multiverse

multiverse

Data Science Unit 4

Time Series

Before we start…

- → Make sure you are comfortable
- → Have water and maybe a strong coffee handy
- → If you need a break… take it!
- → If you need a stretch – please go ahead!
- → Please mute yourselves if you are not talking
- → Have your video on at all times

…and let's get started!

In this session we will…

1. **Define** time series data
2. **Calculate** appropriate statistics for analysing time series data
3. **Investigate** techniques for EDA with time series data

multiverse

What’s the Time?

Time Series – What is it?

DateTime!

DateTime

DateTime

```python
# The datetime library is something you should already have from Anaconda.
from datetime import datetime
```

```python
# Let's just set a random datetime
lesson_date = datetime(2020, 8, 21, 12, 21, 12, 844089)
```

The components of the date are accessible via the object's attributes.

```python
print("Micro-Second", lesson_date.microsecond)
print("Second", lesson_date.second)
print("Minute", lesson_date.minute)
print("Hour", lesson_date.hour)
print("Day", lesson_date.day)
print("Month",lesson_date.month)
print("Year", lesson_date.year)
```

```
Micro-Second 844089
Second 12
Minute 21
Hour 12
Day 21
Month 8
Year 2020
```

timedelta

```
# Import timedelta() from the DateTime library.
from datetime import timedelta

# Timedeltas represent time as an amount rather than as a fixed position.
offset = timedelta(days=1, seconds=20)

# The timedelta() has attributes that allow us to extract values from it.
print('offset days', offset.days)
print('offset seconds', offset.seconds)
print('offset microseconds', offset.microseconds)
```

```
offset days 1
offset seconds 20
offset microseconds 0
```

timedelta

```python
now = datetime.now()
print("Like Right Now: ", now)
```

```
Like Right Now:  2020-09-03 14:25:36.312817
```

```python
print("Future: ", now + offset)
print("Past: ", now - offset)
```

```
Future:  2020-09-04 14:25:56.312817
Past:  2020-09-02 14:25:16.312817
```

Other Functions

```
ts = pd.to_datetime('3/9/2020')
ts
```

```
Timestamp('2020-03-09 00:00:00')
```

```
aapl.head()
```

	Date	Open	High	Low	Close	Volume
0	2017-01-13	119.11	119.62	118.81	119.04	26111948
1	2017-01-12	118.90	119.30	118.21	119.25	27086220
2	2017-01-11	118.74	119.93	118.60	119.75	27588593
3	2017-01-10	118.77	119.38	118.30	119.11	24462051
4	2017-01-09	117.95	119.43	117.94	118.99	33561948

```
aapl.Date.dt.weekday_name.head()
```

```
0       Friday
1     Thursday
2    Wednesday
3      Tuesday
4       Monday
Name: Date, dtype: object
```

```
aapl.Date.dt.dayofyear.head()
```

```
0    13
1    12
2    11
3    10
4     9
Name: Date, dtype: int64
```

multiverse

Let’s Practice

Trends

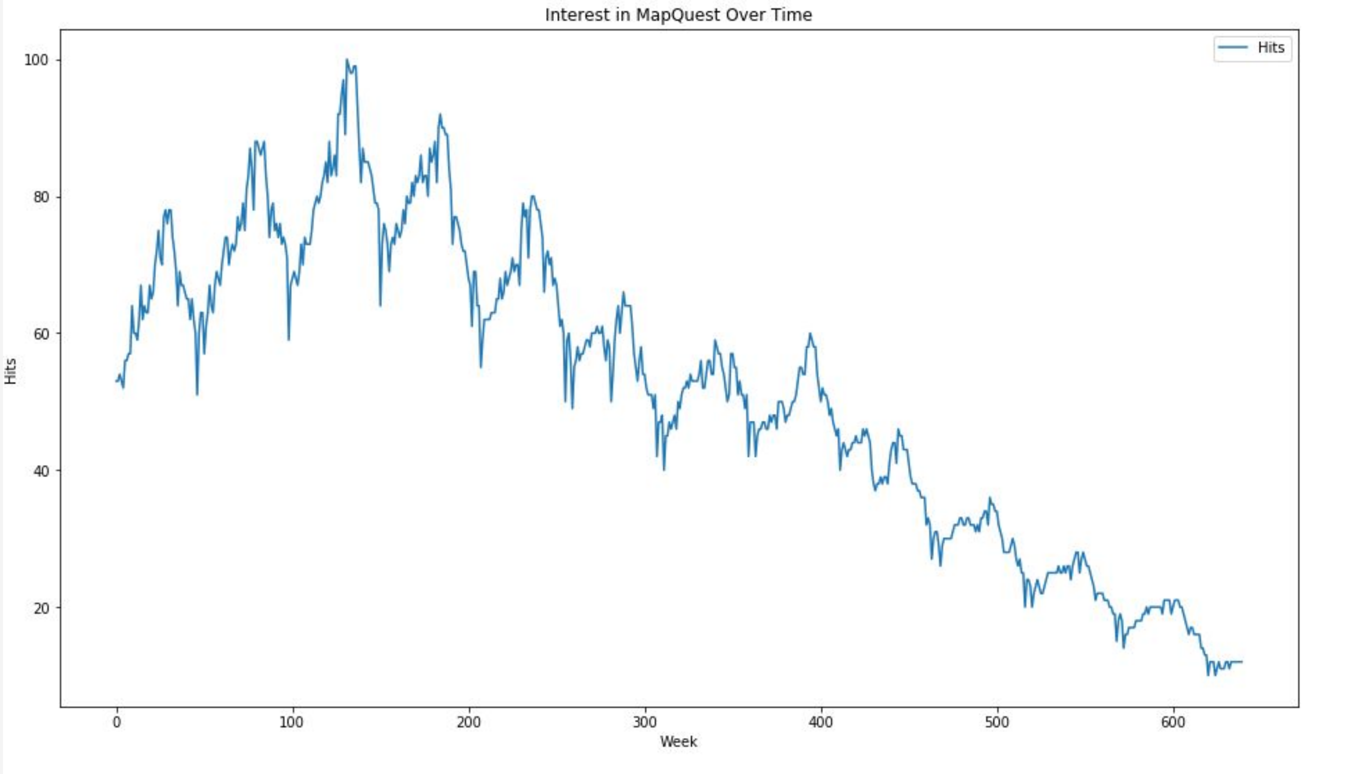
Trends

- A trend is any long-term change in the value we're measuring. Trends may "change direction," going from an increasing trend to a decreasing trend.
- Trends can only be measured within the scope of the data collected; there may be trends that are unmeasurable if the data are not complete.

An example of an upward trend:

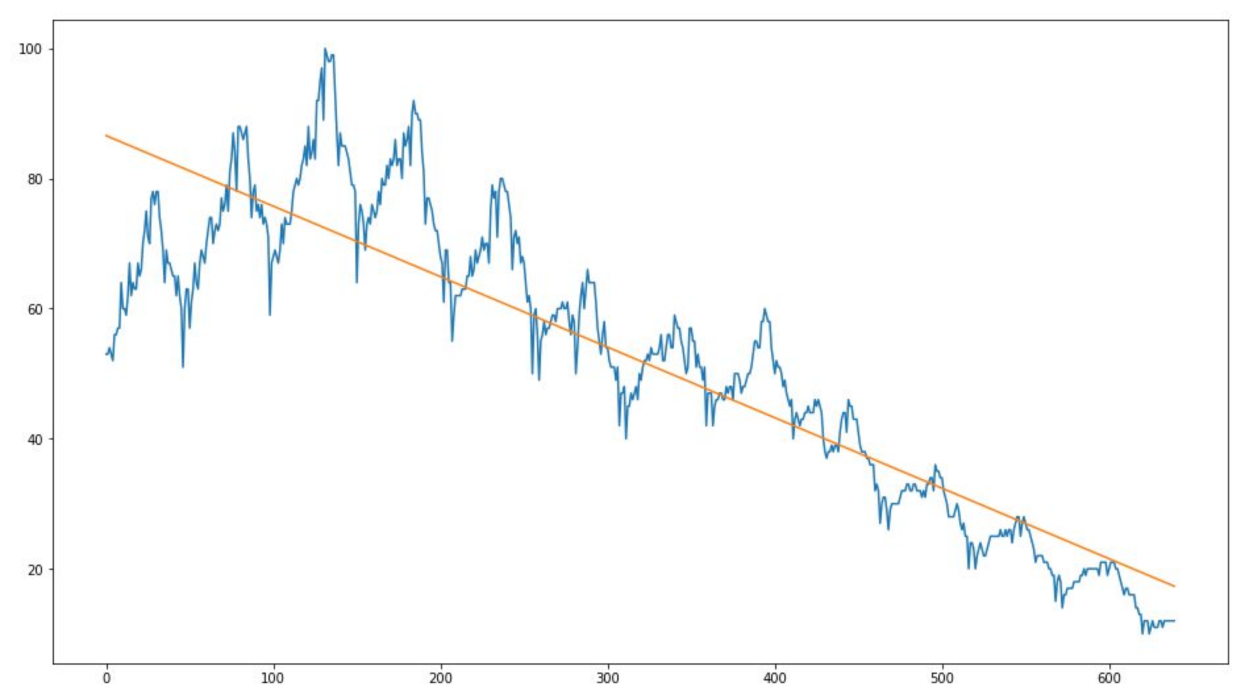
- When patterns repeat over *known, fixed periods of time* within a data set, we call this **seasonality**.
- A seasonal pattern exists when a series is influenced by factors related to the cyclic nature of time – i.e., time of month, quarter, year, etc. Seasonality is of a fixed and known period, otherwise it is not truly seasonality. Additionally, it must be either attributed to another factor or counted as a set of anomalous events in the data.

