PROPHET Docs GitHub

```
Documentation
Installation
   Using R
   Using Python
Quick Start
   Python API
   RAPI
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
   and seasonality
   Coefficients of
Multiplicative Seasonality
Uncertainty Intervals
   Uncertainty in the
   trend
   Uncertainty in
   seasonality
Outliers
Non-Daily Data
```

Prior scale for holidays Additional regressors additional regressors

Sub-daily data Data with regular gaps Monthly data Holidays with aggregated data **Diagnostics Cross validation** Parallelizing cross validation Hyperparameter tuning

**Handling Shocks Treating COVID-19** lockdowns as a oneoff holidays Sense checking the trend Changes in seasonality between pre- and post-COVID **Additional Topics** Saving models

trends **Updating fitted** models External references **Getting Help and Contributing** 

Flat trend and custom

## **Python API**

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

Prophet follows the sklearn model API. We create an instance of the Prophet

First we'll import the data: # Python

3

import pandas as pd

from prophet import Prophet

```
1
    # Python
2
   df = pd.read_csv('https://raw.githubusercontent.com/facebook/prophet/main/examp
3
   df.head()
           DS
                       Υ
```

```
2007-12-10
                9.590761
   2007-12-11
                8.519590
1
```

```
2
     2007-12-12
                 8.183677
     2007-12-13
                 8.072467
 3
     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
```

Prophet.make\_future\_dataframe. By default it will also include the dates from the

history, so we will see the model fit as well.

```
future = m.make_future_dataframe(periods=365)
tuture.tail()
```

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.

The predict method will assign each row in future a predicted value which it

```
2017-01-17
                    8.323968
                                               9.035461
                               7.541829
        2017-01-18
                    8.156621
                                               8.830642
 3268
                               7.404457
 3269
        2017-01-19
                    8.168561
                               7.438865
                                               8.908668
You can plot the forecast by calling the Prophet.plot method and passing in your
```

forecast dataframe.

13

12

11

weekly

Sunday

1.0

0.5

0.0

-0.5

-1.0

this model object.

library(prophet)

R[write to console]: Loading required package: Rcpp

R[write to console]: Loading required package: rlang

to Peyton Manning's Wikipedia page, available here.

described in later pages of this documentation.

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

df <- read.csv('https://raw.githubusercontent.com/facebook/prophet/main/example</pre>

yearly

Monday

Tuesday

fig1 = m.plot(forecast)

2017

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. fig2 = m.plot\_components(forecast) 8.5 trend 80 75 7.0 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 ds 0.3

March 1 May 1 July 1 September 1 January 1 November 1 January 1 Day of year 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. from prophet.plot import plot\_plotly, plot\_components\_plotly plot\_plotly(m, forecast) plot\_components\_plotly(m, forecast) 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. **RAPI** In R, we use the normal model fitting API. We provide a prophet function that

## dataframe. By default it will also include the historical dates so we can evaluate

in-sample fit.

tail(future)

3267 2017-01-16 3268 2017-01-17 3269 2017-01-18 3270 2017-01-19

seasonal components.

forecast <- predict(m, future)</pre>

ds

plot(m, forecast)

13-

11-

85

75-

trend

3265 2017-01-14 7.818359 7.071228

3266 2017-01-15 8.200125 7.475725 8.869495 3267 2017-01-16 8.525104 7.747071 9.226915

3265 2017-01-14 3266 2017-01-15

As with most modeling procedures in R, we use the generic predict function to

containing the forecast. It has additional columns for uncertainty intervals and

get our forecast. The forecast object is a dataframe with a column yhat

tail(forecast[c('ds', 'yhat', 'yhat\_lower', 'yhat\_upper')])

yhat yhat\_lower yhat\_upper

8.550957

future <- make\_future\_dataframe(m, periods = 365)</pre>

```
3269 2017-01-18 8.145098 7.390770
                                         8.863692
    3270 2017-01-19 8.156964 7.381716
                                         8.866507
You can use the generic plot function to plot the forecast, by passing in the
model and the forecast dataframe.
```

prophet\_plot\_components(m, forecast)

J.D -2010 2014 2016 2012 02weekly 0.0 02-Sanday Tuesday Satisting Monday Wednesday Thursday Friday Day of week 1.0 -

Jumpary D1 Day of year An interactive plot of the forecast using Dygraphs can be made with the command dyplot.prophet(m, forecast).

docstrings, for example, via ?prophet or ?fit.prophet. This documentation is also

available in the reference manual on CRAN.

GitHub

**Twitter** 

**Open Source Projects** 

**Terms** 

**Privacy** 

More details about the options available for each method are available in the

Edit on GitHub

**GitHub** 

Contribute to this project on



(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.

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

DS 2017-01-15 3265 2017-01-16 3266 2017-01-17 3267 2017-01-18 3268 2017-01-19 3269

forecast = m.predict(future) forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail() YHAT\_LOWER YHAT\_UPPER DS **YHAT** 3265 2017-01-15 8.211542 7.444742 8.903545 3266 2017-01-16 8.536553 7.847804 9.211145 3267

					4	
					1	
7		/ N	1		1	
1	Ty II	1	14	•	* 1	
1	8 2009	2010	2011 20	012 2013	2014	2015 2016

0.2 0.1 0.0 -0.1-0.2-0.3

Wednesday

Day of week

Thursday.

Friday

Saturday

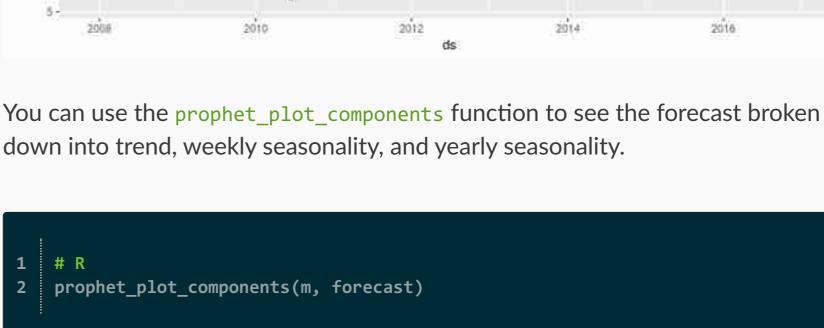


<- prophet(df)</pre> Predictions are made on a dataframe with a column ds containing the dates for which predictions are to be made. The make\_future\_dataframe function takes the model object and a number of periods to forecast and produces a suitable

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

```
3268 2017-01-17 8.312482 7.551904
                                  9.046774
```



05. yearly 05 April 01 01 ولنظ October 01 Junuary C1

**Facebook Open Source**