CHAPTER 8

Prophet

Prophet is an open source framework from Facebook used for framing and forecasting time series. It focuses on an additive model where nonlinear trends fit with daily, weekly, and yearly seasonality and additional holiday effects. Prophet is powerful at handling missing data and shifts within the trends and generally handles outliers well. It also allows you to accumulate exogenous variables to the model.

The Prophet Model

Prophet uses a decomposable time-series model.

$$\mathbf{y}(\boldsymbol{t}) = \mathbf{g}(\boldsymbol{t}) + \mathbf{s}(\boldsymbol{t}) + \mathbf{h}(\boldsymbol{t}) + \boldsymbol{\epsilon}_{\mathbf{t}}$$

where

g(t) = Trend (linear/logistic)

s(t) = Periodic change/seasonality

$$s(t) = \sum_{n=1}^{N}\left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right)\right)$$

Here, P is the consistent period we anticipate the time series to have (e.g., P = 365.25 for yearly data or P = 7 for weekly data, when we are scaling our time variable in days).

Fitting seasonality requires calculating the 2N parameters for β = [a1, b1,…., aN, bN]^T by developing a matrix of seasonality vectors for each and every value of t in our historical data and future data.

h(t) = Effect of holiday

For each holiday *i*, let D *i* be the set of past and future dates for that holiday, as shown here:

$$Z(t) = [1(t \in D_1),\ldots,1(t \in D_L)]$$

and taking the following:

$$h(t) = Z(t)\kappa$$

As with seasonality, we use a prior $\kappa \sim \text{Normal}(0,\nu^2)$.

$\boldsymbol{\epsilon t}$ = Changes that are not adopted by the model

Prophet's core procedure is implemented using Stan (a probabilistic programming language). Stan performs map optimization to find parameters and facilitates estimating parameter uncertainty using the Hamiltonian Monte Carlo algorithm.

In next section, you will learn how to use Prophet and solve time-series problems.

Implementing Prophet

In the previous section, you learned about the high-level math behind Prophet; now let's implement it on a time-series dataset.

Note The input to the Prophet model should always be a DataFrame with the columns ds and y, with ds being a date field (YYYY-MM-DD or YYYY-MM-DD HH:MM:SS) with a corresponding y variable.

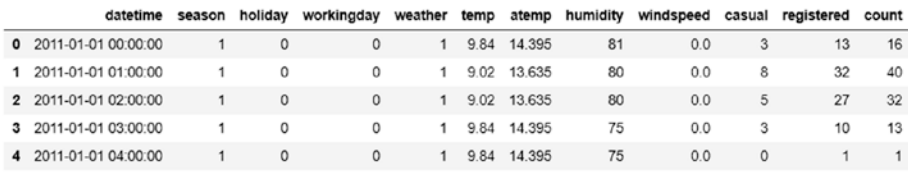
Import the required libraries and load the CSV data, which contains bike-sharing data. The data is from January 2011 to December 2012.

Each day comprises 24 observations as data is at the hourly level. Using this dataset, let's try to forecast the count (the target variable) for the next 48 hours, which is two days.

```
import warnings
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
from fbprophet import Prophet
```

Let's take a sneak peek at the data.

```
df = pd.read_csv('Data\Bike_Sharing_Demand.csv',parse_dates = True)
df.head()
```



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Define the time-series evaluation function.

```
def timeseries_evaluation_metrics_func(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    #print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_
    true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error(y_true,
    y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',
    end='\n\n')
```

Change the data into a format that Prophet accepts.

```
df = df.rename(columns={'datetime': 'ds', 'count': 'y'})
```

Train/test the split, and let's hold back two days of data (i.e., 48 records), which we can use to validate the results after training on past data.

```
validate = df[['ds','y']].tail(48)
df.drop(df[['ds','y']].tail(48).index,inplace=True)
train = df[['ds','y']]
```

Let's initialize the `Prophet` class and then fit it with the default parameter, which will internally find the best parameters for most parameters.

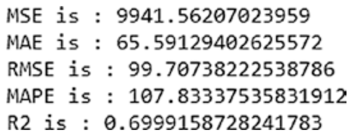
```
m = Prophet(yearly_seasonality=False)
m.fit(train)
```

Prophet requires a future DataFrame to do any forecasting, so let's create one with a frequency of `H` for the next 48 hours.