Trends


Interest in MapQuest Over Time
Hits
100
80
60
40
20
0
100
200
300
400
500
600
Week
Hits

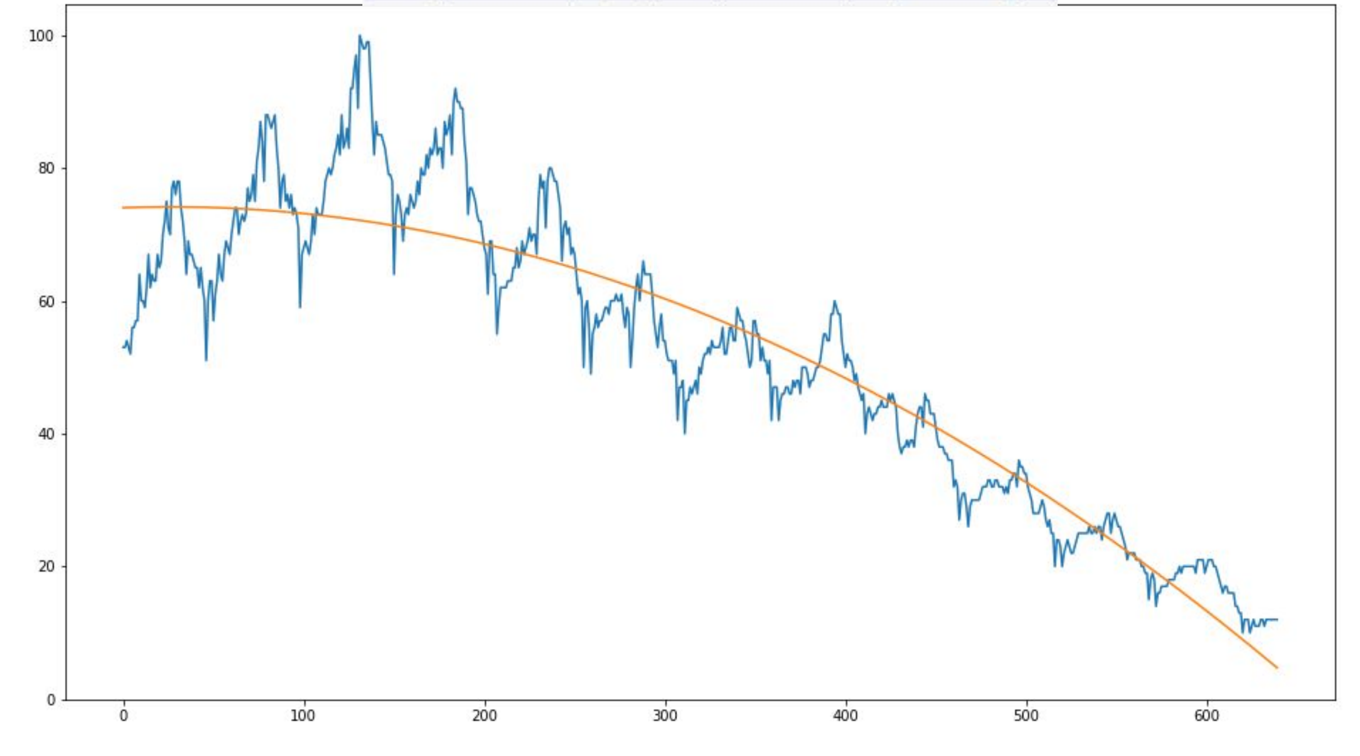
Trends

```
line_coef = np.polyfit(df.index,df['Hits'],1)
```



Trends

```
line_coef = np.polyfit(df.index,df['Hits'],2)
```



Aggregate Data

.resample()

Syntax	Calculates per:
‘A’	Year
‘M’	Month
‘W’	Week
‘D’	Day

Combine with .sum(), .mean(), .median() or .count()

```
data[['Sales']].resample('A').mean()
```

Aggregate Data

```
data[['Sales']].resample('A').mean()
```

Date	Sales
2013-12-31	5658.533675
2014-12-31	5833.290704
2015-12-31	5878.245380

```
data[['Sales']].resample('M').mean()
```

Date	Sales
2013-01-31	5211.555578
2013-02-28	5494.371397
2013-03-31	5820.349168
2013-04-30	5483.749836
2013-05-31	5364.127383
2013-06-30	5402.162960
2013-07-31	6042.062260
2013-08-31	5729.574049
2013-09-30	5322.988430
2013-10-31	5429.258788
2013-11-30	5864.601614
2013-12-31	6703.618140
2014-01-31	5431.875799
2014-02-28	5731.091512
2014-03-31	5584.257312
2014-04-30	5815.993333
2014-05-31	5632.670534
2014-06-30	5681.526188
2014-07-31	5999.403381

multiverse

Let’s Practice

multiverse

Rolling Statistics

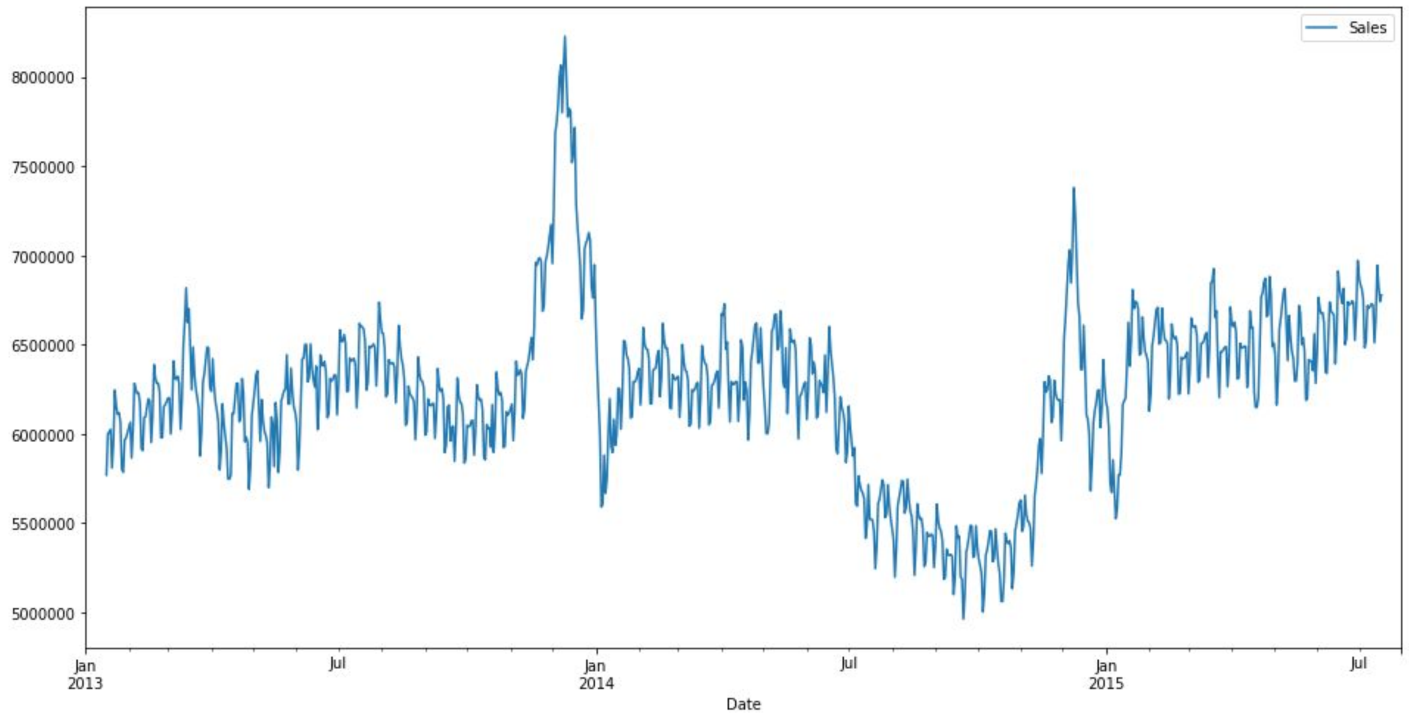
We be Rolling

```
daily_store_sales.rolling( window=3, center=True).mean().head()
```

