Jhirsc21

May 3, 2024

[1]:
```
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime, timedelta
```

[2]:
```
%matplotlib inline
```

1 Module 15 Lab - Time Series

1.1 Directions

The due dates for each are indicated in the Syllabus and the course calendar. If anything is unclear, please email EN685.648@gmail.com the official email for the course or ask questions in the Lab discussion area on Canvs.

The Labs also present technical material that augments the lectures and "book". You should read through the entire lab at the start of each module.

Please follow the directions and make sure you provide the requested output. Failure to do so may result in a lower grade even if the code is correct or even 0 points.

1. Show all work/steps/calculations using Code and Markdown cells.
2. Submit your notebook (.ipynb).
3. You may use any core Python libraries or Numpy/Scipy. **Additionally, code from the Module notebooks and lectures is fair to use and modify.** You may also consult Stackoverflow (SO). If you use something from SO, please place a comment with the URL to document the code.

[3]:
```
import numpy as np
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import random
import patsy
from patsy.highlevel import dmatrices
import sklearn.linear_model as linear

sns.set(style="whitegrid")
# load whatever other libraries you need including models.py
```

This lab covers time series data. The exact flow may look a little different from the ETL/EDA/Modeling sequence that we've become familiar with from linear models, but we will follow the same general process. Complete each of the following according to the course notes in Module 15.

1.2 1: Load the data, cleaning and transforming if necessary.

[4]:
```
df = pd.read_csv('https://raw.githubusercontent.com/
↪fundamentals-of-data-science/datasets/master/timeseries.csv')
# add the rest of your code below here
```

Let's look at the data

[5]:
```
df.head()
```

[5]:

	date	value
0	2019-01-01	64.51
1	2019-01-02	1.67
2	2019-01-03	84.41
3	2019-01-04	119.14
4	2019-01-05	20.78

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    180 non-null    object
 1   value   180 non-null    float64
dtypes: float64(1), object(1)
memory usage: 2.9+ KB
```

Only 2 columns with dates and values. We have 180 observations. I'm going to rearrange the time format. Note all the observations have the same year.

[7]:
```
days_of_the_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",␣
↪"Saturday", "Sunday"]
```

[8]:
```
calendar = []

today = datetime(2018, 12, 31, 0, 0)
for _ in range(0, 180):
    today = today + timedelta(days=1)
    date = today.strftime("%m/%d/%y")
    day = days_of_the_week[today.weekday()]
    calendar.append((date, day))
```

It might be useful to look at the days of the week later. I'll concat them into our dataframe along with just the month and day

[9]:
```
dates = pd.Series([i[0][:5] for i in calendar], name='date_trunc')
days = pd.Series([i[1] for i in calendar], name='day')
df = pd.concat([df, dates, days], axis=1)
```

Here is the adjusted dataframe

[10]:
```
df
```

[10]:
```
           date   value date_trunc        day
0    2019-01-01   64.51      01/01    Tuesday
1    2019-01-02    1.67      01/02  Wednesday
2    2019-01-03   84.41      01/03   Thursday
3    2019-01-04  119.14      01/04     Friday
4    2019-01-05   20.78      01/05   Saturday
..          …       …          …        …
175  2019-06-25  178.84      06/25    Tuesday
176  2019-06-26  354.54      06/26  Wednesday
177  2019-06-27  279.00      06/27   Thursday
178  2019-06-28  286.76      06/28     Friday
179  2019-06-29  187.20      06/29   Saturday

[180 rows x 4 columns]
```

Good enough for now. Let's move on to EDA.

1.3 2: Perform EDA on the time variable and describe what kind of time series (trend, seasonality, etc).

Time is an ordered variable, technically numeric but we wouldn't traditionally perform mathematical operations on time series data.

[11]:
```
df['date'].describe()
```

[11]:
```
count             180
unique            180
top        2019-01-01
freq                1
Name: date, dtype: object
```

Here are the head and tail of the dataframe once again

[12]:
```
df
```

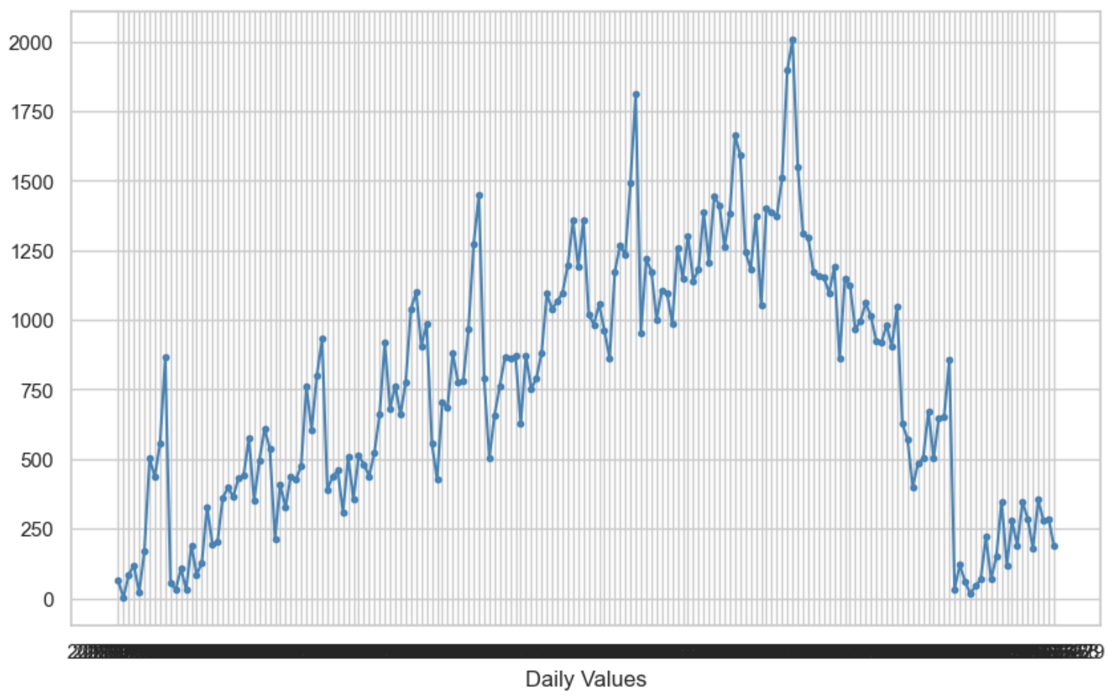
[12]:
```
           date   value date_trunc        day
0    2019-01-01   64.51      01/01    Tuesday
1    2019-01-02    1.67      01/02  Wednesday
2    2019-01-03   84.41      01/03   Thursday
```

```
3     2019-01-04  119.14       01/04      Friday
4     2019-01-05   20.78       01/05    Saturday
..           …       …       …        …
175   2019-06-25  178.84       06/25     Tuesday
176   2019-06-26  354.54       06/26   Wednesday
177   2019-06-27  279.00       06/27    Thursday
178   2019-06-28  286.76       06/28      Friday
179   2019-06-29  187.20       06/29    Saturday

