Pingboard

YOU Can Predict the Future

or, an introduction to forecasting techniques

Outline

- Ordinary Least Squares (OLS)
- Polynomial Fit
- Autoregressive Integrated Moving Average (ARIMA) Models
- Prophet, from Facebook
- How good is my model?

Bike Share Data

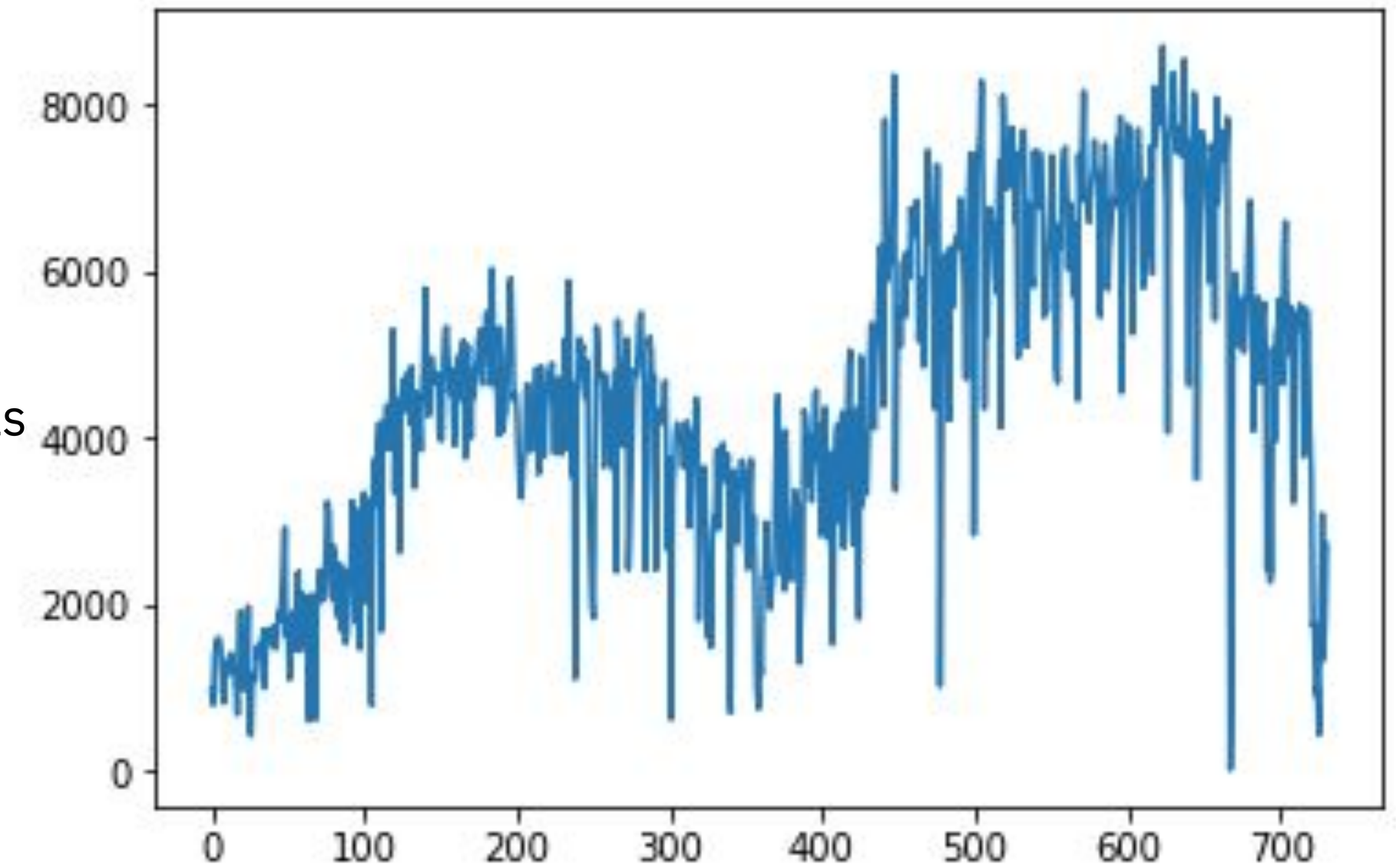
Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble detectors and background knowledge", Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg, doi:10.1007/s13748-013-0040-3.

https://www.kaggle.com/contactprad/bike-share-daily-data

via

https://toolbox.google.com/datasetsearch

photo by Bernard Spragg https://flic.kr/p/yRrj8m



Ordinary Least Squares

photo by Paul Wasneski https://flic.kr/p/29Jdkhe

```
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(data['x'],data['cnt'])
```

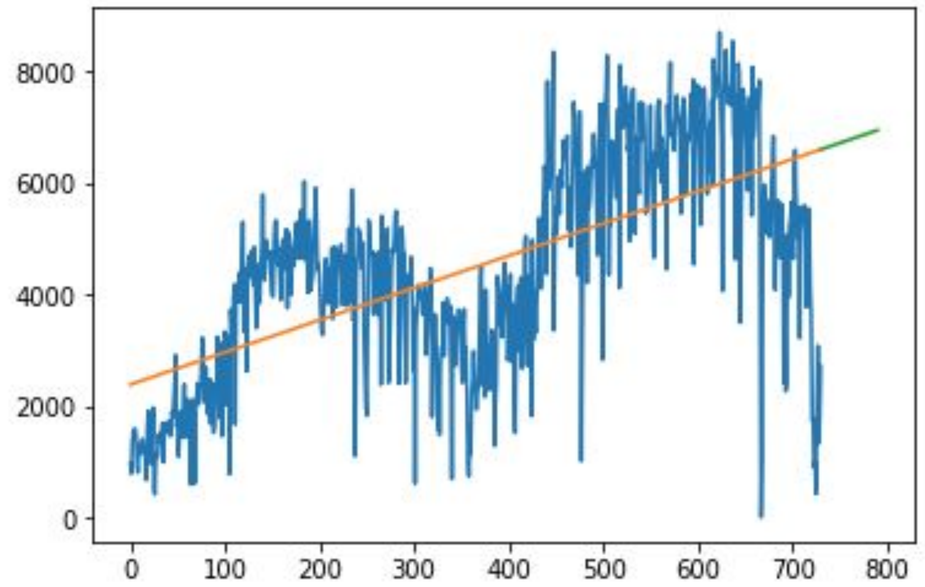
```
slope
```

```
5.7688183284113785
```

```
line = slope*data['x']+intercept
line2 = slope*xout+intercept
```

```
plt.plot(data['x'],data['cnt'])
plt.plot(data['x'],line)
plt.plot(xout, line2)
```

```
[<matplotlib.lines.Line2D at 0x112b0b4e0>]
```