	Sales
Date	
2013-01-01	NaN
2013-01-02	4.464961e+06
2013-01-03	6.645534e+06
2013-01-04	6.312789e+06
2013-01-05	4.244817e+06

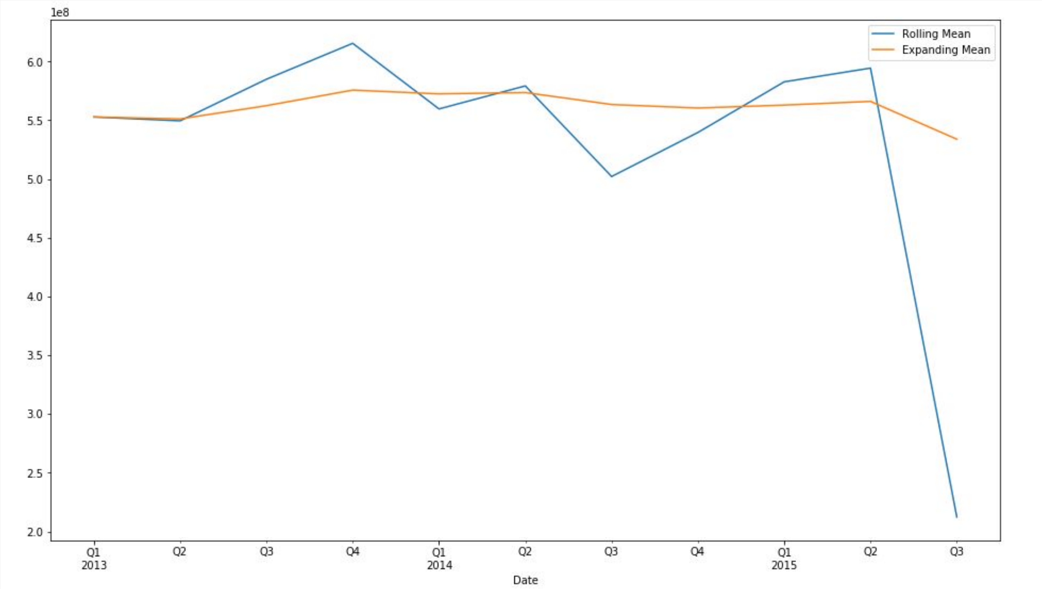
We be Rolling

```
daily_store_sales.rolling(window=30, center=True).mean().plot();
```



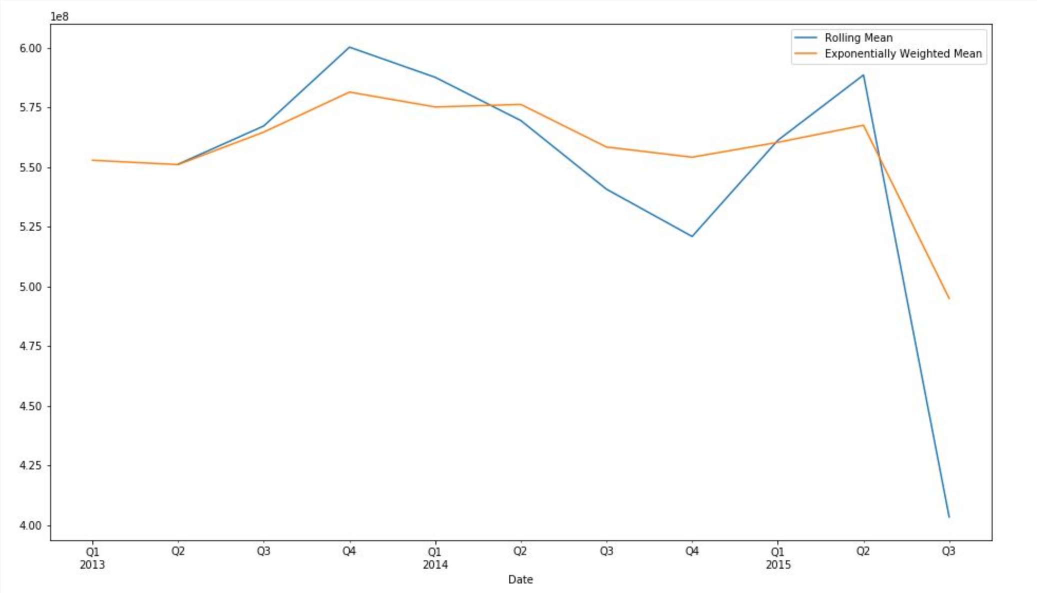
Expanding Mean

```
rolling_mean = data.Sales.resample('Q').sum().rolling(window=1, center=False).mean()
expanding_mean = data.Sales.resample('Q').sum().expanding().mean()
```



Exponentially Weighted Mean

```
rolling_mean = data.Sales.resample('Q').sum().rolling(window=2, center=True).mean()
exp_weighted_mean = data.Sales.resample('Q').sum().ewm(span=10).mean()
```



multiverse

Let’s Practice

multiverse

Shifting

Shifting

```
store1_data.head()
```

Date	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Year	Month
2015-07-31	1	5	5263	555	1	1	0	1	2015	7
2015-07-30	1	4	5020	546	1	1	0	1	2015	7
2015-07-29	1	3	4782	523	1	1	0	1	2015	7
2015-07-28	1	2	5011	560	1	1	0	1	2015	7
2015-07-27	1	1	6102	612	1	1	0	1	2015	7

```
shifted_forward = store1_data.shift(1)
shifted_forward.head()
```

Date	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Year	Month
2015-07-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015-07-30	1.0	5.0	5263.0	555.0	1.0	1.0	0	1.0	2015.0	7.0
2015-07-29	1.0	4.0	5020.0	546.0	1.0	1.0	0	1.0	2015.0	7.0
2015-07-28	1.0	3.0	4782.0	523.0	1.0	1.0	0	1.0	2015.0	7.0
2015-07-27	1.0	2.0	5011.0	560.0	1.0	1.0	0	1.0	2015.0	7.0

multiverse

Let’s Practice

multiverse

Autocorrelation

Autocorrelation

Autocorrelation

```
store1_data['Sales'].autocorr(lag=1)
```

```
-0.1273251433914022
```

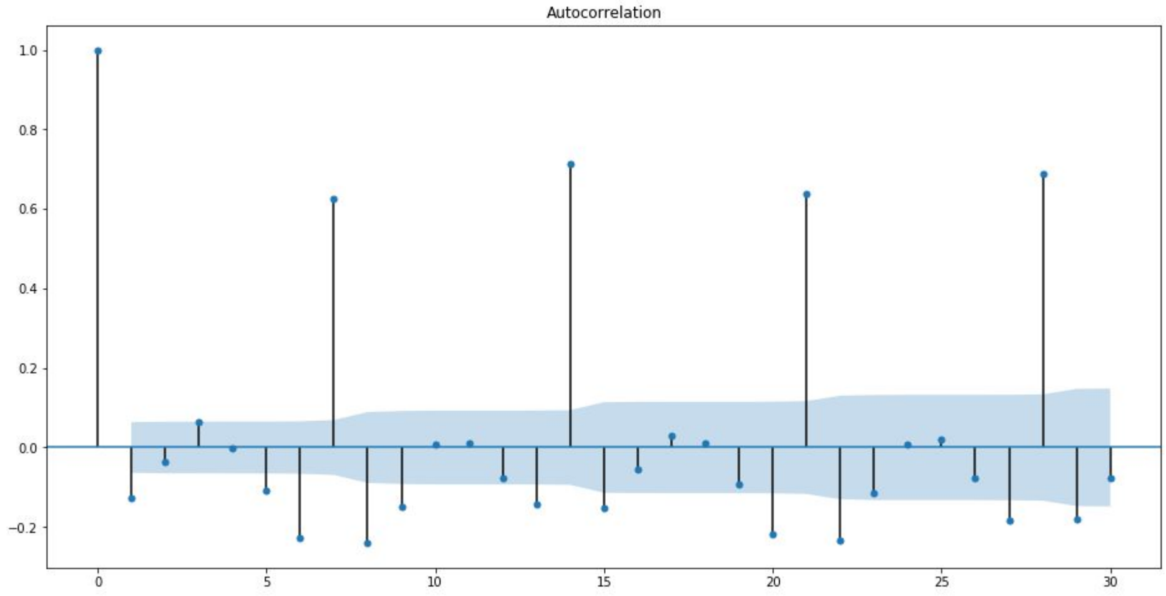
```
store1_data['Sales'].autocorr(lag=7)
```

```
0.630719243284029
```

