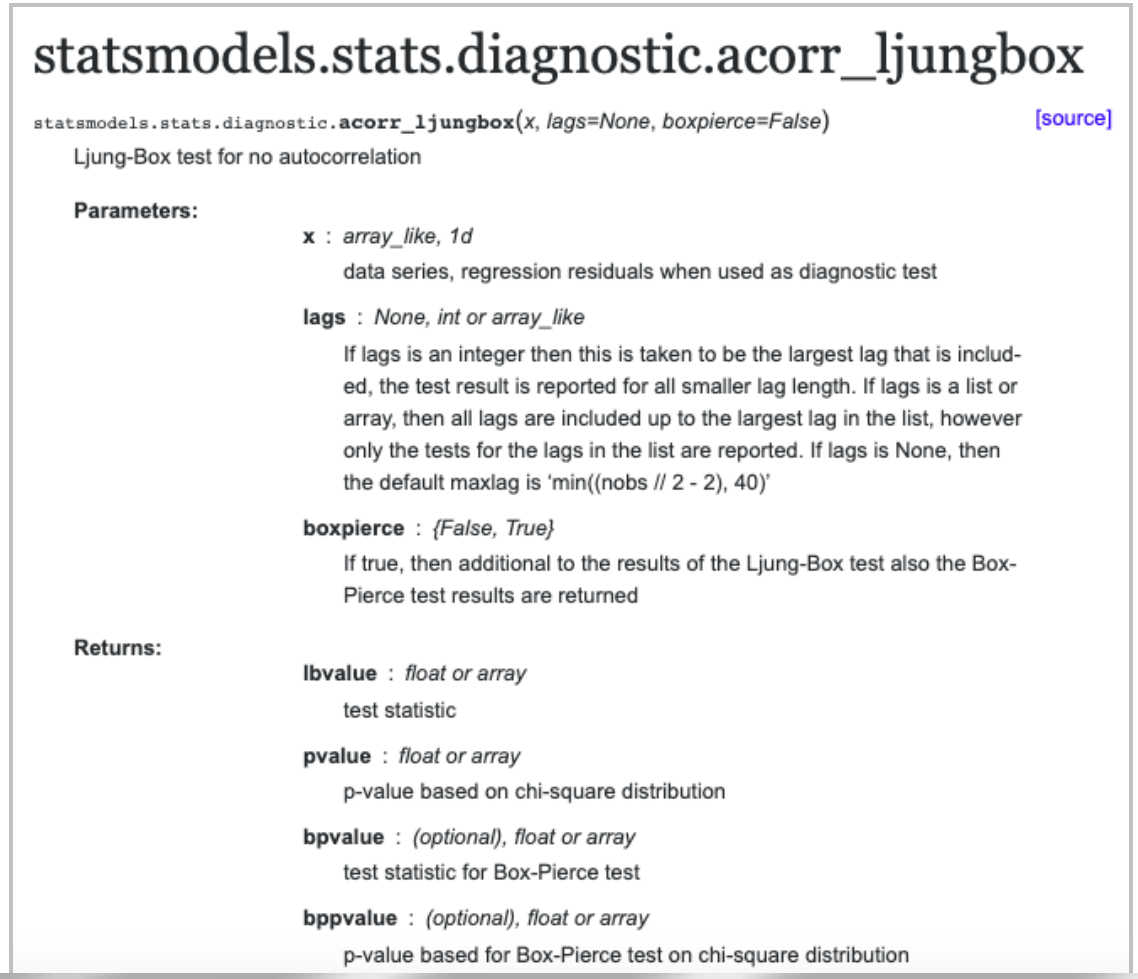
$$H_0 : \rho = 0$$
$$H_a : \rho \neq 0$$

- We can specify at which lag we think autocorrelation drops to 0 by changing the H_0 to reflect this:

$$H_0 : \rho_1 = \rho_2 = \ldots = \rho_k = 0$$
$$H_a : \text{at least one of the AC's is different than } 0$$

