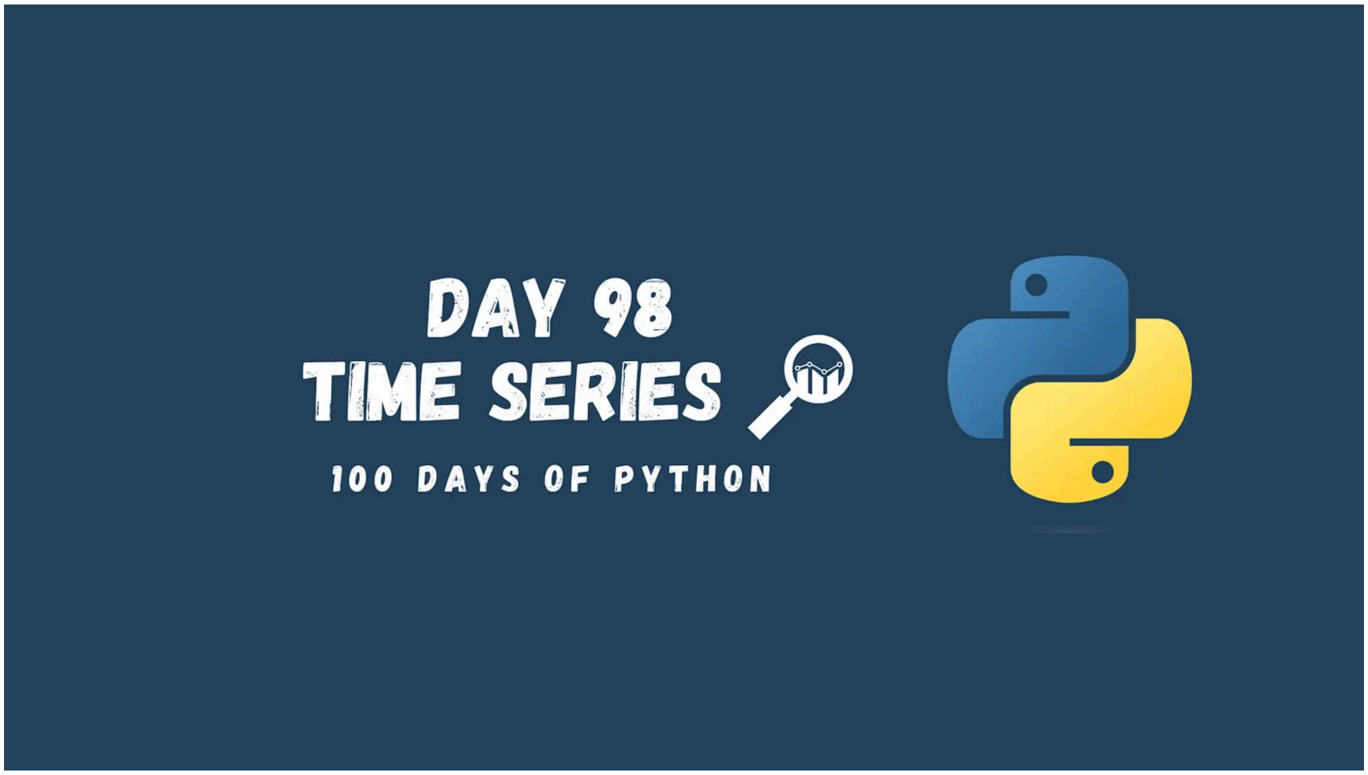
Time Series Analysis with Python using Prophet (98/100 Days of Python)

Martin Mirakyan Follow 5 min read · Apr 9, 2023

220



Day 98 of the “100 Days of Python” blog post series covering time series analysis with Prophet

Time series analysis is a valuable skill for anyone working with data that changes over time, such as sales, stock prices, or even climate trends. In this tutorial, we will introduce the powerful Python library, Prophet, developed by Facebook for time series forecasting. This tutorial will provide a step-by-step guide to using Prophet for time series analysis, from data preprocessing to model evaluation.

Introduction to Time Series Analysis

Time series analysis is the process of analyzing data points collected sequentially over time to understand underlying patterns, trends, and seasonality. This analysis can be used to make predictions about future data points.

What is Prophet?

Prophet is an open-source library developed by Facebook's Core Data Science team for time series forecasting. It provides an easy-to-use interface and works well with missing data, outliers, and seasonality. Prophet is particularly useful for business applications, as it can handle holidays and other special events that affect data trends.

