GENERAL ASSEMBLY

TIME SERIES MODELING

Jonathan Balaban

DAT2

LEARNING OBJECTIVES

- Model and predict from time series data using AR, ARMA, or ARIMA models

- Specifically, coding these models in `statsmodels`

PRE-WORK REVIEW

- Prior definition and Python functions for moving averages and autocorrelation
- Prior exposure to linear regression with discussion of coefficients and residuals
- `pip install statsmodels` (should be included with Anaconda)

TIME SERIES MODELING

- In the last class, we focused on exploring time series data and common statistics for time series analysis.
- In this class, we will advance those techniques to show how to predict or forecast forward from time series data.
- With a sequence of values (a time series), we will use the techniques in this class to predict a future value.

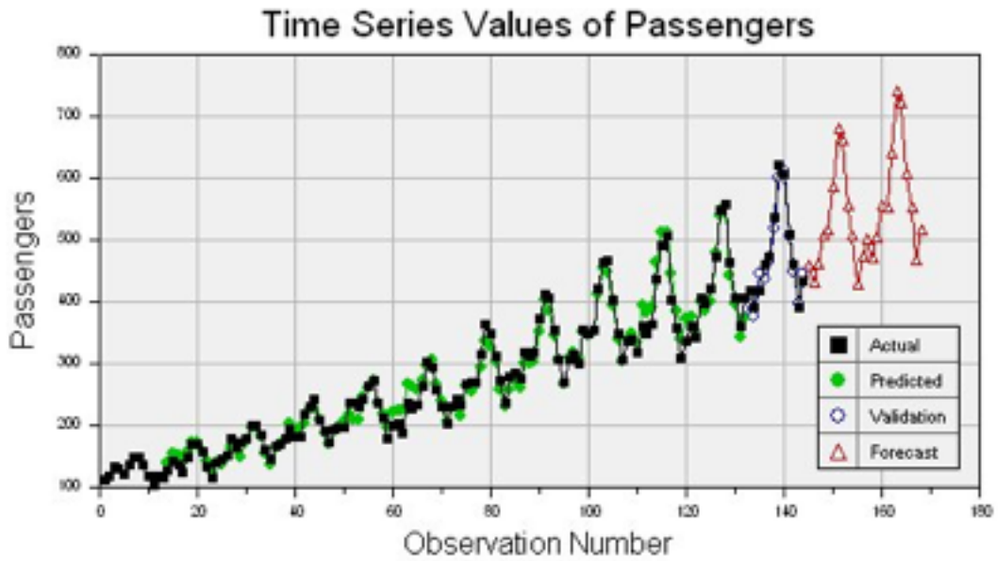
TIME SERIES MODELING

- There are many times when you may want to use a series of values to predict a future value.
  - The number of sales in a future month
  - Anticipated website traffic when buying a server
  - Financial forecasting
  - The number of visitors to your store during the holidays

WHAT ARE TIME SERIES MODELS?

WHAT ARE TIME SERIES MODELS?

- Time series models are models that will be used to predict a future value in the time series.
- **Like** other predictive models, we will use prior history to predict the future.
- **Unlike** previous models, we will use the earlier in time *outcome* variables as *inputs* for predictions.



WHAT ARE TIME SERIES MODELS?

- **Like** previous modeling exercises, we will have to evaluate the different types of models to ensure we have chosen the best one.

- We will want to evaluate on a held-out set or test data to ensure our model performs well on unseen data.

WHAT ARE TIME SERIES MODELS?

- **Unlike** previous modeling exercises, we won't be able to use standard cross-validation for evaluation.

- Since there is a time component to our data, we cannot choose training and test examples at random.

- Suppose we did select a random 80% sample of data points for training and a random 20% for testing. What could go wrong?

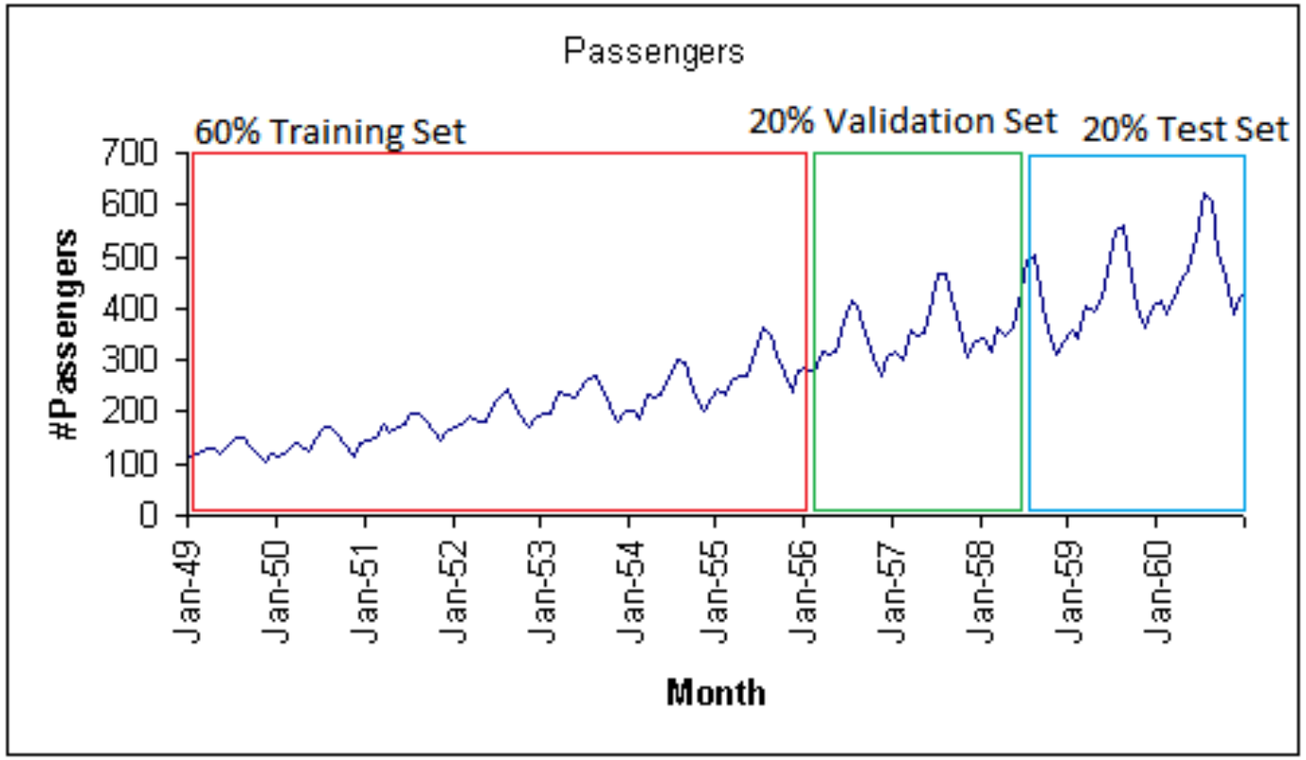
WHAT ARE TIME SERIES MODELS?

- The training dataset would likely contain data from *before* AND *after* a test dataset.

- This would not be possible in real life (you can't use future, unseen data points when building your model). Therefore, it's not a valid test of how our model would perform in practice.

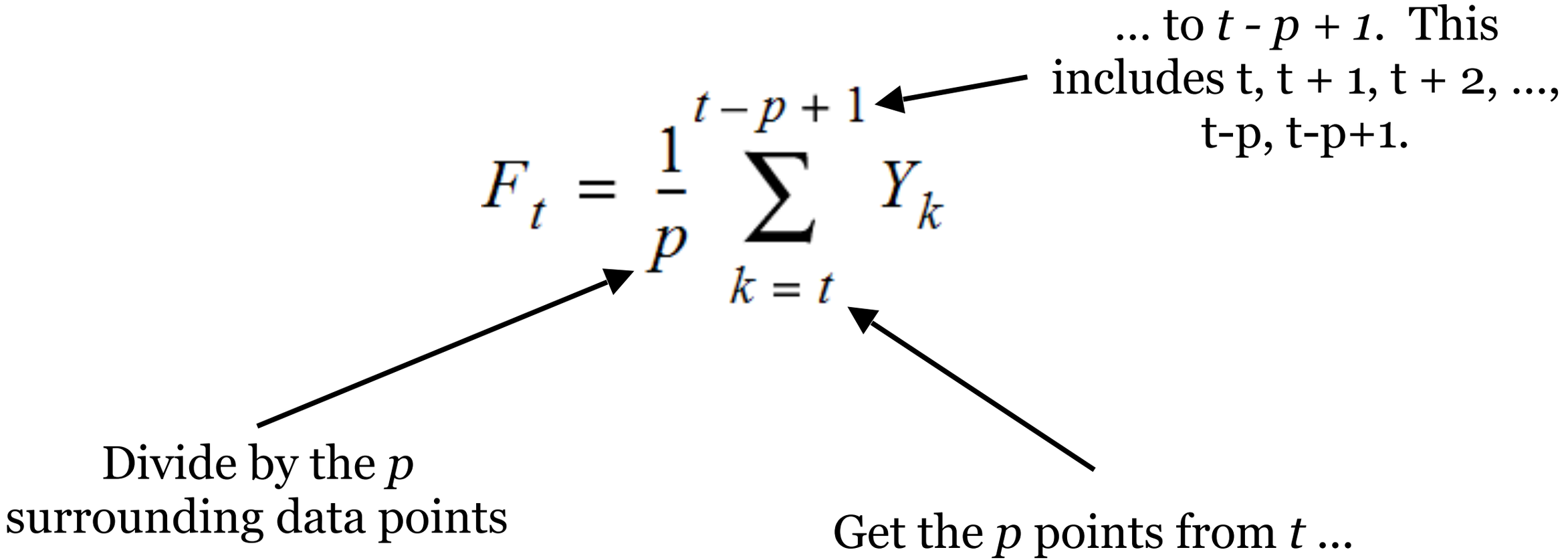
WHAT ARE TIME SERIES MODELS?

