# Sunspot Prediction with Time Series and Deep Learning

## Main objective

Our objective is to preidct SunSpots.

Sunspots are temporary phenomena on the Sun's photosphere that appear as spots darker than the surrounding areas.

They are regions of reduced surface temperature caused by concentrations of magnetic field flux that inhibit convection.

## About Data Set

The Dataset is downloaded from Kaggle. REf - https://www.kaggle.com/datasets/robervalt/sunspots

Dataset contains contains sunspots on a monthly basis from 1749 until 2018.

Sunspots usually appear in pairs of opposite magnetic polarity.

Their number varies according to the approximately 11-year solar cycle.

Column Description:

- Index - Index column of the time series. We will use this for our modeling
- Date - Date of the observation(from 1749 to 2018). We will use this for our visualization
- Monthly Mean Total Sunspot Number - Monthly mean total sunspot for the date

In [ ]:
```
!pip install -q fbprophet
```

In [ ]:
```
# import all required libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from datetime import datetime
```

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from fbprophet import Prophet
from statsmodels.tsa.arima_model import ARIMA
import seaborn as sns
import sys, os
import warnings
warnings.simplefilter(action='ignore')
print(f'Tensorflow version: {tf.__version__}')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use t
he functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
Tensorflow version: 2.8.2
```

In [ ]:

```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/Sunspots.csv \
    -O /tmp/sunspots.csv
```

```
--2022-06-05 06:27:22--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/Sunspots.csv
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.2.112, 172.217.1.208, 172.217.15.112, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.2.112|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 70827 (69K) [application/octet-stream]
Saving to: ‘/tmp/sunspots.csv’

/tmp/sunspots.csv   100%[===================>]  69.17K  --.-KB/s    in 0.001s

2022-06-05 06:27:22 (119 MB/s) - ‘/tmp/sunspots.csv’ saved [70827/70827]
```

In [ ]:

```
# read Sunspots.csv into a pandas dataframe with date parse
df = pd.read_csv('/tmp/sunspots.csv',
                    parse_dates=['Date'])
df.tail()
```

Out[ ]:

| | Unnamed: 0 | Date | Monthly Mean Total Sunspot Number |
|---|---|---|---|
| **3230** | 3230 | 2018-03-31 | 2.5 |
| **3231** | 3231 | 2018-04-30 | 8.9 |
| **3232** | 3232 | 2018-05-31 | 13.2 |
| **3233** | 3233 | 2018-06-30 | 15.9 |
| **3234** | 3234 | 2018-07-31 | 1.6 |

In [ ]:
```
# rename the Unnamed: 0 column to timestep
df.rename(columns={'Unnamed: 0': 'time_step',
                   'Monthly Mean Total Sunspot Number': 'sunspots'}, inplace=True)
```

# Exploratory Data Analysis

In [ ]:
```
# Lets look at the data
df.info()
```

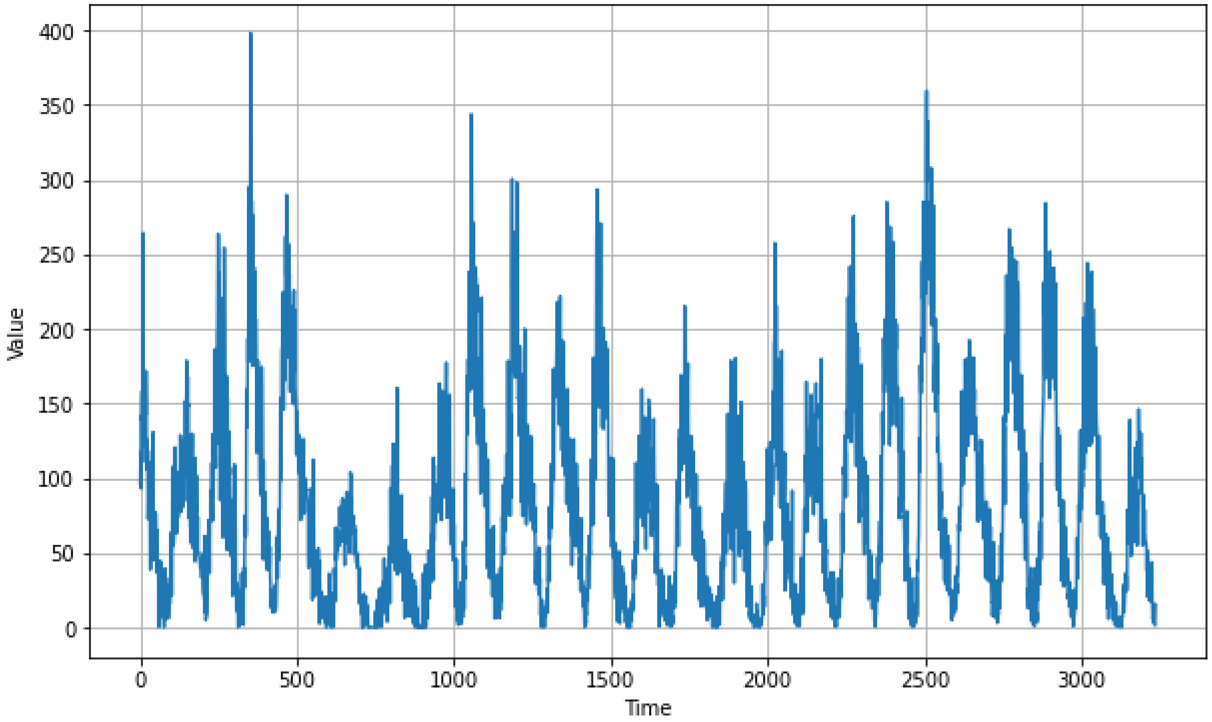
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3235 entries, 0 to 3234
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   time_step  3235 non-null   int64
 1   Date       3235 non-null   datetime64[ns]
 2   sunspots   3235 non-null   float64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 75.9 KB
```

