We create an instance of the Prophet class and then call its fit and predict methods.

- The input to Prophet is always a dataframe with two columns: ds and y
- The ds (datestamp) column should be of a format expected by Pandas, ideally YYYY-MM-DD for a date or YYYY-MM-DD HH:MM:SS for a timestamp.
- The y column must be numeric, and represents the measurement we wish to forecast.

Looking at a time series of the log daily page views for the Wikipedia page for Peyton Manning, scraped this data using the Wikipediatrend package in R.

Peyton Manning provides a nice example because it illustrates some of Prophet's features, like multiple seasonality, changing growth rates, and the ability to model special days (such as Manning's playoff and superbowl appearances).

In [4]:
```
import pandas as pd
from prophet import Prophet
```

In [8]:
```
df = pd.read_csv('C:/Users/xxxxxxx/example_wp_log_peyton_manning.csv')
df.head()
df.tail()
```

Out[8]:

	ds	y
2900	2016-01-16	7.817223
2901	2016-01-17	9.273878
2902	2016-01-18	10.333775
2903	2016-01-19	9.125871
2904	2016-01-20	8.891374

We fit the model by instantiating a new Prophet object. Any settings to the forecasting procedure are passed into the constructor. Then you call its fit method and pass in the historical dataframe. Fitting should take 1-5 seconds.

In [6]:
```
m = Prophet()
m.fit(df)
```

```
16:07:53 - cmdstanpy - INFO - Chain [1] start processing
16:07:54 - cmdstanpy - INFO - Chain [1] done processing
```

Out[6]:
```
<prophet.forecaster.Prophet at 0x17efdfcae90>
```

Predictions are then made on a dataframe with a column ds containing the dates for which a prediction is to be made. You can get a suitable dataframe that extends into the future a specified number of days using the helper method Prophet.make_future_dataframe. By default it will also include the dates from the history, so we will see the model fit as well.

In [7]:
```
future = m.make_future_dataframe(periods=365)
future.tail()
```

Out[7]:

	ds
3265	2017-01-15

3266	2017-01-16
3267	2017-01-17
3268	2017-01-18
3269	2017-01-19

The predict method will assign each row in future a predicted value which it names yhat. If you pass in historical dates, it will provide an in-sample fit. The forecast object here is a new dataframe that includes a column yhat with the forecast, as well as columns for components and uncertainty intervals.