- Instead, we will exclusively train on values earlier (in time) in our data and test our model on values at the end of the data period.



PROPERTIES FOR TIME-SERIES PREDICTION

- A *moving average* is an average of *p* surrounding data points in time.



PROPERTIES FOR TIME-SERIES PREDICTION

- *Autocorrelation* is how correlated a variable is with itself. Specifically, how related are variables earlier in time with variables later in time.

$$r_k = \frac{\sum_{t=k+1}^{n} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{n} (y_t - \bar{y})^2}$$

- We fix a *lag*, k, which is how many time points earlier we should use to compute the correlation.

PROPERTIES FOR TIME-SERIES PREDICTION

- We can use these values to assess how we plan to model our time series.
- Typically, for a high quality model, we require some autocorrelation in our data.
- We can compute autocorrelation at various lag values to determine how far back in time we need to go.

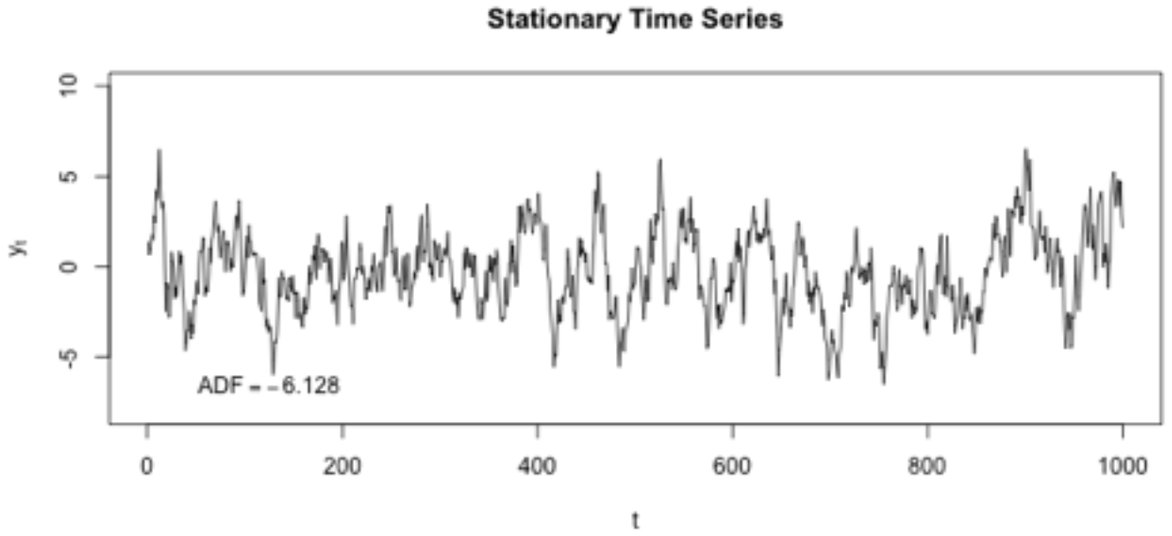
PROPERTIES FOR TIME-SERIES PREDICTION

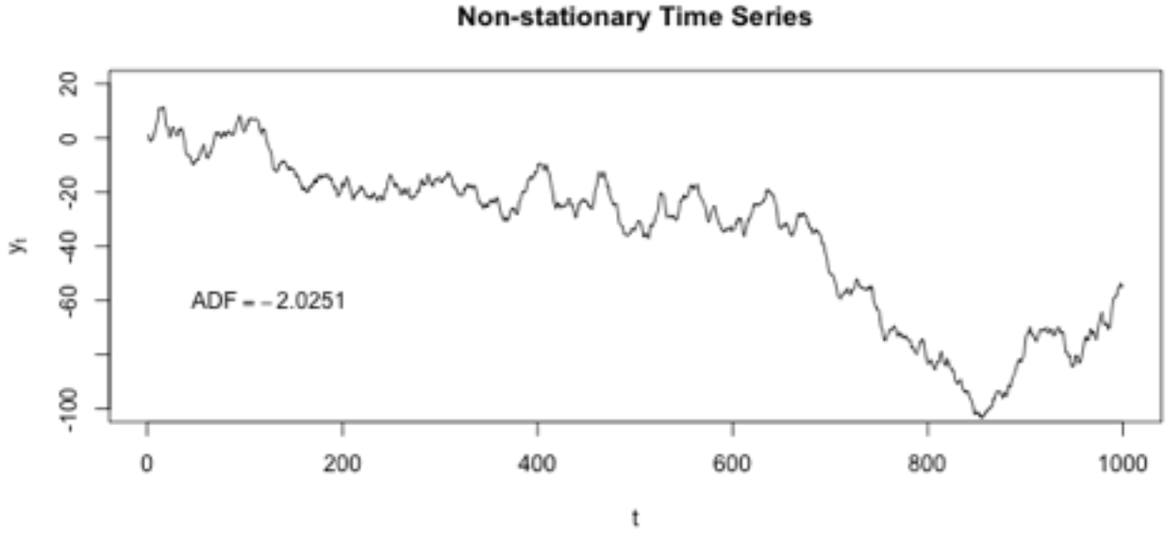
- Many models make an assumption of *stationarity*, assuming the mean and variance of our values is the *same* throughout.

- While the values (e.g. of sales) may shift up or down over time, the mean and variance of sales is constant (i.e. there aren't many dramatic swings up or down).

- These assumptions may not represent real world data; we must be aware of that when we are breaking the assumptions of our model.

