Patrycja Paradowska, 244952

Prophet and Neural Prophet

1. Prophet

Apply Prophet, SARIMA, and Holt-Winters' exponential smoothing to the Google trends vacation data set. The csv file is available on ePortal. Predict 12 months in the future. Don't forget to use monthly time series.

In [29]:
```
import matplotlib.pyplot as plt
import pandas as pd
from fbprophet import Prophet
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.api import ExponentialSmoothing, Holt
from pmdarima import auto_arima
import warnings
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
from statsmodels.tools.eval_measures import rmse
warnings.filterwarnings("ignore")
```

In [30]:
```
df=pd.read_csv('vacation.csv', parse_dates=True)
df
```

Out[30]:

	Month	Num_Search_Vacation
0	2004-01-01	94
1	2004-02-01	89
2	2004-03-01	86
3	2004-04-01	79
4	2004-05-01	89
...	...	...
185	2019-06-01	58
186	2019-07-01	56
187	2019-08-01	45
188	2019-09-01	38
189	2019-10-01	37

190 rows × 2 columns

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 190 entries, 0 to 189
Data columns (total 2 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Month                190 non-null    object
 1   Num_Search_Vacation  190 non-null    int64
dtypes: int64(1), object(1)
memory usage: 3.1+ KB
```

In [32]:
```
from statsmodels.tsa.stattools import adfuller
adfuller(df['Num_Search_Vacation'])
```

Out[32]:
```
(-2.336474118969219,
 0.16047397049001244,
 12,
 177,
 {'1%': -3.467845319799907,
  '5%': -2.878011745497439,
  '10%': -2.575551186759871},
 885.8015084229178)
```

In [33]:
```
df.columns = ['ds', 'y']
```

In [34]:
```
df['ds'] = pd.to_datetime(df['ds'])
```

In [35]:
```
df
```

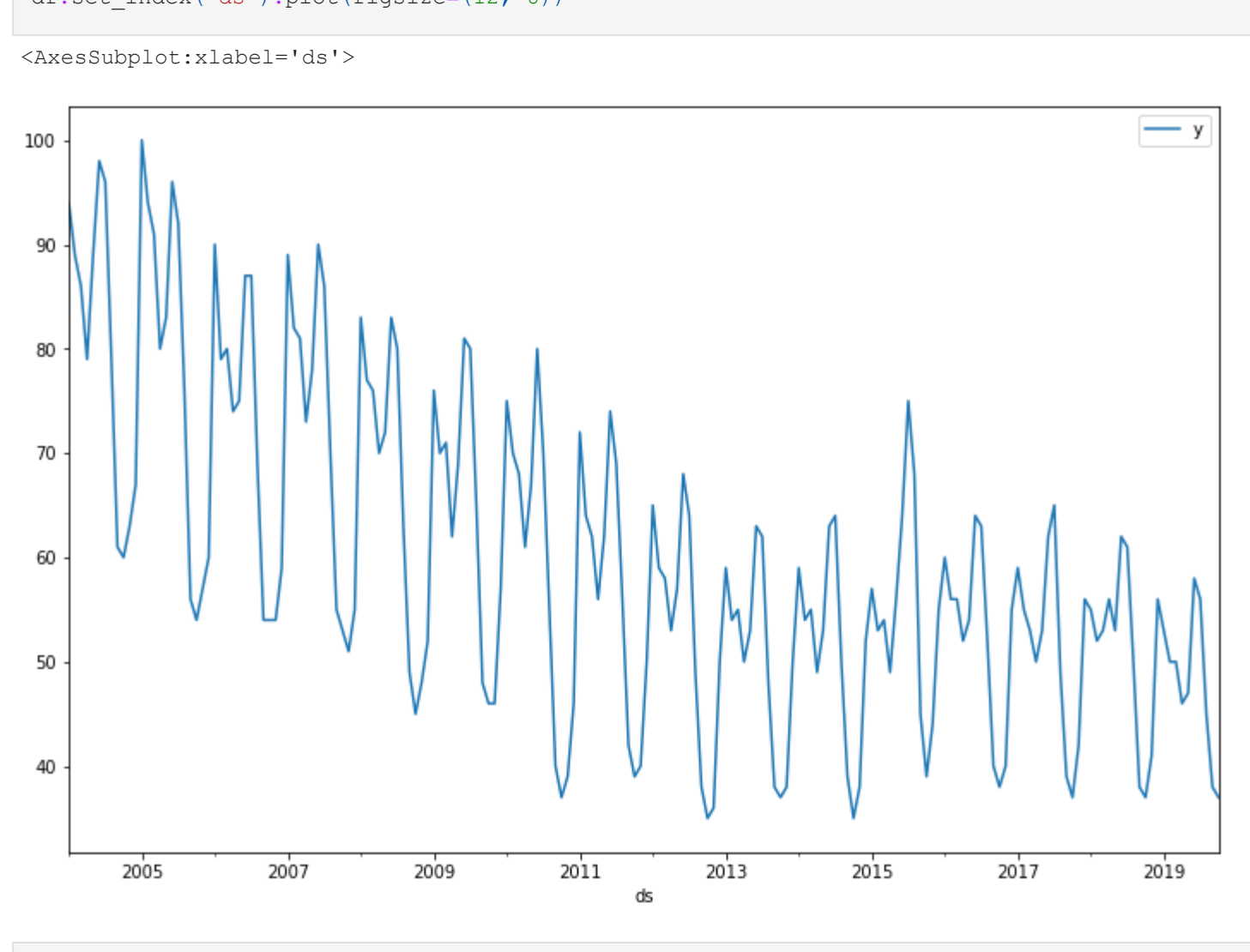
Out[35]:

	ds	y
0	2004-01-01	94
1	2004-02-01	89
2	2004-03-01	86
3	2004-04-01	79
4	2004-05-01	89
...	...	...
185	2019-06-01	58
186	2019-07-01	56
187	2019-08-01	45
188	2019-09-01	38
189	2019-10-01	37

190 rows × 2 columns

In [36]:
```
df.set_index('ds').plot(figsize=(12, 8))
```

Out[36]: <AxesSubplot:xlabel='ds'>



In [37]:
```
len(df)
```

Out[37]: 190

In [38]:
```
train, test = df[0:-12], df[-12:]
print("Train set size:", len(train), "\nTest set size:", len(test))
```