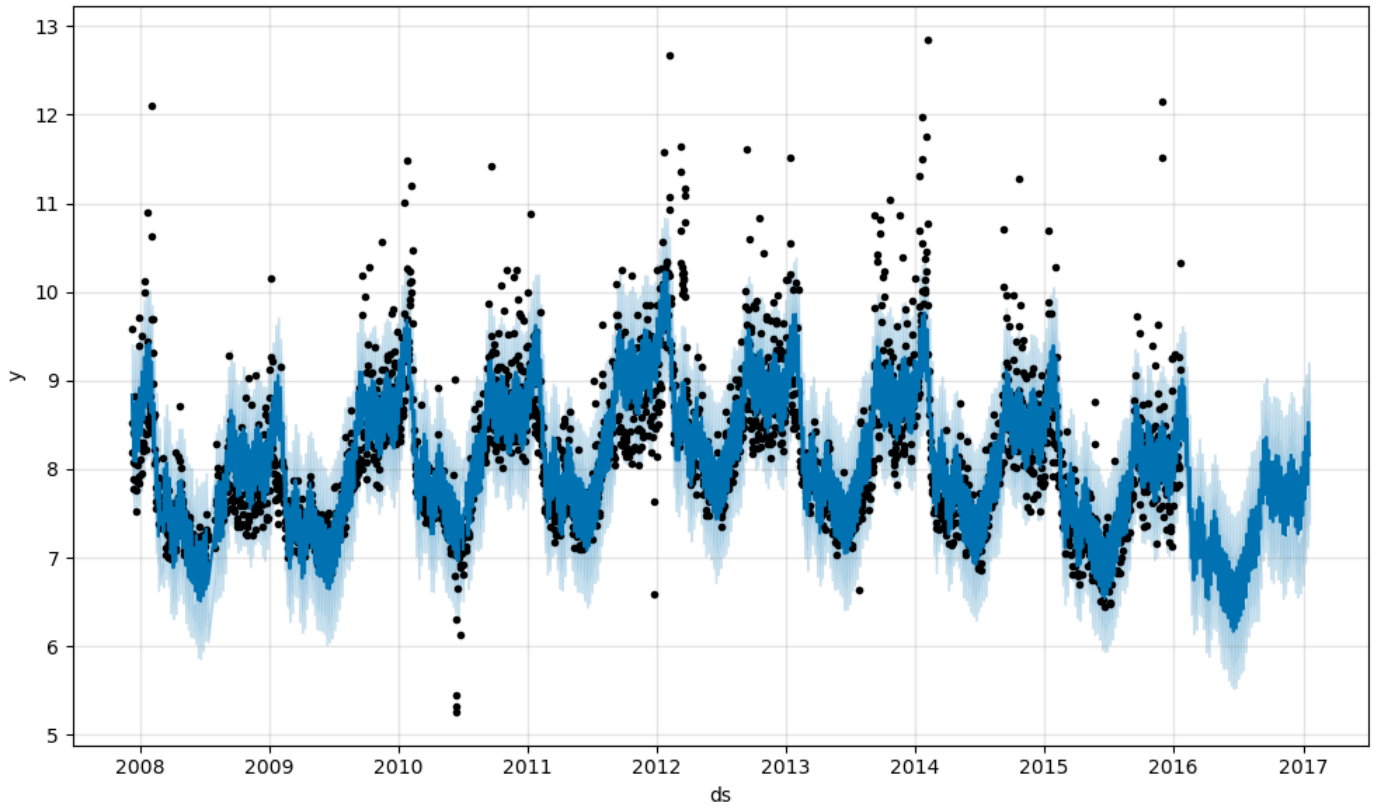
In [9]:
```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[9]:

	ds	yhat	yhat_lower	yhat_upper
3265	2017-01-15	8.206605	7.521255	8.961589
3266	2017-01-16	8.531590	7.792124	9.205459
3267	2017-01-17	8.318985	7.606815	9.046137
3268	2017-01-18	8.151637	7.384339	8.862642
3269	2017-01-19	8.163528	7.396275	8.852262