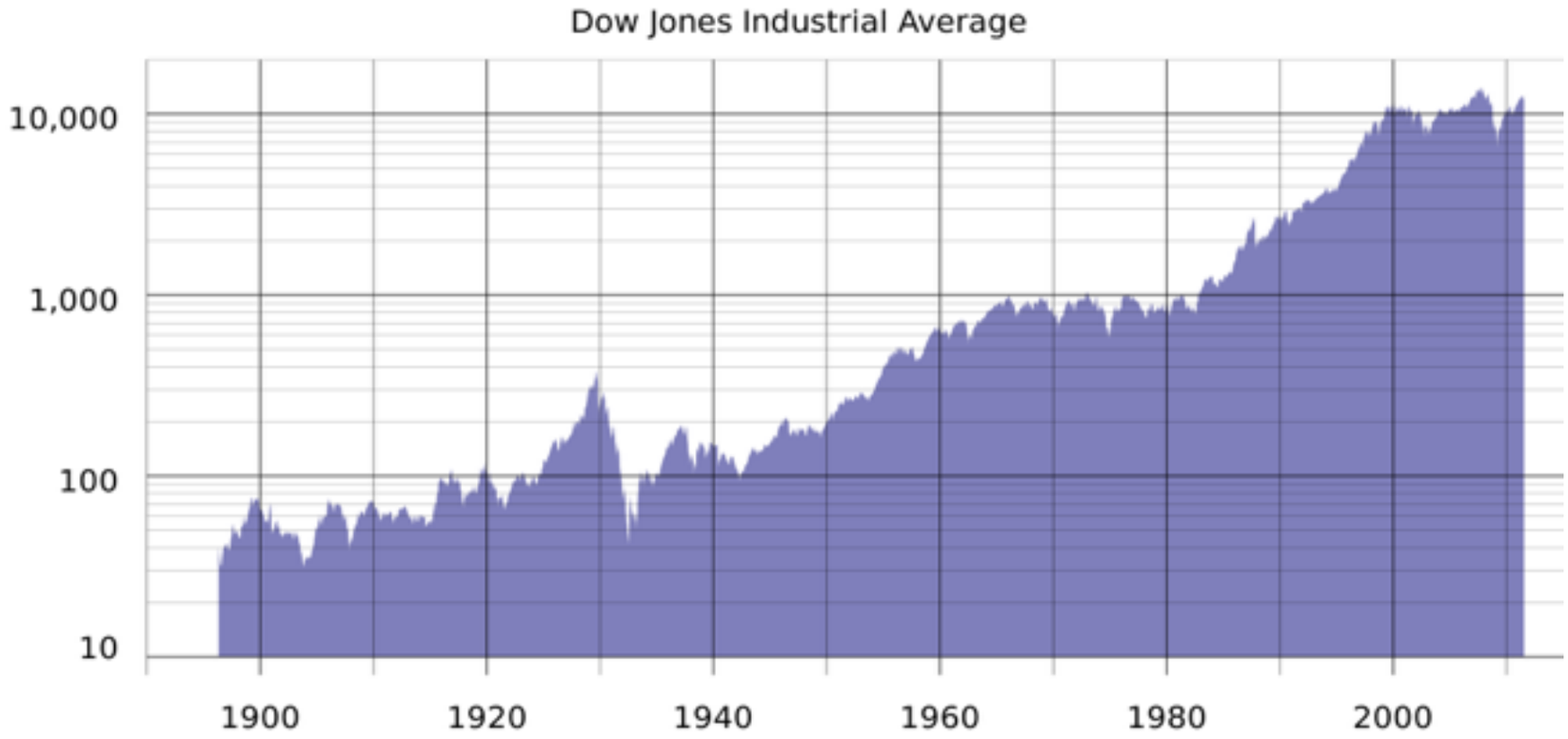
PROPERTIES FOR TIME-SERIES PREDICTION





PROPERTIES FOR TIME-SERIES PREDICTION

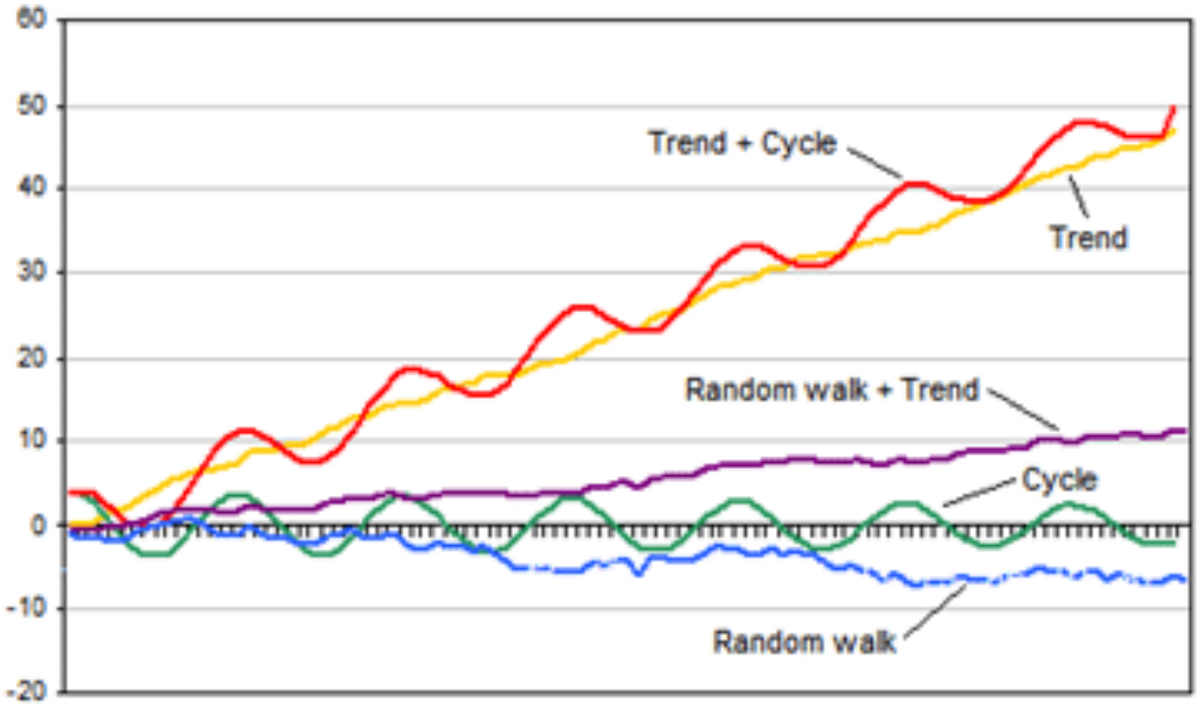
- For example, typical stock or market performance is not stationary. In this plot of Dow Jones performance since 1986, the mean is clearly increasing over time.



PROPERTIES FOR TIME-SERIES PREDICTION

- Below are simulated examples of non-stationary time series and why they might occur.

Table 1 Non-stationary behavior

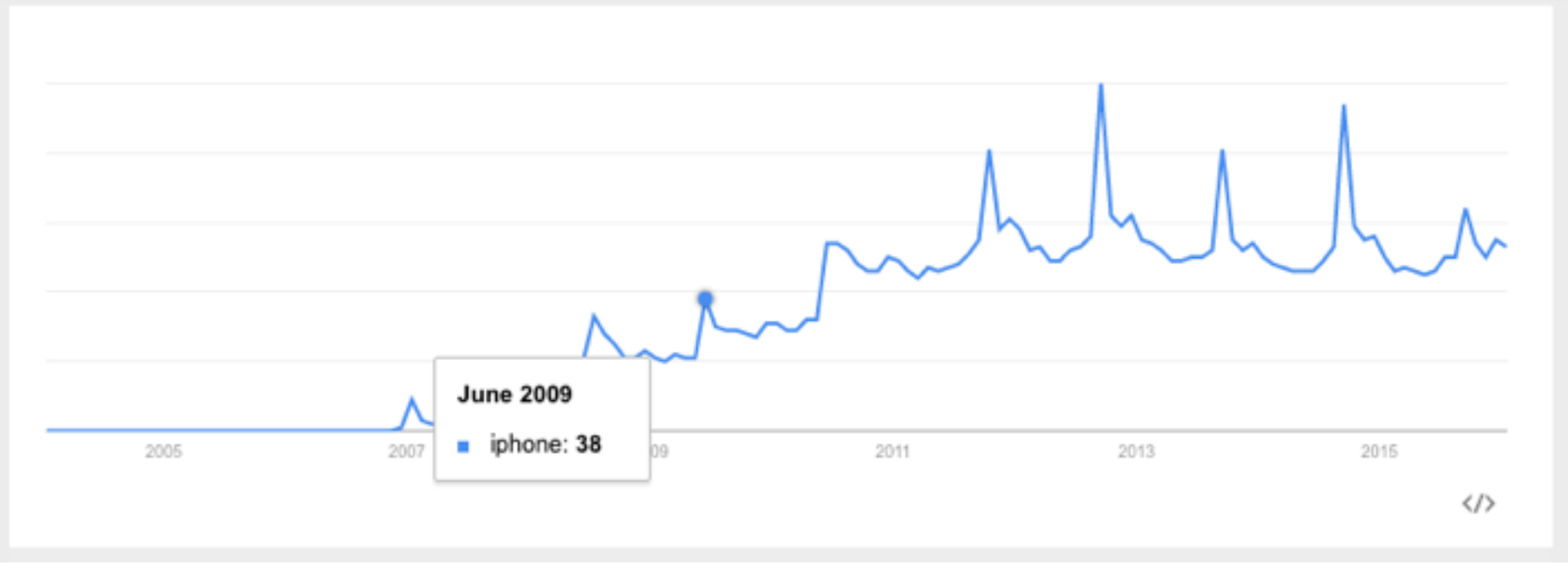


PROPERTIES FOR TIME-SERIES PREDICTION

- Often, if these assumptions don't hold, we can alter our data to make them true. Two common methods are *detrending* and *differencing*.
- *Detrending* would mean to remove any major trends in our data.
- We could do this is many ways, but the simplest is to fit a line to the trend and make a new series that is the difference between the line and the true series.

PROPERTIES FOR TIME-SERIES PREDICTION

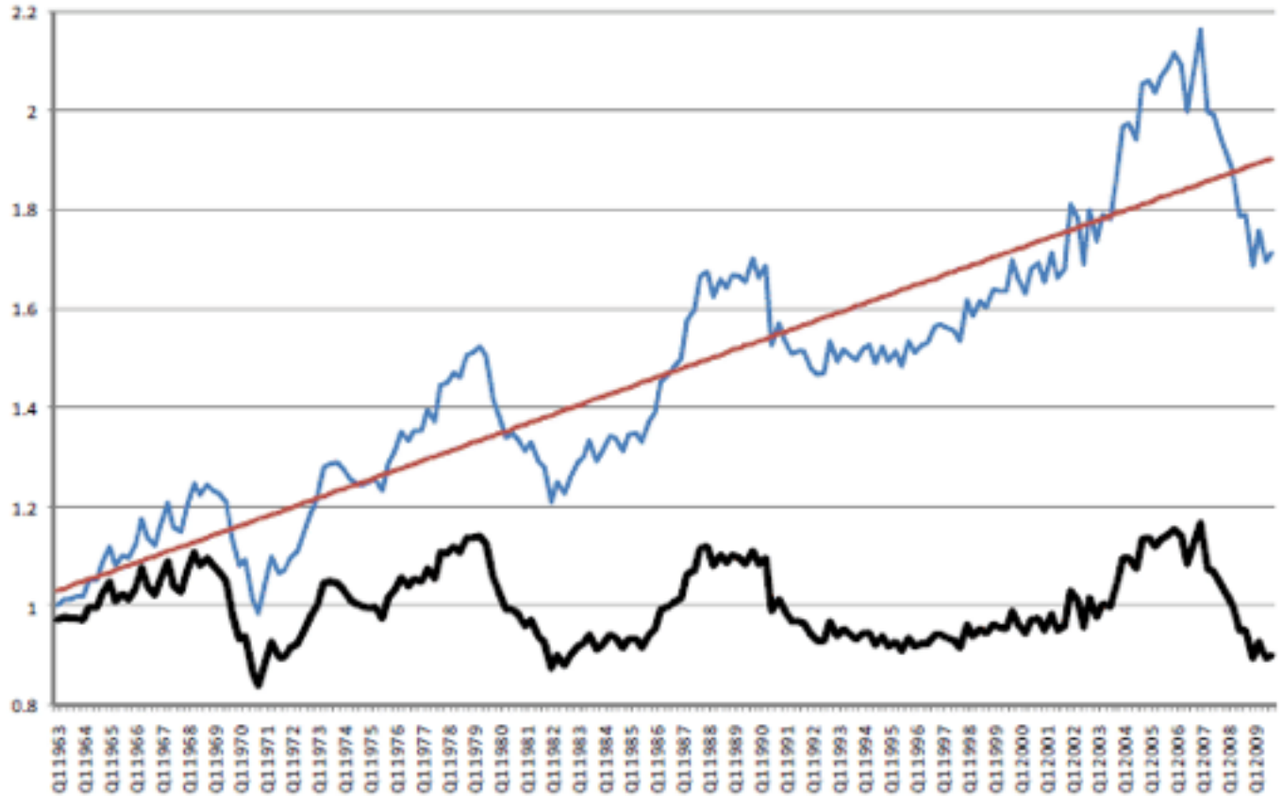
- For example, there is a clear upward (non-stationary) trend in google searches for “iphone”.