Autocorrelation

```
from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf
```

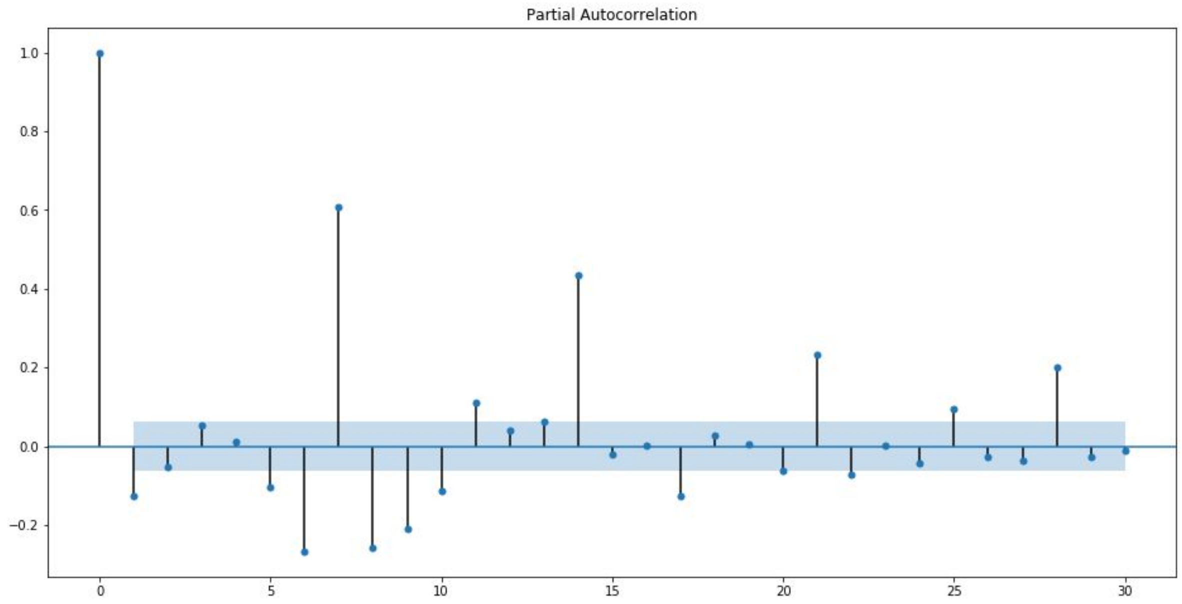
```
plot_acf(store1_data.Sales.values, lags=30)
plt.show()
```



Partial Autocorrelation

```
from statsmodels.tsa.stattools import pacf
from statsmodels.graphics.tsaplots import plot_pacf
```

```
plot_pacf(store1_data.Sales.values, lags=30)
plt.show()
```



Autocorrelation Problems

Models like linear regression analysis require that there is little or no autocorrelation in the data. That is, linear regressions requires that the residuals are independent of one another. So far, we have assumed all of the independent values in our models have been independent, but this is unlikely with time series data, because consecutive data points are often related which means that they will often contain autocorrelation.

What are some problems that could arise when using autocorrelated data with a linear model?

➔ Estimated regression coefficients are still unbiased, but they no longer have the minimum variance property.

➔ The MSE may seriously underestimate the true variance of the errors.

➔ The standard error of the regression coefficients may seriously underestimate the true standard deviation of the estimated regression coefficients.

➔ Statistical intervals and inference procedures are no longer strictly applicable

Let’s Practice

multiverse

Decomposition

Decomposition

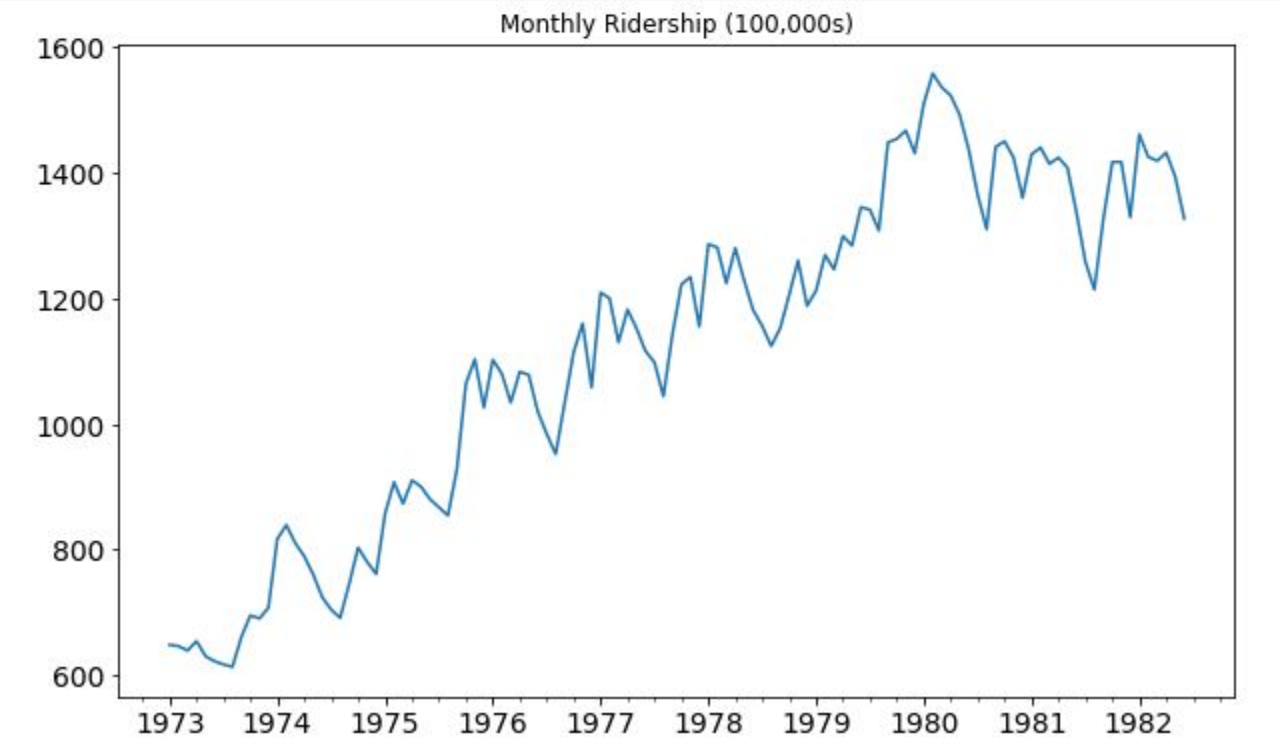
Splitting a time series into several components is useful for both understanding the data and diagnosing the appropriate forecasting model. Each of these components will represent an underlying pattern.

➔ **Trend:** A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. Sometimes, we will refer to a trend “changing direction” when, for example, it might go from an increasing trend to a decreasing trend.

➔ **Seasonal:** A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, or day of the week). Seasonality is always of a fixed and known period.

➔ **Residual:** The leftover or error component.

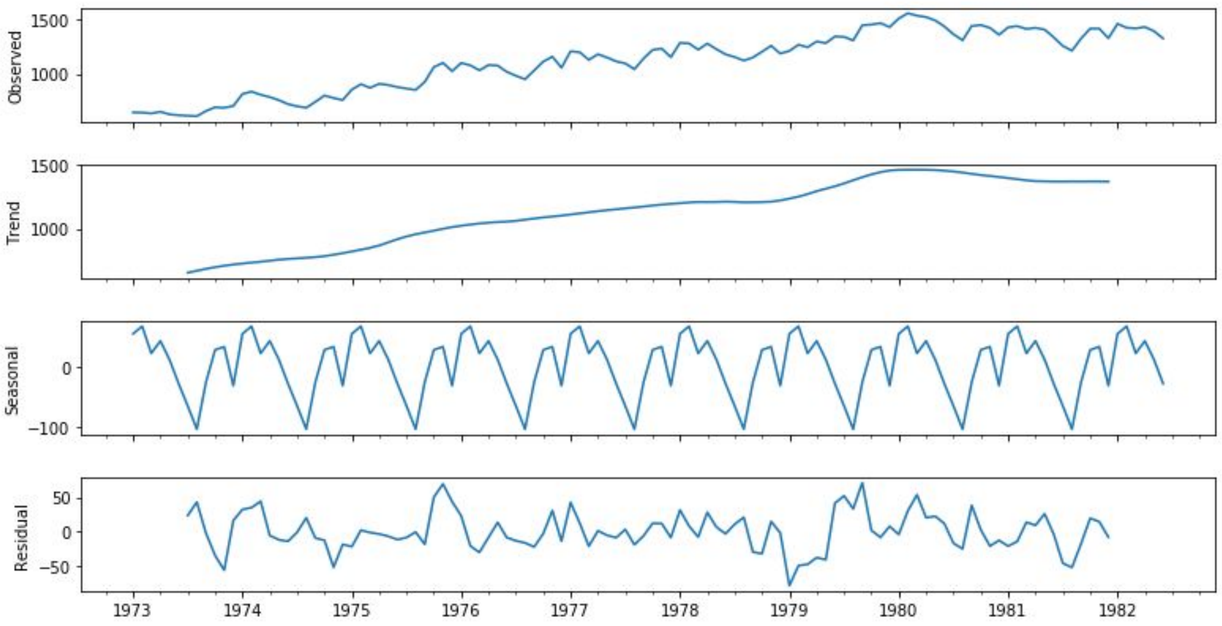
Decomposition



Decomposition

```
decomposition = seasonal_decompose(bus.riders, freq=12)
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(12, 6)
```

```
<matplotlib.figure.Figure at 0x1c1f016e10>
```