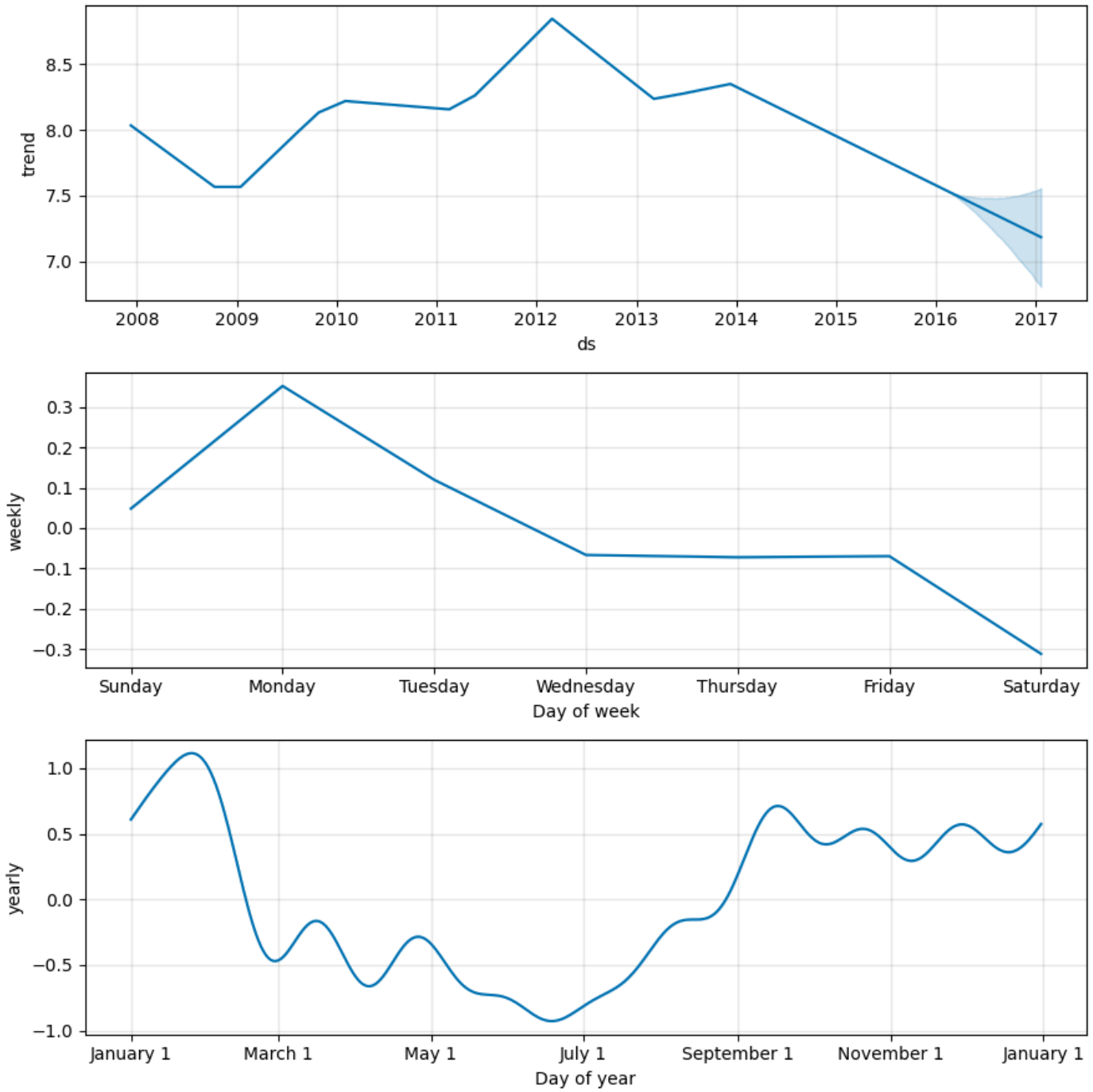
You can plot the forecast by calling the Prophet.plot method and passing in your forecast dataframe.

In [10]:
```
fig1 = m.plot(forecast)
```



If you want to see the forecast components, you can use the Prophet.plot_components method. By default you'll see the trend, yearly seasonality, and weekly