Polynomial Fit

photo by Dan Keck https://flic.kr/p/2f5wG3N

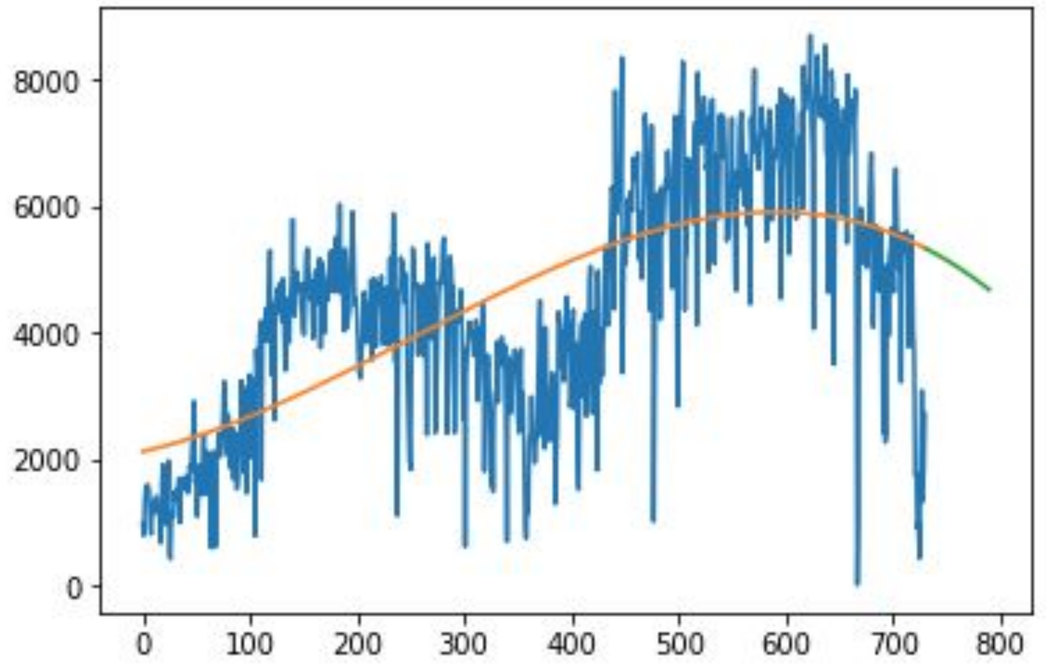
```
import numpy as np
```

```
z = np.polyfit(data['x'],data['cnt'], 3)
```

```
p = np.poly1d(z)
```

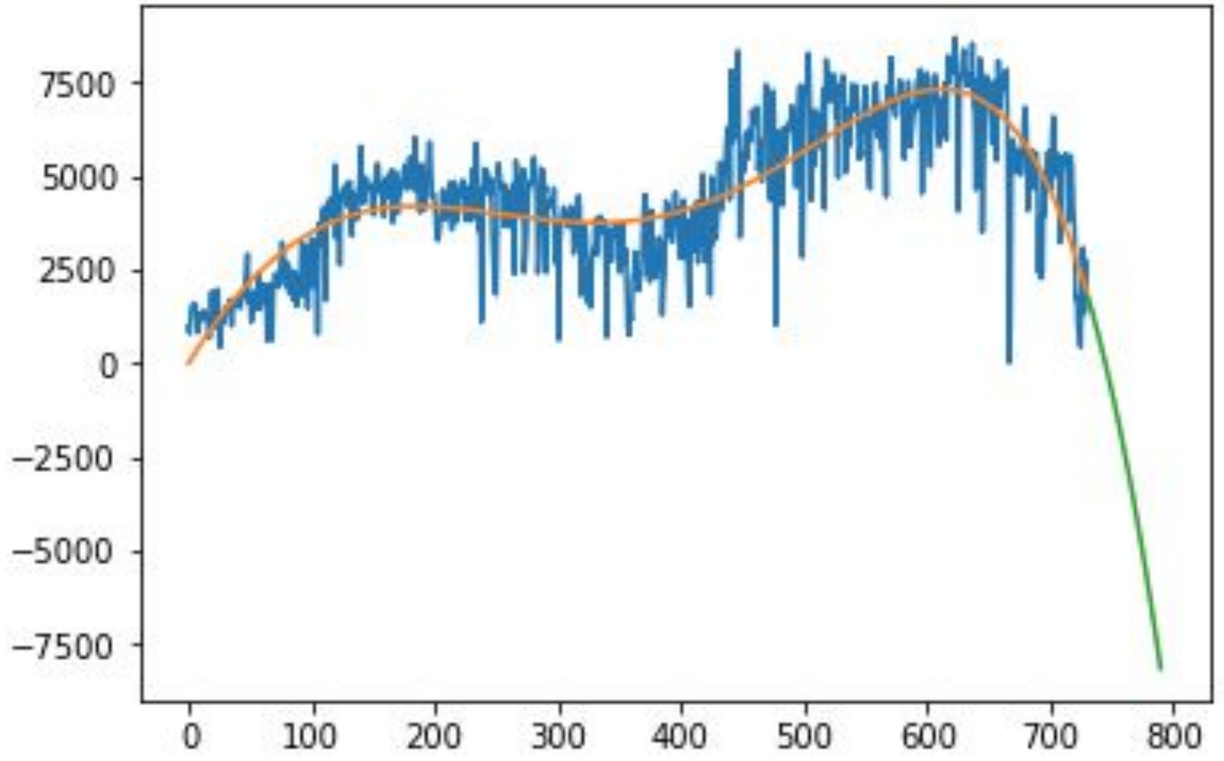
```
plt.plot(data['x'],data['cnt'])
plt.plot(data['x'],p(data['x']))
plt.plot(xout,p(xout))
```

```
[<matplotlib.lines.Line2D at 0x112d3ac18>]
```



```
z = np.polyfit(data['x'],data['cnt'], 5)
p = np.poly1d(z)
plt.plot(data['x'],data['cnt'])
plt.plot(data['x'],p(data['x']))
plt.plot(xout,p(xout))
```

```
[<matplotlib.lines.Line2D at 0x1176db320>]
```