- If we fit a line to this data first, we can create a new series that is the difference between the true number of searches and the predicted searches.

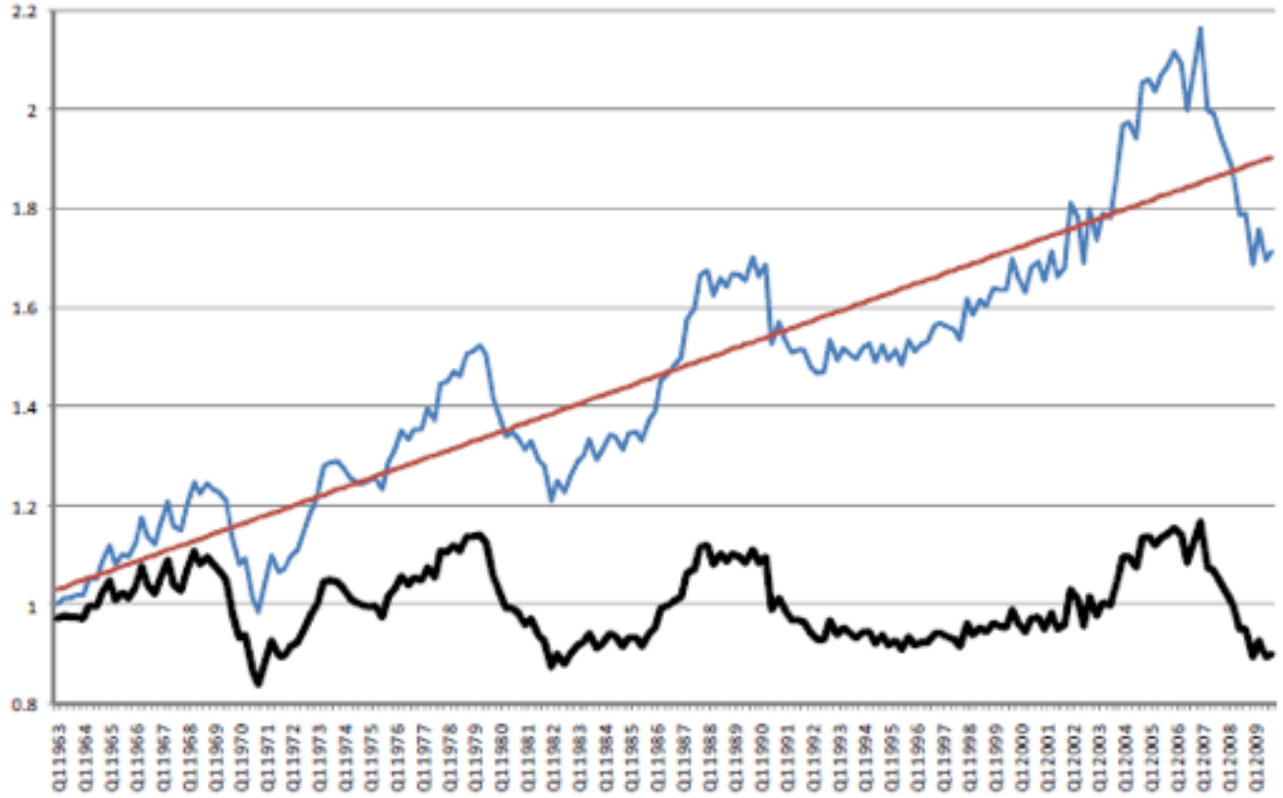
PROPERTIES FOR TIME-SERIES PREDICTION

- Below is an example where we look at US housing prices over time. Clearly, there is an upward trend, making the time series non-stationary (i.e.: the mean house price is increasing).



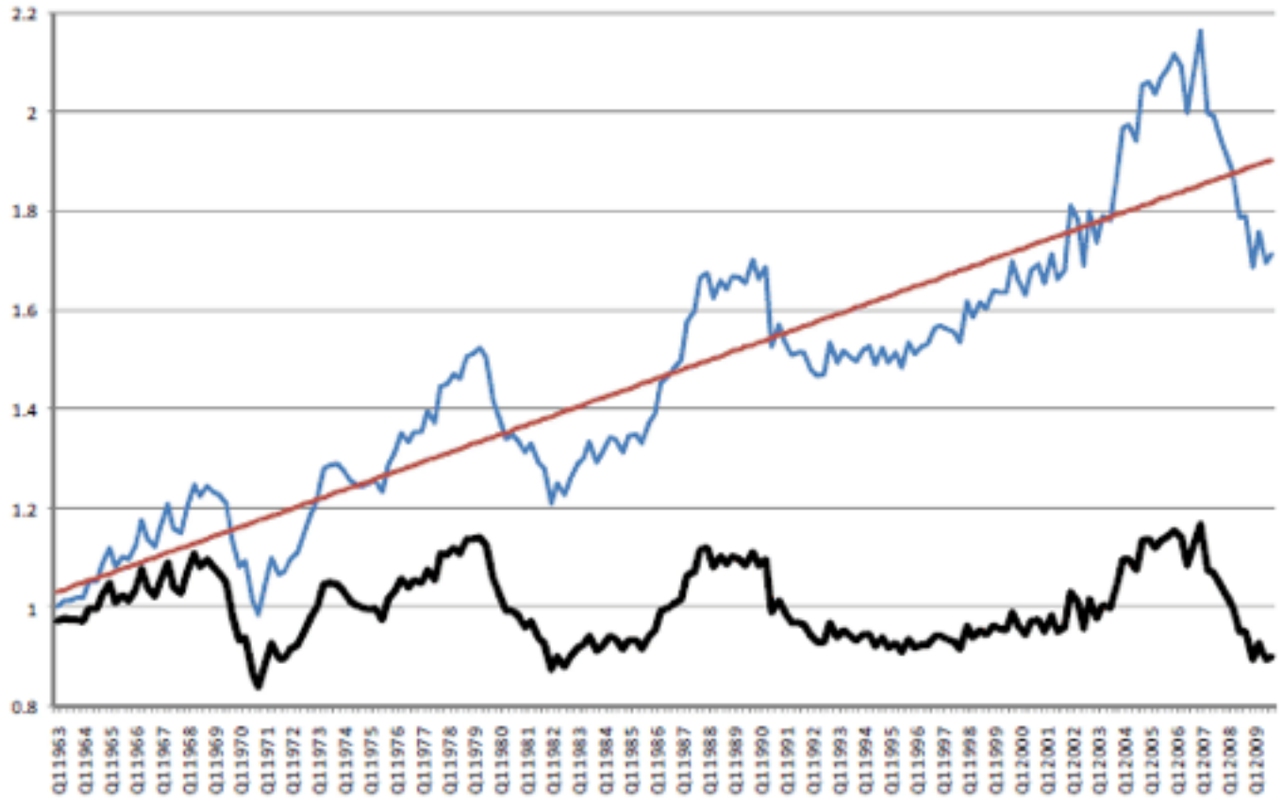
PROPERTIES FOR TIME-SERIES PREDICTION

- We can fit a line that represents the trend. With our trend line, we can subtract the trend line value from the original value to get the bottom figure.



PROPERTIES FOR TIME-SERIES PREDICTION

- The data now has a fixed mean and will be easier to model. This pattern is similar to mean-scaling our features in earlier models with `StandardScaler`.