seasonality of the time series. If you include holidays, you'll see those here, too.

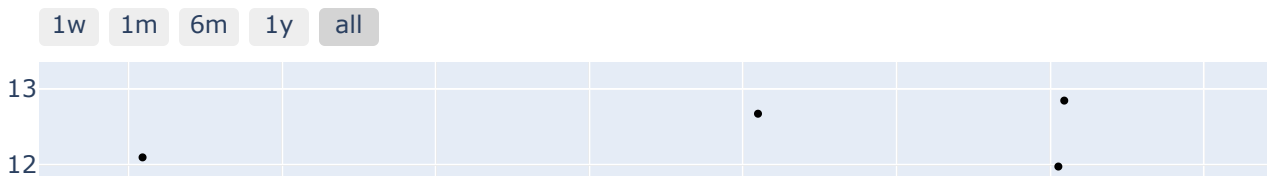
In [11]:
```
fig2 = m.plot_components(forecast)
```

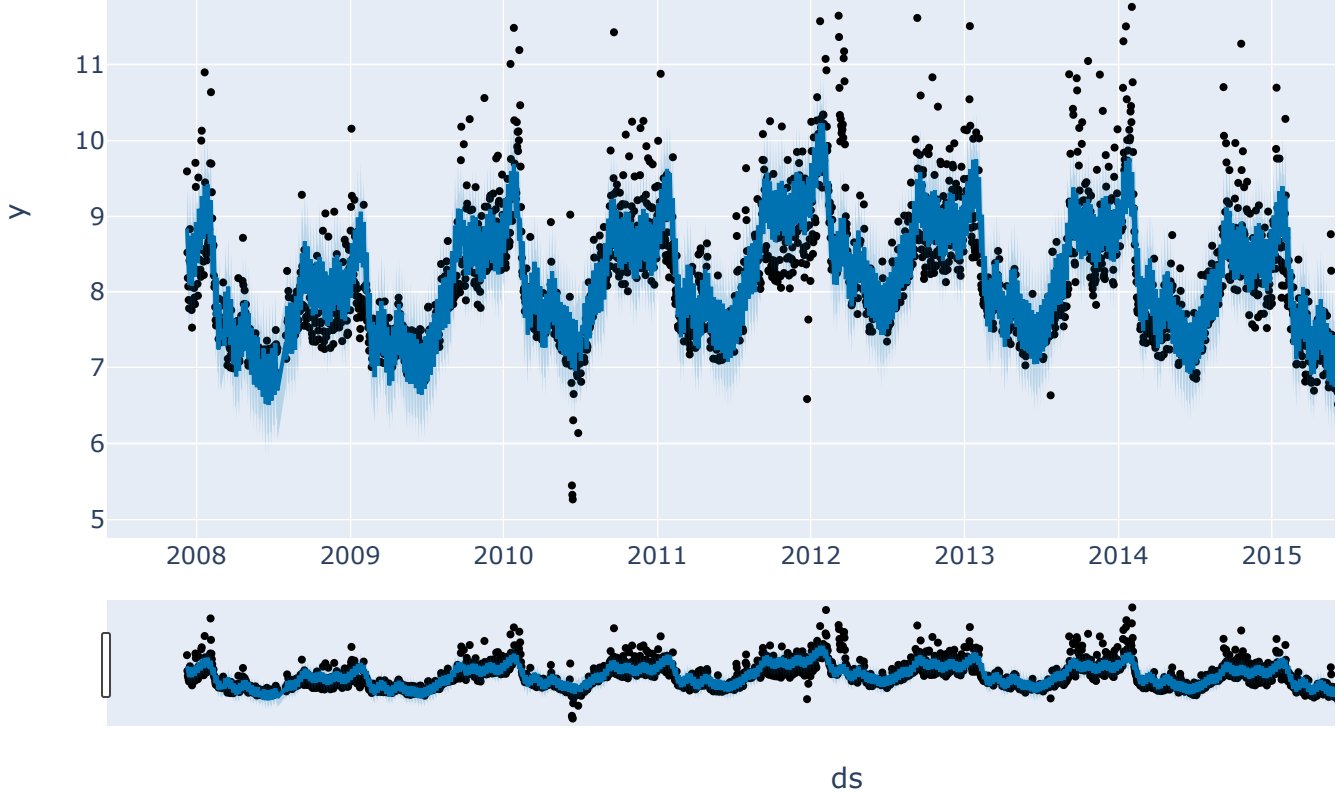


An interactive figure of the forecast and components can be created with plotly. You will need to install plotly 4.0 or above