In [ ]:
```
# Check any missing values
df.isnull().sum()
```

Out[ ]:
```
time_step    0
Date         0
sunspots     0
dtype: int64
```

In [ ]:
```
# Plot date vs sunspots
import matplotlib.pyplot as plt
```

```
def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(True)
series = np.array(df.sunspots)
time = np.array(df.time_step)
plt.figure(figsize=(10, 6))
plot_series(time, series)
```

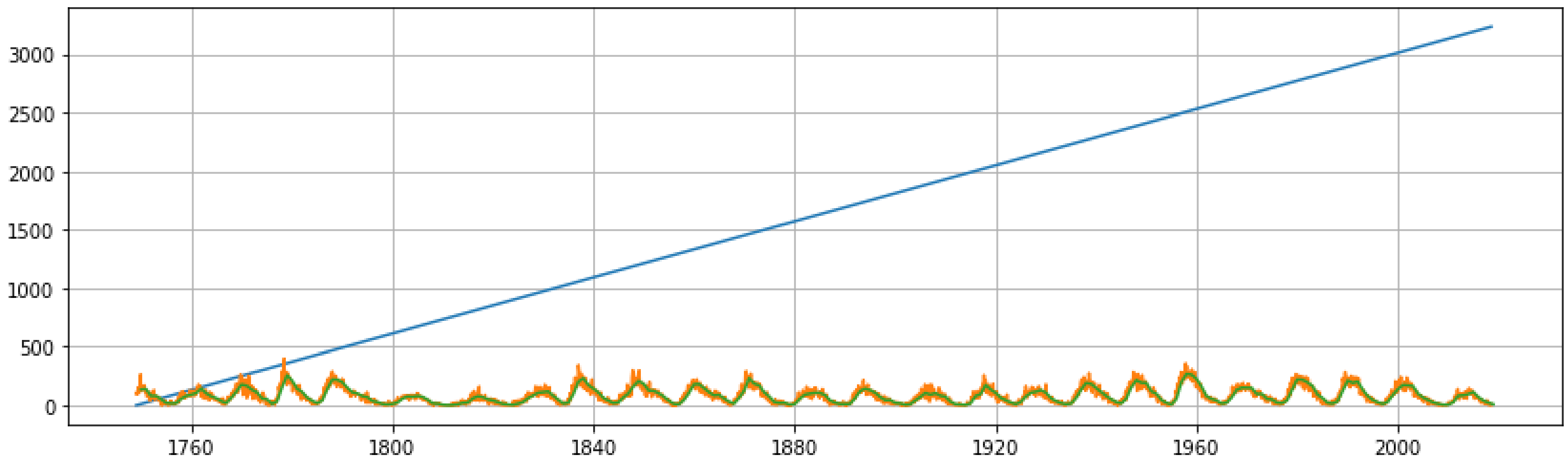



## Time Series Specific EDAs

We will check if the data is stationary or not.

In [ ]:
```
# resample to annual and plot each
plt.rcParams['figure.figsize'] = [14, 4]
df_full_timeseries = df.set_index('Date')
annual_sunspot = df_full_timeseries.resample('A').mean()
```

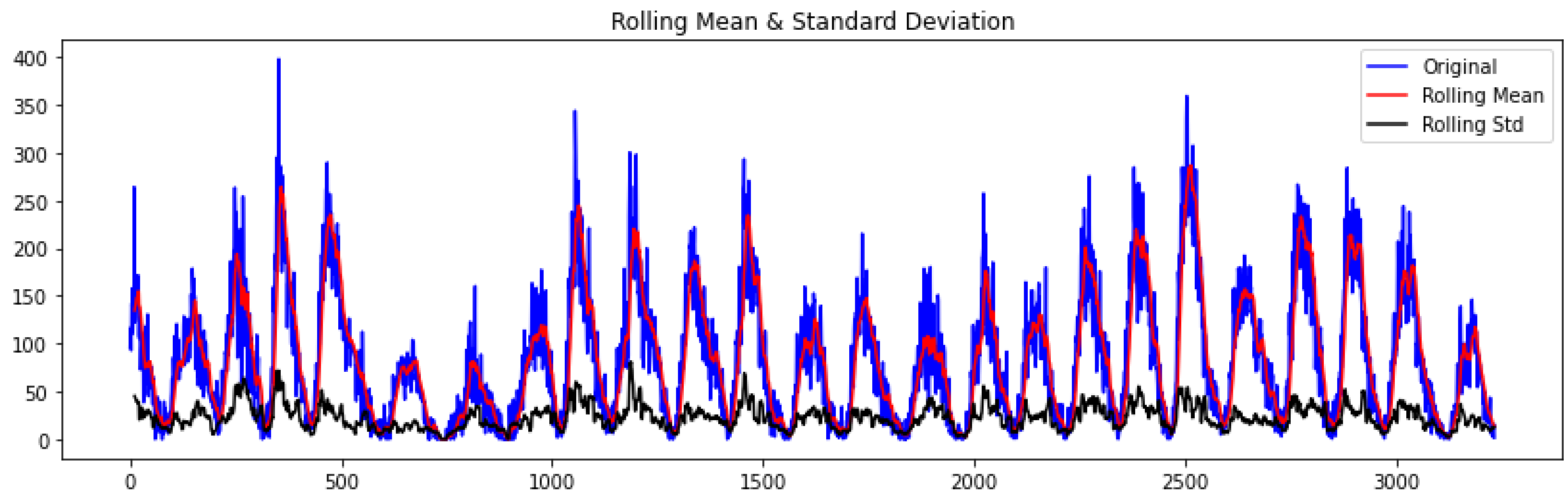
```
plt.plot(df_full_timeseries)
plt.plot(annual_sunspot.sunspots)
plt.grid(b=True);
```




In [ ]:

```
# check if the data is stationary
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):
    #Determing rolling statistics
    df_timeseries = pd.DataFrame(timeseries)
    rolmean = df_timeseries.rolling(window=12).mean()
    rolstd = df_timeseries.rolling(window=12).std()
    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

In [ ]:
```
test_stationarity(series)
```




```
Results of Dickey-Fuller Test:
Test Statistic                -1.049256e+01
p-value                        1.137033e-18
#Lags Used                     2.800000e+01
Number of Observations Used    3.206000e+03
Critical Value (1%)           -3.432391e+00
Critical Value (5%)           -2.862442e+00
Critical Value (10%)          -2.567250e+00
dtype: float64
```

## Our Observations

- ADF value is negative, so we can assmume the data is stationary.
- PValue is less than 0.05, so we can assmume the data is stationary.
- Critical value - Here we see a test statistic of roughly -2.56 and lower is sufficient to reject the null using a significance level of 5%.

# Modeling

## Modeling Objective

We will try with both Time series( FB Prophet) modeling and Deep Learning.

In [ ]:
```
# FB Prophet
m = Prophet()
# Format the dataframe for Prophet with ds and y
df_Prophet = pd.DataFrame()
df_Prophet['ds'] = df['Date']
df_Prophet['y'] = df['sunspots']
m.fit(df_Prophet)
```

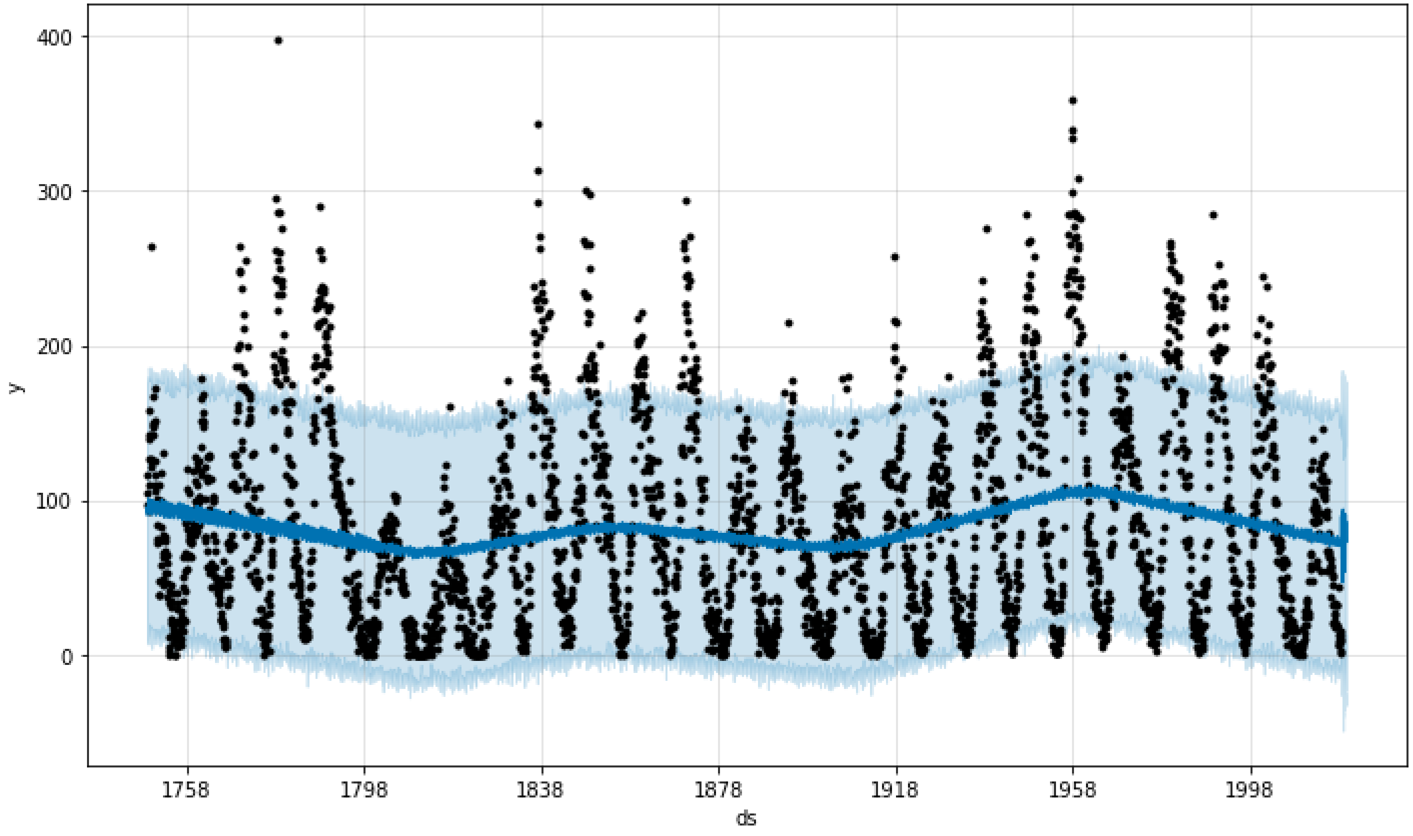
```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

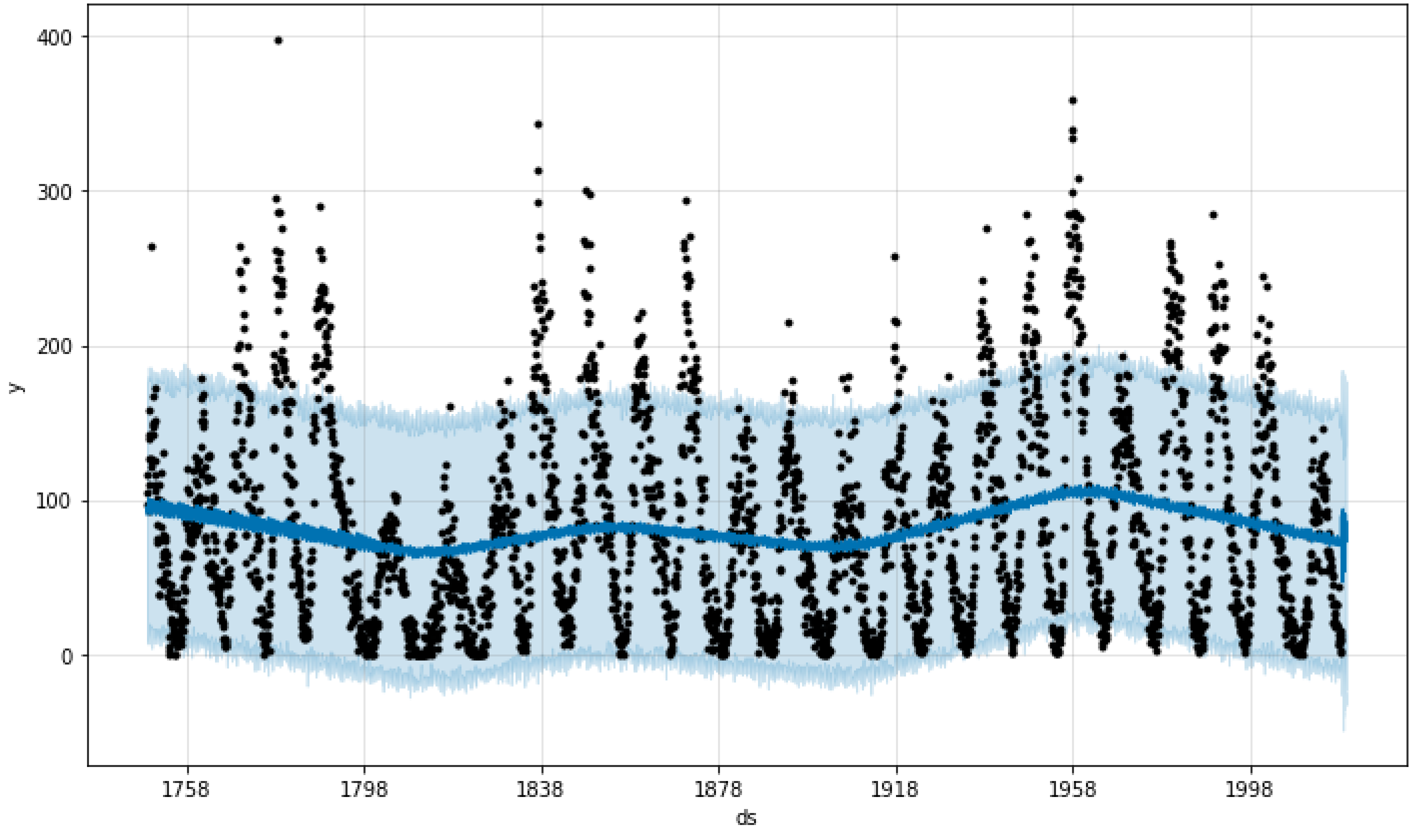
Out[ ]:
```
<fbprophet.forecaster.Prophet at 0x7fccfdf4ef90>
```

