Time Series Forecasting with Facebook Prophet

Introduction

In this exercise, you will use **Facebook Prophet** to forecast Microsoft's closing stock price.

You will:

1. Load and preprocess the data into Prophet's required format
2. Initialize, configure, and fit a Prophet model
3. Generate future forecasts and visualise results
4. Evaluate forecast accuracy with Mean Absolute Error (MAE)

Fill in every `_____` blank to complete the code.

Dataset: Microsoft Stock — Time Series Analysis — download `Microsoft_Stock.csv` and place it in the same directory as this notebook.

The dataset contains daily OHLCV data for Microsoft (MSFT) with columns: `Date`, `Open`, `High`, `Low`, `Close`, `Volume`.

Step 1: Import Libraries

We need `pandas` for data handling, `matplotlib` for plotting, and `Prophet` for time series forecasting.

Task: Import `Prophet` from the correct module.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt

from prophet import Prophet
```

Step 2: Load and Preprocess the Dataset

Prophet requires a DataFrame with exactly two columns:

- `ds` — the datestamp column
- `y` — the target value to forecast

We will forecast the **closing price** of Microsoft stock.

Hint: The relevant columns in this dataset are `Date` and `Close`.

In [2]:
```
df = pd.read_csv("Microsoft_Stock.csv")

df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Volume
0	4/1/2015 16:00:00	40.60	40.76	40.31	40.72	36865322
1	4/2/2015 16:00:00	40.66	40.74	40.12	40.29	37487476
2	4/6/2015 16:00:00	40.34	41.78	40.18	41.55	39223692
3	4/7/2015 16:00:00	41.61	41.91	41.31	41.53	28809375
4	4/8/2015 16:00:00	41.48	41.69	41.04	41.42	24753438

In [3]:
```
df = df[["Date", "Close"]]
df.columns = ["ds", "y"]
```

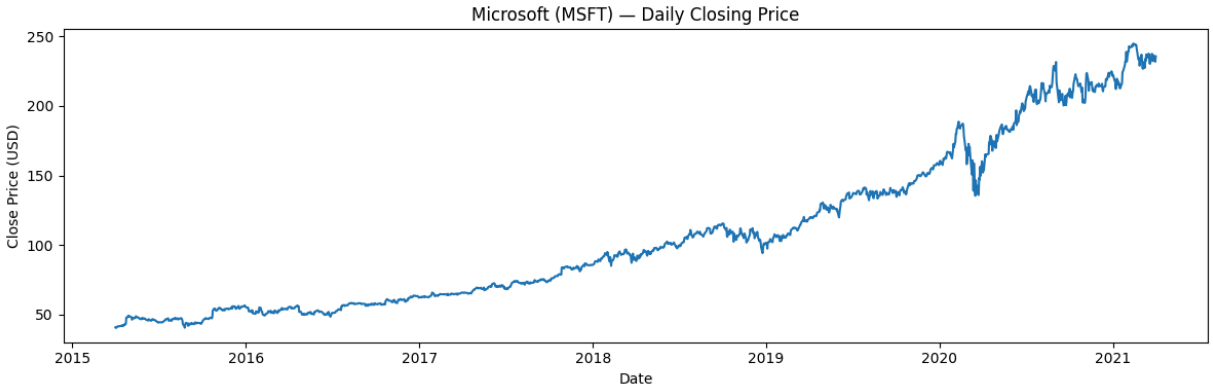
In [4]:
```
# Ensure 'ds' is datetime type
df['ds'] = pd.to_datetime(df['ds'])

# Sort by date
df = df.sort_values('ds').reset_index(drop=True)

print(f"Date range: {df['ds'].min()} to {df['ds'].max()}")
print(f"Number of data points: {len(df)}")
```

```
Date range: 2015-04-01 16:00:00 to 2021-03-31 16:00:00
Number of data points: 1511
```

In [5]:
```
# Check for missing values
print(df.isnull().sum())

# Drop missing values if any
df = df.dropna()
```

```
ds    0
y     0
dtype: int64
```

In [6]:
```
# Visualise the historical closing price
plt.figure(figsize=(12, 4))
plt.plot(df['ds'], df['y'])
plt.xlabel('Date')
plt.ylabel('Close Price (USD)')
plt.title('Microsoft (MSFT) – Daily Closing Price')
plt.tight_layout()
plt.show()
```



Step 3: Initialize and Fit the Prophet Model

Prophet accepts several configuration parameters. A common one is `yearly_seasonality`, which tells the model whether to fit a yearly seasonal component.