separately, as it will not by default be installed with prophet. You will also need to install the notebook and ipywidgets packages.

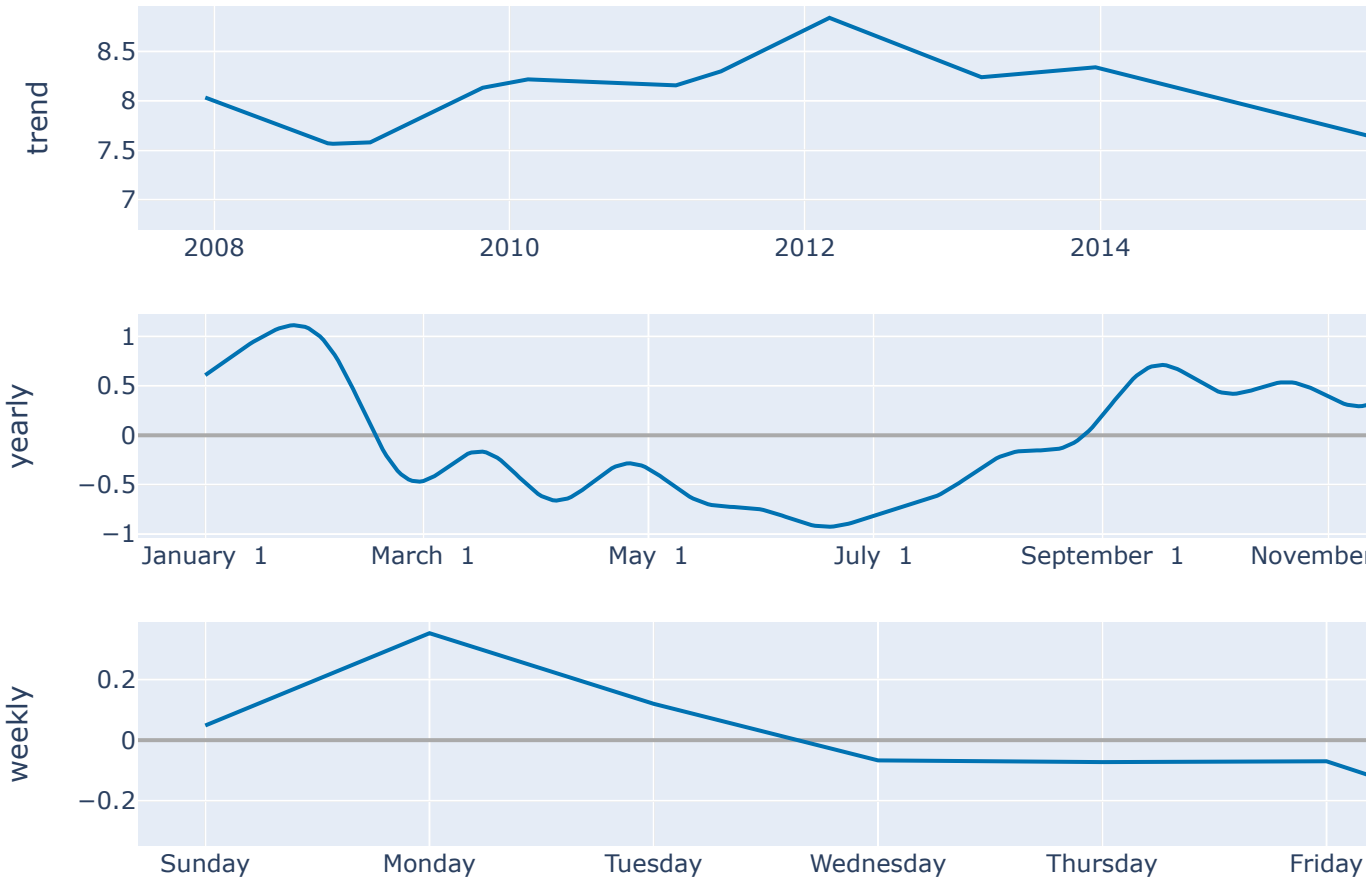
In [12]:
```
from prophet.plot import plot_plotly, plot_components_plotly

plot_plotly(m, forecast)
```





In [13]:
```
plot_components_plotly(m, forecast)
```



In [ ]: