Econ 412: Final Project Source Code

Modeling and Forecasting Sunspots Over Time

Dylan Schmerer, Rhea Sethi, Talia Lieberman

Introduction

For this project, we chose to work with Sunspots data, which are relatively colder areas on the sun. They follow a cyclical pattern known as a solar cycle which lasts approximately 11 years. Modelling and predicting sunspots can be useful because they can affect agricultural yields and communication systems.

In [1]:
```
# Import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from fredapi import Fred
import statsmodels.formula.api as smf
import seaborn as sns
```

In [3]:
```
# Importing the data
df = pd.read_csv('Sunspots.csv', index_col = 0)
```

Data Cleaning

In [4]:
```
df['Date'] = pd.to_datetime(df['Date'])
df = df.rename(columns={"Monthly Mean Total Sunspot Number": "Sunspots"})
```

In [5]:
```
# Subset data to observe last 60 years (12 months * 60 (+1 to start in January))
df_sub = df.tail(721)
df_sub
```

Out[5]:

	Date	Sunspots
2544	1961-01-31	82.1
2545	1961-02-28	65.4
2546	1961-03-31	75.2
2547	1961-04-30	86.9
2548	1961-05-31	72.3
...	...	...
3260	2020-09-30	0.6
3261	2020-10-31	14.4
3262	2020-11-30	34.0
3263	2020-12-31	21.8
3264	2021-01-31	10.4

721 rows × 2 columns

In [6]:
```
#Add column for months
df_sub['Month']=df_sub.Date.dt.month
```

```
/var/folders/_9/0pcw32b15m70tyjpxdx6wc_m0000gn/T/ipykernel_50197/2835959519.py:2: SettingWithCopyWarning
:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
tml#returning-a-view-versus-a-copy
  df_sub['Month']=df_sub.Date.dt.month
```

In [8]:
```
#Add column for year of the decade
df_sub['Year'] =[int(str(i)[3]) for i in (df_sub.Date.dt.year)]
df_sub['Year'].replace(0,10,inplace=True)
df_sub.head()
```

```
/var/folders/_9/0pcw32b15m70tyjpxdx6wc_m0000gn/T/ipykernel_50197/3895228640.py:2: SettingWithCopyWarning
:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
tml#returning-a-view-versus-a-copy
  df_sub['Year'] =[int(str(i)[3]) for i in (df_sub.Date.dt.year)]
/var/folders/_9/0pcw32b15m70tyjpxdx6wc_m0000gn/T/ipykernel_50197/3895228640.py:3: SettingWithCopyWarning
:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
tml#returning-a-view-versus-a-copy
  df_sub['Year'].replace(0,10,inplace=True)
```

Out[8]:

	Date	Sunspots	Month	Year
2544	1961-01-31	82.1	1	1
2545	1961-02-28	65.4	2	1
2546	1961-03-31	75.2	3	1
2547	1961-04-30	86.9	4	1
2548	1961-05-31	72.3	5	1

In [9]:
```
sunspot = df_sub.copy()
```

In [10]:
```
#set index
sunspot = sunspot.set_index('Date')
```

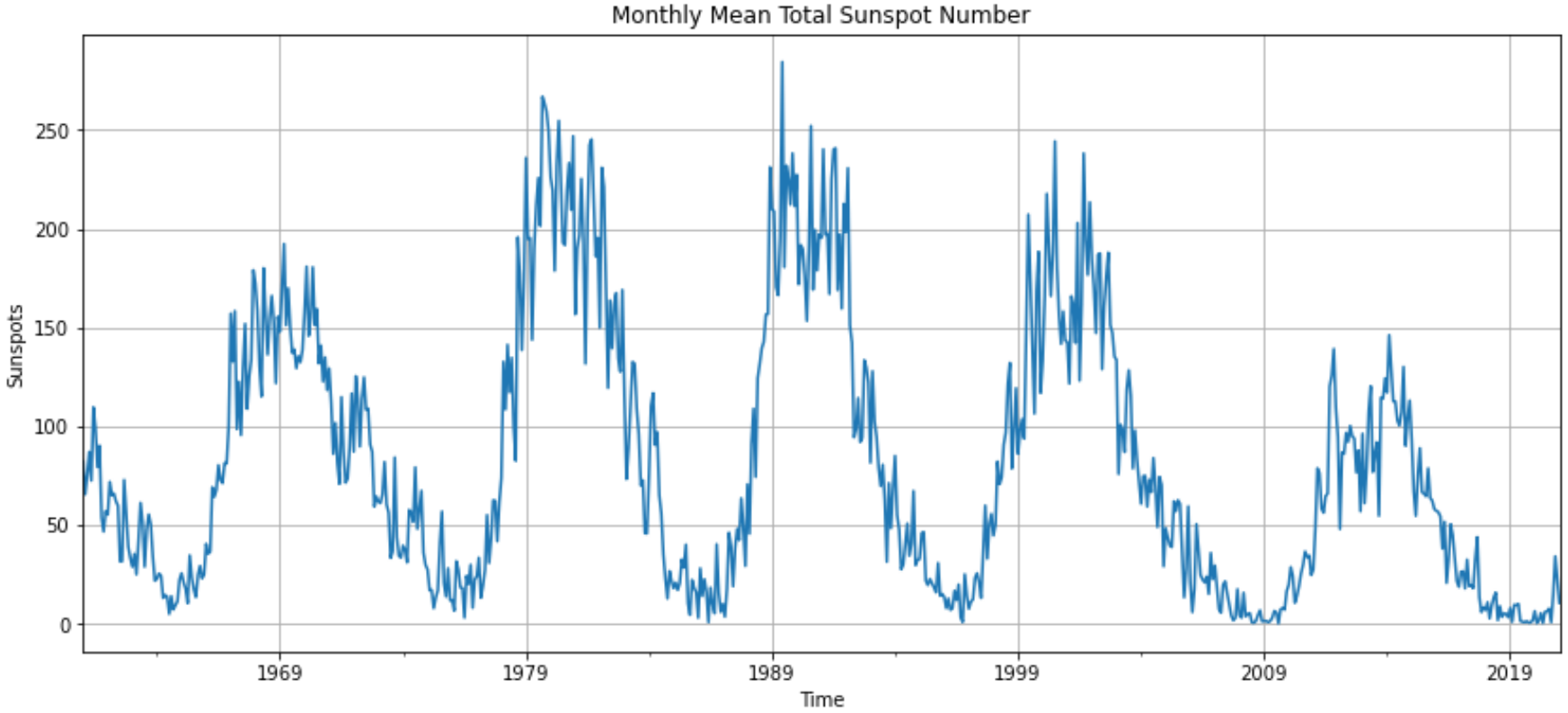
In [11]:
```
# Apply differencing
sunspot['Diff'] = sunspot['Sunspots'].diff()
```

Exploratory Analysis

Time series plots

In [12]:
```
plt.figure(figsize = (14,6))

sunspot['Sunspots'].plot(grid = True)
plt.title("Monthly Mean Total Sunspot Number")
plt.xlabel("Time")
plt.ylabel('Sunspots')
```

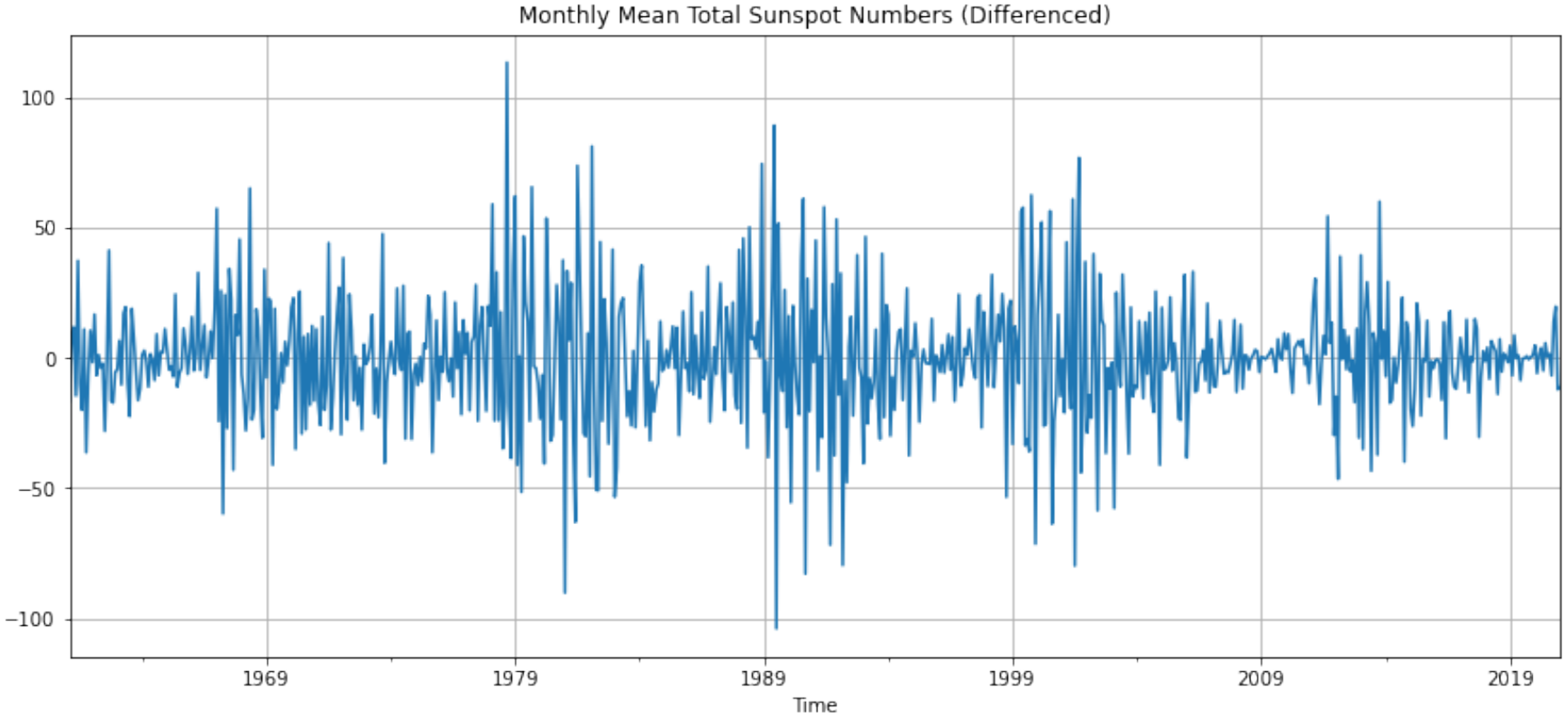
Out[12]:
```
Text(0, 0.5, 'Sunspots')
```



This is our series plotted over time. The 11-year solar cycle is immediately clear. There is no clear trend based on an initial visual inspection.

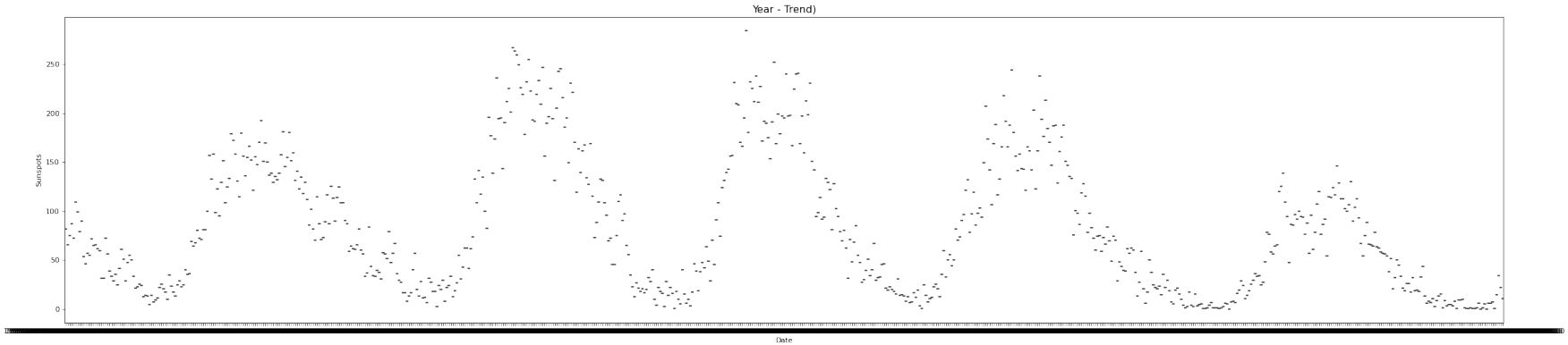
In [51]:
```
diff = sunspot['Diff']
diff = diff.dropna()

#Plot differenced data
plt.figure(figsize = (14,6))
diff.plot(grid = True)
plt.title("Monthly Mean Total Sunspot Numbers (Differenced)")
plt.xlabel("Time")
```

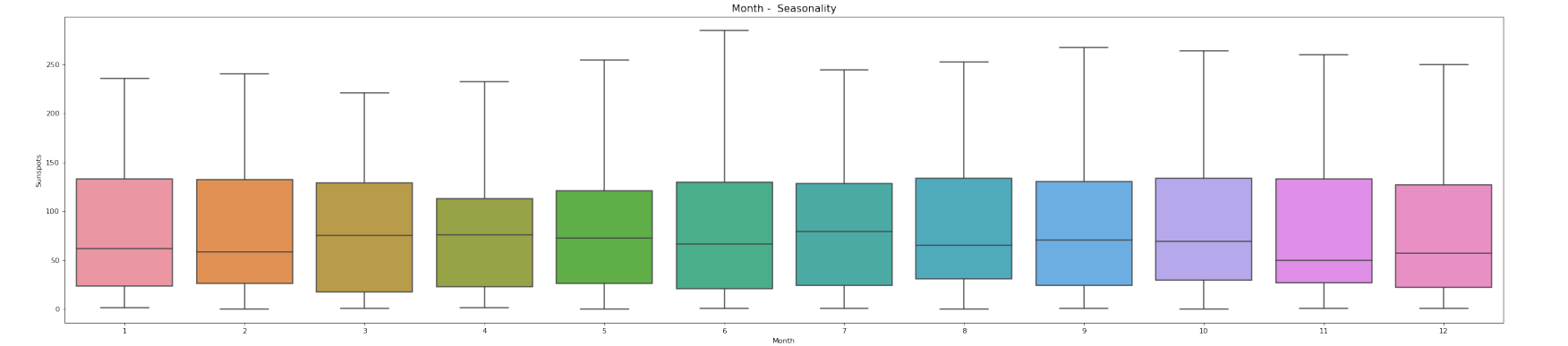
Out[51]:
```
Text(0.5, 0, 'Time')
```

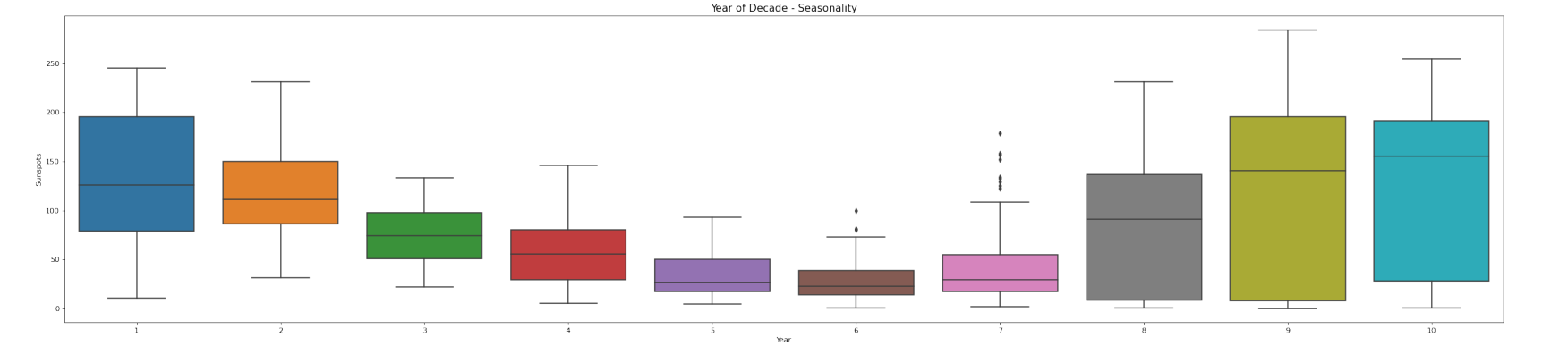


We plot differenced data to get a better understanding. There are some dynamics left in the amplitude after differencing.

In [52]:
```
#Plot data using Seaborn

fig, axes = plt.subplots(3, 1, figsize=(30,20), dpi= 80)
sns.boxplot(x='Date', y='Sunspots', data=df_sub, ax=axes[0])
sns.boxplot(x='Month', y='Sunspots', data=df_sub,ax = axes[1])
sns.boxplot(x='Year', y='Sunspots', data=df_sub,ax = axes[2])
# Set Title
axes[0].set_title('Year - Trend)', fontsize=14);
axes[1].set_title('Month -  Seasonality', fontsize=14)
axes[2].set_title('Year of Decade - Seasonality', fontsize=14)
fig.tight_layout()
plt.show()
plt.savefig('graph.png', dpi=300)
```







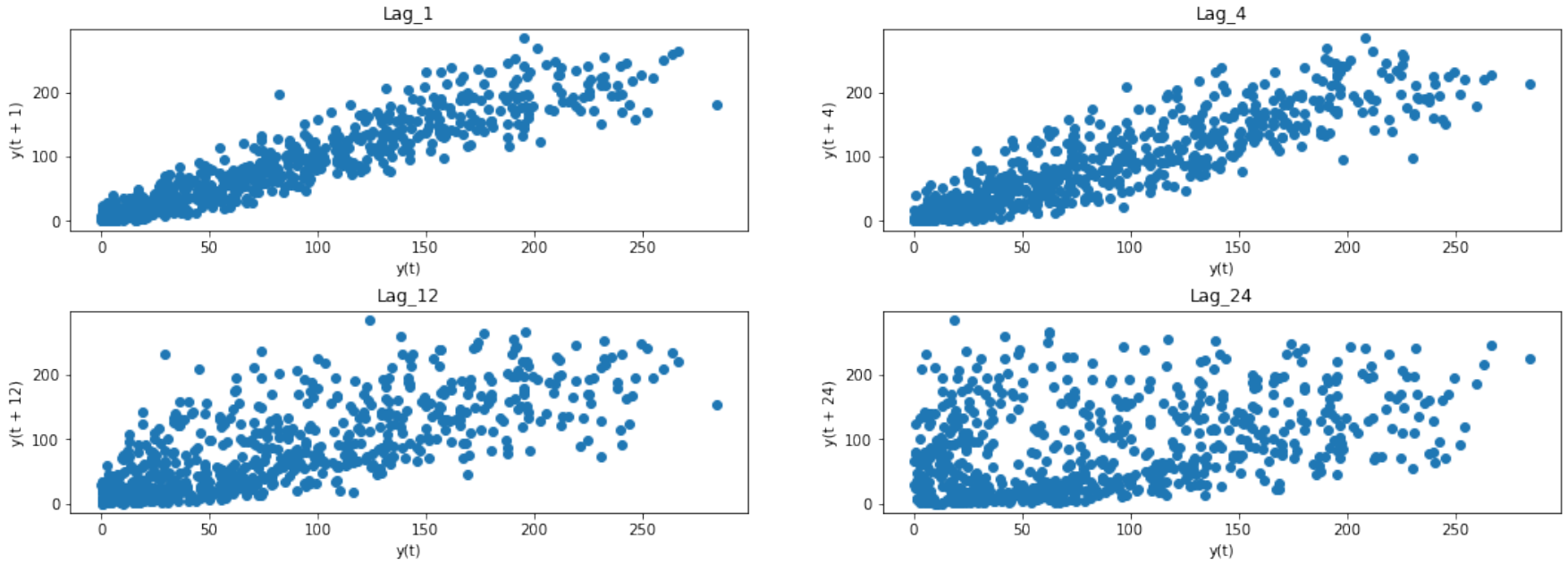
<Figure size 432x288 with 0 Axes>

Interpretation

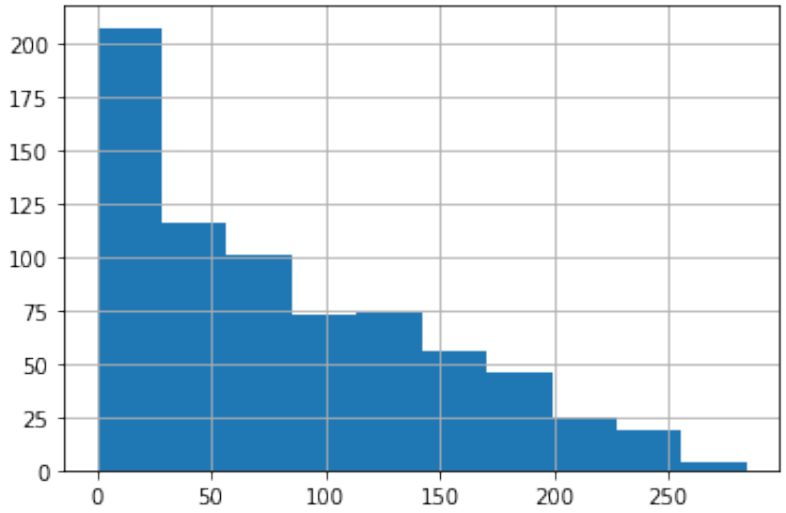
Interestingly, the sunspots do not vary much month to month, but do vary based on the year of the decade the observation falls within. This reflects the 11 year solar cycle we would expect to see, and shows our data does not have much seasonality month to month.

Plot Lags and Distribution

In [25]:
```
fig=plt.figure(figsize=(18,6))
fig.subplots_adjust(hspace=0.4, wspace=0.2)
ax1=fig.add_subplot(2,2,1)
pd.plotting.lag_plot(sunspot['Sunspots'],lag=1)
plt.title('Lag_1')
ax2=fig.add_subplot(2,2,2)
pd.plotting.lag_plot(sunspot['Sunspots'],lag=4)
plt.title('Lag_4')
ax3=fig.add_subplot(2,2,3)
pd.plotting.lag_plot(sunspot['Sunspots'],lag=12)
plt.title('Lag_12')
ax3=fig.add_subplot(2,2,4)
pd.plotting.lag_plot(sunspot['Sunspots'],lag=24)
plt.title('Lag_24')
plt.show()
```



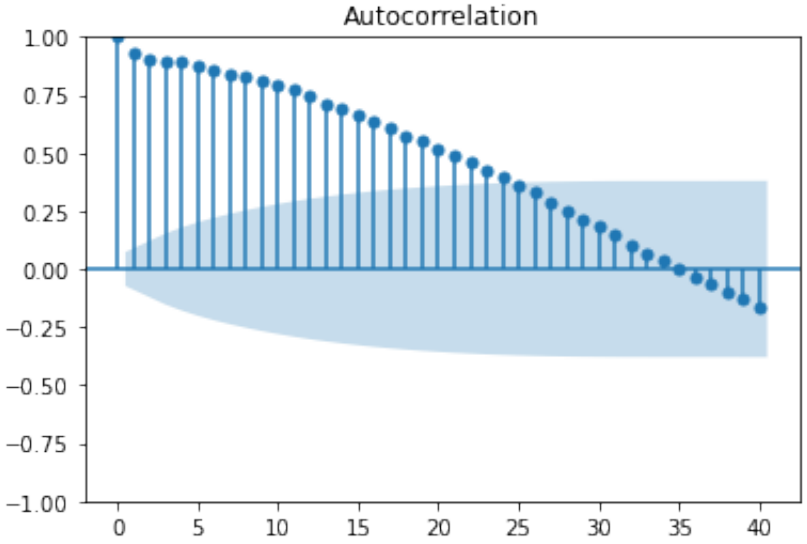
In [13]:
```
sunspot['Sunspots'].hist()
plt.show()
```

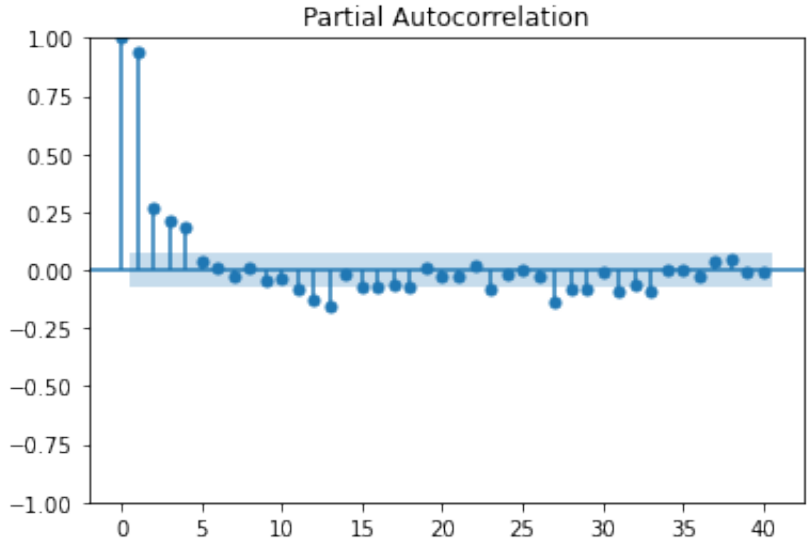


ACF and PACF

In [53]:
```
#Plot ACF and PACF of data
plot_acf(sunspot['Sunspots'], lags = 40)

plot_pacf(sunspot['Sunspots'], lags = 40)
plt.show()
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: Fu
tureWarning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13,
the default will change tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method=
'ywm'.
  warnings.warn(
```

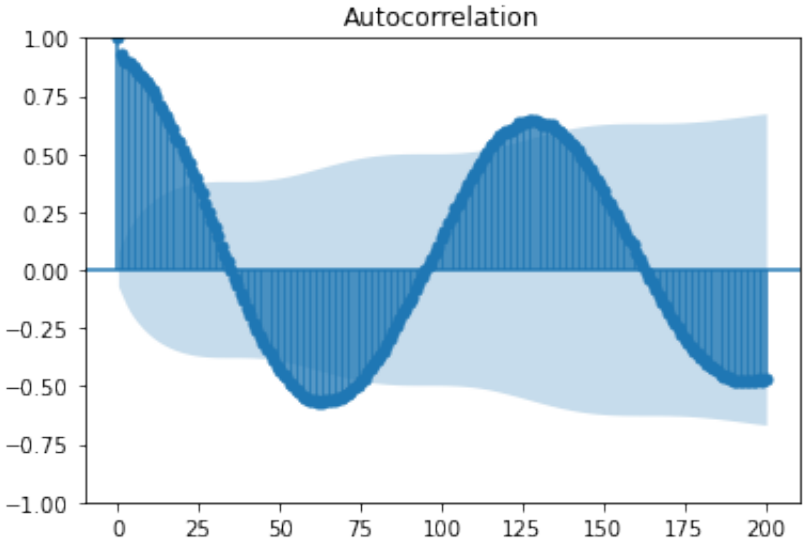


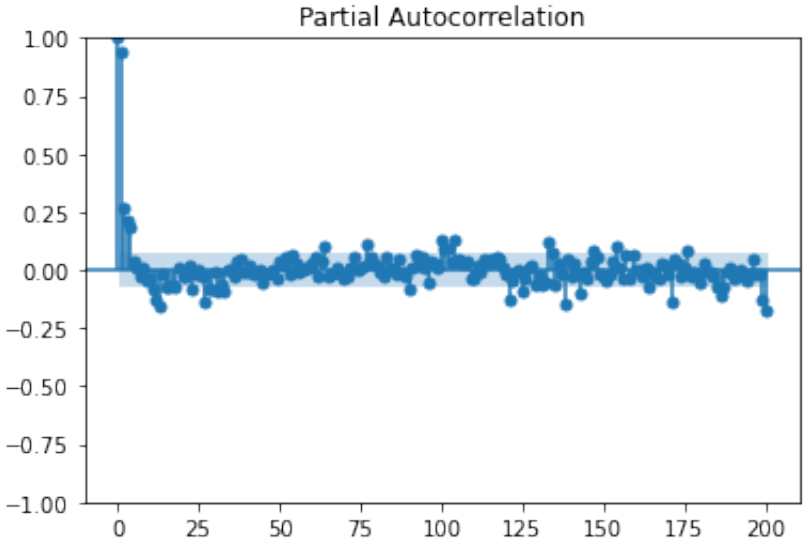


The ACF decays to 0 and there appear to be 4 significant spikes in the PACF, indicating a potential AR(4) process.

In [14]:
```
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot, plot_acf, plot_pacf, quarter_plot

#Plot ACF and PACF of data with lag up to 200 months to observe the 11 year cycles
plot_acf(sunspot['Sunspots'], lags = 200)

plot_pacf(sunspot['Sunspots'], lags = 200)
plt.show()
```

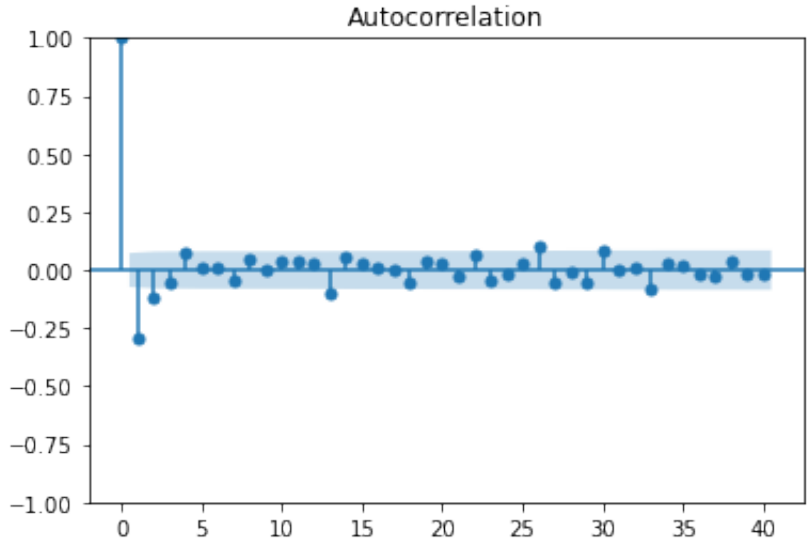
```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: Fu
tureWarning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13,
the default will change tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method=
'ywm'.
  warnings.warn(
```

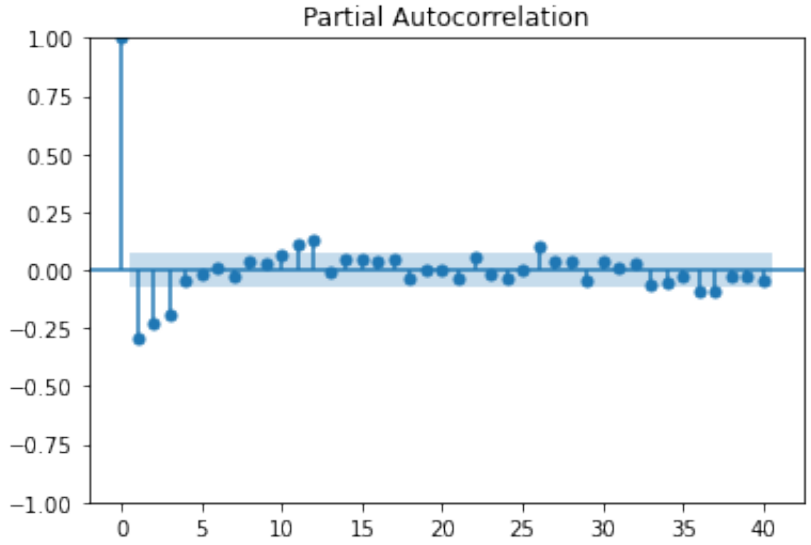

Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
25
50
75
100
125
150
175
200


Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
25
50
75
100
125
150
175
200