Task: Create a Prophet model with `yearly_seasonality` set to `True`, then fit it to the data.

In [7]:
```
model = Prophet(yearly_seasonality=True)

model.fit(df)
```

Out[7]:
```
<prophet.forecaster.Prophet at 0x10c37da60>
```

Step 4: Generate the Forecast

We create a dataframe of future dates and use the trained model to predict.

Task: Generate a future dataframe covering 90 days beyond the training data.

Note: Stock markets are closed on weekends. By default, `make_future_dataframe` generates calendar days (including weekends). We can pass `freq='B'` to generate only **business days**.

In [8]:
```
future = model.make_future_dataframe(periods=90, freq='B')

print(f"Future dataframe shape: {future.shape}")
future.tail()
```

```
Future dataframe shape: (1601, 1)
```

Out[8]:

	ds
1596	2021-07-29 16:00:00
1597	2021-07-30 16:00:00
1598	2021-08-02 16:00:00
1599	2021-08-03 16:00:00
1600	2021-08-04 16:00:00

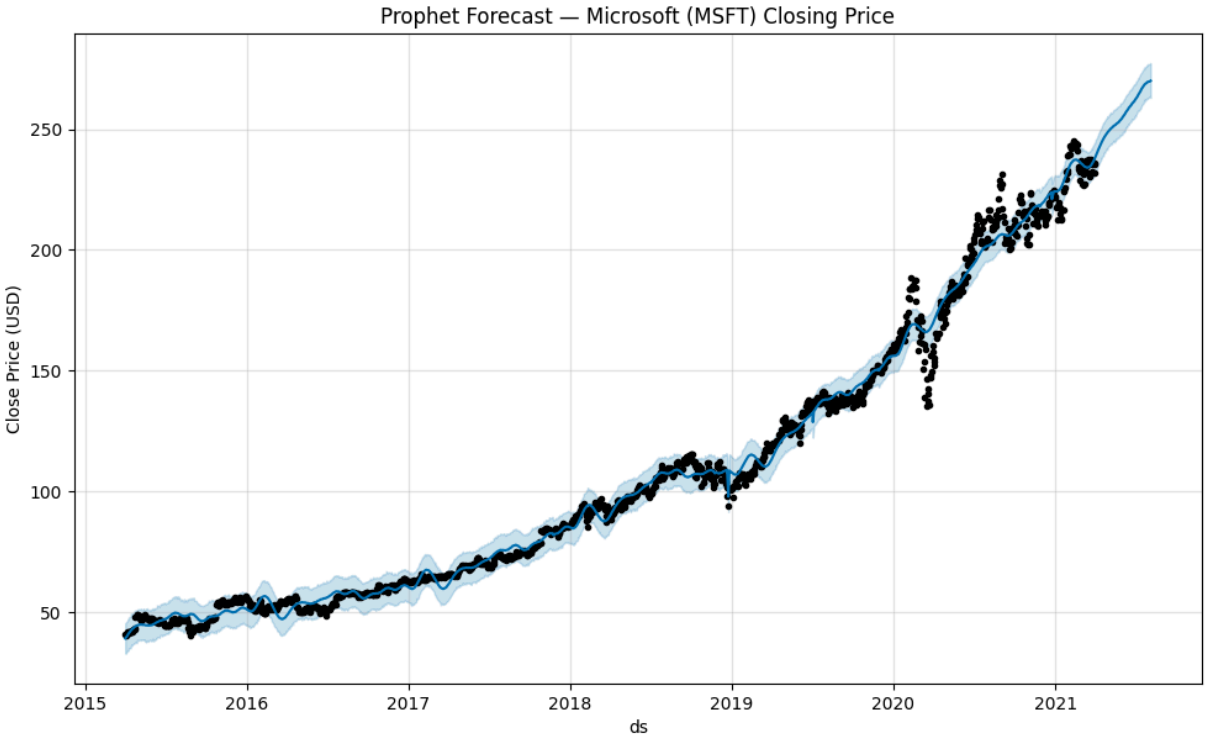
In [9]:
```
# Generate the forecast
forecast = model.predict(future)

# Inspect key forecast columns
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[9]:

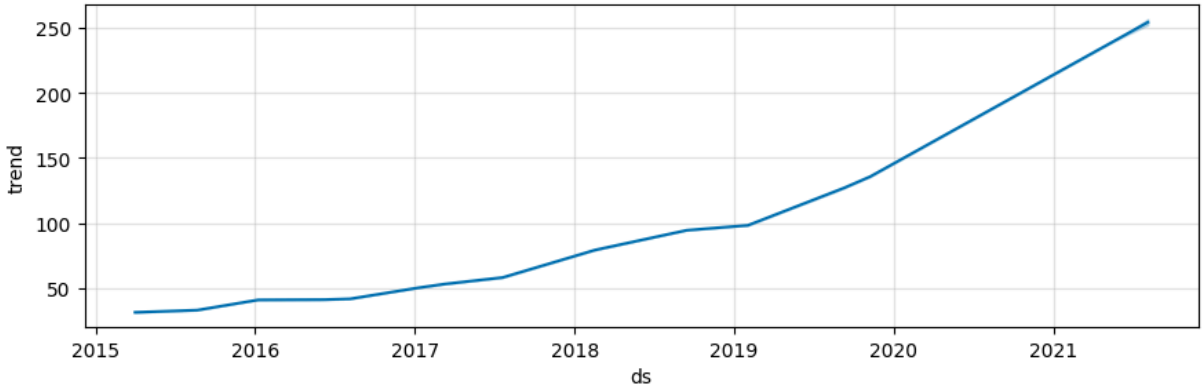
	ds	yhat	yhat_lower	yhat_upper
1596	2021-07-29 16:00:00	269.615859	262.871134	276.667741
1597	2021-07-30 16:00:00	269.677017	262.981906	276.886238
1598	2021-08-02 16:00:00	269.797029	263.620519	277.003209
1599	2021-08-03 16:00:00	269.956245	263.189859	277.431183
1600	2021-08-04 16:00:00	270.073899	263.020765	277.064481

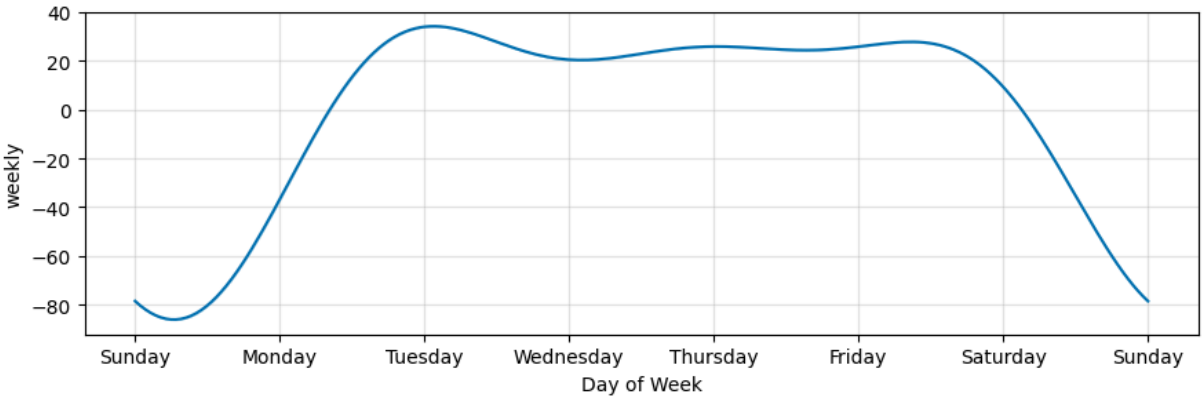
Step 5: Visualise the Forecast and Components

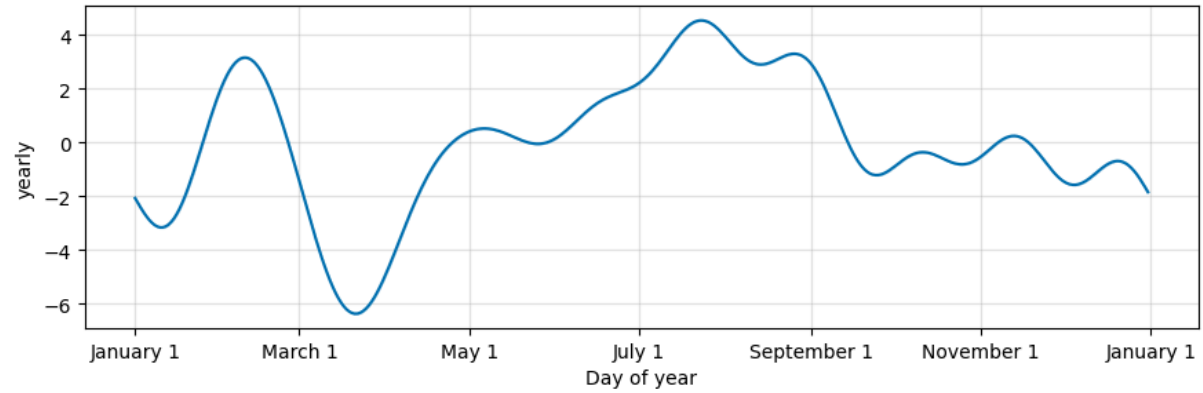
In [10]:
```
# Plot the forecast (black dots = actual, blue line = prediction, shaded = u
fig = model.plot(forecast)