Autoregressive Integrated Moving Average Models

photo by Paul Wasneski https://flic.kr/p/26P5M2p

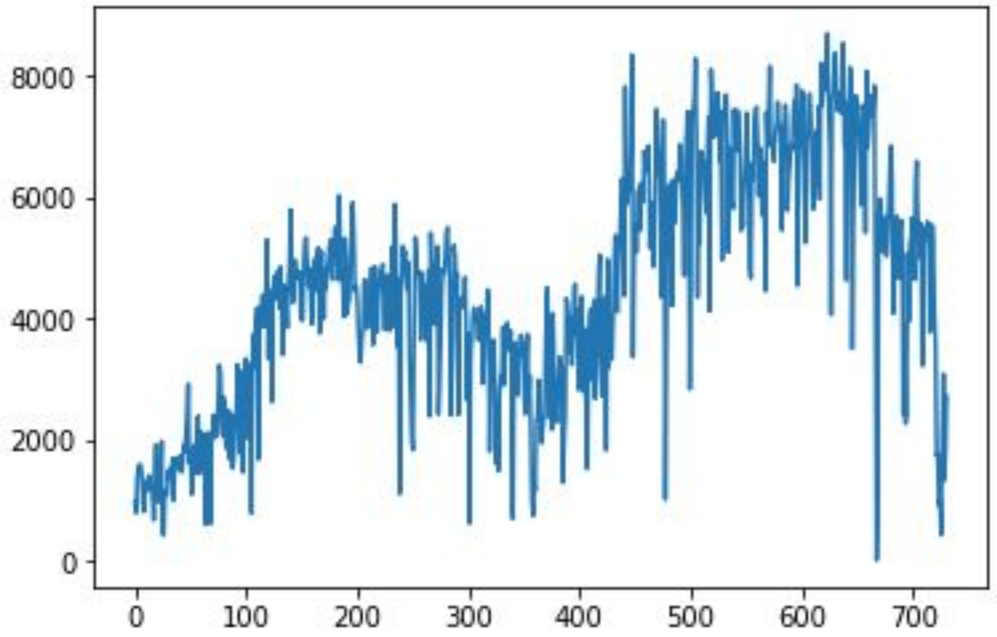
Stationarity

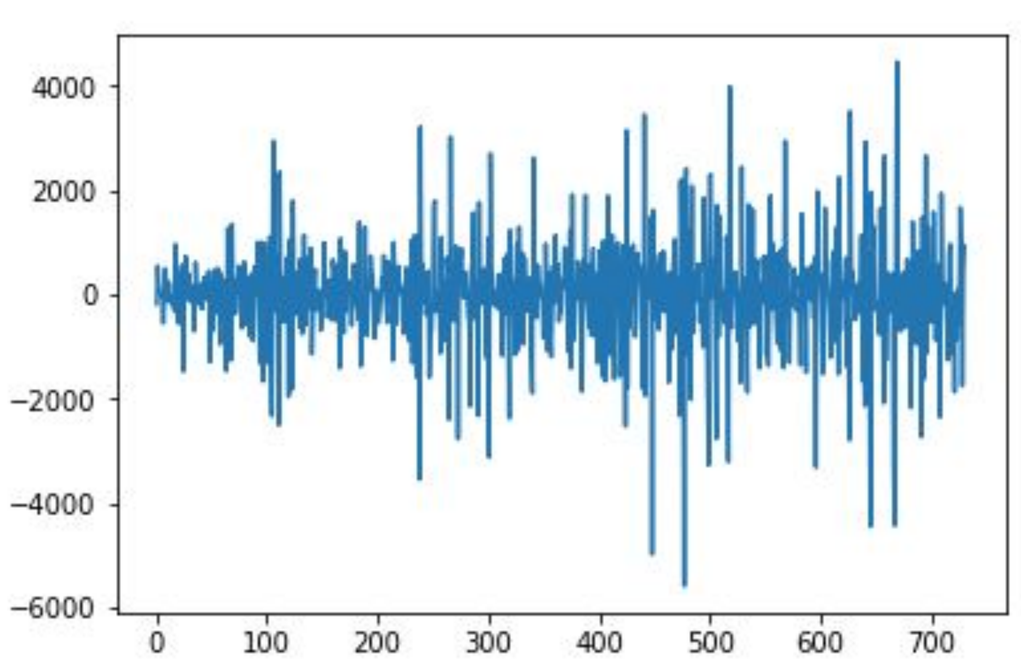
Stable, Predictable, Converging to long-run Equilibrium

It means the AR or MA model can be solved - the matrix can be inverted.

We often **difference** the time series - look at Z(t) = Y(t)-Y(t-1) to get closer to stationarity

The number of times we do this is called **the Order of Integration** and it's the **I** in the ARIMA model.





Autoregressive (AR)

Y(t) = f(Y(t-i))

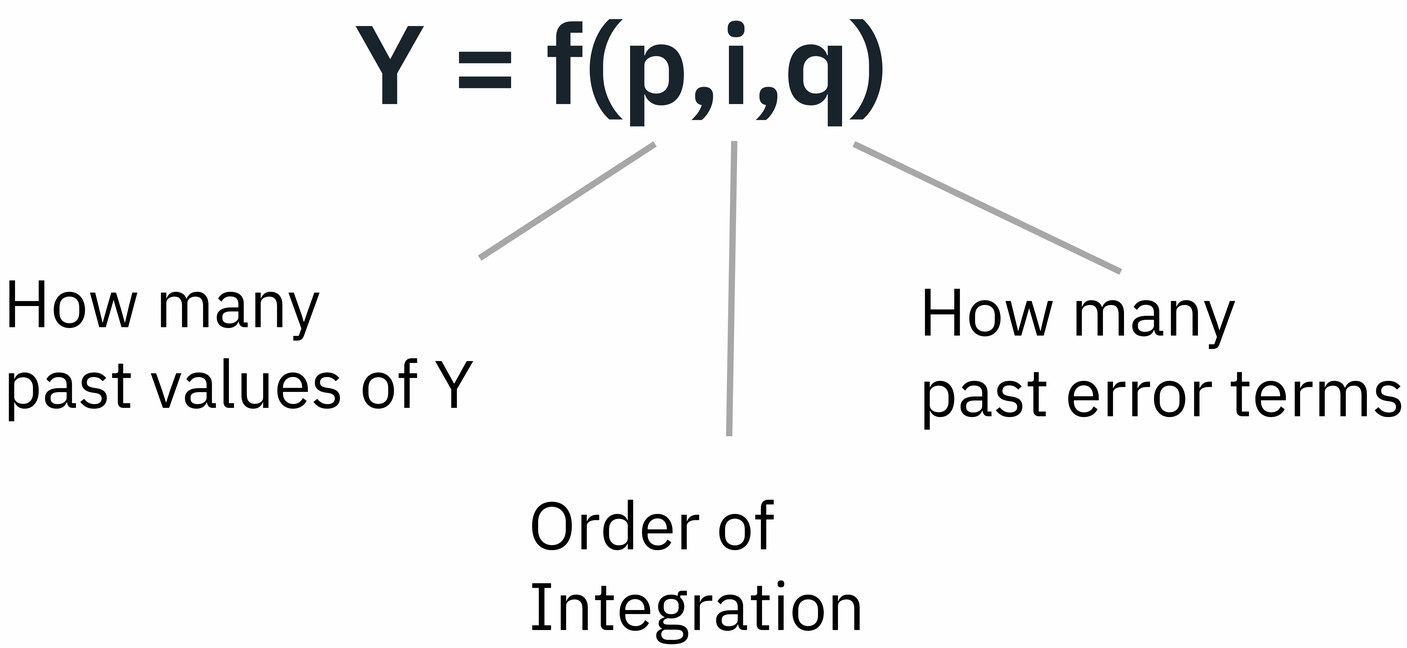
The future is a function of past values

Moving Average (MA)

Y(t) = f(error(t-i))

The future is a function of past errors

Order of an ARIMA model



Autocorrelation Function

Corr(Y(t), Y(t-i))

Should show q significant lags for a Moving Average model, quickly decay for a Autoregressive model