```
Train set size: 178
Test set size: 12
```

In [41]:
```
m = Prophet()
m.fit(train)
```

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

Out[41]: <fbprophet.forecaster.Prophet at 0x24a37814e88>

In [42]:
```
future = m.make_future_dataframe(periods = 12, freq = 'MS')
```

In [43]:
```
future
```

Out[43]:

	ds
0	2004-01-01
1	2004-02-01
2	2004-03-01
3	2004-04-01
4	2004-05-01
...	...
185	2019-06-01
186	2019-07-01
187	2019-08-01
188	2019-09-01
189	2019-10-01

190 rows × 1 columns

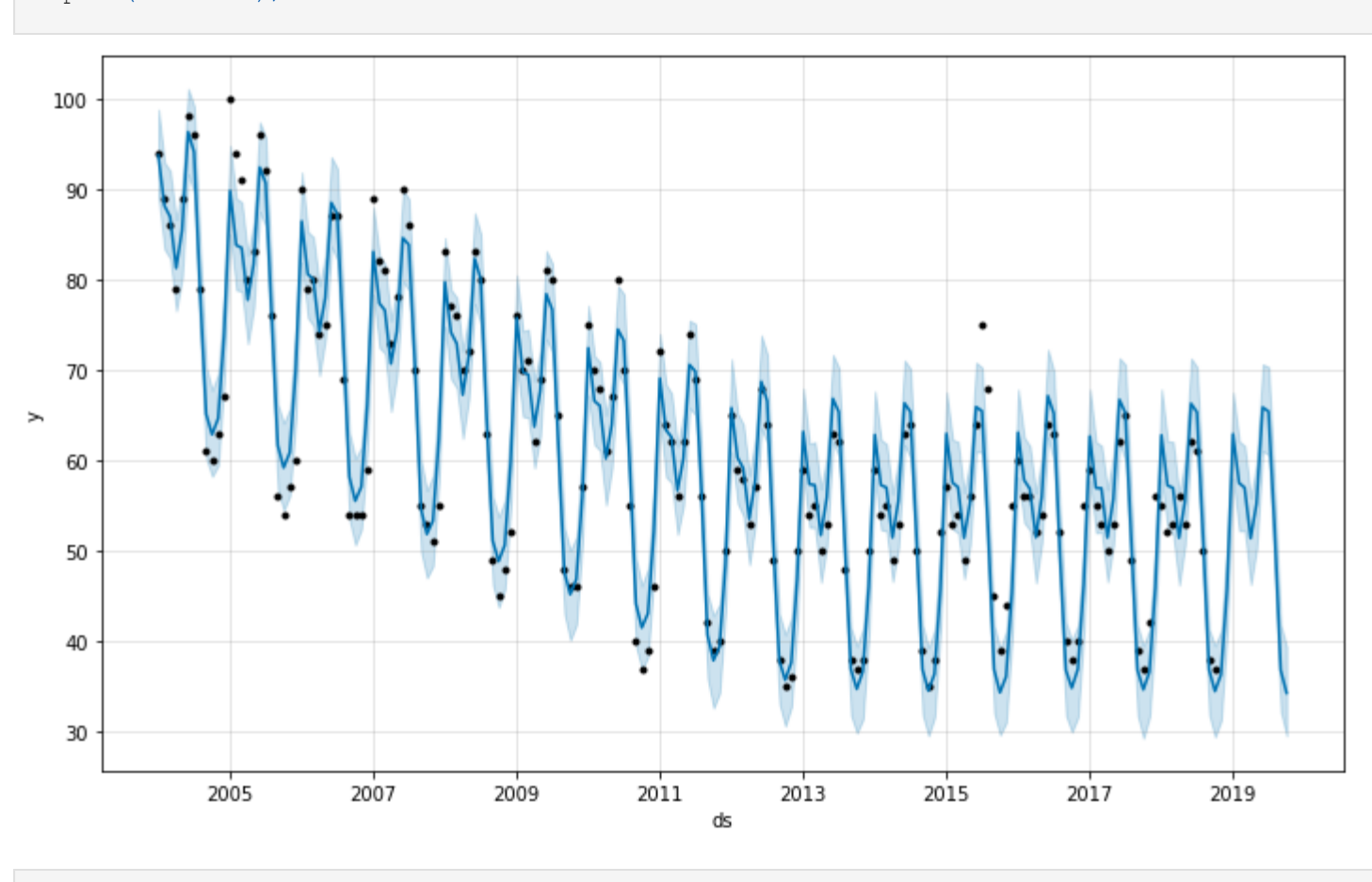
In [44]:
```
forecast = m.predict(future)
forecast
```

Out[44]:

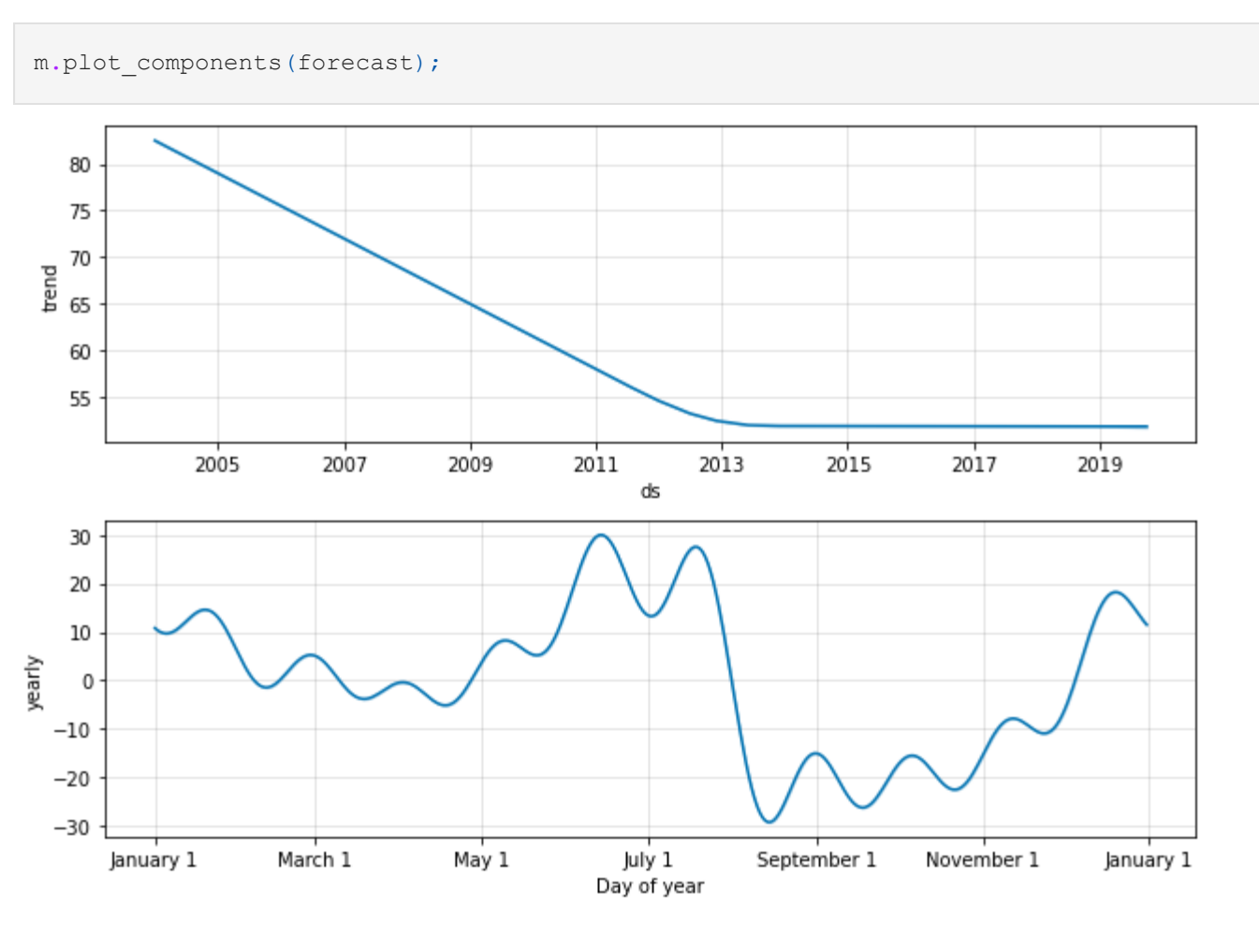
	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	yearl
0	2004-01-01	82.502445	89.089835	98.768631	82.502445	82.502445	11.201315	11.201315	11.201315	11.20131
1	2004-02-01	82.205215	83.324934	92.932877	82.205215	82.205215	5.939019	5.939019	5.939019	5.93901
2	2004-03-01	81.927161	82.317757	91.997286	81.927161	81.927161	4.989236	4.989236	4.989236	4.98923
3	2004-04-01	81.629931	76.532691	86.587569	81.629931	81.629931	-0.388946	-0.388946	-0.388946	-0.38894
4	2004-05-01	81.342289	80.202191	90.298266	81.342289	81.342289	4.079325	4.079325	4.079325	4.07932
...	...	...	...	...	...	...	...	...	...	
185	2019-06-01	51.829643	61.043549	70.623177	51.760563	51.899560	14.016379	14.016379	14.016379	14.01637
186	2019-07-01	51.828721	60.597527	70.307135	51.744242	51.914465	13.568549	13.568549	13.568549	13.56854
187	2019-08-01	51.827768	47.447702	57.133378	51.721502	51.934106	0.651789	0.651789	0.651789	0.65178
188	2019-09-01	51.826815	32.216854	41.796543	51.698284	51.957344	-14.999750	-14.999750	-14.999750	-14.99975
189	2019-10-01	51.825893	29.551407	39.396414	51.676797	51.974065	-17.541870	-17.541870	-17.541870	-17.54187

190 rows × 16 columns

In [45]:
```
m.plot(forecast);
```



In [46]:
```
m.plot_components(forecast);
```



SARIMA

In [9]:
```
df2 = pd.read_csv("vacation.csv", index_col = 'Month', parse_dates = True)
```

In [10]:
```
df2
```

Out[10]:

Month	Num_Search_Vacation
2004-01-01	94
2004-02-01	89
2004-03-01	86
2004-04-01	79
2004-05-01	89
...	...
2019-06-01	58
2019-07-01	56
2019-08-01	45
2019-09-01	38
2019-10-01	37

190 rows × 1 columns

In [11]:
```
df2_train, df2_test = df2[0:-12], df2[-12:]
```

In [12]:
```
start=len(df2_train)
end=len(df2_train)+len(df2_test)-1
```

In [13]:
```
df2_train
```

Out[13]:

Month	Num_Search_Vacation
2004-01-01	94
2004-02-01	89
2004-03-01	86
2004-04-01	79
2004-05-01	89
...	...
2018-06-01	62
2018-07-01	61
2018-08-01	50
2018-09-01	38
2018-10-01	37

178 rows × 1 columns

In [14]:
```
len(df2_test)
```

Out[14]: 12

In [17]:
```
auto_arima(df2, seasonal=True, m=12, trace=True).summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,0,1)[12] intercept   : AIC=984.654, Time=0.99 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=1402.455, Time=0.01 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=inf, Time=0.20 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=1254.545, Time=0.17 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=1400.635, Time=0.01 sec
 ARIMA(2,1,2)(0,0,1)[12] intercept   : AIC=inf, Time=0.69 sec
 ARIMA(2,1,2)(1,0,0)[12] intercept   : AIC=inf, Time=0.75 sec
 ARIMA(2,1,2)(2,0,1)[12] intercept   : AIC=1006.205, Time=1.89 sec
 ARIMA(2,1,2)(1,0,2)[12] intercept   : AIC=980.473, Time=1.72 sec
 ARIMA(2,1,2)(0,0,2)[12] intercept   : AIC=inf, Time=1.37 sec
 ARIMA(2,1,2)(2,0,2)[12] intercept   : AIC=inf, Time=1.96 sec
 ARIMA(1,1,2)(1,0,2)[12] intercept   : AIC=979.429, Time=1.56 sec
 ARIMA(1,1,2)(0,0,2)[12] intercept   : AIC=inf, Time=1.23 sec
 ARIMA(1,1,2)(1,0,1)[12] intercept   : AIC=inf, Time=0.95 sec
 ARIMA(1,1,2)(2,0,2)[12] intercept   : AIC=inf, Time=1.86 sec
 ARIMA(1,1,2)(0,0,1)[12] intercept   : AIC=inf, Time=0.57 sec
 ARIMA(1,1,2)(2,0,1)[12] intercept   : AIC=979.883, Time=1.37 sec
 ARIMA(0,1,2)(1,0,2)[12] intercept   : AIC=983.571, Time=1.00 sec
 ARIMA(1,1,1)(1,0,2)[12] intercept   : AIC=976.697, Time=1.47 sec
 ARIMA(1,1,1)(0,0,2)[12] intercept   : AIC=inf, Time=1.02 sec
 ARIMA(1,1,1)(1,0,1)[12] intercept   : AIC=978.359, Time=0.70 sec
 ARIMA(1,1,1)(2,0,2)[12] intercept   : AIC=inf, Time=1.64 sec
 ARIMA(1,1,1)(0,0,1)[12] intercept   : AIC=1256.388, Time=0.19 sec
 ARIMA(1,1,1)(2,0,1)[12] intercept   : AIC=977.644, Time=1.61 sec
 ARIMA(0,1,1)(1,0,2)[12] intercept   : AIC=990.860, Time=0.58 sec
 ARIMA(1,1,0)(1,0,2)[12] intercept   : AIC=996.227, Time=0.49 sec
 ARIMA(2,1,1)(1,0,2)[12] intercept   : AIC=inf, Time=1.59 sec
 ARIMA(0,1,0)(1,0,2)[12] intercept   : AIC=1001.651, Time=0.41 sec
 ARIMA(2,1,0)(1,0,2)[12] intercept   : AIC=989.692, Time=0.61 sec
 ARIMA(1,1,1)(1,0,2)[12]             : AIC=975.306, Time=0.63 sec
 ARIMA(1,1,1)(0,0,2)[12]             : AIC=1157.847, Time=0.39 sec
 ARIMA(1,1,1)(1,0,1)[12]             : AIC=976.861, Time=0.31 sec
 ARIMA(1,1,1)(2,0,2)[12]             : AIC=inf, Time=1.39 sec
 ARIMA(1,1,1)(0,0,1)[12]             : AIC=1254.455, Time=0.11 sec
 ARIMA(1,1,1)(2,0,1)[12]             : AIC=976.130, Time=0.54 sec
 ARIMA(0,1,1)(1,0,2)[12]             : AIC=988.860, Time=0.51 sec
 ARIMA(1,1,0)(1,0,2)[12]             : AIC=994.227, Time=0.39 sec
 ARIMA(2,1,1)(1,0,2)[12]             : AIC=inf, Time=2.30 sec
 ARIMA(1,1,2)(1,0,2)[12]             : AIC=976.935, Time=0.91 sec
 ARIMA(0,1,0)(1,0,2)[12]             : AIC=999.651, Time=0.22 sec
 ARIMA(0,1,2)(1,0,2)[12]             : AIC=981.582, Time=0.56 sec
 ARIMA(2,1,0)(1,0,2)[12]             : AIC=987.692, Time=0.57 sec
 ARIMA(2,1,2)(1,0,2)[12]             : AIC=1000.524, Time=1.26 sec