Let's Practice

multiverse

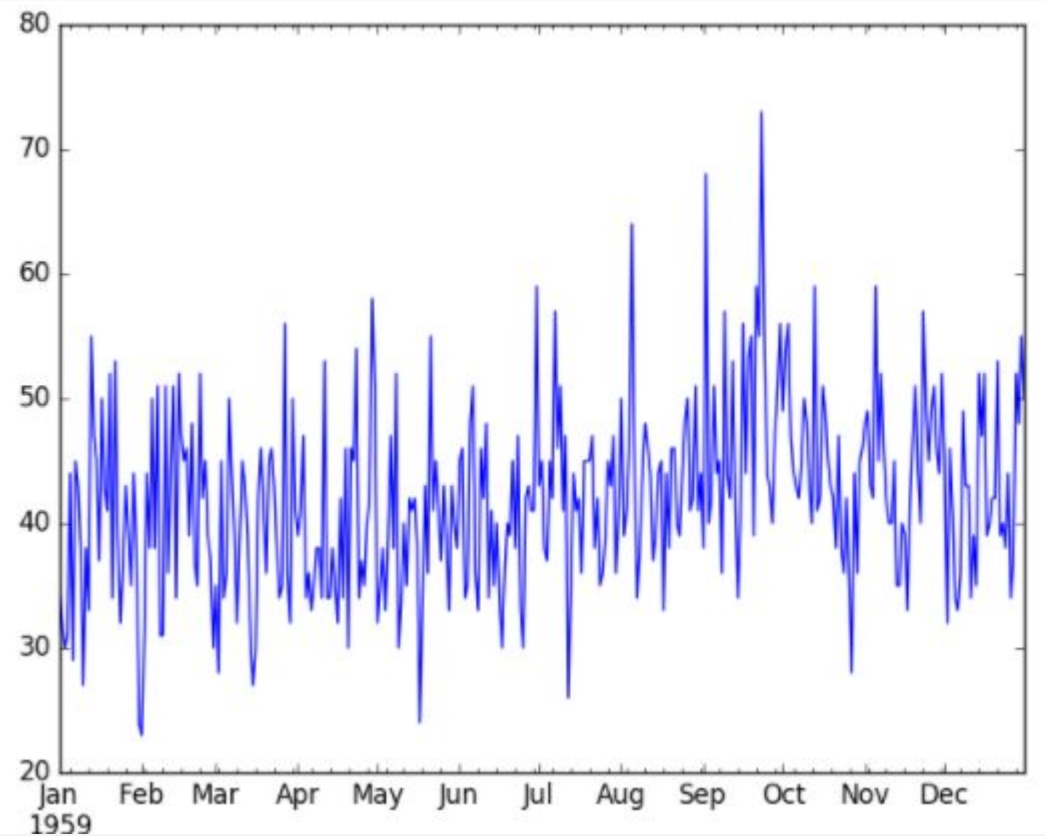
Stationary

Stationarity

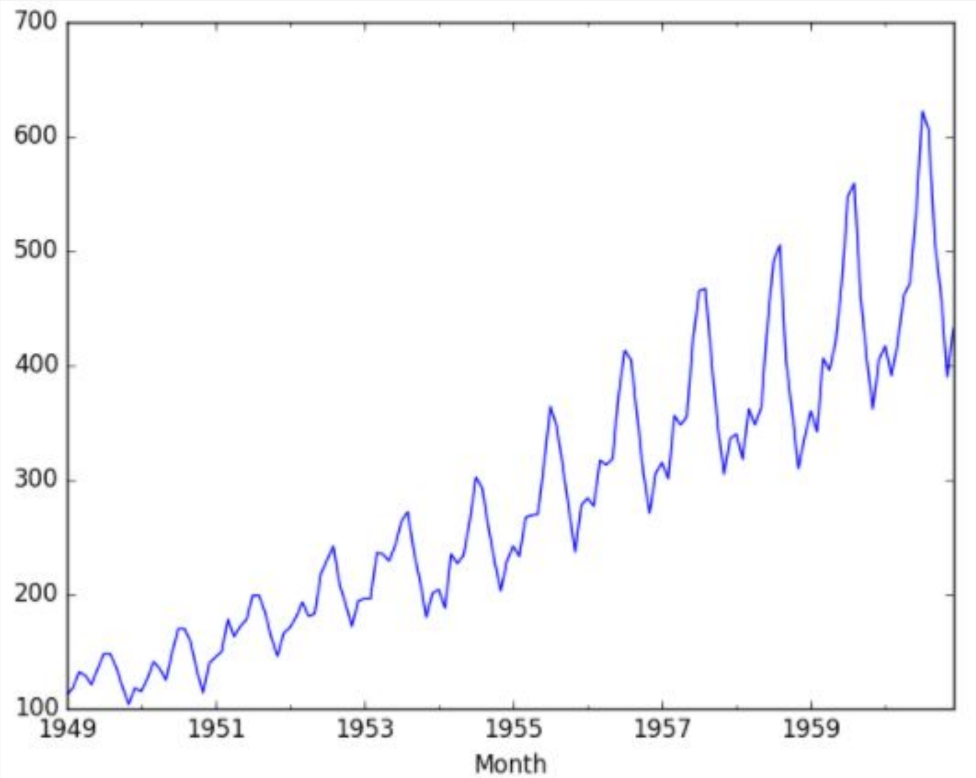
Properties of a Stationary Time Series

- ➔ The mean is not a function of time, but constant (e.g. rolling or expanded mean will be constant)
- ➔ The variance is not a function of time, but constant (homoscedastic)
- ➔ The autocorrelation should be constant over all lags

Stationarity



Stationarity



Checking for Stationarity

Properties of a Stationary Time Series

- ➔ **Plot the time series** – if you can observe trends (seasonal or otherwise) it is not stationary
- ➔ **Plot rolling statistics** – will allow us to observe if the moving average varies over time

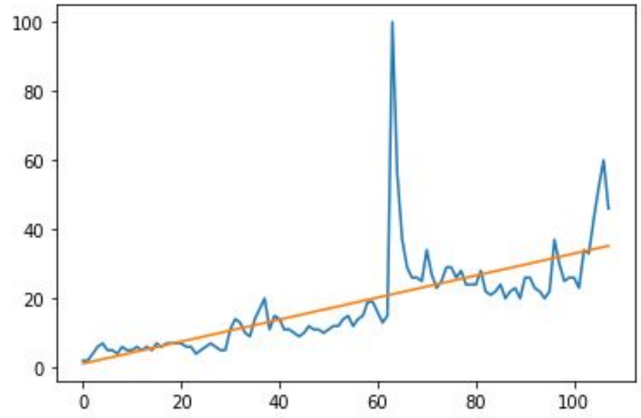
Making a Series Stationary

Properties of a Stationary Time Series

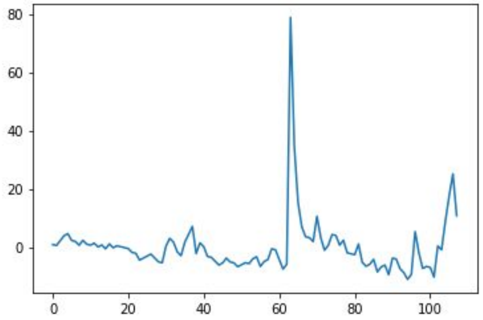
- **Detrending** – we can remove trends in our data. The easiest way to do this is by plotting a trend line and then making a new series from the residuals (remember the decompose function we looked at earlier can do this)
- **Differencing** – instead of using our actual values, we build a series out of the differences between consecutive values. This will force the mean over time to be 0 (so constant)

