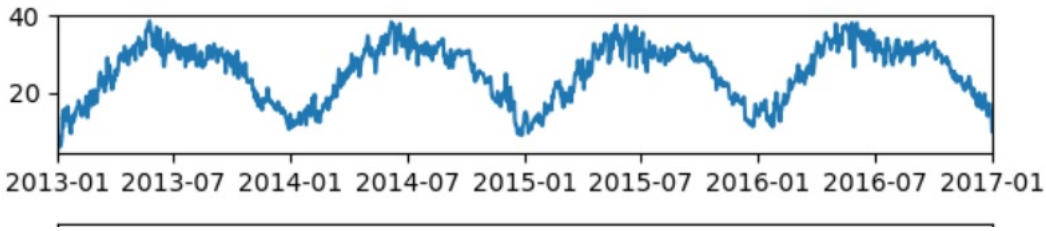
Einführung

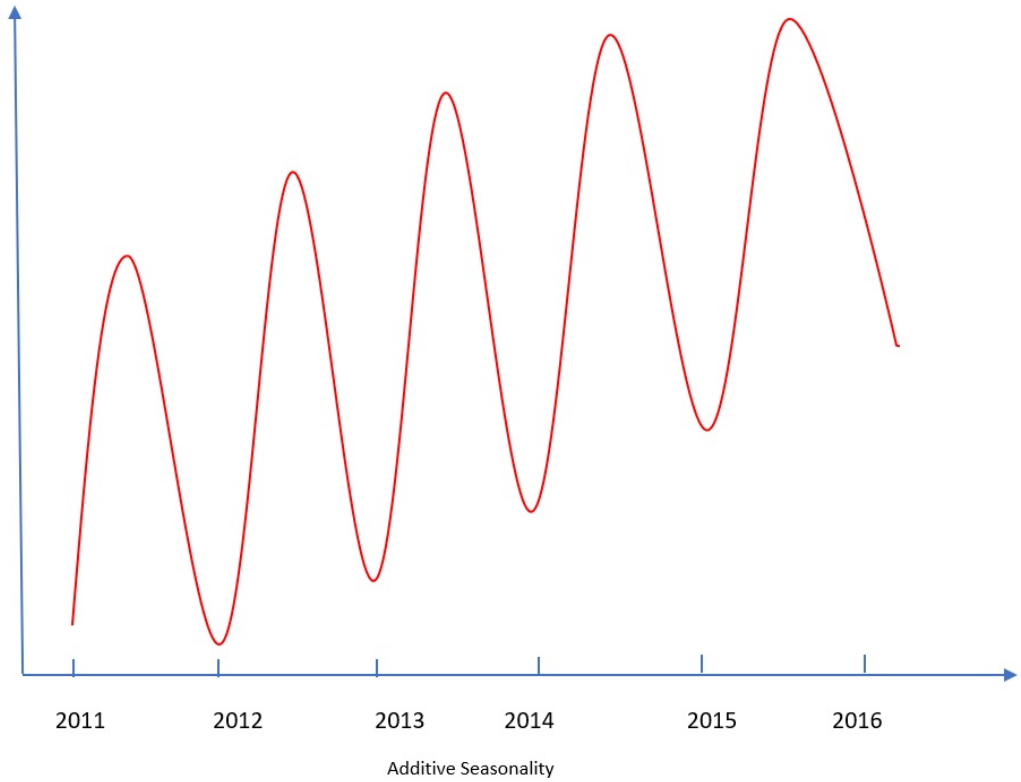
Unter Zeitreihenprognosen versteht man die Vorhersage zukünftiger Werte auf der Grundlage historischer, mit Zeitstempeln versehener Daten. Die Analyse historischer Daten kann uns helfen, zukünftige Trends zu verstehen und das Verhalten einer bestimmten Variablen in der Zukunft vorherzusagen. Im Vergleich zu Standardtechniken des maschinellen Lernens wie Random Forest und zeitverzögerten neuronalen Netzen verfügt die Zeitreihenprognosemethodik über die Fähigkeit, Muster außerhalb der Trainingsdaten zu extrapolieren und übertrifft damit die meisten Algorithmen des maschinellen Lernens, die nur Muster aus dem Trainingsdatensatz lernen können. Hier kommen Techniken zur Zeitreihenprognose ins Spiel.
Zeitreihenvorhersagen werden in vielen industriellen und wissenschaftlichen Bereichen eingesetzt, darunter Steuerungstechnik, Geschäftsplanung, Modellierung der Ausbreitung von Krankheiten, Signalverarbeitung und Wettervorhersage.

Für die Zeitreihenvorhersage werden drei Hauptbegriffe analysiert:

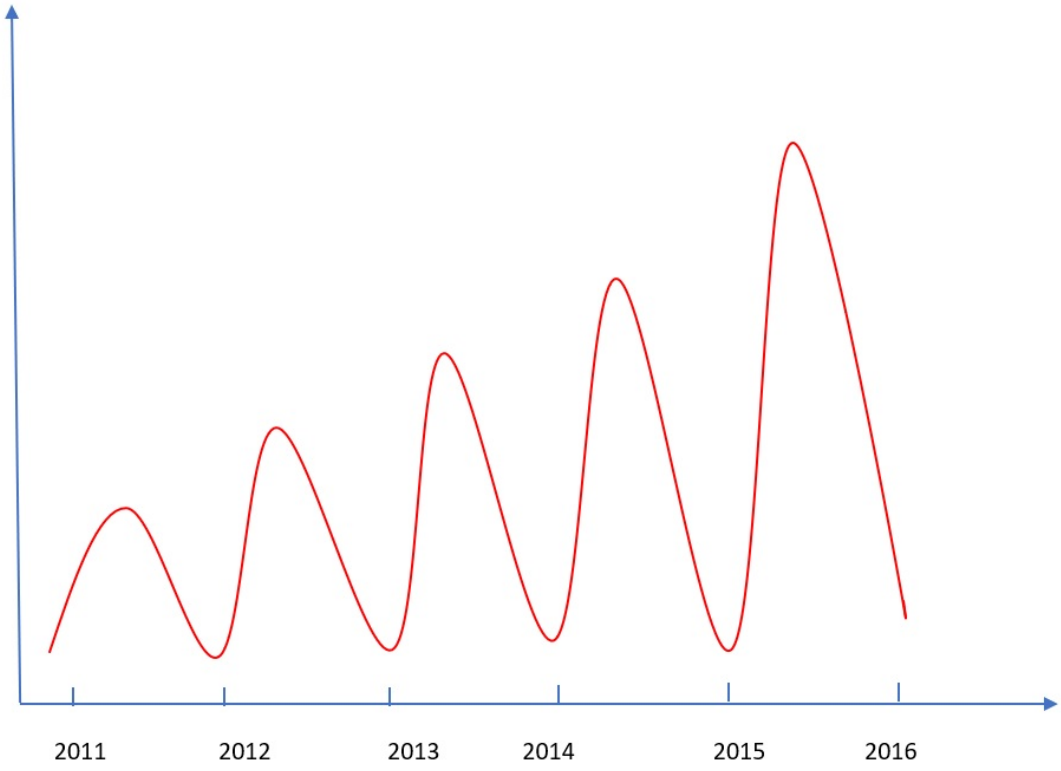
Saisonalität:
Unter Saisonalität versteht man die Wiederkehrenden Datenmuster zu bestimmten Zeitpunkten. Mit anderen Worten: Saisonalitäten sind Pattern, die sich regelmäßig in Zeitreihen über bestimmte Zeiträume wie Wochen oder Monate wiederholen. beispielsweise um Spitzenwerte in den Daten von Reiseunternehmen im November und Dezember aufgrund der Ferienzeit zu beobachten. Die Vorhersage von Zeitreihendaten erfordert die Analyse und Erfassung der Saisonalität. Die folgenden Abbildungen zeigen beispielhaft einen jährlichen Saisonzyklus:



Wir haben zwei Haupttypen der Saisonalität: die additive und die multiplikative Saisonalität. In additiven Saisonperioden zeigt der Trend Linearität, wobei die Amplitude der Saisonalität über die Zeit gleich bleibt, während wir bei multiplikativer Saisonalität Veränderungen in der Breite oder Höhe von Saisonperioden feststellen werden und der Trend über die Zeit nicht linear verläuft. Die folgenden Abbildungen zeigen den Unterschied zwischen den beiden Typen:



Additive Seasonality



Multiplicative Seasonality

Trend:
Der Trend ist ein weiterer wichtiger Faktor bei der Zeitreihenprognose. Er beschreibt die Zunahme oder Abnahme einer Variablen in einem bestimmten Zeitraum. Wenn beispielsweise der Umsatz eines Unternehmens über einen bestimmten Zeitraum steigt, weist die Umsatzvariable einen steigenden Trend auf.

