- Docs
- GitHub

Docs

+Documentation

- Installation
  - Using R
  - Using Python
- Quick Start
  - Python API
  - R API
- Saturating Forecasts
  - Forecasting Growth
  - Saturating Minimum
- Trend Changepoints
  - Automatic changepoint detection in Prophet
  - Adjusting trend flexibility
  - Specifying the locations of the changepoints
- Seasonality, Holiday Effects, And Regressors
  - Modeling Holidays and Special Events
  - Built-in Country Holidays
  - Fourier Order for Seasonalities
  - Specifying Custom Seasonalities
  - Seasonalities that depend on other factors
  - Prior scale for holidays and seasonality
  - Additional regressors
- Multiplicative Seasonality
- Uncertainty Intervals
  - Uncertainty in the trend
  - Uncertainty in seasonality
- Outliers
- Non-Daily Data
  - Sub-daily data
  - Data with regular gaps
  - Monthly data
- Diagnostics
- Getting Help and Contributing
  - Generating documentation

Quick Start

Python API

Prophet follows the `sklearn` model API. 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.

As an example, let's look at a time series of the log daily page views for the Wikipedia page for Peyton Manning. We 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). The CSV is available here.

First we'll import the data:

```
# Python
import pandas as pd
from fbprophet import Prophet
```

```
# Python
df = pd.read_csv('../examples/example_wp_log_peyton_manning.csv')
df.head()
```

	ds	y
0	2007-12-10	9.590761
1	2007-12-11	8.519590
2	2007-12-12	8.183677
3	2007-12-13	8.072467
4	2007-12-14	7.893572

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.

```
# Python
m = Prophet()
m.fit(df)
```

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.

```
# Python
future = m.make_future_dataframe(periods=365)
future.tail()
```

	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.

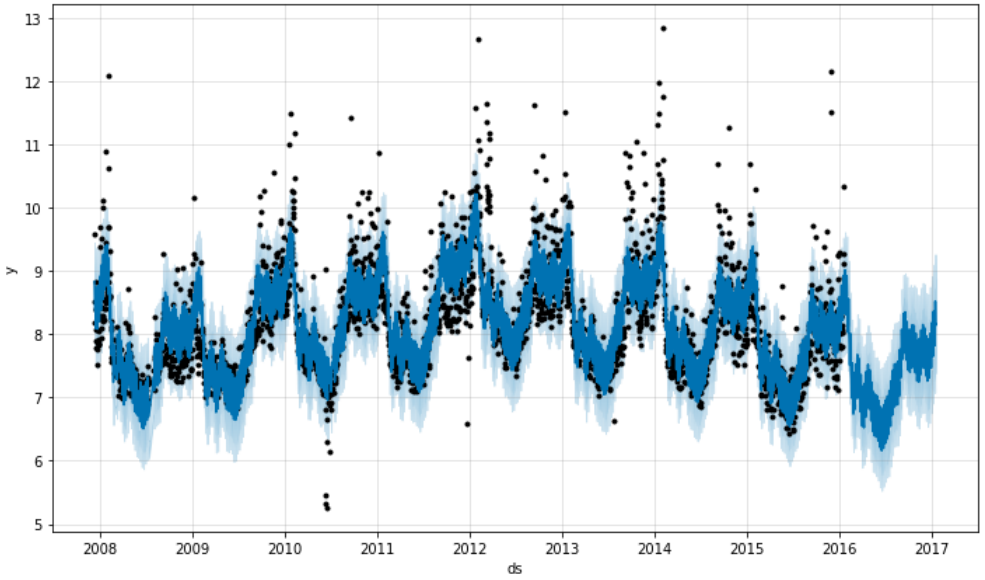
```
# Python
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper

	ds	yhat	yhat_lower	yhat_upper
3265	2017-01-15	8.212942	7.463560	8.937215
3266	2017-01-16	8.537993	7.790259	9.267492
3267	2017-01-17	8.325428	7.525675	9.059391
3268	2017-01-18	8.158059	7.433634	8.883627
3269	2017-01-19	8.170046	7.431801	8.840703

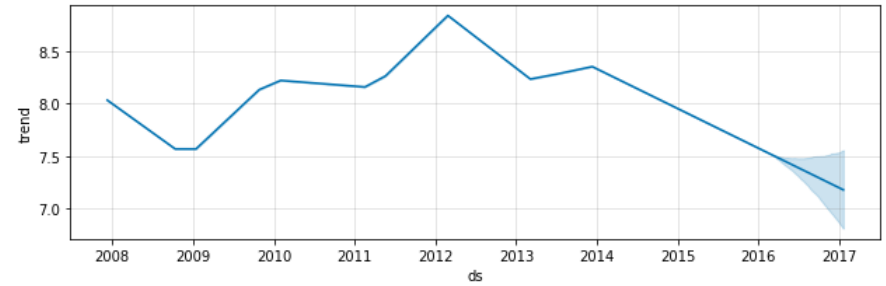
You can plot the forecast by calling the `Prophet.plot` method and passing in your forecast dataframe.

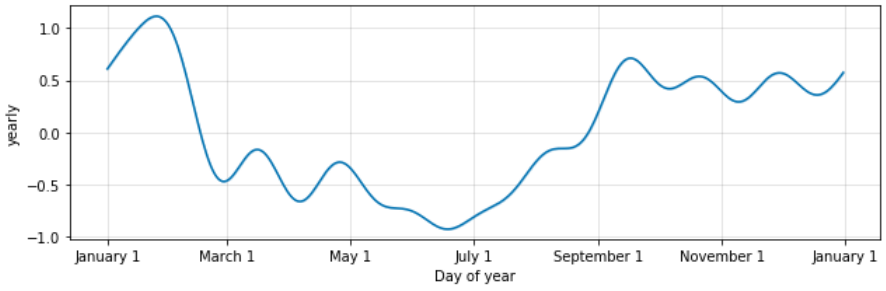
```
1 # Python
2 fig1 = m.plot(forecast)
```

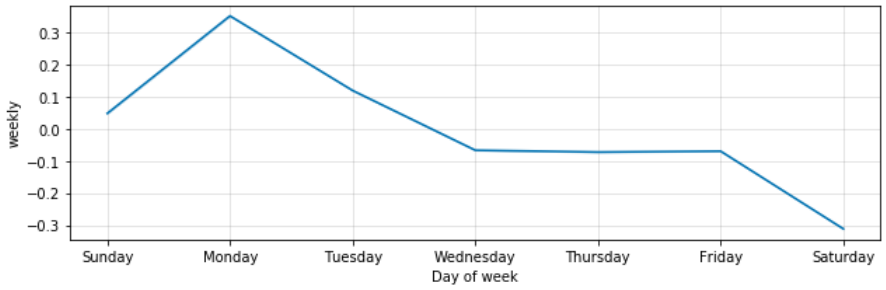


If you want to see the forecast components, you can use the `Prophet.plot_components` method. By default you'll see the trend, yearly seasonality, and weekly seasonality of the time series. If you include holidays, you'll see those here, too.