In [ ]:
```
future = m.make_future_dataframe(periods=365)
forecast = m.predict(future)
print(forecast.columns)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
m.plot(forecast)
```

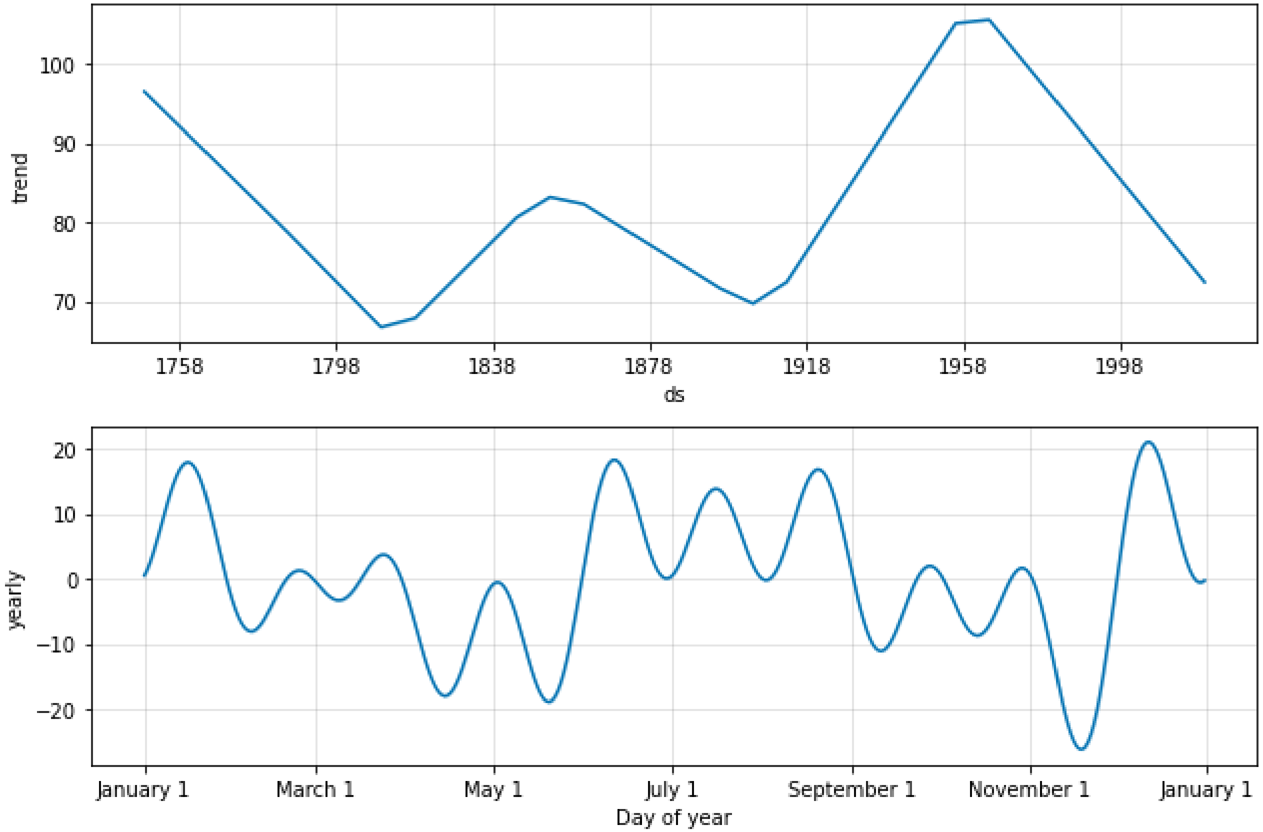
```
Index(['ds', 'trend', 'yhat_lower', 'yhat_upper', 'trend_lower', 'trend_upper',
       'additive_terms', 'additive_terms_lower', 'additive_terms_upper',
       'yearly', 'yearly_lower', 'yearly_upper', 'multiplicative_terms',
       'multiplicative_terms_lower', 'multiplicative_terms_upper', 'yhat'],
      dtype='object')
```

Out[ ]:






In [ ]:
```
m.plot_components(forecast);
```




# Deep Learning Model

## Data set preparation for Deep Learning models

In [ ]:

```
# Lets use Tensorflow dataset feature
split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

```
# Method to create window dataset for all of our deep learning model
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[1:]))
    return ds.batch(batch_size).prefetch(1)

# healper method to predict
def model_forecast(model, series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast

#plot results
def plot_prediction(input_model):
  model_forecast_result = model_forecast(input_model, series[..., np.newaxis], window_size)
  if(model_forecast_result.ndim == 3):
    model_forecast_result = model_forecast_result[split_time - window_size:-1, -1, 0]
  else:
    model_forecast_result = model_forecast_result[split_time-window_size:-1,-1]
  plt.figure(figsize=(10, 6))
  plot_series(time_valid, x_valid)
  plot_series(time_valid, model_forecast_result)
  model_mae = tf.keras.metrics.mean_absolute_error(x_valid, model_forecast_result).numpy()
  print (f'MAE is {model_mae}')
  return model_mae
```

In [ ]:
```
train_set = windowed_dataset(x_train, window_size=60,
                             batch_size=100, shuffle_buffer=shuffle_buffer_size)
```

In [ ]:
```
# we will keep all reusable methods here for deep learning
lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
tf.random.set_seed(51)
np.random.seed(51)
window_size = 64
```

```
batch_size = 256
no_epoch = 100
model_results = {}
```

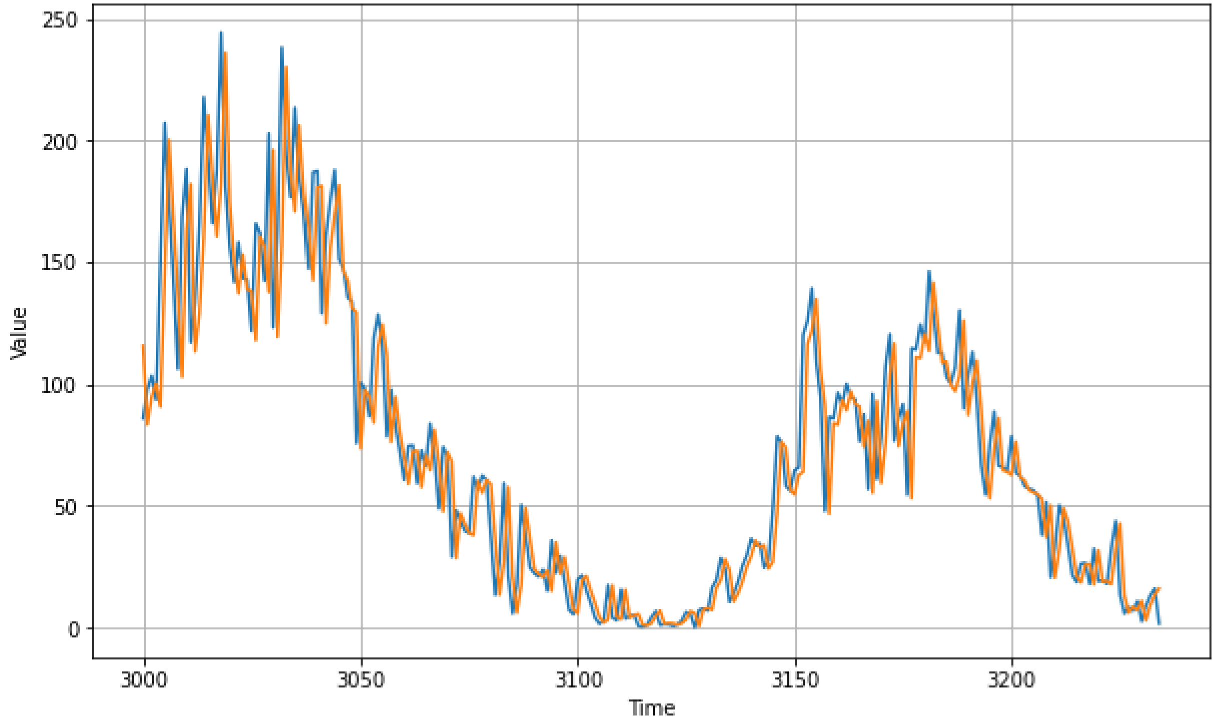
## Simple Deep Neural Network

In [ ]:
```
tf.keras.backend.clear_session()
dnn_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(20, input_shape=[None, 1], activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])

dnn_model.compile(loss=tf.keras.losses.Huber(),
                  optimizer=optimizer, metrics=["mae"])
dnn_model.fit(train_set,epochs=no_epoch,callbacks=[lr_schedule],verbose=0)

results = plot_prediction(dnn_model)
model_results['SimpleDNN'] = results
#model_results.append({'SimpleDNN': results})
```

