Aggregated version of Prophet model for forecasting "Tourism Arrivals in Philippines"

A single generalized model to forecast tourism arrivals in Philippines.

In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from prophet import Prophet
from prophet.diagnostics import cross_validation, performance_metrics
from prophet.plot import plot_cross_validation_metric
import warnings
import plotly.graph_objects as go
import itertools
import holidays
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, mean_absol
ute_percentage_error
warnings.filterwarnings('ignore')

pd.set_option('display.float_format', '{:.10f}'.format)
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("deep")
plt.rcParams['figure.figsize'] = [12, 6]
plt.rcParams['figure.dpi'] = 100
```

Load the CSV

In [10]:

```
df = pd.read_csv("https://raw.githubusercontent.com/aranes-rc/ph-tourist-arrivals-forecas
t/refs/heads/main/monthly-dataset.csv")
df['Date'] = pd.to_datetime(df['Date'])
df = df.loc[:, ~df.columns.str.contains('^Unnamed')].set_index('Date')
df = df[df.index < '2025-05-01'][['Arrivals']]
df.head()
```

Out[10]:

	Arrivals
Date	
2008-01-01	279338
2008-02-01	265827
2008-03-01	263862
2008-04-01	235895
2008-05-01	242822

Data prep

In [11]:

```
df.head()
```

Out[11]:

	Arrivals

Date	Arrivals
2008-01-01	279338
2008-02-01	265827
2008-03-01	263862
2008-04-01	235895
2008-05-01	242822

In [12]:

```
df.describe()
```

Out[12]:

	Arrivals
count	208.0000000000
mean	358473.9134615384
std	182635.7463234746
min	0.0000000000
25%	256378.7500000000
50%	367282.0000000000
75%	467943.2500000000
max	782132.0000000000

Plot raw data

In [13]:

```
plt.figure(figsize=(14, 7))
plt.plot(df.index, df['Arrivals'], marker='.', linestyle='-', color='#1f77b4')
plt.title('Monthly Tourist Arrivals in the Philippines (2008-2025)', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Number of Arrivals', fontsize=12)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Replace 0 values (like January 2022) with an estimate based on neighboring months

January 2022 data is missing so we'll use the average of December 2021 and February 2022 for January 2022.

NOTE: This is a temporary solution only

In [14]:

```
prophet_df = df.reset_index().rename(columns={'Date': 'ds', 'Arrivals': 'y'}) # Prepare for prophet training
```

In [15]:

```
if ((prophet_df['ds'].dt.year == 2022) & (prophet_df['ds'].dt.month == 1)).any():
    jan_2022_idx = prophet_df[(prophet_df['ds'].dt.year == 2022) & (prophet_df['ds'].dt.month == 1)].index[0]
    if prophet_df.loc[jan_2022_idx, 'y'] == 0:
        print("Handling January 2022 anomaly...")
        # Interpolate using the values from December 2021 and February 2022
        if jan_2022_idx > 0 and jan_2022_idx < len(prophet_df) - 1:
            dec_2021_value = prophet_df.loc[jan_2022_idx - 1, 'y']
            feb_2022_value = prophet_df.loc[jan_2022_idx + 1, 'y']
            prophet_df.loc[jan_2022_idx, 'y'] = (dec_2021_value + feb_2022_value) / 2
            print(f"January 2022 value interpolated from 0 to {prophet_df.loc[jan_2022_idx, 'y']}")
```

```
Handling January 2022 anomaly...
January 2022 value interpolated from 0 to 24501.5
```

Handling COVID shocks

> **To prevent large dips and spikes from being captured by the trend component, we can treat the days impacted by COVID-19 as holidays that will not repeat again in the future.**

Based on plotted data, these are likely the possible cases. These dates will be adjusted to first of the month since we have regular data gaps

---

Reference:

- https://facebook.github.io/prophet/docs/handling_shocks.html

In [16]:

```
prophet_df[(prophet_df['ds'].dt.year >= 2019) & (prophet_df['ds'].dt.year <= 2023)][['ds', 'y']]
```