Best model:  ARIMA(1,1,1)(1,0,2)[12]
Total fit time: 38.752 seconds
```

Out[17]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	190
Model:	SARIMAX(1, 1, 1)x(1, 0, [1, 2], 12)	**Log Likelihood**	-481.653
Date:	Mon, 16 May 2022	**AIC**	975.306
Time:	13:00:12	**BIC**	994.757
Sample:	0	**HQIC**	983.186
	- 190		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.6628	0.058	11.367	0.000	0.549	0.777
ma.L1	-0.9636	0.029	-32.979	0.000	-1.021	-0.906
ar.S.L12	0.9751	0.009	108.551	0.000	0.957	0.993
ma.S.L12	-0.4194	0.071	-5.934	0.000	-0.558	-0.281
ma.S.L24	0.1426	0.069	2.074	0.038	0.008	0.277
sigma2	8.1509	0.586	13.915	0.000	7.003	9.299

Ljung-Box (L1) (Q):	0.05	**Jarque-Bera (JB):**	59.16
Prob(Q):	0.82	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.59	**Skew:**	0.29
Prob(H) (two-sided):	0.07	**Kurtosis:**	5.68

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

In [18]:
```
modelSarimax = SARIMAX(df2_train, order=(1,1,1), seasonal_order=(1,0,2,12), enforce_invertibility=False, enforc
fitted = modelSarimax.fit()
fitted.summary()
```

Out[18]:

SARIMAX Results

Dep. Variable:	Num_Search_Vacation	**No. Observations:**	178
Model:	SARIMAX(1, 1, 1)x(1, 0, [1, 2], 12)	**Log Likelihood**	-368.628
Date:	Mon, 16 May 2022	**AIC**	749.256
Time:	13:00:29	**BIC**	767.359
Sample:	01-01-2004	**HQIC**	756.610
	- 10-01-2018		
Covariance Type:	opg		

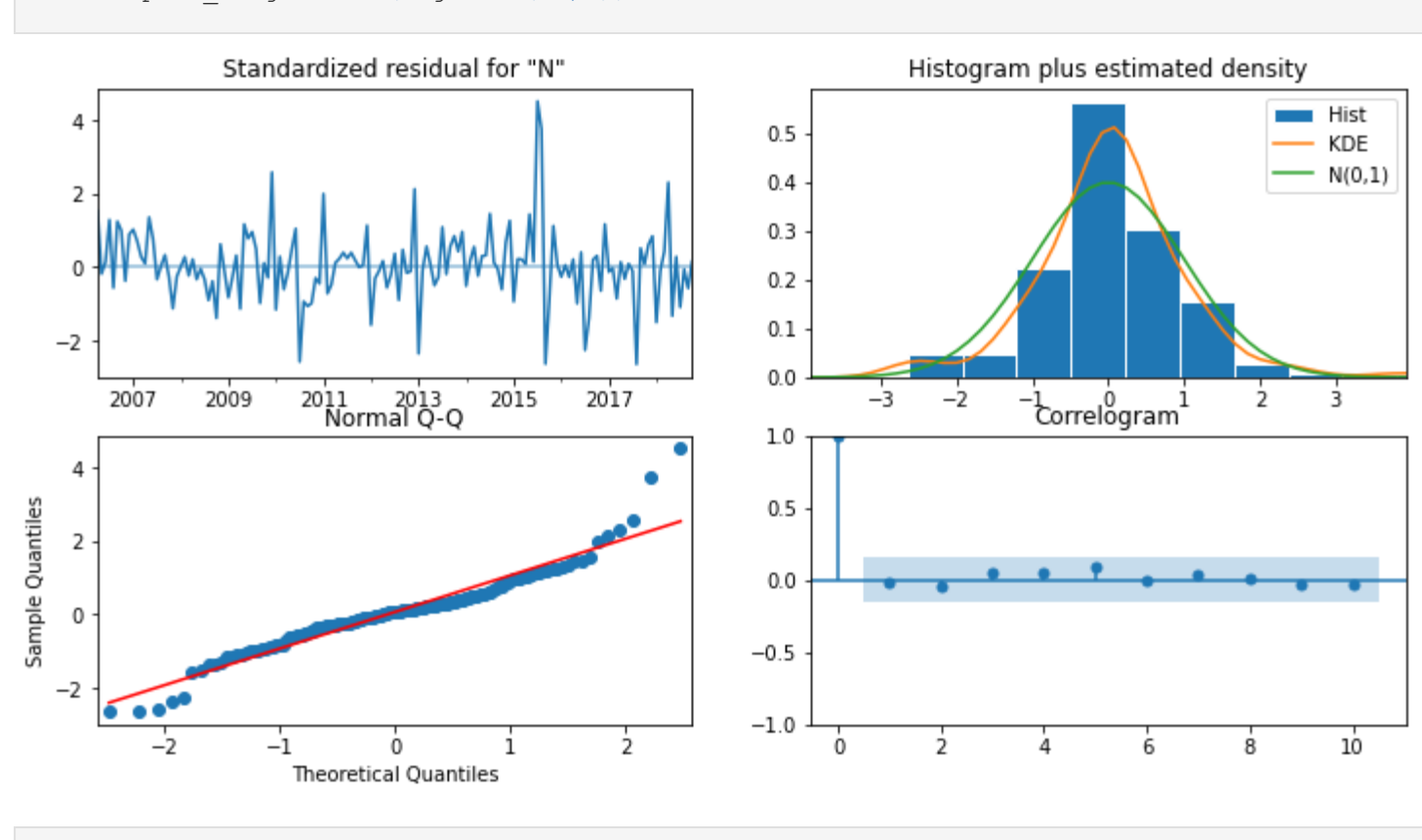
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.5774	0.108	5.329	0.000	0.365	0.790
ma.L1	-0.9143	0.073	-12.580	0.000	-1.057	-0.772
ar.S.L12	0.9339	0.017	54.525	0.000	0.900	0.967
ma.S.L12	-0.4841	0.087	-5.565	0.000	-0.655	-0.314
ma.S.L24	0.0950	0.078	1.224	0.221	-0.057	0.247
sigma2	7.5169	0.547	13.743	0.000	6.445	8.589

Ljung-Box (L1) (Q):	0.08	**Jarque-Bera (JB):**	88.94