Ljung-Box test with statsmodel

- Now we will run the Ljung-Box test on our white noise series
- `statsmodels` has a package that we will use, `acorr_ljungbox`



statsmodels.stats.diagnostic.acorr_ljungbox

statsmodels.stats.diagnostic.**acorr_ljungbox**(*x*, *lags=None*, *boxpierce=False*) [source]

Ljung-Box test for no autocorrelation

Parameters:

x : *array_like, 1d*
data series, regression residuals when used as diagnostic test

lags : *None, int or array_like*
If lags is an integer then this is taken to be the largest lag that is included, the test result is reported for all smaller lag length. If lags is a list or array, then all lags are included up to the largest lag in the list, however only the tests for the lags in the list are reported. If lags is None, then the default maxlag is 'min((nobs // 2 - 2), 40)'

boxpierce : *{False, True}*
If true, then additional to the results of the Ljung-Box test also the Box-Pierce test results are returned

Returns:

lbvalue : *float or array*
test statistic

pvalue : *float or array*
p-value based on chi-square distribution

bpvalue : *(optional), float or array*
test statistic for Box-Pierce test

bppvalue : *(optional), float or array*
p-value based for Box-Pierce test on chi-square distribution

Implement Ljung-Box test on white noise

- Remember, **if we fail to reject the null, we are saying that the series is white noise**
- The output of the test is:
  - `lbvalue` : test statistic at each lag (first row)
  - `pvalue` : p-value based on chi-square distribution at each lag (second row)

```
wn_ljung_box_results = pd.DataFrame(acorr_ljungbox(series, lags = None, boxpierce = False))  # Lags is
the largest lag to  report
print(wn_ljung_box_results)
```

```
          0         1         2   ...         37         38         39
0  0.007156  0.437053  0.441174   ...  41.958073  46.518801  46.60247
1  0.932586  0.803702  0.931611   ...   0.303249   0.190380   0.21925

[2 rows x 40 columns]
```

- From the output of the test, we see that we **fail to reject the null that:**

$$H_0 : \rho_1 = \rho_2 = \ldots = \rho_k = 0$$

and the series **is white noise**

Implement Ljung-Box test on not white noise

- Let's say we were not sure still if our `revenue_passenger_miles` series is white noise
- We can run a Ljung-Box test on the series and check

```
no_wn_ljung_box_results = pd.DataFrame(acorr_ljungbox(passenger_miles['revenue_passenger_miles'],
lags=None,boxpierce=False))  # Lags is the largest lag to  report
print(no_wn_ljung_box_results)
```

```
              0             1   ...           38           39
0  2.492104e+02  4.780598e+02  ...  6197.855967  6274.65984
1  3.860032e-56  1.551106e-104  ...     0.000000     0.00000

[2 rows x 40 columns]
```

- We see here that the p-values are actually all significant at a 95% confidence interval
- Therefore, **we reject the null that:**

$$H_0 : \rho_1 = \rho_2 = \ldots = \rho_k = 0$$

and the series **is not white noise**

Final thoughts on white noise

- We now see the difference between a series that is actually white noise and one that isn't
- We also know multiple ways to validate if the series is white noise
- **But what do we do if we want to predict on a white noise series?**
  - This is possible using a **constant mean model**

Knowledge Check 1

Exercise 1

Module completion checklist

Objective	Complete
Recall key features of time series	✔
Integrate the concepts and implications of a white noise series	✔
Asses the forecasted models based on comparison criteria	
Explain the concept of a random walk model and stationarity	
Predict using random walk model and measure error	

DATA SOCIETY © 2019

Measuring forecast error

- Now that we have our predictions as well as our actual values, we can assess the **forecast error**
- There are multiple metrics that should be familiar before going on to the assessment
- The three metrics we will use are:
  - Mean absolute error (MAE)
  - Mean square error (MSE)
  - Mean absolute percentage error (MAPE)
- Where:
  - $Y_t = \text{actual value at time } t$
  - $F_t = \text{forecast value for time } t \text{at time} t - 1 (\text{one step forecast}) = \hat{Y}_{t-1}(1)$
  - $\epsilon_t = \text{forecast at error time} t = Y_t - F_t$
- We will now review each metric in detail