Zyklische Variation:
Die Werte der Daten weisen im Zeitverlauf Schwankungen auf, die keine feste Häufigkeit haben. Diese seltenen Änderungen in den Daten werden als Zyklen bezeichnet und treten häufig aufgrund wirtschaftlicher Bedingungen auf.

Unregelmäßige Variation:
Sie stellen die Schwankungen dar, die völlig zufällig wirken oder unregelmäßig und haben kein bestimmtes Muster. Die Schwankungen sind nicht vorhersehbar, müssen jedoch bei der Erstellung von Zeitreihenmodellen berücksichtigt werden.

Long Short Term Memory (LSTM)

Die Annahme einer linearen Beziehung zwischen Eingaben und Ausgaben trifft nicht unbedingt auf jede Zeitreihe zu und daher entstand der Bedarf an Algorithmen, die jede nichtlineare Funktion für Zeitreihenprognosen annähern können, ohne vorherige Kenntnisse über die Merkmale der Daten. Long Short-Term Memory (LSTM) ist eines der beliebtesten Zeitreihen-Prognosemodelle, das seine Fähigkeit unter Beweis gestellt hat, anhand historischer Daten zukünftige Prognosen zu erstellen. Dank des Langzeitgedächtnisses ist LSTM in der Lage, noch mehr Parameter und Langzeittrends in der Zeitreihe effizient zu lernen.

In diesem Kapitel wird ausführlich auf die Architektur neuronaler LSTM-Netze eingegangen.

LSTM Übung

Wir werden die hier verfügbaren täglichen Klimazeitreihendaten verwenden: https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data.

LSTM wird auf die Spalte „meantemp" angewendet, um zukünftige Werte vorherzusagen. Werfen wir einen Blick auf den Datensatz:

In [37]:
```
import scalecast
import tensorflow
import pandas as pd
import numpy as np
import pickle
import seaborn as sns
import matplotlib.pyplot as plt
from scalecast.Forecaster import Forecaster
# If a column or index cannot be represented as an array of datetimes,
#say because of an unparsable value or a mixture of timezones,
# the column or index will be returned unaltered as an object data type.

df = pd.read_csv('DailyDelhiClimateTrain.csv',parse_dates=['date'])
# take a look at the data
df.head()
#len(df)
```

Out[37]:

	date	meantemp	humidity	wind_speed	meanpressure
0	2013-01-01	10.000000	84.500000	0.000000	1015.666667
1	2013-01-02	7.400000	92.000000	2.980000	1017.800000
2	2013-01-03	7.166667	87.000000	4.633333	1018.666667
3	2013-01-04	8.666667	71.333333	1.233333	1017.166667
4	2013-01-05	6.000000	86.833333	3.700000	1016.500000

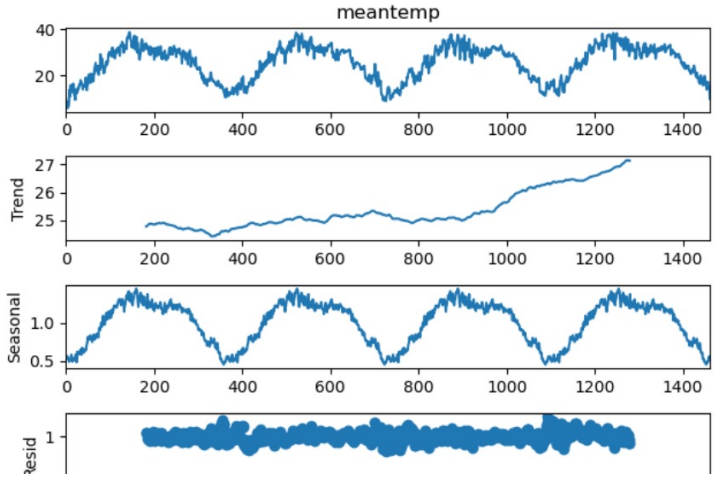
Es ist nicht immer einfach, ein leistungsstarkes LSTM-Modell zu entwerfen und zu implementieren. Aus diesem Grund stellen wir die Scalecast-Bibliothek vor, die ein Wrapper-Prognosetool für viele Zeitreihen-Prognosemodelle wie Arima und LSTM ist. Es verwendet TensorFlow LSTM und implementiert alle Datenverarbeitungsschritte, die zum Erstellen des Modells erforderlich sind, einschließlich Zerlegung der Zeitreihe in ihre Hauptbestandteile, Sacling, Unskalierung, Anpassung und Testen des Modells. Mit Scalecast Forecaster lässt sich LSTM viel einfacher anwenden und Zukunftsprognosen erstellen.

In [38]:
```
#we must first call the Forecaster object with the y and current_dates parameters as 'meantemp' and 'date' variable specified
f = Forecaster( y=df['meantemp'], current_dates=df['date'] )
f
```

Out[38]:
```
Forecaster(
    DateStartActuals=2013-01-01T00:00:00.000000000
    DateEndActuals=2017-01-01T00:00:00.000000000
    Freq=D
    N_actuals=1462
    ForecastLength=0
    Xvars=[]
    TestLength=0
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=None
    CurrentEstimator=mlr
    GridsFile=Grids
)
```

Bevor wir ein LSTM-Modell erstellen, sollten wir die Hauptkomponenten der Zeitreihe wie Saisonalität und Trend überprüfen. Diese Komponenten helfen uns, ein Prognosemodell effizienter zu erstellen und können andere Merkmale wie Stationarität in Zeitreihen zeigen. Stationarität bedeutet, dass sich die Eigenschaften von Zeitreihen unabhängig von dem Zeitpunkt ändern, zu dem die Zeitreihe beobachtet wird, und dass eine stationäre Zeitreihe daher keine signifikante Saisonalität oder einen signifikanten Trend aufweist, da diese Komponenten die Zeitreihe zu bestimmten Zeitpunkten beeinflussen Zeit. Daher dient die Untersuchung des saisonalen Zyklus und der Trends dazu, Einblicke in die Stationarität von Zeitreihen zu gewinnen. Um die Zeitreihe in ihre Hauptkomponenten zu zerlegen, verwenden wir seasonal_decompose Funktion von Statsmodels Bibliotik wie im Folgenden:

In [17]:
```
# Let's further decompose the series into its trend, seasonal, and residual parts:
from matplotlib import pyplot
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df['meantemp'],  model='multiplicative', period=365)# requency of the observations is 1
result.plot()
pyplot.show()
#The figure obviously indicates yearly seasonality and increasing trend over time.
```



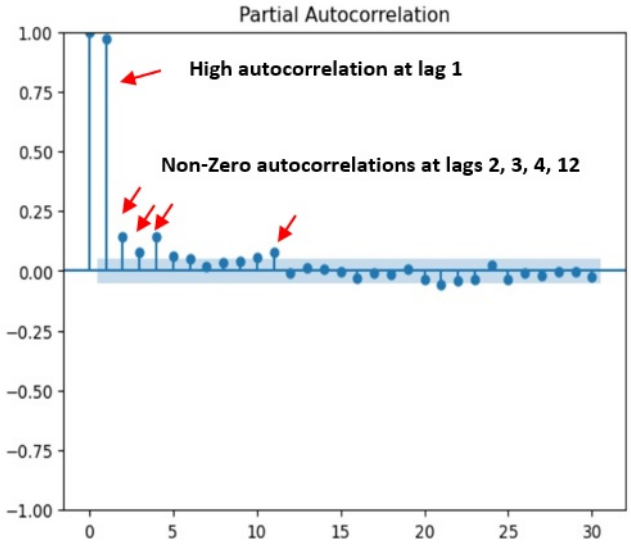
Die vorherige Abbildung zeigt offensichtlich eine erhebliche jährliche Saisonalität und einen zunehmenden Trend im Zeitverlauf.
Lassen Sie uns nun die Autokorrelation der Durchschnittstemperatur mit ihren vergangenen Werten überprüfen. Zu diesem Zweck verwenden wir das Partial Auto Correlation Function (PACF)-Diagramm, das misst, wie stark die y-Variable (meantemp) statistisch mit früheren Werten von sich selbst korreliert. Wir prüfen die Autokorrelationen auf bis zu 30 Verzögerungen.