Prob(Q):	0.78	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.47	**Skew:**	0.60
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.56

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

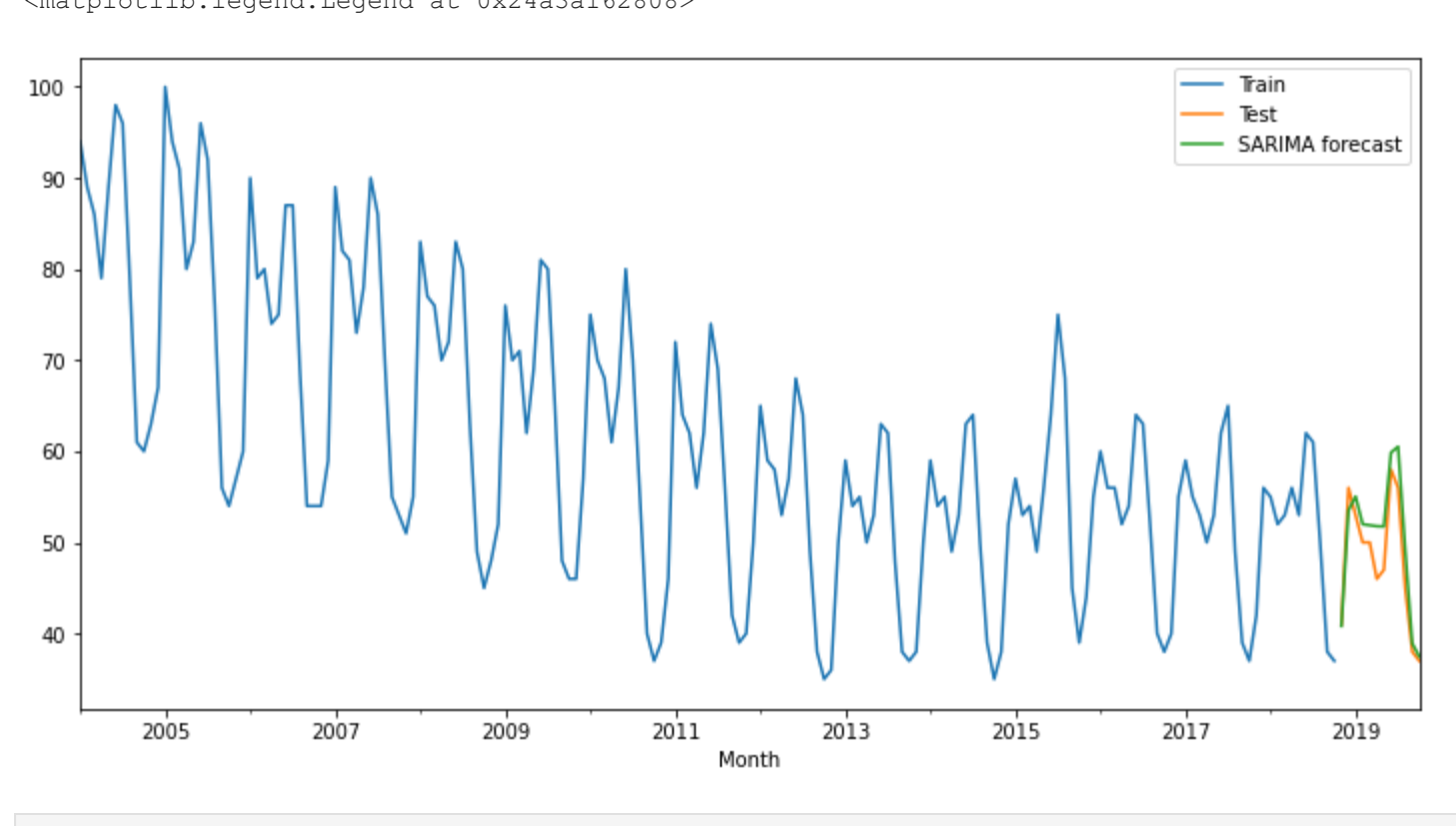
In [20]:
```
pred_Sarimax=fitted.predict(start=start, end=end,dynamic=False)
```

In [21]:
```
fitted.plot_diagnostics(figsize=(12,6));
```



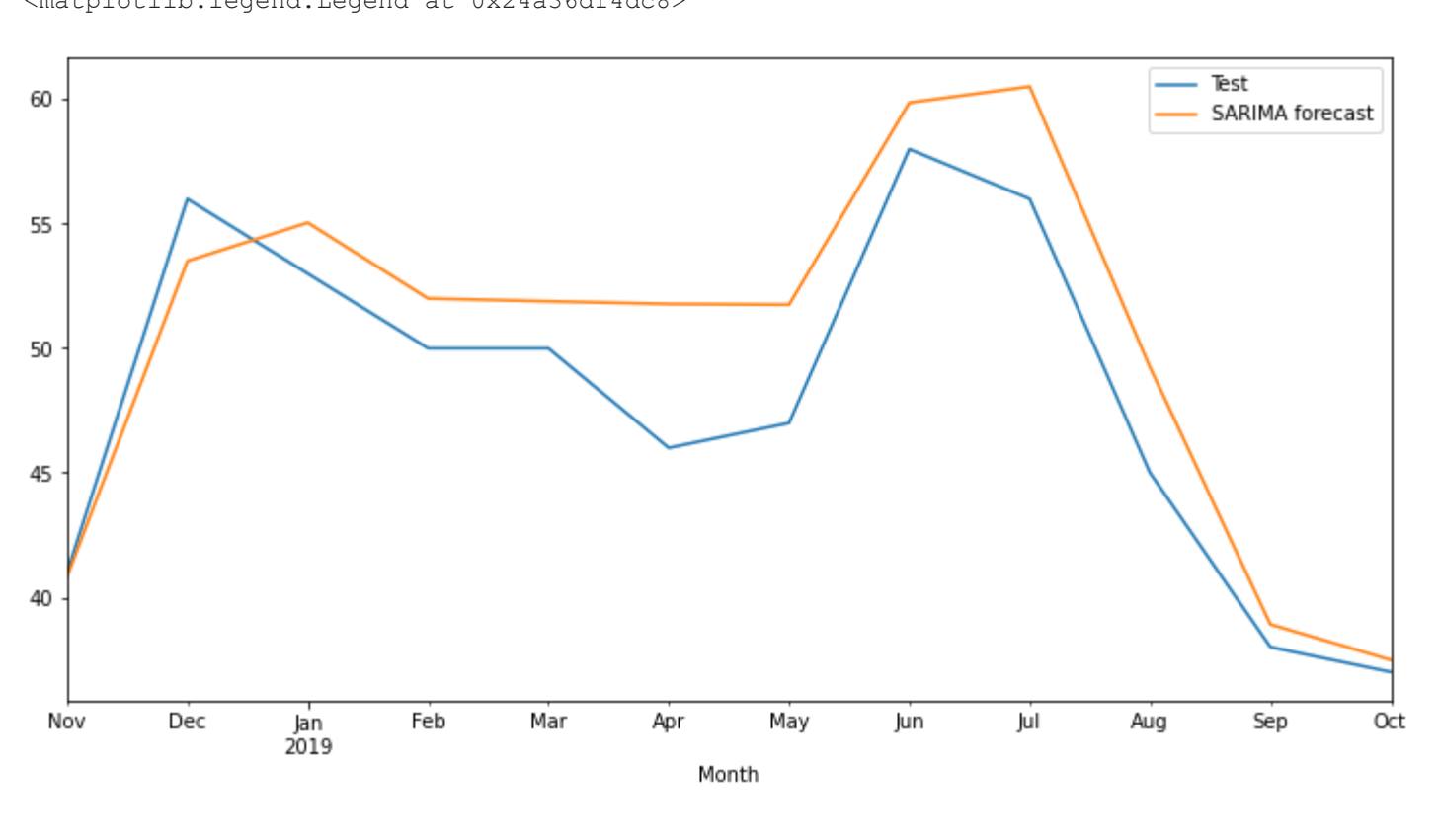
In [22]:
```
ax = df2_train.plot(figsize=(12,6))
df2_test.plot(ax=ax)
pred_Sarimax.plot(ax=ax)
plt.legend(['Train', 'Test', 'SARIMA forecast'])
```

Out[22]: <matplotlib.legend.Legend at 0x24a3a162808>



In [24]:
```
ax = df2_test.plot(figsize=(12,6))
pred_Sarimax.plot(ax=ax)
plt.legend(['Test', 'SARIMA forecast'])
```

Out[24]: <matplotlib.legend.Legend at 0x24a36df4dc8>



Holt-Winters' exponential smoothing

In [25]:
```
add_add = ExponentialSmoothing(df2_train, trend="add", seasonal="add", seasonal_periods=12).fit()
add_mull = ExponentialSmoothing(df2_train, trend="add", seasonal="mul",seasonal_periods=12).fit()
mull_add = ExponentialSmoothing(df2_train, trend="mul", seasonal="add", seasonal_periods=12).fit()
mull_mull = ExponentialSmoothing(df2_train, trend="mul", seasonal="mul", seasonal_periods=12).fit()
```

In [26]:
```
add_add_forecast = add_add.forecast(len(df2_test))
add_mull_forecast = add_mull.forecast(len(df2_test))
mull_add_forecast = mull_add.forecast(len(df2_test))
mull_mull_forecast = mull_mull.forecast(len(df2_test))
```

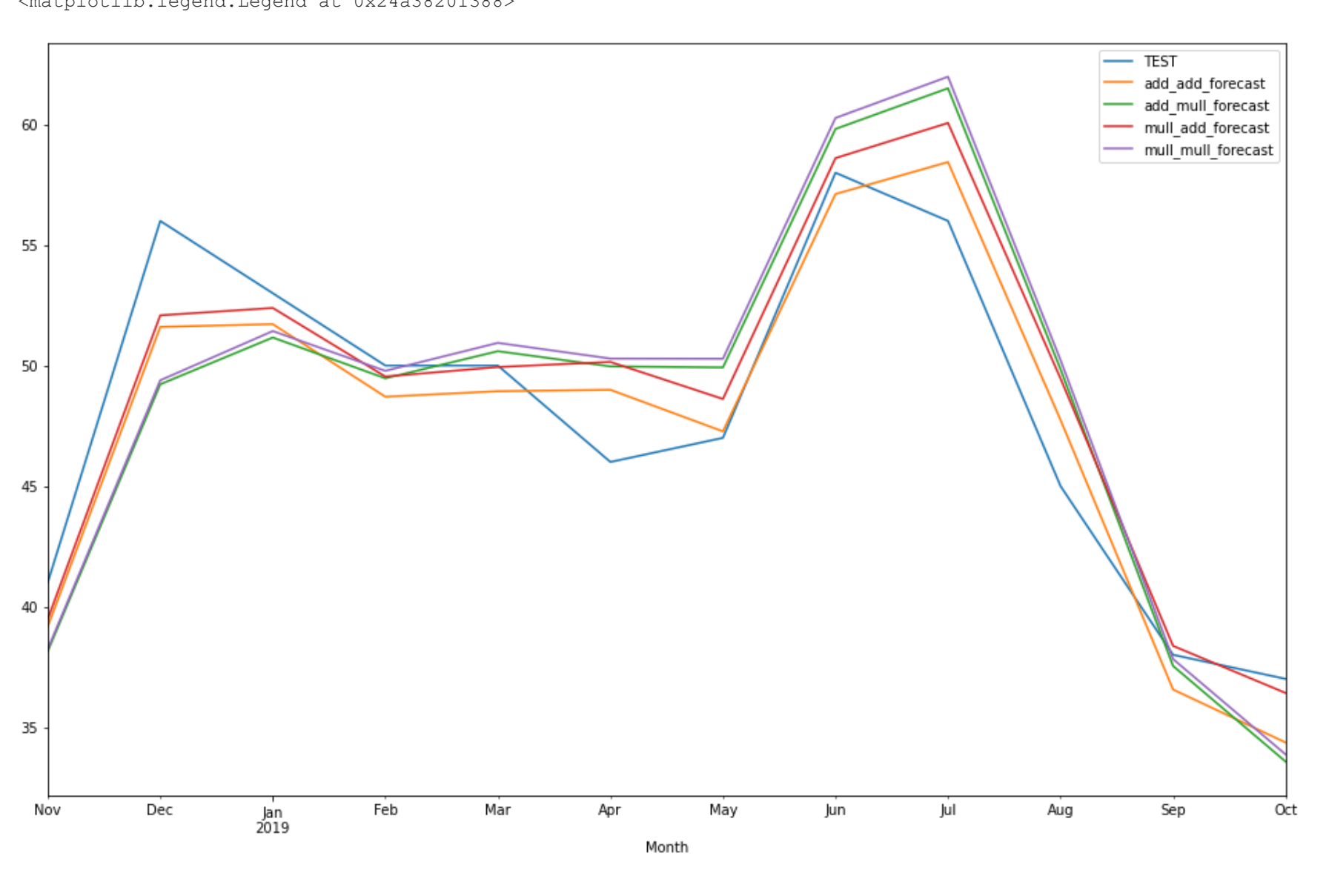
In [27]:
```
mean_squared_errors = [
    (mean_squared_error(df2_test, add_add_forecast)),
    (mean_squared_error(df2_test, add_mull_forecast)),
    (mean_squared_error(df2_test, mull_add_forecast)),
    (mean_squared_error(df2_test, mull_mull_forecast))
]