In [15]:
```
#Plot ACF and PACF of differenced data

plot_acf(diff, lags = 40)

plot_pacf(diff, lags = 40)
plt.show()
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```

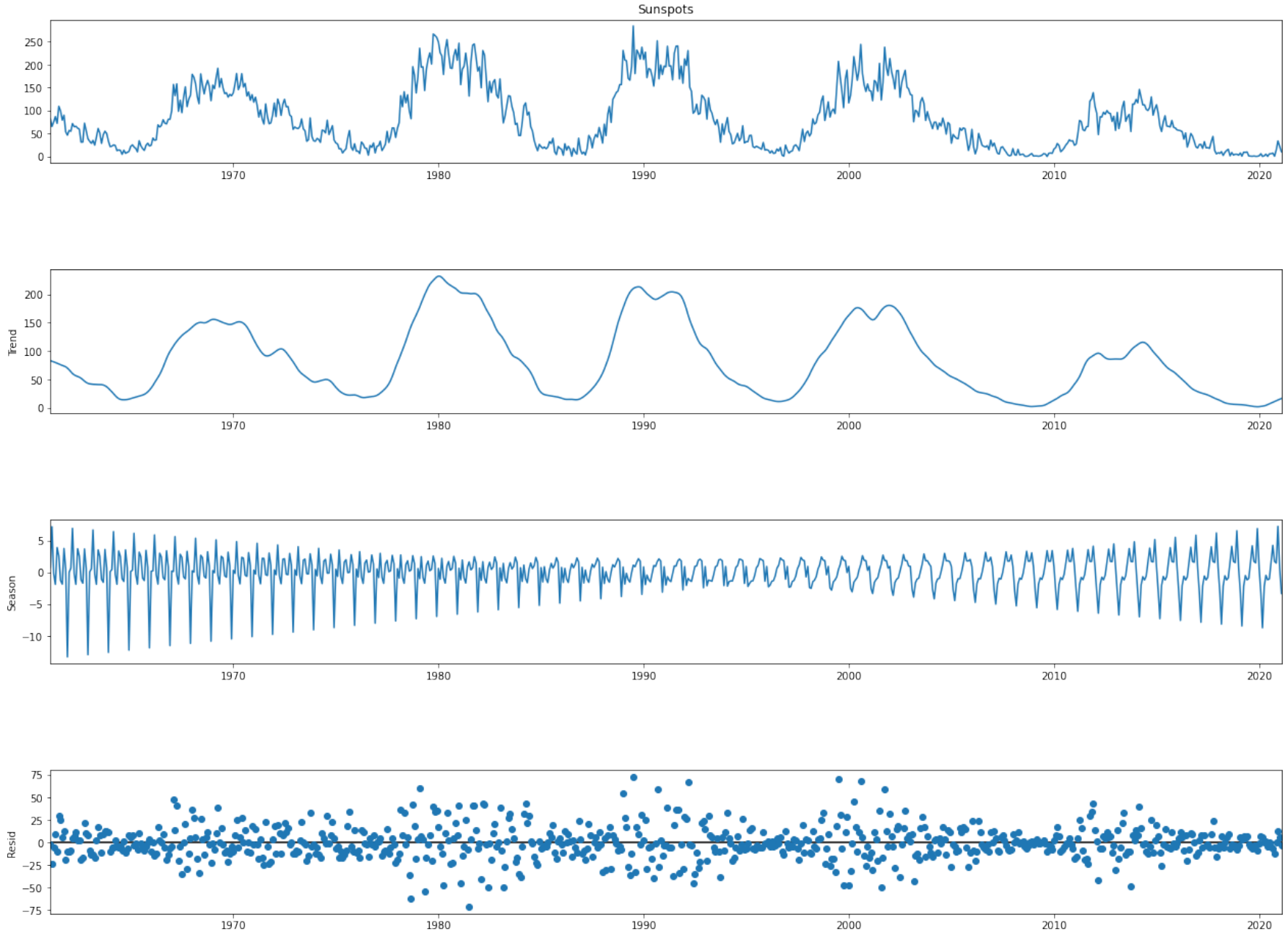




Plot the STL decomposition

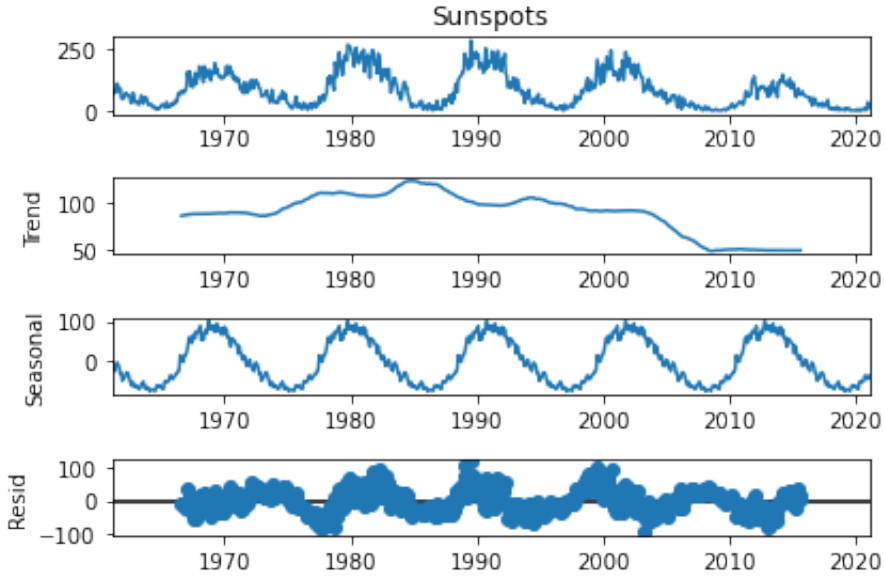
In [54]:
```
from statsmodels.tsa.seasonal import STL
```

In [60]:
```
# Plotting the STL decomposition
stl = STL(sunspot['Sunspots'], seasonal = 133)
results = stl.fit()

# Plot the results
fig = results.plot()

fig.set_figwidth(20)
fig.set_figheight(15)
plt.show()
```


Sunspots
250
200
150
100
50
0
1970
1980
1990
2000
2010
2020
Trend
200
150
100
50
0
1970
1980
1990
2000
2010
2020
Season
5
0
−5
−10
1970
1980
1990
2000
2010
2020
Resid
75
50
25
0
−25
−50
−75
1970
1980
1990
2000
2010
2020

In [14]:
```
#Plot decomposition using seasonal decompose with 132 month (11 year) period.

from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt

# Perform seasonal decomposition
result = seasonal_decompose(sunspot['Sunspots'], model="additive", period=11*12)

# Set larger figure size
fig = plt.figure(figsize=(25, 15))

# Plot the decomposition
result.plot()

# Set the DPI (dots per inch) for high resolution
plt.savefig('seasonal_decomposition.png', dpi=300)

# Show the plot
plt.show()
```

```
<Figure size 1800x1080 with 0 Axes>
```



The trend flattens out and the 11 year cyclicty now appears to show up in the seasonal part. Both decomposition methods exhibit remaining patterns of cyclicity in the residuals.

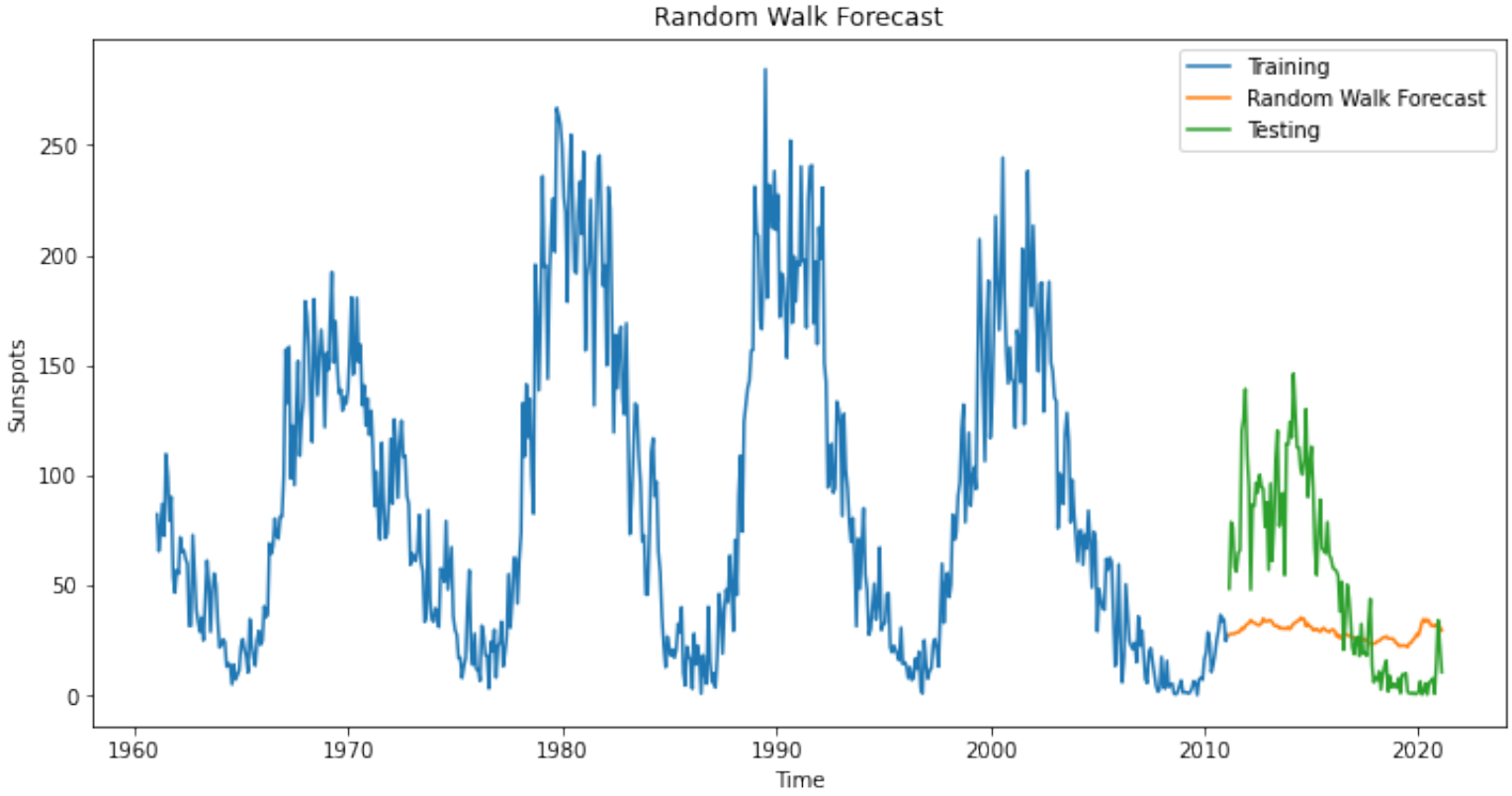
Models

We fit several models and use the out-of-sample Mean Absolute Percentage Error to evaluate their performance.

Random Walk (Baseline Model)

In [22]:
```
# Number of out-of-sample forecast steps
num_forecast_steps = 120

# Initialize forecast list with available data
forecast = list(train["Sunspots"].copy())

# Perform out-of-sample random walk forecast
for i in range(num_forecast_steps):
    forecasted_value = forecast[-1] + np.random.normal(0, 1)  # Random noise
    forecast.append(forecasted_value)

# Creating a dataframe for the forecasted values
test_forecast_df = pd.DataFrame({"Test Forecast": forecast[-120:]}, index = test.index)
```

In [25]:
```
# Plot original data and out-of-sample forecasted values
plt.figure(figsize = (12,6))
plt.plot(train, label='Training')
plt.plot(test_forecast_df["Test Forecast"], label='Random Walk Forecast')
plt.plot(test, label = 'Testing')
plt.xlabel("Time")
plt.ylabel("Sunspots")
plt.legend()
plt.title("Random Walk Forecast")
plt.savefig("rw.png", dpi = 300)
plt.show()
```



In [82]:
```
# Model Evaluation: Calculating the MAPE
mape_rw = np.mean(np.abs((test["Sunspots"] - test_forecast_df["Test Forecast"])/test["Sunspots"]))*100

print("Random Walk MAPE:", round(mape_rw,2), "%")
```

```
Random Walk MAPE: 941.98 %
```

We use the Random Walk Forecast as a baseline model. As seen in the plot of the forecast as well as in the exorbitant MAPE of 941.98%, it fails to capture the dynamics of our data.

Seasonal Naive

In [37]:
```
import rpy2

import warnings
warnings.filterwarnings('ignore')

from rpy2.robjects import pandas2ri
import rpy2.rinterface as rinterface
pandas2ri.activate()

%load_ext rpy2.ipython
```

In [38]:
```
%R -i df
%R -i train
%R -i test
```

In [39]:
```
%%R
library(forecast)

# Converting data to time series format in R
train_ts <- ts(train, frequency = 12, start = 1961)
test_ts <- ts(test, frequency = 12, start = 2011)

# Fitting the model and forecasting over testing set
snaive_fc <- snaive(train_ts, h = length(test_ts))
```

```
R[write to console]: Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

In [40]:
```
%%R
summary(snaive_fc)
```

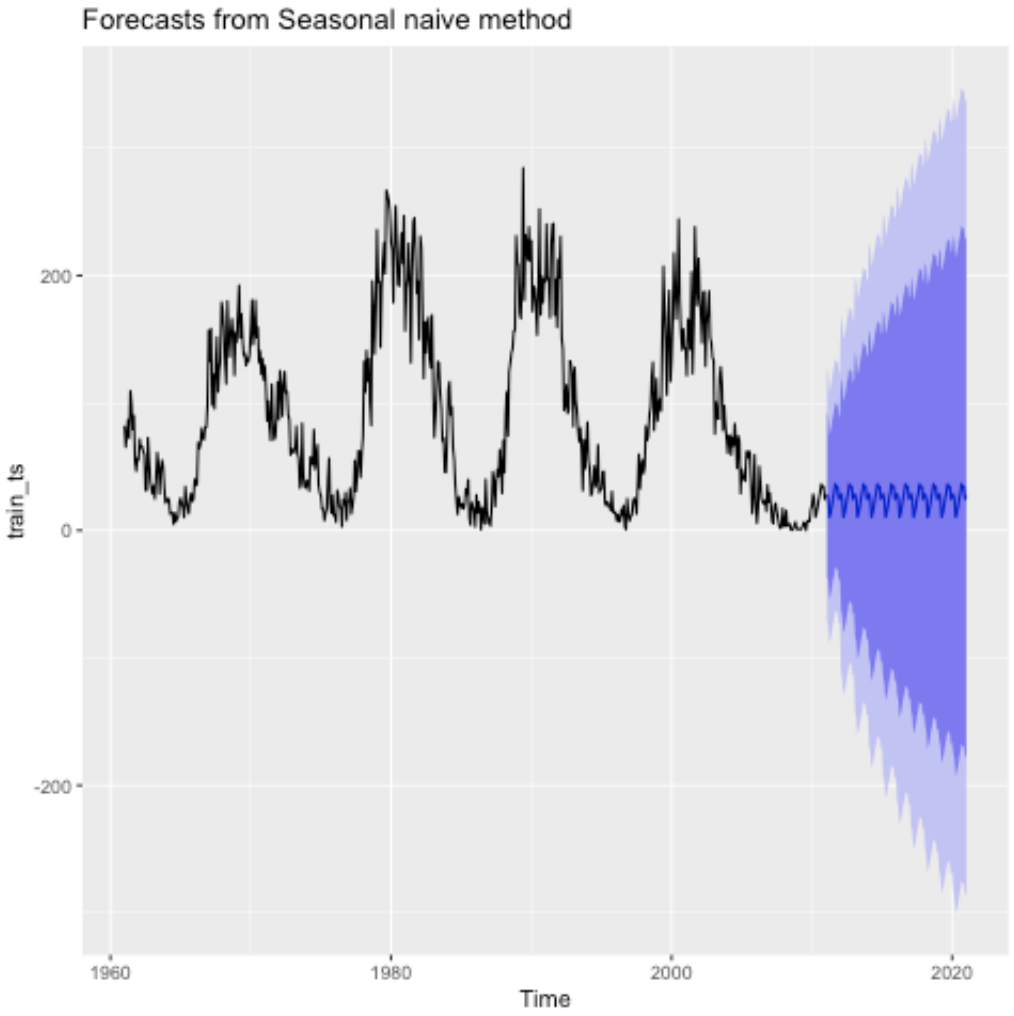
```
Forecast method: Seasonal naive method

Model Information:
Call: snaive(y = train_ts, h = length(test_ts))

Residual sd: 50.1705

Error measures:
                    ME     RMSE      MAE  MPE MAPE MASE      ACF1
Training set -1.036503 50.17051 37.83447 -Inf  Inf    1 0.7462501

Forecasts:
         Point Forecast      Lo 80    Hi 80     Lo 95    Hi 95
Feb 2011           28.5  -35.79610  92.7961  -69.8324 126.8324
Mar 2011           24.0  -40.29610  88.2961  -74.3324 122.3324
Apr 2011           10.4  -53.89610  74.6961  -87.9324 108.7324
May 2011           13.9  -50.39610  78.1961  -84.4324 112.2324
Jun 2011           18.8  -45.49610  83.0961  -79.5324 117.1324
Jul 2011           25.2  -39.09610  89.4961  -73.1324 123.5324
Aug 2011           29.6  -34.69610  93.8961  -68.7324 127.9324
Sep 2011           36.4  -27.89610 100.6961  -61.9324 134.7324
```

Oct 2011	33.6	-30.69610	97.8961	-64.7324	131.9324
Nov 2011	34.4	-29.89610	98.6961	-63.9324	132.7324
Dec 2011	24.5	-39.79610	88.7961	-73.8324	122.8324
Jan 2012	27.3	-36.99610	91.5961	-71.0324	125.6324
Feb 2012	28.5	-62.42842	119.4284	-110.5630	167.5630
Mar 2012	24.0	-66.92842	114.9284	-115.0630	163.0630
Apr 2012	10.4	-80.52842	101.3284	-128.6630	149.4630
May 2012	13.9	-77.02842	104.8284	-125.1630	152.9630
Jun 2012	18.8	-72.12842	109.7284	-120.2630	157.8630
Jul 2012	25.2	-65.72842	116.1284	-113.8630	164.2630
Aug 2012	29.6	-61.32842	120.5284	-109.4630	168.6630
Sep 2012	36.4	-54.52842	127.3284	-102.6630	175.4630
Oct 2012	33.6	-57.32842	124.5284	-105.4630	172.6630
Nov 2012	34.4	-56.52842	125.3284	-104.6630	173.4630
Dec 2012	24.5	-66.42842	115.4284	-114.5630	163.5630
Jan 2013	27.3	-63.62842	118.2284	-111.7630	166.3630
Feb 2013	28.5	-82.86411	139.8641	-141.8167	198.8167
Mar 2013	24.0	-87.36411	135.3641	-146.3167	194.3167
Apr 2013	10.4	-100.96411	121.7641	-159.9167	180.7167
May 2013	13.9	-97.46411	125.2641	-156.4167	184.2167
Jun 2013	18.8	-92.56411	130.1641	-151.5167	189.1167
Jul 2013	25.2	-86.16411	136.5641	-145.1167	195.5167
Aug 2013	29.6	-81.76411	140.9641	-140.7167	199.9167
Sep 2013	36.4	-74.96411	147.7641	-133.9167	206.7167
Oct 2013	33.6	-77.76411	144.9641	-136.7167	203.9167
Nov 2013	34.4	-76.96411	145.7641	-135.9167	204.7167
Dec 2013	24.5	-86.86411	135.8641	-145.8167	194.8167
Jan 2014	27.3	-84.06411	138.6641	-143.0167	197.6167
Feb 2014	28.5	-100.09220	157.0922	-168.1648	225.1648
Mar 2014	24.0	-104.59220	152.5922	-172.6648	220.6648
Apr 2014	10.4	-118.19220	138.9922	-186.2648	207.0648
May 2014	13.9	-114.69220	142.4922	-182.7648	210.5648
Jun 2014	18.8	-109.79220	147.3922	-177.8648	215.4648
Jul 2014	25.2	-103.39220	153.7922	-171.4648	221.8648
Aug 2014	29.6	-98.99220	158.1922	-167.0648	226.2648
Sep 2014	36.4	-92.19220	164.9922	-160.2648	233.0648
Oct 2014	33.6	-94.99220	162.1922	-163.0648	230.2648
Nov 2014	34.4	-94.19220	162.9922	-162.2648	231.0648
Dec 2014	24.5	-104.09220	153.0922	-172.1648	221.1648

Jan 2015	27.3	-101.29220	155.8922	-169.3648	223.9648
Feb 2015	28.5	-115.27045	172.2705	-191.3779	248.3779
Mar 2015	24.0	-119.77045	167.7705	-195.8779	243.8779
Apr 2015	10.4	-133.37045	154.1705	-209.4779	230.2779
May 2015	13.9	-129.87045	157.6705	-205.9779	233.7779
Jun 2015	18.8	-124.97045	162.5705	-201.0779	238.6779
Jul 2015	25.2	-118.57045	168.9705	-194.6779	245.0779
Aug 2015	29.6	-114.17045	173.3705	-190.2779	249.4779
Sep 2015	36.4	-107.37045	180.1705	-183.4779	256.2779
Oct 2015	33.6	-110.17045	177.3705	-186.2779	253.4779
Nov 2015	34.4	-109.37045	178.1705	-185.4779	254.2779
Dec 2015	24.5	-119.27045	168.2705	-195.3779	244.3779
Jan 2016	27.3	-116.47045	171.0705	-192.5779	247.1779
Feb 2016	28.5	-128.99264	185.9926	-212.3642	269.3642
Mar 2016	24.0	-133.49264	181.4926	-216.8642	264.8642
Apr 2016	10.4	-147.09264	167.8926	-230.4642	251.2642
May 2016	13.9	-143.59264	171.3926	-226.9642	254.7642
Jun 2016	18.8	-138.69264	176.2926	-222.0642	259.6642
Jul 2016	25.2	-132.29264	182.6926	-215.6642	266.0642
Aug 2016	29.6	-127.89264	187.0926	-211.2642	270.4642
Sep 2016	36.4	-121.09264	193.8926	-204.4642	277.2642
Oct 2016	33.6	-123.89264	191.0926	-207.2642	274.4642
Nov 2016	34.4	-123.09264	191.8926	-206.4642	275.2642
Dec 2016	24.5	-132.99264	181.9926	-216.3642	265.3642
Jan 2017	27.3	-130.19264	184.7926	-213.5642	268.1642
Feb 2017	28.5	-141.61149	198.6115	-231.6631	288.6631
Mar 2017	24.0	-146.11149	194.1115	-236.1631	284.1631
Apr 2017	10.4	-159.71149	180.5115	-249.7631	270.5631
May 2017	13.9	-156.21149	184.0115	-246.2631	274.0631
Jun 2017	18.8	-151.31149	188.9115	-241.3631	278.9631
Jul 2017	25.2	-144.91149	195.3115	-234.9631	285.3631
Aug 2017	29.6	-140.51149	199.7115	-230.5631	289.7631
Sep 2017	36.4	-133.71149	206.5115	-223.7631	296.5631
Oct 2017	33.6	-136.51149	203.7115	-226.5631	293.7631
Nov 2017	34.4	-135.71149	204.5115	-225.7631	294.5631
Dec 2017	24.5	-145.61149	194.6115	-235.6631	284.6631
Jan 2018	27.3	-142.81149	197.4115	-232.8631	287.4631
Feb 2018	28.5	-153.35684	210.3568	-249.6260	306.6260
Mar 2018	24.0	-157.85684	205.8568	-254.1260	302.1260

```
Apr 2018          10.4 -171.45684 192.2568 -267.7260 288.5260
May 2018          13.9 -167.95684 195.7568 -264.2260 292.0260
Jun 2018          18.8 -163.05684 200.6568 -259.3260 296.9260
Jul 2018          25.2 -156.65684 207.0568 -252.9260 303.3260
Aug 2018          29.6 -152.25684 211.4568 -248.5260 307.7260
Sep 2018          36.4 -145.45684 218.2568 -241.7260 314.5260
Oct 2018          33.6 -148.25684 215.4568 -244.5260 311.7260
Nov 2018          34.4 -147.45684 216.2568 -243.7260 312.5260
Dec 2018          24.5 -157.35684 206.3568 -253.6260 302.6260
Jan 2019          27.3 -154.55684 209.1568 -250.8260 305.4260
Feb 2019          28.5 -164.38830 221.3883 -266.4972 323.4972
Mar 2019          24.0 -168.88830 216.8883 -270.9972 318.9972
Apr 2019          10.4 -182.48830 203.2883 -284.5972 305.3972
May 2019          13.9 -178.98830 206.7883 -281.0972 308.8972
Jun 2019          18.8 -174.08830 211.6883 -276.1972 313.7972
Jul 2019          25.2 -167.68830 218.0883 -269.7972 320.1972
Aug 2019          29.6 -163.28830 222.4883 -265.3972 324.5972
Sep 2019          36.4 -156.48830 229.2883 -258.5972 331.3972
Oct 2019          33.6 -159.28830 226.4883 -261.3972 328.5972
Nov 2019          34.4 -158.48830 227.2883 -260.5972 329.3972
Dec 2019          24.5 -168.38830 217.3883 -270.4972 319.4972
Jan 2020          27.3 -165.58830 220.1883 -267.6972 322.2972
Feb 2020          28.5 -174.82213 231.8221 -282.4544 339.4544
Mar 2020          24.0 -179.32213 227.3221 -286.9544 334.9544
Apr 2020          10.4 -192.92213 213.7221 -300.5544 321.3544
May 2020          13.9 -189.42213 217.2221 -297.0544 324.8544
Jun 2020          18.8 -184.52213 222.1221 -292.1544 329.7544
Jul 2020          25.2 -178.12213 228.5221 -285.7544 336.1544
Aug 2020          29.6 -173.72213 232.9221 -281.3544 340.5544
Sep 2020          36.4 -166.92213 239.7221 -274.5544 347.3544
Oct 2020          33.6 -169.72213 236.9221 -277.3544 344.5544
Nov 2020          34.4 -168.92213 237.7221 -276.5544 345.3544
Dec 2020          24.5 -178.82213 227.8221 -286.4544 335.4544
Jan 2021          27.3 -176.02213 230.6221 -283.6544 338.2544
```

In [41]:
```
%%R
# Plotting the forecasts
autoplot(snaive_fc)
```



In [44]:
```
%%R

# Saving forecasts and calling into python
write.csv(snaive_fc, "SN_fore.csv")
```

In [45]:
```
sn_fore = pd.read_csv("SN_fore.csv")
sn_fore = sn_fore["Point.Forecast"]
sn_fore.index = test["Sunspots"].index
```

In [47]:
```
%%R
sn_fitted <- fitted(snaive_fc) # Fitted values
```

In [48]:
```
%R -o sn_fitted # Transferring fitted values to python
```

In [83]:
```
# Evaluating the model: calculating the MAPE
mape_sn_fore = np.mean(np.abs((test["Sunspots"] - sn_fore)/test["Sunspots"]))*100

# Print the MAPE
print("Seasonal Naive MAPE:", round(mape_sn_fore,2), "%")
```

```
Seasonal Naive MAPE: 628.21 %
```

The Seasonal Naive forecast is an improvement over the Random Walk but still has a very high MAPE.

ARIMA model that includes, trend, seasonality and cyclical components.

In [15]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [ ]:
```
#Fit custom ARIMA model
```

In [35]:
```
model_arima = ARIMA(sunspot['Sunspots'], order=(4, 0, 0), seasonal_order = (2, 1, 0, 12),
              trend = (0, 1, 0)).fit()
print(model_arima.summary())
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

Dep. Variable:	Sunspots	No. Observations:	721
Model:	ARIMA(4, 0, 0)x(2, 1, 0, 12)	Log Likelihood	-3311.868
Date:	Thu, 08 Jun 2023	AIC	6639.735
Time:	23:06:12	BIC	6676.246
Sample:	01-31-1961	HQIC	6653.840
	- 01-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
x1	-0.1499	0.957	-0.157	0.876	-2.026	1.726
ar.L1	0.5847	0.032	18.028	0.000	0.521	0.648
ar.L2	0.0580	0.037	1.571	0.116	-0.014	0.130
ar.L3	0.1363	0.035	3.878	0.000	0.067	0.205
ar.L4	0.1737	0.032	5.467	0.000	0.111	0.236
ar.S.L12	-0.6041	0.031	-19.184	0.000	-0.666	-0.542
ar.S.L24	-0.2817	0.032	-8.722	0.000	-0.345	-0.218
sigma2	662.3621	26.975	24.554	0.000	609.492	715.233

Ljung-Box (L1) (Q):	0.04	Jarque-Bera (JB):	72.61
Prob(Q):	0.83	Prob(JB):	0.00
Heteroskedasticity (H):	0.72	Skew:	-0.02
Prob(H) (two-sided):	0.01	Kurtosis:	4.57

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [4]:
```
#Fit custom SARIMA model
```

In [36]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model_sarimax = SARIMAX(sunspot['Sunspots'], order=(4, 0, 0), seasonal_order=(2, 1, 0, 12),
                        trend='c').fit()
print(model_sarimax.summary())
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
 This problem is unconstrained.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            8     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.82969D+00    |proj g|=  4.46347D-01

At iterate    5    f=  4.62399D+00    |proj g|=  3.42981D-02

At iterate   10    f=  4.61136D+00    |proj g|=  1.16686D-02

At iterate   15    f=  4.59912D+00    |proj g|=  3.47076D-03

At iterate   20    f=  4.59876D+00    |proj g|=  2.07340D-02

At iterate   25    f=  4.59400D+00    |proj g|=  2.67040D-02

At iterate   30    f=  4.59347D+00    |proj g|=  3.08857D-04
```

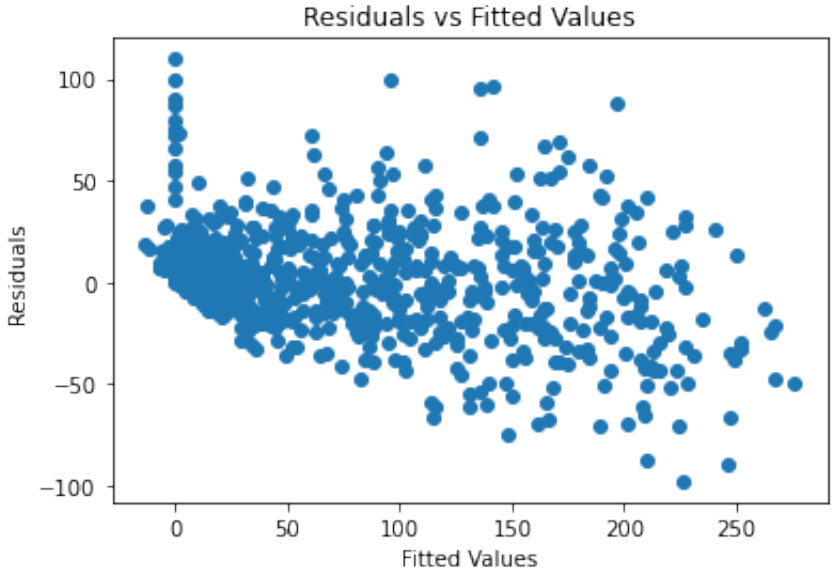
```
At iterate   35    f=  4.59344D+00    |proj g|=  2.90051D-03

At iterate   40    f=  4.59344D+00    |proj g|=  3.78380D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    8     41     54      1     0     0   2.044D-05   4.593D+00
  F =   4.5934363068338326

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
                                 SARIMAX Results
==========================================================================================
Dep. Variable:                             Sunspots   No. Observations:                  721
Model:             SARIMAX(4, 0, 0)x(2, 1, 0, 12)   Log Likelihood               -3311.868
Date:                            Thu, 08 Jun 2023   AIC                           6639.735
Time:                                    23:06:35   BIC                           6676.246
Sample:                                01-31-1961   HQIC                          6653.840
                                     - 01-31-2021
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.1451      1.012     -0.143      0.886      -2.128       1.838
ar.L1          0.5848      0.032     18.028      0.000       0.521       0.648
ar.L2          0.0580      0.037      1.572      0.116      -0.014       0.130
ar.L3          0.1362      0.035      3.877      0.000       0.067       0.205
ar.L4          0.1736      0.032      5.467      0.000       0.111       0.236
ar.S.L12      -0.6041      0.031    -19.185      0.000      -0.666      -0.542
```

```
ar.S.L24      -0.2817      0.032     -8.723      0.000      -0.345      -0.218
sigma2       662.4005     26.978     24.554      0.000     609.525     715.276
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):                72.61
Prob(Q):                              0.83   Prob(JB):                         0.00
Heteroskedasticity (H):               0.72   Skew:                            -0.02
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.57
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

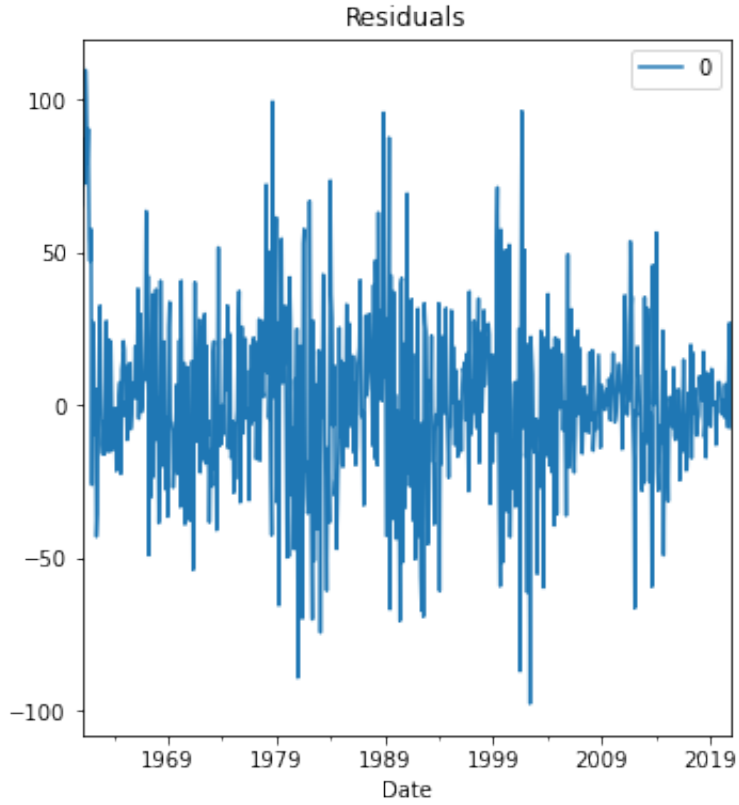
Plot the respective residuals vs. fitted values and discussing our observations.

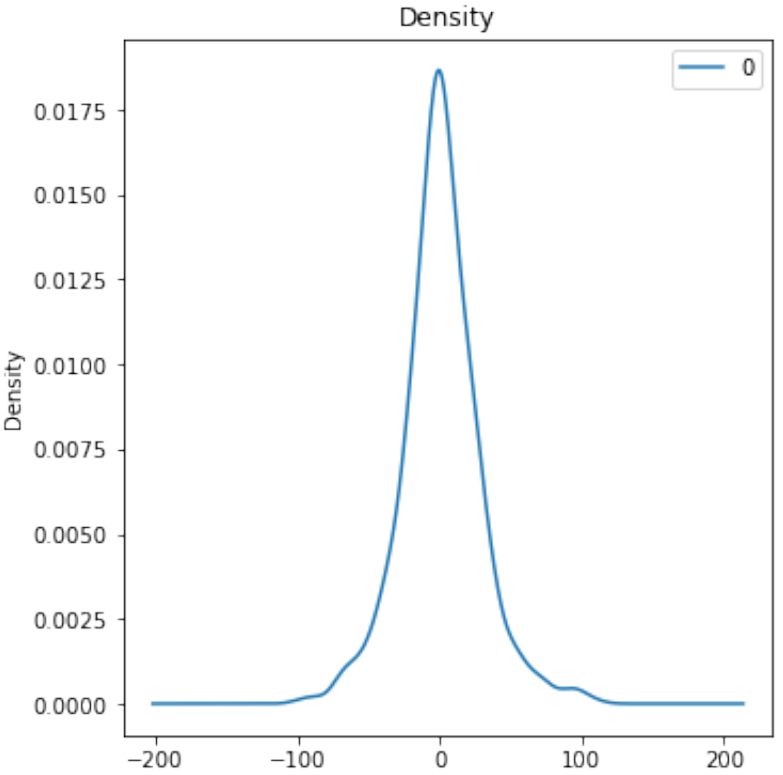
In [17]:
```
# Extracting residuals
residuals = pd.DataFrame(model_arima.resid[1:])