Der blaue Bereich des PACF-Diagramms zeigt den Signifikanzschwellenwert. Das bedeutet, dass Verzögerungen, die sich in diesem Bereich be-

finden, statistisch gesehen nahe Null liegen und daher keine signifikante Autokorrelation zwischen Datenpunkten besteht. PACF stellt intuitiv Korrelationen von 1 bei Verzögerung 0 dar, da dies die Korrelation der Zeitreihe mit sich selbst darstellt.

In [39]:
```
f.plot_pacf(lags=30)
plt.show()
```

```
C:\Users\la2022\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348: FutureWarning: The default method 'yw' can pr
oduce 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(
```



Testen wir die Stationarität der Reihe mit dem Augmented Dickey-Fuller (ADF)-Test. null Die Hypothese dieses Tests ist, dass die Zeitreihe nicht_stationär ist (es gibt eine Einheitswurzel). Wenn der resultierende p_value über einer kritischen Größe liegt (Standard ist 0,05), können wir nicht ausschließen, dass es eine Einheitswurzel gibt und die Reihe daher instationär ist, siehe hier .
Der resultierende p_Wert zeigt an, dass unsere Zeitsequenz nicht stationär ist.

In [8]:
```
# let's test the series' stationarity.
# If bool (full_res = False), returns whether the test suggests stationarity.
# null Hypothesis of Augmented Dickey-Fuller (ADF) test: time series is non_stationary ( there is a unit root,)
# If the pvalue is above a critical size (Default is 0.05), then we cannot reject that there is a unit root.
stat = f.adf_test(full_res=True)
print(stat)

# p_value is 0.28 > 0.0.5, then we cannot reject null hypothesis and thus time series is non stationary
```

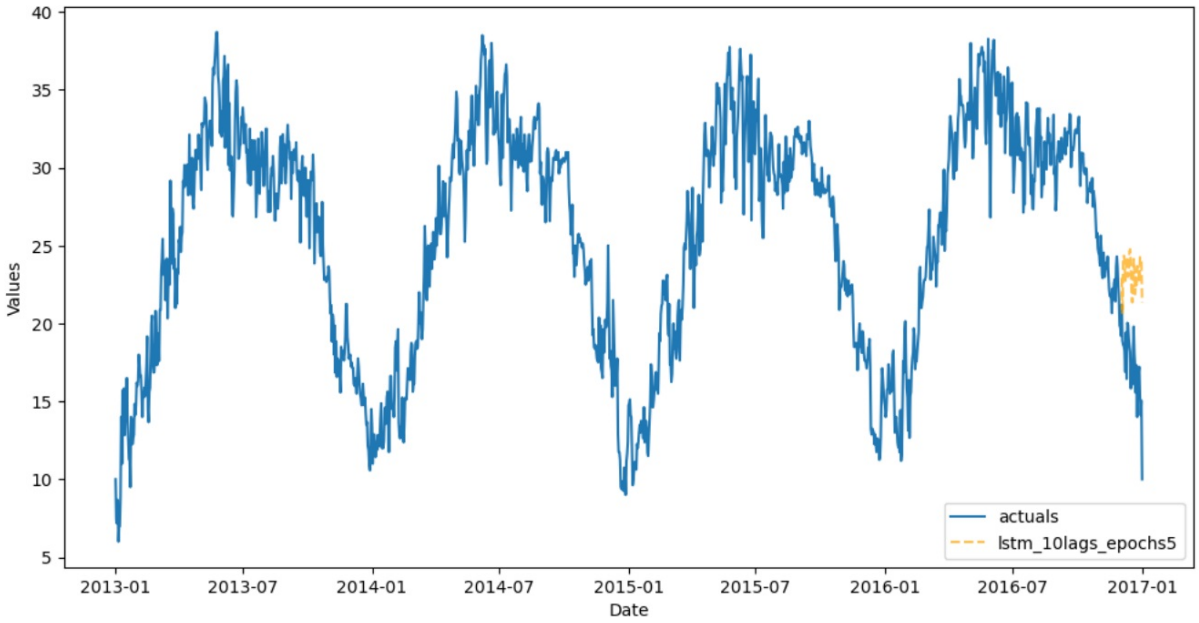
```
(-2.021069055920671, 0.2774121372301611, 10, 1451, {'1%': -3.4348647527922824, '5%': -2.863533960720434, '10%': -2.567831568508
802}, 5423.895746470953)
```

Wir werden unser LSTM mit Scalecast Forecater erstellen. Standardmäßig,
Dieses Modell wird mit einer einzelnen Eingabeebene von 8 Einheiten, Adam-Optimierer, Tanh-Aktivierung, einer Lernrate von 0,001 und ohne Dropout ausgeführt. Wir müssen jedoch einige Parameter festlegen, z. B. Testsatz, Validierungsmetrik, Anzahl der Epochen und Anzahl der Verzögerungen, die für die Vorhersage verwendet werden. für den ersten Lauf werden wir verwenden10 Verzögerungen und 5 Epochen. Schauen wir uns die resultierenden Diagramme an:

In [67]:
```
# Now, to call an LSTM forecast. By default,
# this model will be run with a single input layer of 8 units, Adam optimizer, tanh activation,
#  a learning rate of 0.001, and no dropout.

# generate future dates: The number of dates you generate in this step will determine how long all models will be forecast out.
f.set_validation_metric('mape')
f.set_test_length(30)       #   30 observations to test the results
f.generate_future_dates(30) #  30 future points to forecast
f.set_estimator('lstm')     #  LSTM neural network
f.manual_forecast(call_me='lstm_10lags_epochs5',lags=10, epochs=5)
f.plot_test_set(ci=True)
```

```
Epoch 1/5
44/44 [==============================] - 1s 3ms/step - loss: 0.5967
Epoch 2/5
44/44 [==============================] - 0s 3ms/step - loss: 0.4379
Epoch 3/5
44/44 [==============================] - 0s 3ms/step - loss: 0.2333
Epoch 4/5
44/44 [==============================] - 0s 3ms/step - loss: 0.1348
Epoch 5/5
44/44 [==============================] - 0s 3ms/step - loss: 0.0981
1/1 [==============================] - 0s 231ms/step
Epoch 1/5
45/45 [==============================] - 1s 3ms/step - loss: 0.5467
Epoch 2/5
45/45 [==============================] - 0s 4ms/step - loss: 0.3424
Epoch 3/5
```



Die vorherige Darstellung zeigt Prognosen, die relativ weit von den tatsächlichen Daten entfernt liegen. Daher ist es notwendig, unsere Parameter abzustimmen und dann das Modell zu testen.
Jetzt ändern wir die Anzahl der Verzögerungen auf 5 und die Anzahl der Epochen auf 20. Außerdem werden wir die Anzahl der Ebenen erhöhen und weitere Parameter optimieren. Lassen Sie uns die Leistung und Plots überprüfen:

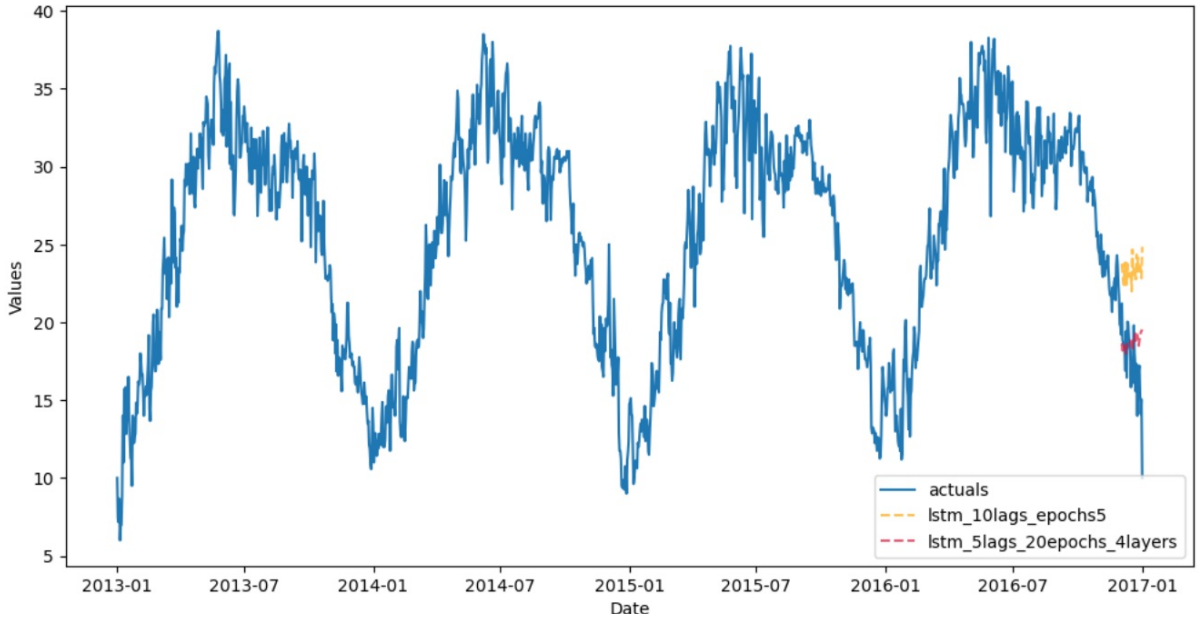
In [*]:

```
# All data is scaled going into the model with a min-max scaler and un-scaled coming out.
#Anything you can pass to the fit() method in TensorFlow,
# you can also pass to the scalecast manual_forecast() method.
#Plots all test-set predictions with the actuals.
#ci (bool) - Default False. Whether to display the confidence intervals.
# 5 lags, since we noticed 5 days autocorrelation

# let's try increasing the number of layers in the network to 4,
#increasing epochs to 10, but monitoring the validation loss value and telling the model to quit after more
#than 5 iterations in which that doesn't improve. This is known as early stopping.
from tensorflow.keras.callbacks import EarlyStopping
f.manual_forecast(
    call_me='lstm_5lags_20epochs_4layers',
    lags=5,
    epochs=20,
    batch_size=16,
    activation='tanh',
    optimizer='Adam',
    shuffle=True,
    learning_rate=0.01,

    lstm_layer_sizes=(72,)*4, # 4 layers, each 72 units (size)
     dropout=(0,)*4, # dropout rate for each layer
     plot_loss=True
)

f.plot_test_set(ci=True)
```



Das zweite Modell zeigt offensichtlich eine bessere Leistung. Die MAPE-Ergebnisse für den Testsatz zeigen, dass das letztere Modell das erste übertrifft:

In [83]:
```
# lets have a look on the statistics of our models
res = f.export(dfs=['model_summaries'])
models =res['ModelNickname']
for m in models:
    print(m,  res.loc[res['ModelNickname'] == m, 'LevelTestSetMAPE'])
```

```
lstm_10lags_epochs5 0    0.37005
Name: LevelTestSetMAPE, dtype: float64
lstm_5lags_earlystop_4layers 1    0.10882
Name: LevelTestSetMAPE, dtype: float64
```


```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "1a69b9d5",
   "metadata": {},
   "outputs": [],
   "source": [
    "# we will use climate time series data (daily):https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-da
    "# install scalecast library\n",
    "!pip install scalecast --upgrade"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "5fb69a3b",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
       "<style scoped>\n",
       "    .dataframe tbody tr th:only-of-type {\n",
       "        vertical-align: middle;\n",
       "    }\n",
       "\n",
       "    .dataframe tbody tr th {\n",
       "        vertical-align: top;\n",
       "    }\n",
       "\n",
       "    .dataframe thead th {\n",
       "        text-align: right;\n",
       "    }\n",
       "</style>\n",
       "<table border=\"1\" class=\"dataframe\">\n",
       "  <thead>\n",
       "    <tr style=\"text-align: right;\">\n",
       "      <th></th>\n",
       "      <th>date</th>\n",
       "      <th>meantemp</th>\n",
       "      <th>humidity</th>\n",
       "      <th>wind_speed</th>\n",
       "      <th>meanpressure</th>\n",
       "    </tr>\n",
       "  </thead>\n",
       "  <tbody>\n",
       "    <tr>\n",
       "      <th>0</th>\n",
       "      <td>2013-01-01</td>\n",
       "      <td>10.000000</td>\n",
       "      <td>84.500000</td>\n",
       "      <td>0.000000</td>\n",
       "      <td>1015.666667</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>1</th>\n",
       "      <td>2013-01-02</td>\n",
       "      <td>7.400000</td>\n",
       "      <td>92.000000</td>\n",
       "      <td>2.980000</td>\n",
       "      <td>1017.800000</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>2</th>\n",
       "      <td>2013-01-03</td>\n",
       "      <td>7.166667</td>\n",
       "      <td>87.000000</td>\n",
       "      <td>4.633333</td>\n",
       "      <td>1018.666667</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>3</th>\n",
       "      <td>2013-01-04</td>\n",
       "      <td>8.666667</td>\n",
       "      <td>71.333333</td>\n",
       "      <td>1.233333</td>\n",
       "      <td>1017.166667</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>4</th>\n",
       "      <td>2013-01-05</td>\n",
       "      <td>6.000000</td>\n",
       "      <td>86.833333</td>\n",
       "      <td>3.700000</td>\n",
       "      <td>1016.500000</td>\n",
       "    </tr>\n",
```

```
        "  </tbody>\n",
        "</table>\n",
        "</div>"
       ],
       "text/plain": [
        "        date   meantemp   humidity  wind_speed  meanpressure\n",
        "0 2013-01-01  10.000000  84.500000    0.000000   1015.666667\n",
        "1 2013-01-02   7.400000  92.000000    2.980000   1017.800000\n",
        "2 2013-01-03   7.166667  87.000000    4.633333   1018.666667\n",
        "3 2013-01-04   8.666667  71.333333    1.233333   1017.166667\n",
        "4 2013-01-05   6.000000  86.833333    3.700000   1016.500000"
       ]
      },
      "execution_count": 1,
      "metadata": {},
      "output_type": "execute_result"
     }
    ],
    "source": [
     "import scalecast\n",
     "import tensorflow\n",
     "import pandas as pd\n",
     "import numpy as np\n",
     "import pickle\n",
     "import seaborn as sns\n",
     "import matplotlib.pyplot as plt\n",
     "from scalecast.Forecaster import Forecaster\n",
     "# If a column or index cannot be represented as an array of datetimes,\n",
     "# say because of an unparsable value or a mixture of timezones,\n",
     "# the column or index will be returned unaltered as an object data type.\n",
     "\n",
     "df = pd.read_csv('DailyDelhiClimateTrain.csv',parse_dates=['date'])\n",
     "# take a look at the data\n",
     "df.head()\n",
     "#len(df)"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 2,
    "id": "7c9592f0",
    "metadata": {},
    "outputs": [
     {
      "data": {
       "text/plain": [
        "Forecaster(\n",
        "    DateStartActuals=2013-01-01T00:00:00.000000000\n",
        "    DateEndActuals=2017-01-01T00:00:00.000000000\n",
        "    Freq=D\n",
        "    N_actuals=1462\n",
        "    ForecastLength=0\n",
        "    Xvars=[]\n",
        "    TestLength=0\n",
        "    ValidationLength=1\n",
        "    ValidationMetric=rmse\n",
        "    ForecastsEvaluated=[]\n",
        "    CILevel=None\n",
        "    CurrentEstimator=mlr\n",
        "    GridsFile=Grids\n",
        ")"
       ]
      },
      "execution_count": 2,
      "metadata": {},
      "output_type": "execute_result"
     }
    ],
    "source": [
     "#we must first call the Forecaster object with the y and current_dates parameters as 'meantemp' and 'date' variable spe
     "f = Forecaster( y=df['meantemp'], current_dates=df['date'] )\n",
     "f"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 3,
    "id": "8f8c814e",
    "metadata": {},
    "outputs": [
     {
      "name": "stderr",
      "output_type": "stream",
      "text": [
       "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\statsmodels\\graphics\\tsaplots.py:348: FutureWarning: The default 
       "  warnings.warn(\n"
      ]
     },
     {
      "data": {
       "image/png": "iVBORw0KGgoAAAANSUhEUgAAAjgAAAGxCAYAAABvIsx7bAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
       "text/plain": [
        "<Figure size 640x480 with 1 Axes>"
       ]
      },
      "metadata": {},
      "output_type": "display_data"
     }
    ],
    "source": [
     "#Let?s decompose this time series by viewing the PACF (Partial Auto Correlation Function) plot,\n",
     "# which measures how much the y variable(meantemp) is correlated to past values of itself.\n",
     "# blue area PACF plots depicts the significance threshold. \n",
     "# That means, lags that located within this area is statistically close to zero and thus insignificant autocorrelation\
```

```
    "# between data points. \n",
    "f.plot_pacf(lags=30)# up to 30 lags\n",
    "plt.show()\n",
    "# PACF will depicts intuitively correlations of 1  at  lag  0,\n",
    "# since this represents the correlation of the time series with itself.\n",
    "#this plot indicate significant autocorrelation at lag 1 which means that adjacent points (have lag of 1) are highly co
    "# there are non zero autocorrelation at different lags as well"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "5c1c43fc",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "image/png": "iVBORw0KGgoAAAANSUhEUgAAAnYAAAHWCAYAAAD6oMSKAAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
      "text/plain": [
       "<Figure size 640x480 with 4 Axes>"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "# Let?s further decompose the series into its trend, seasonal, and residual parts:\n",
    "from matplotlib import pyplot\n",
    "from statsmodels.tsa.seasonal import seasonal_decompose\n",
    "result = seasonal_decompose(df['meantemp'],  model='multiplicative', period=365)# requency of the observations is 1\n",
    "result.plot()\n",
    "pyplot.show()\n",
    "#The figure obviously indicates yearly seasonality and increasing trend over time."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "96cbfb55",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "(-2.021069055920671, 0.2774121372301611, 10, 1451, {'1%': -3.4348647527922824, '5%': -2.863533960720434, '10%': -2.56
     ]
    }
   ],
   "source": [
    "# let?s test the series? stationarity.\n",
    "# If bool (full_res = False), returns whether the test suggests stationarity.\n",
    "# null Hypothesis of Augmented Dickey-Fuller (ADF) test: time series is non_stationary ( there is a unit root,)\n",
    "# If the pvalue is above a critical size (Default is 0.05), then we cannot reject that there is a unit root.\n",
    "stat = f.adf_test(full_res=True)\n",
    "print(stat)\n",
    "\n",
    "# p_value is 0.28 > 0.0.5, then we cannot reject null hypothesis and thus time series is non stationary"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "id": "479c8afe",
   "metadata": {},
   "outputs": [],
   "source": [
    "# Now, to call an LSTM forecast. By default, \n",
    "# this model will be run with a single input layer of 8 units, Adam optimizer, tanh activation,\n",
    "#  a learning rate of 0.001, and no dropout.\n",
    "\n",
    "# generate future dates: The number of dates you generate in this step will determine how long all models will be foreca
    "f.set_validation_metric('mape')\n",
    "f.set_test_length(30)       #   30 observations to test the results\n",
    "f.generate_future_dates(30) #  30 future points to forecast\n",
    "\n",
    "#  LSTM neural network\n",
    "f.set_estimator('lstm')  "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "c430afc2",
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "id": "ed39a4f1",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Epoch 1/5\n",
      "44/44 [==============================] - 2s 5ms/step - loss: 0.5603\n",
      "Epoch 2/5\n",
      "44/44 [==============================] - 0s 5ms/step - loss: 0.3823\n",
      "Epoch 3/5\n",
```

```
      "44/44 [==============================] - 0s 5ms/step - loss: 0.2042\n",
      "Epoch 4/5\n",
      "44/44 [==============================] - 0s 5ms/step - loss: 0.1186\n",
      "Epoch 5/5\n",
      "44/44 [==============================] - 0s 5ms/step - loss: 0.0934\n",
      "1/1 [==============================] - 0s 444ms/step\n",
      "Epoch 1/5\n",
      "45/45 [==============================] - 2s 5ms/step - loss: 0.5340\n",
      "Epoch 2/5\n",
      "45/45 [==============================] - 0s 5ms/step - loss: 0.3052\n",
      "Epoch 3/5\n",
      "45/45 [==============================] - 0s 4ms/step - loss: 0.1584\n",
      "Epoch 4/5\n",
      "45/45 [==============================] - 0s 5ms/step - loss: 0.1089\n",
      "Epoch 5/5\n",
      "45/45 [==============================] - 0s 5ms/step - loss: 0.0955\n",
      "1/1 [==============================] - 0s 426ms/step\n",
      "45/45 [==============================] - 0s 2ms/step\n"
     ]
    },
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\scalecast\\_utils.py:55: Warning: Confidence intervals not found fo
      "  warnings.warn(\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "<Axes: xlabel='Date', ylabel='Values'>"
      ]
     },
     "execution_count": 8,
     "metadata": {},
     "output_type": "execute_result"
    },
    {
     "data": {
      "image/png": "iVBORw0KGgoAAAANSUhEUgAA+UAAAIOCAYAAADeN6x/AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
      "text/plain": [
       "<Figure size 1200x600 with 1 Axes>"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "f.manual_forecast(call_me='lstm_10lags_epochs5',lags=10, epochs=5)\n",
    "#f.tf_model.save('lstm_10lags_epochs5_model.h5')\n",
    "f.plot_test_set(ci=True)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 9,
   "id": "e4fca805",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "<bound method Model.summary of <keras.engine.sequential.Sequential object at 0x000002205F08F040>>"
      ]
     },
     "execution_count": 9,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "#del f.tf_model\n",
    "from keras.models import load_model\n",
    "    \n",
    "model=load_model('lstm_10lags_epochs5_model.h5')\n",
    "model.summary\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 10,
   "id": "ffa3a48a",
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\scalecast\\_utils.py:55: Warning: Confidence intervals not found fo
      "  warnings.warn(\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "<Axes: xlabel='Date', ylabel='Values'>"
      ]
     },
     "execution_count": 10,
     "metadata": {},
     "output_type": "execute_result"
    },
    {
```

```
    {
     "data": {
      "image/png": "iVBORw0KGgoAAAANSUhEUgAAA+UAAAIOCAYAAADeN6x/AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
      "text/plain": [
       "<Figure size 1200x600 with 1 Axes>"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "f.plot_test_set(ci=True)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 18,