Detrending

```python
import numpy as np
import matplotlib.pyplot as plt
drones = pd.read_csv('data/gt_drone_racing.csv', header=1)
drones.columns = ['week','drone_racing_interest']
coefs = np.polyfit(drones.index, drones.drone_racing_interest,1)
lineFunction = np.poly1d(coefs)
plt.plot(drones.index, drones.drone_racing_interest, drones.index, lineFunction(drones.index));
```

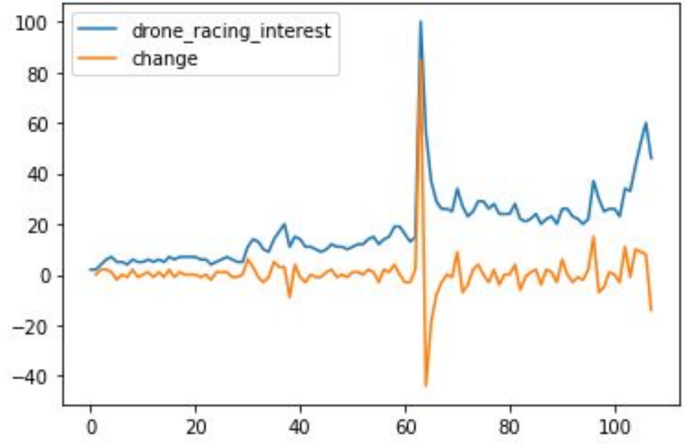



```python
import scipy.signal
ffty = scipy.signal.detrend(drones.drone_racing_interest.values)
plt.plot(drones.index, ffty);
```



Differencing

```
drones['change'] = drones.drone_racing_interest.diff(1);
drones.plot();
```



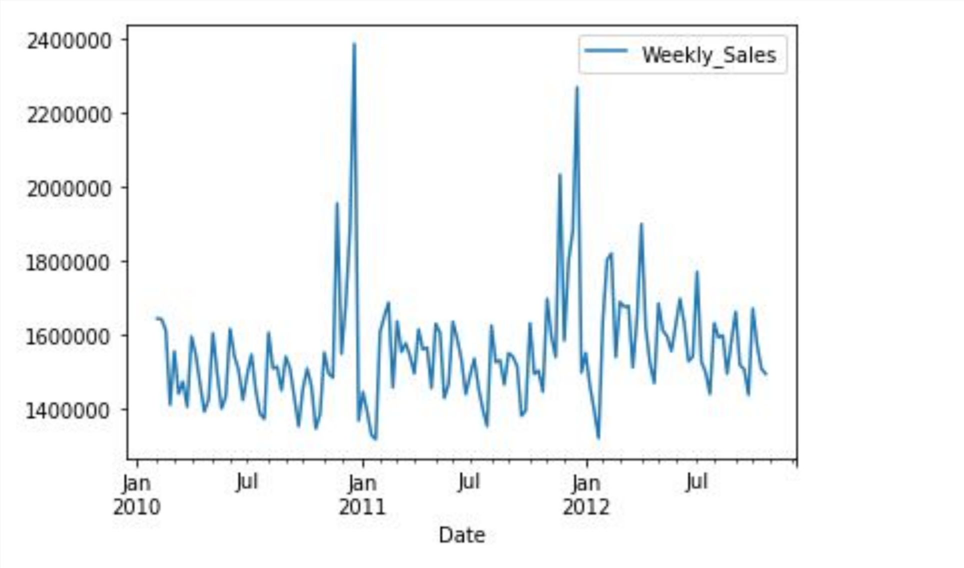
multiverse

Let’s Practice

multiverse

Forecasting

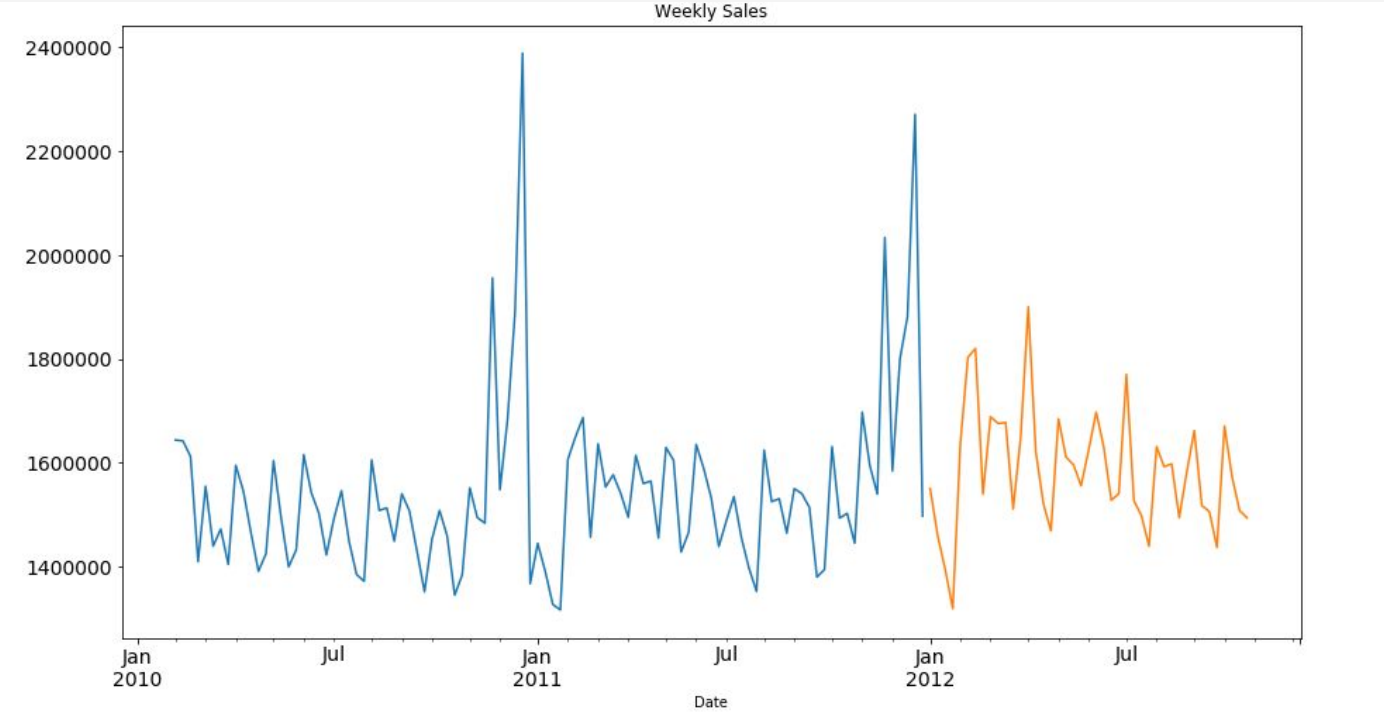
The Data



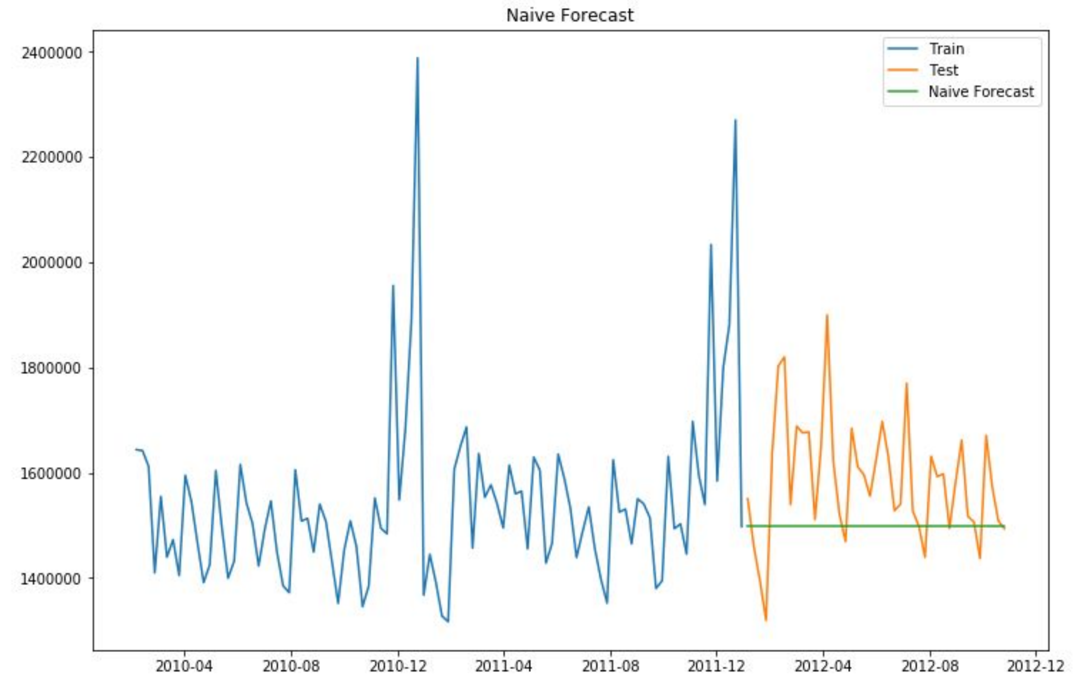
Weekly_Sales
2400000
2200000
2000000
1800000
1600000
1400000
Jan 2010
Jul
Jan 2011
Jul
Jan 2012
Jul
Date