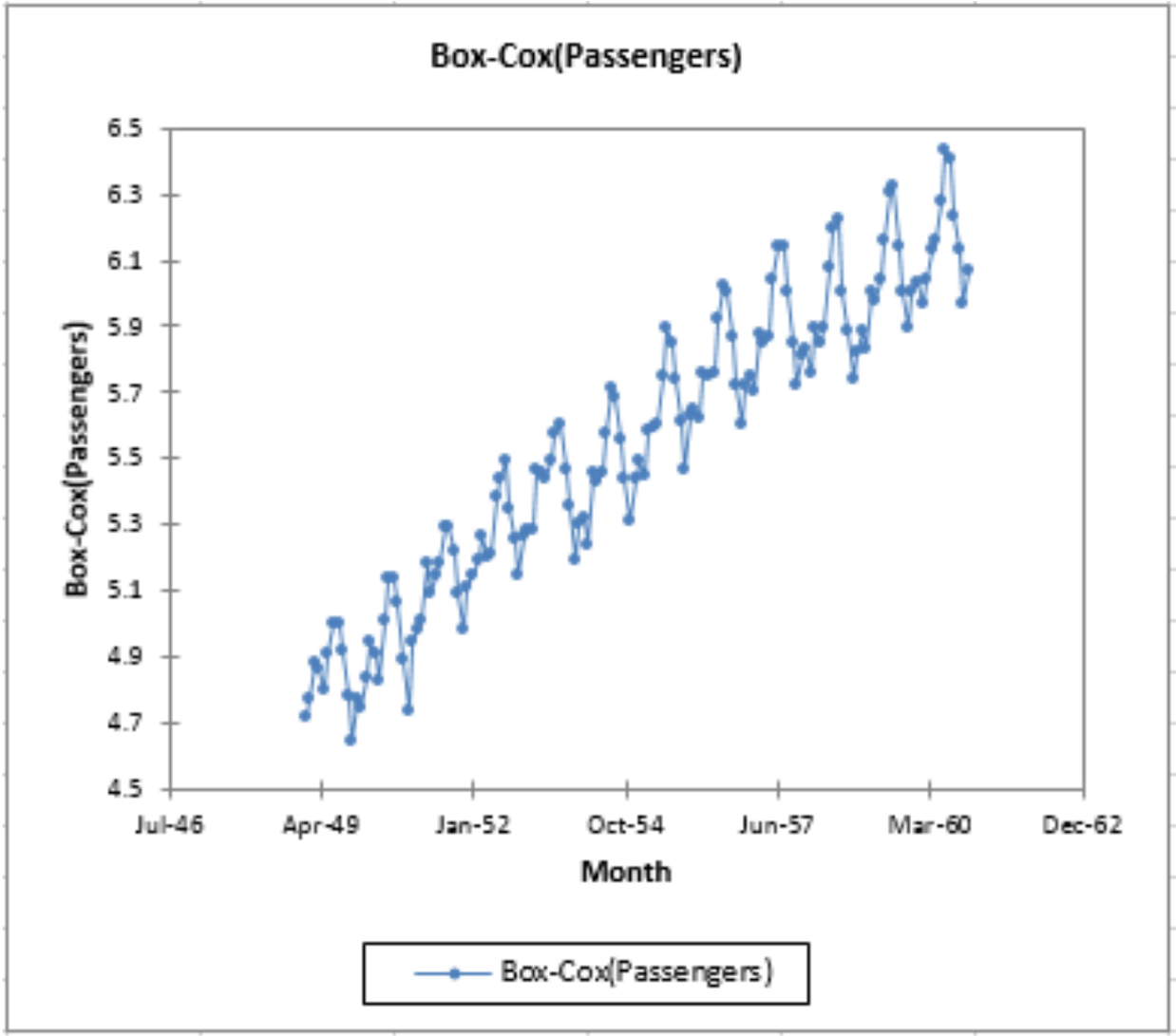


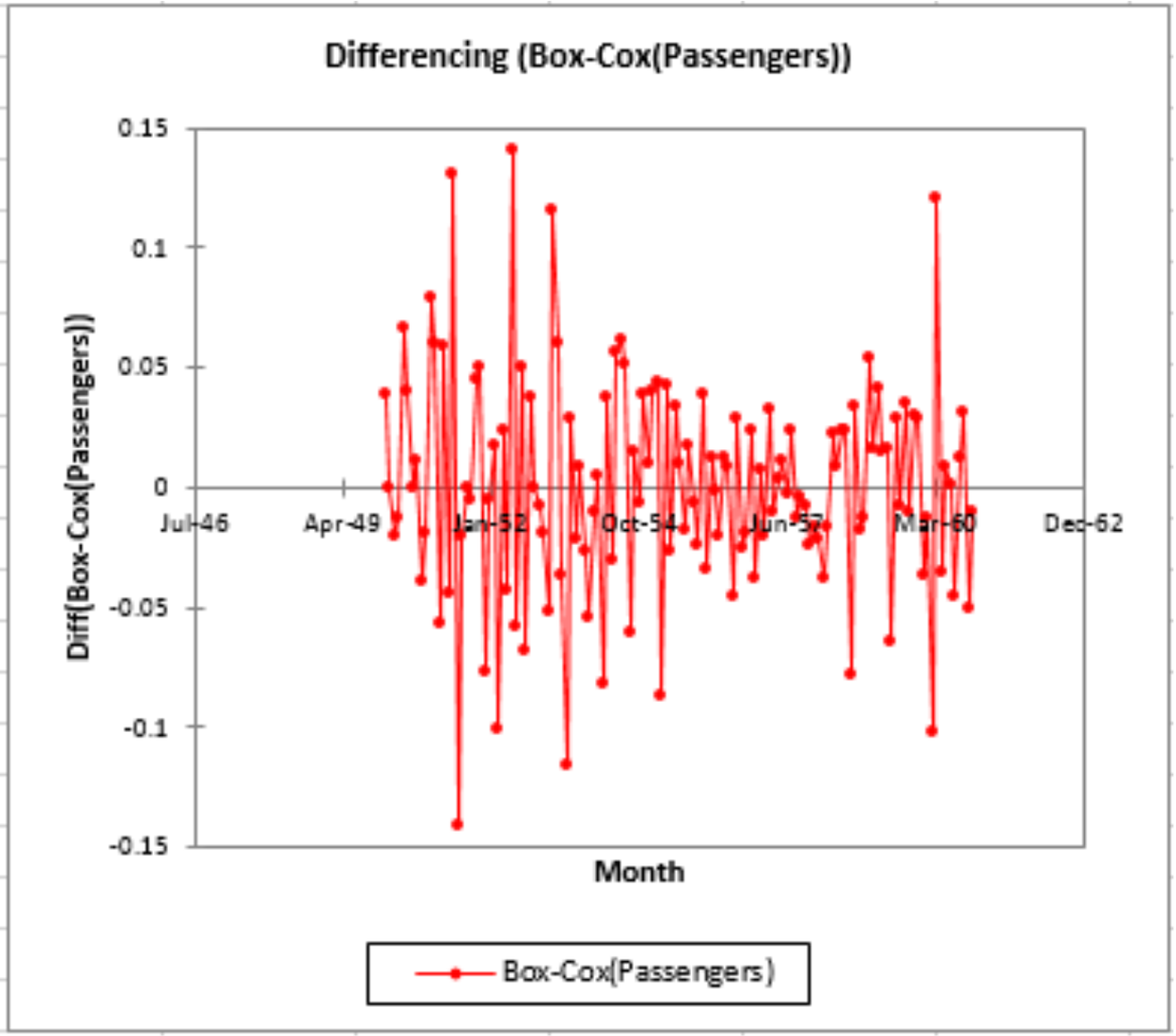
PROPERTIES FOR TIME-SERIES PREDICTION

- A simpler method is *differencing*. This is very closely related to the `diff` function we saw in the last class.

- Instead of predicting the series (again our non-stationary series), we can predict the difference between two consecutive values.