plt.title('Prophet Forecast – Microsoft (MSFT) Closing Price')
plt.ylabel('Close Price (USD)')
plt.show()
```

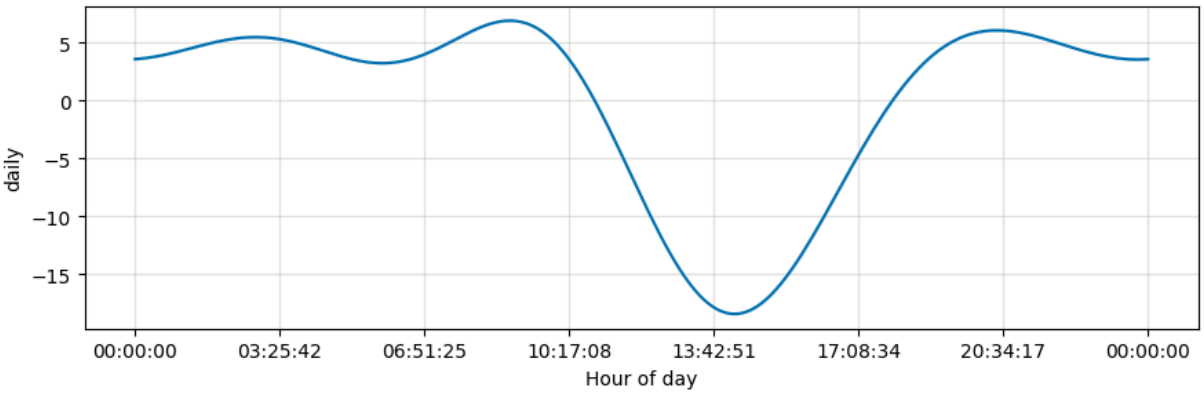


In [11]:
```
# Plot trend + seasonal components
fig2 = model.plot_components(forecast)
plt.show()
```









Step 6: Evaluate Forecast Accuracy

We compute the **Mean Absolute Error (MAE)** on the historical (in-sample) predictions.

In [12]:
```
from sklearn.metrics import mean_absolute_error

# Merge actual and predicted values
df_forecast = forecast[['ds', 'yhat']].set_index('ds')
df_actual = df.set_index('ds')
df_combined = pd.merge(df_actual, df_forecast, left_index=True, right_index=

# Calculate Mean Absolute Error
mae = mean_absolute_error(df_combined['y'], df_combined['yhat'])
print(f'Mean Absolute Error: ${mae:,.2f}')
```

```
Mean Absolute Error: $3.54
```