The Data

```
train = store1_sales['2010': '2011']
test = store1_sales['2012']
```



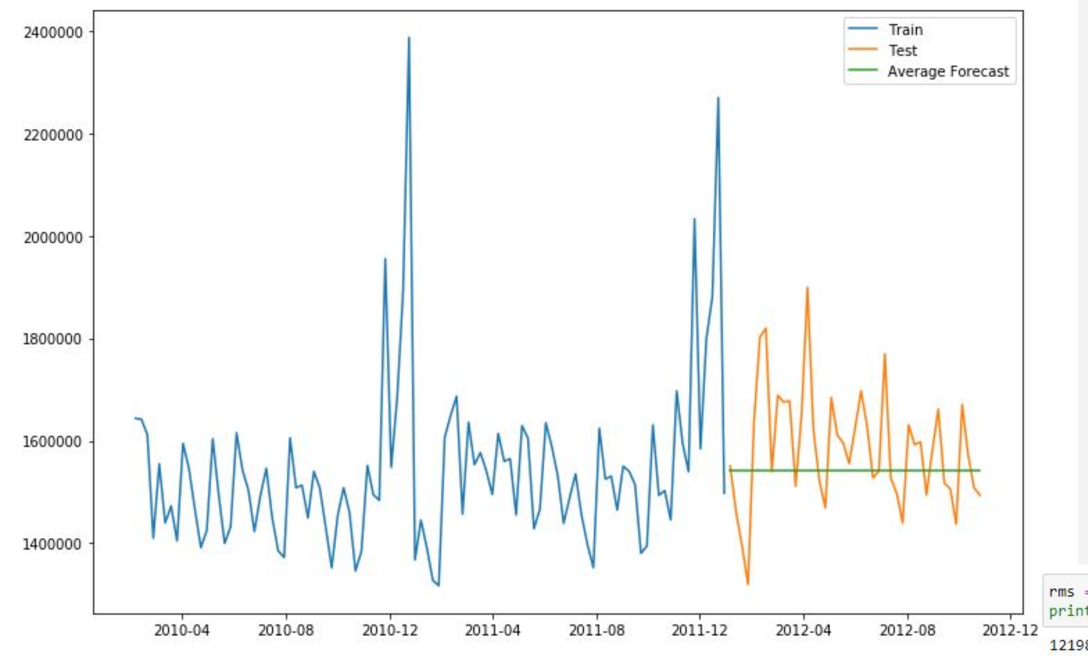
Naive Approach




```python
from sklearn.metrics import mean_squared_error
from math import sqrt
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat.naive))
print(rms)
```

```
144192.4921506529
```

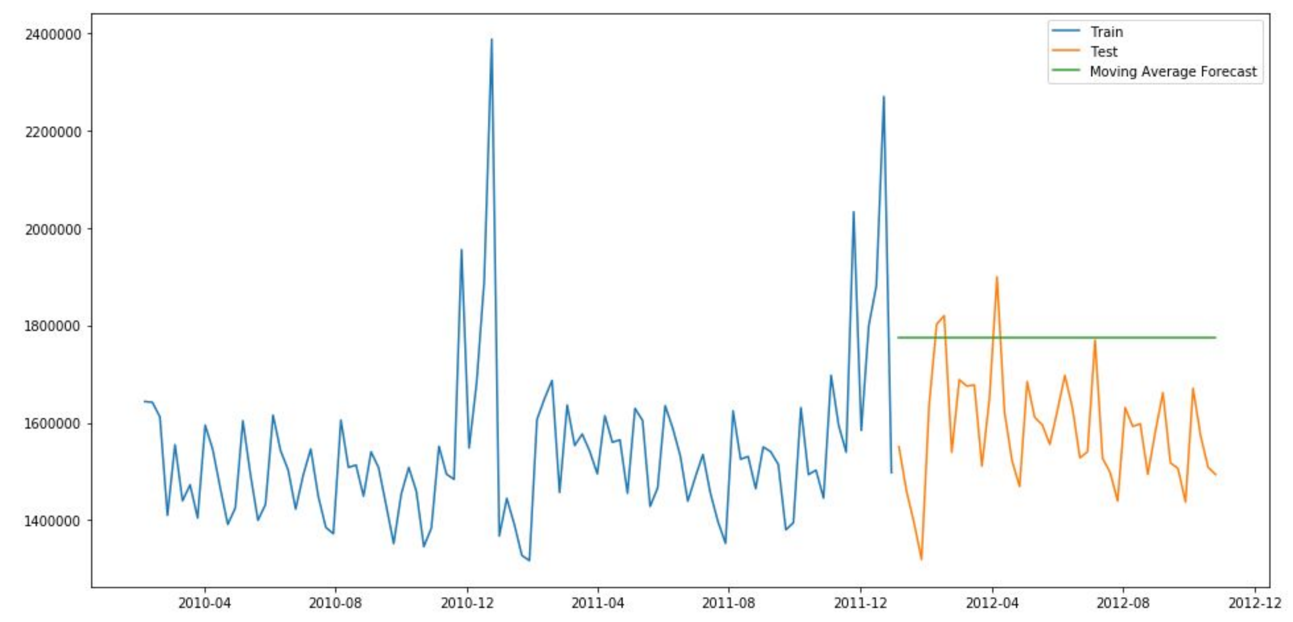
Simple Average




```
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat_avg.avg_forecast))
print(rms)
```

```
121981.78781090611
```

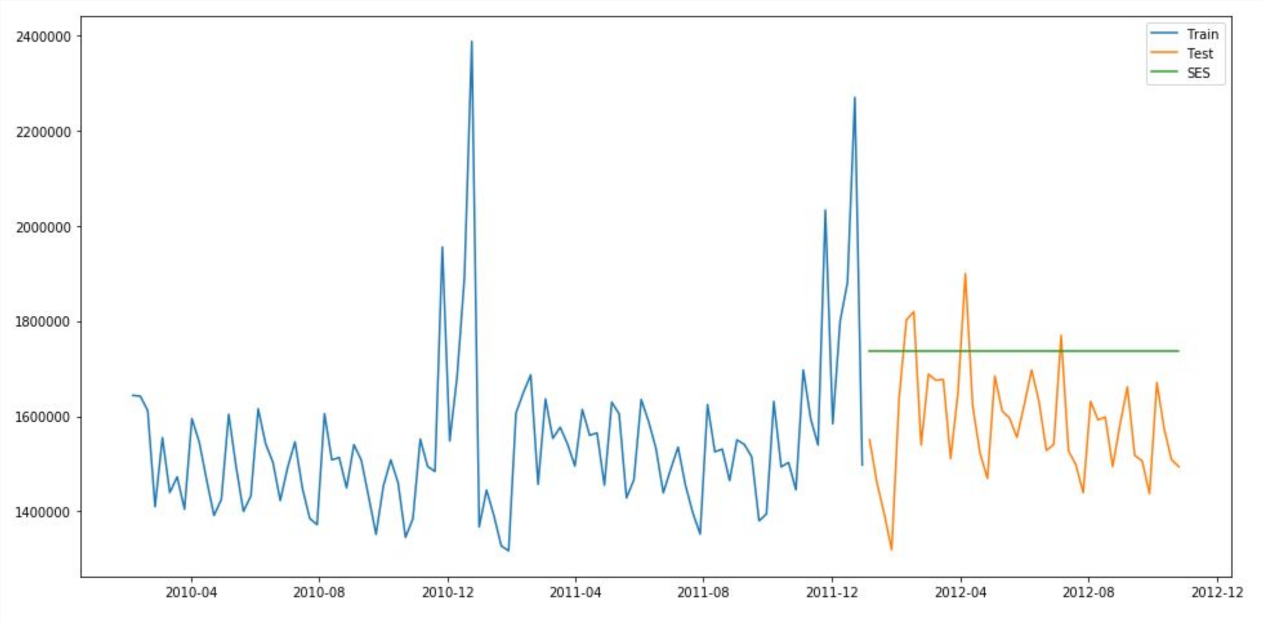
Moving Average




```
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat_avg.moving_avg_forecast))
print(rms)
```

```
220538.43986419958
```