Installation and Setup

To install Prophet, run the following command:

```
# For prophet
pip install prophet

# You will also need the following libraries for data manipulation and visualiza
pip install pandas matplotlib numpy
```

Data Preprocessing

Before using Prophet, we need to preprocess the data. Ensure your dataset has two columns: “ds” for dates and “y” for the target variable. For example, if you are working with monthly sales data, “ds” would contain the dates and “y” the sales figures. Your dataset should look like this, which represents daily sales for three years:

```
ds,y
2018-01-01,216
2018-01-02,231
2018-01-03,268
2018-01-04,215
...
2018-12-30,329
2018-12-31,257
2019-01-01,188
2019-01-02,329
...
2019-12-30,258
2019-12-31,317
2020-01-01,282
2020-01-02,287
...
2020-12-28,342
2020-12-29,370
2020-12-30,284
2020-12-31,364
```

Now, let’s import the necessary libraries and load the dataset:

```
import pandas as pd

data = pd.read_csv('your_dataset.csv')    # Read dataset
data['ds'] = pd.to_datetime(data['ds'])   # Convert the 'ds' column to a datetim
```

Creating a Prophet Model

Once your data is ready, you can create a Prophet model

```
from prophet import Prophet

model = Prophet()      # Initialize the model
model.fit(data)        # Fit the model to the data
```

Forecasting with Prophet

To make predictions with Prophet, you first need to create a future DataFrame with the desired frequency and horizon:

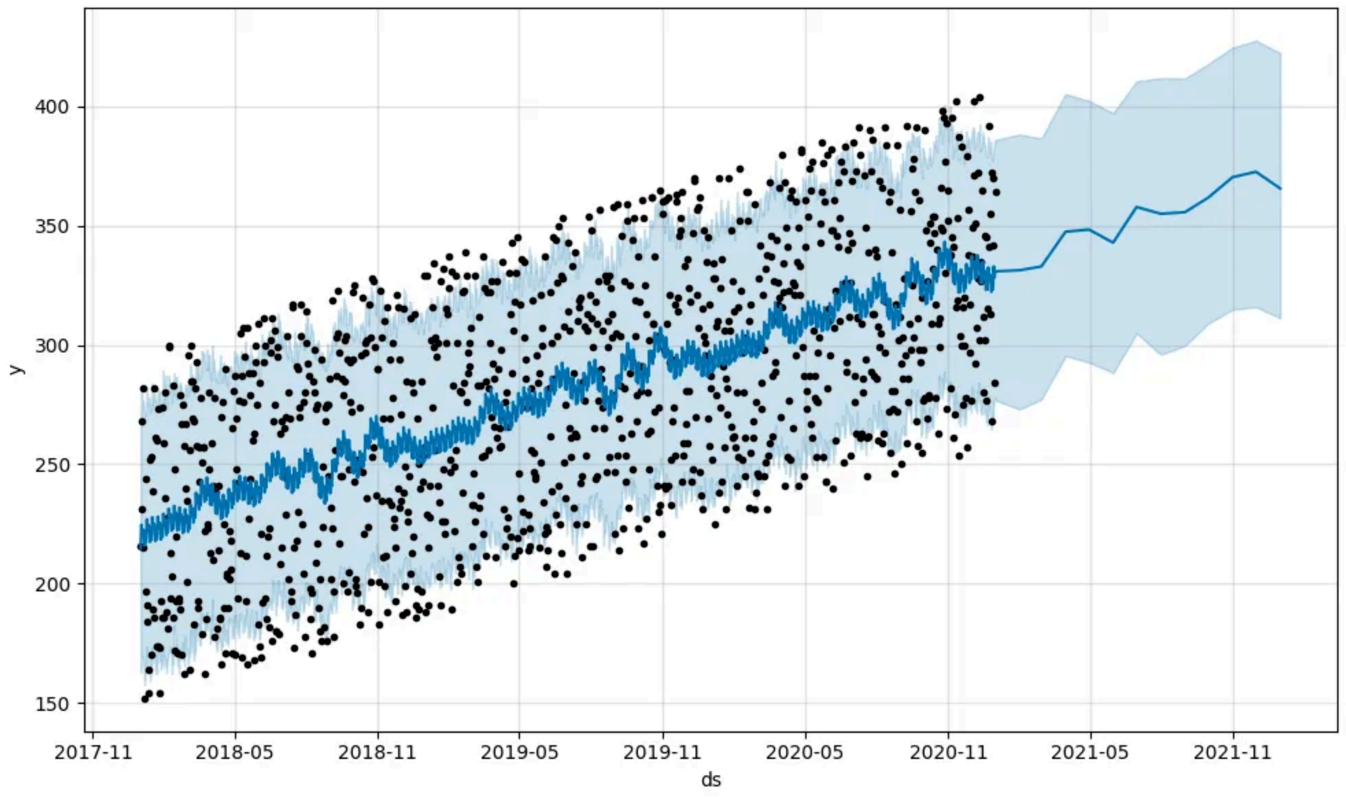
```
future = model.make_future_dataframe(periods=12, freq='M')   # Create a future D
forecast = model.predict(future)                             # Generate the fore
```