# Extracting fitted values
fv = pd.DataFrame(model_arima.fittedvalues[1:])
```

In [18]:
```
# Plotting residuals vs fitted values

plt.scatter(fv, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted Values')
plt.show()
```



In [19]:
```
#Additionnal residual diagnostics

fig, ax = plt.subplots(1, 2, figsize = (12,6))

residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
print(residuals.describe())
```

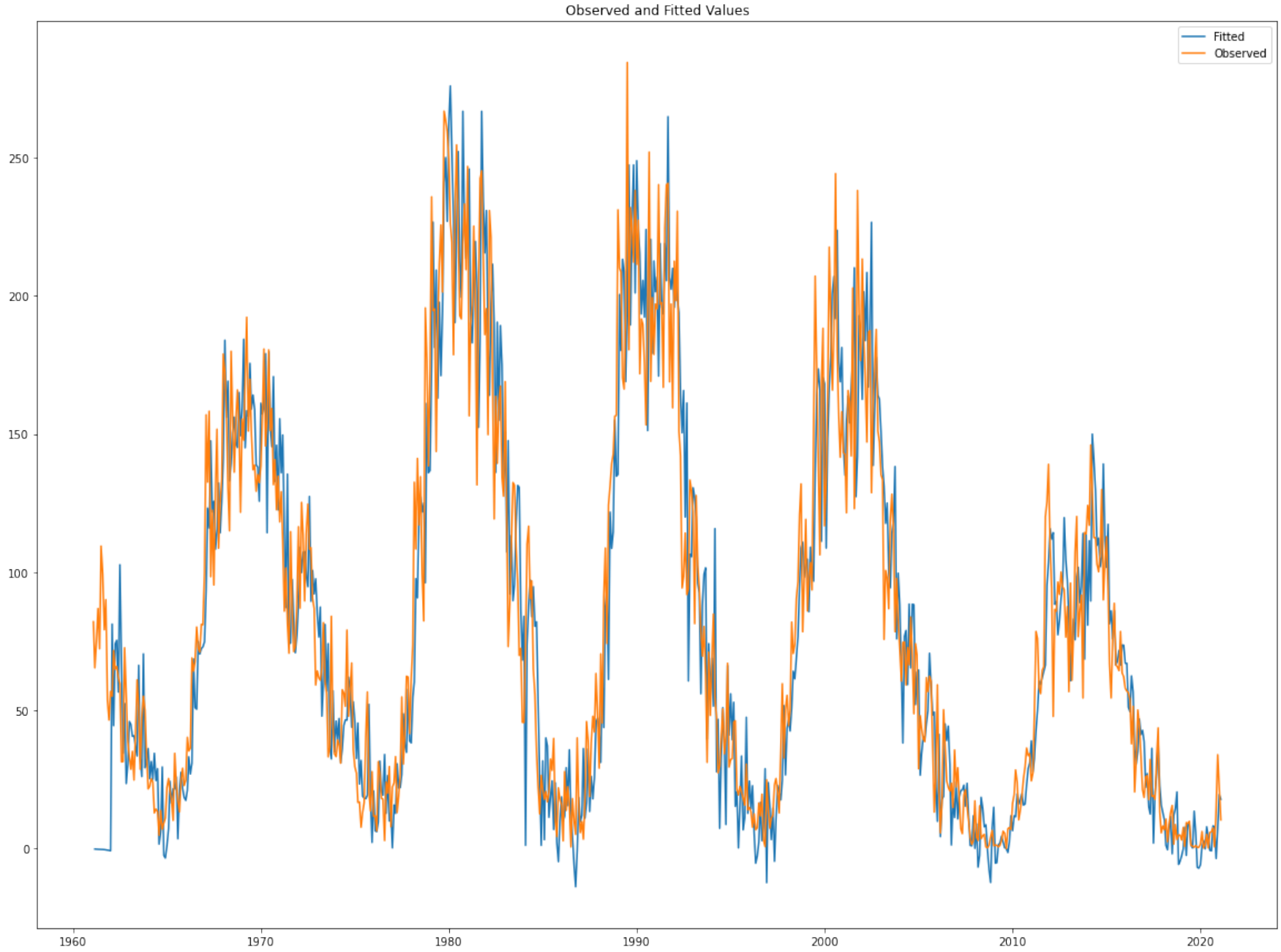




	0
count	720.000000
mean	1.269741
std	27.371350
min	-97.883337
25%	-13.641199
50%	-0.051166
75%	16.197404
max	109.812052

In [20]:
```
# Plotting fitted values and observed values
plt.figure(figsize = (20,15))
plt.title("Observed and Fitted Values")
plt.plot(fv, label = "Fitted")
plt.plot(sunspot['Sunspots'], label = "Observed")
plt.legend()
```

Out[20]:
```
<matplotlib.legend.Legend at 0x7fd62d109220>
```


Observed and Fitted Values
Fitted
Observed
250
200
150
100
50
0
1960
1970
1980
1990
2000
2010
2020

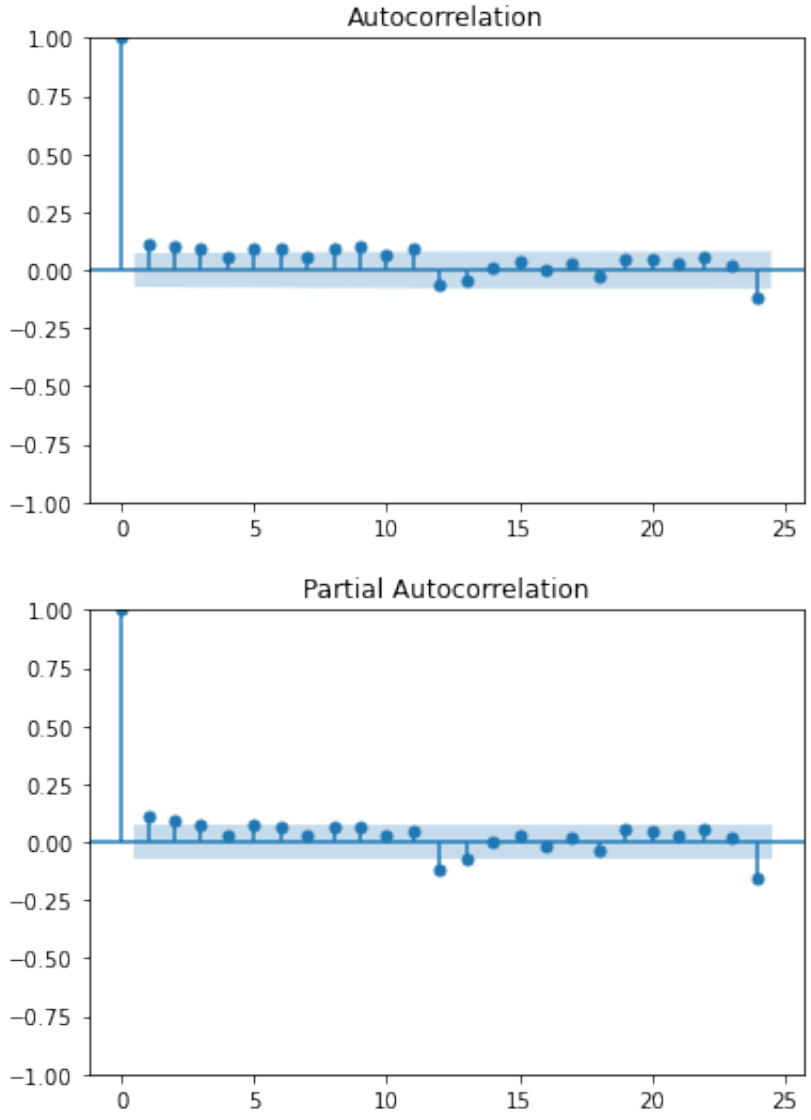
Fitted values overlap almost perfectly with observed values.

Plot the ACF and PACF of the residuals

In [23]:
```
from statsmodels.graphics.tsaplots import plot_acf
```

In [24]:
```
from statsmodels.graphics.tsaplots import plot_pacf
plot_acf(model_arima.resid, lags = 24)
plot_pacf(model_arima.resid, lags = 24)
plt.show()
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: Fu
tureWarning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13,
the default will change tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method=
'ywm'.
  warnings.warn(
```



The residuals look mostly clean, but the model could still be improved as there are a few spikes outside of the confidence bands.

In [ ]:
```
#Ljung Box Test
```

In [25]:
```
model_arima.test_serial_correlation('ljungbox', lags = 5)
```

Out[25]:
```
array([[[0.04456889, 0.17953738, 0.26990777, 1.15324052, 1.36146388],
        [0.83279874, 0.91414261, 0.96558496, 0.88573561, 0.92848184]]])
```

According to the results we can reject the null and conclude there is no autocorrelation at lag 5. Furhter analysis is needed, however, since 5 month lag is not very informative for our data.

Cusum Plot

In [26]:
```
# Convert to a model that recursive residuals can handle
#Fit model with differenced data
model_dif = ARIMA(diff, order=(4, 0, 0), seasonal_order = (1, 0, 0, 12),  # changed first terms from (4,1
              trend = (0, 1, 0)).fit()
```

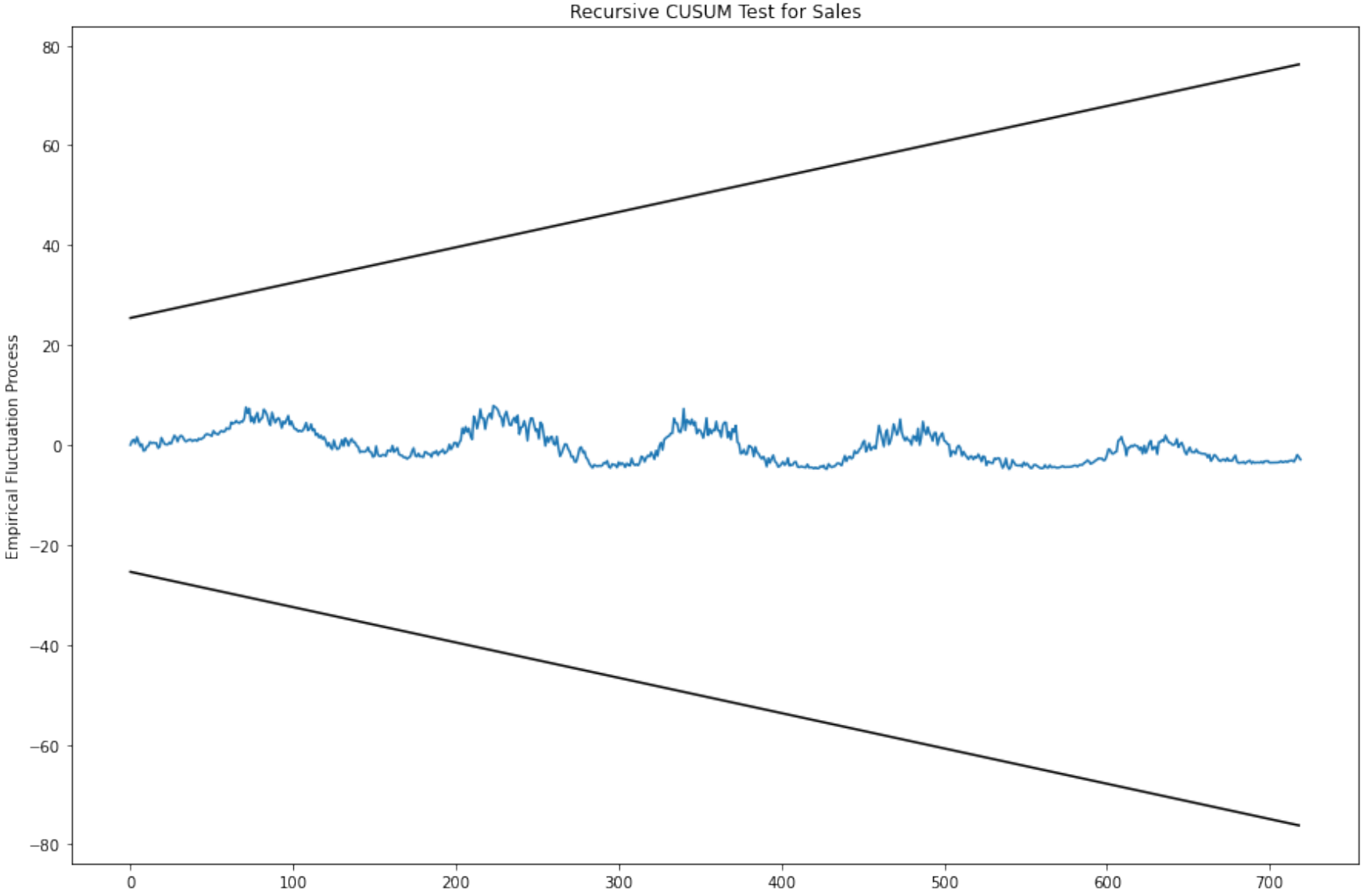
```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

In [107…

```
# Compute the recursive residuals
from statsmodels.stats.diagnostic import recursive_olsresiduals

recursive_resid = recursive_olsresiduals(model_dif)

plt.figure(figsize = (15,10))
plt.title('Recursive CUSUM Test for Sales')

# pull out and plot the first band of the CI
plt.plot(recursive_resid[6][0], color = 'black')

# pull out and plot the cusum test values
plt.plot(recursive_resid[5])

# pull out and plot the second band of the CI
plt.plot(recursive_resid[6][1], color = 'black')
plt.ylabel('Empirical Fluctuation Process')
```

Out[107]:

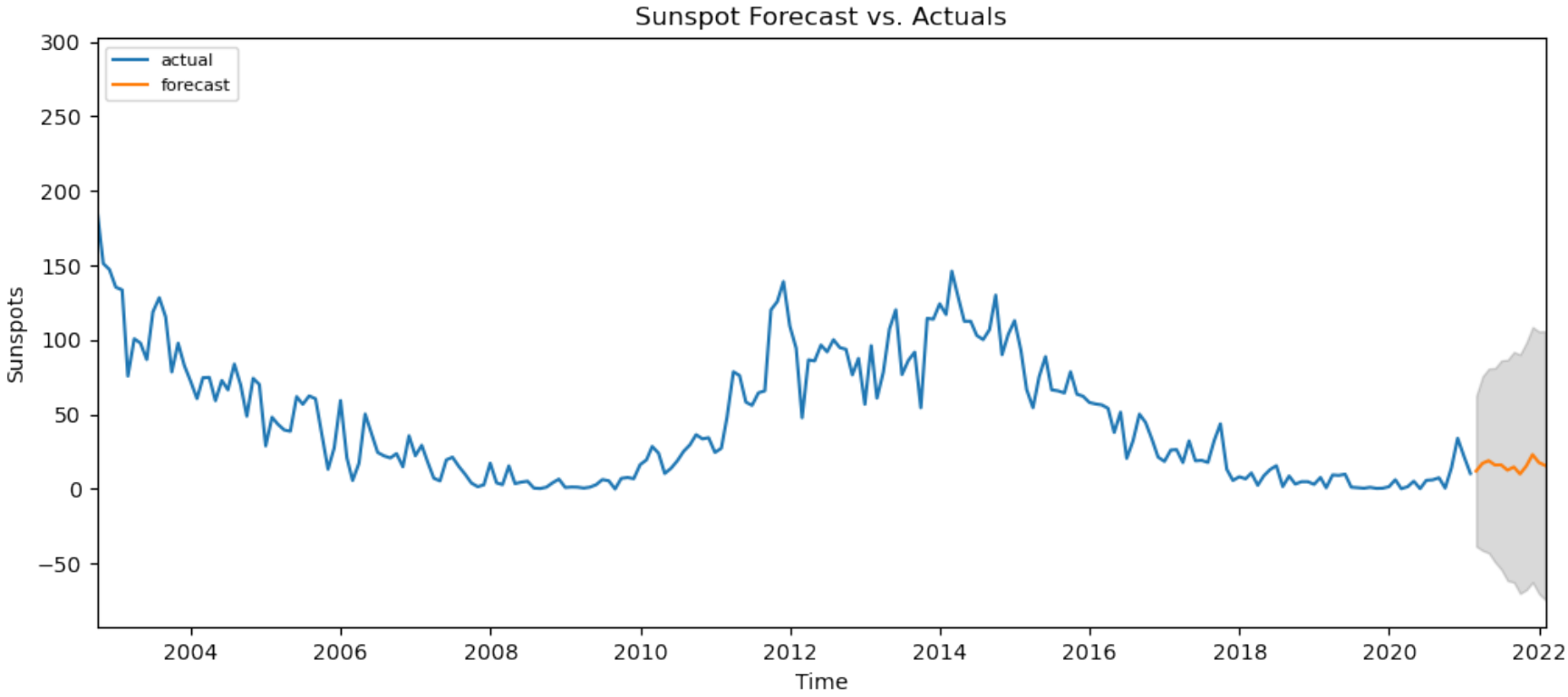
```
Text(0, 0.5, 'Empirical Fluctuation Process')
```



There appear to be no structural breaks in our plot.

12-step ahead forecast with custom model

In [37]:
```
# Forecast
fc = model_arima.forecast(12, alpha=0.05) # 95% conf

# Confidence
lower_series = model_arima.get_forecast(12).conf_int().iloc[:,0]
upper_series = model_arima.get_forecast(12).conf_int().iloc[:,1]

# Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(sunspot['Sunspots'], label='actual')
plt.plot(fc, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.15)
plt.title('Sunspot Forecast vs. Actuals')
# Set x-axis limits
start_index = sunspot.index[500]  # Index to start the plot
end_index = fc.index[-1]  # Last index of the data
plt.xlim(start_index, end_index)
plt.legend(loc='upper left', fontsize=8)
plt.xlabel('Time')
plt.ylabel('Sunspots')
plt.show()
```



Autoarima

In [113…

```
#!pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.3-cp39-cp39-macosx_10_9_x86_64.whl (608 kB)
     |████████████████████████████████| 608 kB 716 kB/s eta 0:00:01
Requirement already satisfied: scipy>=1.3.2 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-pa
ckages (from pmdarima) (1.7.3)
Requirement already satisfied: pandas>=0.19 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-pa
ckages (from pmdarima) (1.4.2)
Requirement already satisfied: scikit-learn>=0.22 in /Users/talialieberman/opt/anaconda3/lib/python3.9/s
ite-packages (from pmdarima) (1.0.2)
Requirement already satisfied: joblib>=0.11 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-pa
ckages (from pmdarima) (1.1.0)
Requirement already satisfied: statsmodels>=0.13.2 in /Users/talialieberman/opt/anaconda3/lib/python3.9/
site-packages (from pmdarima) (0.13.2)
Requirement already satisfied: urllib3 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-package
s (from pmdarima) (1.26.9)
Requirement already satisfied: numpy>=1.21.2 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-p
ackages (from pmdarima) (1.21.6)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /Users/talialieberman/opt/anaconda3/l
ib/python3.9/site-packages (from pmdarima) (0.29.28)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /Users/talialieberman/opt/anaconda3/lib/py
thon3.9/site-packages (from pmdarima) (61.2.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/talialieberman/opt/anaconda3/lib/python3
.9/site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-pa
ckages (from pandas>=0.19->pmdarima) (2022.7.1)
Requirement already satisfied: six>=1.5 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-packag
es (from python-dateutil>=2.8.1->pandas>=0.19->pmdarima) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/talialieberman/opt/anaconda3/lib/python3.9
/site-packages (from scikit-learn>=0.22->pmdarima) (2.2.0)
Requirement already satisfied: patsy>=0.5.2 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site-pa
ckages (from statsmodels>=0.13.2->pmdarima) (0.5.2)
Requirement already satisfied: packaging>=21.3 in /Users/talialieberman/opt/anaconda3/lib/python3.9/site
-packages (from statsmodels>=0.13.2->pmdarima) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/talialieberman/opt/anaconda3/lib/pytho
n3.9/site-packages (from packaging>=21.3->statsmodels>=0.13.2->pmdarima) (3.0.4)
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.3
```

In [29]:
```
sunspot['Sunspots'][:-120]
```

Out[29]:
```
Date
1961-01-31    82.1
1961-02-28    65.4
1961-03-31    75.2
1961-04-30    86.9
1961-05-31    72.3
              ...
2010-09-30    36.4
2010-10-31    33.6
2010-11-30    34.4
2010-12-31    24.5
2011-01-31    27.3
Name: Sunspots, Length: 601, dtype: float64
```

In [14]:
```
#Autoarima ----------------------------
import pmdarima as pm

# Seasonal - fit stepwise auto-ARIMA
#Train model excluding last 120 observations (10 years)
smodel = pm.auto_arima(sunspot['Sunspots'][:-120], start_p=1, start_q=1,
                       test='adf',
                       max_p=4, max_q=3, m=12,
                       start_P=0, seasonal=True,
                       d=None, D=1, trace=True,
                       error_action='ignore',
                       suppress_warnings=True,
                       stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=2.16 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=6287.632, Time=0.04 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=5704.750, Time=0.57 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=6007.788, Time=0.97 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=6285.883, Time=0.04 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=5810.668, Time=0.14 sec
 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=5662.865, Time=2.44 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=7.97 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.80 sec
 ARIMA(0,0,0)(2,1,0)[12] intercept   : AIC=6258.808, Time=1.73 sec
 ARIMA(2,0,0)(2,1,0)[12] intercept   : AIC=5629.772, Time=6.02 sec
 ARIMA(2,0,0)(1,1,0)[12] intercept   : AIC=5679.091, Time=0.84 sec
 ARIMA(2,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=7.06 sec
 ARIMA(2,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.68 sec
 ARIMA(3,0,0)(2,1,0)[12] intercept   : AIC=5589.368, Time=8.74 sec
 ARIMA(3,0,0)(1,1,0)[12] intercept   : AIC=5634.931, Time=1.48 sec
 ARIMA(3,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=13.71 sec
 ARIMA(3,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=2.85 sec
 ARIMA(4,0,0)(2,1,0)[12] intercept   : AIC=5571.864, Time=10.70 sec
 ARIMA(4,0,0)(1,1,0)[12] intercept   : AIC=5611.918, Time=1.94 sec
 ARIMA(4,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=6.71 sec
 ARIMA(4,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=3.92 sec
 ARIMA(4,0,1)(2,1,0)[12] intercept   : AIC=5572.560, Time=11.47 sec
 ARIMA(3,0,1)(2,1,0)[12] intercept   : AIC=5573.648, Time=11.65 sec
 ARIMA(4,0,0)(2,1,0)[12]             : AIC=5569.880, Time=1.99 sec
 ARIMA(4,0,0)(1,1,0)[12]             : AIC=5609.930, Time=0.71 sec
 ARIMA(4,0,0)(2,1,1)[12]             : AIC=inf, Time=5.32 sec
 ARIMA(4,0,0)(1,1,1)[12]             : AIC=inf, Time=2.76 sec
 ARIMA(3,0,0)(2,1,0)[12]             : AIC=5587.386, Time=1.59 sec
 ARIMA(4,0,1)(2,1,0)[12]             : AIC=5570.575, Time=2.40 sec
 ARIMA(3,0,1)(2,1,0)[12]             : AIC=5571.663, Time=2.39 sec

Best model:  ARIMA(4,0,0)(2,1,0)[12]
Total fit time: 123.824 seconds
```

The best model selected by the autoARIMA procedure is ARIMA(4, 0, 0)(2, 1, 0)[12]. This means it has an autoregressive order of 4, no differencing (d=0), no moving average terms (q=0), two seasonal autoregressive terms (P=2), one seasonal differencing (D=1), no seasonal moving average terms (Q=0), and a seasonal period of 12 (monthly data with yearly seasonality).

In [15]:
```
# Predict and Format for autoarima
auto_cast = smodel.predict(120)
```

In [16]:
```
auto_cast
```

Out[16]:
```
2011-02-28    33.167328
2011-03-31    33.768596
2011-04-30    22.880735
2011-05-31    24.998233
2011-06-30    28.437375
                 ...
2020-09-30    54.197560
2020-10-31    52.579789
2020-11-30    52.343915
2020-12-31    48.455596
2021-01-31    50.297469
Freq: M, Length: 120, dtype: float64
```

In [17]:
```
auto_cast = pd.Series(auto_cast)
```

In [18]:
```
sunspot_ets = sunspot['Sunspots'].resample('M').last()
```

Holt-Winters Forecast

In [19]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
hw_model = HWES(sunspot_ets[:-120], seasonal_periods=132, trend='add', seasonal='add')
hw_fitted = hw_model.fit()

hw_cast = hw_fitted.forecast(120)
```

ETS Forecast

In [20]:
```
import statsmodels.api as sm
ets_model=sm.tsa.statespace.ExponentialSmoothing(sunspot_ets[:-120],
                                                 trend=True,
                                                 freq='M',
                                                 initialization_method= 'heuristic',
                                                 seasonal=132,
                                                 damped_trend=False).fit()

ets_cast = ets_model.forecast(120)
ets_cast.index = pd.Series(ets_cast.index).apply(lambda dt: dt.replace(day=1))
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/statespace/exponential_s
moothing.py:161: RuntimeWarning: ExponentialSmoothing should not be used with seasonal terms. It has a s
erious bug that has not been fixed. Instead use ETSModel.
  warnings.warn(
 This problem is unconstrained.
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            3     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.65150D+00    |proj g|=  1.87434D-01

At iterate    5    f=  4.52361D+00    |proj g|=  7.56788D-03

At iterate   10    f=  4.51395D+00    |proj g|=  2.46658D-03

At iterate   15    f=  4.51368D+00    |proj g|=  4.93082D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    3     16     17      1     0     0   5.915D-06   4.514D+00
  F =   4.5136698156373756

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

In [22]:
```
from statsmodels.tsa.arima.model import ARIMA
```

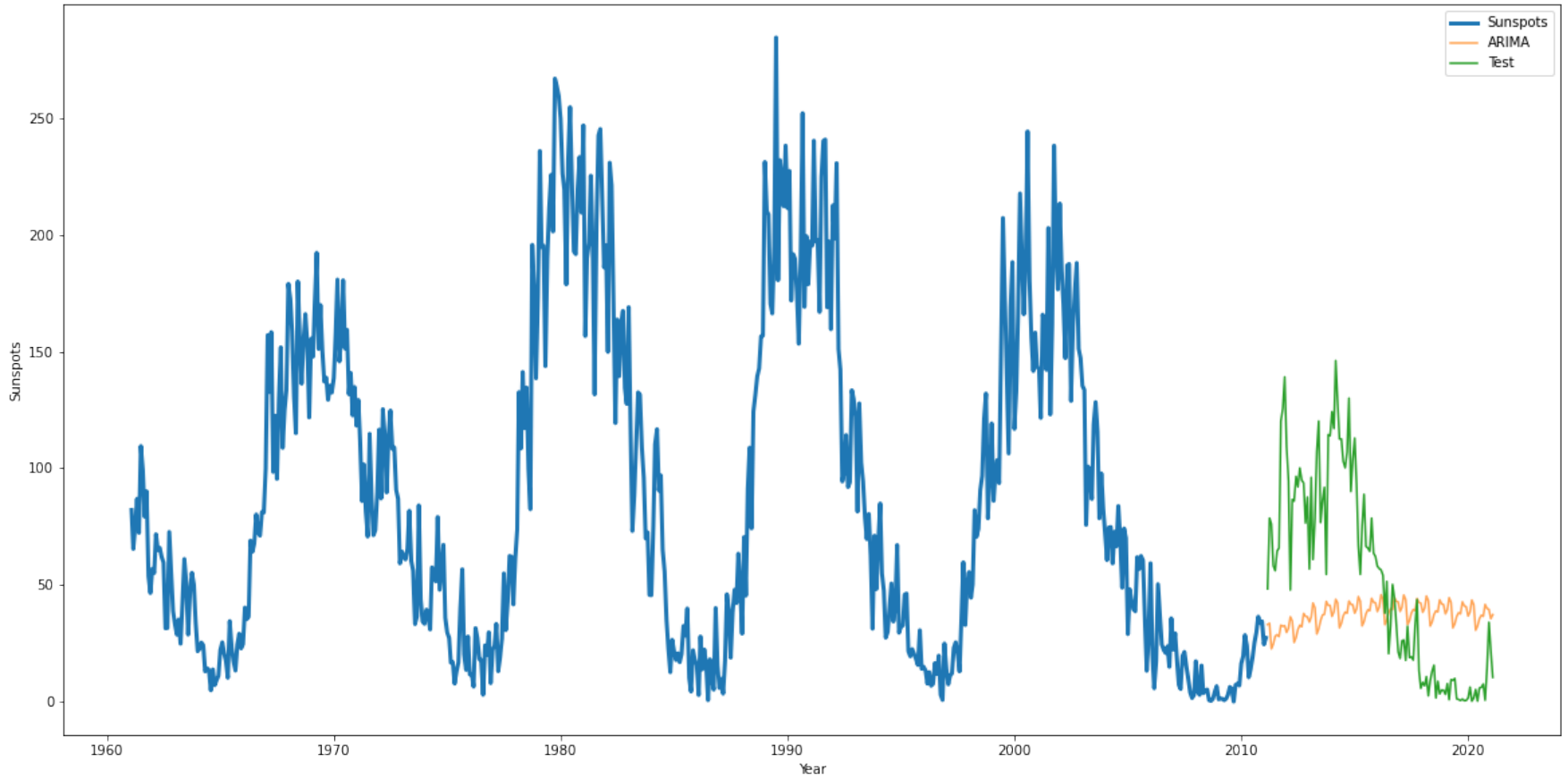
In [24]:
```
#custom ARIMA model

custom_model = ARIMA(sunspot['Sunspots'][:-120], order=(4, 0, 0), seasonal_order = (2, 1, 0, 12),
              trend = (0, 1, 0)).fit()


custom_cast = custom_model.forecast(120)
```

```
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/Users/talialieberman/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:471: V
alueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

In [25]:
```
custom_cast = custom_model.forecast(120)
```

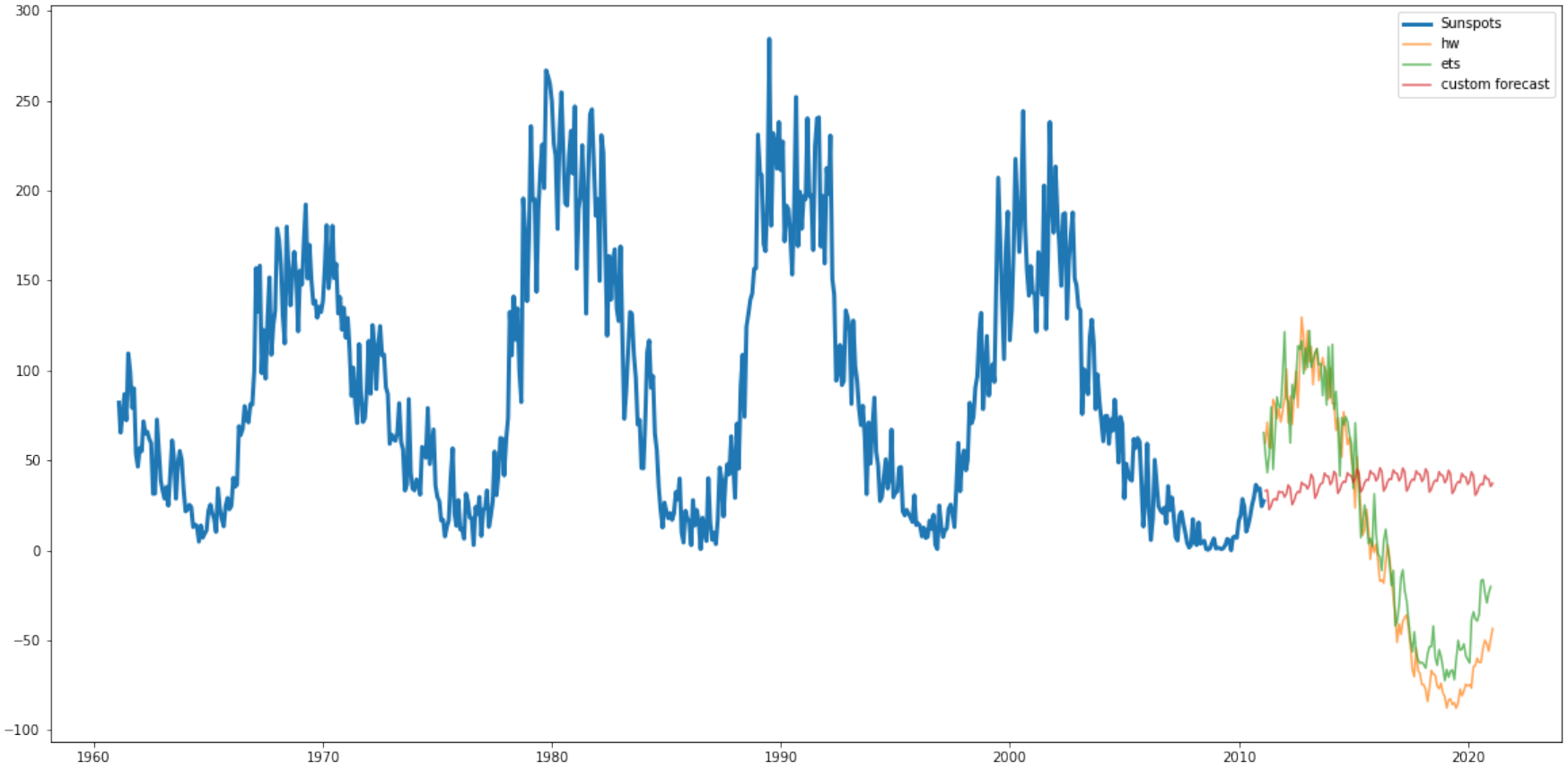
In [26]:
```
# Plot custom ARIMA model
plt.figure(figsize = (20,10))
plt.plot(sunspot['Sunspots'][:-120], linewidth=3, label='Sunspots')
plt.plot(custom_cast, alpha = 0.7, label='ARIMA')
plt.plot(sunspot['Sunspots'][-120:], label = 'Test')
plt.legend()
plt.xlabel('Year')
plt.ylabel('Sunspots')
```

Out[26]:
```
Text(0, 0.5, 'Sunspots')
```



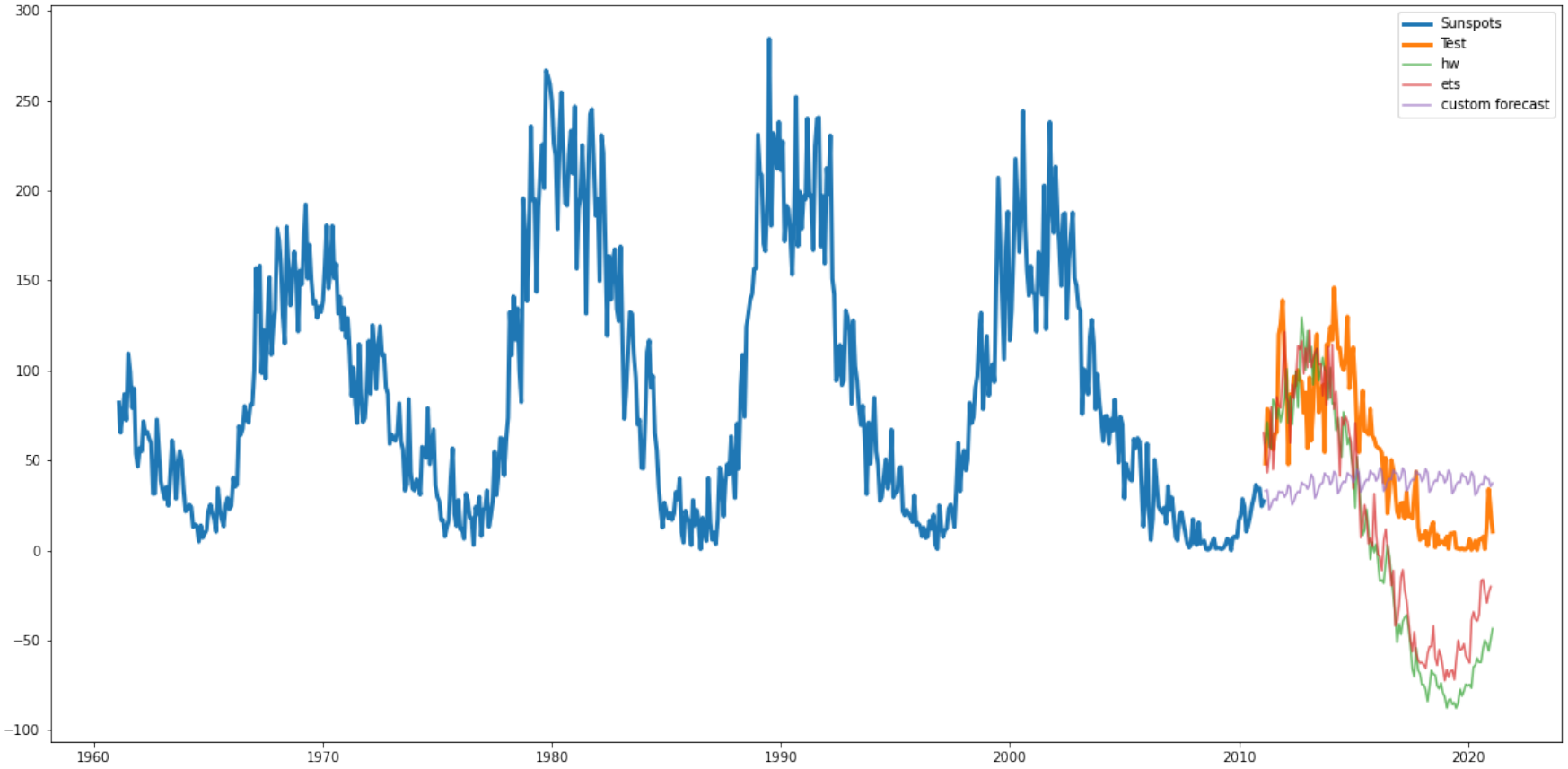
In [28]:
```
# Plot ETS and Holt Winters Models along with custom forecast