Partial Autocorrelation Function

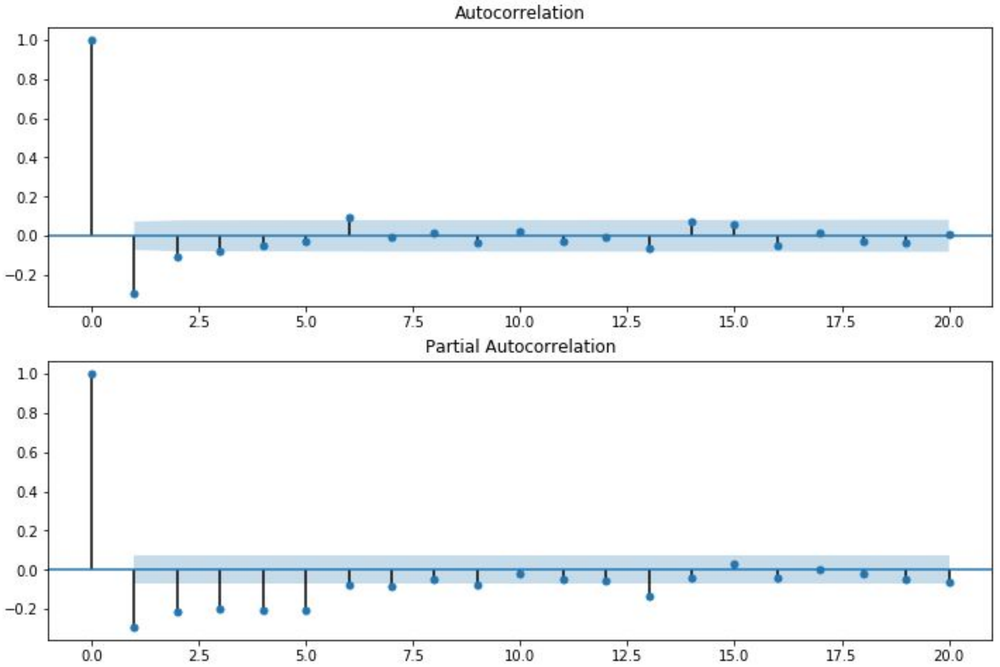
Correlation of Y(t) with the residual of the OLS regression on the lags

Corr(Y(t), Y(t-1)-B*Y(t-1))

Should show p significant lags for a Autoregressive model, quickly decay for a Moving Average model

```
#https://machinelearningmastery.com/time-series-forecast-case-study-python-monthly-armed-robberies-boston/
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib import pyplot

#pandas has a built in differencing function, but it leaves an NA in the first position
series2 = data['cnt'].diff()
pyplot.figure(figsize=(12,8))
pyplot.subplot(211)
plot_acf(series2[1:], lags=20, ax=pyplot.gca())
pyplot.subplot(212)
plot_pacf(series2[1:], lags=20, ax=pyplot.gca())
pyplot.show()
```



```
#https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
from statsmodels.tsa.arima_model import ARIMA
```

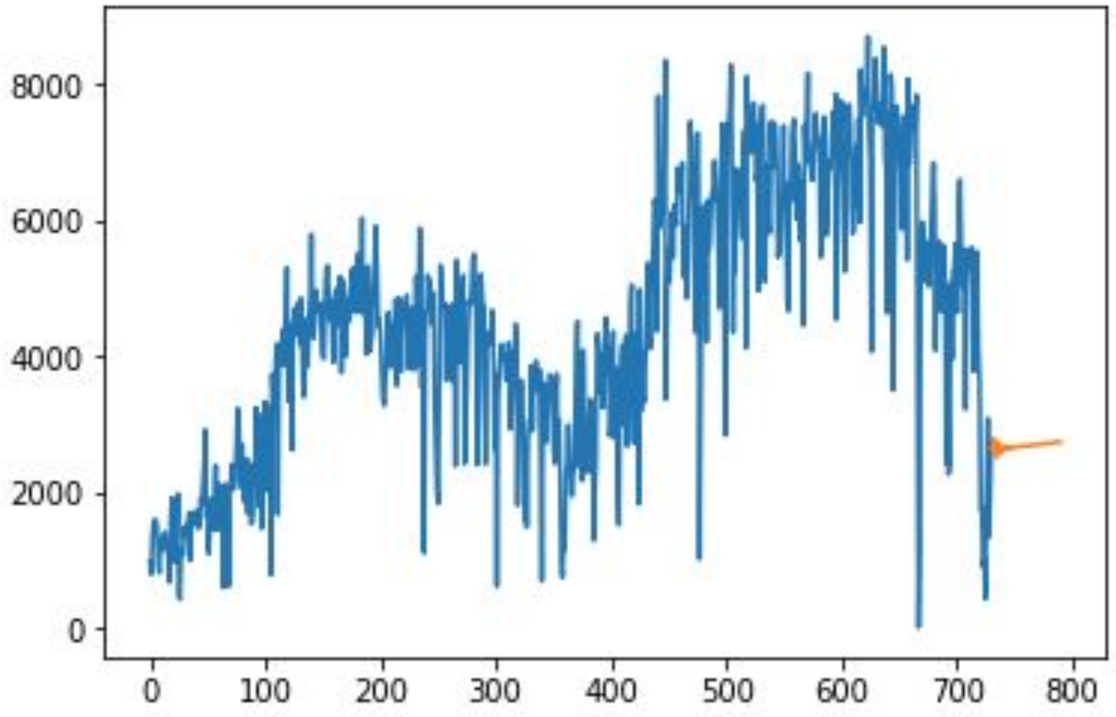
```
model = ARIMA(data['cnt'], order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
output = model_fit.forecast(60)
```

```
plt.plot(data['x'],data['cnt'])
plt.plot(xout,output[0])
```

```
[<matplotlib.lines.Line2D at 0x1134f7080>]
```

```
==============================================================================
                 coef    std err          z      P>|z|
------------------------------------------------------------------------------
const          1.3613     12.739      0.107      0.915
ar.L1.D.cnt   -0.4776      0.036    -13.180      0.000
ar.L2.D.cnt   -0.3969      0.039    -10.219      0.000
ar.L3.D.cnt   -0.3491      0.039     -8.864      0.000
ar.L4.D.cnt   -0.2918      0.039     -7.515      0.000
ar.L5.D.cnt   -0.2055      0.036     -5.666      0.000
```



ARMA models

All AR(p) models can be approximated by an MA model of some order

and

All MA(q) models can be approximated by an AR model of some order

Simpler Models are Better

By combining AR and MA models into ARMA (fitting both the past values and the past errors) we can often get a model that has fewer things to estimate.

But how do we really choose models?

We use **Information Criteria** to choose the best model - these are functions of how much of the variation is explained by the model which also account for how complex the model is. (Simpler models are better)

We'll let the computer fit several models and compare the AIC (Akaike Information Criteria) to choose the best one

```
#https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.arma_order_select_ic.html
import statsmodels.api as sm
```

```
res = sm.tsa.arma_order_select_ic(series2[1:], ic=['aic'], trend='nc')
```

```
/usr/local/lib/python3.7/site-packages/scipy/signal/signaltools.py:1341: FutureWarning: Using a non-
r multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futur
terpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a differe
  out_full[ind] += zi
/usr/local/lib/python3.7/site-packages/scipy/signal/signaltools.py:1344: FutureWarning: Using a non-
r multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futur
terpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a differe
  out = out_full[ind]
/usr/local/lib/python3.7/site-packages/scipy/signal/signaltools.py:1350: FutureWarning: Using a non-
r multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futur
terpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a differe
  zf = out_full[ind]
```

```
res.aic_min_order
```

```
(1, 1)
```