```
MAE is 15.785479545593262
```



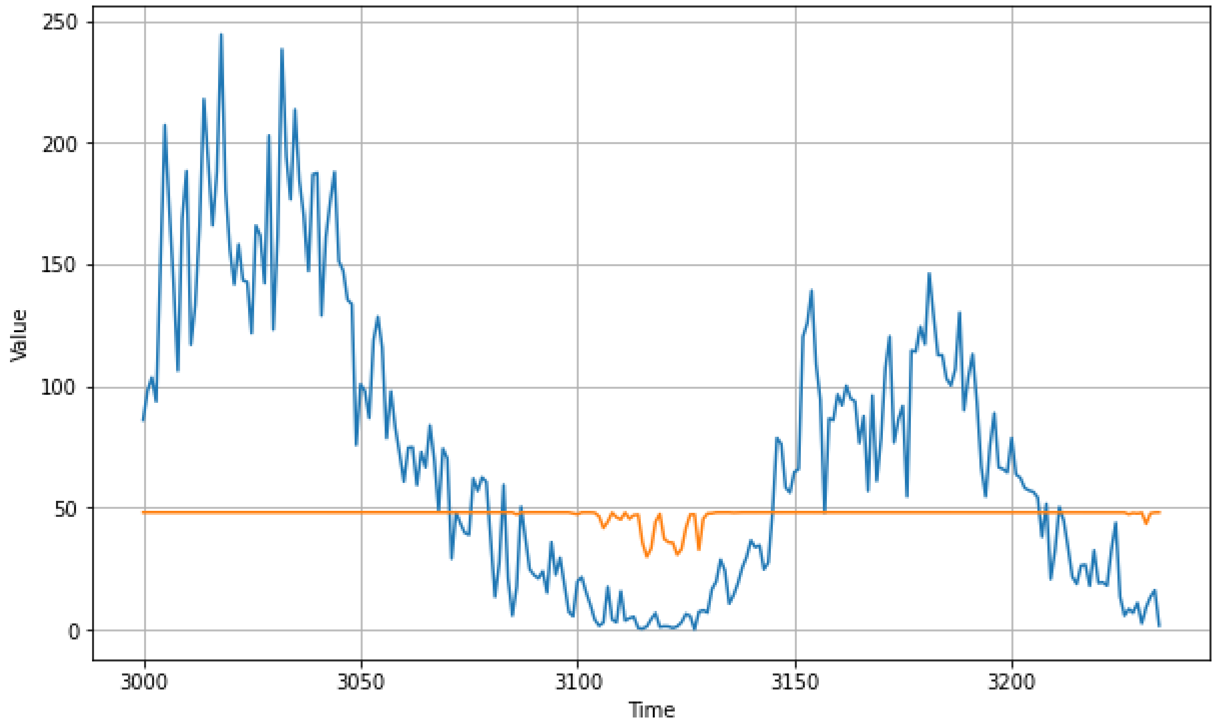

## RNN Model

In [ ]:
```
tf.keras.backend.clear_session()
rnn_model = tf.keras.models.Sequential([
  #tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
  #                     input_shape=[None, 1]),
  tf.keras.layers.SimpleRNN(40,input_shape=[None, 1], return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
])

rnn_model.compile(loss=tf.keras.losses.Huber(),
                  optimizer=optimizer, metrics=["mae"])
rnn_model.fit(train_set,epochs=no_epoch,callbacks=[lr_schedule],verbose=0)

results = plot_prediction(rnn_model)
model_results['SimpleRNN'] = results
```

```
MAE is 48.94666290283203
```

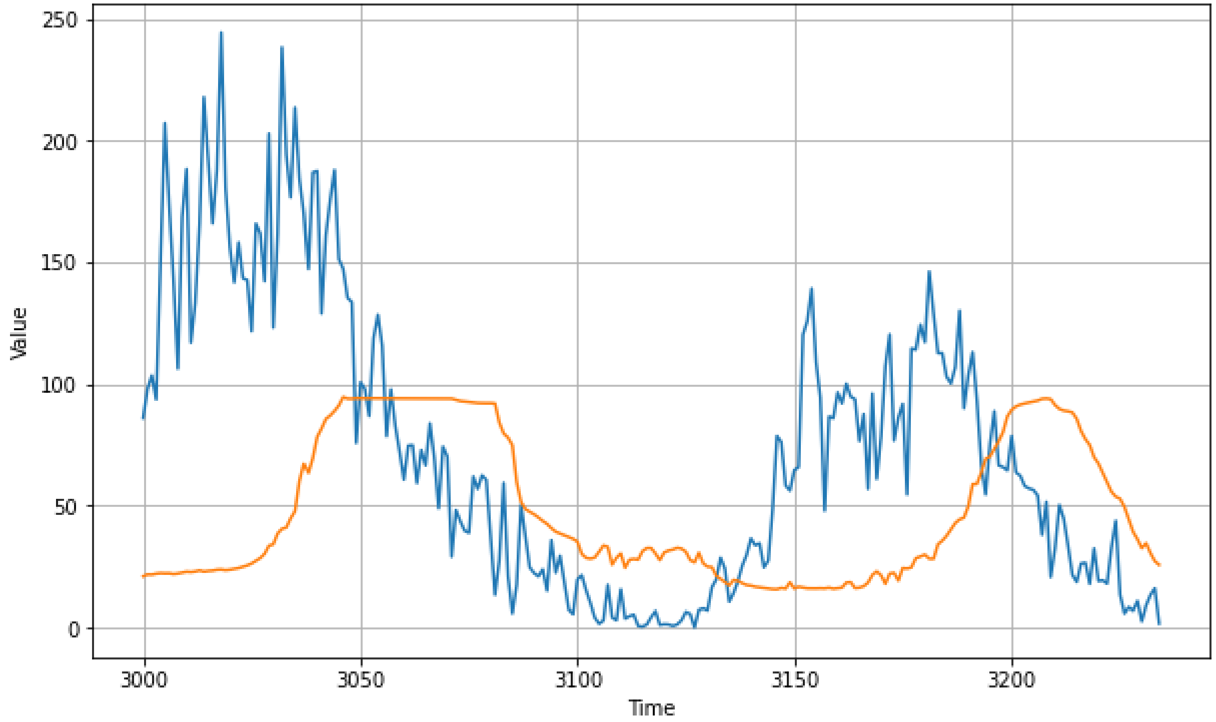



In [ ]:
```
tf.keras.backend.clear_session()
lstm_model =tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100)
])