plt.figure(figsize = (20,10))
plt.plot(sunspot['Sunspots'][:-120], linewidth=3, label='Sunspots')
plt.plot(hw_cast[-120:], alpha = 0.7, label='hw')
plt.plot(ets_cast[-120:], alpha = 0.7, label='ets')
plt.plot(custom_cast, alpha = 0.7, label='custom forecast')
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x7f9e96f3e070>
```



In [33]:
```
plt.figure(figsize = (20,10))
# Plot against testing data
plt.plot(sunspot['Sunspots'][:-120], linewidth=3, label='Sunspots')
plt.plot(sunspot['Sunspots'][-120:], linewidth=3, label='Test')
plt.plot(hw_cast[-120:], alpha = 0.7, label='hw')
plt.plot(ets_cast[-120:], alpha = 0.7, label='ets')
plt.plot(custom_cast, alpha = 0.7, label='custom forecast')
plt.legend()
```

Out[33]:
```
<matplotlib.legend.Legend at 0x7f9dd1aa2940>
```



In [3]:
```
##Compare MAPES
real = sunspot['Sunspots'][-120:]

hw_mape     = np.mean( np.abs(real - hw_cast) / real )*100
ets_mape    = np.mean( np.abs(real - ets_cast) / real )*100
auto_mape   = np.mean( np.abs(real - auto_cast) / real )*100
custom_mape = np.mean( np.abs(real - custom_cast) / real )*100

print('MAPES: ')
print('HW: ', round(hw_mape, 4), 'ETS: ', round(ets_mape, 4))
print('AutoARIMA: ', round(auto_mape, 4), 'Custom Model: ', round(custom_mape,4))
```

```
MAPES:
HW:  3113.9068 ETS:  1607.5896
AutoARIMA:  1188.03 Custom Model:  923.5365
```

We see that our custom-fit ARIMA model performs the best out of these 4 models. Holt-winters and ETS have very high MAPEs relatively.

Prophet Model

In [50]:
```
from fbprophet import Prophet
```

In [51]:
```
# Prophet package requires the dataframe to be in a particular format;
# Importing the data again
proph_df = pd.read_csv('Sunspots-2.csv', index_col = 0)
proph_df['Date'] = pd.to_datetime(proph_df['Date'])

# Subset data to observe last 60 years (12 months * 60 (+1 to start in January))
proph_df = proph_df.tail(721)

# Renaming columns to conform to Prophet's requirements
proph_df = proph_df.rename(columns={'Date': 'ds',
                        'Monthly Mean Total Sunspot Number': 'y'})
```

In [52]:
```
# Specifying the model
prophet_model = Prophet(growth='linear', changepoints=None)

# Adding custom seasonality: "decadely" to capture the 11-year cycles
prophet_model.add_seasonality(name='decadely',
                        period=30*12*11,
                        fourier_order=5,
                        mode='multiplicative')
```

Out[52]:
```
<fbprophet.forecaster.Prophet at 0x7fb3044b79d0>
```

In [53]:
```
# fitting the model (leaving last 120 observations for testing)
prophet_model.fit(proph_df[:-120])
```

```
INFO:numexpr.utils:NumExpr defaulting to 8 threads.
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.
Initial log joint probability = -24.7297
```

Out[53]:
```
<fbprophet.forecaster.Prophet at 0x7fb3044b79d0>
```

```
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
      99       1073.76    0.00410436       83.1269           1           1      117
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
     199       1075.04   0.000142848         69.903        0.77        0.77      249
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
     299         1075.3   3.64073e-07       65.8339           1           1      378
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
     304         1075.3   8.16205e-09       64.1781     0.04919     0.04919      386
Optimization terminated normally:
  Convergence detected: absolute parameter change was below tolerance
```

In [54]:
```
from pandas.tseries.offsets import MonthEnd

# Making dataframe for forecasts
future = prophet_model.make_future_dataframe(periods = 120, freq = 'MS')

# Fixing dates
future['ds'] = future['ds'] + MonthEnd(0)
```

In [55]:
```
# Making forecasts
forecast = prophet_model.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

Out[55]:

	ds	yhat	yhat_lower	yhat_upper
0	1961-01-31	68.963341	37.646322	104.059368
1	1961-02-28	75.345629	41.810177	110.044192
2	1961-03-31	72.051783	36.007757	107.396845
3	1961-04-30	69.167848	32.531792	105.424003
4	1961-05-31	68.801541	32.413394	105.058149
...	...	...	...	...
716	2020-09-30	-18.449953	-55.232886	15.856390
717	2020-10-31	-24.598603	-58.869077	13.150108
718	2020-11-30	-33.918350	-71.082366	0.373996
719	2020-12-31	-28.815838	-67.036376	6.577541
720	2021-01-31	-41.167882	-75.917289	-4.362819

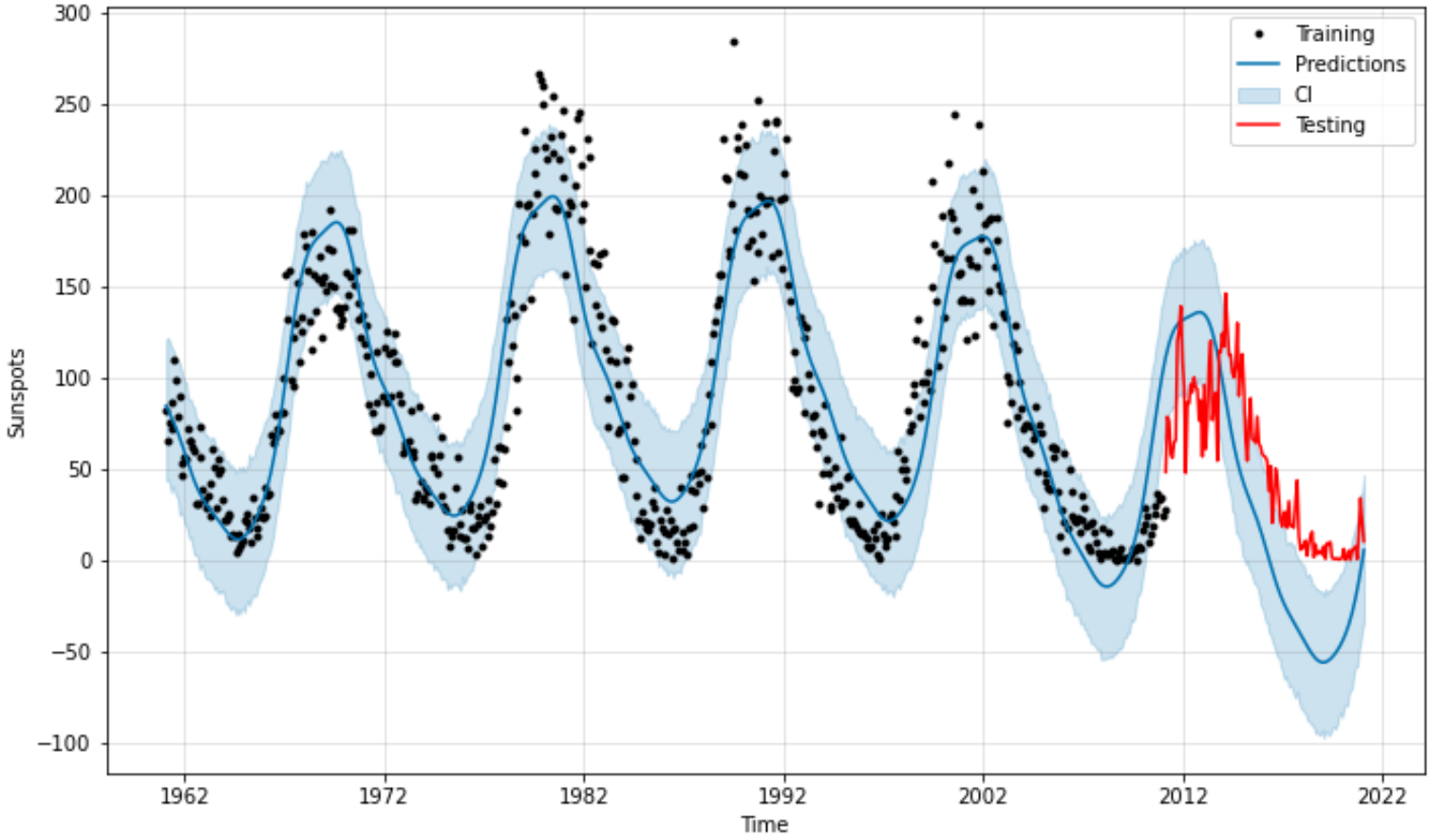
721 rows × 4 columns

In [56]:
```
# Fitted values
proph_fit = forecast[:-120][["ds","yhat"]]
proph_fit.set_index("ds", inplace = True)
```

In [57]:
```
# Forecasts over testing set
proph_fore = forecast[-120:][["ds","yhat"]]
proph_fore.set_index("ds", inplace = True)
```

In [237…

```
# Plotting actual observations, forecasts and fitted values

prophet_model.plot(forecast);
plt.plot(df.iloc[-120:], color = "red")
plt.legend(["Training", "Predictions", "CI", "Testing"])
plt.xlabel("Time")
plt.ylabel("Sunspots")
plt.savefig('prophet.png', dpi=300)
```



In [64]:
```
# Evaluating the model: calculating MAPE
mape_proph_fore = np.mean(np.abs((test["Sunspots"]) - proph_fore["yhat"])/test["Sunspots"])*100

# Print the MAPE
print("Prophet MAPE:", round(mape_proph_fore,2), "%")
```

```
 Prophet MAPE: 342.31 %
```

The Prophet model is highly customizable. We specify "decadely" seasonality with a multiplicative mode while fitting the model to try and capture the 11-year cyclicity in the data. We use a "multiplicative" mode since the amplitude of the cycles (modelled here as seasonality) changes over time. After several attempts, the fourier order that works out the best is 5.

From the plot and from the MAPE, we can see that this model performs significantly better than the others.

NNAR Model

In [174…
```
import rpy2

import warnings
warnings.filterwarnings('ignore')

from rpy2.robjects import pandas2ri
import rpy2.rinterface as rinterface
pandas2ri.activate()

%load_ext rpy2.ipython
```

```
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

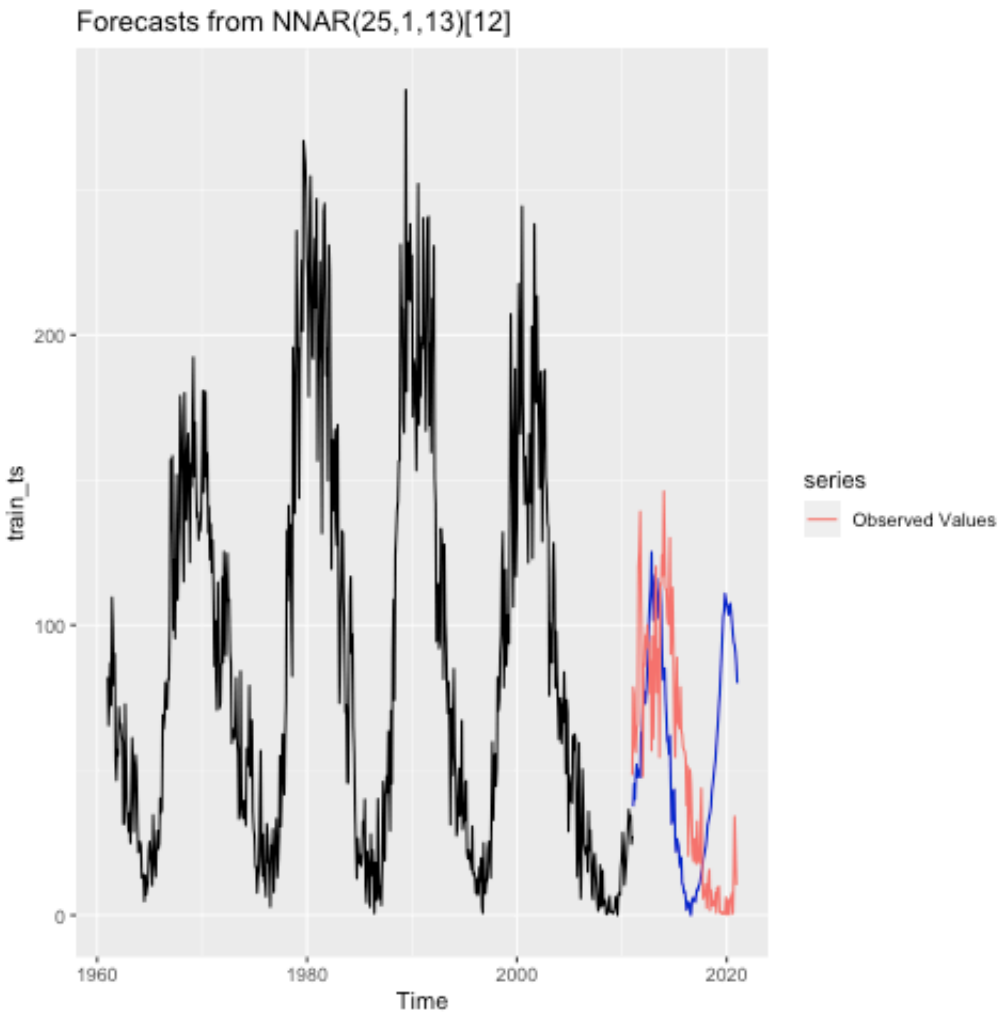
In [ ]:
```
%R -i df
%R -i train
%R -i test
```

In [177…
```
%%R
library(forecast)
library(ggplot2)
library(nnet)

# Converting data to time series format in R
train_ts <- ts(train, frequency = 12, start = 1961)
test_ts <- ts(test, frequency = 12, start = 2011)
```

In [189…
```
%%R

# Fitting the model
fit <- nnetar(train_ts)
fit
```

```
Series: train_ts
Model:  NNAR(25,1,13)[12]
Call:   nnetar(y = train_ts)

Average of 20 networks, each of which is
a 25-13-1 network with 352 weights
options were - linear output units

sigma^2 estimated as 38.26
```

In [277…
```
%%R

# Making forecasts and plotting
forecast <- forecast(fit, h = 120)
autoplot(forecast) + autolayer(test_ts, series="Observed Values", PI=FALSE)
```



In [ ]:
```
%%R
# Saving forecasted values
write.csv(forecast, "NNAR_fore.csv")
```

In [191…
```
%%R
# Extracting fitted values
nnar_fitted <- fitted(fit)
```

In [ ]:
```
%R -o nnar_fitted # Transferring data to python
```

In [279…
```
# Calling data in python
nnar_fit = pd.DataFrame(nnar_fitted, index = train.index)
nnar_fore = pd.read_csv("NNAR_fore.csv")
```

In [193…
```
# Fixing forecasted values format
melted_df = pd.melt(nnar_fore, id_vars="Unnamed: 0", var_name="Month", value_name="Value")
melted_df.rename(columns={melted_df.columns[0]: "Year"}, inplace=True)
melted_df["Date"] = pd.to_datetime(melted_df["Year"].astype(str) + "-" + melted_df["Month"], format="%Y-%
melted_df.set_index("Date", inplace=True)
melted_df.drop(["Year", "Month"], axis=1, inplace=True)
melted_df.sort_index(inplace=True)

# Convert the DataFrame to a time series
nnar_fore = pd.Series(melted_df["Value"].values, index=melted_df.index)
```

In [195…
```
nnar_fore.dropna(inplace = True)
nnar_fore.index = test["Sunspots"].index
```

In [196…
```
nnar_fore
```

Out[196]:
```
Date
2011-02-28    35.056158
2011-03-31    41.998224
2011-04-30    38.796568
2011-05-31    46.901243
2011-06-30    47.001880
                 ...
2020-09-30    28.362897
2020-10-31    25.106872
2020-11-30    22.797485
2020-12-31    20.530885
2021-01-31    17.986854
Length: 120, dtype: float64
```

In [195…]:
```
nnar_fore.dropna(inplace = True)
nnar_fore.index = test["Sunspots"].index
```

In [196…]:
```
nnar_fore
```

Out[196]:
```
Date
2011-02-28    35.056158
2011-03-31    41.998224
2011-04-30    38.796568
2011-05-31    46.901243
2011-06-30    47.001880
                ...
2020-09-30    28.362897
2020-10-31    25.106872
2020-11-30    22.797485
2020-12-31    20.530885
2021-01-31    17.986854
Length: 120, dtype: float64
```

In [65]:
```
# Evaluating the model: Calculating MAPE
mape_nnar_fore = np.mean(np.abs((test["Sunspots"] - nnar_fore)/test["Sunspots"]))*100

# Print the MAPE
print("NNAR MAPE:", round(mape_nnar_fore,2), "%")
```

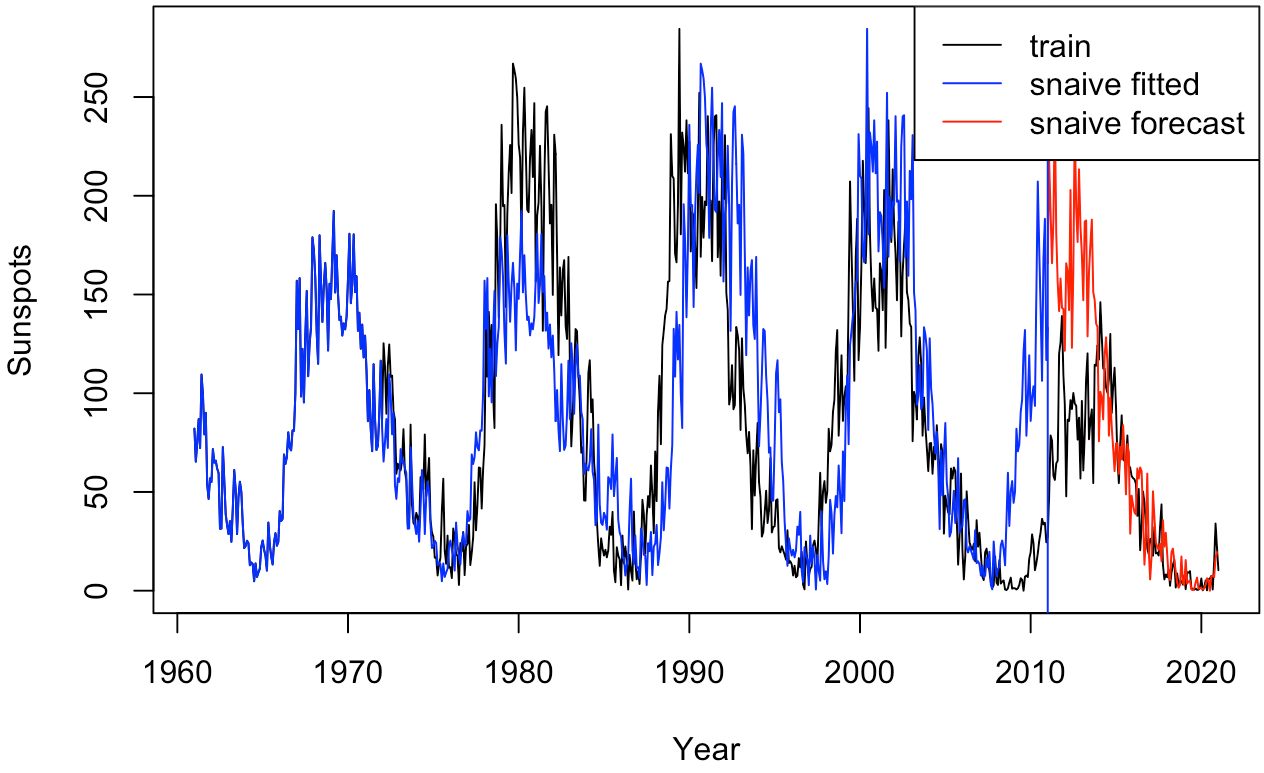
```
NNAR MAPE: 1308.23 %
```

NNAR's MAPE of 1308.23% is worse than the Random Walk's. This model doesn't perform well for our data.

Seasonal Naive **Modified (Bonus Model 1)**

Seasonal Naive models take the last season's value as the projection for the next season. In our case, we manually fit a SARIMA with one difference at the 132nd month thusetreating the predictable cycle as a 'season'.

```
# Seasonal Naive Model
snaive_arima = arima(df_train[,2], order = c(0, 0, 0), seasonal = list(order = c(0,
1, 0), period = 132))
snaive_cast = predict(snaive_arima, 120) # predictions are first value
snaive_cast  = ts(data = snaive_cast$pred, start = c(2011,1), freq=12)

snaive_fitted = df_train[,2] - snaive_arima$residuals

# Plot
# autoplot(df[,3]) +
#  autolayer(snaive_cast)
plot(df[,3],
     type = "l",
     xlab = "Year",
     ylab = "Sunspots",
     main = 'Seasonal Naive Sunspot Forecast (0,0,0)(0,1,0)[132]')
lines(snaive_fitted,
      type = "l", col ='blue')
lines(snaive_cast,
      type = "l", col ='red')
abline(v=2011, col="blue")
legend("topright",                          # Add legend to plot
       c("train", "snaive fitted", 'snaive forecast'),
       lty = 1,
       col = c('black', 'blue', 'red'))
```




```
# Acc
accuracy(snaive_cast, df_test[,2])
```

```
##                ME     RMSE     MAE      MPE     MAPE      ACF1 Theil's U
## Test set -23.1375 54.03811 33.8025 -116.644 145.1068 0.7943385 0.5500038
```

```
accuracy(snaive_fitted, df_train[,2])
```

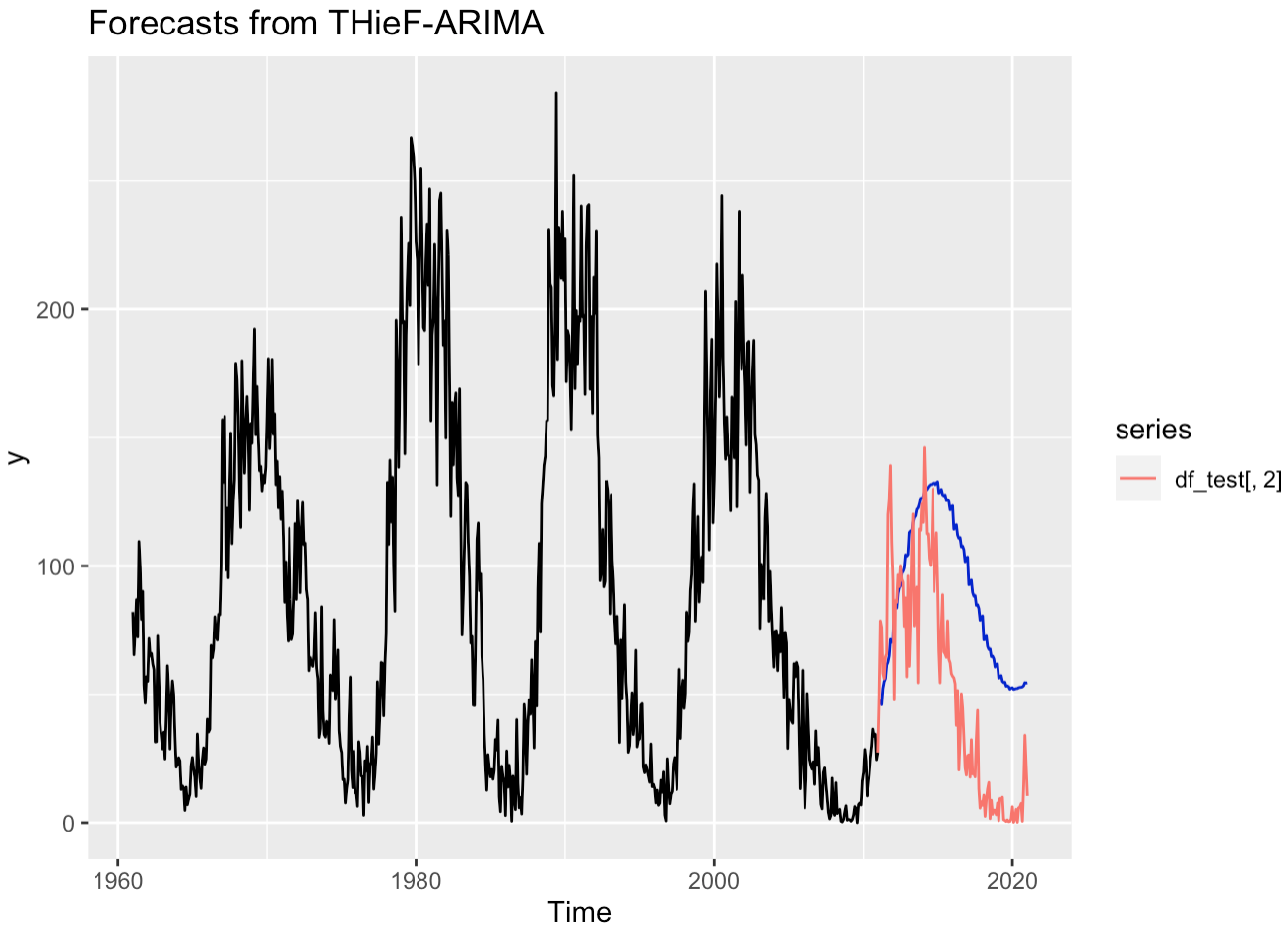
```
##                 ME    RMSE      MAE  MPE MAPE      ACF1 Theil's U
## Test set -3.339137 48.6073 32.14472 -Inf  Inf 0.7409168         0
```

The performance of SNAIVE is far better than many methods that fail to capture the cycle. If we projected a shorter time horizon (rather than 120 mos), other models might perform better.

Thief

THIeF is an acronym for temporal hierarchical forecasting. Thief computes increasingly higher level temporal aggregates, forecasts at each aggregation, then reconciles the forecasts.

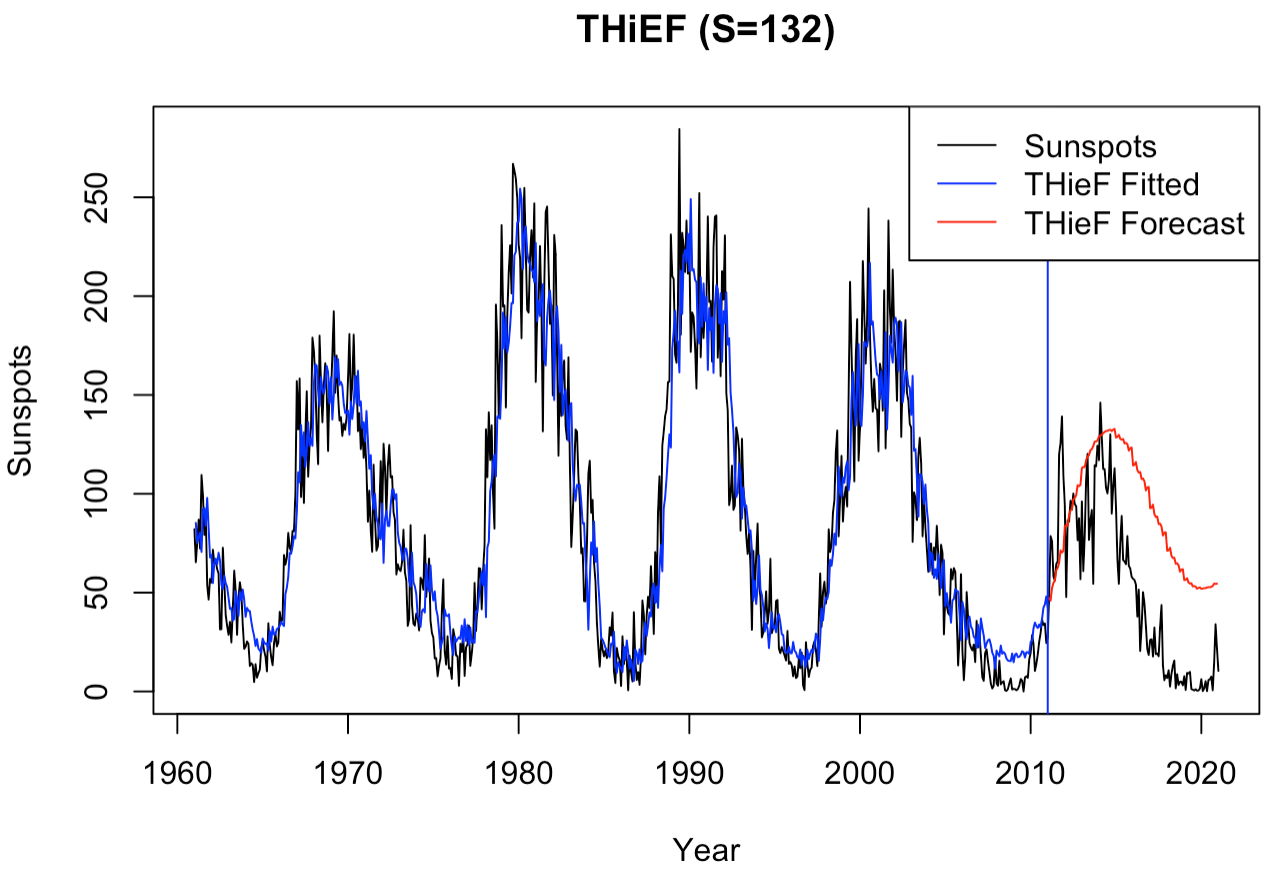
```
library(forecast)
library(thief)