Notice this is an ARMA order select function, so we have to run it against the differenced (stationary) data.

```
model = ARIMA(data['cnt'], order=(1,1,1))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
output = model_fit.forecast(60)
plt.plot(data['x'],data['cnt'])
plt.plot(xout,output[0])
```

```
[<matplotlib.lines.Line2D at 0x114452ba8>]
```

```
==============================================================
                 coef    std err          z      P>|z|
--------------------------------------------------------------
const          2.1305      5.919      0.360      0.719
ar.L1.D.cnt    0.3592      0.042      8.497      0.000
ma.L1.D.cnt   -0.8904      0.019    -46.490      0.000
```

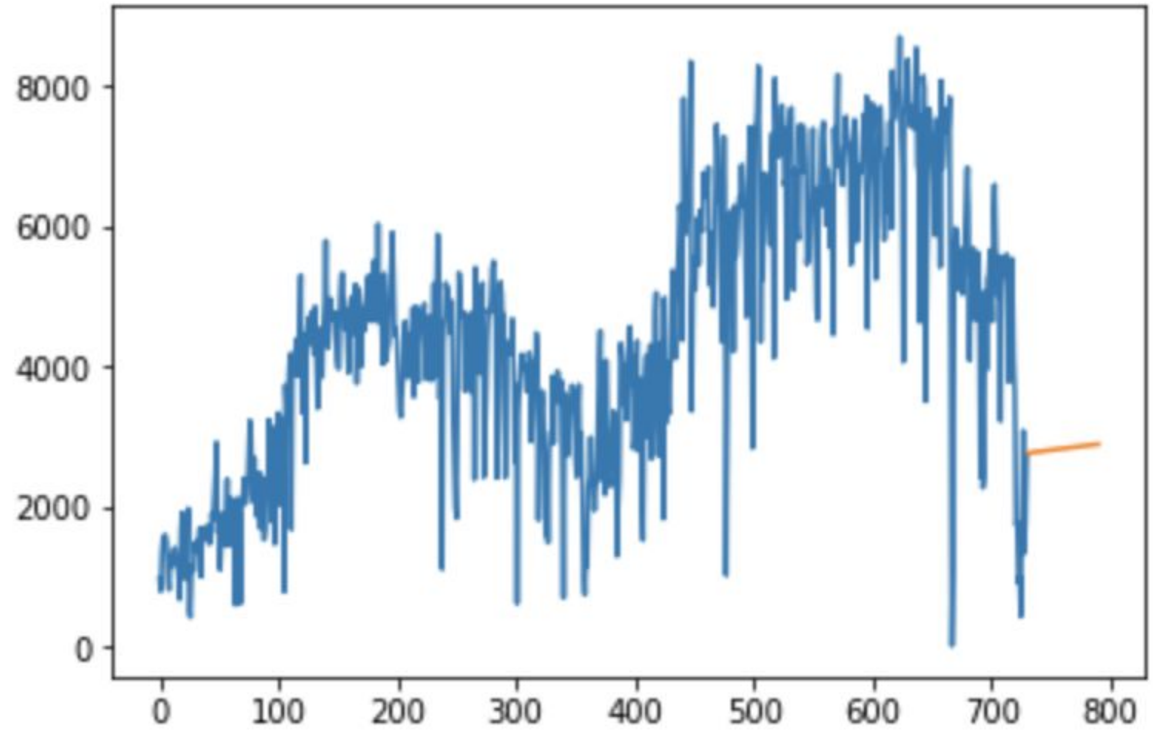


photo by Paul Wasneski
https://flic.kr/p/VP7kUd

Model	AIC value	Model Parameters
ARIMA(5,1,0)	12072.617	6
ARIMA(1,1,1)	12051.781	3

Prophet

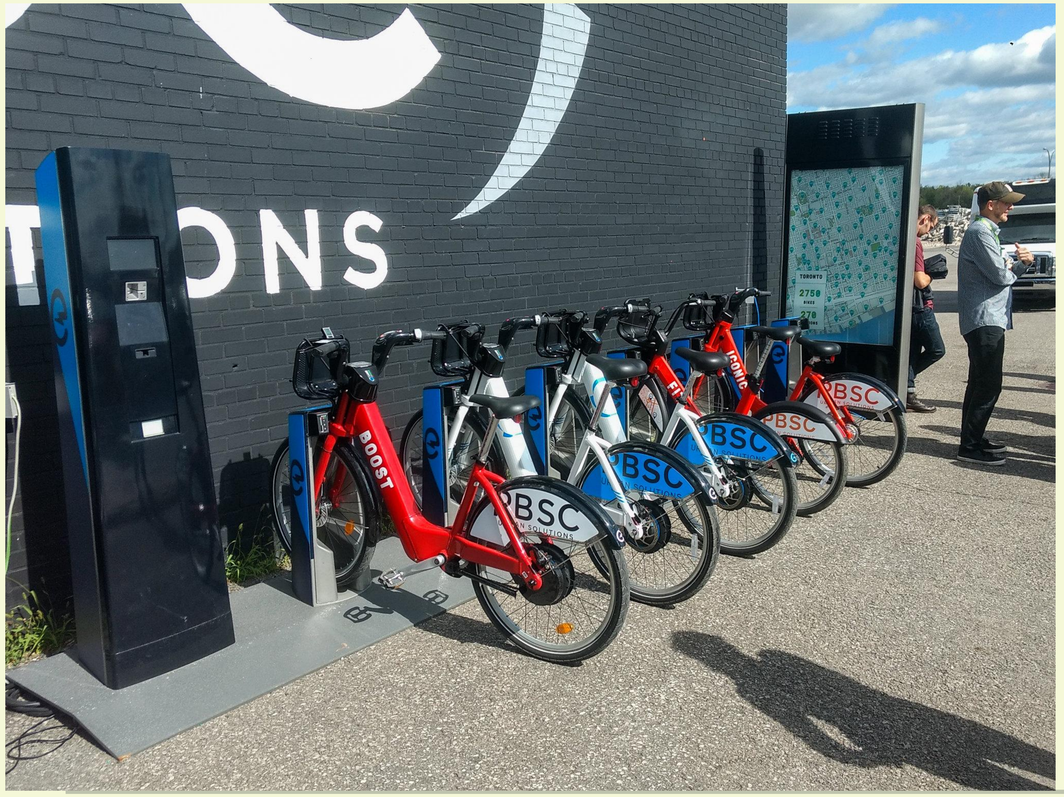


photo by Paul Wasneski https://flic.kr/p/XKxXA9

https://facebook.github.io/prophet/

PROPHET

Docs GitHub

Forecasting at scale.

Prophet is a forecasting procedure implemented in R and Python. It is fast and provides completely automated forecasts that can be tuned by hand by data scientists and analysts.

INSTALL PROPHET | GET STARTED IN R | GET STARTED IN PYTHON | READ THE PAPER

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet is open source software released by Facebook's Core Data Science team. It is available for download on CRAN and PyPI.

```
from fbprophet import Prophet
```

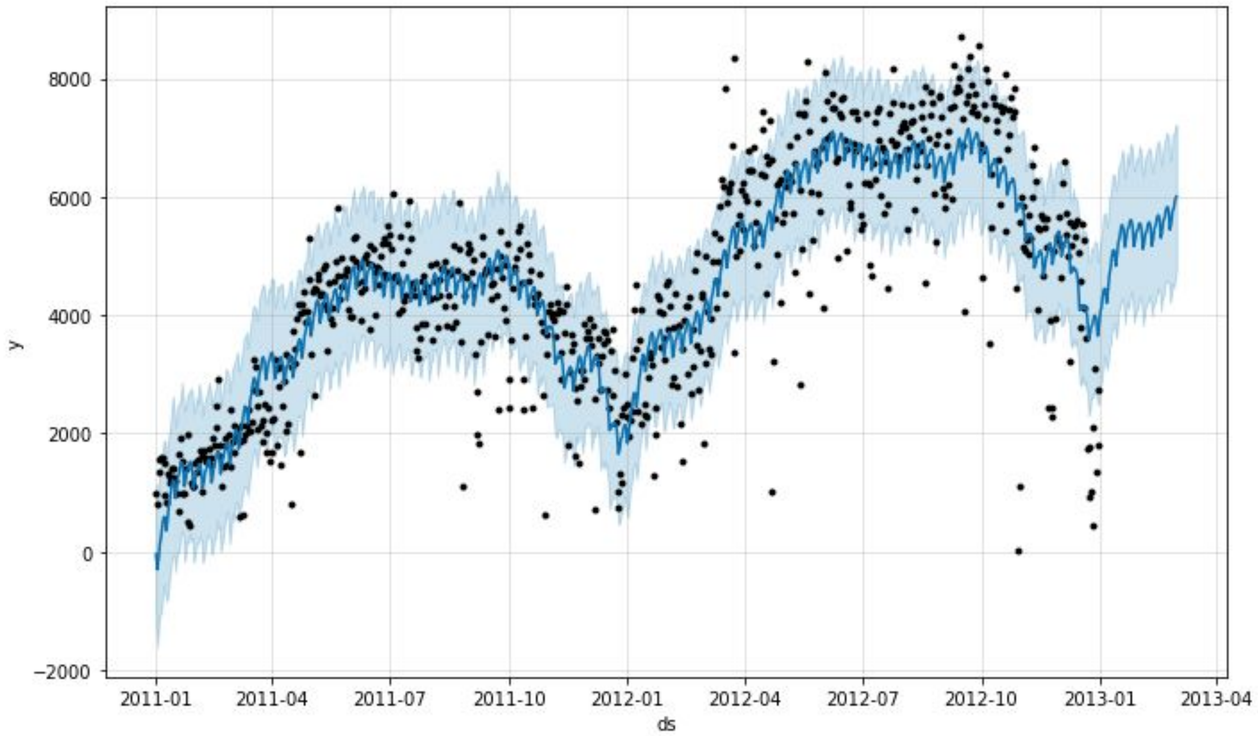
```
mydf = data[['dteday', 'cnt']]
mydf = mydf.rename(index=str, columns={"dteday": "ds", "cnt": "y"})
```

```
m2 = Prophet()
m2.fit(mydf)
```

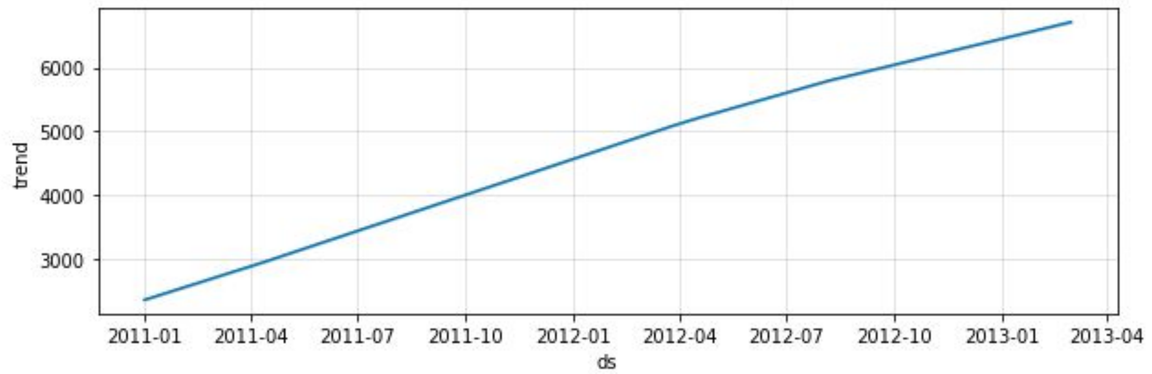
```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

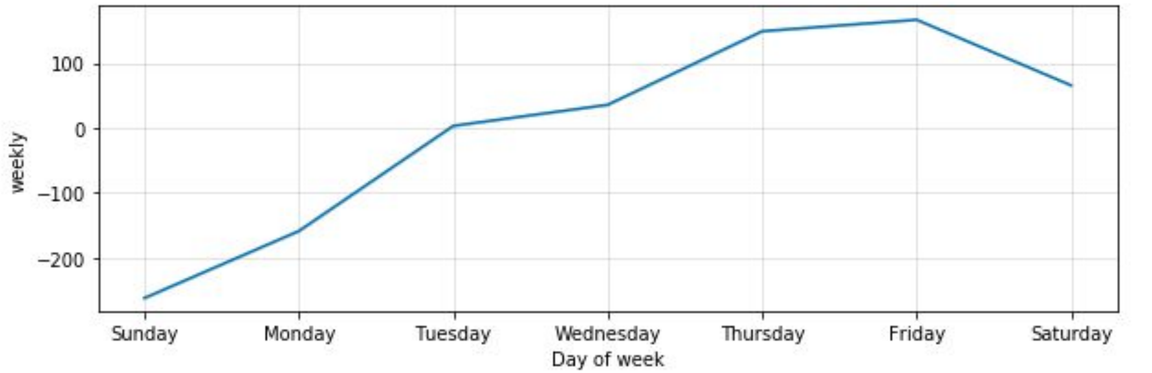
```
<fbprophet.forecaster.Prophet at 0x1148e2320>
```

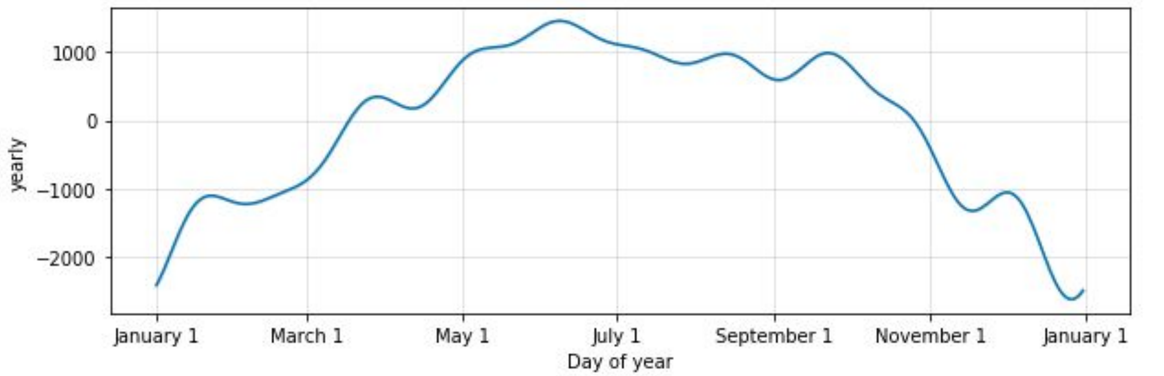
```
future = m2.make_future_dataframe(periods=60)
forecast = m2.predict(future)
fig1 = m2.plot(forecast)
```