```
1 # Python
2 fig2 = m.plot_components(forecast)
```







An interactive figure of the forecast can be created with plotly. You will need to install plotly separately, as it will not by default be installed with fbprophet.

```
# Python
from fbprophet.plot import plot_plotly
import plotly.offline as py
py.init_notebook_mode()

fig = plot_plotly(m, forecast)  # This returns a plotly Figure
py.iplot(fig)
```

More details about the options available for each method are available in the docstrings, for example, via `help(Prophet)` or `help(Prophet.fit)`. The R reference manual on CRAN provides a concise list of all of the available functions, each of which has a Python equivalent.

R API

In R, we use the normal model fitting API. We provide a `prophet` function that performs fitting and returns a model object. You can then call `predict` and `plot` on this model object.

```
# R
library(prophet)
```

First we read in the data and create the outcome variable. As in the Python API, this is a dataframe with columns `ds` and `y`, containing the date and numeric value respectively. The ds column should be YYYY-MM-DD for a date, or YYYY-MM-DD HH:MM:SS for a timestamp. As above, we use here the log number of views to Peyton Manning's Wikipedia page, available here.

```
# R
df <- read.csv('../examples/example_wp_log_peyton_manning.csv')
```

We call the `prophet` function to fit the model. The first argument is the historical dataframe. Additional arguments control how Prophet fits the data and are described in later pages of this documentation.

```
# R
m <- prophet(df)
```

Predictions are made on a dataframe with a column `ds` containing the dates for which predictions are to be made. The `make_future_dataframe` function takes the model object and a number of periods to forecast and produces a suitable dataframe. By default it will also include the historical dates so we can evaluate in-sample fit.