lstm_model.compile(loss=tf.keras.losses.Huber(),
                  optimizer=optimizer, metrics=["mae"])
lstm_model.fit(train_set,epochs=no_epoch,callbacks=[lr_schedule],verbose=0)

results = plot_prediction(lstm_model)
model_results['BiDirectional LSTM'] = results
```

```
MAE is 55.21773910522461
```

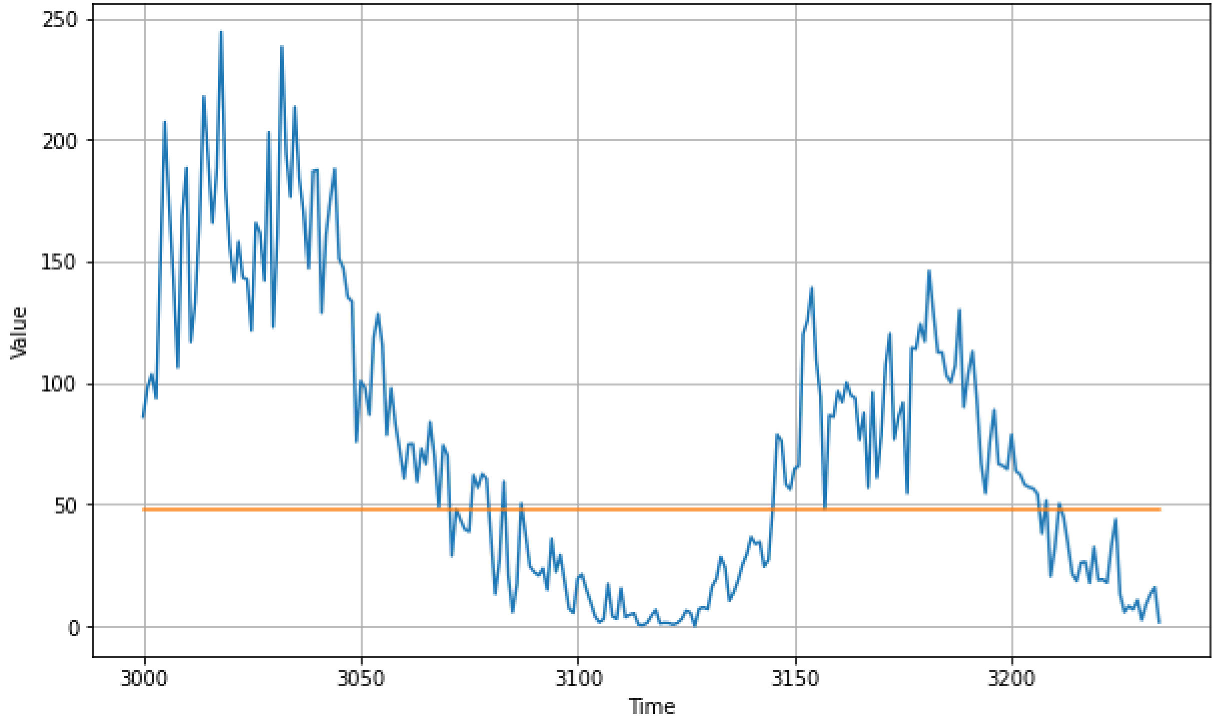



## LSTM with Conv1D

In [ ]:
```
tf.keras.backend.clear_session()
conv1d_model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])
conv1d_model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
```

```
history = conv1d_model.fit(train_set, epochs=no_epoch,
                           callbacks=[lr_schedule], verbose=0)
results = plot_prediction(conv1d_model)
model_results['LSTM with Conv1D'] = results
```

MAE is 49.659053802490234




In [ ]:

```
# Convert restults to Datafram
df_result = pd.DataFrame.from_dict(model_results, orient='index', columns=['MAE'])
df_result
```

Out[ ]:

| | MAE |
|---|---|
| **SimpleDNN** | 15.785480 |
| **SimpleRNN** | 48.946663 |
| **BiDirectional LSTM** | 55.217739 |
| **LSTM with Conv1D** | 49.659054 |

# Key Findings

- Based on our analysis, SimpleDNN works better for this dataset as described in above table
- If we train the Deep Learning model for few more epoch it will converge

# Possible Flaws and future enhancement

- At some point the model seems overfitting the dataset.
- Its advisable to add dropout in the network
- Source for few more datasources to improve the accuracy.