[180 rows x 4 columns]
```

As such, we can't really get a mean, median, or other usual summary statistics.

We see the date ranges we have are from Jan 1st 2019 to Jun 29th 2019, the first half of the year. Let's plot it to see what values we get for each date.

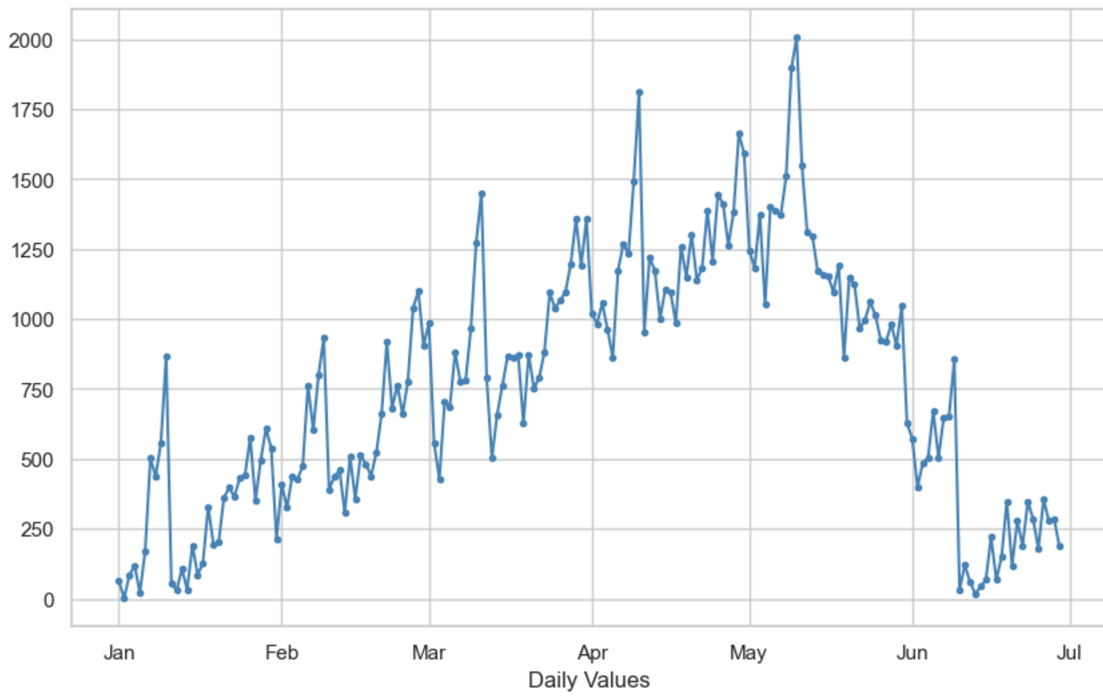
[13]:
```
figure = plt.figure(figsize=(10, 6))

axes = figure.add_subplot(1, 1, 1)

axes.plot(df["date"], df[ "value"], color="steelblue", marker=".")
axes.set_xlabel("Daily Values")

plt.show()
```



We definitely see a pattern here, we steadily increase until about 75% through the data, then start decreasing. Here is another plot only showing by month, which might be easier to read:

[14]:
```
import matplotlib.dates as mdates
```

[15]:
```
figure = plt.figure(figsize=(10, 6))

axes = figure.add_subplot(1, 1, 1)

axes.xaxis.set_major_locator(mdates.MonthLocator())
fmt = mdates.DateFormatter('%b')
axes.xaxis.set_major_formatter(fmt)

axes.plot(df["date"], df[ "value"], color="steelblue", marker=".")
axes.set_xlabel("Daily Values")

plt.show()
```



A bit better. We see that values increase until beginning of May, then decrease afterwards, and rise a bit in Jun as well, following a sharp drop.

I'll consider this good enough for the purposes of our EDA. Let's move on to creating the models in the next step.

1.4 3: Create models with each of Simple Exponential Smoothing, Holts Trend Correction and Holts Winters Seasonality Adjustment. Choose an error (SSE, etc) and compare the three models. How are they different, and why?

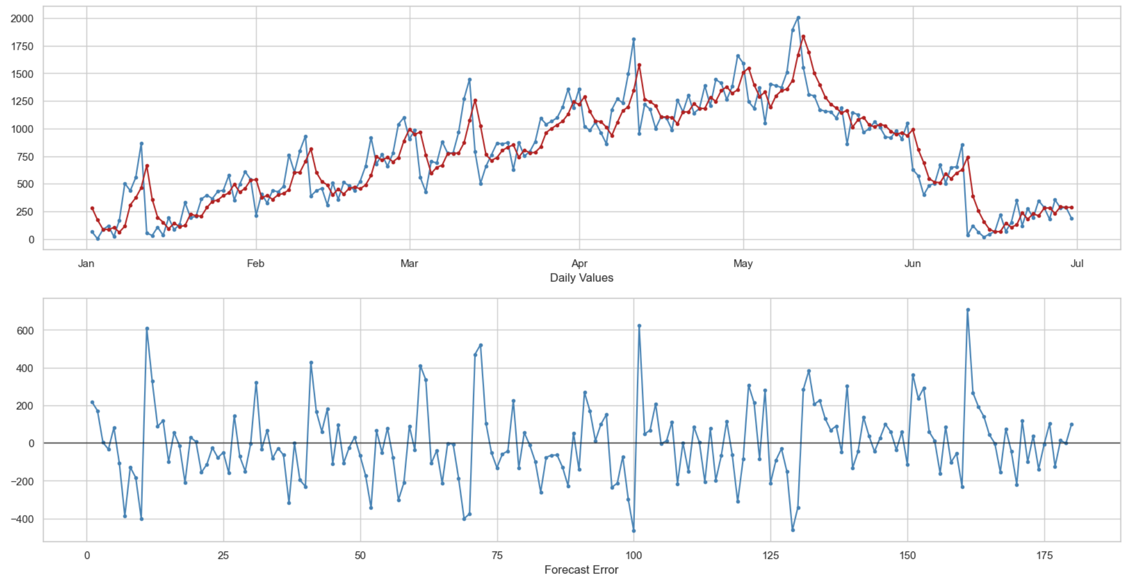
We'll start with simple exponential smoothing. I'll take the average of the first 10 points since our data is fairly noisy. I'll start with an α of 0.5.

[16]:
```
df['t'] = pd.Series(range(1, 181))
```

[17]:
```
alpha = 0.5

n = len(df['value'])
level = df['value'][0:10].mean()
ses = [level]
for i in range(1, n):
    level = level + alpha * (df['value'][i-1] - level)
    ses.append(level)
```

Let's look at a plot again compared to the original data

[18]:
```
df['ses'] = pd.Series(ses)

figure = plt.figure(figsize=(20, 10))

axes = figure.add_subplot(2, 1, 1)

axes.xaxis.set_major_locator(mdates.MonthLocator())
fmt = mdates.DateFormatter('%b')
axes.xaxis.set_major_formatter(fmt)

axes.plot( df["t"], df[ "value"], color="steelblue", marker=".")
axes.plot( df["t"], df[ "ses"], color="firebrick", marker=".")
axes.set_xlabel( "Daily Values")

axes = figure.add_subplot(2, 1, 2)

axes.plot( df["t"], df[ "ses"] - df[ "value"], color="steelblue", marker=".")
axes.axhline(y=0.0, xmin=0, xmax=40, c="black", alpha=0.5)
axes.set_xlabel( "Forecast Error")

plt.show()
```



Here we can see the smoothing from ses. The forecast errors seem to exhibit somewhat of a cyclic pattern, but no major upward/downward trends.

Now let's do Holts Trend Correction.

[19]:
```
def lm( formula, data=None):
    if data is None:
        raise ValueError( "The parameter 'data' must be assigned a non-nil␣
 ↪reference to a Pandas DataFrame")

    result = {}
    result[ "formula"] = formula
    result[ "n"] = data.shape[ 0]

    y, X = dmatrices( formula, data, return_type="matrix")
    model = linear.LinearRegression( fit_intercept=False).fit( X, y)

    result[ "coefficients"] = model.coef_[ 0]

    result[ "r^2"] = model.score( X, y)

    y_hat = model.predict( X)
    result[ "residuals"] = y - y_hat

    sum_squared_error = sum([ e**2 for e in result[ "residuals"]])[ 0]

    n = len( result[ "residuals"])
    k = len( result[ "coefficients"])
```

```
    result[ "sigma"] = np.sqrt( sum_squared_error / (n - k))

    return result
```

[20]:
```
def describe_fit( result):
    formula = result[ "formula"]
    print("regression: ", formula)
    print("n: ", result[ "n"])
    print("----------")
    variables = formula.split("~")[1].split( "+")
    variables = ["intercept"] + variables
    coefficients = result[ "coefficients"]
    for variable, coefficient in zip( variables, coefficients):
        print(variable.strip() + ": ", coefficient)
    print("----------")
    print("sigma", result[ "sigma"])
    print("R^2", result[ "r^2"])
```

[21]:
```
def holts_one_step( alpha, gamma, level, trend, error):
    this_trend = trend + gamma * alpha * error
    this_level = level + trend + alpha * error
    return (this_trend, this_level)

def holts_forecast( level_0, trend_0, alpha, gamma, actual):
    n = len( actual)
    trend, level = trend_0, level_0
    trends = []
    levels = []
    forecasts = []

    for i in range( 0, n):
        forecast = level + trend
        forecasts.append( forecast)

        error = actual[ i] - forecast

        trend, level = holts_one_step( alpha, gamma, level, trend, error)

        trends.append( trend)
        levels.append( level)

    return (pd.Series( forecasts), pd.Series( levels), pd.Series( trends))
```

We start by regressing time as our independent variable and values as our dependent variable in order to find an intercept/starting level.

[22]:
```
describe_fit(lm('value ~ t', data=df.loc[0:30]))
```

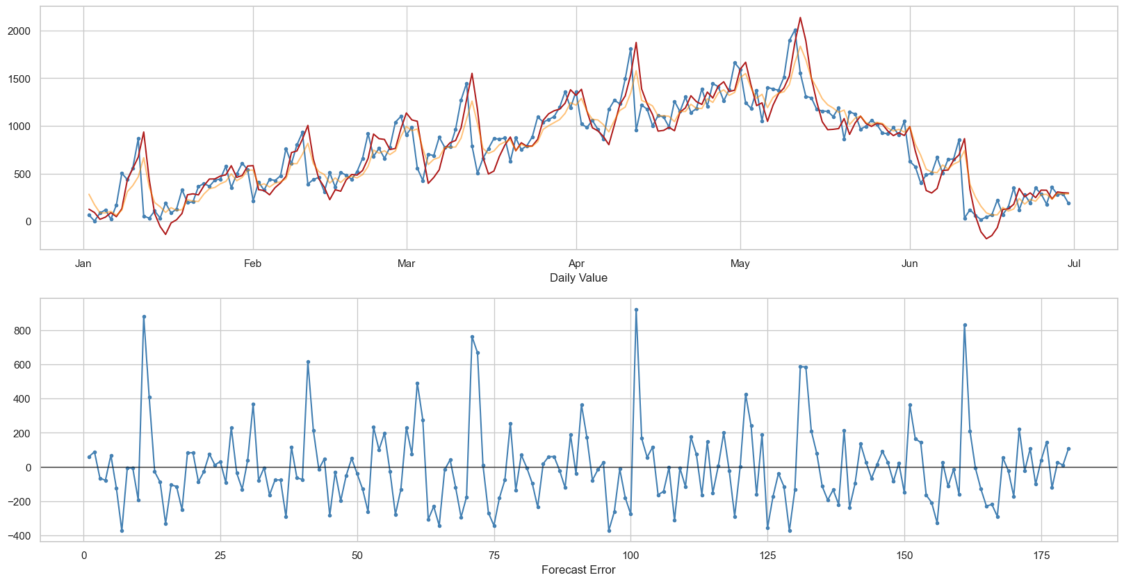
```
regression:  value ~ t
n:  31
----------
intercept:  114.38516129032266
t:  10.908427419354837
----------
sigma 198.86938888826222
R^2 0.20464574007030012
```

We see intercept of 114.39 and slope of 10.9084. Now for the smoothing.

[23]:
```
holts, levels, trends = holts_forecast(114.39, 10.9084, 0.5, 0.5, df[ "value"])
```

Now let's plot again

[24]:
```
df[ "holts"] = holts
df[ "holts_level"] = levels
df[ "holts_trend"] = trends

figure = plt.figure(figsize=(20, 10))

axes = figure.add_subplot(2, 1, 1)

axes.xaxis.set_major_locator(mdates.MonthLocator())
fmt = mdates.DateFormatter('%b')
axes.xaxis.set_major_formatter(fmt)

axes.plot( df["t"], df[ "value"], color="steelblue", marker=".")
axes.plot( df["t"], df[ "holts"], color="firebrick")
axes.plot( df["t"], df[ "ses"], color="darkorange", alpha=0.5)
axes.set_xlabel( "Daily Value")

axes = figure.add_subplot(2, 1, 2)

axes.plot(df["t"], df[ "holts"] - df[ "value"], color="steelblue", marker=".")
axes.axhline(y=0.0, xmin=0, xmax=40, c="black", alpha=0.5)
axes.set_xlabel( "Forecast Error")

plt.show()
```



Very interesting. I think the forecast error is slightly higher than with ses.

Finally let's create a model for Holts-Winters which includes seasonality. We start with finding moving averages. Let's use 15 days as our window (basically half a month).

[25]:
```
ma_sales = df['value'].rolling(window=15, center=True).mean()

shifted_ma = []
for i in range(0, 181):
    current = None
    if 7 < i < 172:
        current = df["value"][ i-7:i+8].mean()
    shifted_ma.append( current)
shifted_ma_sales = pd.Series( shifted_ma)
```

[26]:
```
smoothed_sales = (ma_sales + shifted_ma_sales) / 2.0
smoothed_sales[172] = None

seasonal_factor_estimate = df[ "value"] / smoothed_sales
seasonal_factor_estimate[0:30]
```

[26]:
```
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
5          NaN
6          NaN
7          NaN
```

```
8     2.564082
9     3.836202
10    0.224706
11    0.126020
12    0.412970
13    0.119562
14    0.721087
15    0.326481
16    0.507409
17    1.474398
18    0.753366
19    0.726282
20    1.185543
21    1.153631
22    0.995783
23    1.147002
24    1.119017
25    1.461702
26    0.855580
27    1.164083
28    1.406158
29    1.176989
dtype: float64
```

Sure. We have 163 values and need 15, so look for the indices.

[27]:
```
initial_seasonal_factors = [
    seasonal_factor_estimate[[0, 16, 32]].mean(),
    seasonal_factor_estimate[[1, 17, 33]].mean(),
    seasonal_factor_estimate[[2, 18, 34]].mean(),
    seasonal_factor_estimate[[3, 19, 35]].mean(),
    seasonal_factor_estimate[[4, 20, 36]].mean(),
    seasonal_factor_estimate[[5, 21, 37]].mean(),
    seasonal_factor_estimate[[6, 22, 38]].mean(),
    seasonal_factor_estimate[[7, 23, 39]].mean(),
    seasonal_factor_estimate[[8, 24, 40]].mean(),
    seasonal_factor_estimate[[9, 25, 41]].mean(),
    seasonal_factor_estimate[[10, 26, 42]].mean(),
    seasonal_factor_estimate[[11, 27, 43]].mean(),
    seasonal_factor_estimate[[12, 28, 44]].mean(),
    seasonal_factor_estimate[[13, 29, 45]].mean(),
    seasonal_factor_estimate[[14, 30, 46]].mean(),
    seasonal_factor_estimate[[15, 31, 47]].mean()

    ]
initial_seasonal_factors
```

[27]:
```
[0.560131210756337,
 1.1613958620636133,
 0.7836418600140784,
 0.8202005261131294,
 1.3523946503064268,
 1.1815725377360222,
 1.2837133584355724,
 1.4770485595202205,
 1.4744693521841727,
 2.042819204677536,
 0.647250109293286,
 0.6189842667230683,
 0.9112604117185996,
 0.64502548921687,
 0.6964849330467725,
 0.6806972015789926]
```

Now to deseasonsalize the data like so

[28]:
```
from copy import deepcopy
```

[29]:
```
seasonal_factors = pd.Series(
                                initial_seasonal_factors
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                                + deepcopy(initial_seasonal_factors)
                            )
```

[30]:
```
df[ "seasonal"] = seasonal_factors

df[ "deseasonalized"] = df[ "value"] / df[ "seasonal"]
print(df[["value", "deseasonalized"]])
```

```
      value  deseasonalized
0     64.51      115.169444
1      1.67        1.437925
2     84.41      107.715022
3    119.14      145.257161
4     20.78       15.365337
..      …             …
```

```
175  178.84       262.730623
176  354.54       632.958837
177  279.00       240.228168
178  286.76       365.932468
179  187.20       228.236869

[180 rows x 2 columns]
```

Now we fit the data with trend line

[31]:
```
describe_fit( lm( "deseasonalized ~ t", data=df))
```

```
regression:  deseasonalized ~ t
n:  180
----------
intercept:  571.384909046989
t:  3.0296928328743227
----------
sigma 588.7420604102006
R^2 0.06742684582273151
```

We see initial level of 571.38 and slope of 3.0297.

[32]:
```
def holt_winters_one_step( alpha, gamma, delta, level, trend, error, factor):
    this_factor = factor + delta * (1.0 - alpha) * error / ( level + trend)
    this_trend = trend + (gamma * alpha * error) / factor
    this_level = level + trend + (alpha * error) / factor
    return (this_factor, this_trend, this_level)

def holt_winters_forecast( level_0, trend_0, alpha, gamma, delta, actual,␣
 ↪seasonality):
    n = len( actual)
    trend, level, factor = trend_0, level_0, seasonality[ 0]
    factors = []
    trends = []
    levels = []
    forecasts = []

    for i in range( 0, n):
        if i < 12:
            factor = seasonality[ i]
        else:
            factor = factors[ i - 12]

        forecast = (level + trend) * factor
        forecasts.append( forecast)

        error = actual[ i] - forecast
```

```
        factor, trend, level = holt_winters_one_step( alpha, gamma, delta,␣
↪level, trend, error, factor)

        factors.append( factor)
        trends.append( trend)
        levels.append( level)


        print(forecast, error, level, trend, factor)

    return (pd.Series( forecasts), pd.Series( levels), pd.Series( trends), pd.
↪Series( factors))
```

Now we can use our helper functions as found in the module notes.

[33]:
```
winters, levels, trends, factors = holt_winters_forecast(571.38, 3.0297, 0.5, 0.
↪5, 0.0, df[ "value"], initial_seasonal_factors)
```

```
321.74480073118434 -257.23480073118435 344.78957190193887 -111.78036404903057
0.560131210756337
270.6159298230881 -268.9459298230881 117.22356636404744 -169.673184793461
1.1613958620636133
-41.101716543054316 125.51171654305432 27.632701663083466 -129.63202474721248
0.7836418600140784
-83.65989845678568 202.79989845678568 21.62891903480721 -67.81790368774438
0.8202005261131294
-62.46573574771787 83.24573574771787 -15.411823663407969 -52.42932319297978
1.3523946503064268
-80.15923605402425 248.90923605402423 37.48849990865648 0.23550018954233565
1.1815725377360222
48.42680285968267 454.00319714031735 214.55599851784564 88.65149939936575
1.2837133584355724
447.8521980343474 -10.672198034347389 299.5948211485431 86.8451610150316
1.4770485595202205
569.7939101587892 -12.9539101587892 382.04724584131736 84.64879285390295
1.4744693521841727
953.3756305935265 -88.02563059352644 445.15090381691823 73.87622541475193
2.042819204677536
335.940366121379 -281.580366121379 301.50660503366845 -34.88403668424894
0.647250109293286
165.0351749615866 -133.9151749615866 158.4492413676694 -88.970700175124
0.6189842667230683
38.917099399764496 68.0829006002355 130.2526056374673 -58.58366795266305
0.560131210756337
83.23600766562662 -50.71600766562661 49.83486313613467 -69.50070522699784
1.1613958620636133
-15.410977074827164 206.46097707482718 112.06587593599035 -3.6348462135710804
0.7836418600140784
```

88.93518762531666 -3.9951876253166603 105.99553529263052 -4.852593428465456
0.8202005261131294
136.78517349335078 -9.035173493350783 97.80250662534496 -6.522811047875512
1.3523946503064268
107.85358154724214 221.67641845275784 185.08537037654094 40.38002635166023
1.1815725377360222
289.43294164496785 -94.85294164496784 188.52064538567305 21.90765068039616
1.2837133584355724
310.812811586682 -107.732811586682 173.95934895789964 3.6731771263113657
1.4770485595202205
261.9137156622248 100.25628433777524 211.62993816648412 20.671883167447927
1.4744693521841727
474.55062190252613 -77.81062190252612 213.2569098399634 11.149427420463612
2.042819204677536
145.24702631791737 220.25297368208263 394.5515180179634 96.22201779923179
0.647250109293286
303.7810971948941 127.06890280510595 593.4166154852898 147.54355763327905
0.6189842667230683
415.034918891129 26.29508110887099 764.4324244446154 159.27968329630232
0.560131210756337
1072.7954196683604 -496.74541966836046 709.8550431971696 52.35115102442823
1.1613958620636133
597.2966797540648 -246.14667975406485 605.1531497673093 -26.175371202716036
0.7836418600140784
474.87787858649034 20.212121413509635 591.2992297036783 -20.014645633173487
0.8202005261131294
772.6022152994829 -163.83221529948287 510.7134278394588 -50.300223748696496
1.3523946503064268
544.0115979646952 -5.8215979646951155 457.94970829225224 -51.53197164795152
1.1815725377360222
521.7238776354393 -308.0338776354393 286.44006576813564 -111.52080708603407
1.2837133584355724
258.36423906874296 150.77576093125703 225.95879965028493 -86.00103660194239
1.4770485595202205
206.36343221503557 118.73656778496445 180.2219325304266 -65.86895186090035
1.4744693521841727
233.60246502382728 205.89753497617272 164.74841813768785 -40.67123312681956
2.042819204677536
80.30897155908777 345.99102844091226 391.3548752523501 92.96761199392135
0.647250109293286
299.78799962562596 176.78200037437404 627.1225630141631 164.3676498778672
0.6189842667230683
443.33837124900384 318.3316287509961 1075.6483017808441 306.4466943222741
0.560131210756337
1605.1594094529873 -1001.2194094529873 951.0535906025073 90.92599157196858
1.1613958620636133
816.5388178718985 -16.678817871898445 1031.3377196586073 85.60506031403438
0.7836418600140784

916.117055771822 15.902944228177944 1126.637326441382 90.45233354840452
0.8202005261131294
1645.9855451134551 -1256.4055451134552 752.5782302718496 -141.8033813105639
1.3523946503064268
721.6747882725219 -284.64478827252196 490.3231715645167 -202.02922000894839
1.1815725377360222
370.0867967680609 88.70320323193914 322.84341022134066 -184.75449067606223
1.2837133584355724
203.96403970005713 104.86596029994286 173.5874004936657 -167.0052502018686
1.4770485595202205
9.705178876724963 498.14482112327505 175.5055736187585 -82.5435385383879
1.4744693521841727
189.90463056808784 167.89536943191217 134.0560704819065 -61.996520837619954
2.042819204677536
46.64055138288943 464.85944861711056 431.1629641842063 117.55518643233995
0.647250109293286
339.647902097021 142.712097902979 663.9974117991475 175.19481702364058
0.6189842667230683
470.0577591878172 -31.287759187817244 811.2632734396647 161.23033933207887
0.560131210756337
1129.4500577563967 -607.4700577563967 710.9677723588889 30.467419125651503
1.1613958620636133
581.0196525348397 79.27034746516028 792.0133646973242 55.75650573204341
0.7836418600140784
695.3412937490268 222.6287062509732 983.4858930134998 123.61451702410952
0.8202005261131294
1497.2366718869143 -816.5666718869144 805.2038180546788 -27.3337789673557
1.3523946503064268
919.1098761132272 -155.2298761132272 712.1822073395326 -60.17769484125092
1.1815725377360222
836.9869024543173 -177.76690245431723 582.7651837625594 -94.79735920911203
1.2837133584355724
720.7521723488852 55.49782765111479 506.7545554613134 -85.40399375517906
1.4770485595202205
621.2684897612812 418.1215102387189 563.1376763787262 -14.51043641888316
1.4744693521841727
1120.7462619991982 -21.63626199919827 543.3315530117135 -17.158279892947917
2.042819204677536
340.56570853332704 563.544291466673 961.5106205948371 200.51039384508786
0.647250109293286
719.2727255398929 266.63727446010705 1377.4039319021872 308.201852576219
0.6189842667230683
944.1604089177748 -387.87040891777485 1339.3740431744006 135.08598192421624
0.560131210756337
1712.431771927745 -1285.6217719277452 920.9787299081025 -141.6546656710409
1.1613958620636133
610.7109592524621 91.69904074753788 837.8323225541266 -112.40053651250844
0.7836418600140784

594.9995325705223 92.70046742947773 781.9426419104741 -84.14510857808047
0.8202005261131294
943.6976510757497 -63.23765107574968 674.4176526661173 -95.83504891121866
1.3523946503064268
683.637315408591 92.44268459140903 617.7011011975253 -76.27580018990531
1.1815725377360222
695.0348914984826 85.71510850151742 574.8109115640046 -59.58299491171296
1.2837133584355724
761.0166521158717 204.02334788412827 584.2924530106629 -25.050726732527338
1.4770485595202205
824.5847858596811 447.69521414031885 711.0574332228529 50.85712673983127
1.4744693521841727
1556.4536954152052 -109.05369541520508 735.2226003498364 37.5111469334074
2.042819204677536
500.15200238369 288.81799761631004 995.8453338781555 149.06694023086322
0.647250109293286
708.6826844516114 -206.03268445161143 978.4842277691996 65.8529170609537
0.6189842667230683
584.9658293715297 73.72417062847035 1110.1468776326885 98.7577834622214
0.560131210756337
1404.0168710250432 -641.7968710250432 932.6005635907766 -39.394265289845265
1.1613958620636133
699.9538449768315 167.54615502316847 1000.10854763978 14.056859379579116
0.7836418600140784
831.8190004030145 28.780999596985566 1031.7105064680127 22.829409103905867
0.8202005261131294
1426.1541403540537 -552.7741403540537 850.1712646648754 -79.3549163496157
1.3523946503064268
910.775428807275 -282.785428807275 651.1514865416812 -139.18734723640497
1.1815725377360222
657.2152046661533 215.8647953338467 596.0424087707102 -97.14821250368797
1.2837133584355724
736.8909539492032 14.909046050796746 503.94110076948465 -94.62476025245671
1.4770485595202205
603.5243994405384 186.72560055946155 472.6359342009725 -62.96496341048443
1.4744693521841727
836.8837267296989 45.57627327030116 420.8262098752644 -57.38734386809626
2.042819204677536
235.23584574456748 858.4441542554325 1026.5860342577407 274.1862402571901
0.647250109293286
805.1575725143222 234.31242748567786 1490.0439895507093 368.8220977750794
0.6189842667230683
1041.2089121276888 24.731087872311264 1880.9422432312213 379.8601757277957
0.560131210756337
2625.68657432241 -1528.2365743224098 1602.871465250012 50.894698873293294
1.1613958620636133
1295.9603928819347 -101.29039288193462 1589.1381764861244 18.58070505470284
0.7836418600140784

1318.6518724817984 38.228127518201745 1631.0230165060668 30.232772537322624
0.8202005261131294
2246.673441892862 -1055.8734418928618 1270.8839986589699 -164.95312265488718
1.3523946503064268
1306.737551720766 51.61244827923383 1127.7714514367713 -154.03283493854286
1.1815725377360222
1250.0012696233487 -230.7212696233487 883.8738238218779 -198.9652312767181
1.2837133584355724
1011.6432500218499 -28.20325002184984 675.3614284251762 -203.7388133367099
1.4770485595202205
695.3930917448964 360.9469082551035 594.0215336283542 -142.539354066766
1.4744693521841727
922.296466978084 41.053533021915996 461.5304336919375 -137.51522700159134
2.042819204677536
209.7188779430132 652.1611220569868 827.8089586675092 114.38164898699017
0.647250109293286
583.2011623923823 588.8588376076176 1417.856039932014 352.2143651257476
0.6189842667230683
991.4716791089638 276.42832089103626 2016.823590367485 475.59095778060924
0.560131210756337
2894.6799427663473 -1661.8099427663474 1776.9780647540606 117.87271608359242
1.1613958620636133
1484.8843903447473 9.375609655252674 1900.832856409193 120.86375386936238
0.7836418600140784
1658.1966233916016 152.92337660839826 2114.919774455913 167.47533595804106
0.8202005261131294
3086.698937209377 -2131.798937209377 1494.2379934339538 -226.60322253195892
1.3523946503064268
1497.8024331770912 -279.33243317709116 1149.4310956065651 -285.70506017967375
1.1815725377360222
1108.776649706097 65.443350293903 889.2158964865508 -272.960129649844
1.2837133584355724
910.2396927021866 90.46030729781342 646.8777483263326 -257.6491389050311
1.4770485595202205
573.9056555849728 533.1843444150271 570.0341119653883 -167.2463876329877
1.4744693521841727
822.8224986745893 271.39750132541076 469.2149198238028 -134.0327898872866
2.042819204677536
216.94667023456645 770.3833297654336 930.3024077890709 163.52734903899076
0.647250109293286
677.0634099500897 581.8965900499103 1563.8712597652025 398.54810050756123
0.6189842667230683
1099.2123322812595 50.957667718740595 2007.9066199899407 421.29173036614975
0.560131210756337
2821.260912235319 -1518.4709122353192 1775.471674622443 94.42839249932604
1.1613958620636133
1465.3319666397533 -324.9519666397532 1662.5655797617426 -9.23885118068722
0.7836418600140784

1356.0594526190807 -175.9794526190808 1546.048418572505 -62.8780061849624
0.8202005261131294
2005.8317312056893 -618.5517312056893 1254.4828280845666 -177.22179833645038
1.3523946503064268
1272.8620487236021 -65.47204872360203 1049.5555581699382 -191.07453412553937
1.1815725377360222
1102.0435585292446 341.3664414707555 991.4415635712173 -124.59426436213016
1.2837133584355724
1280.3755546207758 131.59444537922423 911.3937173112683 -102.3210553110396
1.4770485595202205
1192.9528438094014 69.45715619059865 832.6259342631313 -90.54441917958835
1.4744693521841727
1515.9383704488641 -134.24837044886408 709.222912094725 -106.97372067399733
2.042819204677536
389.805854968859 1272.124145031141 1584.9636991248185 384.38353317804814
0.647250109293286
1218.994952510094 372.995047489906 2270.6432971160802 535.0315655846549
0.6189842667230683
1571.5460578331822 -328.5260578331822 2512.4167371719163 388.40250282024556
0.560131210756337
3368.9994619214126 -2185.8494619214125 1959.7751337915806 -82.11955028004496
1.1613958620636133
1471.4095139287997 -98.07951392879977 1815.0762861734395 -113.40919894909314
0.7836418600140784
1395.7082402108053 -344.54824021080526 1491.628060644106 -218.42871223921327
0.8202005261131294
1721.867987556405 -318.36798755640507 1155.4940663393843 -277.2813532719675
1.3523946503064268
1037.672024051105 350.47797594889516 1026.5226833636275 -203.1263681238622
1.1815725377360222
1057.0048491599143 315.76515084008565 946.3852787669892 -141.6318863602502
1.2837133584355724
1188.6598390233846 321.21016097661527 913.4871773951459 -87.26499386604681
1.4770485595202205
1218.2392877083435 677.9107122916566 1056.1051279549865 27.67647834689687
1.4744693521841727
2213.9698790297557 -207.8598790297558 1032.9058659148218 2.2386081533660906
2.042819204677536
669.9973739749756 881.1026260250243 1715.7952869255819 342.56401458206307
0.647250109293286
1274.0920228963166 35.66797710368337 2087.171000787589 356.9698642220351
0.6189842667230683
1369.0395819768817 -73.66958197688177 2378.3798606572627 324.08936204585444
0.560131210756337
3138.63657260167 -1966.5465726016703 1855.8386134346144 -99.225942588397
1.1613958620636133
1376.555220706228 -220.07522070622804 1616.1944313826734 -169.4350623201691
0.7836418600140784

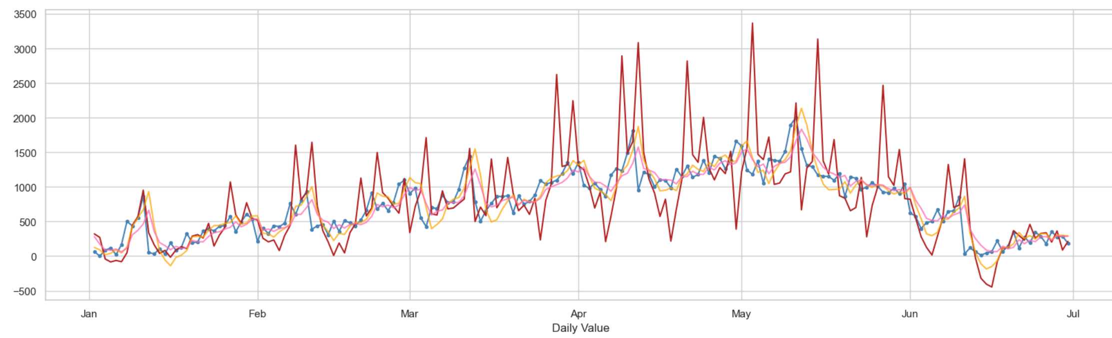
1186.6327956641653 -34.32279566416537 1425.8359518179323 -179.89677094245522
0.8202005261131294
1685.0014828231665 -588.3914828231666 1028.4022796869635 -288.6652215367119
1.3523946503064268
874.0529930559723 315.82700694402786 873.3839553391348 -221.84177294227032
1.1815725377360222
836.3934031271212 24.79659687287881 661.200334160237 -217.01269706058406
1.2837133584355724
656.0867095347329 490.0532904652672 610.077000488101 -134.06801536636004
1.4770485595202205
701.8606599262989 423.239340073701 619.531581724629 -62.306717064916086
1.4744693521841727
1138.3096548507024 -172.42965485070238 515.0210185102636 -83.40864013964075
2.042819204677536
279.36115907272074 716.8688409272793 985.3927722511338 193.48155680061473
0.647250109293286
729.7046621267457 330.6653378732542 1445.9775137771148 327.0331491632979
0.6189842667230683
993.118609316709 19.40139068329097 1790.329275357219 335.69245537170104
0.560131210756337
2469.1528407258897 -1544.0628407258896 1461.2773093124626 3.320244663472181
1.1613958620636133
1147.719951369771 -230.47995136977102 1317.5406118125654 -70.20822641821249
0.7836418600140784
1023.0626787383931 -40.942678738393056 1222.3734409442577 -82.68769864326013
0.8202005261131294
1541.304900918378 -636.4149009183781 904.3938839761445 -200.3336278056867
1.3523946503064268
831.8982636024017 215.6117363975983 795.2995705212829 -154.7139706302741
1.1815725377360222
822.3282918015527 -194.27829180155277 564.9151667195745 -192.54918721599122
1.2837133584355724
550.0026336401037 22.52736635989629 379.991783751757 -188.73628509190436
1.4770485595202205
282.00037121065384 116.96962878934619 230.92049020955008 -168.90378931705564
1.4744693521841727
126.68890759393013 358.90109240606984 149.8612569805412 -124.98151127303225
2.042819204677536
16.103418128374337 486.9865818716256 401.0763464337355 63.11678909008103
0.647250109293286
287.3282476100914 382.8017523899086 773.4108111333106 217.72562689482808
0.6189842667230683
555.1664530574245 -51.046453057424515 945.5699242567518 194.94237000913463
0.560131210756337
1324.5862591930788 -675.7262591930788 849.6010377058615 49.48674172912206
1.1613958620636133
704.562819792358 -51.69281979235802 866.1053531315766 32.99552857741863
0.7836418600140784

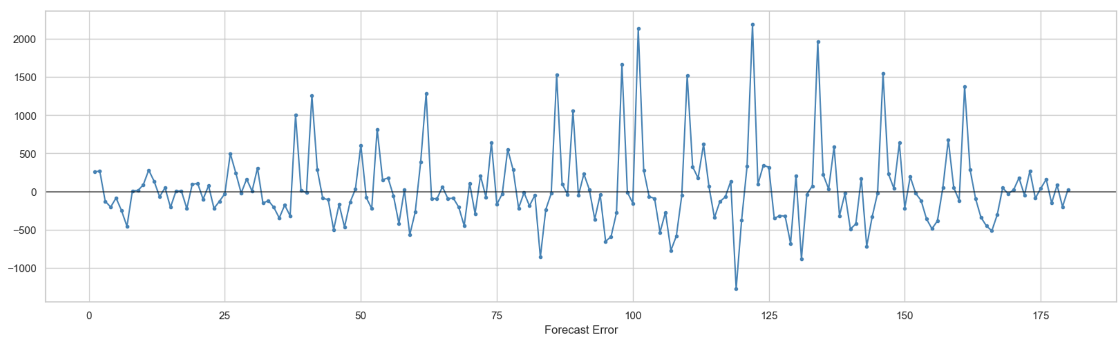
```
737.4430162064965 118.12698379350354 971.1119204931928 69.00104796951739
0.8202005261131294
1406.6432142633064 -1374.2032142633063 532.0500247236403 -185.03042390001752
1.3523946503064268
410.02883038930935 -289.9288303893094 224.33190238369716 -246.37427311998033
1.1815725377360222
-28.296085765756057 89.22608576575605 12.71074808865977 -228.99771370750886
1.2837133584355724
-319.4663510103205 336.57635101032054 -102.35152698992265 -172.02999439304565
1.4770485595202205
-404.567144084853 448.067144084853 -122.4396911166699 -96.05907925989646
1.4744693521841727
-446.35348432367687 513.9134843236768 -92.71341376083019 -33.166400952028376
2.042819204677536
-81.4757238307163 302.2457238307163 107.60467566500313 83.57584423690247
0.647250109293286
118.33773392321599 -49.867733923215994 150.8986124253084 63.43489049860387
0.6189842667230683
120.05488449841786 29.705115501582128 240.84971459998698 76.69299633664124
0.560131210756337
368.79279051026214 -21.502790510262116 308.28540633763686 72.06434403714555
1.1613958620636133
298.0579858395849 -178.5579858395849 266.42144016661086 15.10018893305979
0.7836418600140784
230.90418829977514 46.94581170022488 310.1401255560785 29.40943716126371
0.8202005261131294
459.2050121328202 -268.90501213282016 240.13146309978922 -20.299612647512788
1.3523946503064268
259.747277414102 87.31272258589803 256.7795281433962 -1.8257738019528986
1.1815725377360222
327.2875402314121 -41.50754023141212 238.78677284255315 -9.909264551397982
1.2837133584355724
338.06319392802806 -159.22319392802805 174.97840223206003 -36.85881758094554
1.4770485595202205
203.65309450447583 150.8869054955242 189.28609593600868 -11.275561938498448
1.4744693521841727
363.64333748501735 -84.64333748501735 157.29324847091894 -21.6342047017941
2.042819204677536
87.80533090618873 198.95466909381128 289.3513075765765 55.21192720193173
0.647250109293286
213.27922121910333 -26.079221219103346 323.497092534564 44.678856079959615
0.6189842667230683
```

[34]:
```python
df[ "winters"] = winters
df[ "winters_level"] = levels
df[ "winters_trend"] = trends
df[ "winters_factors"] = factors
```

Finally let's plot it

[35]:
```
figure = plt.figure(figsize=(20, 12))

