In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/conten
t/drive", force_remount=True).
```

Forecasting Timeseries Data Using Facebook FbProphet

Steps Required With FbProphet

1. Introduction And Installation
2. Data Preprocessing With Time Seires
3. Model Fitting
4. Obtaining The Forecasts
5. Plotting The Forecasts
6. Cross Validation
7. Computing Performance Metrics
8. Visualize the Performance MEtrics
9. Conclusions

In [2]:

```
### pip install pystan
### conda install -c conda-forge fbprophet
import pandas as pd
import fbprophet
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
df=pd.read_csv('/content/drive/MyDrive/Weather_Data/Final_Weather_Report.csv')
df.head(3)
```

Out[3]:

	Unnamed: 0	Formatted Date	Summary	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	(n
0	0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	
1	1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	
2	2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	

In [4]:
```
df.tail(2)
```

Out[4]:

	Unnamed: 0	Formatted Date	Summary	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Louc Cove

	Unnamed: 0	Formatted Date	Summary	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Louc Cove
27852	27852	2009-12-11 12:00:00.000 +0100	Foggy	5.000000	5.000000	0.93	1.6100	170.0	1.932	0.(
27853	27853	2009-12-11 13:00:00.000 +0100	Foggy	5.033333	5.033333	0.93	0.7406	218.0	1.932	0.(

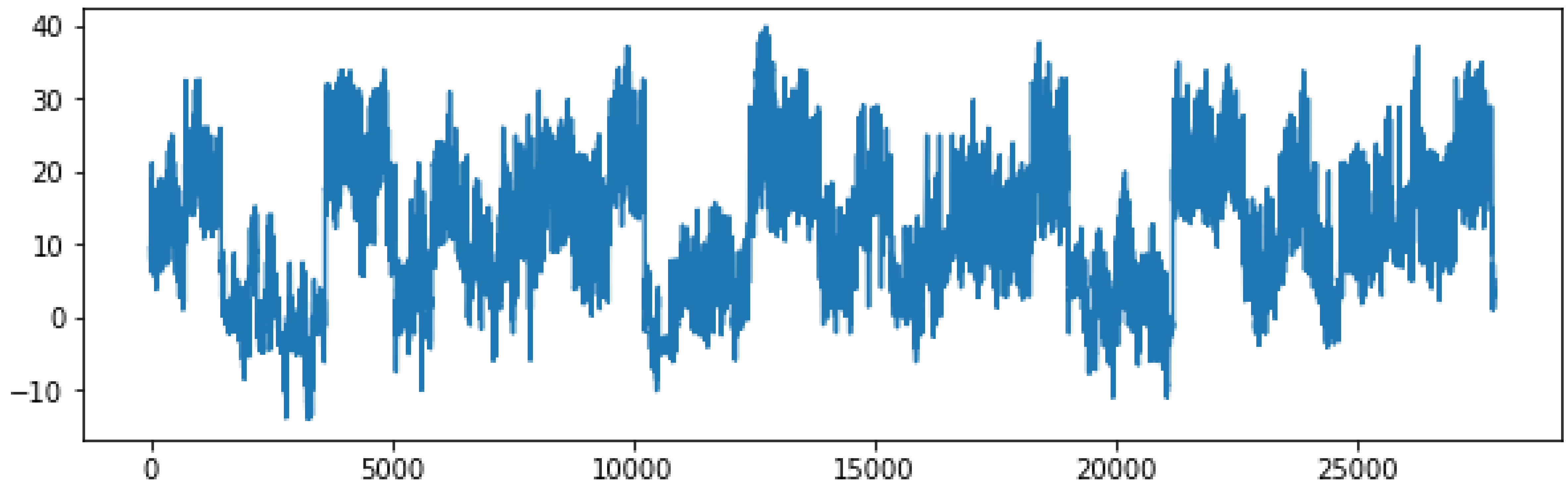
In [5]:
```
from fbprophet import Prophet
```

In [ ]:
```
dir(Prophet)
```

In [7]:
```
df['ds']=pd.to_datetime(df['Formatted Date'])
```

In [8]:
```
df['y'] = df['Temperature (C)']
```

In [9]:
```
plt.figure(figsize=[10,3])
df['y'].plot()
plt.show()
```



In [10]:
```
df = df[["ds","y"]]
```

In [11]:
```
df.head()
```

Out[11]:

	ds	y
0	2006-04-01 00:00:00+02:00	9.472222
1	2006-04-01 01:00:00+02:00	9.355556
2	2006-04-01 02:00:00+02:00	9.377778
3	2006-04-01 03:00:00+02:00	8.288889

	ds	y
4	2006-04-01 04:00:00+02:00	8.755556

In [12]:
```
df['ds'] = pd.to_datetime(df['ds'], utc=True)
```

In [13]:
```
df['ds'] = df['ds'].dt.tz_localize(None)
```

In [14]:
```
### intiialize the Model
model=Prophet()
model.fit(df)
```

```
INFO:numexpr.utils:NumExpr defaulting to 2 threads.
```

Out[14]:
```
<fbprophet.forecaster.Prophet at 0x7f15038aa7d0>
```

In [15]:
```
model
```

Out[15]:
```
<fbprophet.forecaster.Prophet at 0x7f15038aa7d0>
```

In [16]:
```
model.seasonalities
```

Out[16]:
```
OrderedDict([('yearly',
              {'condition_name': None,
               'fourier_order': 10,
               'mode': 'additive',
```

```
                 'period': 365.25,
                 'prior_scale': 10.0}),
               ('weekly',
                {'condition_name': None,
                 'fourier_order': 3,
                 'mode': 'additive',
                 'period': 7,
                 'prior_scale': 10.0}),
               ('daily',
                {'condition_name': None,
                 'fourier_order': 4,
                 'mode': 'additive',
                 'period': 1,
                 'prior_scale': 10.0})])
```

In [17]:
```
model.component_modes
```

Out[17]:
```
{'additive': ['yearly',
  'weekly',
  'daily',
  'additive_terms',
  'extra_regressors_additive',
  'holidays'],
 'multiplicative': ['multiplicative_terms', 'extra_regressors_multiplicative']}
```

In [18]:
```
#### Create future dates of 365 days
future_dates=model.make_future_dataframe(periods=365)
```

In [19]:
```
df.tail()
```

Out[19]:

	ds	y
27849	2009-12-11 08:00:00	3.888889
27850	2009-12-11 09:00:00	4.011111
27851	2009-12-11 10:00:00	5.000000
27852	2009-12-11 11:00:00	5.000000
27853	2009-12-11 12:00:00	5.033333

Forecasting the ds value for the next 365 future days.

In [20]:

```
future_dates
```

Out[20]:

	ds
0	2005-12-31 23:00:00
1	2006-01-01 00:00:00
2	2006-01-01 01:00:00
3	2006-01-01 02:00:00
4	2006-01-01 03:00:00
...	...
28214	2011-07-29 21:00:00
28215	2011-07-30 21:00:00

	ds
28216	2011-07-31 21:00:00
28217	2011-08-01 21:00:00
28218	2011-08-02 21:00:00

28219 rows × 1 columns

In [21]:

```
### Prediction
prediction=model.predict(future_dates)
```

In [22]:

```
prediction.head()
```

Out[22]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	addit
0	2005-12-31 23:00:00	7.823828	-13.054277	-2.543698	7.823828	7.823828	-15.446207	-15.446207	
1	2006-01-01 00:00:00	7.823831	-13.055648	-2.913369	7.823831	7.823831	-15.851381	-15.851381	
2	2006-01-01 01:00:00	7.823835	-13.463634	-3.705616	7.823835	7.823835	-16.248033	-16.248033	

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	addit
3	2006-01-01 02:00:00	7.823839	-13.878894	-3.719839	7.823839	7.823839	-16.603030	-16.603030	
4	2006-01-01 03:00:00	7.823842	-13.392342	-3.949903	7.823842	7.823842	-16.794907	-16.794907	

In [23]:

```
prediction[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

Out[23]:

	ds	yhat	yhat_lower	yhat_upper
28214	2011-07-29 21:00:00	25.109676	16.725553	33.771885
28215	2011-07-30 21:00:00	25.010906	16.952794	33.655577
28216	2011-07-31 21:00:00	25.064065	16.672337	33.869137
28217	2011-08-01 21:00:00	25.097503	16.574115	33.756381
28218	2011-08-02 21:00:00	24.698355	16.445429	33.138730

In [24]:

```
prediction[['ds','yhat','yhat_lower','yhat_upper']].head()
```

Out[24]:

ds	yhat	yhat_lower	yhat_upper

	ds	yhat	yhat_lower	yhat_upper
0	2005-12-31 23:00:00	-7.622379	-13.054277	-2.543698
1	2006-01-01 00:00:00	-8.027550	-13.055648	-2.913369
2	2006-01-01 01:00:00	-8.424198	-13.463634	-3.705616
3	2006-01-01 02:00:00	-8.779192	-13.878894	-3.719839
4	2006-01-01 03:00:00	-8.971064	-13.392342	-3.949903

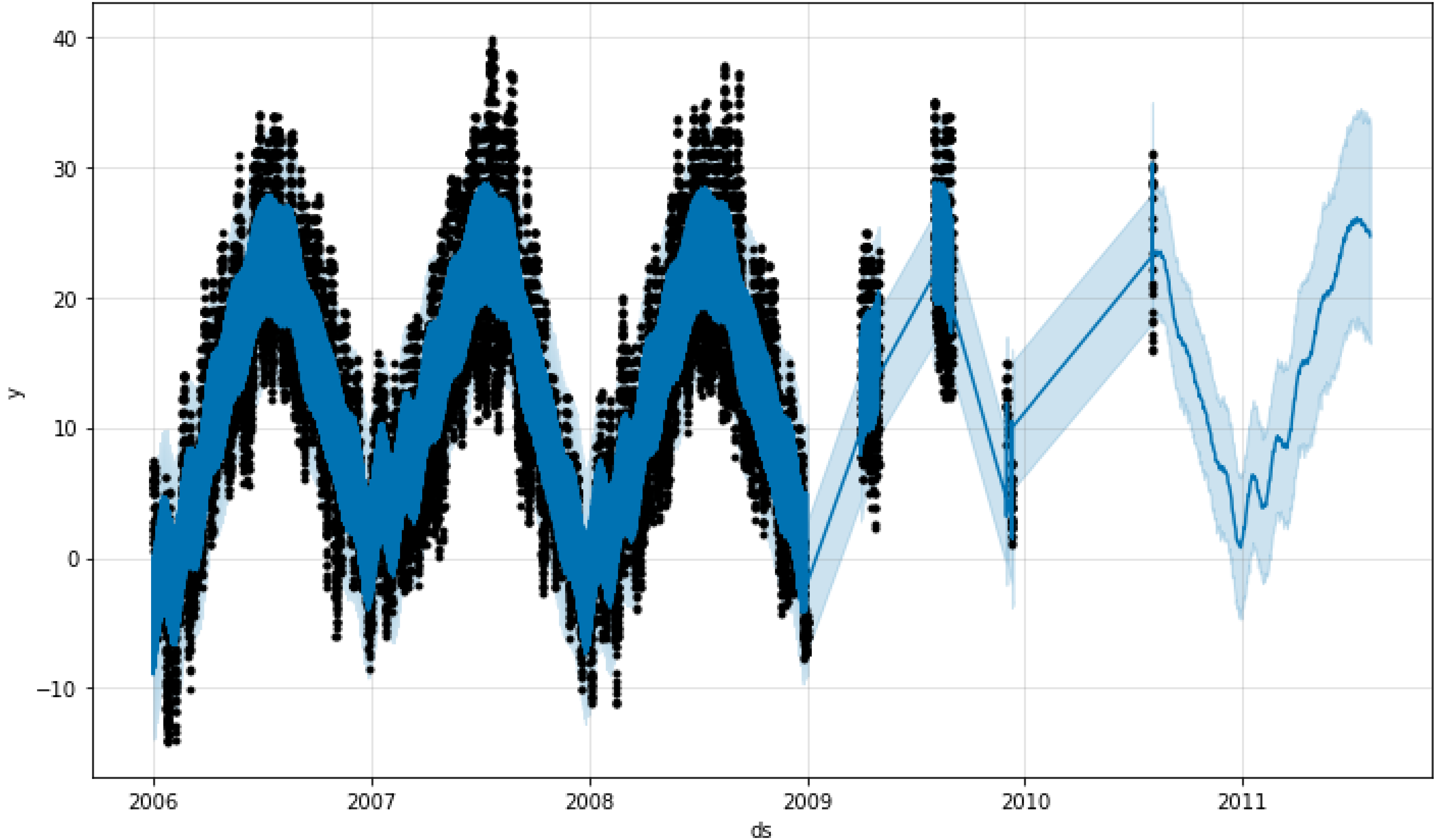
Plotting the Forecasts

Prophet has an inbuilt feature that enables us to plot the forecasts we just generated. This is achieved using model.plot() and passing in our forecasts as the argument. The blue line in the graph represents the predicted values while the black dots represents the data in our dataset.

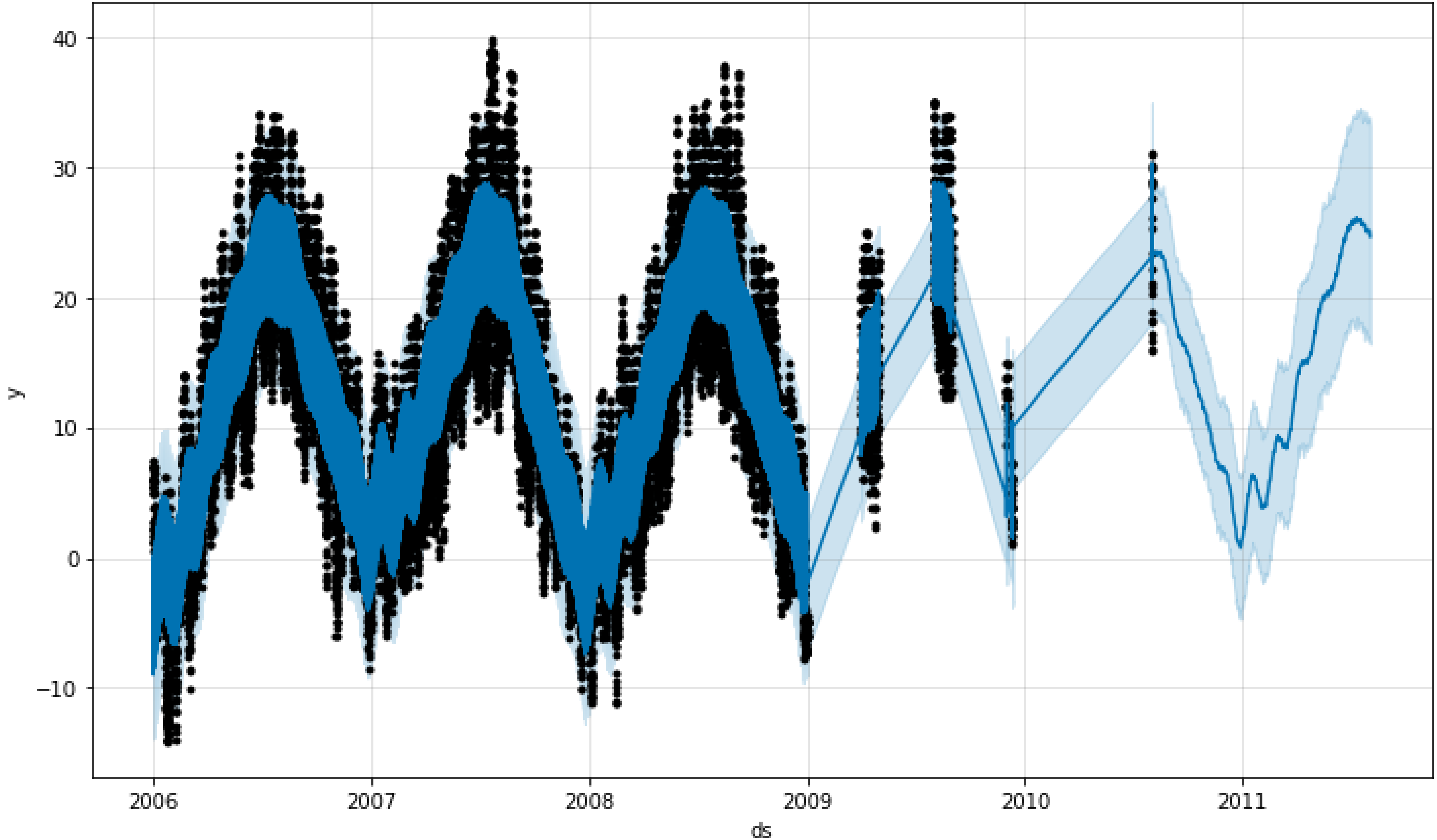
In [25]:

```
#### plot the predicted projection
model.plot(prediction)
```

Out[25]:



40
30
20
10
0
−10
y
2006
2007
2008
2009
2010
2011
ds

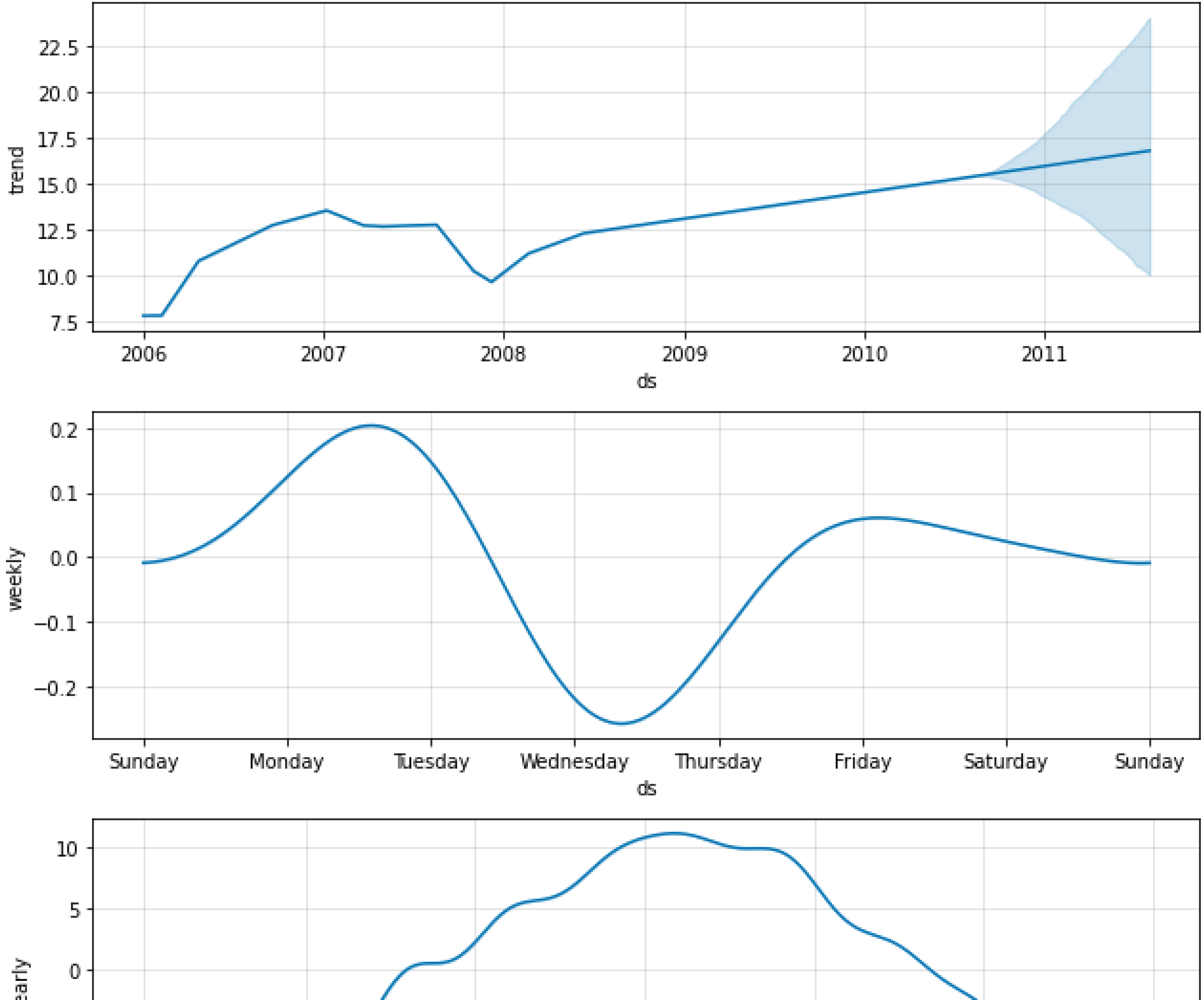


Weather Trends **daily** , weekly **wise**

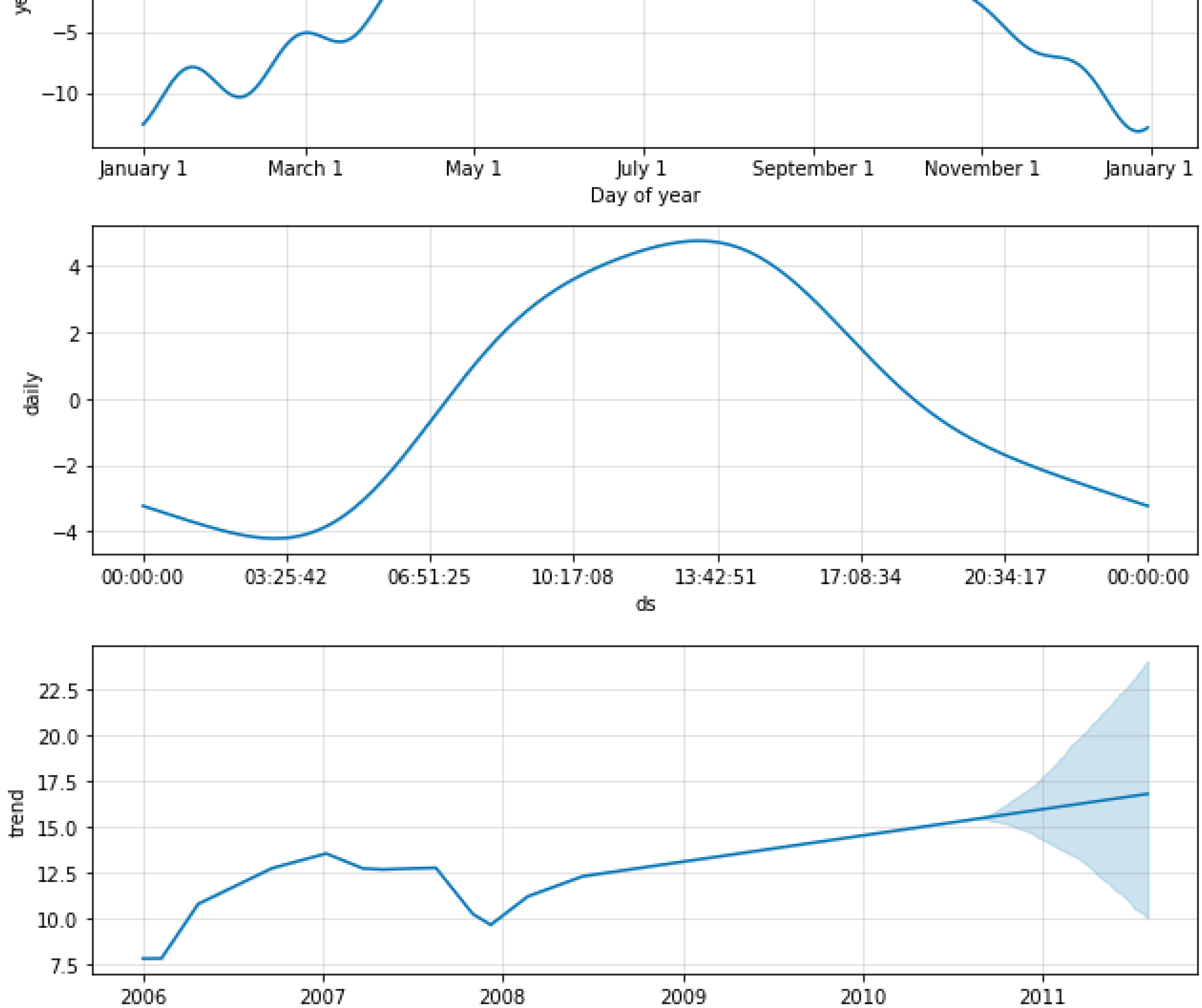
In [26]:

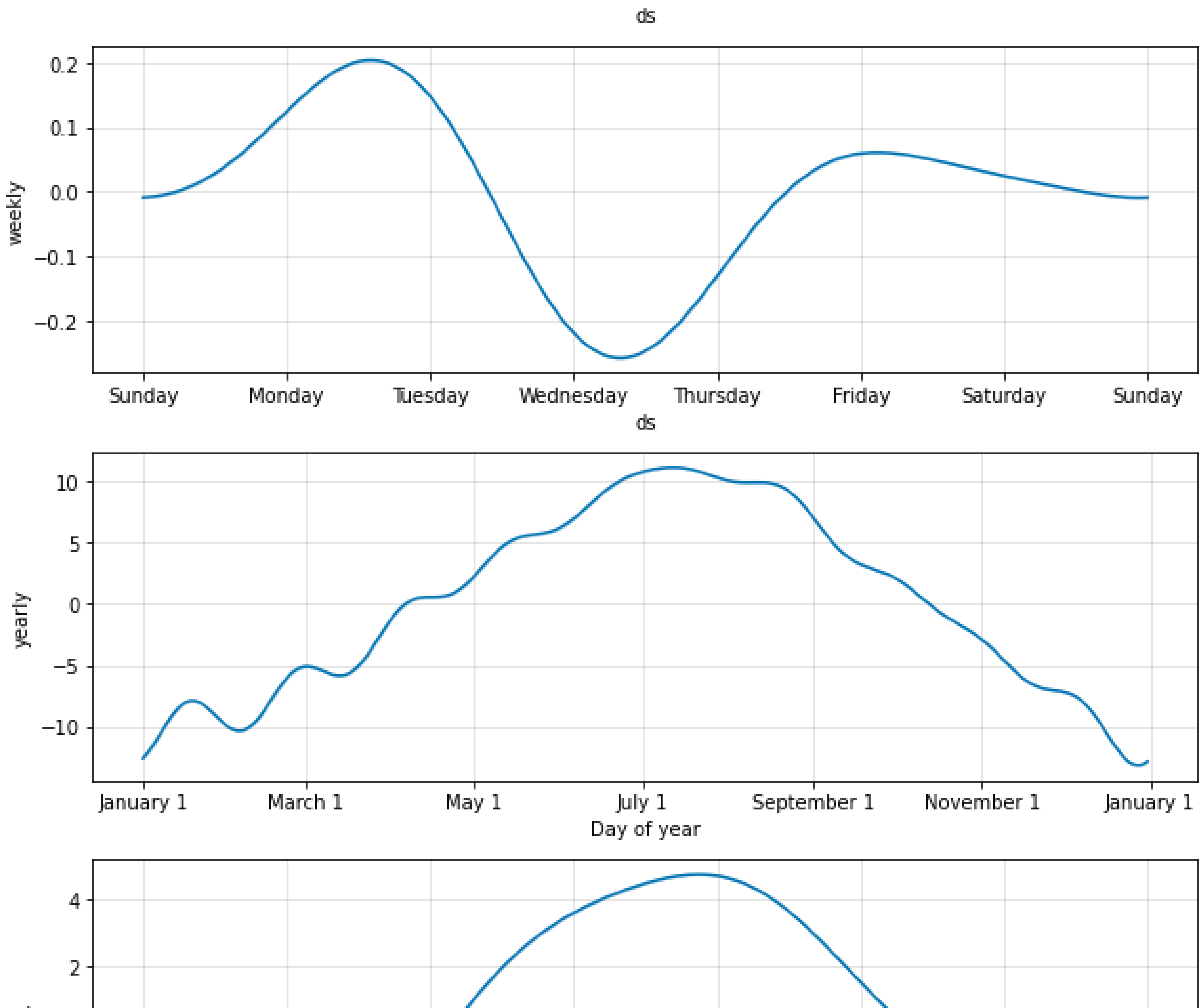
```
##### Visualize Each Components[Trends,Weekly]
model.plot_components(prediction)
```

Out[26]:

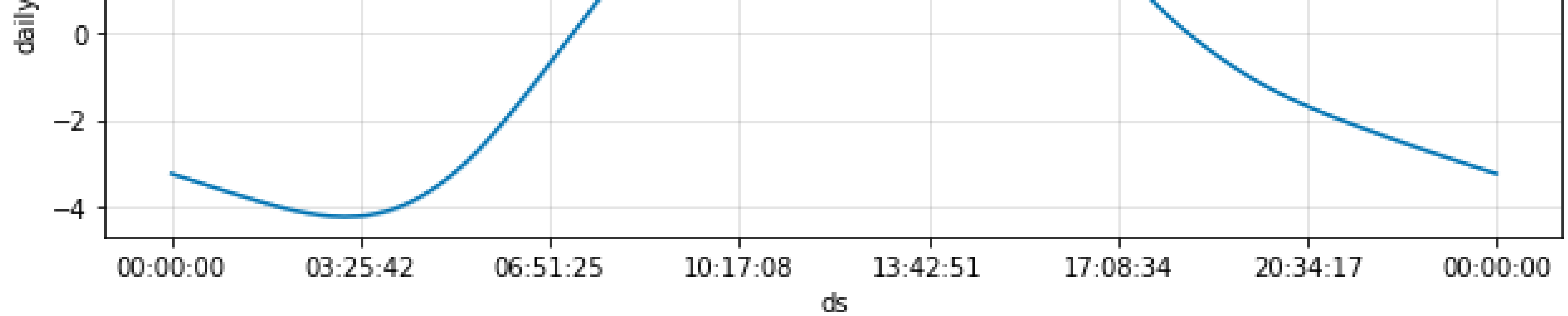


22.5
20.0
17.5
15.0
12.5
10.0
7.5
trend
2006
2007
2008
2009
2010
2011
ds
0.2
0.1
0.0
−0.1
−0.2
weekly
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
ds
10
5
0


−5
−10
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
daily
4
2
0
−2
−4
00:00:00
03:25:42
06:51:25
10:17:08
13:42:51
17:08:34
20:34:17
00:00:00
ds
trend
22.5
20.0
17.5
15.0
12.5
10.0
7.5
2006
2007
2008
2009
2010
2011



ds
weekly
0.2
0.1
0.0
−0.1
−0.2
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
ds
yearly
10
5
0
−5
−10
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
4
2



Cross Validation

Next let's measure the forecast error using the historical data. We'll do this by comparing the predicted values with the actual values. In order to perform this operation we select cut of points in the history of the data and fit the model with data upto that cut off point. Afterwards we compare the actual values to the predicted values. The cross_validation method allows us to do this in Prophet. This method take the following parameters as explained below:

1. horizon the forecast horizon
2. initial the size of the initial training period
3. period the spacing between cutoff dates

In [27]:

```
df.shape
```

Out[27]:

```
(27854, 2)
```

In [28]:

```
from fbprophet.diagnostics import cross_validation
```

In [29]:
```
df_cv=cross_validation(model,horizon="365 days",period='180 days',initial='1095 days')
```

```
INFO:fbprophet:Making 2 forecasts with cutoffs between 2009-02-03 21:00:00 and 2009-08-02 21:00:00
```

In [30]:
```
df_cv.head()
```

Out[30]:

	ds	y
0	2006-03-31 22:00:00	9.472222
1	2006-03-31 23:00:00	9.355556
2	2006-04-01 00:00:00	9.377778
3	2006-04-01 01:00:00	8.288889
4	2006-04-01 02:00:00	8.755556

Obtaining the Performance Metrics

We use the performance_metrics utility to compute the Mean Squared Error(MSE), Root Mean Squared Error(RMSE),Mean Absolute Error(MAE), Mean Absolute Percentage Error(MAPE) and the coverage of the the yhat_lower and yhat_upper estimates.

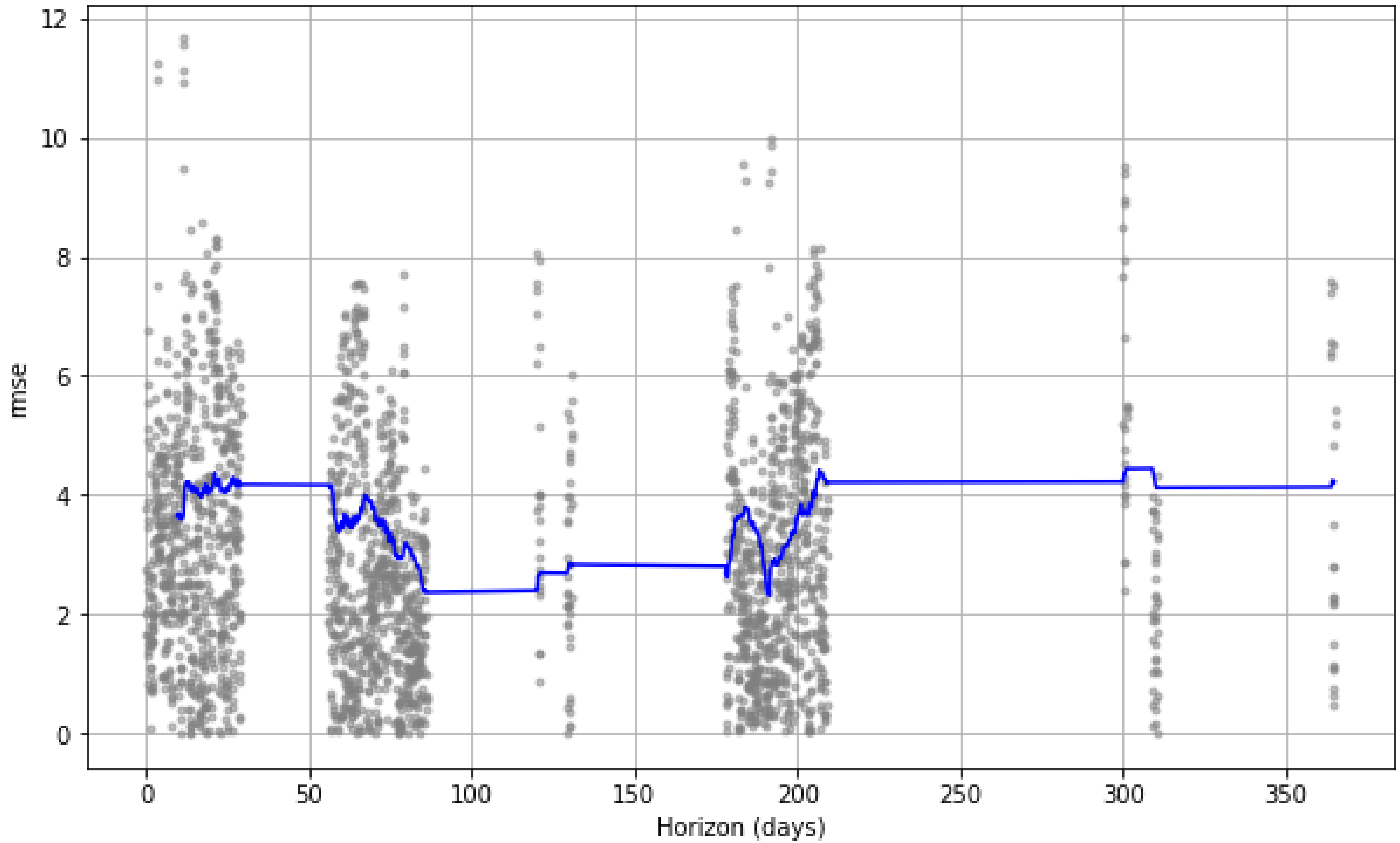
In [33]:
```
from fbprophet.diagnostics import performance_metrics
df_performance=performance_metrics(df_cv)
df_performance.head()
```

Out[33]:

	horizon	mse	rmse	mae	mape	mdape	coverage
0	9 days 14:00:00	13.292055	3.645827	3.256107	0.155322	0.149862	0.882609
1	9 days 15:00:00	13.252874	3.640450	3.249697	0.155075	0.149862	0.882609
2	9 days 16:00:00	13.252334	3.640375	3.249561	0.155027	0.149862	0.882609
3	9 days 17:00:00	13.234191	3.637883	3.245626	0.154834	0.149862	0.882609
4	9 days 18:00:00	13.229400	3.637224	3.244011	0.154726	0.149862	0.882609

In [34]:

```
from fbprophet.plot import plot_cross_validation_metric
fig=plot_cross_validation_metric(df_cv,metric='rmse')
```




```
In [ ]:
```