# Convert for thief to univariate
dft = df_train[,2]  # third col has real vals

# Forecast periods
h_ = 120  # forecast horizon
m_ = 12 # seasonal period

fc <- thief(dft, m = m_, h=h_,  comb = 'struc', usemodel='arima')
autoplot(fc) +  autolayer(df_test[,2])
```




```
# Save thief forecast
thief_cast = fc$mean

# Thief
accuracy(thief_cast, df_test[,2])
```

```
##                 ME     RMSE      MAE       MPE     MAPE      ACF1 Theil's U
## Test set -37.84991 49.94649 44.04551 -1304.898 1310.826 0.8512767  10.74413
```

```
thief_cast  = ts(data = thief_cast, start = c(2011,1), freq=12)
plot(df[,3],
     type = "l",
     xlab = "Year",
     ylab = "Sunspots",
     main = 'THiEF (S=132)')
lines(fc$fitted,
      type = "l", col ='blue')
lines(thief_cast,
      type = "l", col ='red')
abline(v=2011, col="blue")
legend("topright",                              # Add legend to plot
       c('Sunspots', "THieF Fitted", 'THieF Forecast'),
       lty = 1,
       col = c('black', 'blue',  'red'))
```



Thief appears to produce fair estimated in sample and captures the cycle somewhat for out of sample prediction. However, its accuracy is not great.

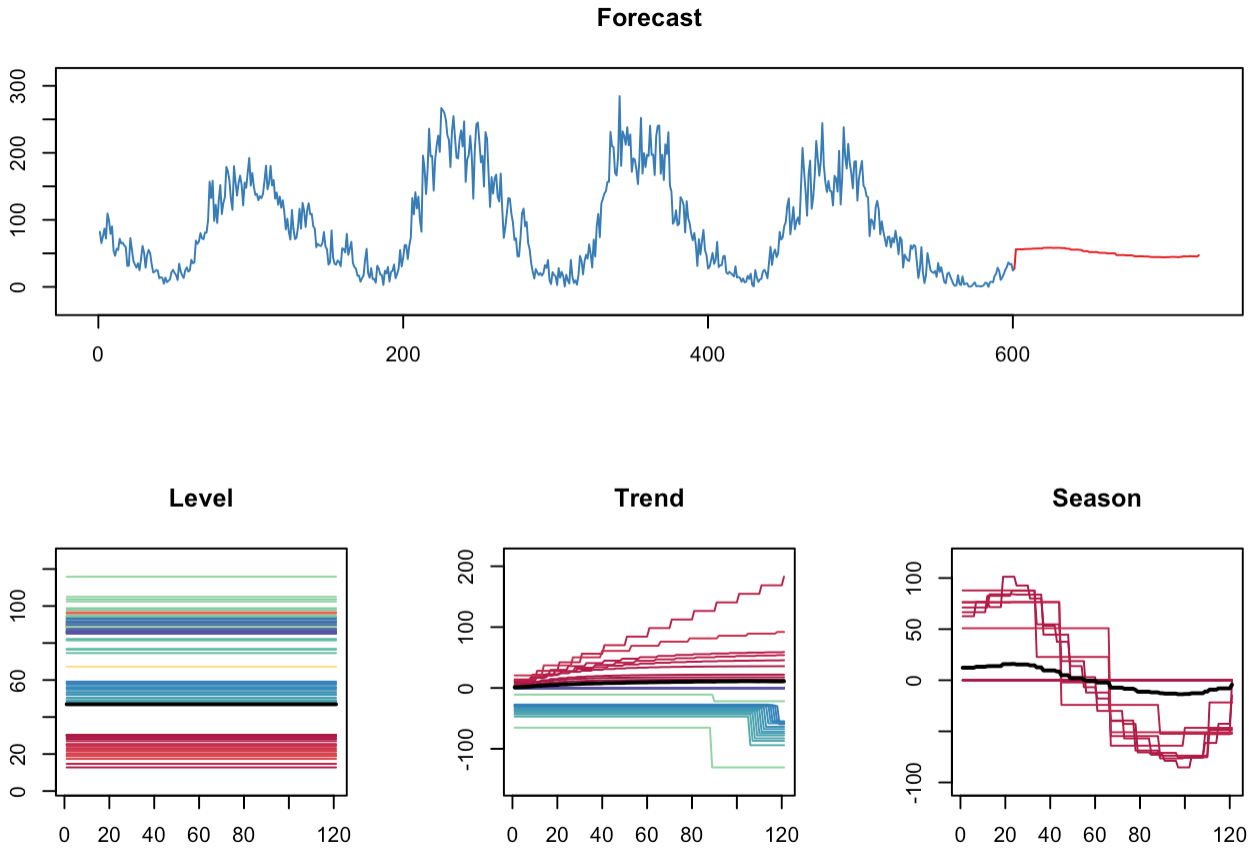
MAPA

```
# Seed for Repeatability
set.seed(1)
data = df_train[,2]

#Dynamic Fit to the data
mapafit       = mapaest(data, ppy = 132, outplot=2, paral=2)
```

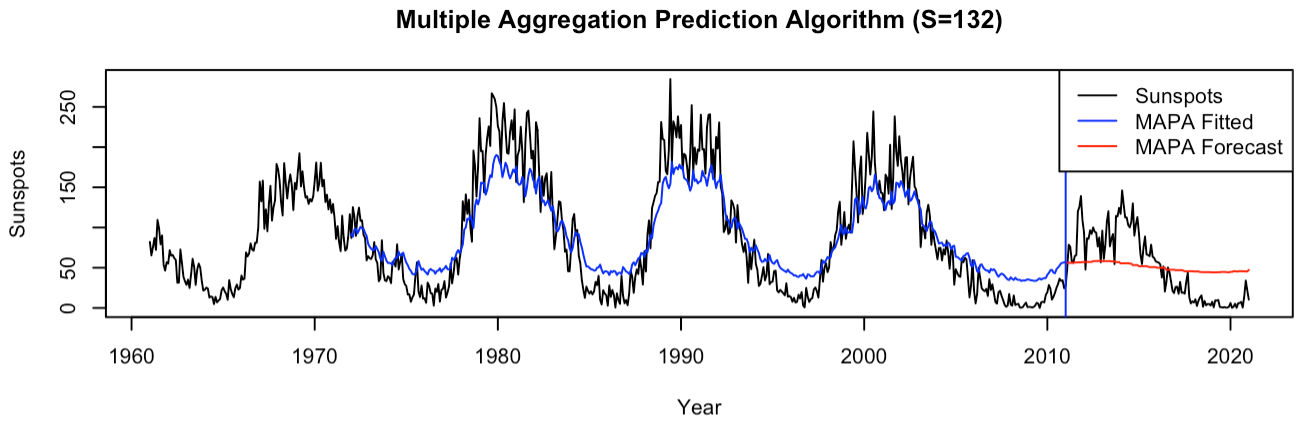
```
## Running with 10 cores
```

```
mapa_forecast_w_fit = mapafor(data,mapafit,fh=132)  # <<<< actual forecast

# Detailed view of the data at each temporal aggregation level (daily, monthly, quart
erly, annual)
mapa_forecast = mapasimple(data, ppy = 132, fh = 121, outplot=2,paral=2)
```

```
## Running with 10 cores
```



```
# Plot
mapa_cast  = ts(data = mapa_forecast$forecast, start = c(2011,1), freq=12)
mapa_fitted  = ts(data = c(mapa_forecast_w_fit$infor),  start = c(1961,1), freq=12)
plot(df[,3],
     type = "l",
     xlab = "Year",
     ylab = "Sunspots",
     main = 'Multiple Aggregation Prediction Algorithm (S=132)')
lines(mapa_cast,
      type = "l", col ='red')
lines(mapa_fitted,
      type = "l", col ='blue')
abline(v=2011, col="blue")
legend("topright",                              # Add legend to plot
       c('Sunspots', "MAPA Fitted", 'MAPA Forecast'),
       lty = 1,
       col = c('black', 'blue',  'red'))
```

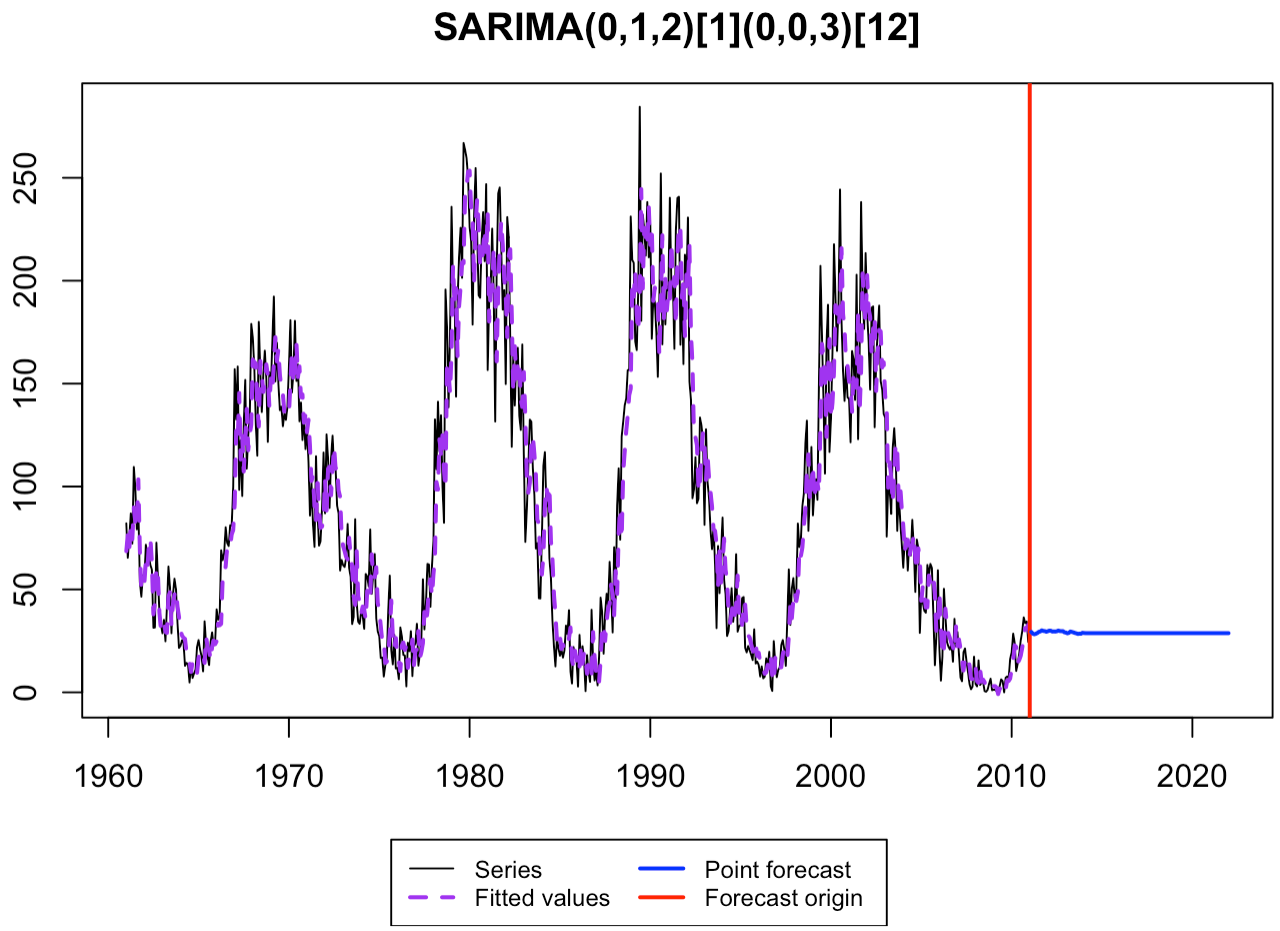


MAPA projections in sample don't capture the peakness of periods and project flatly out of sample. This may suggest that the longer run trend (no trend) is dominating the forecast and flattening it.

State Space Arima

```
require(smooth)
ssarima_fit = auto.ssarima(df_train[,2], h=132, silent=FALSE)
```

```
## Estimation progress:     0%1%1%2%2%3%3%9%9%10%10%11%11%12%12%12%13%14%14%14%17%17%
18%19%19%20%20%22%22%23%25%26%27%28%34%50%67%83%100%... Done!
```




```
predict(ssarima_fit, 120)
```

```
##          Point forecast Lower bound (2.5%) Upper bound (97.5%)
## Feb 2011       29.35035          -16.20832            74.90901
## Mar 2011       28.54522          -24.33636            81.42680
## Apr 2011       28.21151          -28.74174            85.16476
## May 2011       28.44182          -32.31082            89.19446
## Jun 2011       28.80039          -35.52764            93.12841
## Jul 2011       29.38208          -38.33281            97.09696
## Aug 2011       29.58469          -41.35554           100.52493
## Sep 2011       30.09773          -43.92746           104.12292
## Oct 2011       29.82315          -47.16348           106.80977
## Nov 2011       29.74824          -50.09004           109.58652
## Dec 2011       29.45407          -53.13747           112.04561
## Jan 2012       29.68902          -55.56691           114.94495
## Feb 2012       29.96564          -58.65449           118.58577
## Mar 2012       29.91472          -61.62588           121.45531
## Apr 2012       29.54725          -64.73177           123.82626
## May 2012       29.58802          -67.35209           126.52813
## Jun 2012       29.54257          -69.98751           129.07265
## Jul 2012       29.67422          -72.38012           131.72857
## Aug 2012       29.99600          -74.52167           134.51366
## Sep 2012       29.77403          -77.15022           136.69828
## Oct 2012       29.71612          -79.56172           138.99396
## Nov 2012       29.73413          -81.84767           141.31594
## Dec 2012       29.17454          -84.66461           143.01369
## Jan 2013       29.03752          -87.01507           145.09011
## Feb 2013       28.67732          -89.69868           147.05333
## Mar 2013       28.91121          -91.68188           149.50429
## Apr 2013       29.52506          -93.22681           152.27693
## May 2013       29.34197          -95.53137           154.21530
## Jun 2013       29.12796          -97.83139           156.08732
## Jul 2013       28.84975         -100.16190           157.86141
## Aug 2013       28.64901         -102.38280           159.68082
## Sep 2013       28.38091         -104.64038           161.40220
## Oct 2013       28.50871         -106.47274           163.49016
## Nov 2013       28.47569         -108.43786           165.38924
## Dec 2013       28.91235         -109.90641           167.73111
## Jan 2014       28.82295         -111.87522           169.52112
## Feb 2014       28.75799         -113.48307           170.99905
## Mar 2014       28.75228         -115.13841           172.64296
## Apr 2014       28.75228         -116.80741           174.31196
## May 2014       28.75228         -118.45749           175.96205
## Jun 2014       28.75228         -120.08929           177.59384
## Jul 2014       28.75228         -121.70338           179.20793
## Aug 2014       28.75228         -123.30034           180.80490
## Sep 2014       28.75228         -124.88071           182.38526
## Oct 2014       28.75228         -126.44498           183.94953
## Nov 2014       28.75228         -127.99364           185.49819
## Dec 2014       28.75228         -129.52715           187.03170
## Jan 2015       28.75228         -131.04594           188.55049
## Feb 2015       28.75228         -132.55043           190.05499
## Mar 2015       28.75228         -134.04102           191.54558
```

```
## Apr 2015       28.75228        -135.51809          193.02264
## May 2015       28.75228        -136.98199          194.48654
## Jun 2015       28.75228        -138.43307          195.93763
## Jul 2015       28.75228        -139.87167          197.37622
## Aug 2015       28.75228        -141.29810          198.80265
## Sep 2015       28.75228        -142.71266          200.21721
## Oct 2015       28.75228        -144.11564          201.62020
## Nov 2015       28.75228        -145.50734          203.01189
## Dec 2015       28.75228        -146.88800          204.39255
## Jan 2016       28.75228        -148.25789          205.76245
## Feb 2016       28.75228        -149.61727          207.12182
## Mar 2016       28.75228        -150.96636          208.47092
## Apr 2016       28.75228        -152.30540          209.80996
## May 2016       28.75228        -153.63461          211.13917
## Jun 2016       28.75228        -154.95421          212.45876
## Jul 2016       28.75228        -156.26439          213.76894
## Aug 2016       28.75228        -157.56536          215.06991
## Sep 2016       28.75228        -158.85730          216.36186
## Oct 2016       28.75228        -160.14041          217.64497
## Nov 2016       28.75228        -161.41487          218.91942
## Dec 2016       28.75228        -162.68084          220.18539
## Jan 2017       28.75228        -163.93849          221.44304
## Feb 2017       28.75228        -165.18799          222.69254
## Mar 2017       28.75228        -166.42948          223.93404
## Apr 2017       28.75228        -167.66313          225.16769
## May 2017       28.75228        -168.88909          226.39364
## Jun 2017       28.75228        -170.10748          227.61203
## Jul 2017       28.75228        -171.31845          228.82301
## Aug 2017       28.75228        -172.52214          230.02669
## Sep 2017       28.75228        -173.71867          231.22322
## Oct 2017       28.75228        -174.90817          232.41273
## Nov 2017       28.75228        -176.09077          233.59532
## Dec 2017       28.75228        -177.26658          234.77113
## Jan 2018       28.75228        -178.43571          235.94026
## Feb 2018       28.75228        -179.59828          237.10284
## Mar 2018       28.75228        -180.75441          238.25896
## Apr 2018       28.75228        -181.90418          239.40874
## May 2018       28.75228        -183.04772          240.55227
## Jun 2018       28.75228        -184.18511          241.68967
## Jul 2018       28.75228        -185.31647          242.82102
## Aug 2018       28.75228        -186.44187          243.94642
## Sep 2018       28.75228        -187.56142          245.06597
## Oct 2018       28.75228        -188.67520          246.17976
## Nov 2018       28.75228        -189.78331          247.28787
## Dec 2018       28.75228        -190.88583          248.39038
## Jan 2019       28.75228        -191.98284          249.48739
## Feb 2019       28.75228        -193.07443          250.57898
## Mar 2019       28.75228        -194.16067          251.66522
## Apr 2019       28.75228        -195.24164          252.74619
## May 2019       28.75228        -196.31742          253.82197
## Jun 2019       28.75228        -197.38808          254.89264
```

```
## Jul 2019       28.75228         -198.45370           255.95826
## Aug 2019       28.75228         -199.51435           257.01890
## Sep 2019       28.75228         -200.57009           258.07464
## Oct 2019       28.75228         -201.62099           259.12554
## Nov 2019       28.75228         -202.66711           260.17167
## Dec 2019       28.75228         -203.70853           261.21309
## Jan 2020       28.75228         -204.74531           262.24986
## Feb 2020       28.75228         -205.77750           263.28205
## Mar 2020       28.75228         -206.80517           264.30972
## Apr 2020       28.75228         -207.82838           265.33293
## May 2020       28.75228         -208.84717           266.35173
## Jun 2020       28.75228         -209.86162           267.36618
## Jul 2020       28.75228         -210.87178           268.37633
## Aug 2020       28.75228         -211.87769           269.38225
## Sep 2020       28.75228         -212.87942           270.38397
## Oct 2020       28.75228         -213.87701           271.38156
## Nov 2020       28.75228         -214.87051           272.37507
## Dec 2020       28.75228         -215.85999           273.36454
## Jan 2021       28.75228         -216.84547           274.35002
```

Review Results

```
# Snaive
accuracy(snaive_cast, df_test[,2])
```

```
##                ME     RMSE     MAE      MPE     MAPE      ACF1 Theil's U
## Test set -23.1375 54.03811 33.8025 -116.644 145.1068 0.7943385 0.5500038
```

```
# Mapa
mapa_cast = c()
for (i in 1:121){
  mapa_cast = c(mapa_cast, mapa_forecast$forecast[[i]])
}
accuracy( mapa_cast , df_test[,2])
```

```
##                ME     RMSE      MAE       MPE     MAPE      ACF1 Theil's U
## Test set 1.198164 37.56026 32.55045 -1008.659 1042.087 0.8896103  9.111666
```

```
# Thief
accuracy(thief_cast, df_test[,2])
```

```
##                 ME     RMSE     MAE       MPE     MAPE      ACF1 Theil's U
## Test set -37.70907 48.95883 43.0957 -1297.736 1302.723 0.8483401  10.77155
```

Output Predicitons

```
# write.csv(thief_cast, 'thief_fc.csv')
# write.csv(mapa_cast, 'mapa_fc.csv')
# write.csv(snaive_cast, 'snaive_fc.csv')

# write.csv(fc$fitted, 'thief_fitted.csv')
# write.csv(t(mapa_forecast_w_fit$infor), 'mapa_fitted.csv')
# write.csv(snaive_fitted, 'snaive_fitted.csv')
```

Combinations

We combine our top 3 models - Prophet, Seasonal Naive and ARIMA, using 2 methods-

1. Combination using OLS
2. Combination using Mean of Forecasts

In [35]:
```
# Importing ARIMA forecasted and fitted values
arima_fit = pd.read_csv("arima_fit.csv", index_col = 0, parse_dates = True)
arima_fc = pd.read_csv("arima_fc.csv", index_col = 0, parse_dates = True)
```

In [49]:
```
# Importing Seasonal Naive forecasted and fitted values
sn_fc = pd.read_csv("sn_fore.csv", index_col = 0, parse_dates = True)
sn_fc = sn_fc["Point.Forecast"]
sn_fc.index = arima_fc.index
sn_fit = pd.DataFrame(sn_fitted, index = train.index)
```

In [60]:
```
# Making a dataframe for all fitted values
combine_train = pd.concat([train, sn_fit, proph_fit, arima_fit], axis = 1)
combine_train.columns = ["Sunspots", "Snaive", "Prophet", "ARIMA"]
combine_train.dropna(inplace = True)
```

In [61]:
```
# Making a dataframe for all the forecasted values
combine_test = pd.concat([test, sn_fore, proph_fore, arima_fc], axis = 1)
combine_test.columns = ["Sunspots", "Snaive", "Prophet", "ARIMA"]
```

Combination 1: OLS

OLS 1: Using all 3 models

In [213]:
```
combine_ols = smf.ols('Sunspots ~ Snaive + Prophet + ARIMA', data = combine_train).fit()
ols_results = combine_ols.summary()
ols_results
```

Out[213]:

OLS Regression Results

Dep. Variable:	Sunspots	**R-squared:**	0.892
Model:	OLS	**Adj. R-squared:**	0.891
Method:	Least Squares	**F-statistic:**	1607.
Date:	Tue, 06 Jun 2023	**Prob (F-statistic):**	6.09e-282
Time:	15:32:05	**Log-Likelihood:**	-2688.6
No. Observations:	589	**AIC:**	5385.
Df Residuals:	585	**BIC:**	5403.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3.1452	1.704	1.846	0.065	-0.202	6.492
Snaive	-0.0996	0.025	-3.990	0.000	-0.149	-0.051
Prophet	0.4578	0.039	11.657	0.000	0.381	0.535
ARIMA	0.6084	0.041	14.991	0.000	0.529	0.688

Omnibus:	31.521	**Durbin-Watson:**	1.790
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	54.508

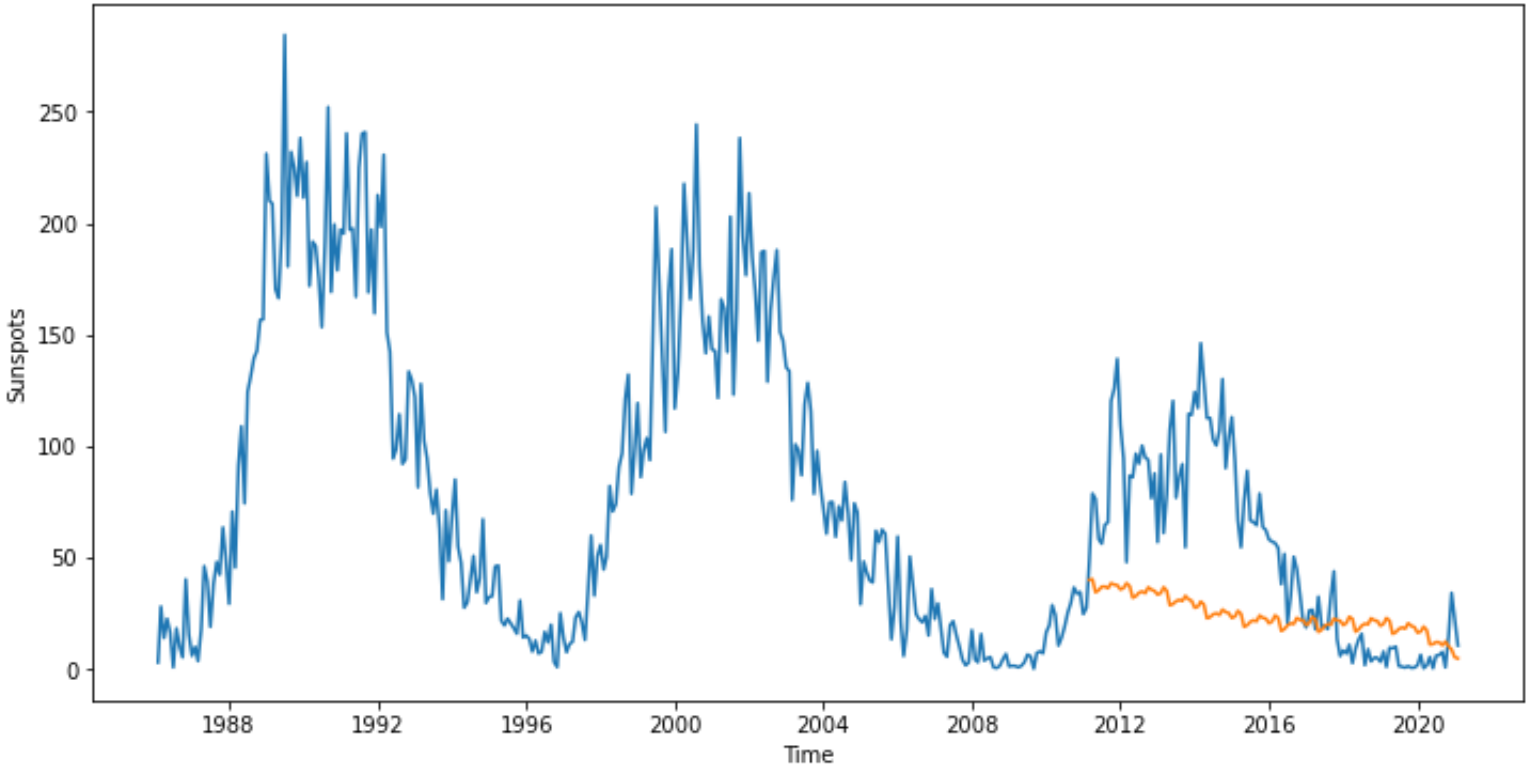
Skew:	0.375	**Prob(JB):**	1.46e-12
Kurtosis:	4.287	**Cond. No.**	340.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [216]:
```
# Making predictions
comb_ols_fore = combine_ols.predict(combine_test)
```

In [203]:
```
# Plotting observed values and predictions
plt.figure(figsize = (12,6))
plt.xlabel("Time")
plt.ylabel("Sunspots")
plt.plot(df[300:])
plt.plot(comb_ols_fore)
```

Out[203]: [<matplotlib.lines.Line2D at 0x7fe3add612e0>]



In [67]:
```
# Model Evaluation: Calculating the MAPE
mape_comb_ols_fore = np.mean(np.abs((combine_test["Sunspots"] - comb_ols_fore) /combine_test["Su

# Print the MAPE
print("MAPE:", round(mape_comb_ols_fore,2), "%")
```

```
OLS MAPE: 461.38 %
```

It's interesting to note that Seasonal Naive gets a negative weight in this OLS and the magnitude is also very close to 0. We fit another OLS model without the Seasonal Naive and see how it performs in comparison to OLS 1.

OLS 2: Using only Prophet and ARIMA

In [63]:
```
combine_ols2 = smf.ols('Sunspots ~ Prophet + ARIMA', data = combine_train).fit()
ols_results2 = combine_ols2.summary()
ols_results2
```

Out[63]:

OLS Regression Results

Dep. Variable:	Sunspots	**R-squared:**	0.890
Model:	OLS	**Adj. R-squared:**	0.889
Method:	Least Squares	**F-statistic:**	2362.
Date:	Thu, 08 Jun 2023	**Prob (F-statistic):**	3.66e-281
Time:	17:11:55	**Log-Likelihood:**	-2694.4
No. Observations:	589	**AIC:**	5395.
Df Residuals:	586	**BIC:**	5408.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.2526	1.654	0.757	0.449	-1.995	4.501
Prophet	0.4652	0.040	11.773	0.000	0.388	0.543
ARIMA	0.5218	0.035	14.810	0.000	0.453	0.591

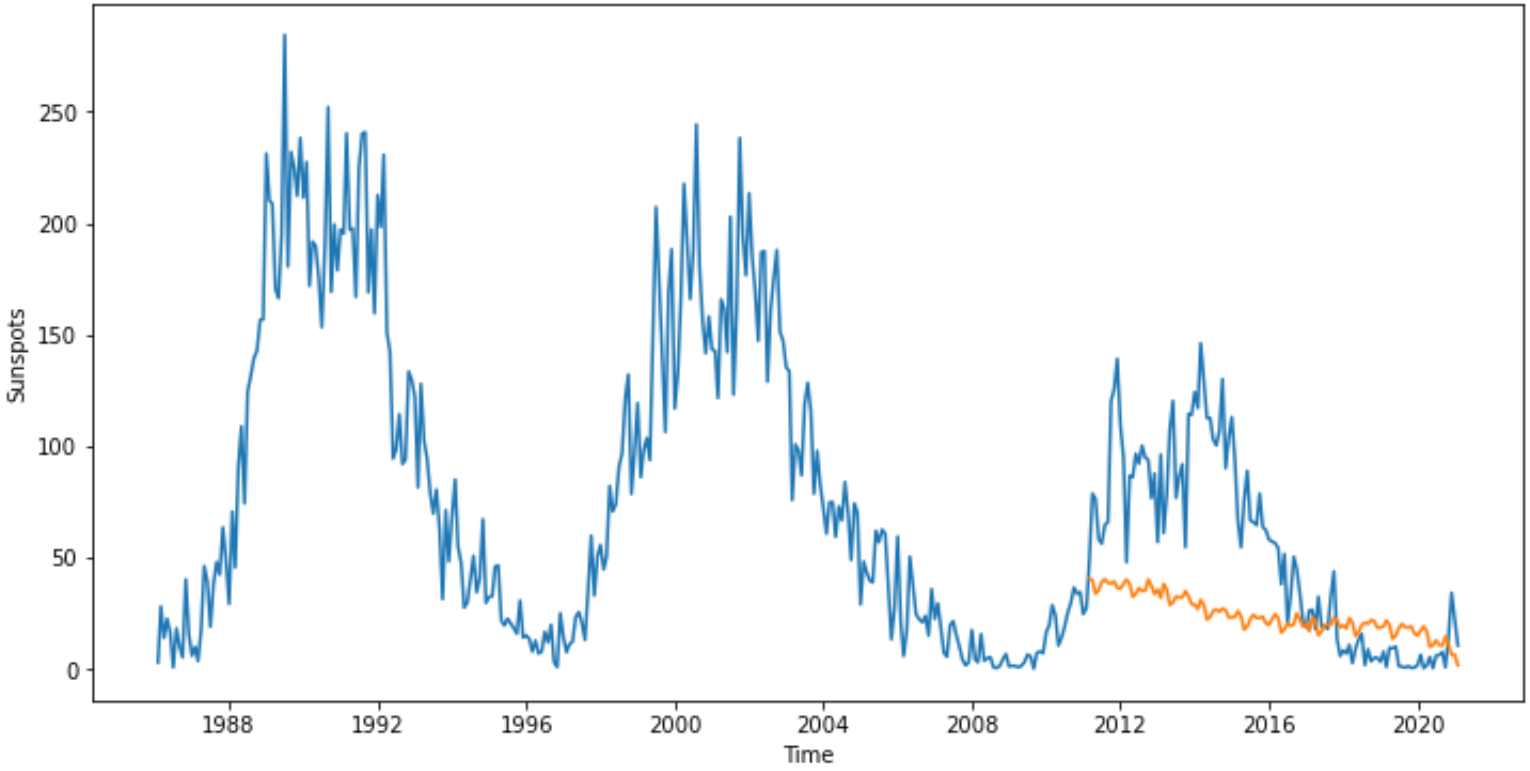
Omnibus:	33.932	**Durbin-Watson:**	1.658
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	61.406
Skew:	0.386	**Prob(JB):**	4.63e-14
Kurtosis:	4.380	**Cond. No.**	269.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [73]:
```
# Making predictions
comb_ols_fore2 = combine_ols2.predict(combine_test)
```

In [74]:
```
# Plotting observed values and predictions
plt.figure(figsize = (12,6))
plt.xlabel("Time")
plt.ylabel("Sunspots")
plt.plot(df[300:])
plt.plot(comb_ols_fore2)
```

Out[74]: [<matplotlib.lines.Line2D at 0x7fb2e0a92160>]



In [70]:
```
# MAPE
mape_comb_ols_fore2 = np.mean(np.abs((combine_test["Sunspots"] - comb_ols_fore2) /combine_test["S