Mean absolute error (MAE)

- This has a clear interpretation as the average absolute difference between y_i and x_i
- Where y_i is the prediction and x_i is the true value
- It is the average of the absolute errors

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} | Y_t - F_t | = \frac{1}{n}\sum_{t=1}^{n} | \epsilon_t |$$

- We use MAE over the other two metrics when we want to lead towards forecasts of the median
- A forecast that minimizes the MAE will lead to forecasts of the median
- It is also the easiest to interpret compared to RMSE and MAPE

Root mean square error (RMSE)

- Root mean square error is not as clear of an interpretation as MAE, but is another commonly used metric
- It is based on the square error

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(Y_t - F_t)^2} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\epsilon_t^2}$$

- A forecast that minimizes the RMSE will lead to forecasts of the mean
- RMSE is more widely used, despite being more difficult to interpret

Mean absolute percentage error (MAPE)

- Percentage errors have the advantage of being unit-free
- They are frequently used to compare forecast performance between datasets
- Mean absolute percentage error (MAPE) is the most commonly used

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\frac{|Y_t - F_t|}{Y_t} = \frac{1}{n}\sum_{t=1}^{n}\epsilon_t$$

- They have the disadvantage of being infinite or undefined if $y_t = 0$ for any t in the period or of having extreme values if any y_t is close to zero

Knowledge Check 2

Module completion checklist

Objective	Complete
Recall key features of time series	✔
Integrate the concepts and implications of a white noise series	✔
Asses the forecasted models based on comparison criteria	✔
Explain the concept of a random walk model and stationarity	
Predict using random walk model and measure error	

DATA SOCIETY © 2019

Random walk model

- Now we know what **white noise** is and how to deal with it
- We also know when it is helpful to have white noise and when it means there is probably no predictive power to your time series
- **A random walk model is a naive forecast and is used often to forecast stock markets**
- Now, let's learn about the most simple but effective forecast, the **random walk**
- The main idea behind this forecast is to **use the most recent observed value of the series as your forecast for all future periods**

Random walk model - math

- Random walk is many things, however, it is NOT a sequence of random numbers!
- A random walk specifically is different from random numbers because the next value in the sequence is a modification of the previous value in the sequence
- To understand this model, we are going to use a sample dataset
- A simple model of random walk is as follows:

1. Start with a random number of either -1 or 1
2. Randomly select a -1 or 1 and add it to the observation from the previous time step
3. Repeat step 2 for as long as you like

Random walk model - math

- The random walk equation is:

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

Random walk model - create dataset

- Let's create a sample dataset to illustrate what a random walk looks like
- We will then test our **`passenger_miles`** dataset to see if it is suitable for a random walk model or not
- We are going to build a list of random numbers making sure all points are only one step ahead or behind of the previous point