   "id": "045bc465",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Epoch 1/20\n",
      "88/88 [==============================] - 8s 11ms/step - loss: 0.1365\n",
      "Epoch 2/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0938\n",
      "Epoch 3/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0934\n",
      "Epoch 4/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0937\n",
      "Epoch 5/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0886\n",
      "Epoch 6/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0886\n",
      "Epoch 7/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0884\n",
      "Epoch 8/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0881\n",
      "Epoch 9/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0865\n",
      "Epoch 10/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0857\n",
      "Epoch 11/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0892\n",
      "Epoch 12/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0854\n",
      "Epoch 13/20\n",
      "88/88 [==============================] - 1s 12ms/step - loss: 0.0846\n",
      "Epoch 14/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0841\n",
      "Epoch 15/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0866\n",
      "Epoch 16/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0867\n",
      "Epoch 17/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0843\n",
      "Epoch 18/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0831\n",
      "Epoch 19/20\n",
      "88/88 [==============================] - 1s 12ms/step - loss: 0.0833\n",
      "Epoch 20/20\n",
      "88/88 [==============================] - 1s 11ms/step - loss: 0.0836\n",
      "1/1 [==============================] - 1s 1s/step\n",
      "Epoch 1/20\n",
      "90/90 [==============================] - 8s 11ms/step - loss: 0.1291\n",
      "Epoch 2/20\n",
      "90/90 [==============================] - 1s 12ms/step - loss: 0.0950\n",
      "Epoch 3/20\n",
      "90/90 [==============================] - 1s 11ms/step - loss: 0.0947\n",
      "Epoch 4/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0958\n",
      "Epoch 5/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0913\n",
      "Epoch 6/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0903\n",
      "Epoch 7/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0892\n",
      "Epoch 8/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0881\n",
      "Epoch 9/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0855\n",
      "Epoch 10/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0850\n",
      "Epoch 11/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0849\n",
      "Epoch 12/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0866\n",
      "Epoch 13/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0854\n",
      "Epoch 14/20\n",
      "90/90 [==============================] - 1s 11ms/step - loss: 0.0866\n",
      "Epoch 15/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0870\n",
      "Epoch 16/20\n",
      "90/90 [==============================] - 1s 10ms/step - loss: 0.0867\n"
     ]
    },
    {
     "data": {
      "image/png": "iVBORw0KGgoAAAANSUhEUgAAA+UAAAIwAAAHFCAYAAADR1KI+AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
      "text/plain": [
```

```
        "<Figure size 640x480 with 1 Axes>"
       ]
      },
      "metadata": {},
      "output_type": "display_data"
     },
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "1/1 [==============================] - 1s 1s/step\n",
       "45/45 [==============================] - 0s 5ms/step\n"
      ]
     },
     {
      "name": "stderr",
      "output_type": "stream",
      "text": [
       "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\scalecast\\_utils.py:55: Warning: Confidence intervals not found fo
       "  warnings.warn(\n",
       "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\scalecast\\_utils.py:55: Warning: Confidence intervals not found fo
       "  warnings.warn(\n"
      ]
     },
     {
      "data": {
       "text/plain": [
        "<Axes: xlabel='Date', ylabel='Values'>"
       ]
      },
      "execution_count": 18,
      "metadata": {},
      "output_type": "execute_result"
     },
     {
      "data": {
       "image/png": "iVBORw0KGgoAAAANSUhEUgAAA+UAAAOECAYAAADeN6x/AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
       "text/plain": [
        "<Figure size 1200x600 with 1 Axes>"
       ]
      },
      "metadata": {},
      "output_type": "display_data"
     }
    ],
    "source": [
     "# All data is scaled going into the model with a min-max scaler and un-scaled coming out.\n",
     "#Anything you can pass to the fit() method in TensorFlow,\n",
     "# you can also pass to the scalecast manual_forecast() method.\n",
     "#Plots all test-set predictions with the actuals.\n",
     "#ci (bool) ? Default False. Whether to display the confidence intervals.\n",
     "# 5 lags, since we noticed 5 days autocorrelation \n",
     "\n",
     "# let?s try increasing the number of layers in the network to 4,\n",
     "#increasing epochs to 10, but monitoring the validation loss value and telling the model to quit after more\n",
     "#than 5 iterations in which that doesn?t improve. This is known as early stopping.\n",
     "from tensorflow.keras.callbacks import EarlyStopping\n",
     "callback = EarlyStopping(monitor='loss', patience=5)\n",
     "f.manual_forecast(\n",
     "    call_me='lstm_5lags_20epochs_4layers',\n",
     "    lags=5,\n",
     "    epochs=20,\n",
     "    batch_size=16,\n",
     "    activation='tanh',\n",
     "    optimizer='Adam',\n",
     "    shuffle=True,\n",
     "    learning_rate=0.01,\n",
     "    callbacks=[callback],\n",
     "    lstm_layer_sizes=(72,)*4, # 4 layers, each 72 units (size)\n",
     "    dropout=(0,)*4, # dropout rate for each layer\n",
     "    plot_loss=True\n",
     ")\n",
     "\n",
     "f.plot_test_set(ci=True)"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 12,
    "id": "594568f8",
    "metadata": {},
    "outputs": [
     {
      "name": "stderr",
      "output_type": "stream",
      "text": [
       "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\scalecast\\_utils.py:55: Warning: Confidence intervals not found fo
       "  warnings.warn(\n",
       "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\scalecast\\_utils.py:55: Warning: Confidence intervals not found fo
       "  warnings.warn(\n"
      ]
     },
     {
      "data": {
       "text/plain": [
        "<Axes: xlabel='Date', ylabel='Values'>"
       ]
      },
      "execution_count": 12,
      "metadata": {},
      "output_type": "execute_result"
     },
     {
      "data": {
```

```
      "image/png": "iVBORw0KGgoAAAANSUhEUgAAA+UAAAIOCAYAAADeN6x/AAAAOXRFWHRTb2Z0d2FyZQBNYXRwbG90bGliIHZlcnNpb24zLjcuMSwgaHR0
      "text/plain": [
       "<Figure size 1200x600 with 1 Axes>"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "f.plot_test_set(ci=True)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "060e5e65",
   "metadata": {},
   "outputs": [],
   "source": [
    "# plot the best 2 models based on MAPE metric\n",
    "#f.plot_test_set(order_by='LevelTestSetMAPE',models='top_2',ci=True)## MAPE metric is used "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 13,
   "id": "27ec740f",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "lstm_10lags_epochs5 0    0.353709\n",
      "Name: LevelTestSetMAPE, dtype: float64\n",
      "lstm_5lags_20epochs_4layers 1    0.126901\n",
      "Name: LevelTestSetMAPE, dtype: float64\n"
     ]
    }
   ],
   "source": [
    "# lets have a look on the statistics of our models\n",
    "res = f.export(dfs=['model_summaries'])\n",
    "models =res['ModelNickname']\n",
    "for m in models:\n",
    "    print(m,  res.loc[res['ModelNickname'] == m, 'LevelTestSetMAPE'])\n",
    " "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "ba1ef624",
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "bb0a4723",
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.13"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```


Exponentielle Glättung

Exponentielle Glättung ist eine Prognosemethode für univariate Zeitreihendaten. Im Gegensatz zur Methode des gleitenden Durchschnitts, bei der alle vergangenen Beobachtungen gleich gewichtet werden, wenn sie in das Fenster des gleitenden Durchschnitts fallen, verwendet die Methode der exponentiellen Glättung Gewichtungen, die mit zunehmendem Alter der Beobachtungen exponentiell abnehmen, wodurch Ausreißer oder Rauschen aus den Daten entfernt werden. Dies hilft dem Modell, deutliche und sich wiederholende Muster besser zu erkennen, die sonst im Rauschen verborgen wären, und kann erheblich zur Prognoseleistung beitragen, indem die Genauigkeit der Vorhersagen verbessert wird.

Zu den verschiedenen Arten der exponentiellen Glättung gehören die einfache exponentielle Glättung und die dreifache exponentielle Glättung (auch als Holt-Winters-Methode bekannt). Die einfache exponentielle Glättung befasst sich mit Datenreihen, die weder einen Trend noch eine saisonale Komponente aufweisen. Diese Methode verwendet nur einen Parameter namens Alpha (α) und aktualisiert die Pegelkomponente für jede Beobachtung. Da eine Komponente modelliert wird, verwendet es nur einen Gewichtungsparameter, Alpha (α), der den Grad der Glättung bestimmt und Werte zwischen 0 und 1 annimmt. Niedrigere Werte führen zu glatteren angepassten Modellen, da den vergangenen Daten mehr Gewichtung gegeben wird und so die aktuellen Daten effizient mit den alten gemittelt werden. Umgekehrt können hohe Werte ein angepasstes Modell erzeugen, das schnell auf das zufällige Rauschen reagiert. Die einfache exponentielle Glättung wird in der folgenden mathematischen Formel dargestellt:

$$f_t = \alpha x_{t-1} + (1- \alpha) f_{t-1}$$

Dabei ist xt-1 die Beobachtung im vorherigen Zeitschritt. Alpha ist der Prozentsatz, wie wichtig die jüngste Beobachtung im Vergleich zu historischen Daten im Modell ist. ft-1 die Vorhersage des Modells im vorherigen Zeitschritt. Die obige Gleichung kann in die folgende Formel übersetzt werden:

$$\begin{aligned} f_t &= \alpha x_{t-1} + (1- \alpha) (\alpha x_{t-2} + (1 - \alpha) f_{t-2}) \\ &= \alpha x_{t-1} + \alpha (1- \alpha) x_{t-2} + (1- \alpha)^2 f_{t-2} \\ &= \alpha x_{t-1} + \alpha (1- \alpha) x_{t-2} + \alpha (1- \alpha)^2 x_{t-3} + (1- \alpha)^3 f_{t-3} \end{aligned}$$

us dieser Gleichung können wir das Hauptkonzept der exponentiellen Glättung verstehen, bei der das Modell weiteren Beobachtungen exponentiell niedrigere Gewichte zuweist als den jüngsten.
Daher sollte der Alpha-Wert feinabgestimmt werden, um ein gut passendes Prognosemodell zu erhalten. Einfache exponentielle Glättung wird beispielsweise verwendet, um die Nachfrage nach einem Produkt zu modellieren.

Die dreifache exponentielle Glättung (Holt-Winters) wird für Zeitreihen verwendet, die Trend- und Saisonschwankungen aufweisen. Dies ist eine Erweiterung der einfachen exponentiellen Glättung, die die Einbeziehung von Saisonalitäts- und Trendkomponenten ermöglicht, um viele Werte über historische Daten zu berechnen und genauere aktuelle und zukünftige Prognosen zu erstellen.

Exponentielle Glättung Übung

Wir werden Covid-19 tägliche Zeitreihe verwenden. Diese Zeitreihe besteht aus den beiden Spalten „Date" und „Confirmed". Importieren Sie die Daten und werfen Sie einen Blick darauf:

In [6]:
```
# downloading dataset
import pandas as pd
df = pd.read_csv('Covid19_ts.csv',parse_dates=['Date'])
df['Date'] = pd.to_datetime(df['Date'])
df = pd.DataFrame(df, columns=['Date','Confirmed']).set_index('Date')
train = df.iloc[:200, :]
train.index = pd.to_datetime(train.index)# convert index to datetime format
df
```

Out[6]:

	Date	Confirmed
0	2020-01-31	0.0
1	2020-02-01	0.0
2	2020-02-02	1.0
3	2020-02-03	1.0
4	2020-02-04	0.0
...	...	...
836	2022-05-16	324.0
837	2022-05-17	596.0
838	2022-05-18	501.0
839	2022-05-19	556.0
840	2022-05-20	558.0

841 rows × 2 columns

Wir werden ein einfaches exponentielles Glättungsmodell im Trainingssatz trainieren, indem wir die Python-Bibliothek statsmodels verwenden. Wir werden das Modell mit verschiedenen Alpha-Werten anpassen und die aus den Stichproben resultierenden Prognosen überprüfen:

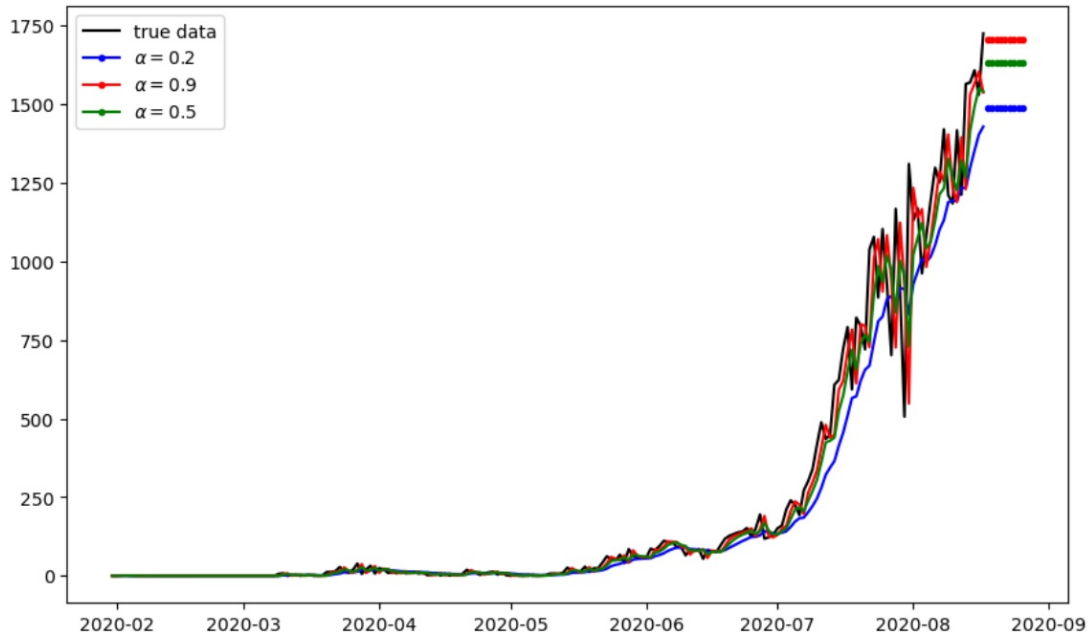
In [90]:
```
#simple exponential smoothing
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from statsmodels.tsa.holtwinters import SimpleExpSmoothing, Holt
import numpy as np
train_data = pd.Series(train.Confirmed, train.index)