# Print the MAPE
print("OLS 2 MAPE:", round(mape_comb_ols_fore2,2), "%")
```

```
OLS 2 MAPE: 406.6 %
```

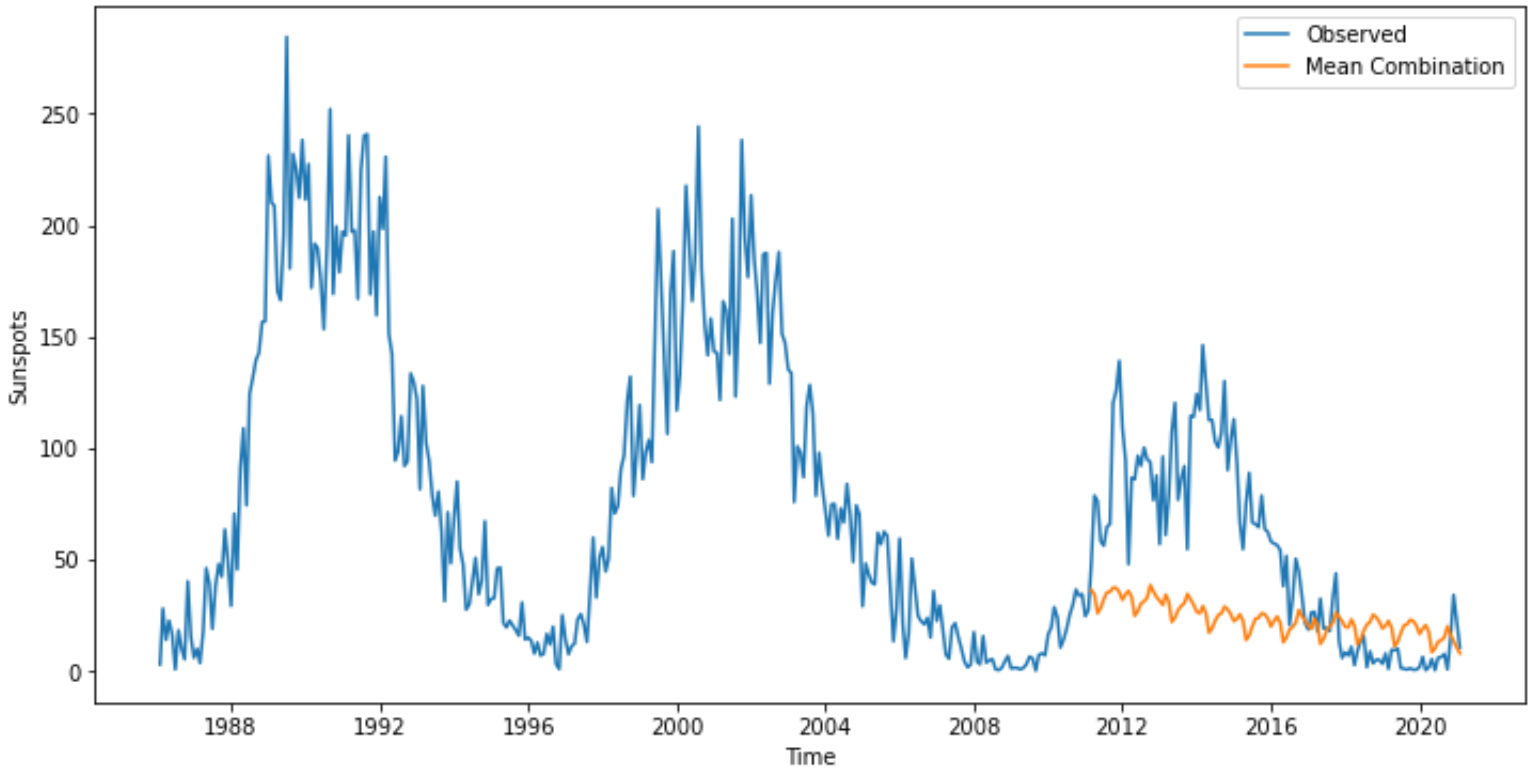
After dropping Seasonal Naive, the MAPE improves from 461.38% to 406.59%

Combination 2: Using Mean of Forecasts

In [79]:
```
comb_mean_fore = combine_test[['Snaive','Prophet', 'ARIMA']].mean(axis = 1)
```

In [81]:
```
plt.figure(figsize = (12,6))
plt.xlabel("Time")
plt.ylabel("Sunspots")
plt.plot(df[300:], label =  "Observed")
plt.plot(comb_mean_fore, label = "Mean Combination")
plt.legend()
#plt.savefig("combs.png", dpi = 300)
```

Out[81]: <matplotlib.legend.Legend at 0x7fb305bfb280>



In [84]:
```
# MAPE
mape_comb_mean_fore = np.mean(np.abs((combine_test["Sunspots"] - comb_mean_fore) /combine_test["S

# Print the MAPE
print("MAPE:", round(mape_comb_mean_fore,2),"%")
```

```
MAPE: 437.29 %
```

Using a simple average of the 3 models' forecasts, we get a MAPE of 437.28%, which lies between OLS1 and OLS2.

None of the combinations outperform the Prophet model alone!

LSTM (Bonus Model 2)

```
# Importing necessary libraries
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt


# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)


# Defining the number of previous time steps to consider for each prediction
n_steps = 200


# Prepare the data in a supervised learning format
X, y = [], []
for i in range(n_steps, len(scaled_data)):
    X.append(scaled_data[i - n_steps:i, 0])
    y.append(scaled_data[i, 0])
X, y = np.array(X), np.array(y)
```

```
# Split the data into training and testing sets
X_train, X_test = X[:-120], X[-120:]
y_train, y_test = y[:-120], y[-120:]


# Reshape the input data to be 3D (samples, timesteps, features)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))


# Define the LSTM model
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(n_steps, 1)))
model.add(LSTM(64))
model.add(Dense(30))
model.add(Dense(10))
model.add(Dense(1))


# Compile the model
model.compile(optimizer='adam', loss='mean_absolute_percentage_error')


# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)
```

```
Epoch 1/100
13/13 [==============================] - 6s 184ms/step - loss: 21925.0293
Epoch 2/100
13/13 [==============================] - 3s 245ms/step - loss: 229502.8594
Epoch 3/100
13/13 [==============================] - 2s 188ms/step - loss: 32389.5840
Epoch 4/100
13/13 [==============================] - 2s 187ms/step - loss: 172973.9219
Epoch 5/100
```

```
13/13 [==============================] - 2s 187ms/step - loss: 2592.8140
Epoch 6/100
13/13 [==============================] - 2s 186ms/step - loss: 98469.4375
Epoch 7/100
13/13 [==============================] - 3s 247ms/step - loss: 78984.2969
Epoch 8/100
13/13 [==============================] - 2s 186ms/step - loss: 8700.3652
Epoch 9/100
13/13 [==============================] - 2s 188ms/step - loss: 39269.7422
Epoch 10/100
13/13 [==============================] - 2s 186ms/step - loss: 19024.9512
Epoch 11/100
13/13 [==============================] - 3s 227ms/step - loss: 28838.5664
Epoch 12/100
13/13 [==============================] - 3s 193ms/step - loss: 22044.4316
Epoch 13/100
13/13 [==============================] - 2s 188ms/step - loss: 42119.1523
Epoch 14/100
13/13 [==============================] - 2s 185ms/step - loss: 8743.4268
Epoch 15/100
13/13 [==============================] - 2s 188ms/step - loss: 43219.1602
Epoch 16/100
13/13 [==============================] - 3s 246ms/step - loss: 13755.5811
Epoch 17/100
13/13 [==============================] - 2s 188ms/step - loss: 95326.1328
Epoch 18/100
13/13 [==============================] - 2s 188ms/step - loss: 50285.6367
Epoch 19/100
13/13 [==============================] - 2s 189ms/step - loss: 19683.7246
Epoch 20/100
13/13 [==============================] - 3s 215ms/step - loss: 15991.9805
Epoch 21/100
13/13 [==============================] - 3s 208ms/step - loss: 29063.3633
Epoch 22/100
13/13 [==============================] - 2s 188ms/step - loss: 11139.5557
Epoch 23/100
13/13 [==============================] - 2s 188ms/step - loss: 41944.6836
```

```
Epoch 24/100
13/13 [==============================] - 2s 187ms/step - loss: 6451.1191
Epoch 25/100
13/13 [==============================] - 3s 245ms/step - loss: 33246.3203
Epoch 26/100
13/13 [==============================] - 2s 188ms/step - loss: 8623.1426
Epoch 27/100
13/13 [==============================] - 2s 188ms/step - loss: 34238.8125
Epoch 28/100
13/13 [==============================] - 2s 188ms/step - loss: 733.7668
Epoch 29/100
13/13 [==============================] - 3s 204ms/step - loss: 63209.1562
Epoch 30/100
13/13 [==============================] - 3s 242ms/step - loss: 33656.7812
```

```
# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
```

```
13/13 [==============================] - 1s 54ms/step
4/4 [==============================] - 0s 55ms/step
```

```
# Inverse transform the predictions to the original scale
train_predictions = scaler.inverse_transform(train_predictions)
y_train = scaler.inverse_transform([y_train])
test_predictions = scaler.inverse_transform(test_predictions)
y_test = scaler.inverse_transform([y_test])
```

```
mape_lstm = np.mean(np.abs((y_test - test_predictions)/y_test))*100
print("LSTM MAPE:", round(mape_lstm,2), "%")
```

```
LSTM MAPE: 93.29 %
```

We use an LSTM model with two LSTM layers of 64 units each followed by three fully connected layers. The optimizer used is 'adam' with 100 epochs and a batch size of 32. The architecture of this LSTM and the hyperparameters can be optimized, for eg., using a gridsearch. However, even without optimization, the model yields the lowest MAPE of all.

SARIMA- 132 Periodicity (Bonus Model 3)

[ ]:
```
# Import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from fredapi import Fred
import statsmodels.formula.api as smf
import seaborn as sns
from sklearn import metrics
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot, plot_acf,␣
↪plot_pacf, quarter_plot
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.arima.model import ARIMA
```

[ ]:
```
# Importing the data
df = pd.read_csv('Sunspots.csv', index_col = 0)
df['Date'] = pd.to_datetime(df['Date'])
df = df.rename(columns={"Monthly Mean Total Sunspot Number": "Sunspots"})

# Subset data to observe last 60 years (12 months * 60 (+1 to start in January))
df_sub = df.tail(721)
#Add column for months
df_sub['Month']=df_sub.Date.dt.month
#Add column for year of the decade
df_sub['Year'] =[int(str(i)[3]) for i in (df_sub.Date.dt.year)]
df_sub['Year'].replace(0,10,inplace=True)

sunspot = df_sub.copy()
#set index
sunspot = sunspot.set_index('Date')
# Apply differencing
```

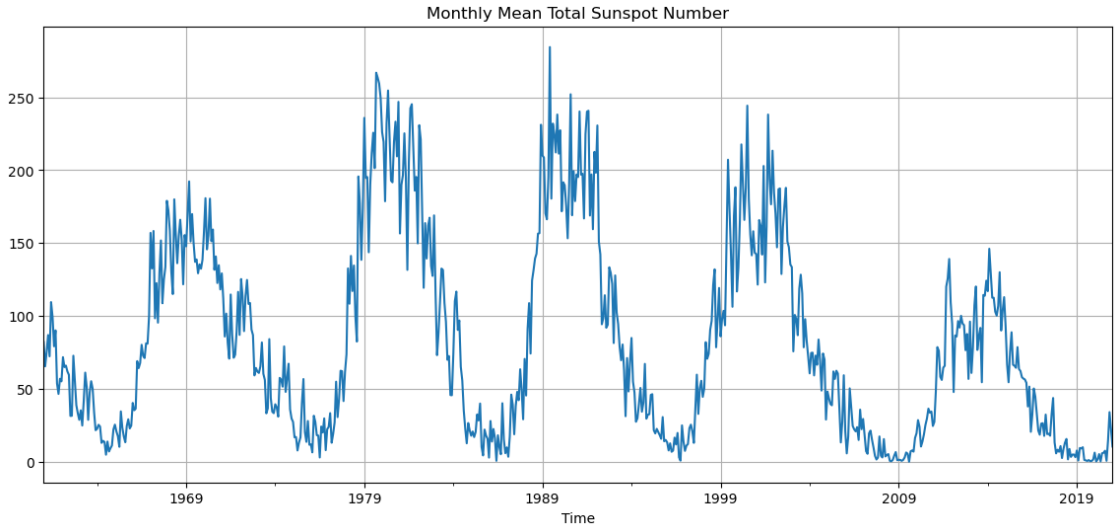
```
#**With this data-- why apply differencing? trend looks cyclical??***
sunspot['Diff'] = sunspot['Sunspots'].diff()
```

```
/var/folders/d7/c1j3512n6q7cl805hgvgjdmc0000gn/T/ipykernel_82302/1409006346.py:9
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_sub['Month']=df_sub.Date.dt.month
/var/folders/d7/c1j3512n6q7cl805hgvgjdmc0000gn/T/ipykernel_82302/1409006346.py:1
1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_sub['Year'] =[int(str(i)[3]) for i in (df_sub.Date.dt.year)]
/var/folders/d7/c1j3512n6q7cl805hgvgjdmc0000gn/T/ipykernel_82302/1409006346.py:1
2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_sub['Year'].replace(0,10,inplace=True)
```

2.2 Time series plots

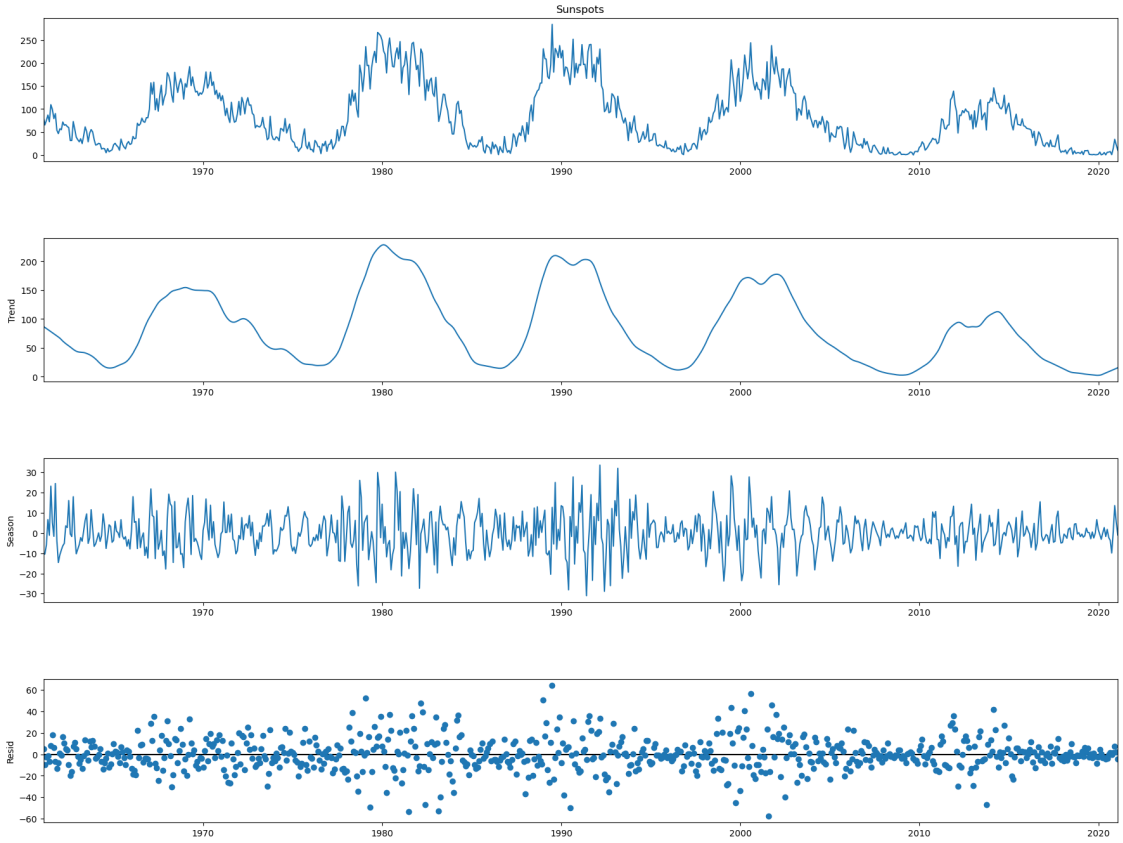
[ ]:
```
plt.figure(figsize = (14,6))

sunspot['Sunspots'].plot(grid = True)
plt.title("Monthly Mean Total Sunspot Number")
plt.xlabel("Time")
```

[ ]:
```
Text(0.5, 0, 'Time')
```

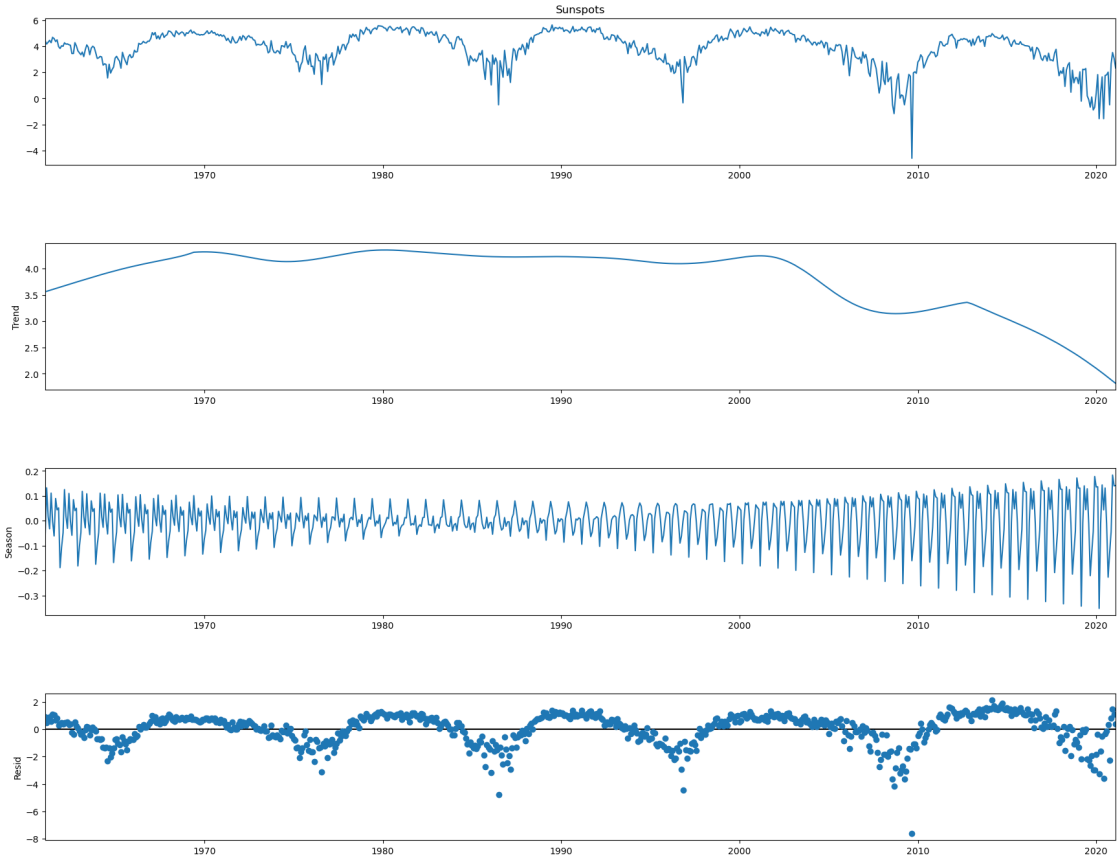


2.3 Plot the STL decomposition

[ ]:
```
# Plotting the STL decomposition
stl = STL(sunspot['Sunspots'])
results = stl.fit()

# Plot the results
fig = results.plot()

fig.set_figwidth(20)
fig.set_figheight(15)
plt.show()
```



[ ]:
```
# Plotting the STL decomposition
stl = STL(np.log(sunspot['Sunspots'] + 0.01), seasonal = 145, trend= 201)
results = stl.fit()

# Plot the results
fig = results.plot()

fig.set_figwidth(20)
fig.set_figheight(15)
plt.show()
```

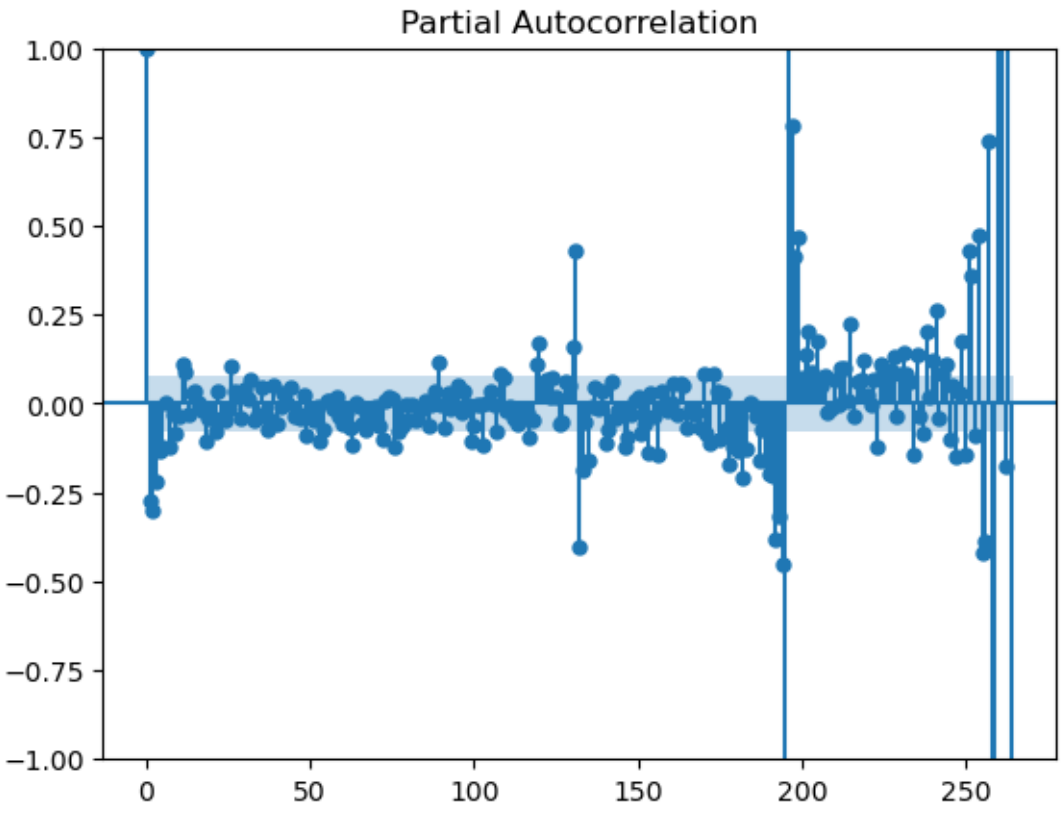


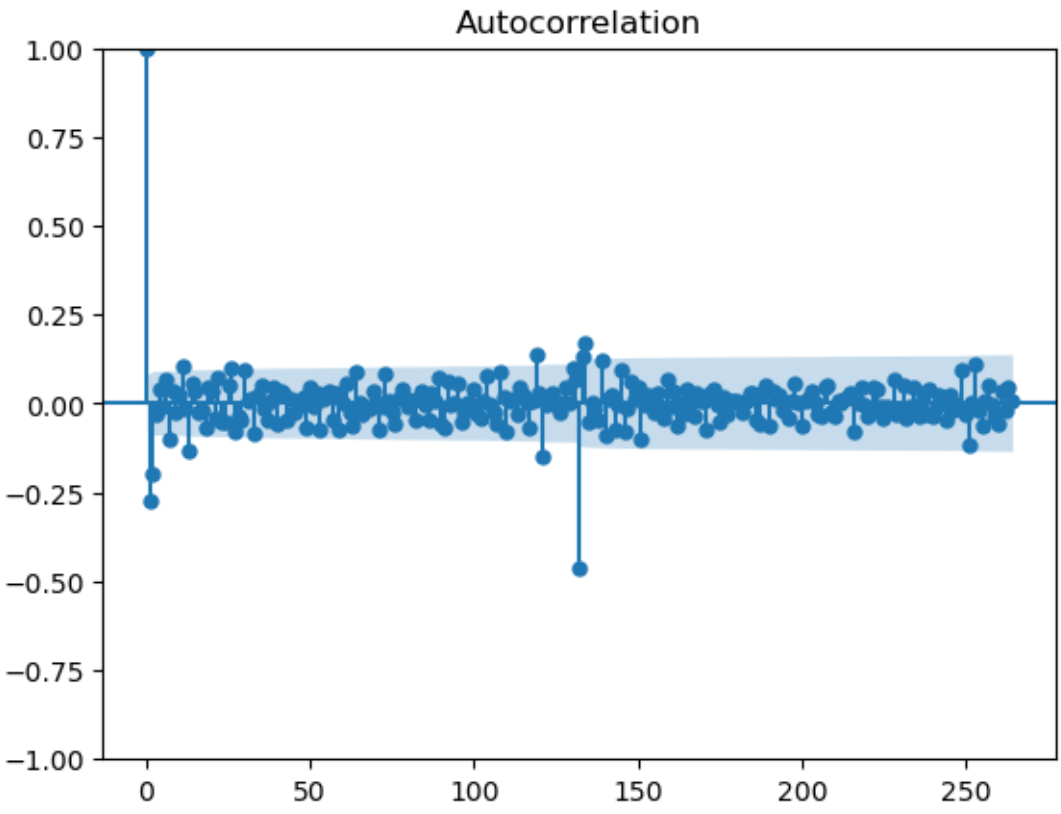
3 View Seasonally (132mo) Differenced ACF/PACF

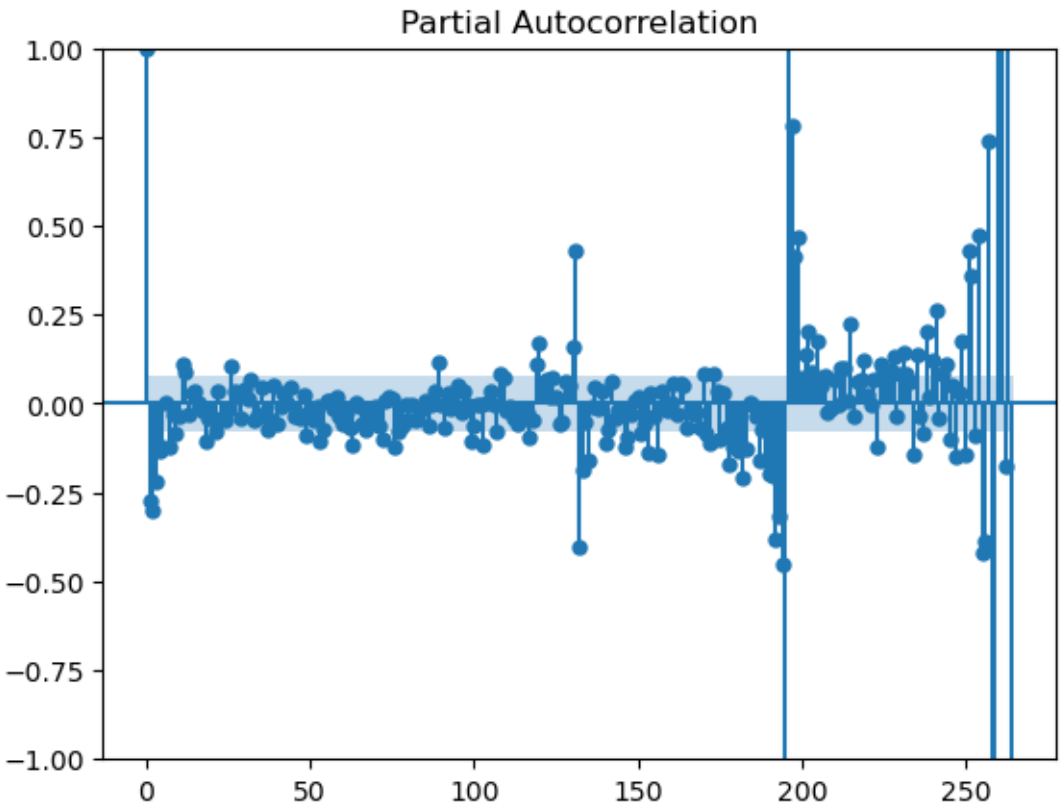
[ ]:
```
sdiff = sunspot['Sunspots'].diff(132).dropna()
sdiff2 = sdiff.diff().dropna()
plot_acf(sdiff2, lags=264)
plot_pacf(sdiff2, lags=264)
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```

[ ]:


Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250



Significant AR lags within the year, but very significant ACF/PACF lags at the 132 and 264 month lags.

---

4 Specify Various 132 month 'Seasonal' ARIMAs

It is unclear exactly which type of model to train from the ACF/PACF, but the 'seasonal' naive model performed quite well out of the box and seasonally (132mo) differenced data reveals two seasonal AR lags and a handful of intrayear AR lags. A bottleneck for gridsearching parameters is compute time. Some of our computers can't run a 132 month SARIMA and many of the models below took over 10 minutes to train. Many models achieve comparable AIC values, but then perform draastically different out of sample when projecting 120 months.