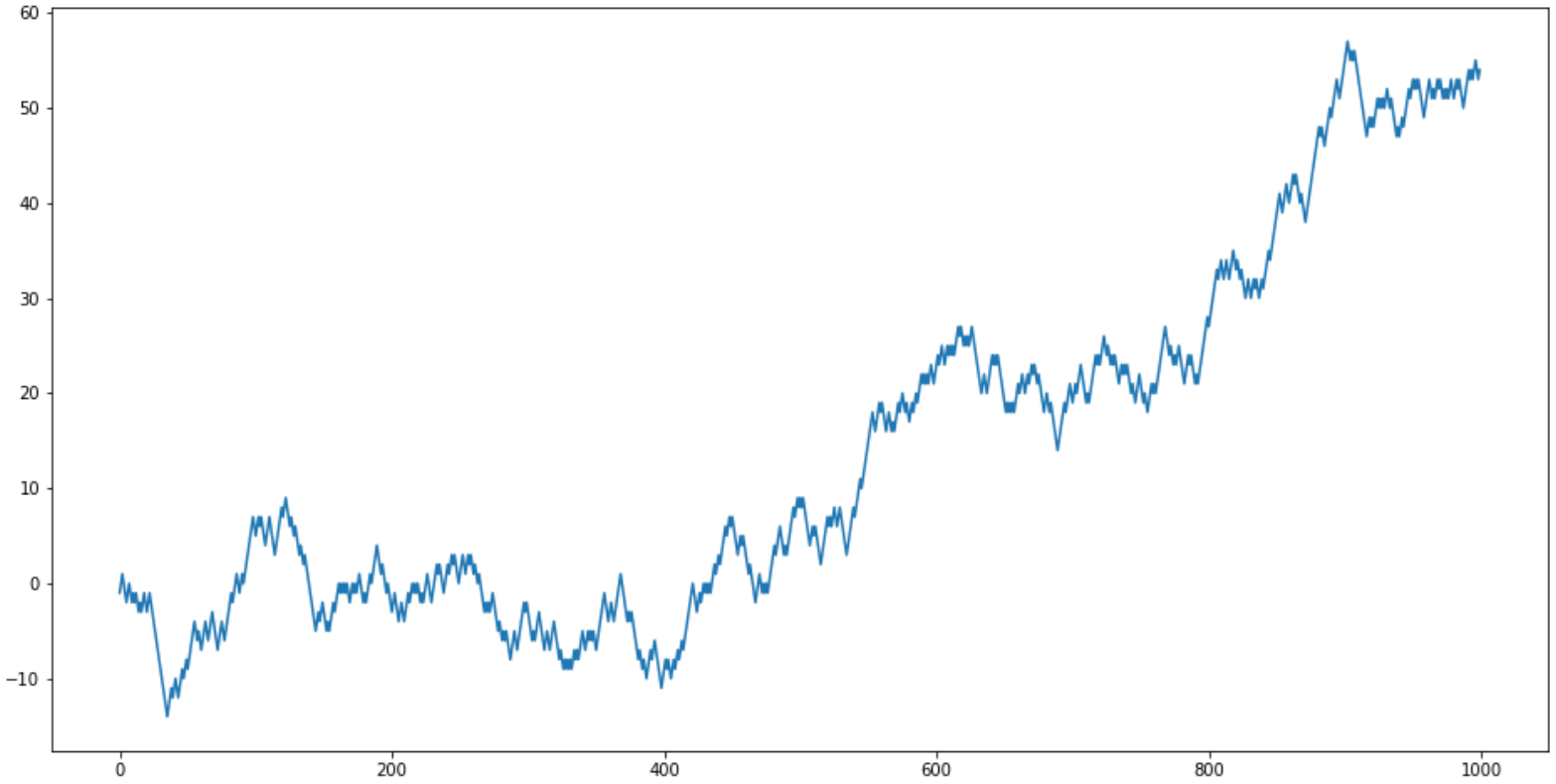
```
from random import seed, random
seed(1)
random_walk = list()
random_walk.append(-1 if random() < 0.5 else 1)
for i in range(1, 1000):
    movement = -1 if random() < 0.5 else 1
    value = random_walk[i-1] + movement
    random_walk.append(value)
print(random_walk[1:5])
```

```
[0, 1, 0, -1]
```

```
# Convert random walk into a df for convenience
random_walk_df = pd.DataFrame(random_walk)
```