mean_squared_errors
```

Out[27]: [4.952623407325601, 12.614135770163188, 6.2756507014344765, 13.50500046631454]

In [53]:
```
ax = df2_test.plot(figsize=(16, 10), label = "TEST")
add_add_forecast.plot(ax=ax, label="add_add_forecast")
add_mull_forecast.plot(ax=ax, label="add_mull_forecast")
mull_add_forecast.plot(ax=ax, label="mull_add_forecast")
mull_mull_forecast.plot(ax=ax, label="mull_mull_forecast")
ax.legend(['TEST','add_add_forecast',
           'add_mull_forecast',
           'mull_add_forecast',
           'mull_mull_forecast'])
```

Out[53]: <matplotlib.legend.Legend at 0x24a38201388>

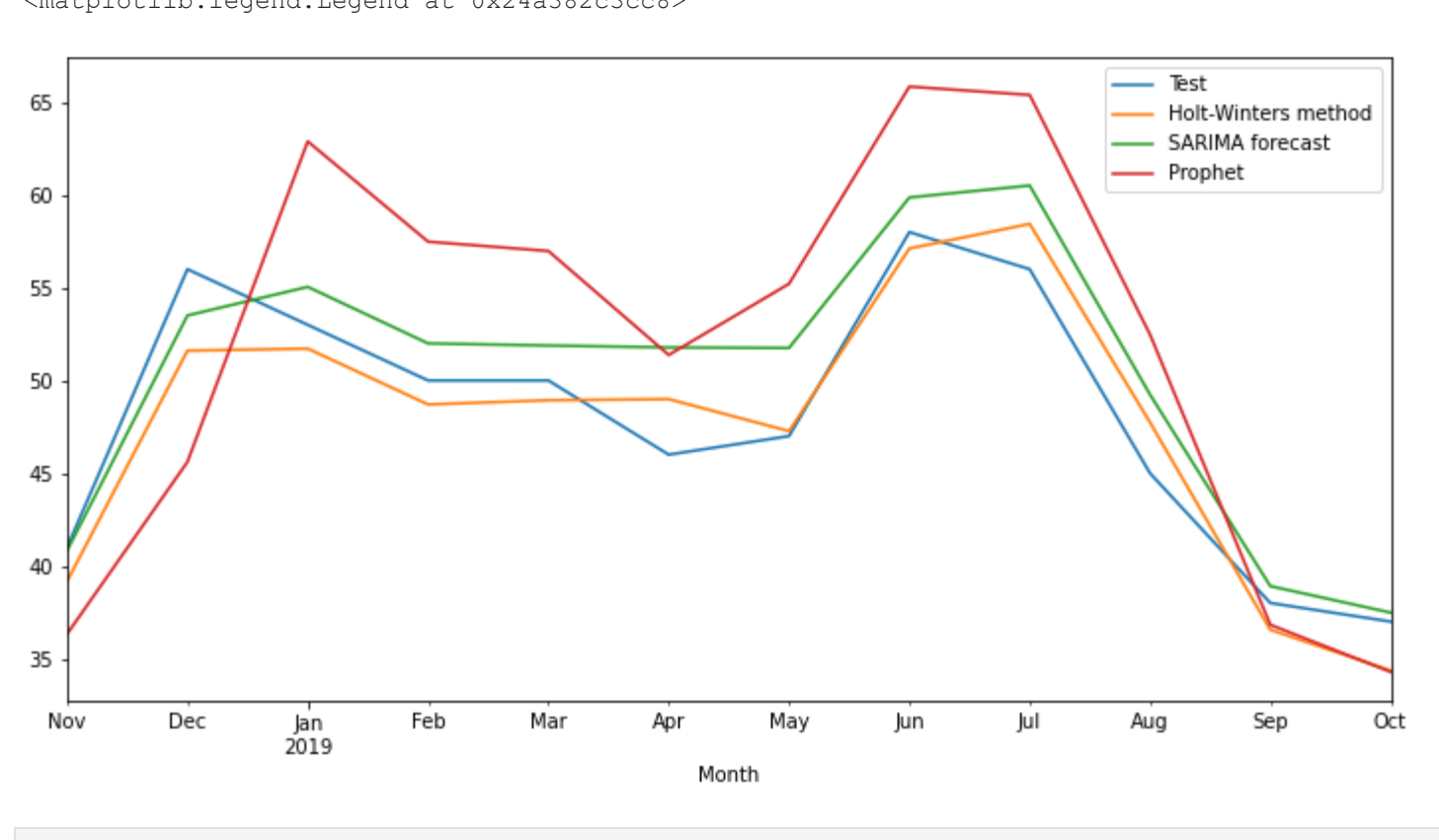


So the best result gives us add_add_forecast (4.952623407325601).

Comparison of the results:

In [56]:
```
prop = pd.Series(data=forecast["yhat"])
prop_df = prop.to_frame()
index = pd.date_range(start = "2004-01-01", end = "2019-11-01", freq = "M")
prop_df = prop_df.set_index(index).iloc[-12:]
ax = df2_test.plot(figsize=(12,6))
add_add_forecast.plot(ax=ax)
pred_Sarimax.plot(ax=ax)
prop_df["yhat"].plot(ax=ax)
plt.legend(['Test','Holt-Winters method', 'SARIMA forecast', 'Prophet'])
```

Out[56]: <matplotlib.legend.Legend at 0x24a382c3cc8>



In [57]:
```
error=mean_squared_error(df2_test,prop_df['yhat'][-12:])
print("PROPHET: ", error)
error=mean_squared_error(df2_test, pred_Sarimax)
print("SARIMA: ", error)
error=mean_squared_error(df2_test, add_add_forecast)
print("Holt-Winters' exponential smoothing: ", error)
```