Other frequency options available are `day`, `week`, `month`, `quarter`, `year`, `1` (1 second), `60` (1 minute), `3600` (1 hour), and `H` (hourly).

```
p = 48
future =m.make_future_dataframe(periods=p,freq='H',
include_history=False)
forecast = m.predict(future)

timeseries_evaluation_metrics_func(validate.y,forecast.yhat)
```



```
MSE is : 9941.56207023959
MAE is : 65.59129402625572
RMSE is : 99.70738222538786
MAPE is : 107.83337535831912
R2 is : 0.6999158728241783
```

Figure 8-1 shows the forecast plot.

```
fig1 = m.plot(forecast)
```

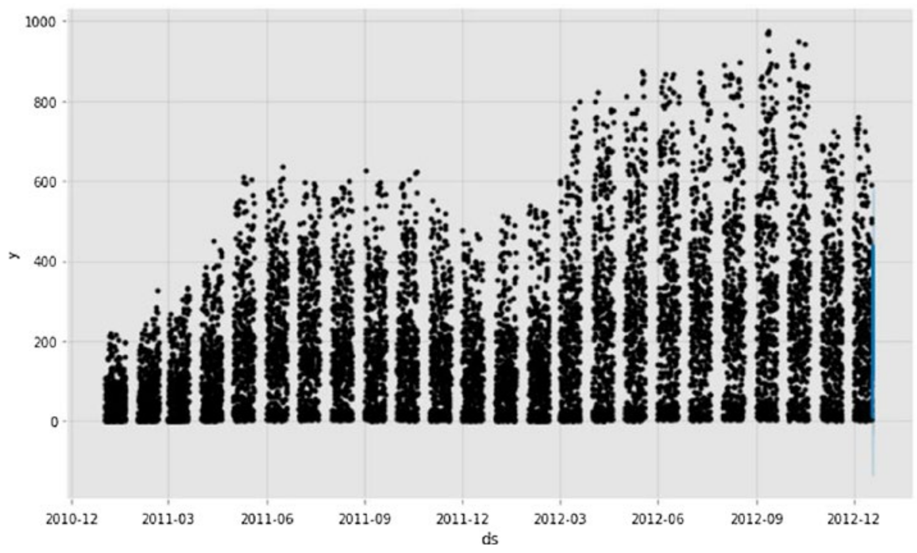


Figure 8-1. *Representation of forecast*

Let's plot the components of the forecast (Figure 8-2).

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

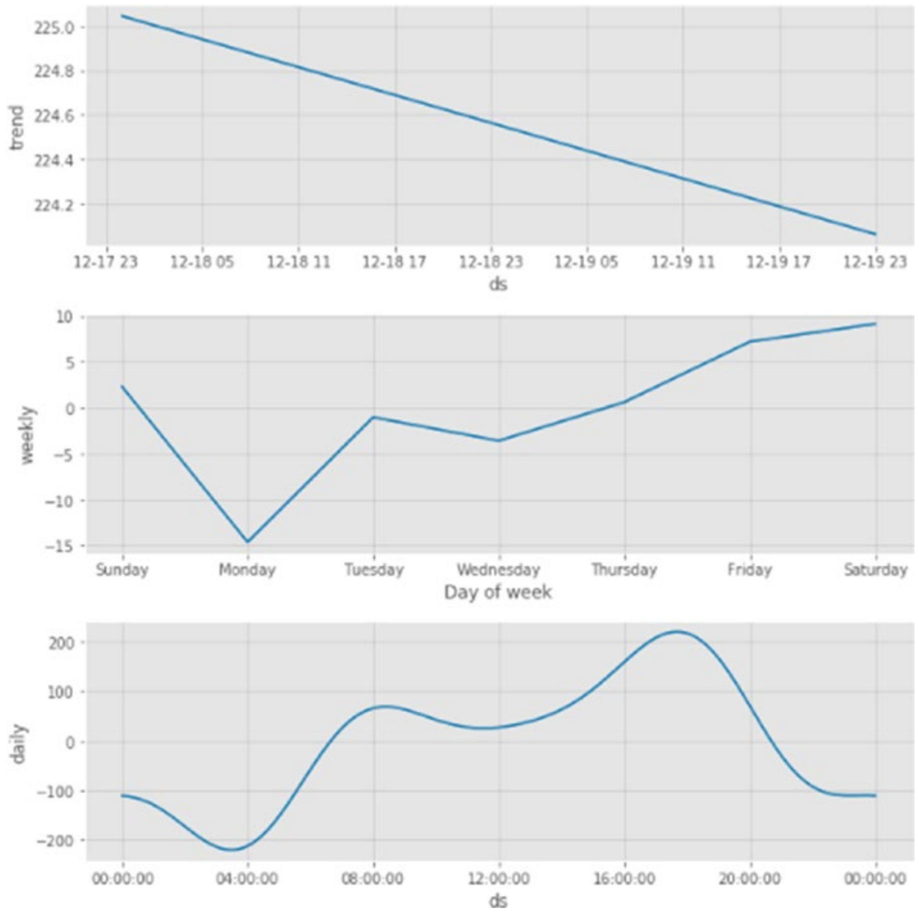


Figure 8-2. *Representation of forecasted components*

The `add_changepoints_to_plot` function adds red lines; the vertical dashed lines are changepoints where Prophet has identified that the trend has changed (Figure 8-3).

```
from fbprophet.plot import add_changepoints_to_plot
fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```

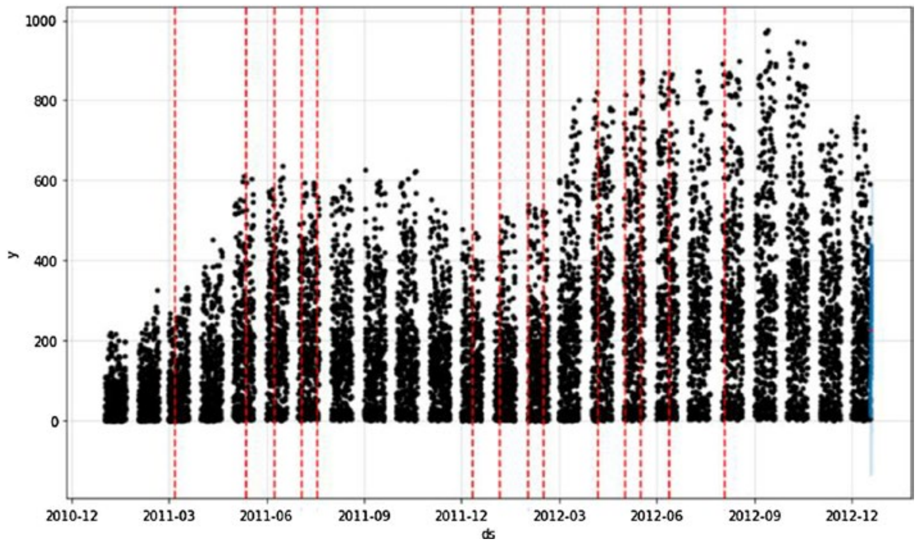


Figure 8-3. *Representation of trend changepoints*

Adding Log Transformation

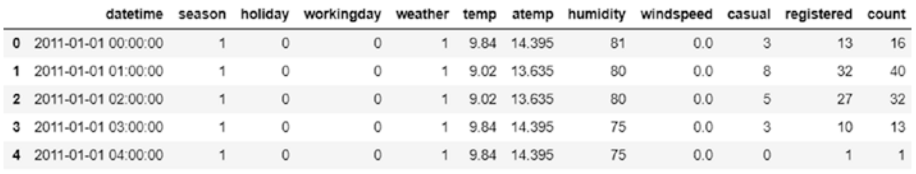
In the previous section, you learned how to model time-series problems using Prophet. In this section, let's apply log transformations on the data and check whether the accuracy increases.

Log-transformed data follows a normal or near-normal distribution. Import the required libraries and load the CSV data.

```
import warnings
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
from fbprophet import Prophet
```

Let's take a sneak peek at the data.

```
df = pd.read_csv('Data\Bike_Sharing_Demand.csv',
parse_dates = True)
df.head()
```



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Define the time-series evaluation function.

```
def timeseries_evaluation_metrics_func(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    #print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_
    true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error
    (y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',
    end='\n\n')
```

Change the data into a format that Prophet accepts.

```
df = df.rename(columns={'datetime': 'ds', 'count': 'y'})
```

Train/test the split. Let's hold back two days of data (i.e., 48 records), which we can use to validate the results after training on past data.

```
validate = df[['ds','y']].tail(48)
df.drop(df[['ds','y']].tail(48).index,inplace=True)
train = df[['ds','y']]
```

We are applying log transformation to make the data normally distributed.

```
train['y']= np.log(train['y'])
```

Let's initialize the `Prophet` class and then fit it.