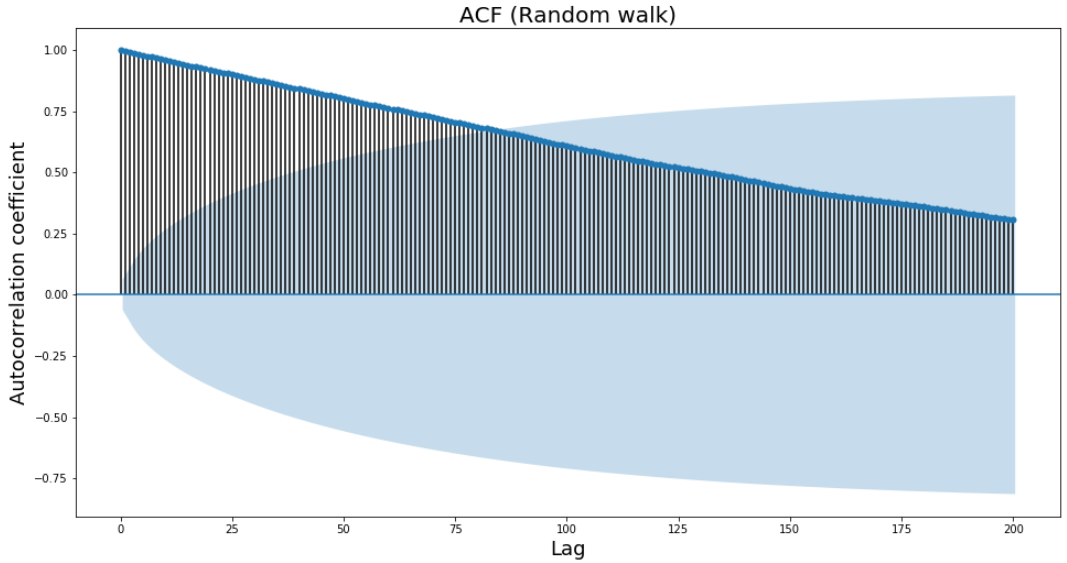
Random walk model - visualize

- This definitely looks different than our white noise series!
- There is a time component that we will have to address



Random walk model - autocorrelation

- White noise models must have all autocorrelations at 0 (within a 95% or otherwise specified confidence interval)
- Let's look at the autocorrelation plot for our random list
- The autocorrelation is definitely not 0 at all lags, but if this is a true random walk model, we should be able to get it there by doing one thing