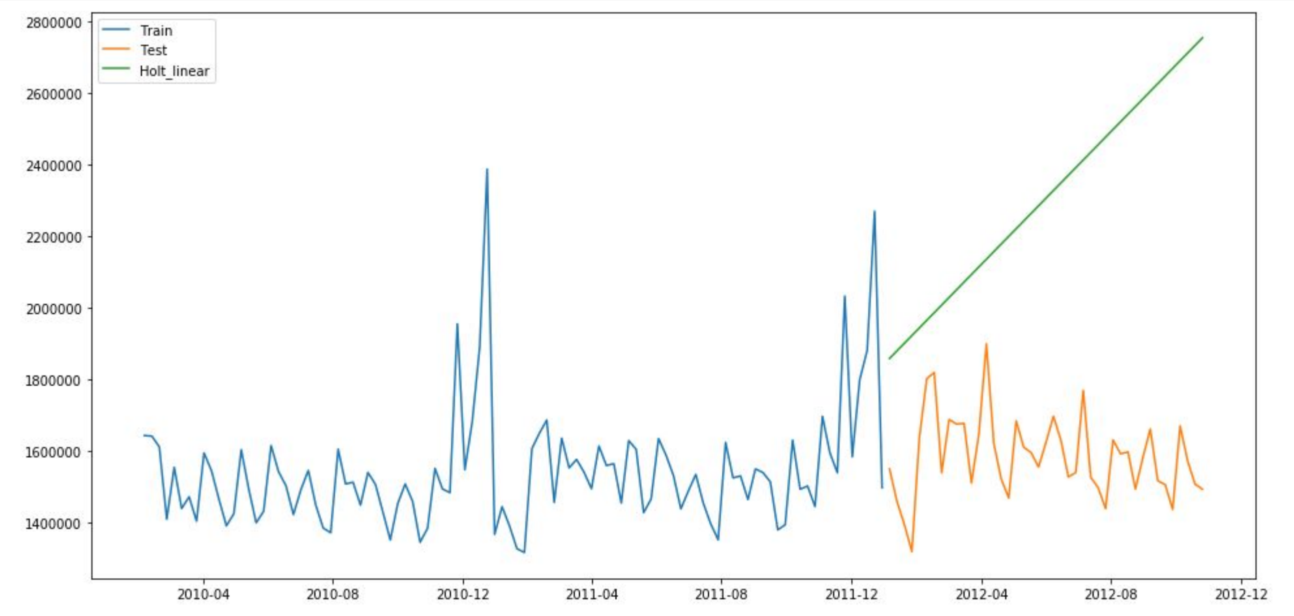
Simple Exponential Smoothing




```
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat_avg.SES))
print(rms)
```

```
188369.92416964
```

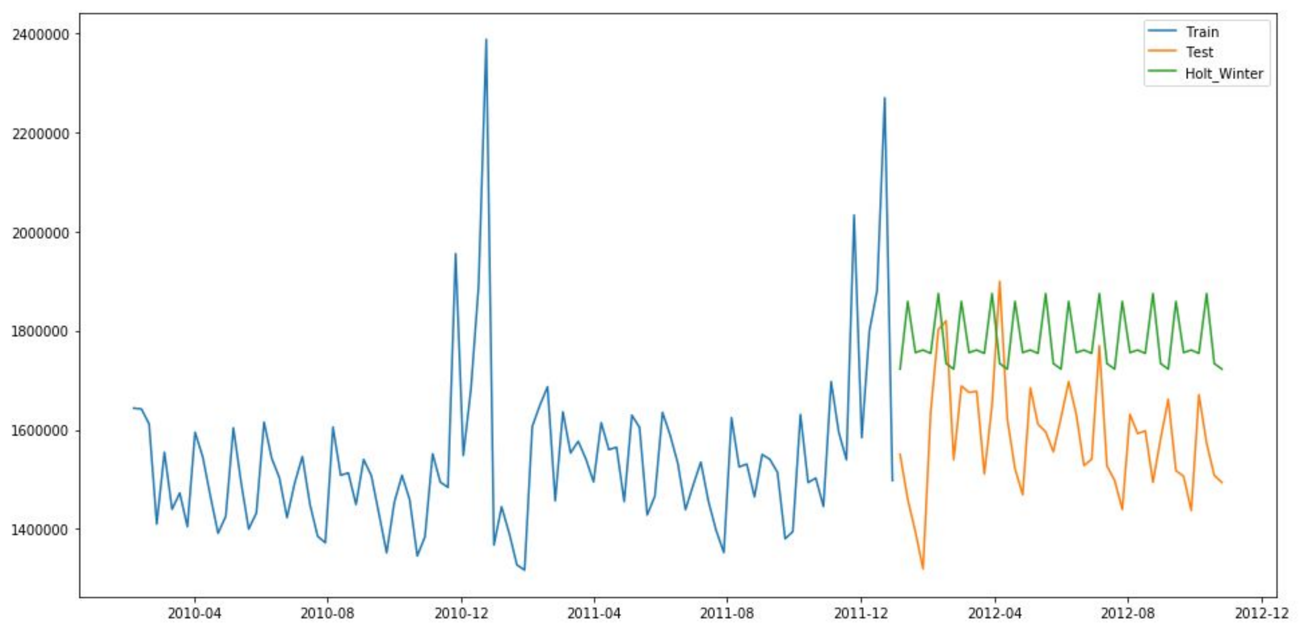
Holt's Linear Trend




```
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat_avg.Holt_linear))
print(rms)
```

781856.6060260109

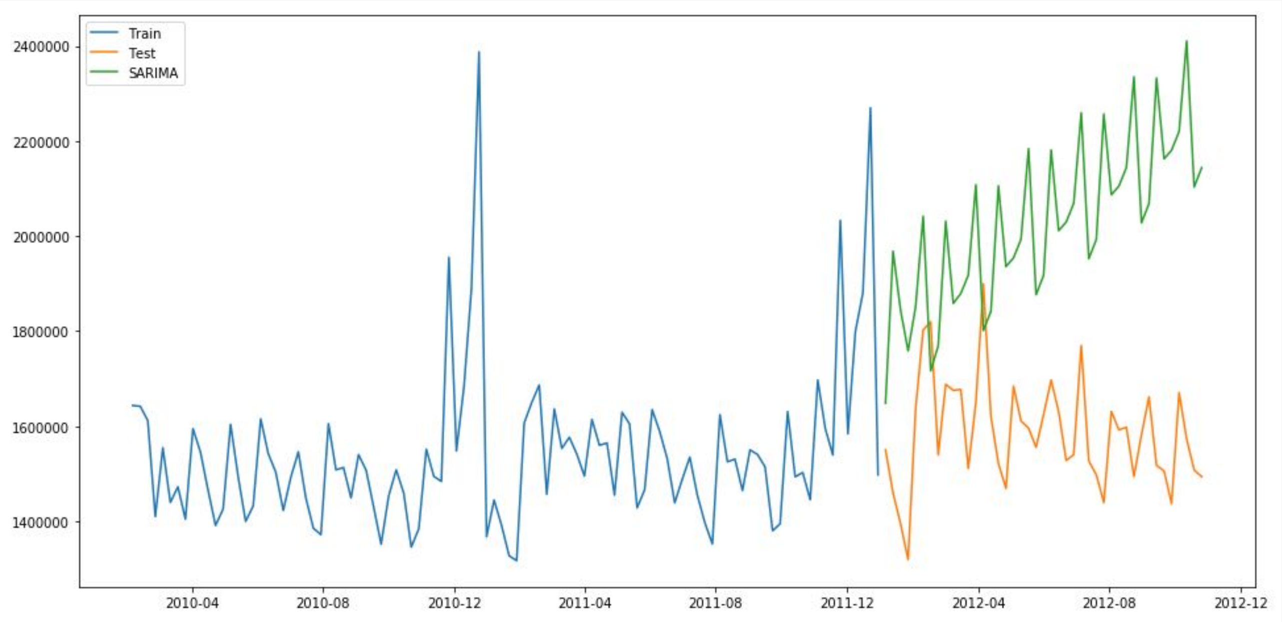
Holt-Winters Method




```
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat_avg.Holt_Winter))
print(rms)
```

```
229011.82036689422
```

ARIMA




```
rms = sqrt(mean_squared_error(test.Weekly_Sales, y_hat_avg.SARIMA))
print(rms)
```

```
489261.9105214533
```

Let’s Practice