Model Evaluation and Diagnostics

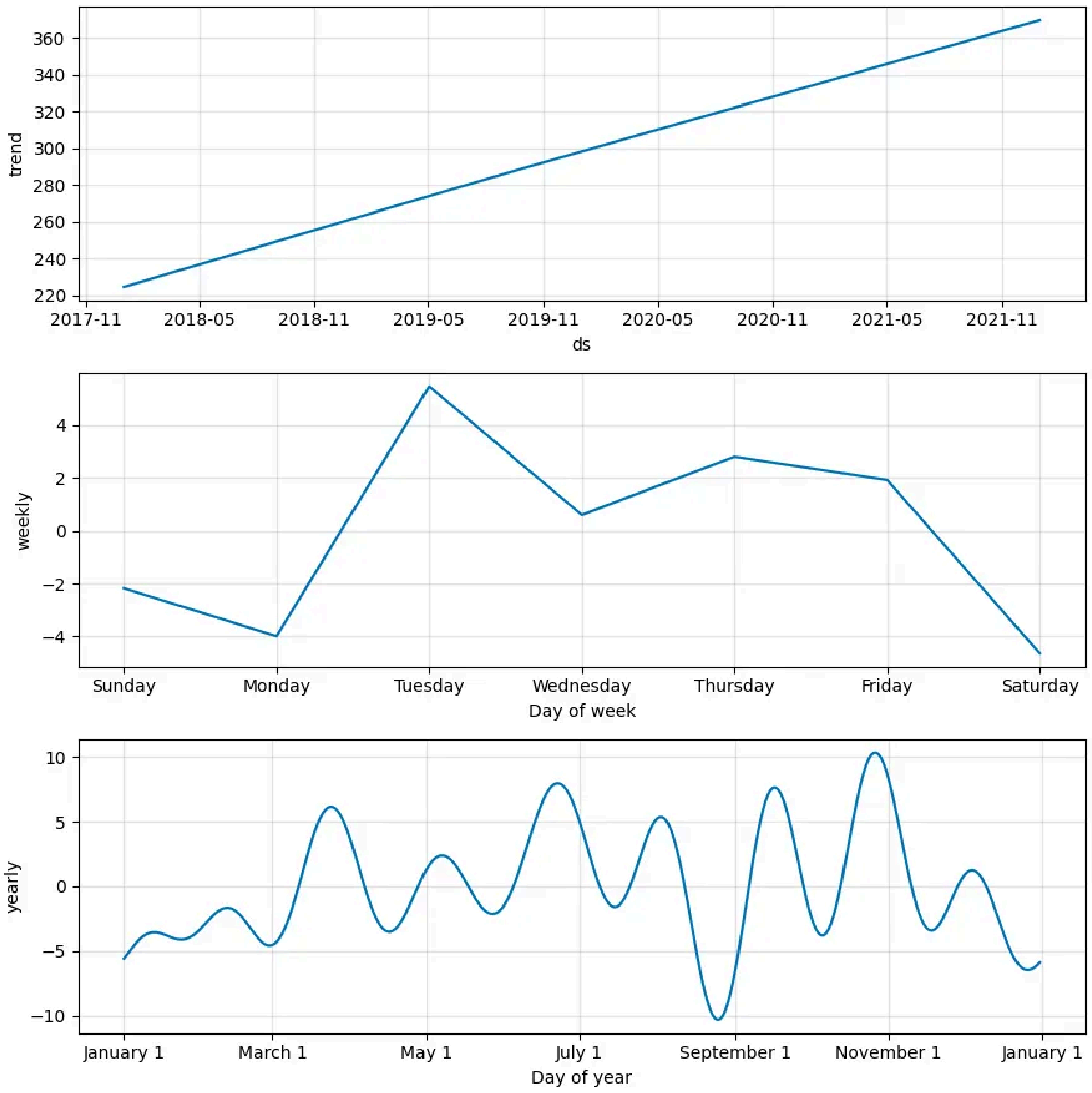
To evaluate the model, you can plot the forecast and its components:

```
from prophet.plot import plot, plot_components
from matplotlib import pyplot as plt

plot(model, forecast)               # Plot the forecast
plot_components(model, forecast)    # Plot the forecast components
plt.show()
```