```
fig2 = m2.plot_components(forecast)
```





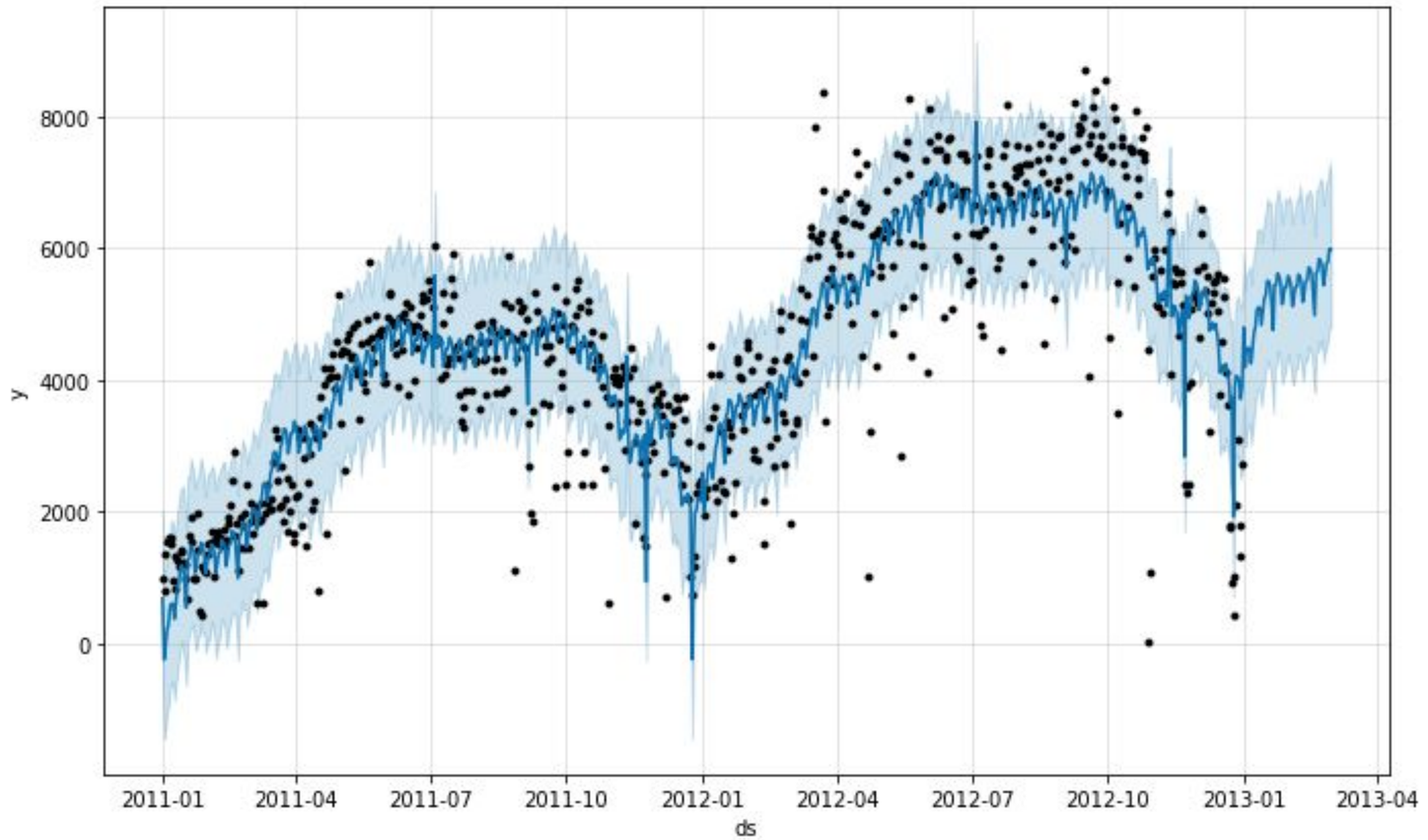


```
m3 = Prophet()
m3.add_country_holidays(country_name='US')
```

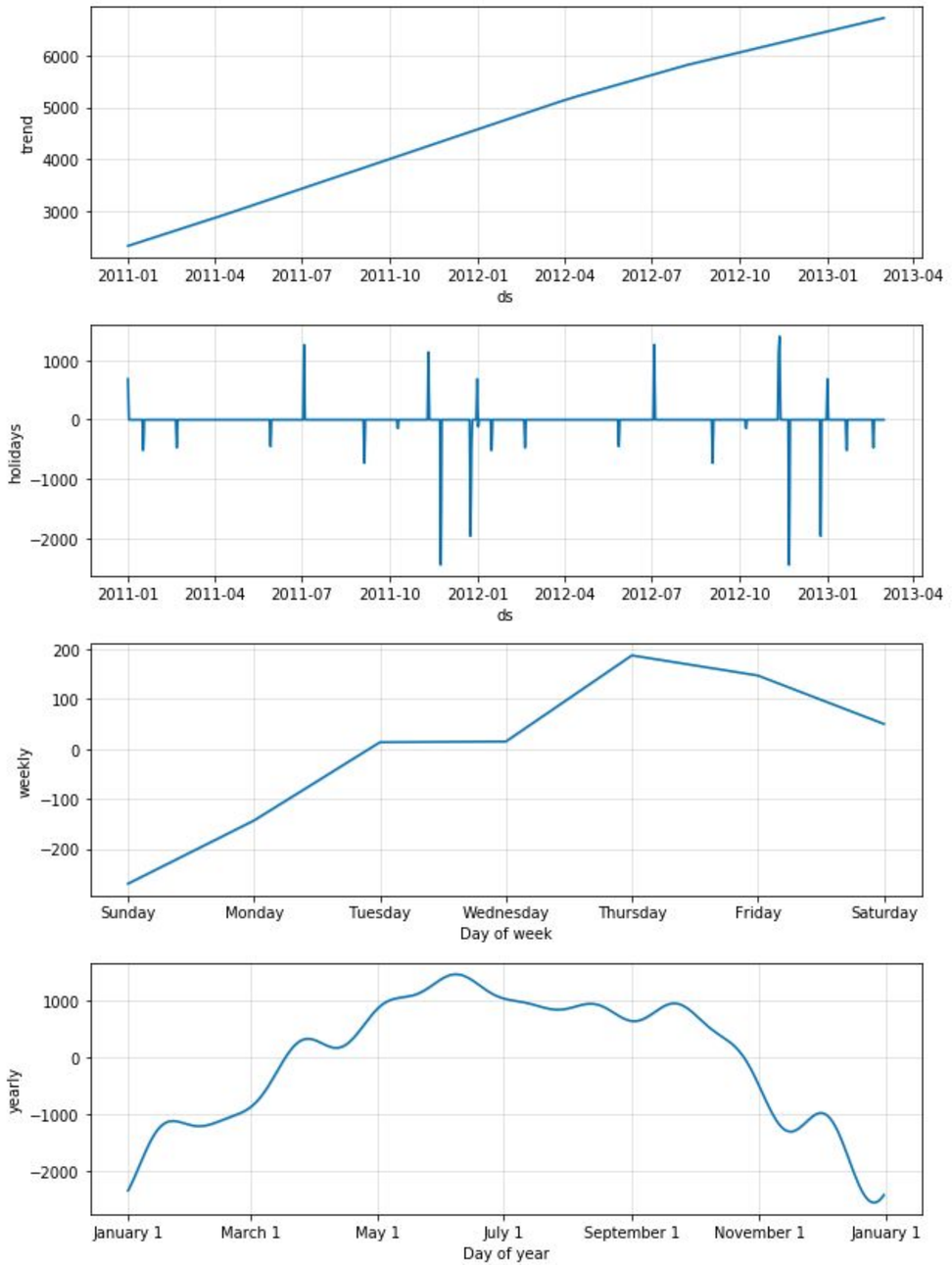
```
<fbprophet.forecaster.Prophet at 0x113580b70>
```