```
# R
future <- make_future_dataframe(m, periods = 365)
tail(future)
```

```
             ds
3265 2017-01-14
3266 2017-01-15
3267 2017-01-16
3268 2017-01-17
3269 2017-01-18
3270 2017-01-19
```

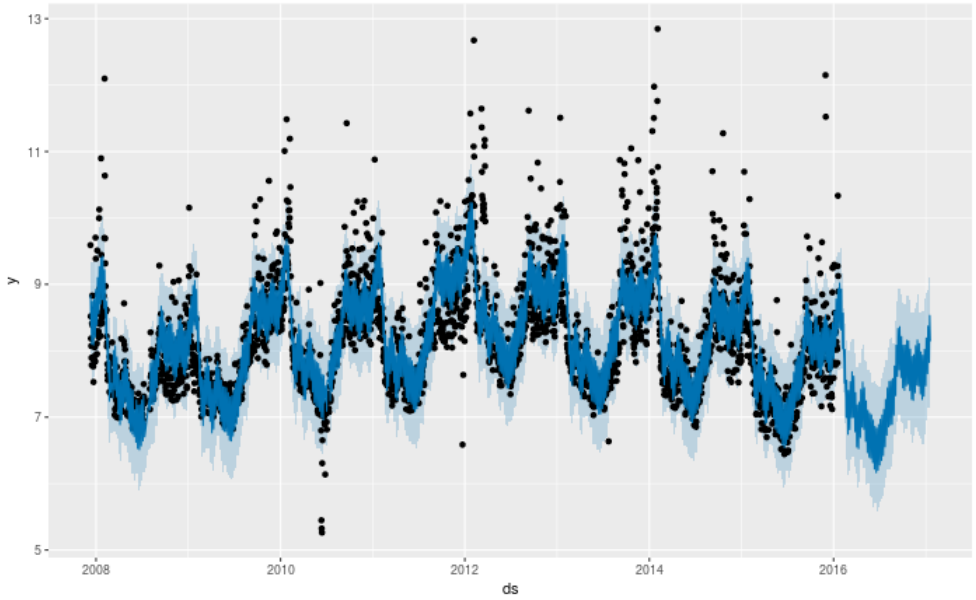
As with most modeling procedures in R, we use the generic `predict` function to get our forecast. The `forecast` object is a dataframe with a column `yhat` containing the forecast. It has additional columns for uncertainty intervals and seasonal components.

```
# R
forecast <- predict(m, future)
tail(forecast[c('ds', 'yhat', 'yhat_lower', 'yhat_upper')])
```

```
             ds     yhat yhat_lower yhat_upper
3265 2017-01-14 7.824163   7.127881   8.609668
3266 2017-01-15 8.205942   7.452071   8.904387
3267 2017-01-16 8.530942   7.742400   9.300974
3268 2017-01-17 8.318327   7.606534   9.071184
3269 2017-01-18 8.150948   7.440224   8.902922
3270 2017-01-19 8.162839   7.385953   8.890669
```

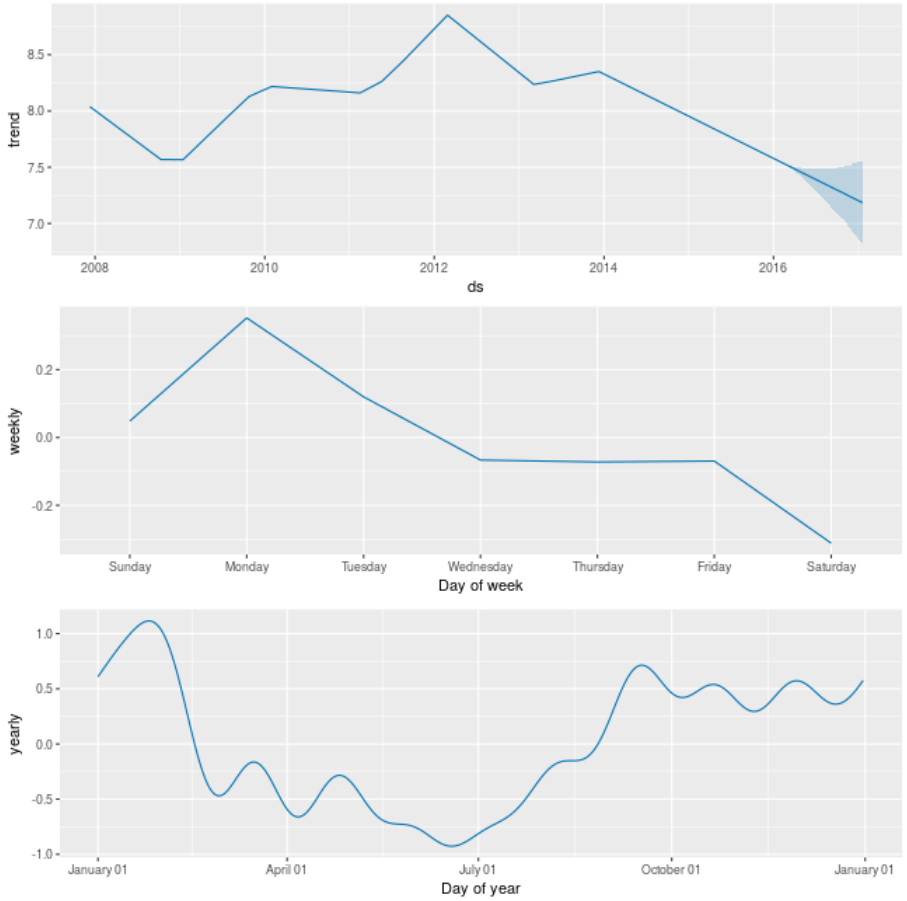
You can use the generic `plot` function to plot the forecast, by passing in the model and the forecast dataframe.