Forecast plot



Components

Cross-Validation and Performance Metrics

Prophet offers a built-in cross-validation function to evaluate the model's performance. You can use different performance metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE), to assess the model's accuracy.

```
import numpy as np
from prophet.diagnostics import cross_validation, performance_metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Perform cross-validation
df_cv = cross_validation(model, initial='730 days', period='180 days', horizon='

# Calculate performance metrics
df_metrics = performance_metrics(df_cv)

# Calculate MAE, MSE, and RMSE
mae = mean_absolute_error(df_cv['y'], df_cv['yhat'])
mse = mean_squared_error(df_cv['y'], df_cv['yhat'])
rmse = np.sqrt(mse)

print(f'Mean Absolute Error: {mae:.2f}')
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
```

```
Mean Absolute Error: 37.07
Mean Squared Error: 1865.31
Root Mean Squared Error: 43.19
```

This dataset has 3 years (1096 days) of daily sales data. The `initial` parameter is set to '730 days', which means the initial training period consists of the first two years of data (2018 and 2019). The `period` parameter is set to '180 days', which indicates that the cross-validation will be performed every 180 days (approximately every 6 months). The `horizon` parameter is set to '365 days', meaning that each cross-validation step will generate forecasts for the next 365 days (1 year).

By examining the performance metrics, you can gain a better understanding of how well your model is performing and make adjustments as needed.

Hyperparameter Tuning

To improve the model's performance, you can experiment with different hyperparameters. Some of the key hyperparameters to consider are:

- `seasonality_mode` : 'additive' (default) or 'multiplicative'
- `changepoint_prior_scale` : Controls the flexibility of the trend; higher values allow for more fluctuations
- `seasonality_prior_scale` : Controls the strength of seasonality; higher values lead to more flexible seasonality

To tune these hyperparameters, you can use a grid search approach or more advanced optimization techniques such as Bayesian optimization. Here, we will demonstrate a simple grid search for hyperparameter tuning:

```
from prophet.diagnostics import cross_validation
from prophet import Prophet
from sklearn.metrics import mean_absolute_error

# Define the hyperparameter grid
param_grid = {
    'seasonality_mode': ['additive', 'multiplicative'],
    'changepoint_prior_scale': [0.01, 0.1, 0.5],
    'seasonality_prior_scale': [1, 10, 30],
}


# Helper function to evaluate the model
def evaluate_model(model, metric_func):
    df_cv = cross_validation(model, initial='730 days', period='180 days', horiz
    return metric_func(df_cv['y'], df_cv['yhat'])


# Grid search
```

```
best_params = {}
best_score = float('inf')

for mode in param_grid['seasonality_mode']:
    for cps in param_grid['changepoint_prior_scale']:
        for sps in param_grid['seasonality_prior_scale']:
            # Create a model with the current hyperparameters
            model = Prophet(seasonality_mode=mode, changepoint_prior_scale=cps,
            model.fit(data)

            # Evaluate the model using Mean Absolute Error (MAE)
            score = evaluate_model(model, mean_absolute_error)

            # Update best parameters if necessary
            if score < best_score:
                best_score = score
                best_params = {
                    'seasonality_mode': mode,
                    'changepoint_prior_scale': cps,
                    'seasonality_prior_scale': sps
                }

print(best_params)
print(best_score)

# Create the best model with the optimal hyperparameters
best_model = Prophet(**best_params)
best_model.fit(data)
```

```
{'seasonality_mode': 'additive', 'changepoint_prior_scale': 0.01, 'seasonality_p
36.797223455418425
```

Using grid search or other optimization techniques, you can fine-tune your model's hyperparameters to achieve better forecasting performance. In our case, the Mean Absolute Error (MAE) improved from 37.07 to 36.79.

What's next?

- If you found this story valuable, please consider clapping multiple times (this really helps a lot!)

- **Hands-on Practice:** Free Python Course
- **Full series:** 100 Days of Python
- **Previous topic:** Creating Custom ChatGPT Using the OpenAI API
- **Next topic:** Developing Your First Flask Application in Python

Prophet · Time Series Analysis · Time Series Forecasting · Fbprophet

Written by Martin Mirakyan

706 followers · 3 following

Follow

Software Engineer | Machine Learning | Founder of Profound Academy (https://profound.academy)

No responses yet

Write a response

What are your thoughts?