For clarity, the top 2 models based on AIC and RMSE/visual fithave been dragged to the top (model_arima_sma2, model_arima_d2: note the model names are misnomers from copying and pasting models with seasonal MAs and second differenced seasonal data)

[ ]:
```
model_arima_sma2 = ARIMA(sunspot['Sunspots'][:-120], order=(4, 0, 0),␣
  ↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_sma2.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

Dep. Variable:	Sunspots	No. Observations:	601
Model:	ARIMA(4, 0, 0)x(1, 1, 0, 132)	Log Likelihood	-2255.890
Date:	Thu, 08 Jun 2023	AIC	4523.779
Time:	23:14:41	BIC	4548.683
Sample:	01-31-1961	HQIC	4533.578
	- 01-31-2011		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4954	0.040	12.396	0.000	0.417	0.574
ar.L2	0.0820	0.043	1.890	0.059	-0.003	0.167
ar.L3	0.1072	0.043	2.503	0.012	0.023	0.191
ar.L4	0.2408	0.040	6.039	0.000	0.163	0.319
ar.S.L132	-0.5848	0.036	-16.406	0.000	-0.655	-0.515
sigma2	781.0509	41.853	18.662	0.000	699.021	863.081

Ljung-Box (L1) (Q):	0.13	Jarque-Bera (JB):	31.37
Prob(Q):	0.72	Prob(JB):	0.00
Heteroskedasticity (H):	1.36	Skew:	0.02
Prob(H) (two-sided):	0.06	Kurtosis:	4.27

```
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_d2= ARIMA(sunspot['Sunspots'][:-120], order=(4, 0, 0),␣
 ↪seasonal_order = (2, 1, 0, 132)).fit()
print(model_arima_d2.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==========================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(4, 0, 0)x(2, 1, 0, 132)   Log Likelihood
-2230.081
Date:                           Thu, 08 Jun 2023   AIC
4474.162
Time:                                   23:29:55   BIC
4503.216
Sample:                               01-31-1961   HQIC
4485.593
                                    - 01-31-2011
Covariance Type:                              opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1          0.5175      0.039     13.114      0.000       0.440       0.595
ar.L2          0.0569      0.044      1.308      0.191      -0.028       0.142
ar.L3          0.1371      0.045      3.059      0.002       0.049       0.225
ar.L4          0.1913      0.043      4.402      0.000       0.106       0.276
ar.S.L132     -0.8486      0.047    -17.895      0.000      -0.942      -0.756
ar.S.L264     -0.5363      0.051    -10.567      0.000      -0.636      -0.437
sigma2       587.1011     42.230     13.902      0.000     504.331     669.871
```

```
==========================================================================
===
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):
27.00
Prob(Q):                              0.76   Prob(JB):
0.00
Heteroskedasticity (H):               1.41   Skew:
0.07
Prob(H) (two-sided):                  0.03   Kurtosis:
4.17
==========================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

---

5 Extra Models: Trained but abandoned

[ ]:
```
model_arima_sma2_ar = ARIMA(sunspot['Sunspots'][:-120], order=(8, 0, 0),␣
↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_sma2_ar.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==========================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(8, 0, 0)x(1, 1, 0, 132)   Log Likelihood
-2252.619
Date:                           Tue, 06 Jun 2023   AIC
4525.239
Time:                                   19:16:29   BIC
```

```
4566.745
Sample:                                01-31-1961   HQIC
4541.570
                                     - 01-31-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4794      0.043     11.186      0.000       0.395       0.563
ar.L2          0.0596      0.047      1.282      0.200      -0.032       0.151
ar.L3          0.1033      0.044      2.328      0.020       0.016       0.190
ar.L4          0.1927      0.046      4.176      0.000       0.102       0.283
ar.L5          0.0455      0.045      1.007      0.314      -0.043       0.134
ar.L6          0.0730      0.043      1.712      0.087      -0.011       0.156
ar.L7         -0.0732      0.051     -1.449      0.147      -0.172       0.026
ar.L8          0.0606      0.038      1.587      0.112      -0.014       0.135
ar.S.L132     -0.5724      0.036    -15.864      0.000      -0.643      -0.502
sigma2       774.8294     42.298     18.318      0.000     691.926     857.733
==============================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
34.86
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):               1.33   Skew:
-0.04
Prob(H) (two-sided):                  0.08   Kurtosis:
4.33
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_sma2
```

[ ]:
```
model_arima_sma2 = ARIMA(sunspot['Sunspots'][:-120], order=(4, 0, 0),␣
 ↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_sma2.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
```

```
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==========================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(4, 0, 0)x(1, 1, 0, 132)   Log Likelihood
-2255.890
Date:                           Tue, 06 Jun 2023   AIC
4523.779
Time:                                   18:11:16   BIC
4548.683
Sample:                               01-31-1961   HQIC
4533.578
                                    - 01-31-2011
Covariance Type:                              opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1          0.4954      0.040     12.396      0.000       0.417       0.574
ar.L2          0.0820      0.043      1.890      0.059      -0.003       0.167
ar.L3          0.1072      0.043      2.503      0.012       0.023       0.191
ar.L4          0.2408      0.040      6.039      0.000       0.163       0.319
ar.S.L132     -0.5848      0.036    -16.406      0.000      -0.655      -0.515
sigma2       781.0509     41.853     18.662      0.000     699.021     863.081
==========================================================================
===
Ljung-Box (L1) (Q):                   0.13   Jarque-Bera (JB):
31.37
Prob(Q):                              0.72   Prob(JB):
0.00
Heteroskedasticity (H):               1.36   Skew:
0.02
Prob(H) (two-sided):                  0.06   Kurtosis:
4.27
==========================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_sma = ARIMA(sunspot['Sunspots'][:-120], order=(2, 0, 0),␣
↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_sma.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==========================================================================
=========
Dep. Variable:                             Sunspots   No. Observations:
601
Model:             ARIMA(2, 0, 0)x(1, 1, 0, 132)   Log Likelihood
-2282.674
Date:                           Tue, 06 Jun 2023   AIC
4573.348
Time:                                   17:03:38   BIC
4589.951
Sample:                                 01-31-1961   HQIC
4579.881
                                      - 01-31-2011
Covariance Type:                              opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1          0.6074      0.039     15.583      0.000       0.531       0.684
ar.L2          0.2421      0.040      6.100      0.000       0.164       0.320
ar.S.L132     -0.5751      0.034    -16.695      0.000      -0.643      -0.508
sigma2       880.5183     45.796     19.227      0.000     790.761     970.276
==========================================================================
===
Ljung-Box (L1) (Q):                   1.39   Jarque-Bera (JB):
47.75
Prob(Q):                              0.24   Prob(JB):
0.00
Heteroskedasticity (H):               1.46   Skew:
0.21
Prob(H) (two-sided):                  0.02   Kurtosis:
```

```
4.51
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_ar4_s2_ma1 = ARIMA(sunspot['Sunspots'][:-120], order=(1, 1, 0),␣
 ↪seasonal_order = (1, 1, 1, 132)).fit()
print(model_arima_ar4_s2_ma1.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/statespace/sarimax.py:1009: UserWarning: Non-invertible
starting seasonal moving average Using zeros as starting parameters.
  warn('Non-invertible starting seasonal moving average'

                                     SARIMAX Results
==============================================================================
===========
Dep. Variable:                             Sunspots   No. Observations:
601
Model:             ARIMA(1, 1, 0)x(1, 1, [1], 132)   Log Likelihood
-2263.219
Date:                             Tue, 06 Jun 2023   AIC
4534.438
Time:                                     16:48:11   BIC
4551.032
Sample:                                 01-31-1961   HQIC
4540.968
                                      - 01-31-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3072      0.037     -8.335      0.000      -0.379      -0.235
```

```
ar.S.L132     -0.1852      0.088     -2.100      0.036      -0.358      -0.012
ma.S.L132     -0.9993    119.251     -0.008      0.993    -234.727     232.728
sigma2       557.9696   6.65e+04      0.008      0.993    -1.3e+05    1.31e+05
==============================================================================
===
Ljung-Box (L1) (Q):                   3.70   Jarque-Bera (JB):
28.75
Prob(Q):                              0.05   Prob(JB):
0.00
Heteroskedasticity (H):               1.29   Skew:
0.03
Prob(H) (two-sided):                  0.11   Kurtosis:
4.21
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_ma = ARIMA(sunspot['Sunspots'][:-120], order=(3, 1, 1),␣
 ↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_ma.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==============================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(3, 1, 0)x(0, 1, 0, 132)   Log Likelihood
-2327.171
Date:                           Tue, 06 Jun 2023   AIC
4662.342
Time:                                   02:59:34   BIC
4678.936
```

```
Sample:                                 01-31-1961   HQIC
4668.872
                                      - 01-31-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4344      0.038    -11.508      0.000      -0.508      -0.360
ar.L2         -0.3916      0.038    -10.222      0.000      -0.467      -0.317
ar.L3         -0.2285      0.037     -6.148      0.000      -0.301      -0.156
sigma2      1220.9054     66.856     18.262      0.000    1089.870    1351.941
==============================================================================
===
Ljung-Box (L1) (Q):                   0.64   Jarque-Bera (JB):
26.98
Prob(Q):                              0.42   Prob(JB):
0.00
Heteroskedasticity (H):               1.38   Skew:
-0.09
Prob(H) (two-sided):                  0.05   Kurtosis:
4.16
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima = ARIMA(sunspot['Sunspots'][:-120], order=(3, 1, 0), seasonal_order␣
↪= (0, 1, 0, 132)).fit()
print(model_arima.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                    SARIMAX Results
==============================================================================
=========
```

```
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(3, 1, 0)x(0, 1, 0, 132)   Log Likelihood
-2327.171
Date:                           Tue, 06 Jun 2023   AIC
4662.342
Time:                                   02:55:55   BIC
4678.936
Sample:                               01-31-1961   HQIC
4668.872
                                    - 01-31-2011
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4344      0.038    -11.508      0.000      -0.508      -0.360
ar.L2         -0.3916      0.038    -10.222      0.000      -0.467      -0.317
ar.L3         -0.2285      0.037     -6.148      0.000      -0.301      -0.156
sigma2      1220.9054     66.856     18.262      0.000    1089.870    1351.941
===================================================================================
===
Ljung-Box (L1) (Q):                   0.64   Jarque-Bera (JB):
26.98
Prob(Q):                              0.42   Prob(JB):
0.00
Heteroskedasticity (H):               1.38   Skew:
-0.09
Prob(H) (two-sided):                  0.05   Kurtosis:
4.16
===================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

The best model selected by the autoARIMA procedure is ARIMA(4, 0, 0)(2, 1, 0)[12]. This means it has an autoregressive order of 4, no differencing (d=0), no moving average terms (q=0), two seasonal autoregressive terms (P=2), one seasonal differencing (D=1), no seasonal moving average terms (Q=0), and a seasonal period of 12 (monthly data with yearly seasonality).

[ ]:
```
model_arima_ar3 = ARIMA(sunspot['Sunspots'][:-120], order=(3, 1, 0),␣
 ↪seasonal_order = (2, 1, 0, 132)).fit()
print(model_arima.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
```

```
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==========================================================================
=========
Dep. Variable:                             Sunspots   No. Observations:
601
Model:              ARIMA(3, 1, 0)x(2, 1, 0, 132)   Log Likelihood
-2229.851
Date:                            Tue, 06 Jun 2023   AIC
4471.701
Time:                                    01:46:49   BIC
4496.592
Sample:                                01-31-1961   HQIC
4481.496
                                     - 01-31-2011
Covariance Type:                               opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1         -0.4564      0.039    -11.732      0.000      -0.533      -0.380
ar.L2         -0.3779      0.040     -9.387      0.000      -0.457      -0.299
ar.L3         -0.2235      0.043     -5.231      0.000      -0.307      -0.140
ar.S.L132     -0.8552      0.048    -17.955      0.000      -0.949      -0.762
ar.S.L264     -0.5225      0.053     -9.921      0.000      -0.626      -0.419
sigma2       604.3862     43.648     13.847      0.000     518.837     689.935
==========================================================================
===
Ljung-Box (L1) (Q):                   0.30   Jarque-Bera (JB):
26.39
Prob(Q):                              0.59   Prob(JB):
0.00
Heteroskedasticity (H):               1.38   Skew:
-0.08
Prob(H) (two-sided):                  0.05   Kurtosis:
4.15
==========================================================================
===

Warnings:
```

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

[ ]:
```
model_arima_ar4_s3 = ARIMA(sunspot['Sunspots'][:-120], order=(4, 1, 0),␣
↪seasonal_order = (3, 1, 0, 132)).fit()
print(model_arima_ar4_s3.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

Dep. Variable:	Sunspots	No. Observations:	601
Model:	ARIMA(4, 1, 0)x(3, 1, 0, 132)	Log Likelihood	-2219.081
Date:	Tue, 06 Jun 2023	AIC	4454.162
Time:	03:53:01	BIC	4487.350
Sample:	01-31-1961	HQIC	4467.222
	- 01-31-2011		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4730	0.042	-11.205	0.000	-0.556	-0.390
ar.L2	-0.4292	0.043	-9.896	0.000	-0.514	-0.344
ar.L3	-0.2789	0.048	-5.803	0.000	-0.373	-0.185
ar.L4	-0.1099	0.041	-2.692	0.007	-0.190	-0.030
ar.S.L132	-1.0210	0.070	-14.665	0.000	-1.158	-0.885
ar.S.L264	-0.9272	0.070	-13.170	0.000	-1.065	-0.789
ar.S.L396	-0.7493	0.039	-19.347	0.000	-0.825	-0.673
sigma2	319.2937	35.846	8.907	0.000	249.038	389.550

```
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
24.84
Prob(Q):                              0.86   Prob(JB):
0.00
Heteroskedasticity (H):               1.45   Skew:
-0.18
Prob(H) (two-sided):                  0.02   Kurtosis:
4.07
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_ar4 = ARIMA(sunspot['Sunspots'][:-120], order=(4, 1, 0),␣
 ↪seasonal_order = (2, 1, 0, 132)).fit()
print(model_arima_ar4.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
================================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(4, 1, 0)x(2, 1, 0, 132)   Log Likelihood
-2227.169
Date:                           Tue, 06 Jun 2023   AIC
4468.337
Time:                                   02:50:01   BIC
4497.377
Sample:                                 01-31-1961   HQIC
4479.764
                                      - 01-31-2011
Covariance Type:                              opg
================================================================================
```

```
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4798      0.040    -11.877      0.000      -0.559      -0.401
ar.L2         -0.4190      0.042     -9.968      0.000      -0.501      -0.337
ar.L3         -0.2718      0.047     -5.813      0.000      -0.363      -0.180
ar.L4         -0.1073      0.039     -2.782      0.005      -0.183      -0.032
ar.S.L132     -0.8470      0.047    -17.847      0.000      -0.940      -0.754
ar.S.L264     -0.5252      0.053     -9.886      0.000      -0.629      -0.421
sigma2       597.8507     42.898     13.936      0.000     513.772     681.930
===================================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
30.22
Prob(Q):                              0.93   Prob(JB):
0.00
Heteroskedasticity (H):               1.39   Skew:
-0.14
Prob(H) (two-sided):                  0.04   Kurtosis:
4.21
===================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_ar4_s1 = ARIMA(sunspot['Sunspots'][:-120], order=(4, 1, 0),␣
 ↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_ar4_s1.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                    SARIMAX Results
===================================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
```

```
Model:             ARIMA(4, 1, 0)x(1, 1, 0, 132)   Log Likelihood
-2251.794
Date:                           Tue, 06 Jun 2023   AIC
4515.589
Time:                                   02:31:13   BIC
4540.479
Sample:                                 01-31-1961   HQIC
4525.383
                                      - 01-31-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5096      0.042    -12.180      0.000      -0.592      -0.428
ar.L2         -0.4234      0.041    -10.320      0.000      -0.504      -0.343
ar.L3         -0.3092      0.045     -6.917      0.000      -0.397      -0.222
ar.L4         -0.0906      0.040     -2.260      0.024      -0.169      -0.012
ar.S.L132     -0.5848      0.035    -16.880      0.000      -0.653      -0.517
sigma2       785.5031     42.056     18.678      0.000     703.075     867.932
===================================================================================
===
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
31.37
Prob(Q):                              0.86   Prob(JB):
0.00
Heteroskedasticity (H):               1.35   Skew:
-0.10
Prob(H) (two-sided):                  0.07   Kurtosis:
4.25
===================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
print(model_arima_ar4.summary())
```

```
                                      SARIMAX Results
==============================================================================
=========
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(4, 1, 0)x(2, 1, 0, 132)   Log Likelihood
-2227.169
Date:                           Tue, 06 Jun 2023   AIC
4468.337
Time:                                   02:10:15   BIC
```

```
4497.377
Sample:                                01-31-1961   HQIC
4479.764
                                     - 01-31-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4798      0.040    -11.877      0.000      -0.559      -0.401
ar.L2         -0.4190      0.042     -9.968      0.000      -0.501      -0.337
ar.L3         -0.2718      0.047     -5.813      0.000      -0.363      -0.180
ar.L4         -0.1073      0.039     -2.782      0.005      -0.183      -0.032
ar.S.L132     -0.8470      0.047    -17.847      0.000      -0.940      -0.754
ar.S.L264     -0.5252      0.053     -9.886      0.000      -0.629      -0.421
sigma2       597.8507     42.898     13.936      0.000     513.772     681.930
==============================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
30.22
Prob(Q):                              0.93   Prob(JB):
0.00
Heteroskedasticity (H):               1.39   Skew:
-0.14
Prob(H) (two-sided):                  0.04   Kurtosis:
4.21
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_s1 = ARIMA(sunspot['Sunspots'][:-120], order=(4, 1, 0),␣
 ↪seasonal_order = (1, 1, 0, 132)).fit()
print(model_arima_s1.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                           Sunspots   No. Observations:
601
Model:             ARIMA(4, 1, 0)x(1, 1, 0, 132)   Log Likelihood
-2251.794
Date:                           Tue, 06 Jun 2023   AIC
4515.589
Time:                                   02:22:42   BIC
4540.479
Sample:                               01-31-1961   HQIC
4525.383
                                    - 01-31-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5096      0.042    -12.180      0.000      -0.592      -0.428
ar.L2         -0.4234      0.041    -10.320      0.000      -0.504      -0.343
ar.L3         -0.3092      0.045     -6.917      0.000      -0.397      -0.222
ar.L4         -0.0906      0.040     -2.260      0.024      -0.169      -0.012
ar.S.L132     -0.5848      0.035    -16.880      0.000      -0.653      -0.517
sigma2       785.5031     42.056     18.678      0.000     703.075     867.932
==========================================================================================
===
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
31.37
Prob(Q):                              0.86   Prob(JB):
0.00
Heteroskedasticity (H):               1.35   Skew:
-0.10
Prob(H) (two-sided):                  0.07   Kurtosis:
4.25
==========================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
model_arima_ar5 = ARIMA(sunspot['Sunspots'][:-120], order=(5, 1, 0),␣
  ↪seasonal_order = (2, 1, 0, 132)).fit()
print(model_arima_ar5.summary())
```

```
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
```

```
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
                                     SARIMAX Results
==========================================================================
=========
Dep. Variable:                             Sunspots   No. Observations:
601
Model:              ARIMA(5, 1, 0)x(2, 1, 0, 132)   Log Likelihood
-2226.980
Date:                            Tue, 06 Jun 2023   AIC
4469.959
Time:                                    02:08:38   BIC
4503.147
Sample:                                01-31-1961   HQIC
4483.018
                                     - 01-31-2011
Covariance Type:                               opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1         -0.4832      0.041    -11.912      0.000      -0.563      -0.404
ar.L2         -0.4274      0.046     -9.281      0.000      -0.518      -0.337
ar.L3         -0.2847      0.049     -5.840      0.000      -0.380      -0.189
ar.L4         -0.1217      0.042     -2.907      0.004      -0.204      -0.040
ar.L5         -0.0291      0.040     -0.734      0.463      -0.107       0.049
ar.S.L132     -0.8405      0.048    -17.608      0.000      -0.934      -0.747
ar.S.L264     -0.5201      0.053     -9.731      0.000      -0.625      -0.415
sigma2       600.4539     42.960     13.977      0.000     516.253     684.655
==========================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
30.78
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):               1.39   Skew:
-0.15
Prob(H) (two-sided):                  0.04   Kurtosis:
4.22
==========================================================================
===
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

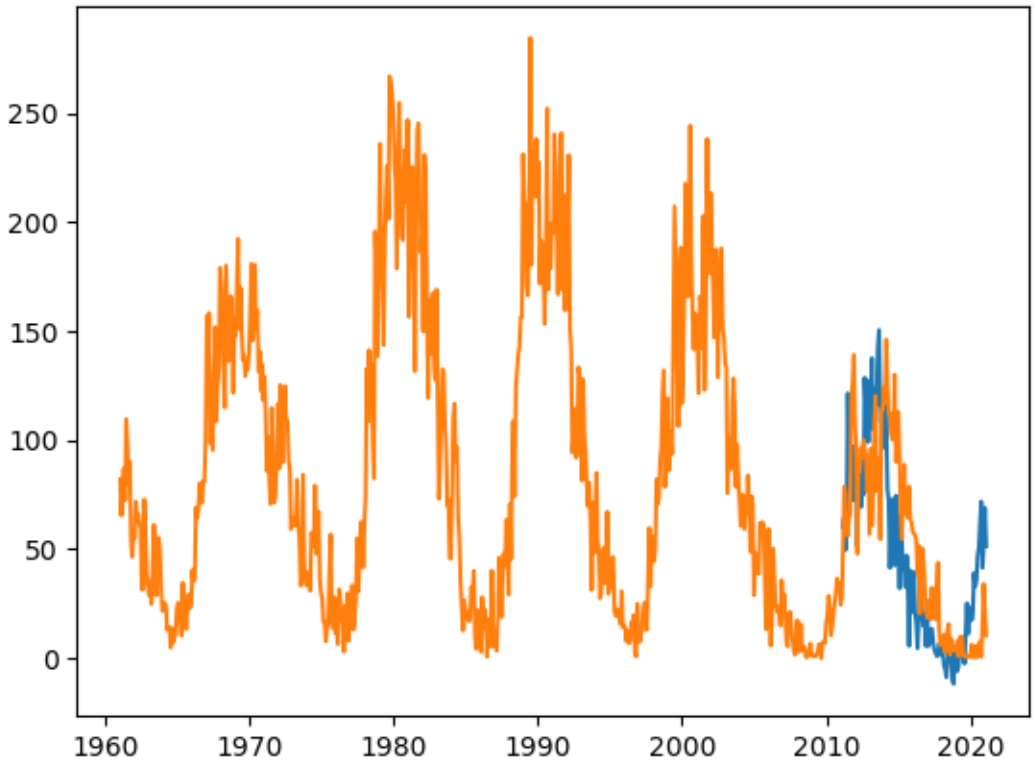
---

6 Plotted fits of abandoned models

Many models trail off and forecast below 0

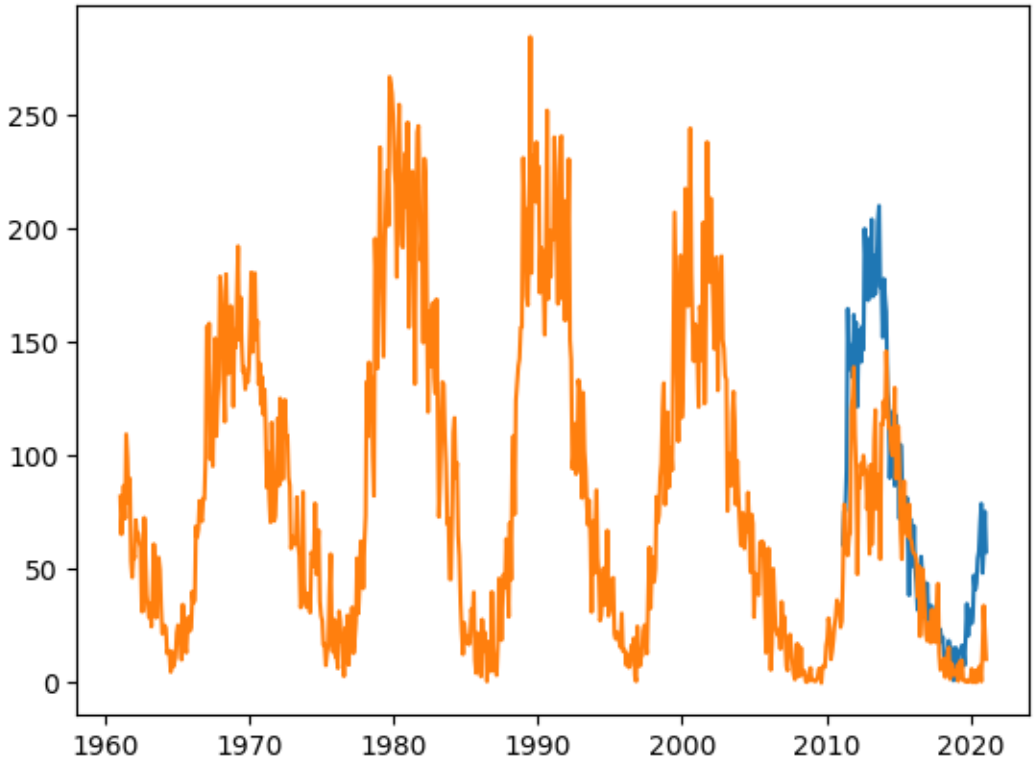
[ ]:
```
# model_arima_sma2_ar
plt.plot(model_arima_sma2_ma.forecast(120))
plt.plot(sunspot['Sunspots'])

print(metrics.mean_absolute_percentage_error(model_arima_sma2_ma.forecast(120),␣
  ↪sunspot['Sunspots'][-120:]))
print( np.sqrt(metrics.mean_squared_error(model_arima_sma2_ma.forecast(120),␣
  ↪sunspot['Sunspots'][-120:])))
```

```
1.8088540433456113
32.42877089030218
```



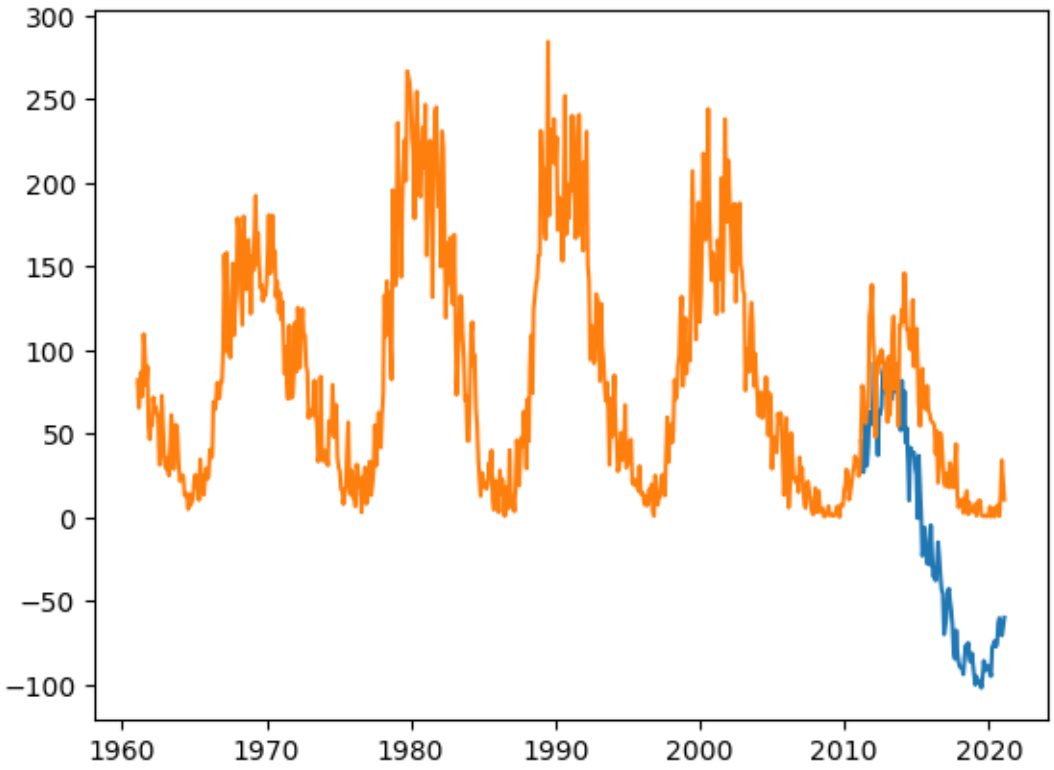
[ ]:
```
plt.plot(model_arima_sma.forecast(120))
plt.plot(sunspot['Sunspots'])

print(metrics.mean_absolute_percentage_error(model_arima_sma.forecast(120),␣
  ↪sunspot['Sunspots'][-120:]))
print(metrics.mean_squared_error(model_arima_sma.forecast(120),␣
  ↪sunspot['Sunspots'][-120:]))
```

```
0.46717114881910277
2245.945887069461
```



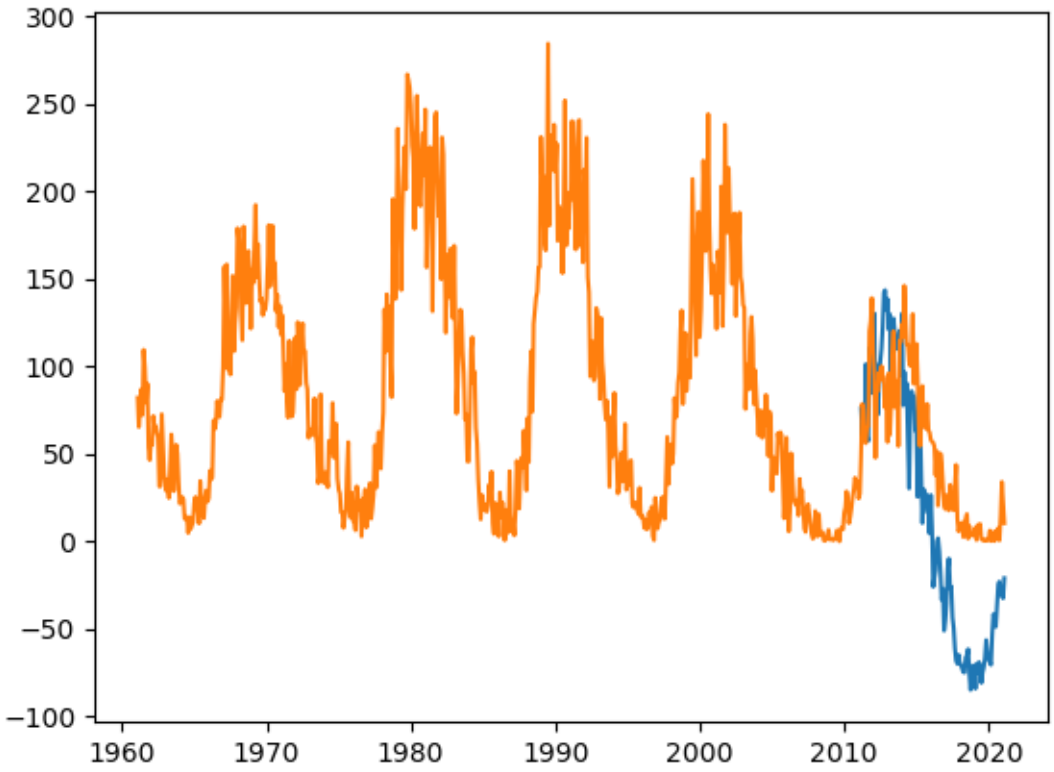
[ ]:
```
plt.plot(model_arima_ar4_s2_ma1.forecast(120))
plt.plot(sunspot['Sunspots'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f7c1cc48a30>]
```



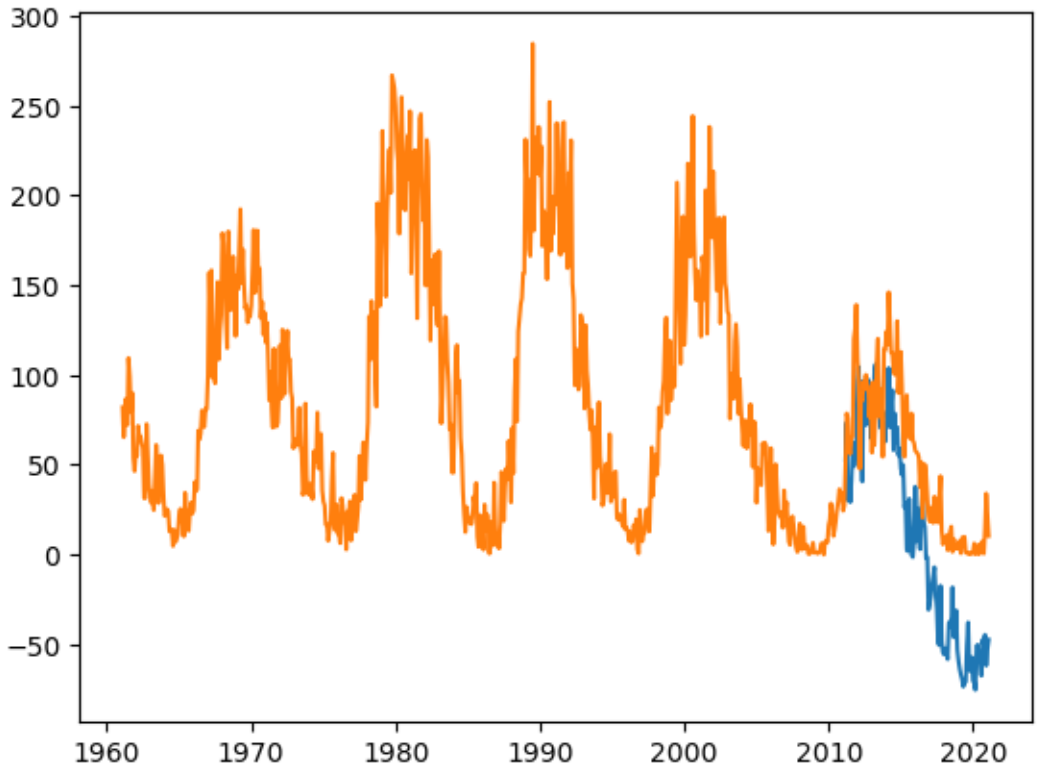
[ ]:
```
plt.plot(model_arima_ar4.forecast(120))
plt.plot(sunspot['Sunspots'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f87cb8a0a30>]
```



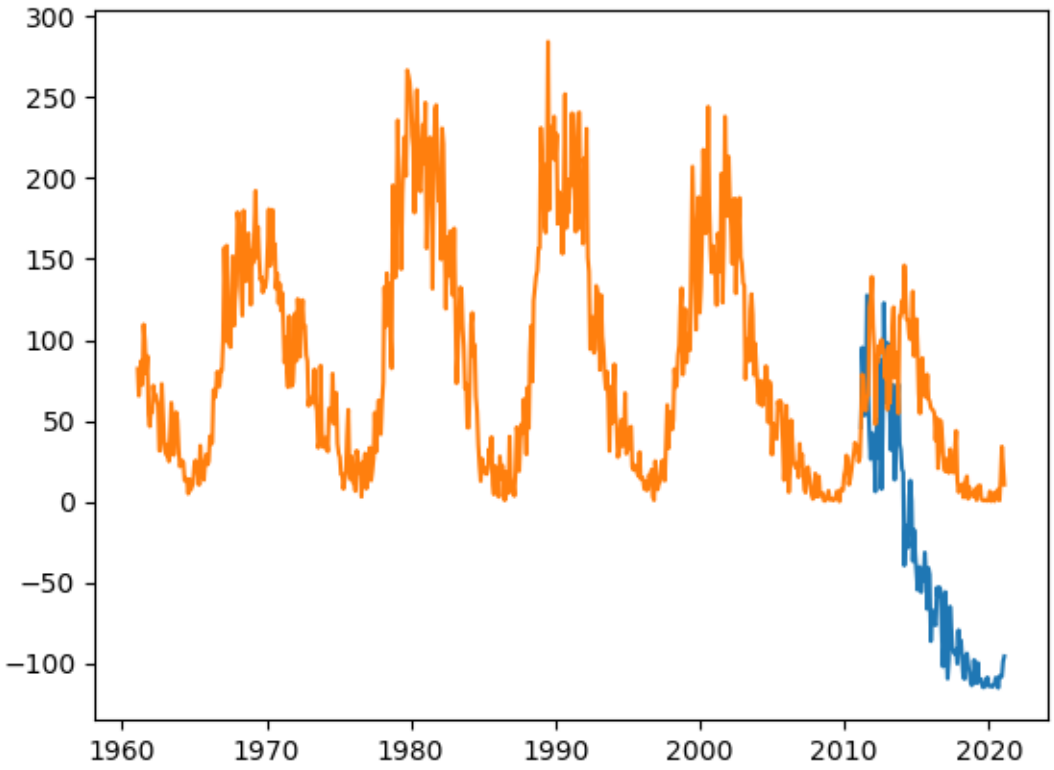
[ ]:
```
plt.plot(model_arima_ar4_s3.forecast(120))
plt.plot(sunspot['Sunspots'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f8548f17970>]
```



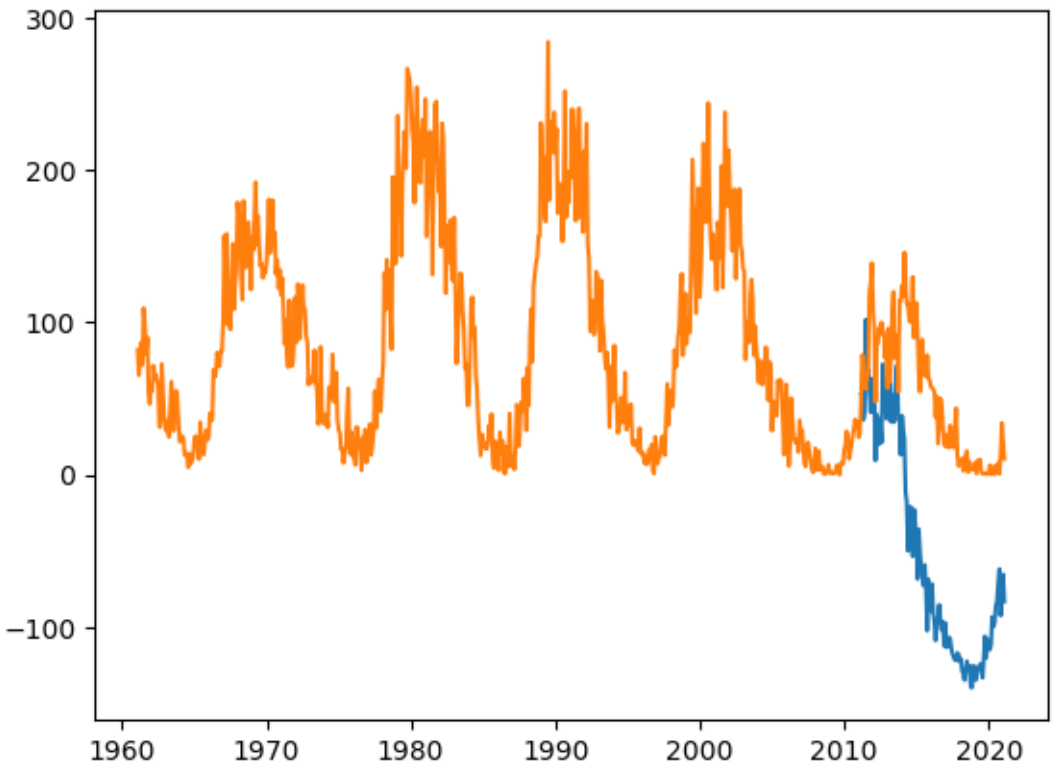
[ ]:
```
plt.plot(model_arima.forecast(120))
plt.plot(sunspot['Sunspots'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f87ba210400>]
```



[ ]:
```
plt.plot(model_arima_ma.forecast(120))
plt.plot(sunspot['Sunspots'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f8789267070>]
```

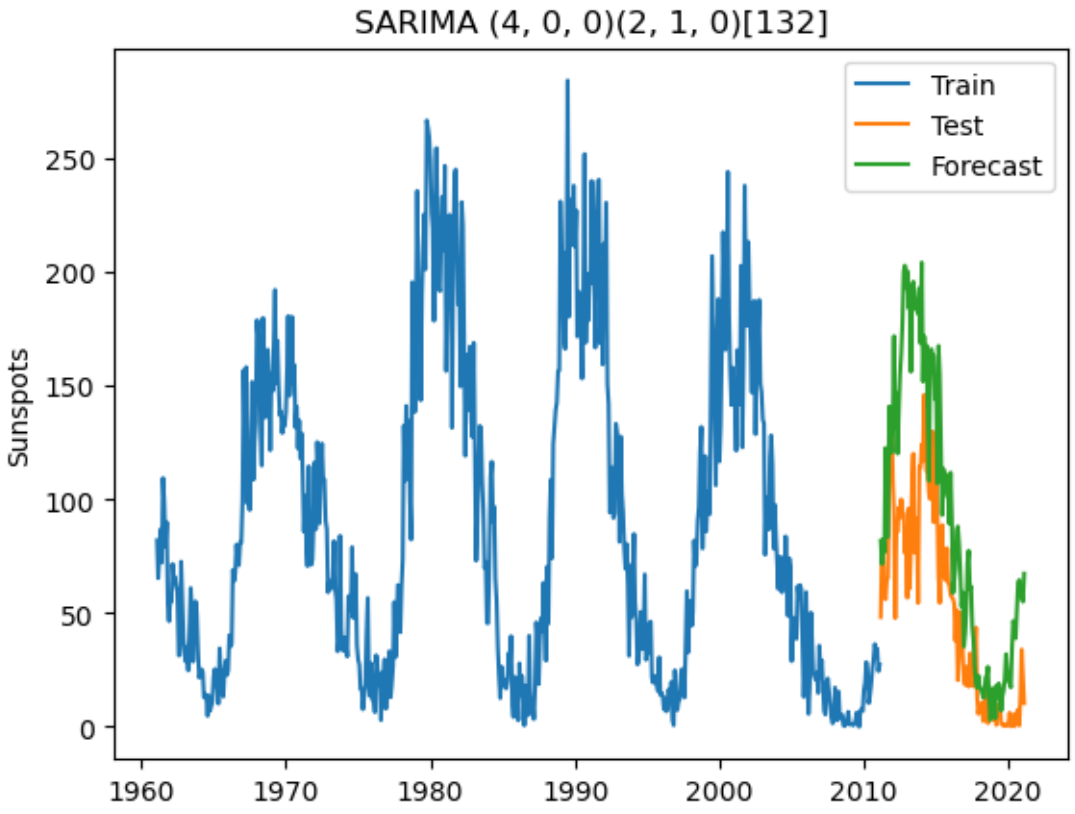


---

7 Best Models

[ ]:
```
# order=(4, 0, 0), seasonal_order = (1, 1, 0, 132) order=(4, 0, 0),␣
  ↪seasonal_order = (2, 1, 0, 132)
plt.title('SARIMA (4, 0, 0)(2, 1, 0)[132]')
plt.ylabel('Sunspots')
plt.plot(sunspot['Sunspots'][:-120], label='Train')
plt.plot(sunspot['Sunspots'][-120:], label='Test')
plt.plot(model_arima_d2.forecast(120), label='Forecast')
plt.legend()