```
# R
plot(m, forecast)
```



You can use the `prophet_plot_components` function to see the forecast broken down into trend, weekly seasonality, and yearly seasonality.

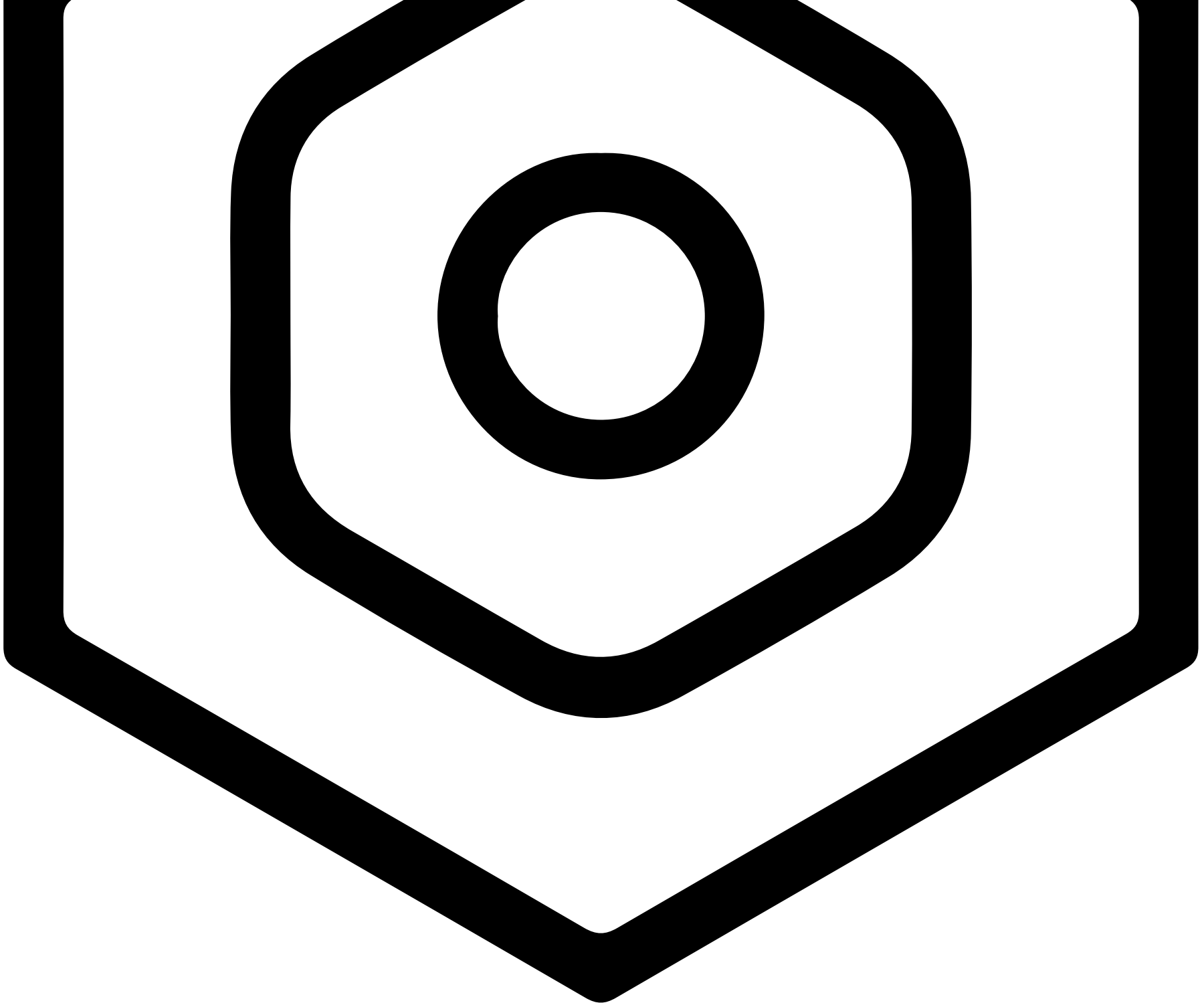
```
1 # R
2 prophet_plot_components(m, forecast)
```



An interactive plot of the forecast using Dygraphs can be made with the command `dyplot.prophet(m, forecast)`.

More details about the options available for each method are available in the docstrings, for example, via `?prophet` or `?fit.prophet`. This documentation is also available in the reference manual on CRAN.

Edit on GitHub



Facebook Open Source