axes = figure.add_subplot(2, 1, 1)

axes.xaxis.set_major_locator(mdates.MonthLocator())
fmt = mdates.DateFormatter('%b')
axes.xaxis.set_major_formatter(fmt)

axes.plot( df["t"], df[ "value"], color="steelblue", marker=".")
axes.plot( df["t"], df[ "winters"], color="firebrick")
axes.plot( df["t"], df[ "holts"], color="orange", alpha=0.75)
axes.plot( df["t"], df[ "ses"], color="hotpink", alpha=0.75)
axes.set_xlabel( "Daily Value")

axes = figure.add_subplot(2, 1, 2)

axes.plot( df["t"], df[ "winters"] - df[ "value"], color="steelblue", marker=".
↪")
axes.axhline(y=0.0, xmin=0, xmax=40, c="black", alpha=0.5)
axes.set_xlabel( "Forecast Error")

plt.show()
```





Forecast error looks high, but no obvious patterns otherwise.

Finally we can compare the SSE of the 3 models.

[36]:
```
sse = ((df[ "ses"] - df[ "value"])**2.0).sum()
print(sse)
```

```
7110719.622925861
```

[37]:
```
sse = ((df[ "holts"] - df[ "value"])**2.0).sum()
print(sse)
```

```
9734958.41369099
```

[38]:
```
sse = ((df[ "winters"] - df[ "value"])**2.0).sum()
print(sse)
```

```
47289680.31275396
```

The sse for SES was the smallest, while Holts and Holts-Winters kept increasing. For the latter 2 we did not find optimal values for α and γ, so these algorithms could be improved as we will see in the next part. This is consistent with the plot, where Holts-Winters showed much bigger peaks, which could also be me not setting up the cycles/parameters correctly to get proper values.

1.5 4: Choose one of the models and find its optimal parameters (ie,).

I'll choose the Holts Trend Correction to find optimal parameters for α and γ. Since we use SSE as our loss function, we can find optimal parameters using grid search.

[39]:
```
for alpha in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
    print(alpha, " -", end=' ')
    for gamma in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
        holts, _, _ = holts_forecast( 155.88, 0.8369, alpha, gamma, df[␣
 ↪"value"])
        sse = ((holts - df[ "value"])**2.0).sum()
        print(int( sse), end=' ')
    print()
```

```
0.1  - 9668371 9658021 10223737 10995890 11490820 11945158 12636876 13621227
14952151
0.2  - 8556401 9243296 10137535 11260385 12656856 14217400 15630565 16529571
16839805
0.3  - 8160857 8984281 9941736 10912227 11721464 12292740 12725814 13152874
13602004
0.4  - 7851551 8605960 9351104 9981716 10476201 10879794 11217656 11479199
11662035
0.5  - 7624737 8278779 8869546 9356591 9755469 10090511 10380310 10649588
10927982
0.6  - 7511579 8102289 8625723 9074328 9471299 9843231 10215116 10607885
11034101
0.7  - 7523368 8095896 8616303 9093442 9553381 10020895 10514667 11045973
11619880
0.8  - 7662349 8257810 8824146 9378440 9946383 10549436 11202912 11917724
```

```
12703464
0.9  - 7932063 8589389 9245916 9923885 10649620 11445518 12330684 13323767
14445666
```

It seems that an α of 0.6 and γ of 0.1 is best here. We can increase granularity an do another search:

[40]:
```python
for alpha in [0.60, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69]:
    print(alpha, " -", end=' ')
    for gamma in [0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14, 0.
 ↪15]:
        holts, _, _ = holts_forecast( 155.88, 0.8369, alpha, gamma, df[␣
 ↪"value"])
        sse = ((holts - df[ "value"])**2.0).sum()
        print(int( sse), end=' ')
    print()
```

```
0.6  - 7212514 7271065 7330555 7390615 7451014 7511579 7572171 7632668 7692960
7752950 7812552
0.61  - 7209145 7267551 7326847 7386674 7446804 7507071 7567339 7627489 7687416
7747026 7806236
0.62  - 7206906 7265187 7324312 7383931 7443822 7503823 7563801 7623641 7683242
7742514 7801377
0.63  - 7205797 7263974 7322952 7382390 7442070 7501836 7561558 7621126 7680441
7739417 7797977
0.64  - 7205816 7263911 7322766 7382049 7441550 7501113 7560614 7619946 7679014
7737736 7796037
0.65  - 7206963 7264998 7323755 7382911 7442261 7501654 7560970 7620103 7678963
7737472 7795558
0.66  - 7209236 7267234 7325919 7384977 7444206 7503462 7562626 7621597 7680289
7738625 7796539
0.67  - 7212635 7270619 7329258 7388245 7447385 7506536 7565583 7624430 7682992
7741197 7798982
0.68  - 7217159 7275153 7333772 7392718 7451799 7510878 7569843 7628602 7687073
7745187 7802884
0.69  - 7222808 7280836 7339462 7398395 7457448 7516488 7575407 7634114 7692533
7750595 7808246
```

Here we see an α of 0.63 and γ of 0.05 is best. We could keep going but I'll leave it at that for now.

We can compare the original sse from using 0.5 for both parameters with the optimal parameters. Here's the original sse.

[41]:
```python
sse = ((df[ "holts"] - df[ "value"])**2.0).sum()
print(sse)
```

```
9734958.41369099
```

And with optimal parameters.

```
[42]: holts, levels, trends = holts_forecast(114.39, 10.9084, 0.63, 0.05, df[␣
      ↪"value"])
```

```
[43]: holts, levels, trends = holts_forecast( 155.88, 0.8369, 0.66, 0.05, df[␣
      ↪"value"])
      df[ "holts"] = holts
      df[ "holts_level"] = levels
      df[ "holts_trend"] = trends
```

```
[44]: sse = ((df[ "holts"] - df[ "value"])**2.0).sum()
      print(sse)
```

```
7209236.814122682
```

We see the sse is reduced from 9.7 million down to 7.2 million, about a 25% reduction.

1.6 5: Finally, pick one of the models from above and forecast some future values.

I'll do 3 forecastings for the Holts Trend Correction since we just found optimal parameters for it. We can then compare with the actual values.

Here is the first

```
[45]: # t = 36 (index 35)
      print('Forecast: ', round((df[ "holts_level"][35] + df[ "holts_trend"][35]), 2))
      print('Actual: ', df['value'].iloc[35])
```

```
Forecast:  467.36
Actual:  476.57
```

Pretty close. And the second:

```
[46]: # t = 100 (index 99)
      print('Forecast: ', round((df[ "holts_level"][99] + df[ "holts_trend"][99]), 2))
      print('Actual: ', df['value'].iloc[99])
```

```
Forecast:  1717.42
Actual:  1811.12
```

Close enough? Perhaps, I'll consider it good enough for now. Finally the third which will be towards the end of the data:

```
[47]: # t = 154 (index 153)
      print('Forecast: ', round((df[ "holts_level"][153] + df[ "holts_trend"][153]),␣
      ↪2))
      print('Actual: ', df['value'].iloc[153])
```

```
Forecast:  438.36
Actual:  485.59
```

Not wildly off but not as close as what we saw for the first forecasted value. It could be interesting to go back and look at the same forecasted values using the initial α and γ values of 0.5, and compare with the optimal values.