PROPERTIES FOR TIME-SERIES PREDICTION





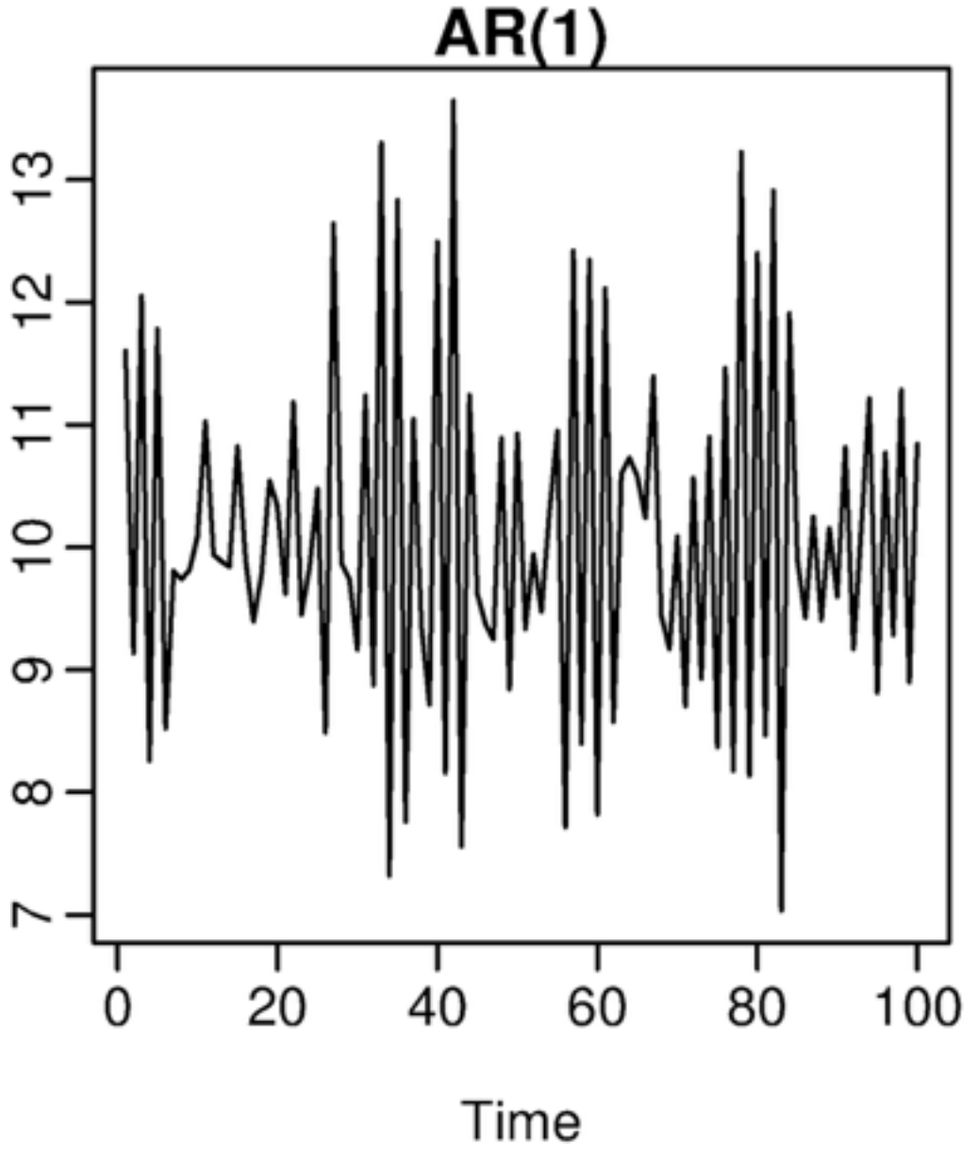
TIME SERIES MODELS

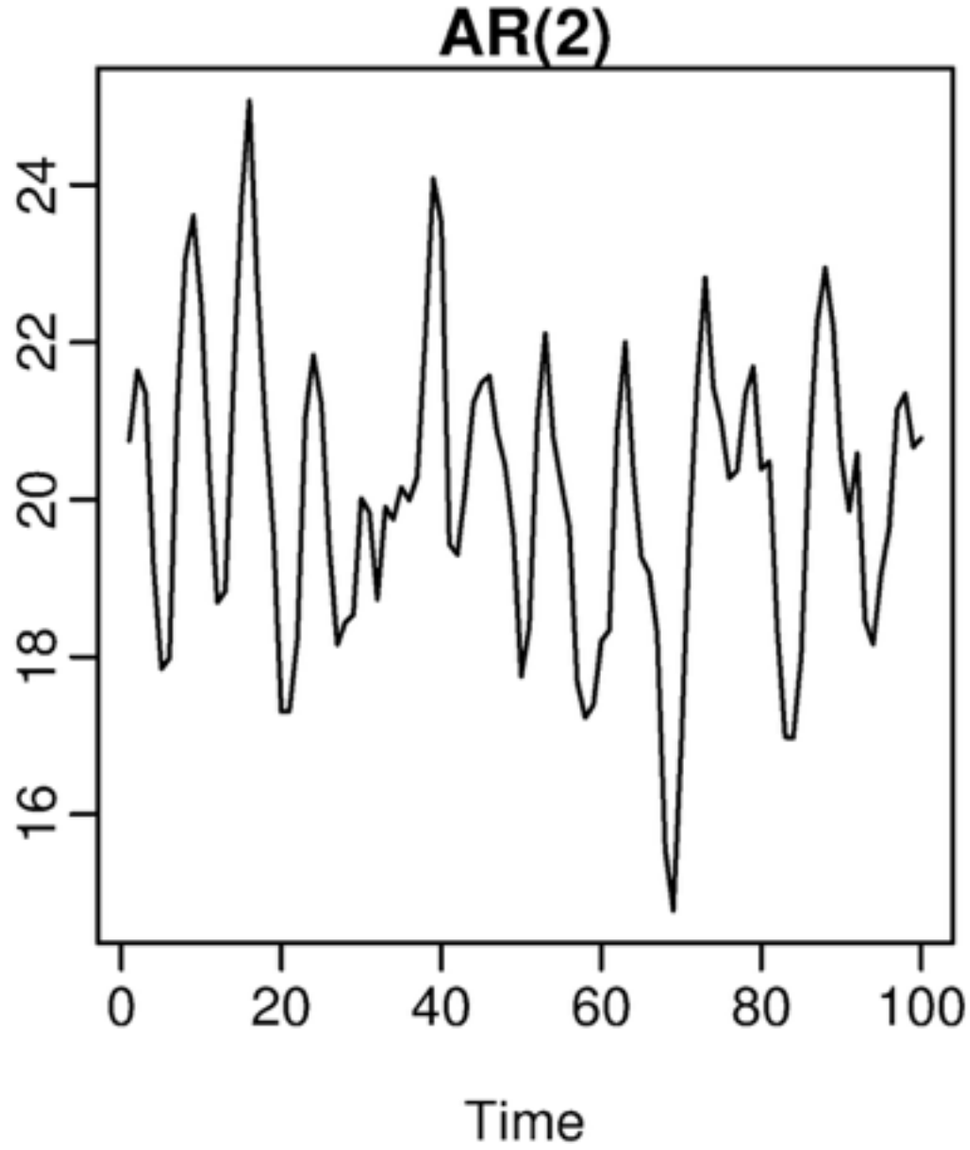
- In the rest of this lesson, we are going to build up to the **ARIMA** time series model.
- This model combines the ideas of differencing and:
  - **AR** - autoregressive models
  - **MA** - moving average models

AR MODELS

- Autoregressive (AR) models are those that use data from previous time points to predict the next.
- This is very similar to previous regression models, except as input, we take the previous outcome.
- If we are attempting to predict weekly sales, we use the sales from a previous week as input.
- AR(p), where *p* indicates the number of previous time points to incorporate, with AR(1) being the most common.

AR MODELS


AR(1)
7
8
9
10
11
12
13
0
20
40
60
80
100
Time


AR(2)
16
18
20
22
24
0
20
40
60
80
100
Time

AR MODELS

- In an autoregressive model, similar to standard regression, we are learning regression coefficients for each of the *p* previous values. Therefore, we will learn *p* coefficients or **β** values.

- If we have a time series of sales per week, y_i, we can regress each y_i from the last *p* values.

$$y_i = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 y_{i-1} + \boldsymbol{\beta}_2 y_{i-2} + \ldots + \boldsymbol{\beta}_p y_{i-p} + \boldsymbol{\varepsilon}$$

- As with standard regression, our model assumes that each outcome variable is a linear combination of the inputs and a random error term.

AR MODELS

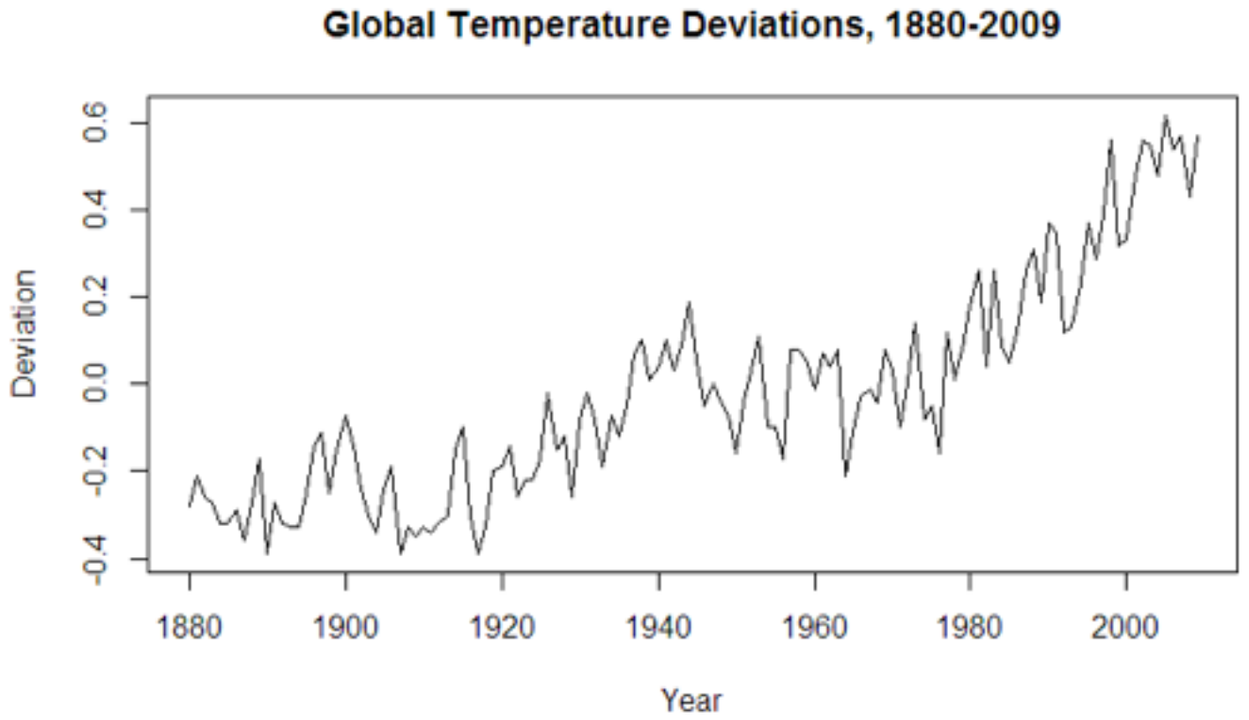
- For an AR(1) model, we will learn a single coefficient.

- This coefficient, $\boldsymbol{\beta}$, will tell us the relationship between the previous value, Y_{t-1}, and the next value, Y_t.