```
m3.fit(mydf)
forecast = m3.predict(future)
fig1 = m3.plot(forecast)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```



```
fig2 = m3.plot_components(forecast)
```

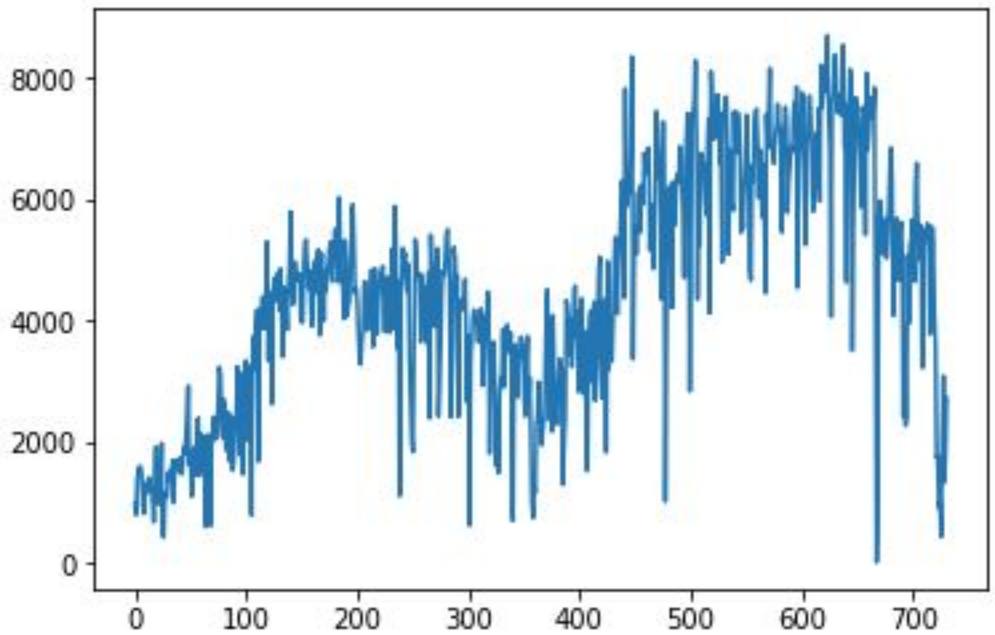


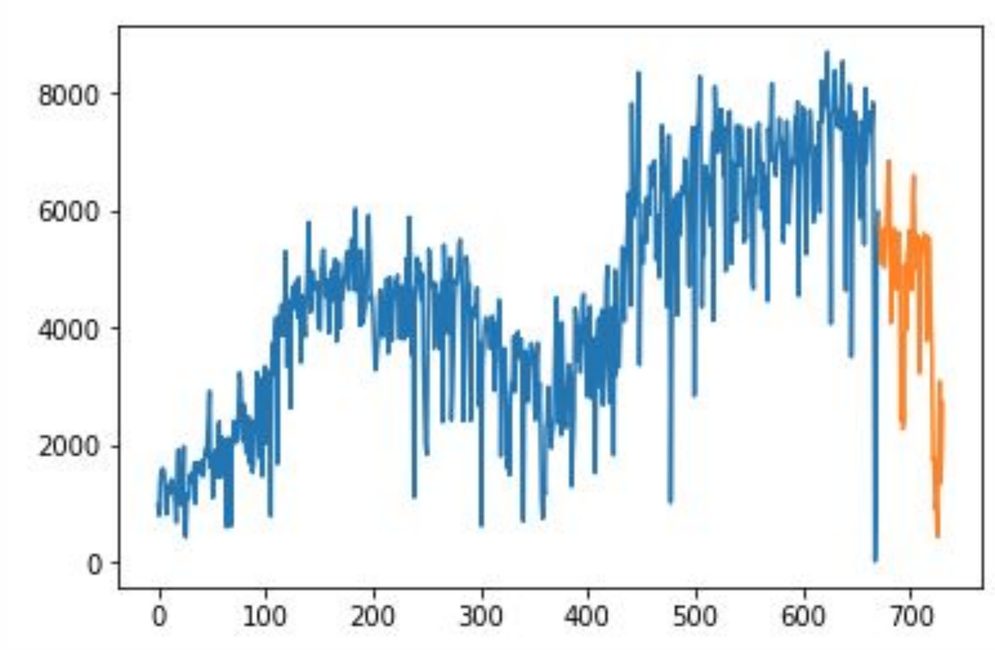
How good is my model?

photo by Matt Popovich https://flic.kr/p/ptiKse

Training vs Testing

We've been just forecasting 60 days off the end of the data. Instead, let's pull the last 60 days off the dataset and use them as a testing dataset. We'll only use the preceding days to train our models.





Root Mean Squared Error

```
from sklearn.metrics import mean_squared_error
import math
```

```
training = data[:-60]
```

```
test = data[-60:]
```

OLS RMSE: 3204

```
#OLS
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(training['x'],training['cnt'])
```

```
fitols = slope*test['x']+intercept
```

```
math.sqrt(mean_squared_error(test['cnt'], fitols))
```

```
3204.640298705231
```

Polynomial RMSE: 4837

```
z = np.polyfit(training['x'],training['cnt'], 3)
p = np.poly1d(z)
math.sqrt(mean_squared_error(test['cnt'], p(test['x'])))
```

```
4837.992211363332
```

ARIMA RMSE: 2318

```
model = ARIMA(data['cnt'], order=(1,1,1))
model_fit = model.fit(disp=0)
arimafit = model_fit.forecast(60)
math.sqrt(mean_squared_error(test['cnt'], arimafit[0]))
```

```
2318.013564362021
```

Prophet RMSE: 1990

```
mytraining = training.rename(index=str, columns={"dteday": "ds", "cnt": "y"})
```

```
m3 = Prophet()
m3.add_country_holidays(country_name='US')
m3.fit(mydf)
future = m3.make_future_dataframe(periods=60)
forecast = m3.predict(future)
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

```
math.sqrt(mean_squared_error(test['cnt'],forecast['yhat'][-60:]))
```

```
1990.6959838455496
```

Thanks for listening!

slides and jupyter notebook posted at: https://github.com/ansate/ml4all2019

photo by Fons Heijnsbroek https://flic.kr/p/FA3keC