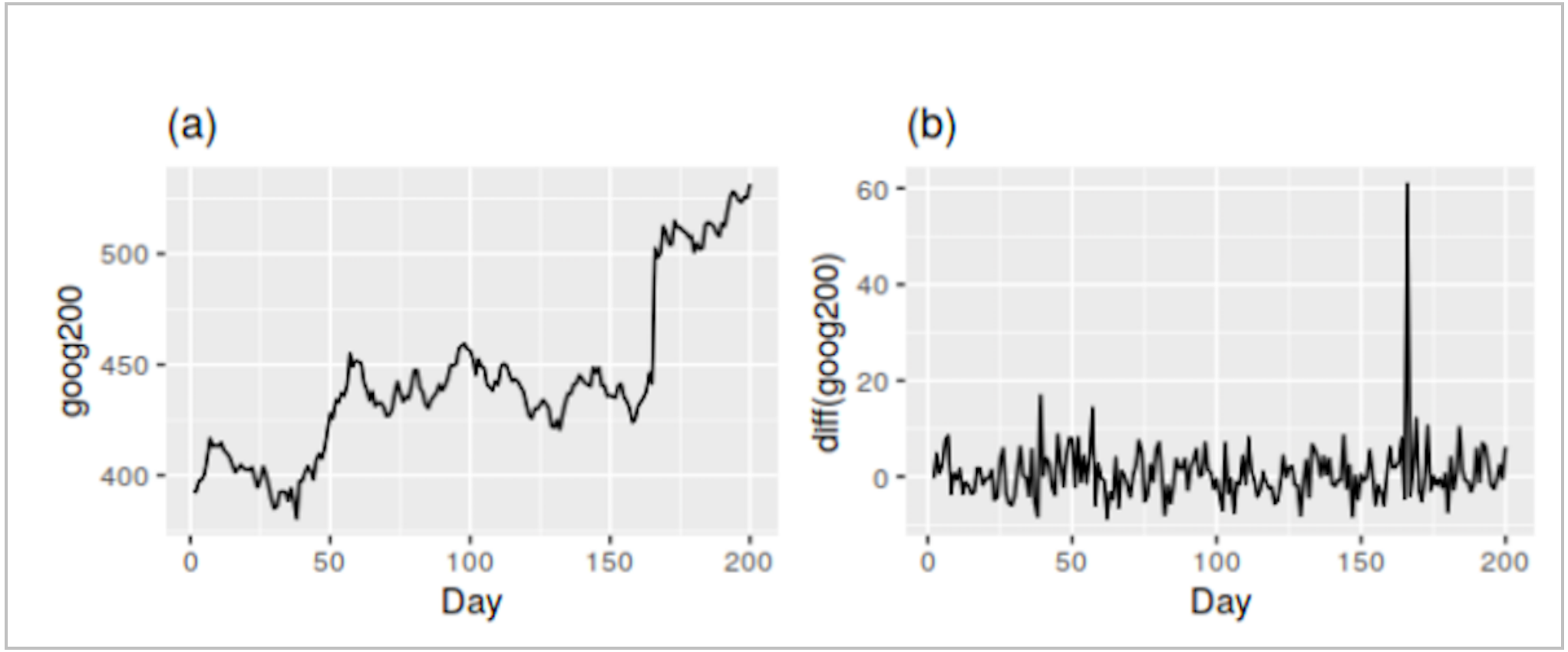


Random walk model - stationarity

- A stationary time series is one whose properties do not depend on the time at which the series is observed
- A **white noise series is stationary**, it does not matter when you observe it, it should look the same at any point in time
- A random walk model is a little more confusing
- This is because **in general, a stationary time series will have no predictable patterns in the long-term**
- Time plots will show the series to be roughly horizontal, with constant variance

Random walk model - stationarity

- Look at the two plots below - which one looks stationary?:



Random walk model - testing for stationarity

- We can usually visually confirm if our series is stationary or not
- However, we should make sure by testing our series using the ***Augmented Dickey-Fuller test (ADF)***
- The ADF test will test for stationarity by looking for a ***unit root***, where:

$$H_0 : \text{there is a unit root}$$
$$H_a : \text{the time series is stationary}$$

Random walk model - testing for stationarity

```
# Statistical test.
result = adfuller(random_walk)
print('ADF Statistic: %f' % result[0])
```

```
ADF Statistic: 0.341605
```

```
print('p-value: %f' % result[1])
```

```
p-value: 0.979175
```

```
print('Critical Values:')
```

```
Critical Values:
```

```
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
    1%: -3.437
    5%: -2.864
    10%: -2.568
```

- We fail to reject the null hypothesis that there is a unit root, the time series is non-stationary