$$Y_t = \boldsymbol{\beta} \cdot Y_{t-1}$$

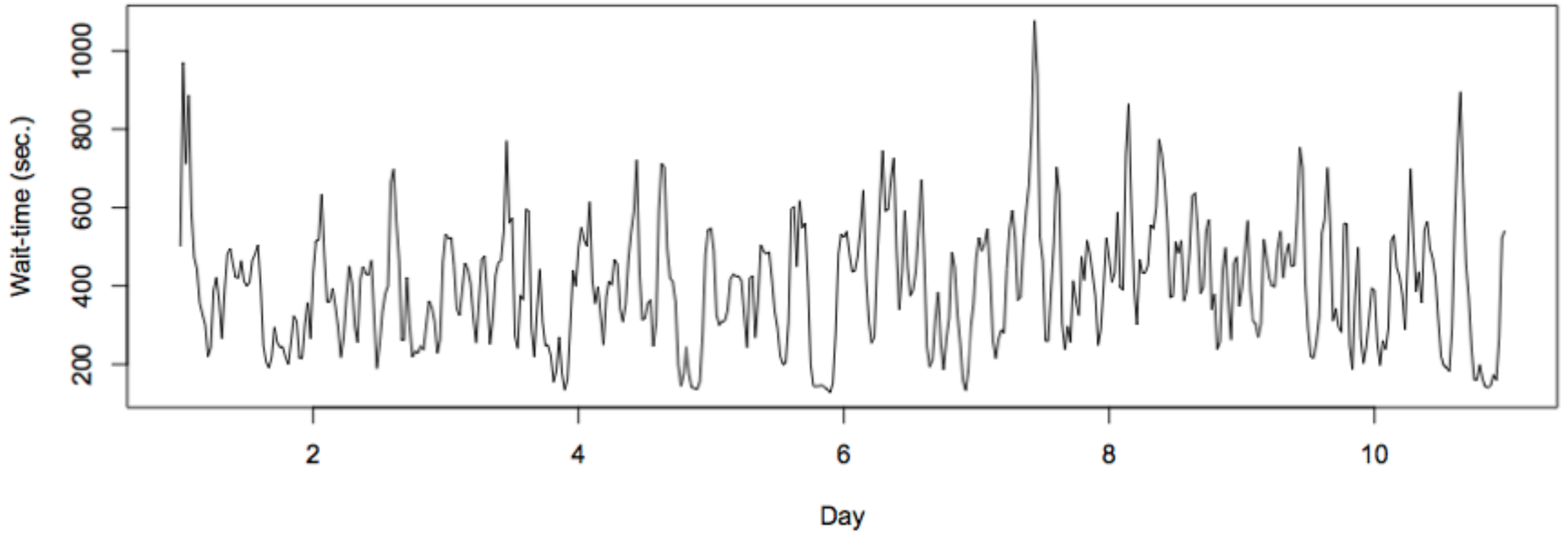
AR MODELS

- A value > 1 would indicate a growth over previous values. This would typically represent non-stationary data, since if we compound the increases, the values are continually increasing.



AR MODELS

- Values between 1 and -1 represent increasing and decreasing patterns from previous patterns.



AR MODELS

- As with other models, interpretation of the model becomes more complex as we add more factors.

- Going from AR(1) to AR(2) can add significant *multi-collinearity*.

AR MODELS

- Recall that *autocorrelation* is the correlation of a value with its series *lagged* behind.

- A model with high correlation implies that the data is highly dependent on previous values and an autoregressive model would perform well.

AR MODELS

- Autoregressive models are useful for learning falls or rises in our series.
- This will weight together the last few values to make a future prediction.
- Typically, this model type is useful for small-scale trends such as an increase in demand or change in tastes that will gradually increase or decrease the series.

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. If we observe an autocorrelation near 1 for lag 1, what do we expect the single coefficient in an AR(1) model to be? >1, between 0 and 1, or <1?
2. What if we observe an autocorrelation of 0?

MA MODELS

- **Moving average (MA) models**, as opposed to AR models, do not take the previous outputs (or values) as inputs. They take the previous error terms.

- We will attempt to predict the next value based on the overall average and how off our previous predictions were.

MA MODELS

- This model is useful for handling specific or abrupt changes in a system.
- AR models slowly incorporate changes in the system by combining previous values; MA models use prior errors to quickly incorporate changes.
- This is useful for modeling a sudden occurrence - something going out of stock, or a sudden rise in popularity affecting sales.

MA MODELS

- As in AR models, we have an order term, q, and we refer to our model as MA(q). The moving average model is dependent on the last q errors.

- If we have a time series of sales per week, y_i, we can regress each y_i from the last q error terms.

$$y_i = \text{mean} + \boldsymbol{\beta}_1\boldsymbol{\varepsilon}_{i-1} + \boldsymbol{\beta}_2\boldsymbol{\varepsilon}_{i-2} + \ldots + \boldsymbol{\beta}_q\boldsymbol{\varepsilon}_{i-q}$$

- We include the mean of the time series (that's why it's called a moving average) as we assume the model takes the mean value of the series and randomly jumps around it.

MA MODELS

- Of course, we don’t have error terms when we start - where do they come from?
- This requires a more complex fitting procedure than we have seen previously.
- We need to iteratively fit a model (perhaps with random error terms), compute the errors and then refit, again and again.

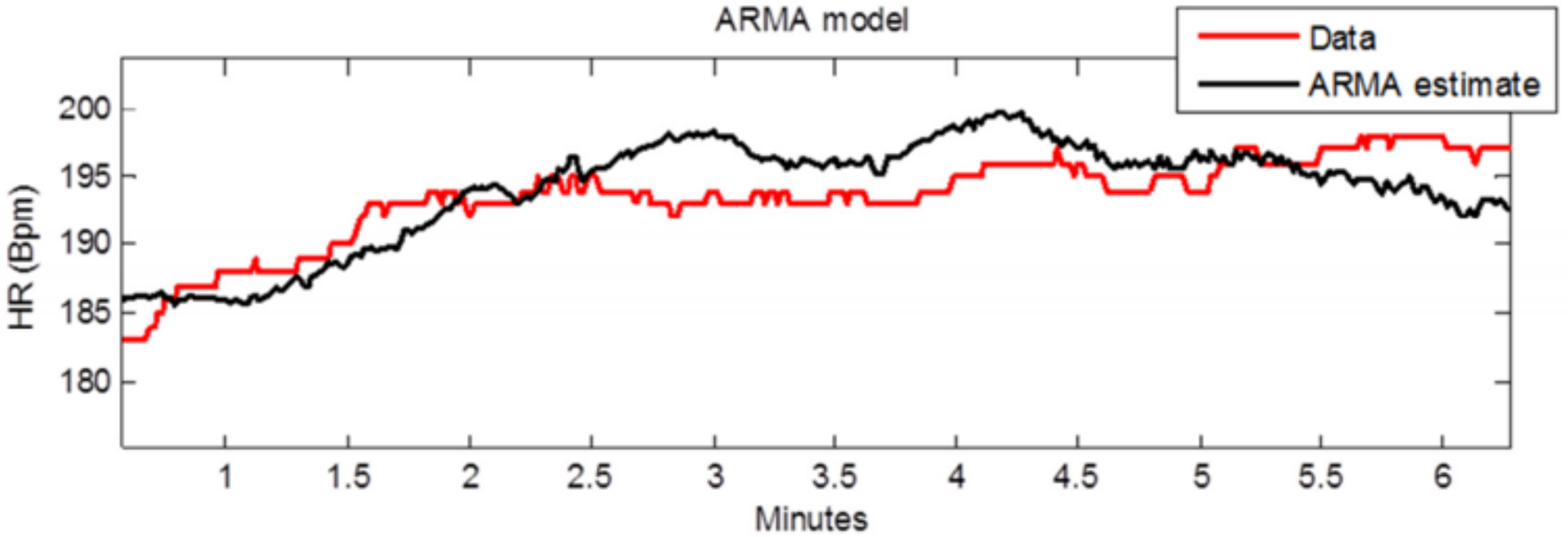
MA MODELS

- In this model, we learn q coefficients.
- In an MA(1) model, we learn one coefficient.
- This value indicates the impact of our previous error term on the next prediction.

ARMA MODELS

- **ARMA** (pronounced ‘R-mah’) models combine the autoregressive and moving average models.

- An ARMA(p,q) model is simply a combination (sum) of an AR(p) model and MA(q) model.



ARMA MODELS

- We specify two model settings, p and q, which correspond to combining an AR(p) model with an MA(q) model.
- Incorporating both models allows us to mix two types of effects.
  - AR models slowly incorporate changes in preferences, tastes, and patterns.
  - Moving average models base their prediction on the prior error, allowing to correct sudden changes based on random events - supply, popularity spikes, etc.

ARIMA MODELS

- **ARIMA** (pronounced ‘uh-ri-mah’) is an **A**uto**R**egressive **I**ntegrated **M**oving **A**verage model.

- In this model, we learn an ARMA(p,q) model to predict *the difference* of the series (as opposed to the value of the series).

ARIMA MODELS

- Recall the pandas `diff` function. This computes the difference between two consecutive values.

- In an ARIMA model, we attempt to predict this difference instead of the actual values.

$$y_t - y_{t-1} = \mathbf{ARIMA(p,q)}$$

- This handles the stationarity assumption we wanted for our data. Instead of detrending or differencing manually, the model does it for us!

ARIMA MODELS

- An ARIMA model has three parameters and is specified ARIMA(p, d, q).
  - *p* is the order of the autoregressive component
  - *q* is the order of the moving average component
  - *d* is the degree of differencing.
- *d* was 1 in our prior example. For d=2, our model would be

$$\text{diff}(\text{diff}(y)) = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = \text{ARIMA}(p,q)$$

ARIMA MODELS

- Compared to an ARMA model, ARIMA models do **not** rely on the underlying series being stationary.
- The differencing operation can *convert* the series to one that is stationary.
- Instead of attempting to predict values over time, our new series is the difference in values over time.
- Since ARIMA models include differencing, they can be used on a broader set of data without the assumption of a constant mean.

DEMO

TIME SERIES MODELING IN STATSMODELS

TIME SERIES MODELING IN STATSMODELS

- To explore time series models, we will continue to use the Rossmann sales data.

- This dataset has sales data for every Rossmann store for a 3-year period and indicators for holidays and basic store information.

TIME SERIES MODELING IN STATSMODELS

- In the last class, we saw that we could plot the sales data at a particular store to identify how the sales changed over time.
- We also computed autocorrelation for the data at varying lag periods.
- This helps us identify if previous timepoints are predictive of future data and which time points are most important - the previous day, week, or month.

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. Compute the autocorrelation of Sales in Store 1 for lag 1 and 2.
2. Will we be able to use a predictive model, particularly an autoregressive one?

TIME SERIES MODELING IN STATSMODELS

```
store1_data.Sales.autocorr(lag=1) # -0.12
store1_data.Sales.autocorr(lag=2) # -0.03
```

- We do see some minimal correlation in time, implying an AR model can be useful. An easier way to diagnose this may be to plot many autocorrelations at once.

TIME SERIES MODELING IN STATSMODELS

```
%matplotlib inline
from pandas.tools.plotting import autocorrelation_plot

autocorrelation_plot(store1_data.Sales)
```

- This shows a typical pattern of an autocorrelation plot, that it should decrease to 0 as lag increases. However, it's hard to observe exactly what the values are.

TIME SERIES MODELING IN STATSMODELS

- In this class, we will use `statsmodels` to code AR, MA, ARMA, and ARIMA models.
- `statsmodels` provides a nice summary utility to help us diagnose models.

TIME SERIES MODELING IN STATSMODELS

- statsmodels also has a better autocorrelation plot that allows us to look at fixed number of lag values.

```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(store1_data.Sales, lags=10)
```

- Here we observe autocorrelation at 10 lag values. 1 and 2 are what we saw before.

- This implies a small but limited impact based on the last few values. An autoregressive model might be useful.

TIME SERIES MODELING IN STATSMODELS

- We also see a larger spike at 7 (the seventh day in the week).
- If we observed a handful of random distributed spikes, a moving average model would be useful.

```
plot_acf(store1_data.Sales, lags=25)
```

- We can expand the window to 25 days to see that the random spikes occur regularly at 7 days. What does this mean?

AR, MA, AND ARMA MODELS IN STATSMODELS

- To explore AR, MA, and ARMA models, we will use `sm.tsa.ARMA`.
- Remember, an ARMA model is a combination of autoregressive and moving average models.
- We can train an AR model by turning off the MA component (q=0).

```
from statsmodels.tsa.arima_model import ARMA

store1_sales_data = store1_open_data[['Sales']].astype(float)
model = ARMA(store1_sales_data, (1, 0)).fit()
model.summary()
```

AR, MA, AND ARMA MODELS IN STATSMODELS

- By passing `(1, 0)` in the second argument, we are fitting an ARMA model with p=1, q=0. This is the same as an AR(1) model.
- In this AR(1) model, we learn an intercept (or base sales) value.
- Additionally, we learn a coefficient that tells us how to include the latest sales value.
- In this case, we add an intercept of ~4700 to 0.68 times the previous month's sales. Note that the coefficient is not equal to the lag 1 autocorrelation. This implies the data is **not** stationary.

AR, MA, AND ARMA MODELS IN STATSMODELS

- We can learn an AR(2) model, which regresses each sales value on the last two.

```
model = ARMA(store1_sales_data, (2, 0)).fit()
model.summary()
```

- In this case, we learn two coefficients, which tell us the effect of the last two sales values on the current sales.

- While this model may perform better, it may be more difficult to interpret.

AR, MA, AND ARMA MODELS IN STATSMODELS

- Residuals are the errors of the model or how off our predictions are.
- Ideally, we want randomly distributed errors that are small.
- If the errors are large, our model does not perform well.
- If the errors have a pattern, particularly over time, we may have overlooked something in the model or have periods of time that are different than the rest of the dataset.

AR, MA, AND ARMA MODELS IN STATSMODELS

- We can use `statsmodels` to plot the residuals.

```
model.resid.plot()
```

- Here we see large spikes at the end of each year, indicating that our model does not account for the holiday spikes.
- Our model considers a short period of time, so it does not take into account the longer seasonal pattern.

AR, MA, AND ARMA MODELS IN STATSMODELS

- We can also plot the autocorrelations of the residuals. In an ideal world, these would all be near 0 and appear random.

```
plot_acf(model.resid, lags=50)
```

- This plot shows a problem: the errors are increasing and decreasing every week in a clear pattern.

- We may need to expand our model.

AR, MA, AND ARMA MODELS IN STATSMODELS

- To expand this AR model to an ARMA model, we can include the moving average component as well.

```
model = ARMA(store1_sales_data, (1, 1)).fit()
model.summary()
```

- Now we learn two coefficients, one for the AR(1) component and one for the MA(1) component.

AR, MA, AND ARMA MODELS IN STATSMODELS

- Remember that this is an AR(1) + MA(1) model. The AR coefficient represents dependency on the last value and the MA component represents any spikes independent of the last value.
- The coefficients here are 0.69 for the AR component and -0.03 for the MA component.
- The AR coefficient is the same as before (decreasing values).
- The MA component is fairly small (which we should have expected from the autocorrelation plots).

ARIMA MODELS IN STATSMODELS

- We can also use `statsmodels` to fit ARIMA models. Let's start by using ARIMA(1, 0, 1) to fit an ARMA(1, 1) model.

```
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(store1_sales_data, (1, 0, 1)).fit()
model.summary()
```

- We can see that this model is the same as our previous ARMA model.

ARIMA MODELS IN STATSMODELS

- We can also fit a true ARIMA model to predict the difference of the series.

```
model = ARIMA(store1_sales_data, (1, 1, 1)).fit()
model.summary()
```

- We can remove the MA component since it does not appear to be useful.

```
model = ARIMA(store1_sales_data, (1, 1, 0)).fit()
model.summary()
```

- We now have an AR(1) model on the differenced series with a coefficient of -0.18.

ARIMA MODELS IN STATSMODELS

- We can compute the lag 1 autocorrelation of the differenced series and see if they match.

```
store1_sales_data.Sales.diff(1).autocorr(1) #-0.181
```

- We can also plot it to see the difference.

```
store1_sales_data.Sales.diff(1).plot()
```

- They match. Note that this is generally true, but the variance is NOT constant. It's mostly the same throughout the series except around the holidays.

ARIMA MODELS IN STATSMODELS

- With our models, we can also plot our predictions against the true series using the `plot_predict` function.

- We can compare the last 50 days of true values against our predictions.

```
model.plot_predict(0, 50)
```

- The function takes two arguments, the start and end index of the dataframe to plot. Here, we are plotting the last 50 values.

ARIMA MODELS IN STATSMODELS

- To plot earlier values with our predictions continuing where the true values stop, we can do the following.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax = store1_sales_data['2014'].plot(ax=ax)

fig = model.plot_predict(0, 200, ax=ax, plot_insample=False)
```

- This plots true values in 2014 and our predictions 200 days out from 2014.

ARIMA MODELS IN STATSMODELS

- The two previous problems remain:
  - Large errors around the holiday period
  - Errors with high autocorrelation

ARIMA MODELS IN STATSMODELS

- We can adjust the AR component of the model to adjust for a piece of this. Let's increase the lag to 7.

```
model = ARIMA(store1_sales_data, (7, 1, 2)).fit()
model.summary()

plot_acf(model.resid, lags=50)
```

- This removes some of the autocorrelation in the residuals but large discrepancies still exist.

- However, they exist where we are breaking our model assumptions.

ARIMA MODELS IN STATSMODELS

- Increasing p increases the dependency on previous values further (longer lag). But our autocorrelation plots show this isn't necessary past a certain point.

- Increasing q increases the likelihood of an unexpected jump at a handful of points. The autocorrelation plots show this doesn't help past a certain point.

- Increasing d increases differencing, but d=1 moves our data towards stationarity (other than a few points). d=2 would imply an exponential trend which we don't have here.

ARIMA MODELS IN STATSMODELS

- There are variants of ARIMA that will better handle the seasonal aspect of our data. This is referred to as Seasonal ARIMA.

- These models fit two ARIMA models, one on the current frequency (daily in our example) and another on the seasonal frequency (maybe monthly or yearly patterns).

- Additionally, issues with seasonality could be handled by preprocessing tricks such as detrending.

INDEPENDENT PRACTICE

WALMART SALES DATA

ACTIVITY: WALMART SALES DATA

EXERCISE

DIRECTIONS (50 minutes)

We will analyze the weekly sales data from Walmart over a two year period from 2010 to 2012. The data is separated by store and department, but we will focus on analyzing one store for simplicity.

To read in the data

```
import pandas as pd
import numpy as np

%matplotlib inline

data = pd.read_csv('lessons/lesson-16/assets/data/
train.csv')
data.set_index('Date', inplace=True)
data.head()
```

ACTIVITY: WALMART SALES DATA

DIRECTIONS

Complete the following tasks:

1. Filter the dataframe to Store 1 sales and aggregate over departments to compute the total sales per store.
2. Plot the `rolling_mean` for `Weekly_Sales`. What general trends do you observe?
3. Compute the 1, 2, 52 autocorrelations for `Weekly_Sales` and/or create an autocorrelation plot.
4. What does the autocorrelation plot say about the type of model you want to build?

ACTIVITY: WALMART SALES DATA

EXERCISE

DIRECTIONS

5. Split the weekly sales data in a training and test set - using 75% of the data for training.
6. Create an AR(1) model on the training data and compute the mean absolute error of the predictions.
7. Plot the residuals - where are their significant errors?
8. Compute and AR(2) model and an ARMA(2, 2) model - does this improve your mean absolute error on the held out set?
9. Finally, compute an ARIMA model to improve your prediction error - iterate on the p, q, and parameters comparing the model's performance..

CONCLUSION

- Time-series models use previous values to predict future values, also known as forecasting.
- AR and MA model are simple models on previous values or previous errors respectively.
- ARMA combines these two types of models to account for both gradual shifts (due to AR models) and abrupt changes (MA models).

CONCLUSION

- ARIMA models train ARMA models on differenced data to account for non-stationary data.
- Note that none of these models may perform well for data that has more random variation.
- For something like iPhone sales (or searches), which may be sporadic with short periods of increase, these models may not work well.

COURSE

BEFORE NEXT CLASS

Final: Part 4

LESSON

Q & A

LESSON

EXIT TICKET

DON’T FORGET TO FILL OUT YOUR EXIT TICKET