Out[16]:

	ds	y
132	2019-01-01	718118.0000000000
133	2019-02-01	762437.0000000000
134	2019-03-01	709399.0000000000
135	2019-04-01	653336.0000000000
136	2019-05-01	612861.0000000000
137	2019-06-01	638440.0000000000
138	2019-07-01	712285.0000000000
139	2019-08-01	699933.0000000000
140	2019-09-01	604552.0000000000

	ds	y
141	2019-10-01	634786.0000000000
142	2019-11-01	679273.0000000000
143	2019-12-01	763057.0000000000
144	2020-01-01	782132.0000000000
145	2020-02-01	439852.0000000000
146	2020-03-01	127721.0000000000
147	2020-04-01	927.0000000000
148	2020-05-01	357.0000000000
149	2020-06-01	1186.0000000000
150	2020-07-01	3380.0000000000
151	2020-08-01	5364.0000000000
152	2020-09-01	6410.0000000000
153	2020-10-01	8304.0000000000
154	2020-11-01	9069.0000000000
155	2020-12-01	13753.0000000000
156	2021-01-01	6109.0000000000
157	2021-02-01	9005.0000000000
158	2021-03-01	10446.0000000000
159	2021-04-01	5098.0000000000
160	2021-05-01	9153.0000000000
161	2021-06-01	10281.0000000000
162	2021-07-01	11794.0000000000
163	2021-08-01	13132.0000000000
164	2021-09-01	13891.0000000000
165	2021-10-01	15656.0000000000
166	2021-11-01	18836.0000000000
167	2021-12-01	22697.0000000000
168	2022-01-01	24501.5000000000
169	2022-02-01	26306.0000000000
170	2022-03-01	69635.0000000000
171	2022-04-01	115514.0000000000
172	2022-05-01	124933.0000000000
173	2022-06-01	153497.0000000000
174	2022-07-01	232315.0000000000
175	2022-08-01	218048.0000000000
176	2022-09-01	207219.0000000000
177	2022-10-01	227669.0000000000
178	2022-11-01	275904.0000000000
179	2022-12-01	374345.0000000000
180	2023-01-01	425188.0000000000
181	2023-02-01	431695.0000000000
182	2023-03-01	436292.0000000000
183	2023-04-01	417320.0000000000
184	2023-05-01	353093.0000000000
185	2023-06-01	407210.0000000000

	ds	y
186	2023-07-01	467919.0000000000
187	2023-08-01	404029.0000000000
188	2023-09-01	356300.0000000000
189	2023-10-01	378123.0000000000
190	2023-11-01	411890.0000000000
191	2023-12-01	514416.0000000000

In [17]:

```
lockdowns = pd.DataFrame([
    {
        'holiday': 'covid_impact_1',  # First major drop
        'ds': '2020-02-01',  # Starting with February 2020 when arrivals began dropping
        'lower_window': 0,
        'ds_upper': '2020-12-01'  # Through the end of 2020
    },
    {
        'holiday': 'covid_impact_2',  # Continued low levels
        'ds': '2021-01-01',
        'lower_window': 0,
        'ds_upper': '2021-12-01'  # Through the end of 2021
    },
    {
        'holiday': 'covid_recovery',  # Recovery period
        'ds': '2022-01-01',
        'lower_window': 0,
        'ds_upper': '2022-07-01'  # First half of 2022 when recovery was still ongoing
    }
])

for t_col in ['ds', 'ds_upper']:
    lockdowns[t_col] = pd.to_datetime(lockdowns[t_col])

lockdowns['upper_window'] = (lockdowns['ds_upper'] - lockdowns['ds']).dt.days
lockdowns
```

Out[17]:

	holiday	ds	lower_window	ds_upper	upper_window
0	covid_impact_1	2020-02-01	0	2020-12-01	304
1	covid_impact_2	2021-01-01	0	2021-12-01	334
2	covid_recovery	2022-01-01	0	2022-07-01	181

Set future years to forecast

In [18]:

```
years_to_forecast = 6
months_to_forecast = 12 * years_to_forecast
```

Adjust regular holidays

`holidays` package in pip is so helpful here to obtain list of country holidays from 2008 - 2025. We'll use this to coerce the dates of the holidays to the first of the month since our dataset is monthly and has days/regular gaps.

This solution came from: https://facebook.github.io/prophet/docs/non-daily_data.html#holidays-with-aggregated-data

In [19]:

```
data_years = list(range(df.index.year.min(), df.index.year.max() + years_to_forecast + 1
))

country_code = 'PH'
country_holidays = holidays.country_holidays(country_code, years=data_years)

holiday_df = pd.DataFrame(
    [(name, date) for date, name in country_holidays.items()],
    columns=['holiday', 'ds']
)

lower_window = 0
upper_window = 0

holiday_df['lower_window'] = lower_window
holiday_df['upper_window'] = upper_window
holiday_df = holiday_df.sort_values(by='ds').reset_index(drop=True)
holiday_df
```

Out[19]:

	holiday	ds	lower_window	upper_window
0	New Year's Day	2008-01-01	0	0
1	Maundy Thursday	2008-03-20	0	0
2	Good Friday	2008-03-21	0	0
3	Day of Valor	2008-04-07	0	0
4	Labor Day	2008-05-01	0	0
...	...	...	...	...
444	Bonifacio Day	2031-11-30	0	0
445	Immaculate Conception	2031-12-08	0	0
446	Christmas Day	2031-12-25	0	0
447	Rizal Day	2031-12-30	0	0
448	New Year's Eve	2031-12-31	0	0

449 rows × 4 columns

In [20]:

```
holiday_adjusted = holiday_df.copy()
```

In [21]:

```
holiday_adjusted['ds'] = pd.to_datetime(holiday_adjusted['ds'])
holiday_adjusted['ds'] = holiday_adjusted['ds'].dt.strftime('%Y-%m-01')
holiday_adjusted['ds'] = pd.to_datetime(holiday_adjusted['ds'])
holiday_adjusted = holiday_adjusted.drop_duplicates(subset=['holiday', 'ds'])
holiday_adjusted
```

Out[21]:

	holiday	ds	lower_window	upper_window
0	New Year's Day	2008-01-01	0	0
1	Maundy Thursday	2008-03-01	0	0
2	Good Friday	2008-03-01	0	0
3	Day of Valor	2008-04-01	0	0
4	Labor Day	2008-05-01	0	0
...	...	...	...	...
444	Bonifacio Day	2031-11-01	0	0

	holiday	ds	lower_window	upper_window
445	Immaculate Conception	2031-12-01	0	0
446	Christmas Day	2031-12-01	0	0
447	Rizal Day	2031-12-01	0	0
448	New Year's Eve	2031-12-01	0	0

447 rows × 4 columns

Building Prophet (v1)

Model cross-validation results (normalized):

mae	mse	rmse	r2	mape
0.0139	0.000382	0.01956	0.9929	4.5%

For training:

- We'll be using `multiplicative` seasonality here since that fits our problem. Tho this might cause a problem for calculating MAPE
- We need to reconsider `monthly` seasonality period of `30.5`
- PH Holidays are added already
- COVID19 phase are treated as holidays also

Performance metrics (R^2 score) of model if it implements the ff.:

Configuration	R^2 Score	Improvement over Base
Base Model (No Binary Month Regressors & No Pre/Post COVID Seasonality)	0.915	-
With Binary Month Regressors Only	0.914	-0.001
With Pre/Post COVID Seasonality Only	0.902	-0.013
With Both Features	**0.935**	+0.020

The synergistic effect of both features together yields better results than either feature individually.

For forecasting:

- We should use `MS` since Prophet will only be able to see the first day of a month. MS stands for Month Start

---

References:

- https://facebook.github.io/prophet/docs/non-daily_data.html#monthly-data
- https://facebook.github.io/prophet/docs/seasonality,_holiday_effects,_and_regressors.html#fourier-order-for-seasonalities
- https://facebook.github.io/prophet/docs/handling_shocks.html#changes-in-seasonality-between-pre--and-post-covid
- https://facebook.github.io/prophet/docs/seasonality,_holiday_effects,_and_regressors.html#built-in-country-holidays

Create and fit the Prophet model

Model properties:

- We use multiplicative since our plotted raw data shows a clear sign of it
- We concat both lockdown and regular holidays
- We use extra binary regressors (one-hot encoded months)
- We add custom monthly seasonality pre- and has- COVID19.
  - Pre-covid starts at the beginning of the dataset to February 2020
  - Has-covid phase starts at the March 2020 to July 2023

- **There is no post-covid seasonality added since adding one leads to bad forecast results and pre-covid and post-covid has very similar trends**

In [28]:

```
model = Prophet(
    yearly_seasonality=False,
    changepoint_range=0.95,
    changepoint_prior_scale=0.01,
    seasonality_mode='multiplicative',
    holidays=pd.concat([lockdowns, holiday_adjusted])
)

months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', '
dec']

for i, month in enumerate(months, 1):
    prophet_df[f'is_{month}'] = (prophet_df['ds'].dt.month == i).astype(int)

for month in months:
    model.add_regressor(f'is_{month}')

covid_outbreak_date = '2020-02-01'
covid_end_recovery_date = '2023-07-01'

prophet_df['pre_covid'] = pd.to_datetime(prophet_df['ds']) < pd.to_datetime(covid_outbre
ak_date)
prophet_df['has_covid'] = (
    (pd.to_datetime(prophet_df['ds']) > pd.to_datetime(covid_outbreak_date)) &
    (pd.to_datetime(prophet_df['ds']) < pd.to_datetime(covid_end_recovery_date))
)

monthly_period = 365.5
fourier_order = 18
model.add_seasonality(name='yearly_pre_covid', period=monthly_period, fourier_order=fouri
er_order, condition_name='pre_covid')
model.add_seasonality(name='yearly_has_covid', period=monthly_period, fourier_order=fouri
er_order, condition_name='has_covid')

model.fit(prophet_df)
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to ov
erride this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to over
ride this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpzuwp3nvu/j06lpdhx.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpzuwp3nvu/8wejlaet.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_mode
l/prophet_model.bin', 'random', 'seed=74862', 'data', 'file=/tmp/tmpzuwp3nvu/j06lpdhx.jso
n', 'init=/tmp/tmpzuwp3nvu/8wejlaet.json', 'output', 'file=/tmp/tmpzuwp3nvu/prophet_model
v672lusr/prophet_model-20250520072724.csv', 'method=optimize', 'algorithm=lbfgs', 'iter=1
0000']
07:27:24 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
07:27:24 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

Out[28]:

```
<prophet.forecaster.Prophet at 0x7c2f009cfb90>
```

Forecast

Since the dates on our rows in the dataset starts at the first day of the month, we have to use `MS (month start)` **for forecasting frequency.**

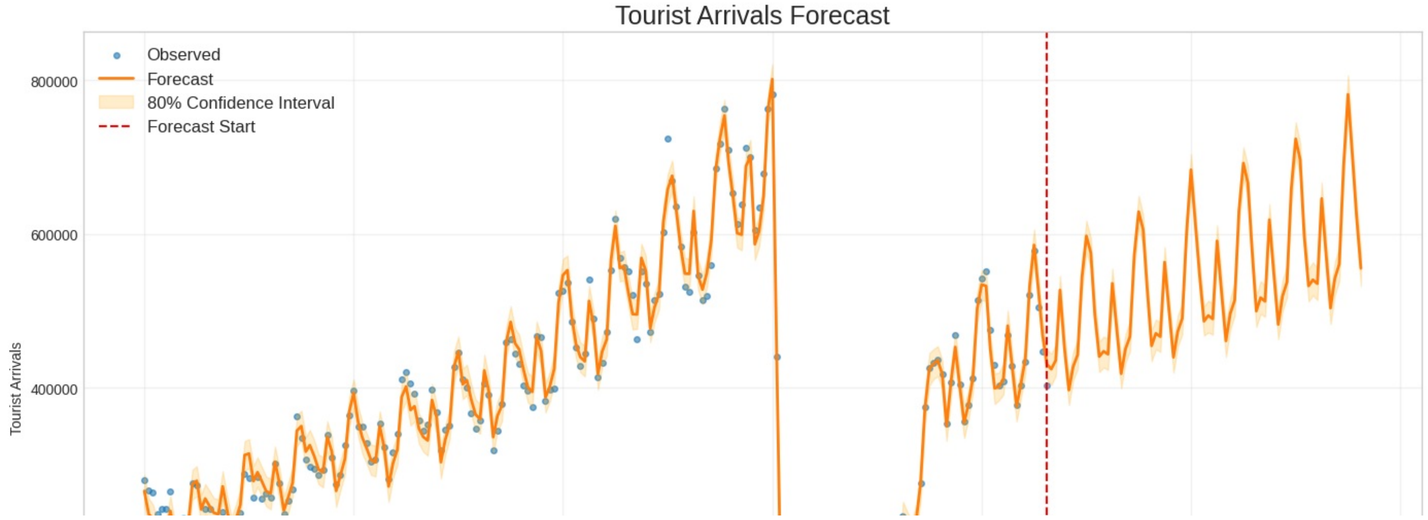
In [29]:

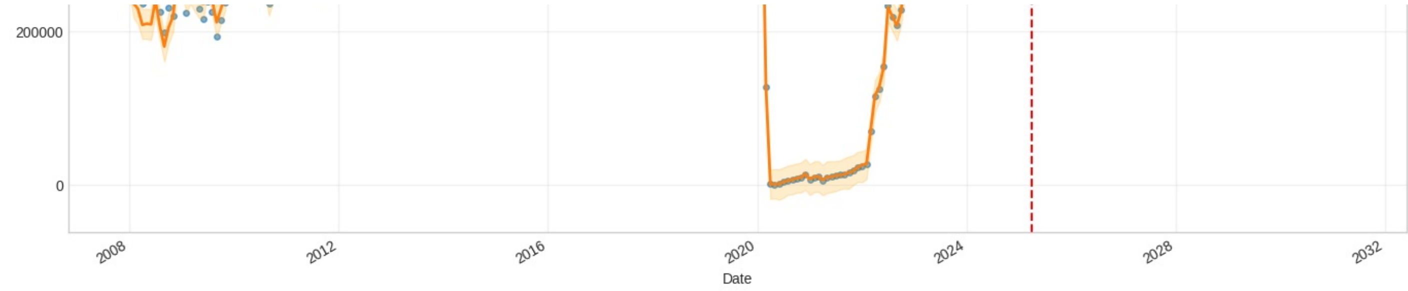
```
future = model.make_future_dataframe(periods=months_to_forecast, freq='MS')

future['pre_covid'] = pd.to_datetime(future['ds']) < pd.to_datetime(covid_outbreak_date)
future['has_covid'] = (
    (pd.to_datetime(future['ds']) > pd.to_datetime(covid_outbreak_date)) &
    (pd.to_datetime(future['ds']) < pd.to_datetime(covid_end_recovery_date))
)

for i, month in enumerate(months, 1):
    future[f'is_{month}'] = (future['ds'].dt.month == i).astype(int)

forecast = model.predict(future)
```

Plot the forecast

In [30]:

```
fig, ax = plt.subplots(figsize=(14, 8))

ax.scatter(prophet_df['ds'], prophet_df['y'], color='#1f77b4', s=16, alpha=0.6, label='O
bserved')

ax.plot(forecast['ds'], forecast['yhat'], color='#ff7f0e', linewidth=2, label='Forecast'
)

ax.fill_between(
    forecast['ds'],
    forecast['yhat_lower'],
    forecast['yhat_upper'],
    color='orange',
    alpha=0.2,
    label='80% Confidence Interval'
)

forecast_start = prophet_df['ds'].iloc[-1]
ymin = min(forecast['yhat_lower'].min(), prophet_df['y'].min())
ymax = max(forecast['yhat_upper'].max(), prophet_df['y'].max())
ax.axvline(forecast_start, color='red', linestyle='--', label='Forecast Start')

ax.set_title('Tourist Arrivals Forecast', fontsize=18)
ax.set_xlabel('Date')
ax.set_ylabel('Tourist Arrivals')

ax.grid(True, which='both', color='lightgray', linewidth=1, alpha=0.3)

fig.autofmt_xdate()

# Legend
ax.legend(fontsize=12)

plt.tight_layout()
plt.show()
```

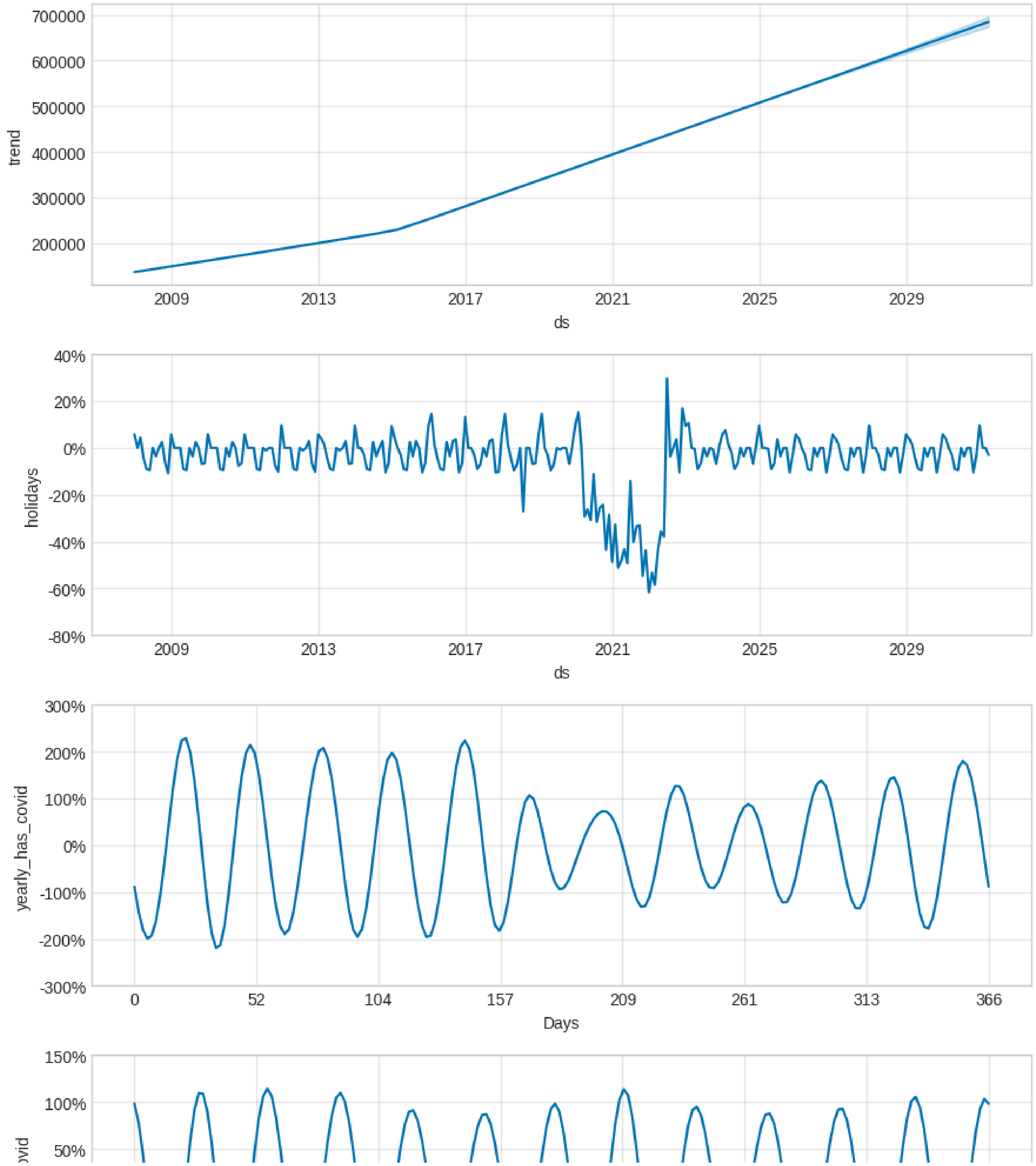


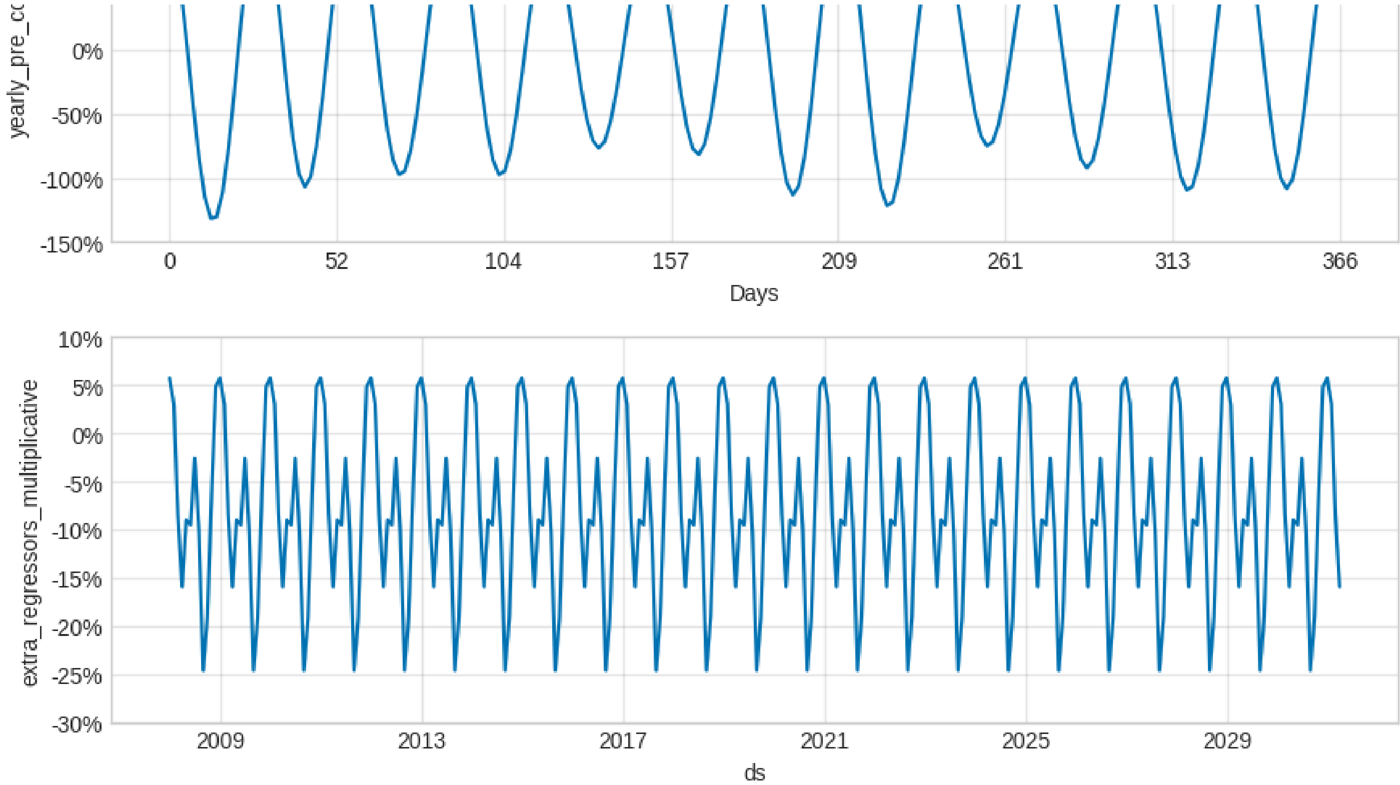


Display the forecast components

In [31]:

```
fig_comp = model.plot_components(forecast)
plt.tight_layout()
plt.show()
```





Model results

This is normalized

In [32]:

```
def regression_report(y_true, y_pred, normalize=True):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    # Metrics
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    mape = mean_absolute_percentage_error(y_true, y_pred) * 100

    # Normalization factor
    norm_factor = np.ptp(y_true) if normalize else 1.0  # ptp = max - min

    report = pd.DataFrame({
        'Metric': ['MAE', 'MSE', 'RMSE', 'MAPE (%)'],
        'Value': [mae / norm_factor,
                  mse / (norm_factor ** 2),
                  rmse / norm_factor,
                  mape]
    })

    return report
```

In [33]:

```
regression_report(prophet_df['y'], forecast['yhat'][:-months_to_forecast])
```

Out[33]:

	Metric	Value
0	MAE	0.0139142179
1	MSE	0.0003827301
2	RMSE	0.0195634888
3	MAPE (%)	4.5584188942

(%)

Metric	Value