```
PROPHET:  53.60896252764628
SARIMA:  9.739922997734785
Holt-Winters' exponential smoothing:  4.952623407325601
```

So the best result gives us Holt-Winters' exponential smoothing (add_add_forecast).

2. Neural Prophet (bonus problem)

Neural Prophet is a successor of Prophet. Compare the accuracy of these models for the vacation data set.

In [428...
```
from neuralprophet import NeuralProphet
```

In [429...
```
df=pd.read_csv('vacation.csv',index_col='Month',parse_dates=True)
df.index.freq='MS'
df
```

Out[429...

Month	Num_Search_Vacation
2004-01-01	94
2004-02-01	89
2004-03-01	86
2004-04-01	79
2004-05-01	89
...	...
2019-06-01	58
2019-07-01	56
2019-08-01	45
2019-09-01	38
2019-10-01	37

190 rows × 1 columns

In [430...
```
neural_prophet_df = pd.DataFrame()
neural_prophet_df['ds'] = df.index
neural_prophet_df['y'] = df.values
```

In [431...
```
neural_prophet_df['ds'] = pd.to_datetime(neural_prophet_df['ds'])
```

In [432...
```
neural_prophet_df = neural_prophet_df.set_index(neural_prophet_df['ds'])
```

In [433...
```
neural_prophet_df
```

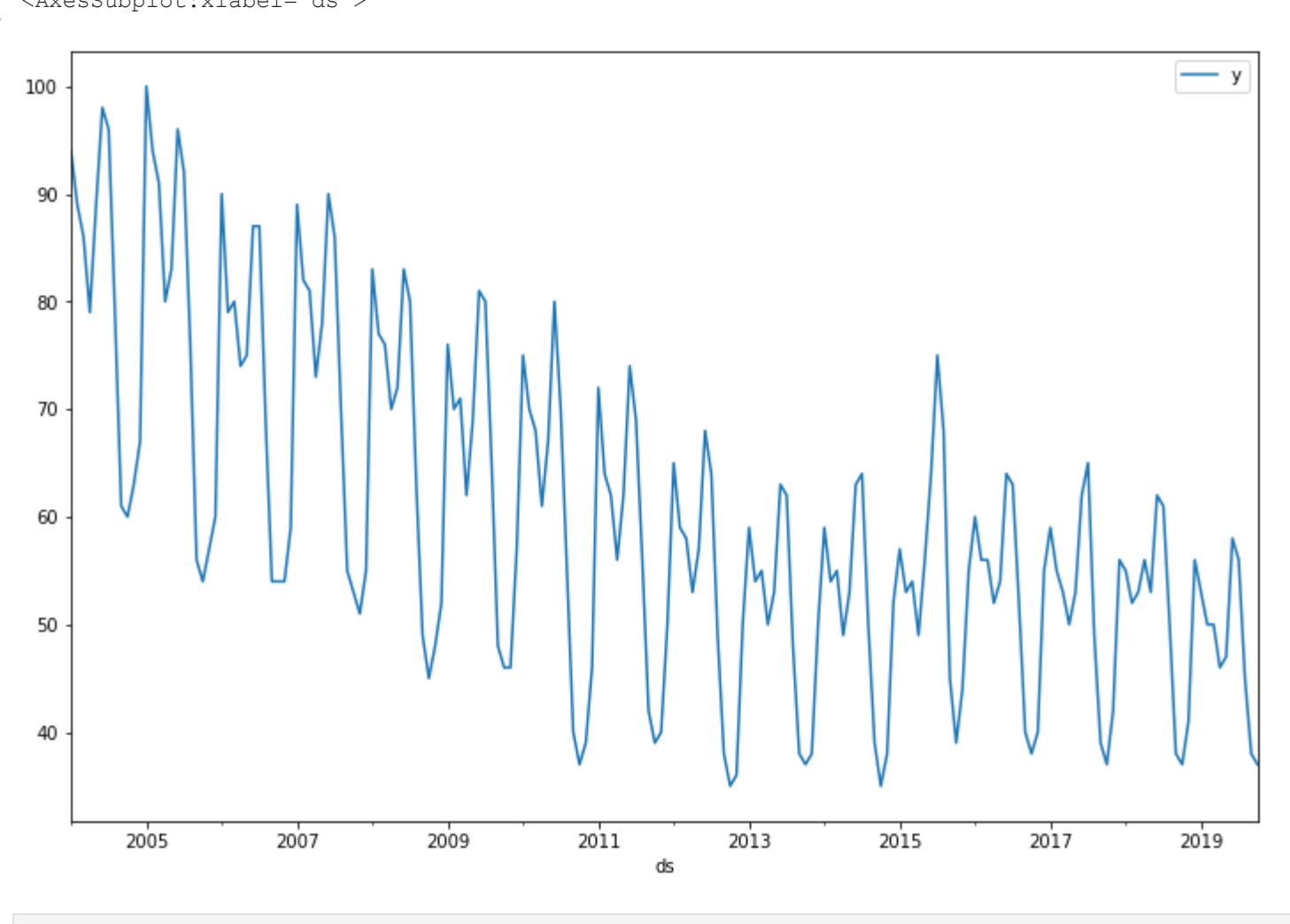
Out[433...

ds	ds	y
2004-01-01	2004-01-01	94
2004-02-01	2004-02-01	89
2004-03-01	2004-03-01	86
2004-04-01	2004-04-01	79
2004-05-01	2004-05-01	89
...	...	...
2019-06-01	2019-06-01	58
2019-07-01	2019-07-01	56
2019-08-01	2019-08-01	45
2019-09-01	2019-09-01	38
2019-10-01	2019-10-01	37

190 rows × 2 columns

In [434...
```
neural_prophet_df.set_index('ds').plot(figsize=(12, 8))
```

Out[434...: <AxesSubplot:xlabel='ds'>



In [435...
```
neural_prophet_df.index
```

Out[435...
```
DatetimeIndex(['2004-01-01', '2004-02-01', '2004-03-01', '2004-04-01',
               '2004-05-01', '2004-06-01', '2004-07-01', '2004-08-01',
               '2004-09-01', '2004-10-01',
               ...
               '2019-01-01', '2019-02-01', '2019-03-01', '2019-04-01',
               '2019-05-01', '2019-06-01', '2019-07-01', '2019-08-01',
               '2019-09-01', '2019-10-01'],
              dtype='datetime64[ns]', name='ds', length=190, freq=None)
```

In [436...
```
train_data = neural_prophet_df.iloc[:-12]
test_data = neural_prophet_df.iloc[-12:]
```

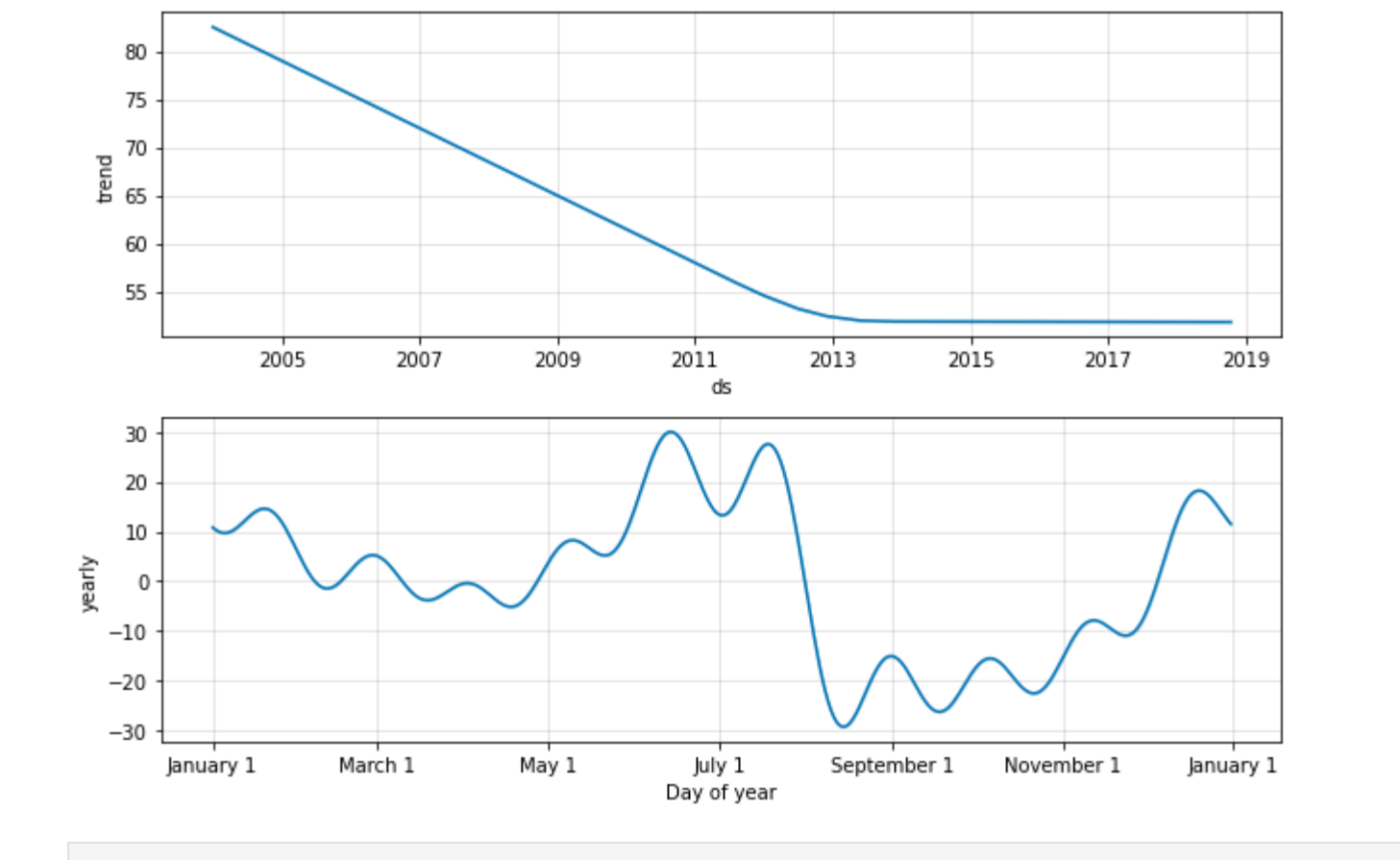
In [437...
```
print(len(test_data))
```