print(metrics.mean_absolute_percentage_error(model_arima_d2.forecast(120),␣
  ↪sunspot['Sunspots'][-120:]))
print( np.sqrt(metrics.mean_squared_error(model_arima_d2.forecast(120),␣
  ↪sunspot['Sunspots'][-120:])))
```

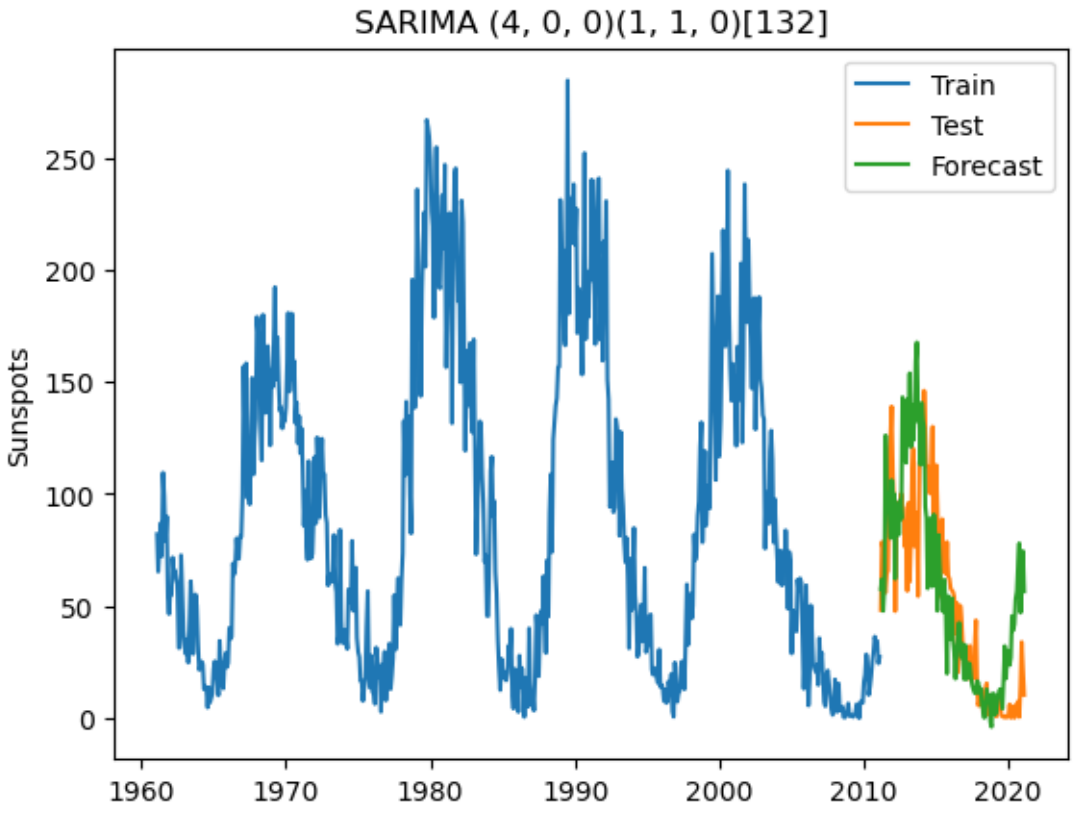
```
0.49879400358317777
51.305573873554806
```



7.1 SARIMA (4, 0, 0)(1, 1, 0)[132]

[ ]:
```
# order=(4, 0, 0), seasonal_order = (1, 1, 0, 132)
plt.title('SARIMA (4, 0, 0)(1, 1, 0)[132]')
plt.ylabel('Sunspots')
plt.plot(sunspot['Sunspots'][:-120], label='Train')
plt.plot(sunspot['Sunspots'][-120:], label='Test')
plt.plot(model_arima_sma2.forecast(120), label='Forecast')
plt.legend()

print(metrics.mean_absolute_percentage_error(model_arima_sma2.forecast(120),␣
  ↪sunspot['Sunspots'][-120:]))
print( np.sqrt(metrics.mean_squared_error(model_arima_sma2.forecast(120),␣
  ↪sunspot['Sunspots'][-120:])))
```

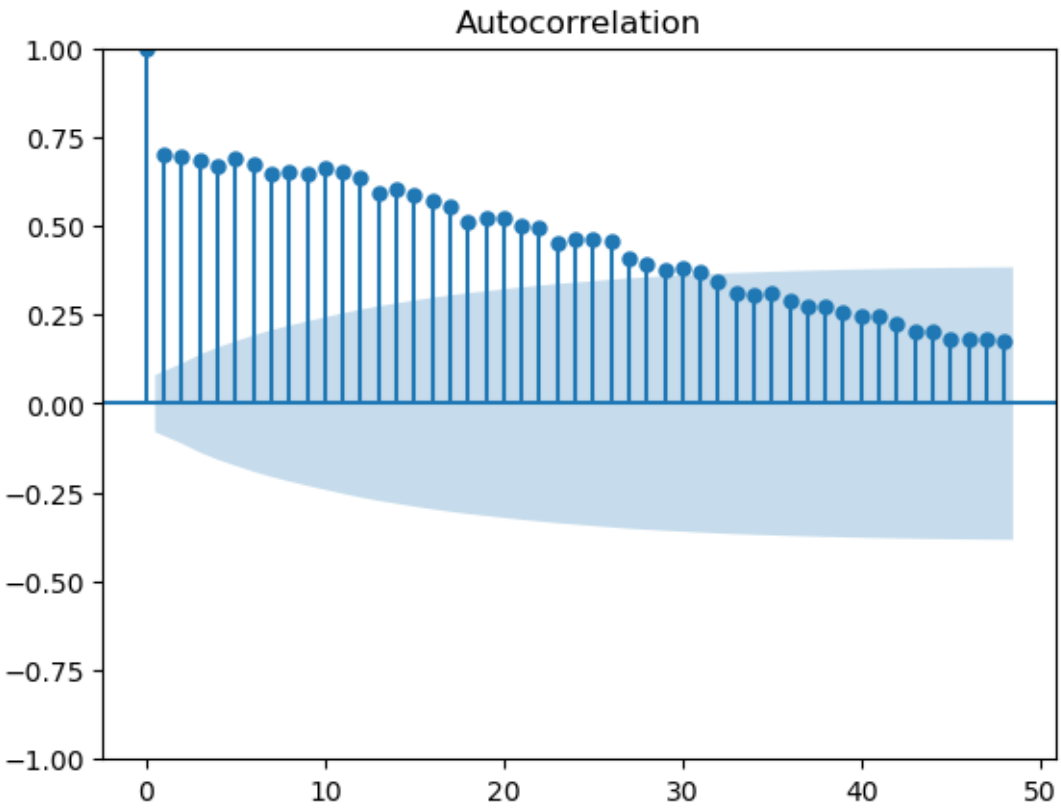
```
0.9409211176962551
31.784202469389918
```

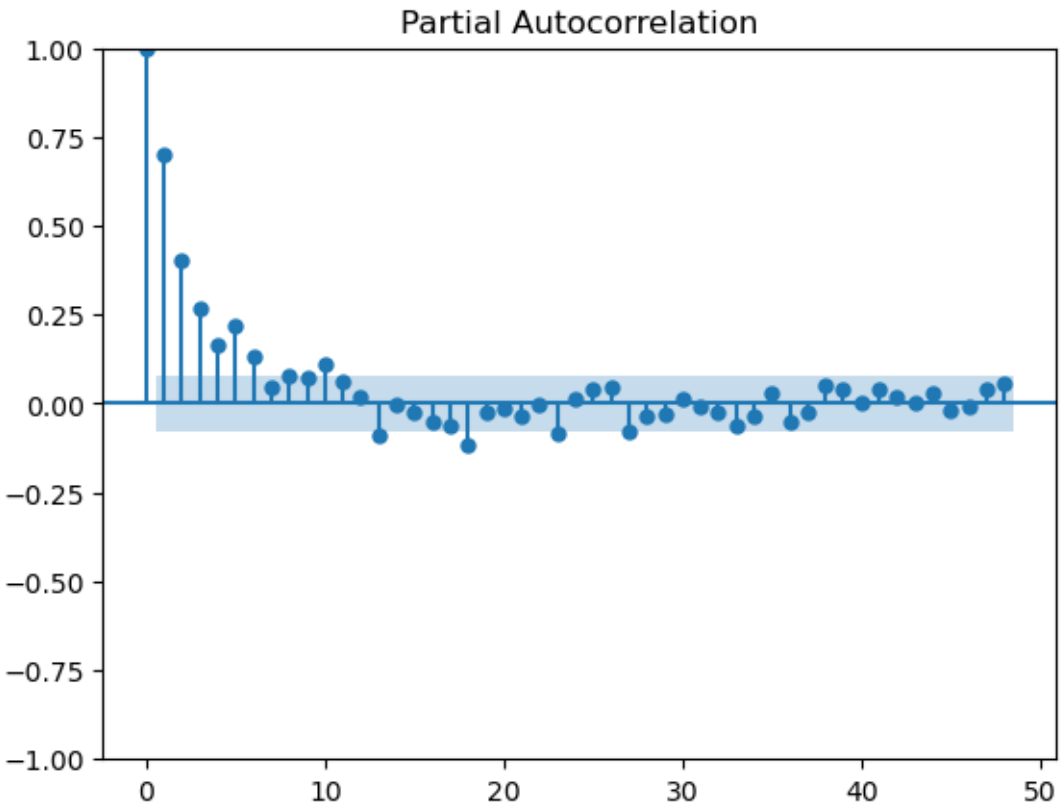


8 Review residuals...

[ ]:
```
#Plot ACF and PACF of differenced data
plot_acf(model_arima_d2.resid,  lags = 48)
plot_pacf(model_arima_d2.resid, lags = 48)
plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```

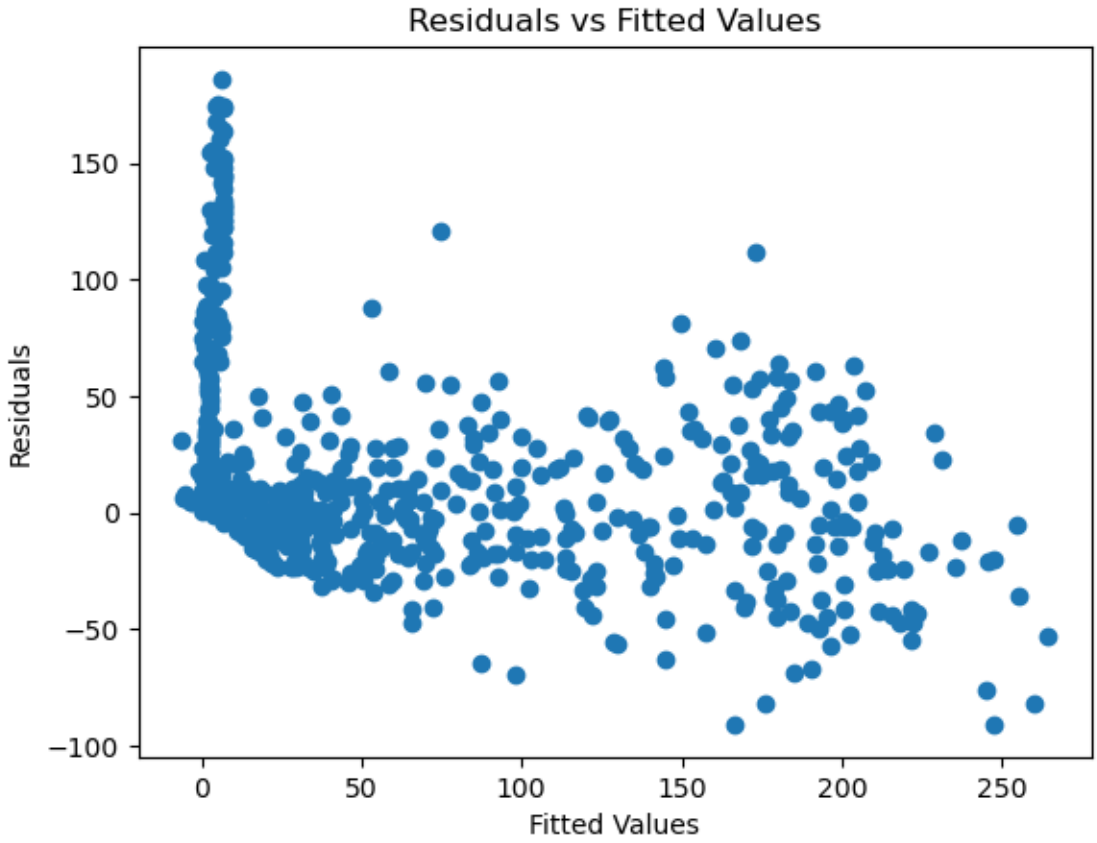

Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50

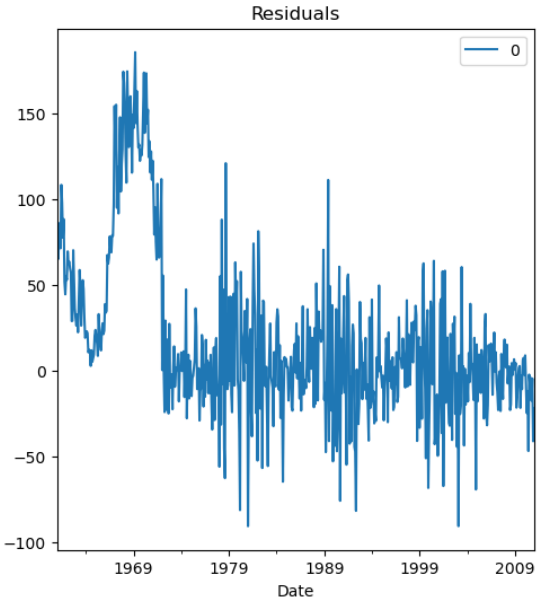


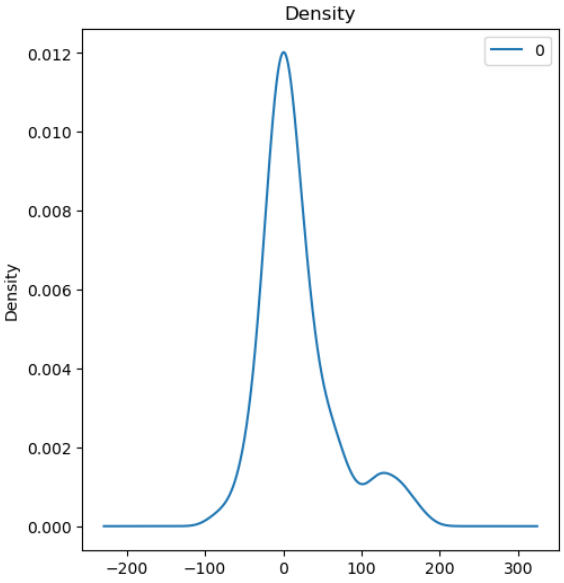
Uh oh, there is still plenty of pattern remaining in the residuals... but wait...

[ ]:
```
# Plotting residuals vs fitted values

# Extracting residuals
residuals = pd.DataFrame(model_arima_d2.resid)

# Extracting fitted values
fv = pd.DataFrame(model_arima_d2.fittedvalues)
plt.scatter(fv, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted Values')
plt.show()

# More diagnostics

fig, ax = plt.subplots(1, 2, figsize = (12,6))

residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
```

```
plt.show()
print(residuals.describe())
```







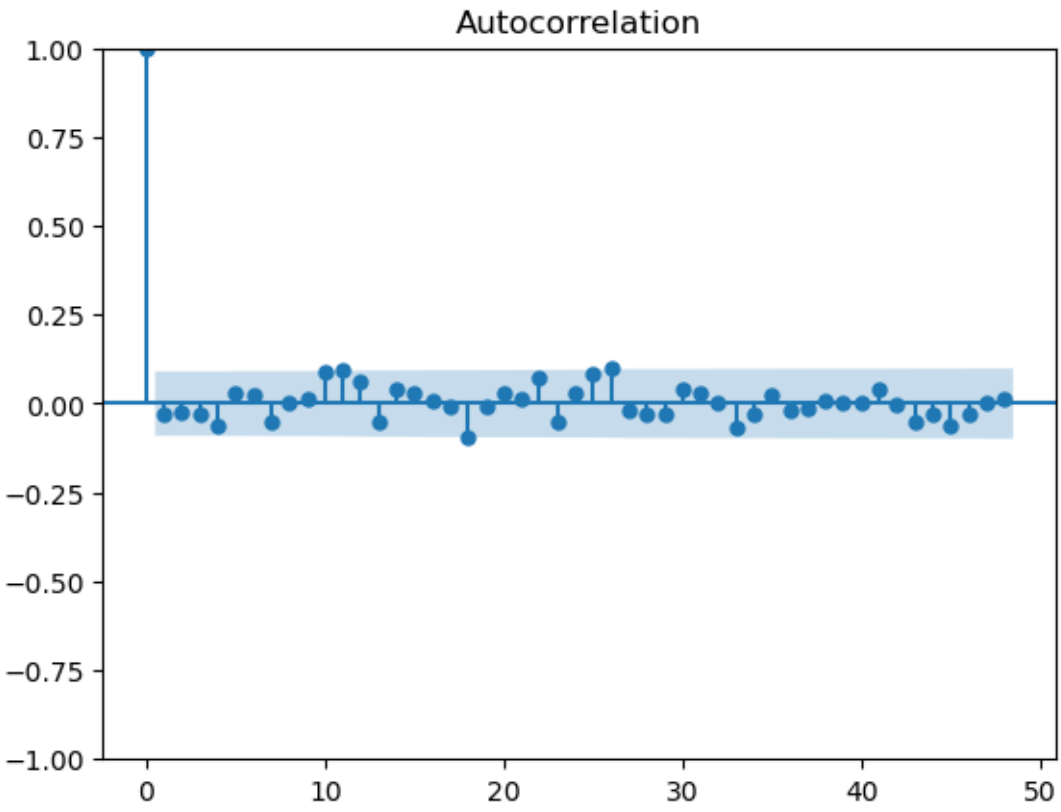
```
                0
count  601.000000
mean    18.328059
std     48.602275
min    -90.738230
25%    -10.764091
50%      6.223884
75%     34.564150
max    185.915118
```

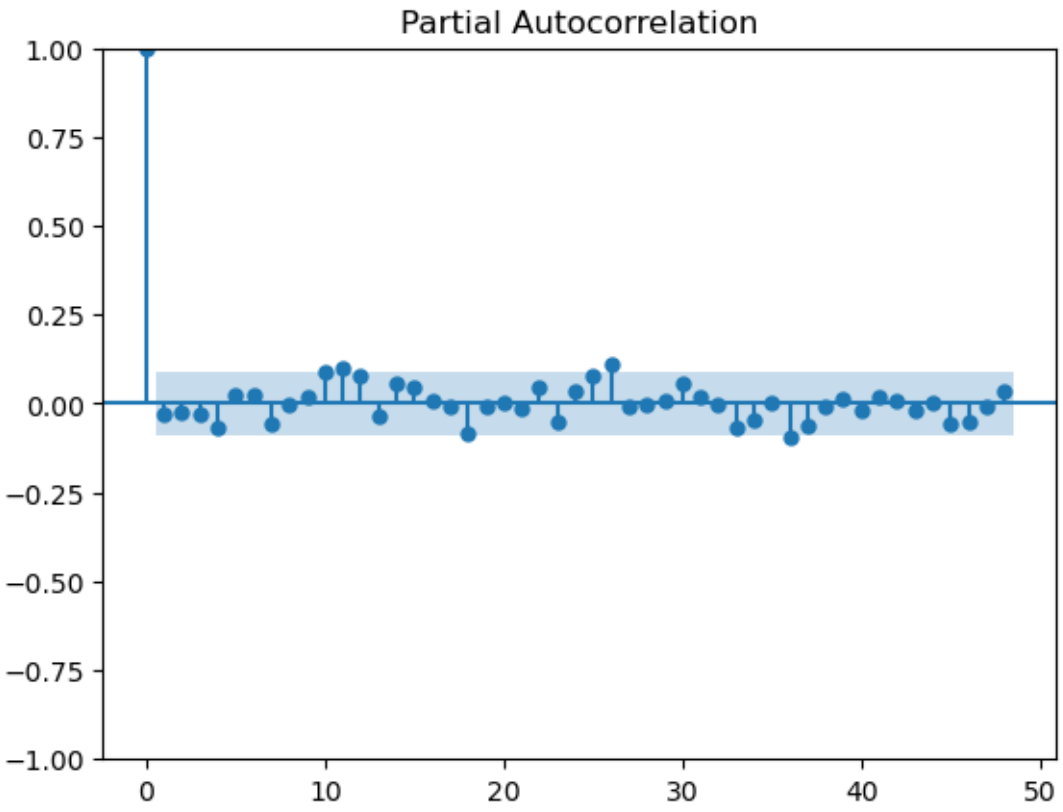
Other than the first ten years, the residuals look stationary in mean, but have higher variance at the peaks of the sunspot cycle. The first ten years should not be predicted with the specificed SARIMA [132] since past seasons wouldn't have been observed.

Thus we drop the first 132 observations and re-review residuals.

[ ]:
```
#Plot ACF and PACF of differenced data
plot_acf(model_arima_d2.resid[132:],  lags = 48)
plot_pacf(model_arima_d2.resid[132:], lags = 48)
plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```

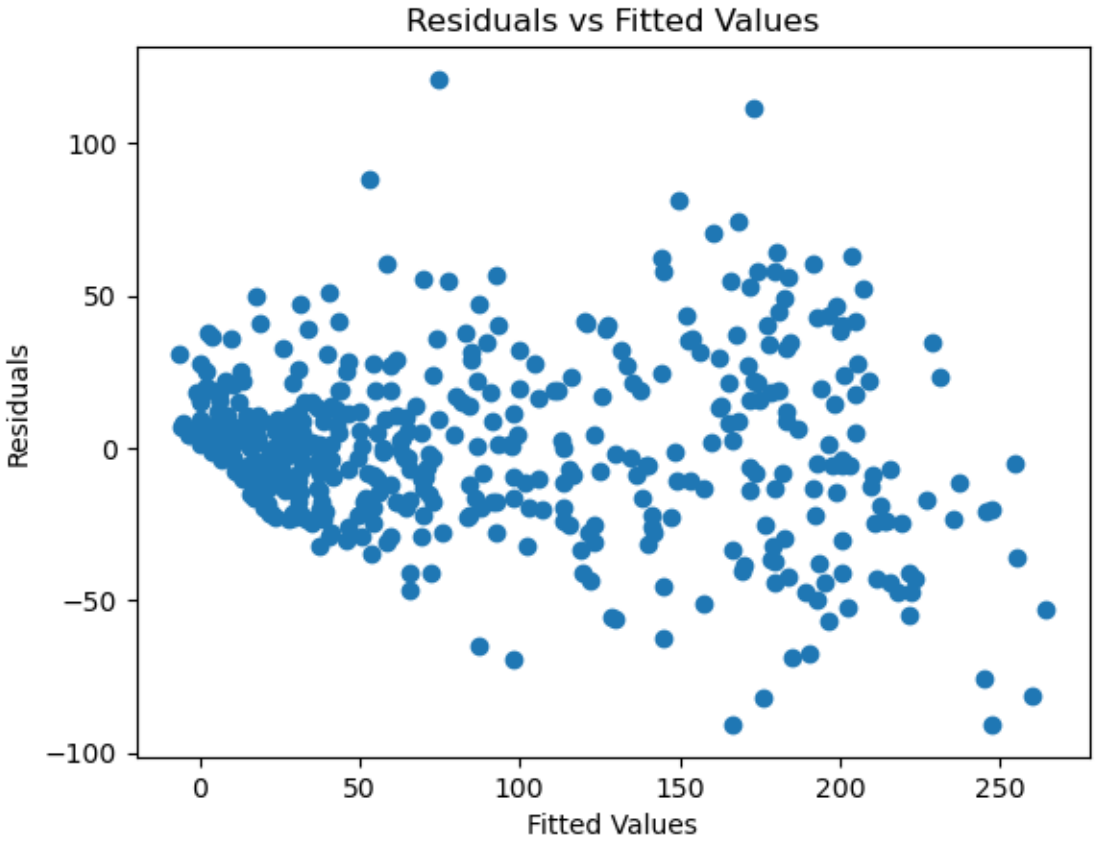

Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50

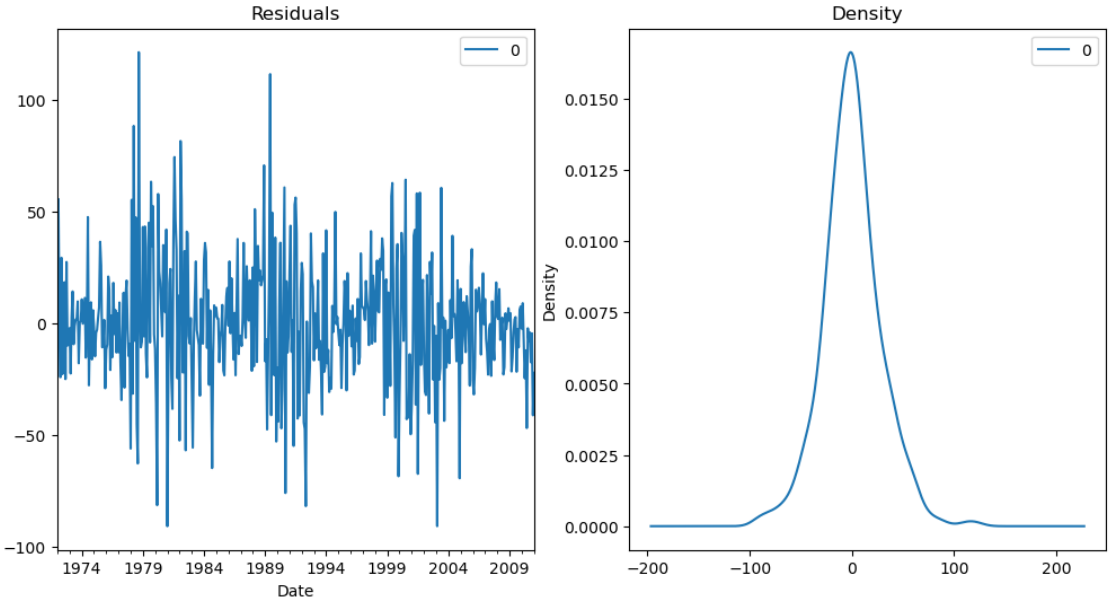


Now this looks more like white noise.

[ ]:
```
# Plotting residuals vs fitted values

# Extracting residuals
residuals = pd.DataFrame(model_arima_d2.resid[132:])

# Extracting fitted values
fv = pd.DataFrame(model_arima_d2.fittedvalues[132:])
plt.scatter(fv, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted Values')
plt.show()

# More diagnostics

fig, ax = plt.subplots(1, 2, figsize = (12,6))

residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
```

```
plt.show()
print(residuals.describe())
```





```
                0
count  469.000000
mean     0.229489
std     28.406280
min    -90.738230
25%    -16.584624
50%     -0.370576
75%     15.786417
max    121.154796
```

Conclusion and Future Work

The key takeaway from this analysis is that more advanced, state-of-the-art models like LSTM, Prophet work better for data as challenging as this one. We saw how much traditional models like ARIMA and Seasonal Naive need to be modified to get them to perform well.

Some potential improvements for the future could be:
- Smoothing the series: By using a 3-month rolling mean of the data (for example) or State Space Models that help to extract the signal from the noise
- GARCH: Modelling the raw data as a GARCH process to predict volatility
- Larger sample size with data from earlier periods to train our models

References

- Kaggle. (2021). Sunspots [Data set]. https://www.kaggle.com/datasets/robervalt/sunspots
- Kurnia, D. (2019). Time series Sunspots. https://rpubs.com/danielkurnia/timeseries_sunspots
- Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice. OTexts. https://otexts.com/fpp3/
- Rojas, R. (2023). Econ 412, Course Material. Retrieved from BruinLearn
- Kunz, N. (2023). Econ 412, Course Material. Retrieved from BruinLearn