# building ES model
model = SimpleExpSmoothing(train_data, initialization_method="heuristic")


# The smoothing_level value of the simple exponential smoothing, if the value is set then this value will be used as the value.
fm1 = model.fit(smoothing_level=.2)#fitting the model
fcast_1 = fm1.forecast(9).rename(r"$\alpha=0.2$")# The number of out of sample forecasts from the end of the sample.
fm2 = model.fit(smoothing_level=.9)
fcast_2 = fm2.forecast(9).rename(r"$\alpha=0.9$")
fm3 = model.fit(smoothing_level=.5)
fcast_3 = fm3.forecast(9).rename(r"$\alpha=0.5$")
```

In [91]:
```
plt.figure(figsize=(12, 8))
(line0,) = plt.plot(train_data,  color="black")# plotting real temporal data
plt.plot(fm1.fittedvalues, color="blue")
(line1,) = plt.plot(fcast_1, marker=".", color="blue")
plt.plot(fm2.fittedvalues, color="red")
(line2,) = plt.plot(fcast_2, marker=".", color="red")
plt.plot(fm3.fittedvalues, color="green")
(line3,) = plt.plot(fcast_3, marker=".", color="green")
plt.legend([line1, line2, line3], [fcast_1.name, fcast_2.name, fcast_3.name])
plt.legend([line0, line1, line2, line3], ['true data',fcast_1.name, fcast_2.name, fcast_3.name])
# this will results in the forecasts with same value for all dates (9 out of the sample forecasts)
# because no trends are incorporated
```

Out[13]: <matplotlib.legend.Legend at 0x1c663aff190>



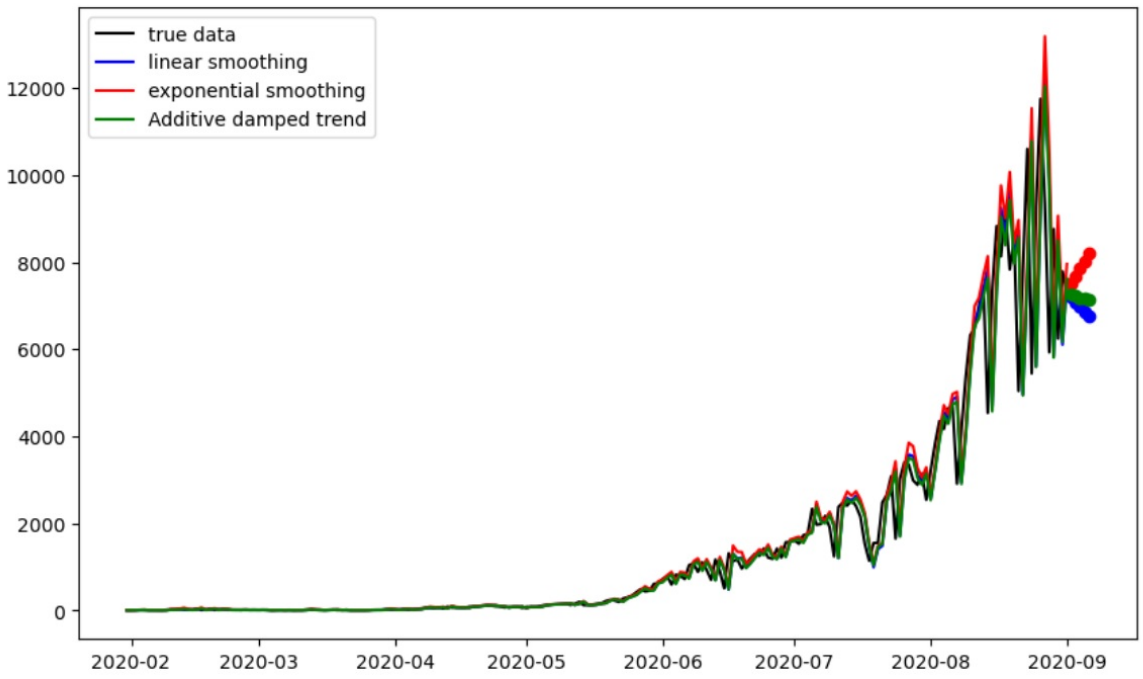
Wie in der Abbildung dargestellt, führt dieses Modell zu Prognosen mit demselben Wert für alle Daten (9 der Beispielprognosen). Dies liegt daran, dass keine Trends in die Berechnung der Prognosen einbezogen werden. Jetzt werden wir die exponentielle Holt-Glättung so aufbauen, dass sie den Trend berücksichtigt. Anschließend werden wir das Modell mit verschiedenen Arten der Glättung anpassen: lineare Glättung, exponentielle Glättung und additiv gedämpfter Trend. Schauen wir uns die resultierenden Prognosen an:

In [94]:
```
# Holt exponential smoothing
# preparing the data; removing rows with zero values
df = pd.read_csv('Covid19_ts.csv',parse_dates=['Date'])
# remove zeros from the dataset, this will result in disonnection in the time series,
# thats why, we generate the time series manually..
df1 = df[df['Confirmed']> 0]
#get values without index
df1 = df1['Confirmed'].values
# convert from np array into list
df1 = df1.tolist()
# select the first 215 values
df1 = df1[:215]
# generate time sequence
index = pd.date_range(start='01/31/2020', end='9/01/2020')
len(index)
# join datetime into the data
train_data = pd.Series(df1, index)