```
12
```

In [438...
```
prophet_model = Prophet()
prophet_model.fit(train_data)
future_df = prophet_model.make_future_dataframe(periods=12)
forecast_p = prophet_model.predict(future_df)
```

```
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.
```

In [439...
```
prophet_model.plot_components(forecast_p);
```



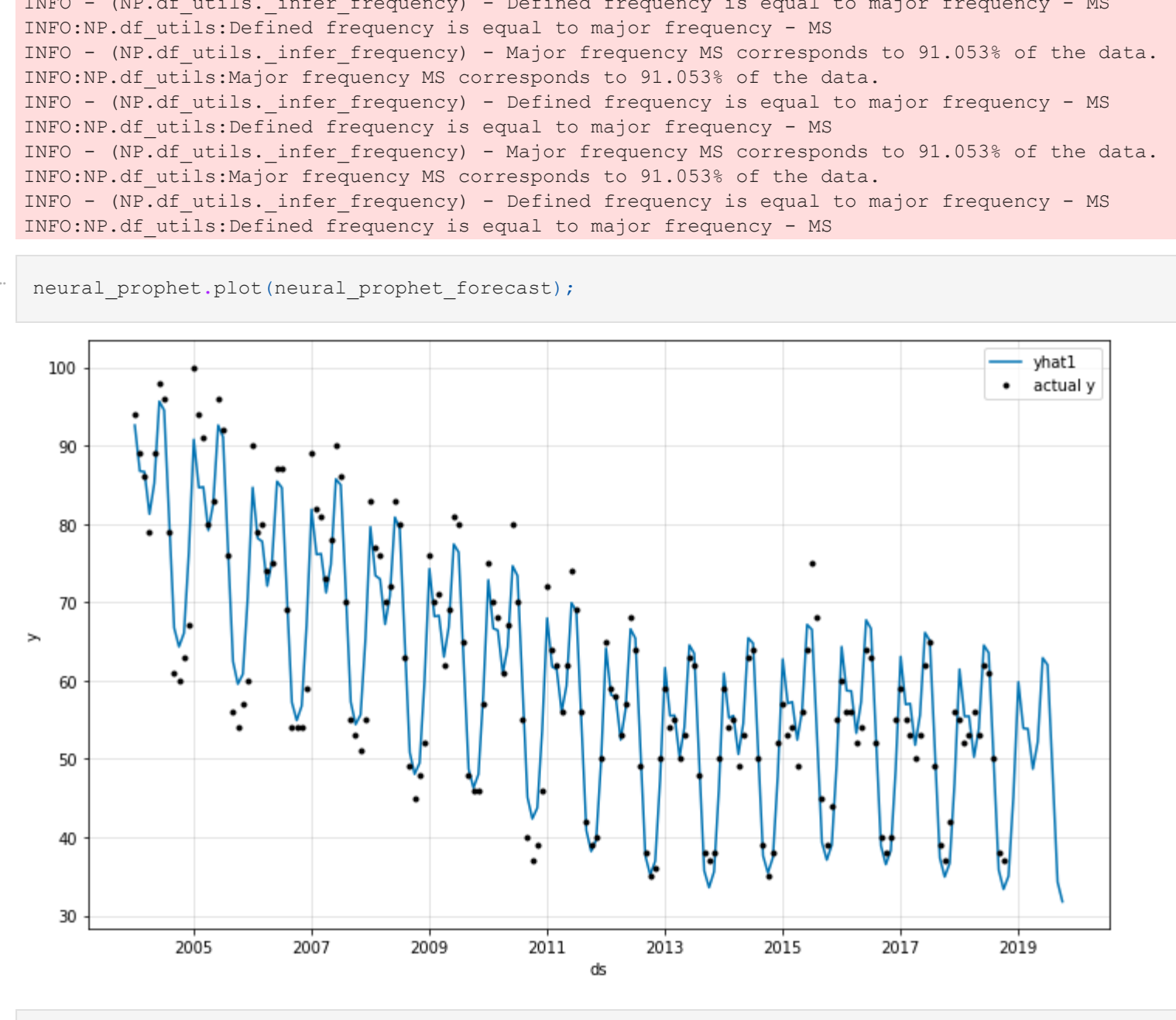
In [440]:
```python
neural_prophet = NeuralProphet()
neural_prophet_fit = neural_prophet.fit(train_data, freq='MS')
future = neural_prophet.make_future_dataframe(train_data, periods=12, n_historic_predictions=len(train_data))
neural_prophet_forecast = neural_prophet.predict(future)
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency MS corresponds to 91.011% of the data.
INFO:NP.df_utils:Major frequency MS corresponds to 91.011% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - MS
INFO:NP.df_utils:Defined frequency is equal to major frequency - MS
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for traini
ng.
INFO:NP.config:Setting normalization to global as only one dataframe provided for training.
INFO - (NP.utils.set_auto_seasonalities) - Disabling weekly seasonality. Run NeuralProphet with weekly_seasonal
ity=True to override this.
INFO:NP.utils:Disabling weekly seasonality. Run NeuralProphet with weekly_seasonality=True to override this.
INFO - (NP.utils.set_auto_seasonalities) - Disabling daily seasonality. Run NeuralProphet with daily_seasonalit
y=True to override this.
INFO:NP.utils:Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to override this.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 16
INFO:NP.config:Auto-set batch_size to 16
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 388
INFO:NP.config:Auto-set epochs to 388
```

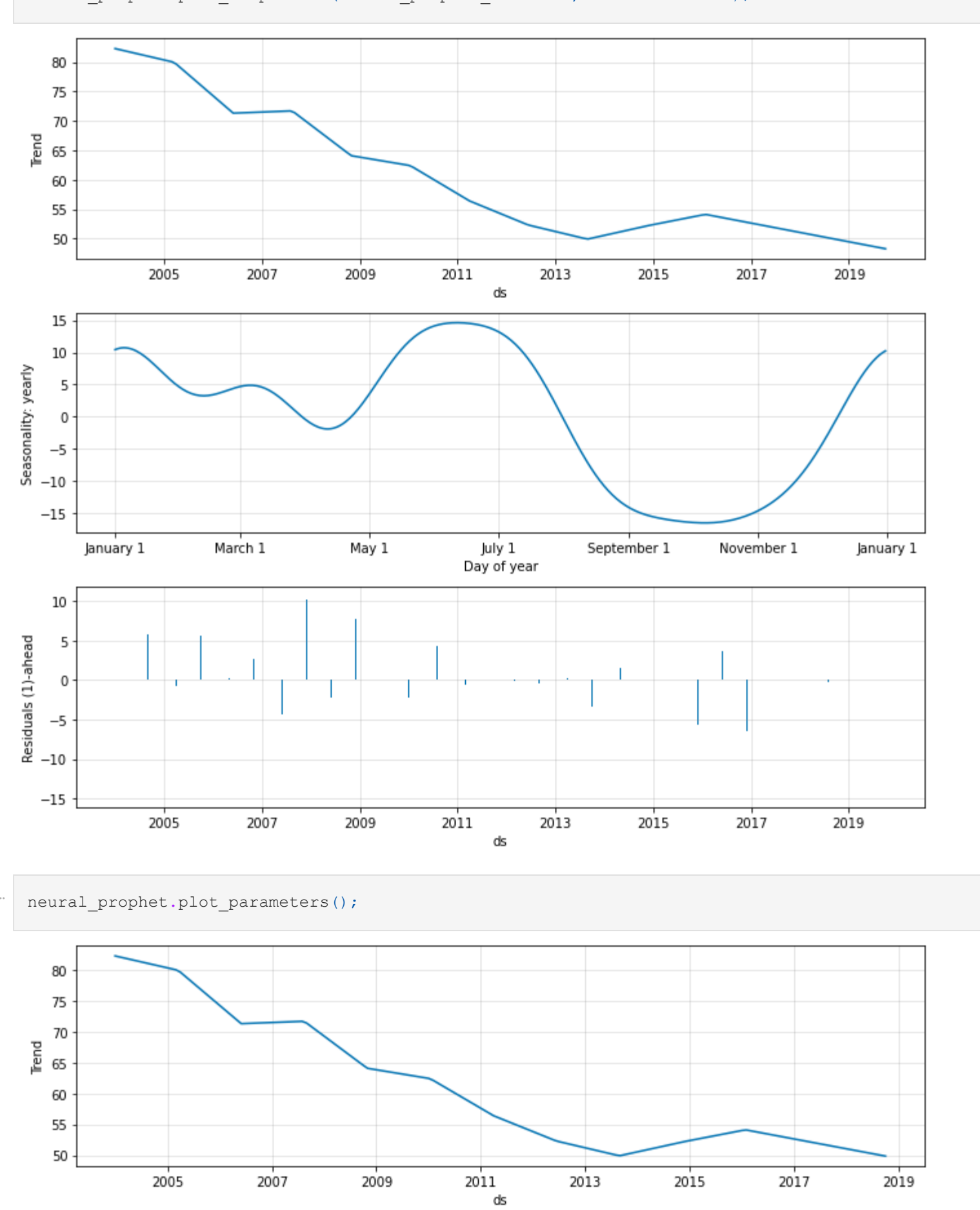
```
INFO - (NP.utils_torch.lr_range_test) - lr-range-test results: steep: 1.13E-01, min: 9.01E-01
INFO:NP.utils_torch:lr-range-test results: steep: 1.13E-01, min: 9.01E-01
```