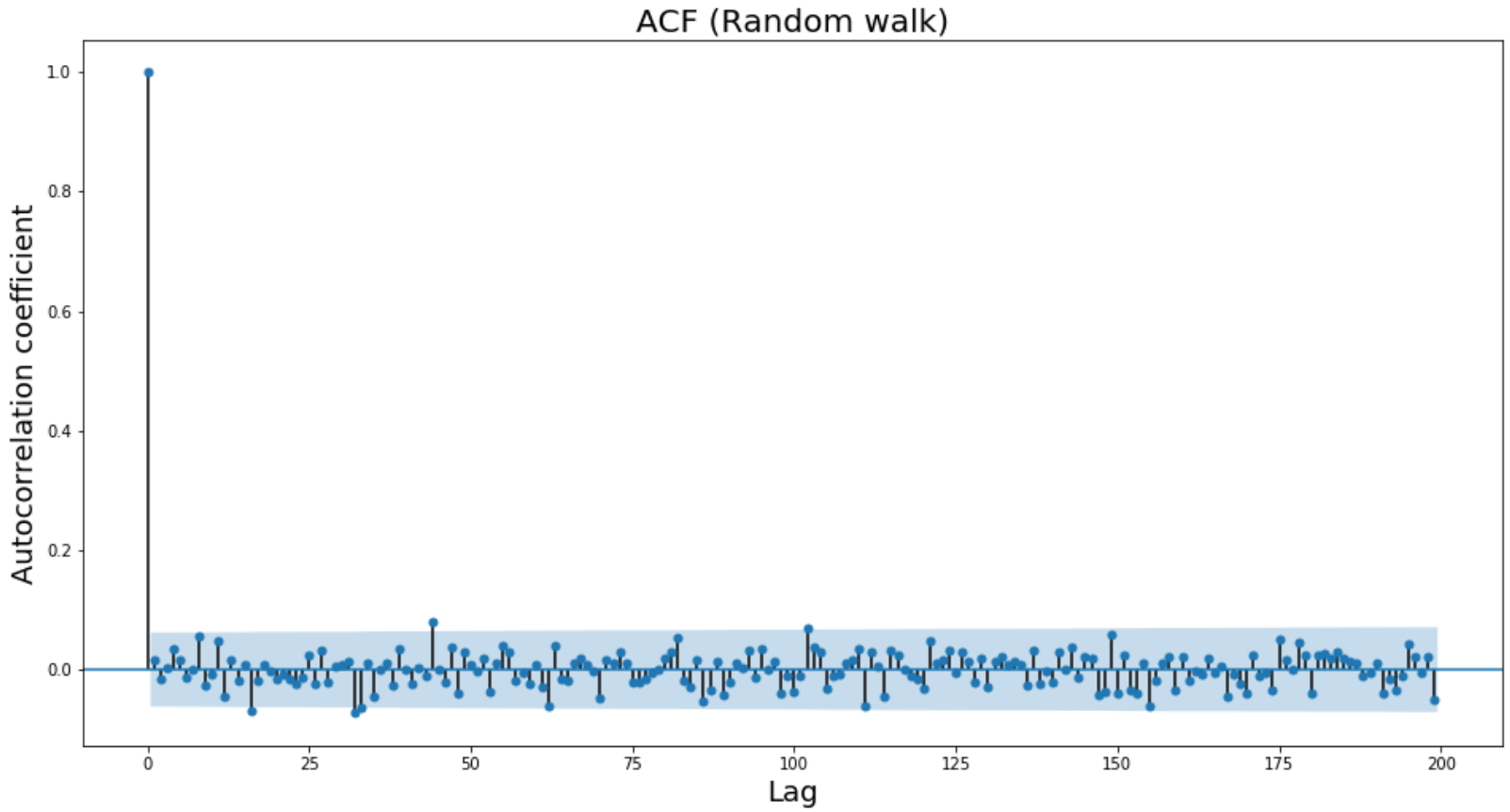
Random walk model - differencing

- We saw the same time series that first appeared non-stationary and then stationary
- This is called **differencing and is one way to make a non-stationary time series stationary**
- Differencing can help stabilize the mean of a time series by removing changes in the level of a time series
- Let's see what happens when we take the first difference of our random series

```
# Take difference
diff = list()
for i in range(1, len(random_walk)):
    value = random_walk[i] - random_walk[i - 1]
    diff.append(value)

diff_df = pd.DataFrame(diff)
```

Random walk model - differencing



- We see that the first difference of the series looks stationary
- Because this is the case on the first difference of the series, the series is a random walk

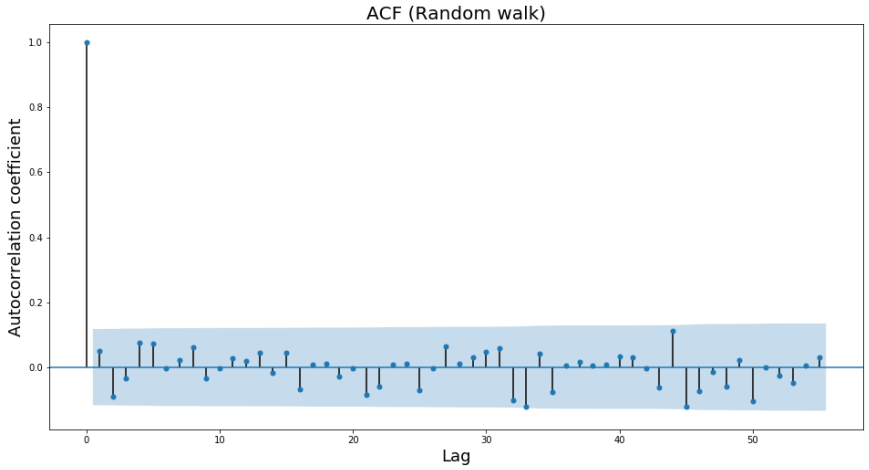
Random walk model - passenger miles

- Now that we know about random walk, let's check our `revenue_passenger_miles` dataset and see if a random walk model might work for it
- Remember, this is what the autocorrelation graph looks like:

Random walk model - first difference

- Let's take the first difference and see if it looks like it becomes stationary

```
# Separate revenue passenger miles.
rpm = passenger_miles['revenue_passenger_miles']
# Take difference of revenue_passenger_miles.
diff_rpm = list()
for i in range(1, len(rpm)):
    value = random_walk[i] - random_walk[i - 1]
    diff_rpm.append(value)
# Save differences as a dataframe.
diff_rpm_df = pd.DataFrame(diff_rpm)
```



- Looks like it is stationary after the first difference!

Random walk model - testing for stationarity

- To validate what we are seeing, let's run the **ADF test** which tests for stationarity by looking for a ***unit root***, where:

$$H_0 : \text{there is a unit root}$$
$$H_a : \text{the time series is stationary}$$

```
ADF Statistic: -12.585263
```

```
p-value: 0.000000
```

```
Critical Values:
```

```
    1%: -3.454
    5%: -2.872
    10%: -2.572
```

- Looks like it meets our criteria for stationarity
- Let's predict based on the assumption that a random walk model will work for this data

Module completion checklist

Objective	Complete
Recall key features of time series	✔
Integrate the concepts and implications of a white noise series	✔
Asses the forecasted models based on comparison criteria	✔
Explain the concept of a random walk model and stationarity	✔
Predict using random walk model and measure error	

DATA SOCIETY © 2019

Random walk model - create lagged dataset

- In order for us to predict using random walk, let's create the lagged dataset

```
# Create lagged dataset.
values = pd.DataFrame(diff_rpm_df.values)
dataframe = pd.concat([values.shift(1), values], axis=1)
dataframe.columns = ['t-1', 't+1']
print(dataframe.head(5))
```

```
   t-1  t+1
0  NaN    1
1  1.0    1
2  1.0   -1
3 -1.0   -1
4 -1.0   -1
```

Random walk model - prediction

- Unfortunately, random walk cannot be reasonably predicted
- The best prediction we could make would be to use the observation at the previous time step as what will happen in the next time step
- This is often called the naive forecast, or a persistence model
- Let's predict on our `revenue_passenger_miles` dataset
- We make sure to keep **the first 70% of the observations** instead of *a random 70%, so that we can predict for a future time period*

```
# Prepare dataset, split into training and test
sets, 70-30 split.
X = dataframe.values
train_size = int(len(X) * 0.70)
train, test = X[1:train_size], X[train_size:]
train_X, train_y = train[:,0], train[:,1]
test_X, test_y = test[:,0], test[:,1]
# Run the persistence model.
def model_persistence(x):
    return x
```

```
# Walk-forward validation.
predictions = list()
for x in test_X:
    yhat = model_persistence(x)
    predictions.append(yhat)
test_score = mean_squared_error(test_y,
predictions)
print('Test MSE: %.3f' % test_score)
```

```
Test MSE: 2.143
```

Knowledge Check 3

Exercise 2

Module completion checklist

Objective	Complete
Recall key features of time series	✔
Integrate the concepts and implications of a white noise series	✔
Asses the forecasted models based on comparison criteria	✔
Explain the concept of a random walk model and stationarity	✔
Predict using random walk model and measure error	✔

DATA SOCIETY © 2019

Workshop: next steps!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code
- This is an exploratory exercise to get you comfortable with the content we discussed today

Tasks for today:

- Read a NY Times Magazine article: ***The Weatherman is Not a Moron***
- Discuss with your peers
- How could the models discussed today apply to the article?
- Find a new time series dataset (different from last class) that you think would fit a random walk model
- Explore and plot the data
- Test data and determine if random walk works best, if not determine what would be next steps

Congratulations on completing this module!