# linear smoothing
Holt_fit1 = Holt(train_data, initialization_method="estimated").fit(
    smoothing_level=0.9, smoothing_trend=0.2, optimized=False)
fcast_holt_1 = Holt_fit1.forecast(5).rename("Holt's linear trend")
#exponential smoothing
Holt_fit2 = Holt(train_data, exponential=True, initialization_method="estimated").fit(
    smoothing_level=0.9, smoothing_trend=0.2, optimized=False)
fcast_holt_2 = Holt_fit2.forecast(5).rename("Exponential trend")

#Additive damped trend
Holt_fit3 = Holt(train_data, damped_trend=True, initialization_method="estimated").fit(
    smoothing_level=0.9, smoothing_trend=0.2)
fcast_holt_3 = Holt_fit3.forecast(5).rename("Additive damped trend")
```

```
plt.figure(figsize=(12, 8))
(line0,) = plt.plot(train_data,  color="black", label ='true data')
plt.plot(Holt_fit1.fittedvalues, color="blue", label ='linear smoothing')
(line1,) = plt.plot(fcast_holt_1, marker="o", color="blue")
#plt.plot(fcast_holt_2 , color="blue")
plt.plot(Holt_fit2.fittedvalues, color="red",label ='exponential smoothing')
(line2,) = plt.plot(fcast_holt_2, marker="o", color="red")
plt.plot(Holt_fit3.fittedvalues, color="green", label ='Additive damped trend')
(line3,) = plt.plot(fcast_holt_3, marker="o", color="green")
plt.legend(loc="upper left")
plt.show()
# forecasts show clearly the trend now..
```




```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "d835e8e8",
   "metadata": {},
   "outputs": [],
   "source": [
    "# downloading dataset\n",
    "import pandas as pd\n",
    "df = pd.read_csv('Covid19_ts.csv',parse_dates=['Date'])\n",
    "df['Date'] = pd.to_datetime(df['Date'])\n",
    "df = pd.DataFrame(df, columns=['Date','Confirmed']).set_index('Date')\n",
    "train = df.iloc[:200, :]\n",
    "train.index = pd.to_datetime(train.index)# convert index to datetime format\n",
    "df"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "8891bebb",
   "metadata": {},
   "outputs": [],
   "source": [
    "#simple exponential smoothing \n",
    "import matplotlib.pyplot as plt\n",
    "import matplotlib.dates as mdates\n",
    "from statsmodels.tsa.holtwinters import SimpleExpSmoothing, Holt\n",
    "import numpy as np\n",
    "train_data = pd.Series(train.Confirmed, train.index)\n",
    "\"\n",
    "\"# building ES model \n",
    "\"model = SimpleExpSmoothing(train_data, initialization_method=\"heuristic\")\n",
    "\"\n",
    "\"\n",
    "\"# The smoothing_level value of the simple exponential smoothing, if the value is set then this value will be used as th\n",
    "\"fm1 = model.fit(smoothing_level=.2)#fitting the model\n",
    "\"fcast_1 = fm1.forecast(9).rename(r\"$\\alpha=0.2$\")# The number of out of sample forecasts from the end of the sample.\n",
    "\"fm2 = model.fit(smoothing_level=.9)\n",
    "\"fcast_2 = fm2.forecast(9).rename(r\"$\\alpha=0.9$\")\n",
    "\"fm3 = model.fit(smoothing_level=.5)\n",
    "\"fcast_3 = fm3.forecast(9).rename(r\"$\\alpha=0.5$\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "77347cf3",
   "metadata": {},
   "outputs": [],
   "source": [
    "fcast_2"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "8a7f3069",
   "metadata": {},
   "outputs": [],
   "source": [
    "plt.figure(figsize=(10, 6))\n",
    "(line0,) = plt.plot(train_data,  color=\"black\")# plotting real temporal data\n",
    "plt.plot(fm1.fittedvalues, color=\"blue\")\n",
    "(line1,) = plt.plot(fcast_1, marker=\".\", color=\"blue\")\n",
    "plt.plot(fm2.fittedvalues, color=\"red\")\n",
```

```
    "(line2,) = plt.plot(fcast_2, marker=\".\", color=\"red\")\n",
    "plt.plot(fm3.fittedvalues, color=\"green\")\n",
    "(line3,) = plt.plot(fcast_3, marker=\".\", color=\"green\")\n",
    "plt.legend([line1, line2, line3], [fcast_1.name, fcast_2.name, fcast_3.name])\n",
    "plt.legend([line0, line1, line2, line3], ['true data',fcast_1.name, fcast_2.name, fcast_3.name])\n",
    "# this will results in the forecasts with same value for all dates (9 out of the sample forecasts)\n",
    "# because no trends are incorporated "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "cddc11a1",
   "metadata": {},
   "outputs": [],
   "source": [
    "# Holt exponential smoothing\n",
    "# preparing the data; removing rows with zero values\n",
    "df = pd.read_csv('Covid19_ts.csv',parse_dates=['Date'])\n",
    "# remove zeros from the dataset, this will result in disonnection in the time series,\n",
    "# thats why, we generate the time series manually..\n",
    "df1 = df[df['Confirmed']> 0]\n",
    "#get values without index\n",
    "df1 = df1['Confirmed'].values\n",
    "# convert from np array into list \n",
    "df1 = df1.tolist()\n",
    "# select the first 215 values\n",
    "df1 = df1[:215]\n",
    "# generate time sequence\n",
    "index = pd.date_range(start='01/31/2020', end='9/01/2020')\n",
    "len(index)\n",
    "# join datetime into the data\n",
    "train_data = pd.Series(df1, index)\n",
    "\n",
    "# linear smoothing\n",
    "Holt_fit1 = Holt(train_data, initialization_method=\"estimated\").fit(\n",
    "    smoothing_level=0.9, smoothing_trend=0.2, optimized=False)\n",
    "fcast_holt_1 = Holt_fit1.forecast(5).rename(\"Holt's linear trend\")\n",
    "#exponential smoothing \n",
    "Holt_fit2 = Holt(train_data, exponential=True, initialization_method=\"estimated\").fit(\n",
    "    smoothing_level=0.9, smoothing_trend=0.2, optimized=False)\n",
    "fcast_holt_2 = Holt_fit2.forecast(5).rename(\"Exponential trend\")\n",
    "\n",
    "#Additive damped trend\n",
    "Holt_fit3 = Holt(train_data, damped_trend=True, initialization_method=\"estimated\").fit(\n",
    "    smoothing_level=0.9, smoothing_trend=0.2)\n",
    "fcast_holt_3 = Holt_fit3.forecast(5).rename(\"Additive damped trend\")\n",
    "\n",
    "plt.figure(figsize=(10, 6))\n",
    "(line0,) = plt.plot(train_data,  color=\"black\", label ='true data')\n",
    "plt.plot(Holt_fit1.fittedvalues, color=\"blue\", label ='linear smoothing')\n",
    "(line1,) = plt.plot(fcast_holt_1, marker=\"o\", color=\"blue\")\n",
    "#plt.plot(fcast_holt_2 , color=\"blue\")\n",
    "plt.plot(Holt_fit2.fittedvalues, color=\"red\",label ='exponential smoothing')\n",
    "(line2,) = plt.plot(fcast_holt_2, marker=\"o\", color=\"red\")\n",
    "plt.plot(Holt_fit3.fittedvalues, color=\"green\", label ='Additive damped trend')\n",
    "(line3,) = plt.plot(fcast_holt_3, marker=\"o\", color=\"green\")\n",
    "plt.legend(loc=\"upper left\")\n",
    "plt.show()\n",
    "# forecasts show clearly the trend now..\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "1c7d91ee",
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.13"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Covid19_ts.csv (14.89 KB)