```
INFO - (NP.utils_torch.lr_range_test) - lr-range-test results: steep: 9.39E-02, min: 1.58E+00
INFO:NP.utils_torch:lr-range-test results: steep: 9.39E-02, min: 1.58E+00
INFO - (NP.forecaster._init_train_loader) - lr-range-test selected learning rate: 1.02E-01
INFO:NP.forecaster:lr-range-test selected learning rate: 1.02E-01
Epoch[388/388]: 100%|██████████| 388/388 [00:07<00:00, 50.36it/s, SmoothL1Loss=0.00231, MAE=2.87, RMSE=3.66, Re
gLoss=0]
INFO - (NP.df_utils._infer_frequency) - Major frequency MS corresponds to 91.011% of the data.
INFO:NP.df_utils:Major frequency MS corresponds to 91.011% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - MS
INFO:NP.df_utils:Defined frequency is equal to major frequency - MS
INFO - (NP.df_utils._infer_frequency) - Major frequency MS corresponds to 91.053% of the data.
INFO:NP.df_utils:Major frequency MS corresponds to 91.053% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - MS
INFO:NP.df_utils:Defined frequency is equal to major frequency - MS
INFO - (NP.df_utils._infer_frequency) - Major frequency MS corresponds to 91.053% of the data.
INFO:NP.df_utils:Major frequency MS corresponds to 91.053% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - MS
INFO:NP.df_utils:Defined frequency is equal to major frequency - MS
```

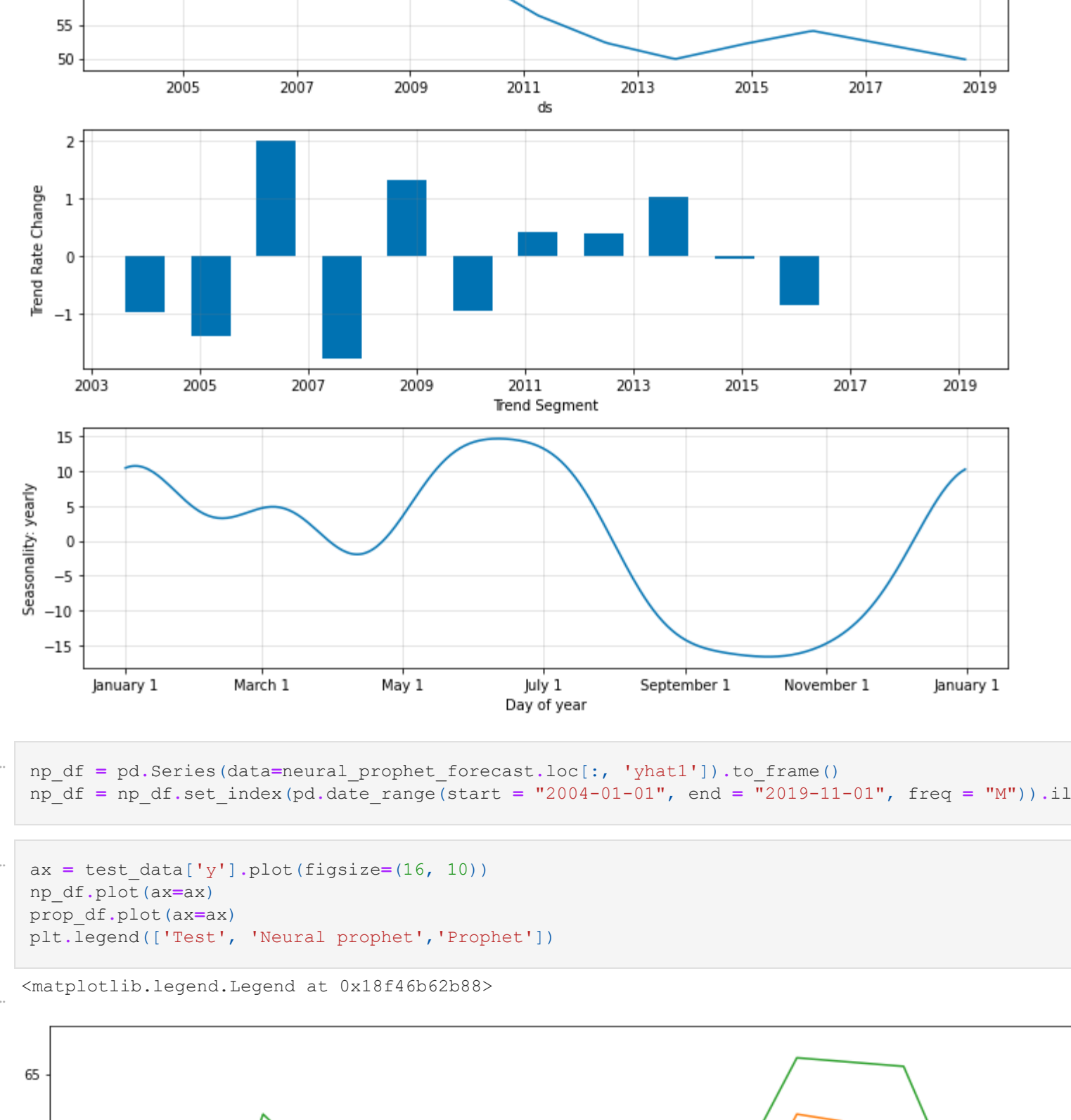
In [441]:
```python
neural_prophet.plot(neural_prophet_forecast);
```



In [442]:
```python
neural_prophet.plot_components(neural_prophet_forecast, residuals=True);
```



In [443]:
```python
neural_prophet.plot_parameters();
```



In [444]:
```python
np_df = pd.Series(data=neural_prophet_forecast.loc[:, 'yhat1']).to_frame()
np_df = np_df.set_index(pd.date_range(start = "2004-01-01", end = "2019-11-01", freq = "M")).iloc[-12:]
```

In [445]:
```python
ax = test_data['y'].plot(figsize=(16, 10))
np_df.plot(ax=ax)
prop_df.plot(ax=ax)
plt.legend(['Test', 'Neural prophet','Prophet'])
```

Out[445]:
```
<matplotlib.legend.Legend at 0x18f46b62b88>
```

In [446]:
```python
print('MSE comparison:')
print(f"Prophet:\t{mean_squared_error(test_data['y'], prop_df['yhat']):.4f}")
print(f"NeuralProphet:\t{mean_squared_error(test_data['y'], np_df['yhat1']):.4f}")
```

```
MSE comparison:
Prophet:        53.6090
NeuralProphet:  31.3549
```

So we see that Neural Prophet gives better results than Prophet.