```
m = Prophet(yearly_seasonality=False)
m.fit(train)
```

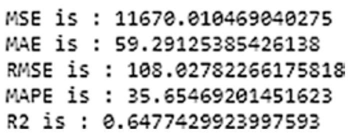
Prophet requires a future DataFrame to do any forecasting, so let's create one with a frequency of `H` for the next 48 hours.

```
p = 48
future = make_future_dataframe(periods=p,freq='H',include_
history=False)
forecast = m.predict(future)
```

Like we applied a log transformation on input data, we need to apply an inverse log transformation on the results to bring them back to the original scale.

```
forecast['yhat'] = np.exp(forecast['yhat'])

timeseries_evaluation_metrics_func(validate.y,forecast['yhat'])
```



```
MSE is : 11670.010469040275
MAE is : 59.29125385426138
RMSE is : 108.02782266175818
MAPE is : 35.65469201451623
R2 is : 0.6477429923997593
```

Figure 8-4 shows the forecast plot.

```
fig1 = m.plot(forecast)
```

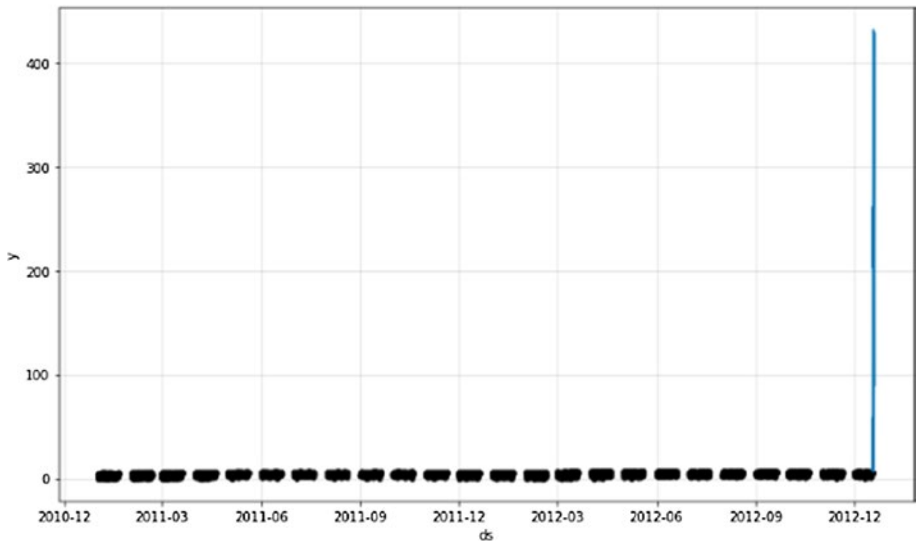


Figure 8-4. *Representation of forecast*

Let's plot the components of the forecast (Figure 8-5).

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

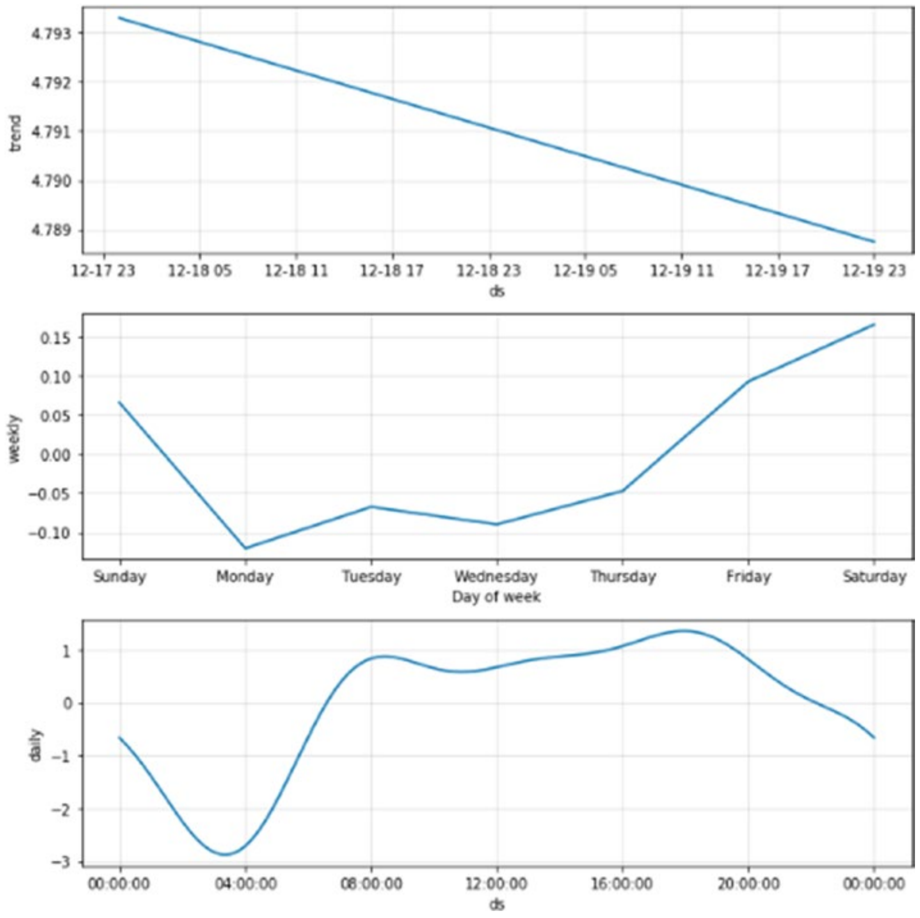


Figure 8-5. *Representation of forecasted components*

Create the changepoints plot (Figure 8-6).

```
from fbprophet.plot import add_changepoints_to_plot
fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```

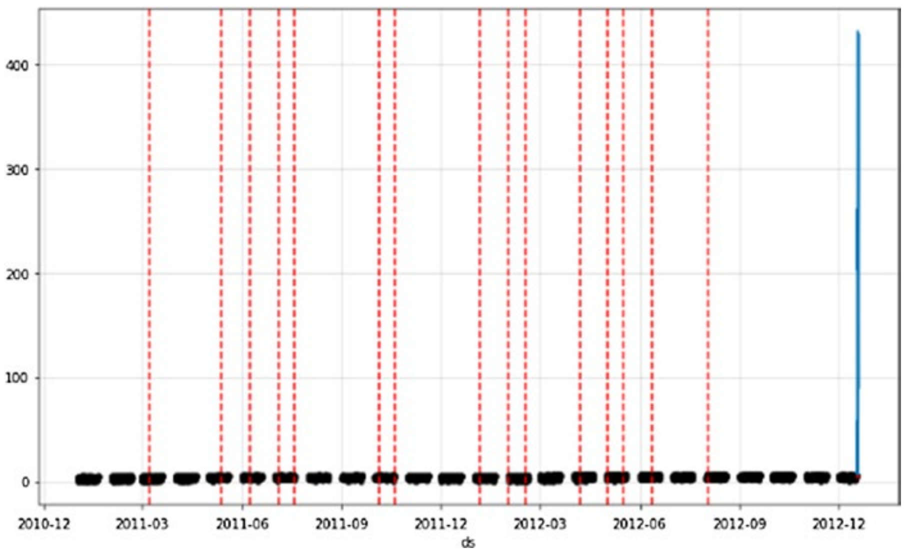


Figure 8-6. *Representation of trend changepoints*

By applying a log transformation, we didn't see an increase in the model accuracy; rather, it was reduced.

Adding Built-in Country Holidays

In this section, let's add US country holidays, which is built in to Prophet, and check whether the accuracy increases.

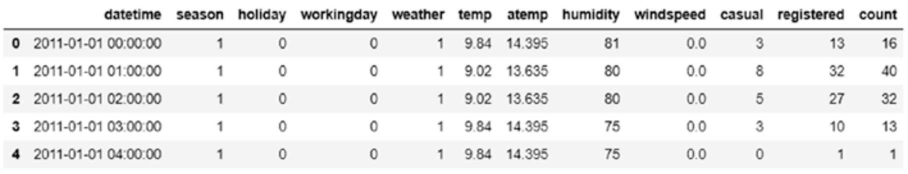
Prophet includes holidays for these countries: Brazil (BR), Indonesia (ID), India (IN), Malaysia (MY), Vietnam (VN), Thailand (TH), Philippines (PH), Turkey (TU), Pakistan (PK), Bangladesh (BD), Egypt (EG), China (CN), and Russia (RU).

Import the required libraries and load the CSV data.

```
import warnings
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
from fbprophet import Prophet
```

Let's take a sneak peek at the data.

```
df = pd.read_csv(r'Data\Bike_Sharing_Demand.csv',
parse_dates = True)
df.head()
```



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Define the time-series evaluation function.

```
def timeseries_evaluation_metrics_func(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    #print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error
    (y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error
    (y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',
    end='\n\n')
```

Change the data into a format that Prophet accepts.

```
df = df.rename(columns={'datetime': 'ds', 'count': 'y'})
```

Train/test the split, and let's hold back two days of data (i.e., 48 records), which we can use to validate the data after training on past data.

```
validate = df[['ds','y']].tail(48)
df.drop(df[['ds','y']].tail(48).index,inplace=True)
train = df[['ds','y']]
```

Let's initialize the `Prophet` class, add country holidays using `add_country_holidays`, and then fit the data with the default parameters.

```
m = Prophet(yearly_seasonality=False)
m.add_country_holidays(country_name='US')
m.fit(train)
```

Prophet requires a future DataFrame to do any forecasting, so let's create one with a frequency of `H` for the next 48 hours.

```
p = 48
future = m.make_future_dataframe(periods=p,freq='H',include_
history=False)
forecast = m.predict(future)

timeseries_evaluation_metrics_func(validate.y,forecast.yhat)
```

```
MSE is : 10299.423817670899
MAE is : 61.793582875969015
RMSE is : 101.48607696463047
MAPE is : 93.2326573364842
R2 is : 0.6891138852322072
```

Create the changepoints plot (Figure 8-7).

```
from fbprophet.plot import add_changepoints_to_plot
fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```

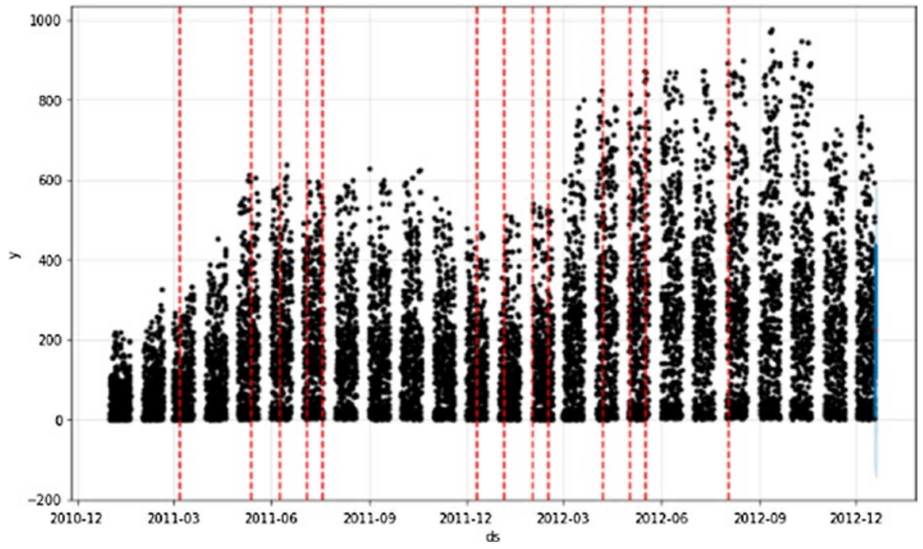


Figure 8-7. *Representation of trend changepoints*

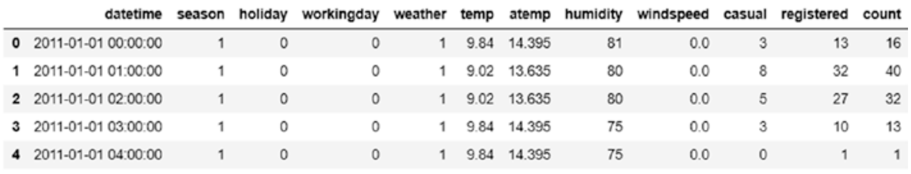
Adding Exogenous variables using add_regressors(function)

In this section, let's add exogenous variables, which might influence target variables, and check whether the accuracy increases. We can call this problem a multivariant time-series problem too.

```
import warnings
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import metrics
from fbprophet import Prophet
```

Import the required libraries and load the CSV data.

```
df = pd.read_csv('Data\Bike_Sharing_Demand.csv',
parse_dates = True)
df.head()
```



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
df['datetime'] = pd.to_datetime(df['datetime'])
```

Define the time-series evaluation function.

```
def timeseries_evaluation_metrics_func(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    #print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error
    (y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error
    (y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',
    end='\n\n')
```

Change the data into a format that Prophet accepts.

```
df = df.rename(columns={'datetime': 'ds', 'count': 'y'})
```

Train/test the split, and let's hold back two days of data (i.e., 48 records), which we can use to validate the results after training on past data. We need to do the same with exogenous and add it to the original DataFrame. Then we can use it further to train and validate the data.

```
validate = df[['ds','y','season','holiday','weather','temp',
'humidity','windspeed']].tail(48)
```

```
df.drop(df[['ds','y','season','holiday','weather','temp','
humidity','windspeed']].tail(48).index,inplace=True)
```

```
train = df[['ds','y','season','holiday','weather','temp','
humidity','windspeed']]
```

Let's initialize the `Prophet` class, then add exogenous variables one by one, and finally fit the model on the training data.

```
m = Prophet(yearly_seasonality=False)
m.add_regressor('season')
m.add_regressor('holiday')
m.add_regressor('weather')
m.add_regressor('temp')
m.add_regressor('humidity')
m.add_regressor('windspeed')
m.fit(train)
```

Prophet requires a future DataFrame to do any forecasting, so let's create one with a 48 frequency of `H` for the next 48 hours.

```
p = 48
future = m.make_future_dataframe(periods=p,freq='H',include_
history=False)
```

Add exogenous into the forecast DataFrame so that the model can use them while forecasting.

```
future['season'] = validate['season'].values
future['holiday'] = validate['holiday'].values
future['weather'] =  validate['weather'].values
future['temp'] = validate['temp'].values
future['humidity'] =  validate['humidity'].values
future['windspeed'] = validate['windspeed'].values
forecast = m.predict(future)
```

```
timeseries_evaluation_metrics_func(validate.y,forecast.yhat)
```

```
MSE is : 10526.397825382928
MAE is : 66.73451882424882
RMSE is : 102.59823500130462
MAPE is : 115.03252471019137
R2 is : 0.6822627187339608
```

Forecast the plot (Figure 8-8).

```
fig1 = m.plot(forecast)
```

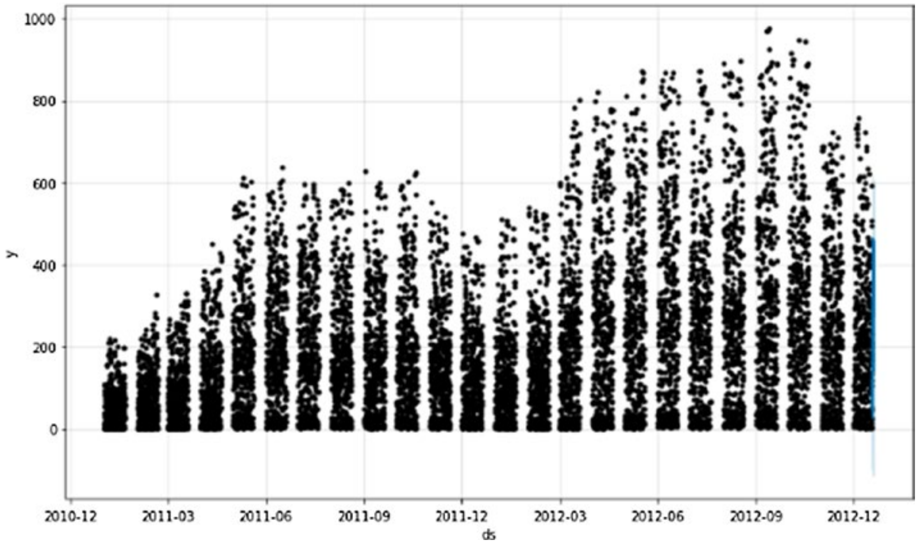


Figure 8-8. *Representation of forecast*

Let's plot the components of the forecast (Figure 8-9).

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

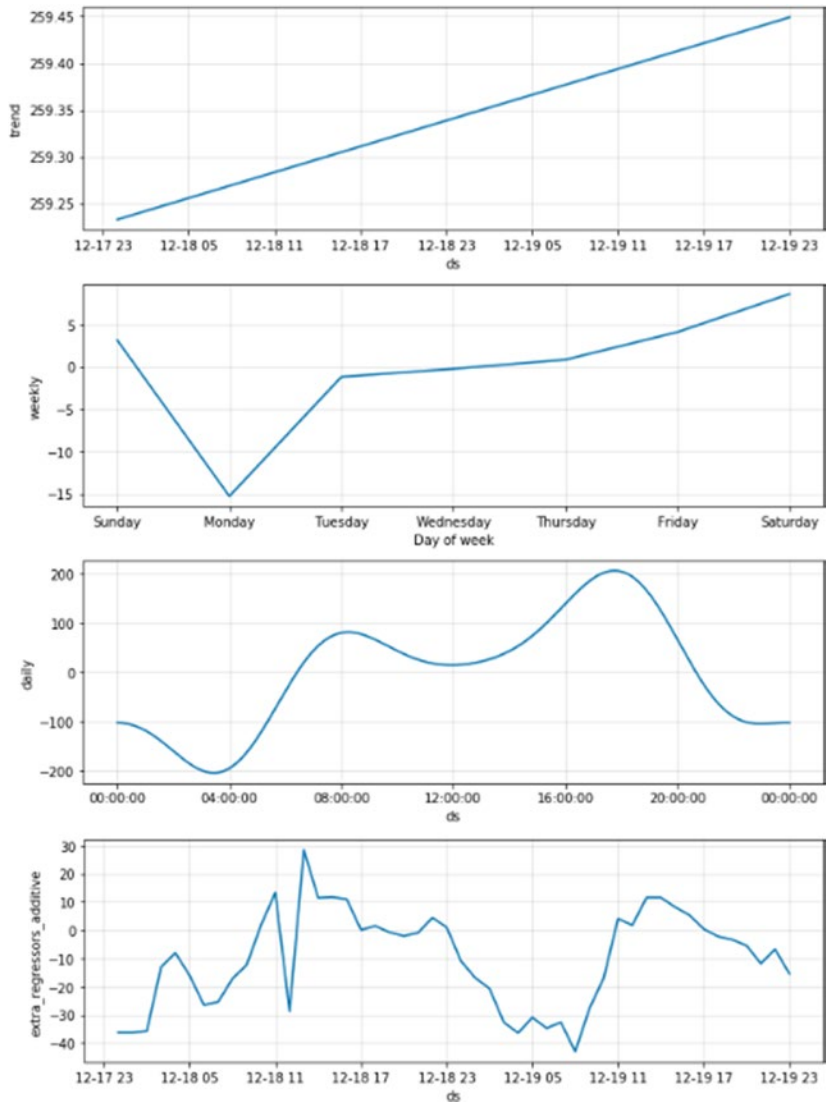


Figure 8-9. *Representation of forecasted components*

Plot the changepoints (Figure 8-10).

```
from fbprophet.plot import add_changepoints_to_plot
fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```

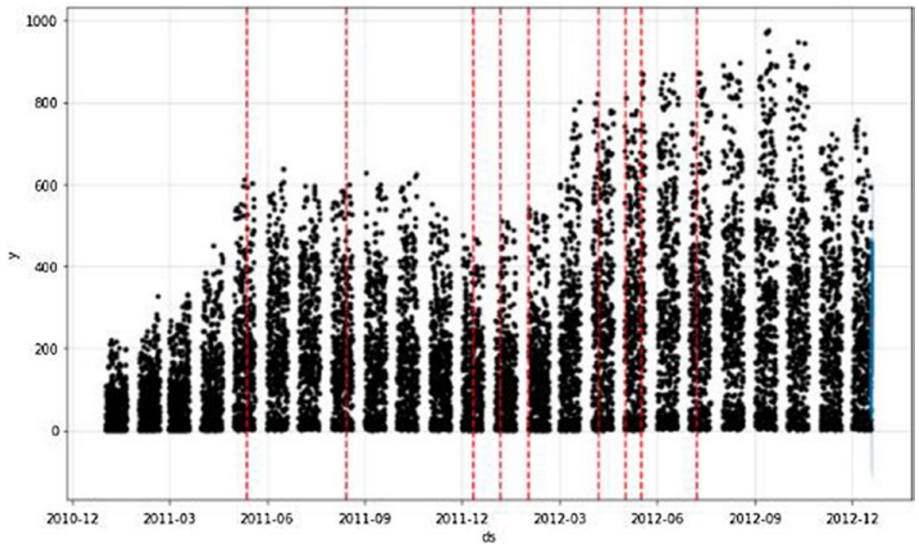


Figure 8-10. *Representation of trend changepoints*

Summary

In this chapter, you learned about the high-level math behind Prophet, how to create a basic model, how to apply log transformations to time-series data, how to add country holidays, and how to handle multivariate time-series data using `add_regressor`.