Excercise 4

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set(rc = {'figure.figsize':(15,8)})
```

a) Data analysis

To analyse the data we first will load the data as a pandas DataFrame

In [2]:

```
weekly_sales = pd.read_csv("Task_4_weekly_sales_TurboPlasmaXPL.csv", names=["sales"])
weekly_sales.insert(loc=0, column='week', value=range(1, len(weekly_sales)+1))
weekly_sales.head()
```

Out[2]:

	week	sales
0	1	43
1	2	57
2	3	10
3	4	64
4	5	62

Plot of data

From the plot you can quickly see that the sales of the TurboPlasmaXML are very unstable in the beginning of the year and start to stabilize at around week 15. A cause for this can be that the market introduction of the TurboPlasmaXML wasn't very smooth and supply problems could have held back sales, especially in week two and ten.

To handle the data I would suggest cutting off the first 15 weeks since they a very high standard deviation and are not representative for the later weeks in the dataset.
Also, one could argue that the sales will contiune on the stabilized level and most likely will not drop to the levels seen in the first weeks.

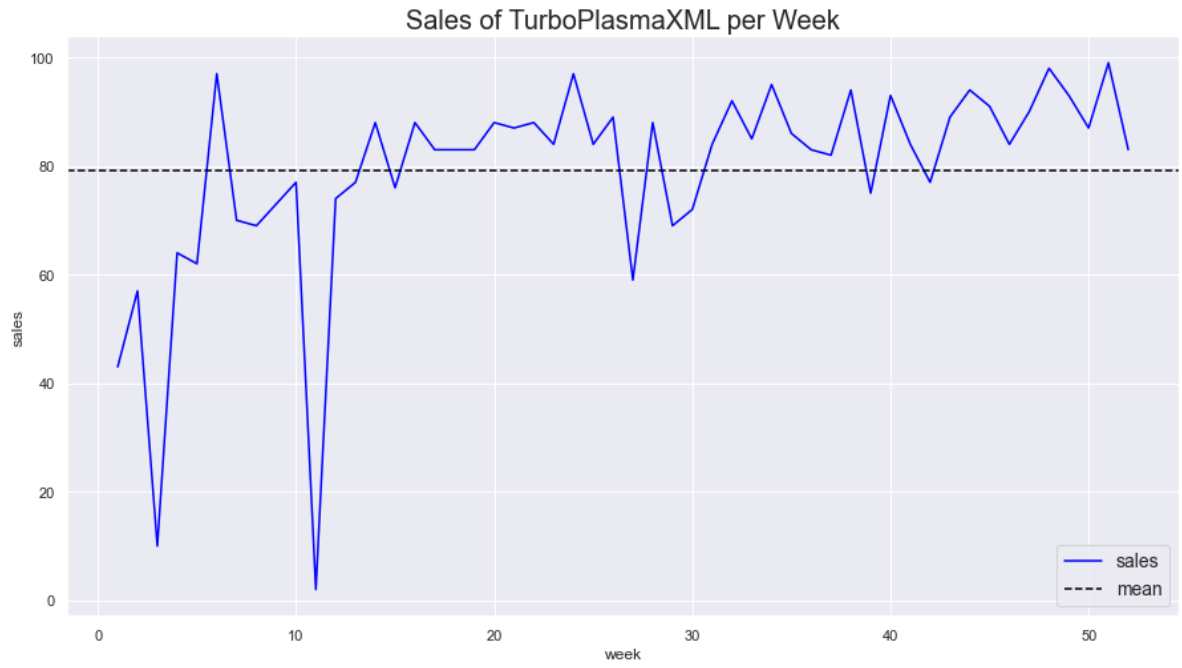
In addition to that the data becomes a lot easier to handle since there is no trend (and no seasonality) in the sales data anymore, whereas there was a strong trend in the first 14 weeks.

In [3]:

```
graph = sns.lineplot(data=weekly_sales, x="week", y="sales", color = "blue", label="sa
graph.axhline(weekly_sales.sales.mean(), color = "black", label="mean", linestyle="--"
graph.legend(loc=4, prop={'size': 14})
graph.set_title("Sales of TurboPlasmaXML per Week", size=20)
```

Out[3]:

```
Text(0.5, 1.0, 'Sales of TurboPlasmaXML per Week')
```



Mean and Standard Deviation

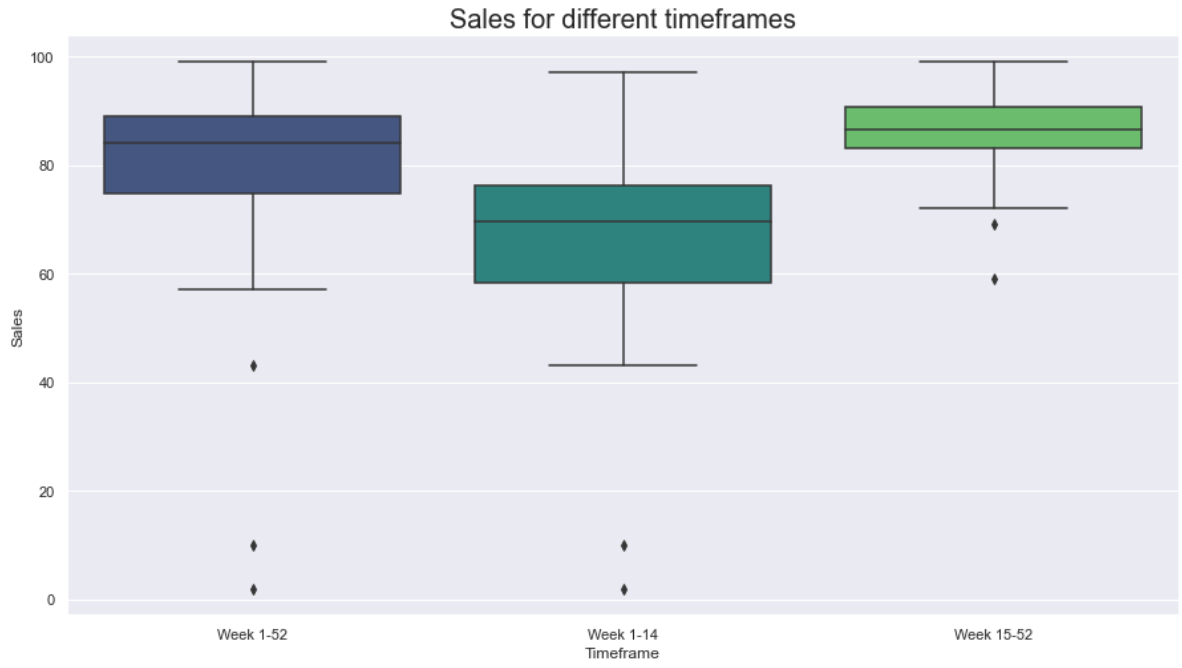
The Standard deviation in the first 14 weeks is more than double the standard deviation in the later weeks (week 15 and after). Also in the boxplot one can see that there are from week 15 and onwards the few outliers that exist are closer to the mean than for week 1 to 14 or the whole dataset.

In [4]:

```
weekly_sales_boxplot = pd.concat([
    pd.DataFrame({"Sales":weekly_sales.sales, "Timeframe":"Week 1-52"}),
    pd.DataFrame({"Sales":weekly_sales.sales[:14], "Timeframe":"Week 1-14"}),
    pd.DataFrame({"Sales":weekly_sales.sales[14:], "Timeframe":"Week 15-52"})
    ])

graph = sns.boxplot(x="Timeframe", y="Sales", data=weekly_sales_boxplot, palette="viri
graph.set_title("Sales for different timeframes", size=20)
```

Out[4]:

```
Text(0.5, 1.0, 'Sales for different timeframes')
```



In [5]:

```
print(f"The average weekly sales are: {weekly_sales.sales.mean()} with a standard devi
print(f"For weeks [1,14] the average weekly sales are: {weekly_sales.sales[:14].mean()
print(f"For weeks [15,52] the average weekly sales are: {weekly_sales.sales[14:].mean(
```

```
The average weekly sales are: 79.21153846153847 with a standard deviation of
18.672971489652994
For weeks [1,14] the average weekly sales are: 61.642857142857146 with a sta
ndard deviation of 26.920415881051895
For weeks [15,52] the average weekly sales are: 85.6842105263158 with a stan
dard deviation of 8.134362425082182
```

b) Forecasting

As argued in a) we will use the data from week 15 and onwards to create and test our forecasts. We will look at moving average and at exponential smoothing forecasts and compare them with different error measures.

In [6]:

```
# Ignore first cut sales figures
cut = 14

weekly_sales_cut_avg = weekly_sales[cut:]
weekly_sales_cut_exp = weekly_sales[cut:]
weekly_sales_cut_holt = weekly_sales[cut:]
weekly_sales_cut = weekly_sales[cut:]
weekly_sales_cut.head(10)
```

Out[6]:

	week	sales
14	15	76
15	16	88
16	17	83
17	18	83
18	19	83
19	20	88
20	21	87
21	22	88
22	23	84
23	24	97

Moving Average

For the moving average forecasts we will compare the forecasts with different parameters. We will use n = 2 to n = 10.

In [7]:

```
# Range for n for Moving average Forecasts
range_n = range(2,11)

def add_rolling_average_forecast(df, n):
    fc = df.sales.rolling(n).mean().shift(periods=1)
    df.insert(loc=len(df.columns), column=f"moving_average_n={n}", value=fc)

for n in range_n:
    add_rolling_average_forecast(weekly_sales_cut_avg,n)

weekly_sales_cut_avg.head(20)
```

Out[7]:

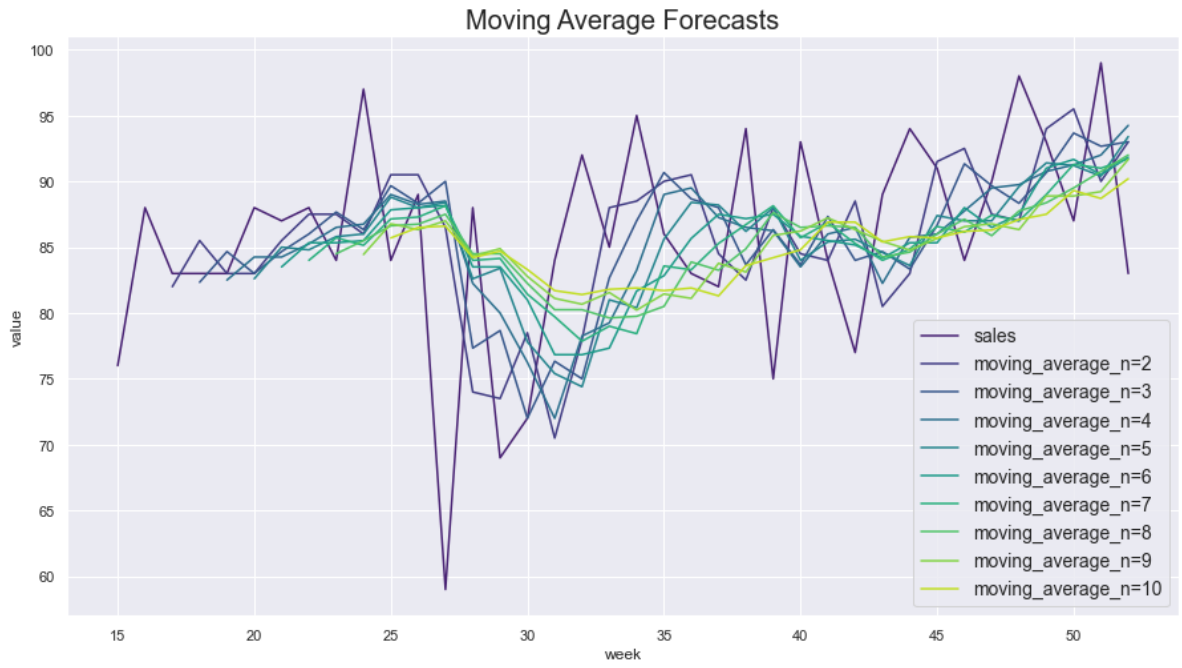
	week	sales	moving_average_n=2	moving_average_n=3	moving_average_n=4	moving_aver
14	15	76	NaN	NaN	NaN	
15	16	88	NaN	NaN	NaN	
16	17	83	82.0	NaN	NaN	
17	18	83	85.5	82.333333	NaN	
18	19	83	83.0	84.666667	82.50	
19	20	88	83.0	83.000000	84.25	
20	21	87	85.5	84.666667	84.25	
21	22	88	87.5	86.000000	85.25	
22	23	84	87.5	87.666667	86.50	
23	24	97	86.0	86.333333	86.75	
24	25	84	90.5	89.666667	89.00	
25	26	89	90.5	88.333333	88.25	
26	27	59	86.5	90.000000	88.50	
27	28	88	74.0	77.333333	82.25	
28	29	69	73.5	78.666667	80.00	
29	30	72	78.5	72.000000	76.25	
30	31	84	70.5	76.333333	72.00	
31	32	92	78.0	75.000000	78.25	
32	33	85	88.0	82.666667	79.25	
33	34	95	88.5	87.000000	83.25	

In [8]:

```
# Plot of Moving Average Forecasts
graph = sns.lineplot(data=pd.melt(weekly_sales_cut_avg, ['week']), x="week", y="value",
graph.set_title("Moving Average Forecasts", size=20)
graph.legend(loc=4, prop={'size': 14})
```

Out[8]:

```
<matplotlib.legend.Legend at 0x1e974a48040>
```



Simple Exponential Smoothing

For this forecasting method we will compare different smoothing parameters alpha. We will run nine forecasts from alpha = 0.1 to 0.9.

In [9]:

```
# Range for alpha for exponential smoothing
range_alpha = np.arange(0.1, 1, 0.1)

def add_exponential_smoothing_forecast(df, a):
    fc = df.sales.ewm(alpha=a, adjust=False).mean().shift(periods=1, fill_value=df.sal
    df.insert(loc=len(df.columns), column=f"exp_smoothing_α={a.round(1)}", value=fc)

for a in range_alpha:
    add_exponential_smoothing_forecast(df=weekly_sales_cut_exp, a=a)

weekly_sales_cut_exp.head(20)
```

Out[9]:

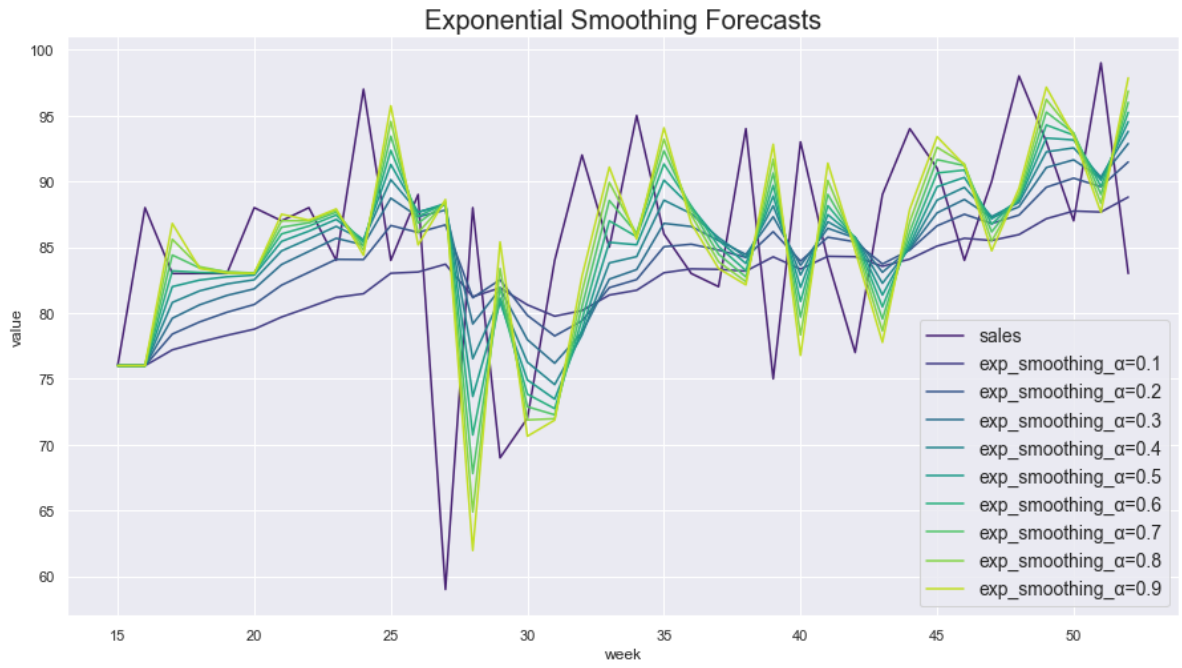
	week	sales	exp_smoothing_α=0.1	exp_smoothing_α=0.2	exp_smoothing_α=0.3	exp_smoot
14	15	76	76.000000	76.000000	76.000000	
15	16	88	76.000000	76.000000	76.000000	
16	17	83	77.200000	78.400000	79.600000	
17	18	83	77.780000	79.320000	80.620000	
18	19	83	78.302000	80.056000	81.334000	
19	20	88	78.771800	80.644800	81.833800	
20	21	87	79.694620	82.115840	83.683660	
21	22	88	80.425158	83.092672	84.678562	
22	23	84	81.182642	84.074138	85.674993	
23	24	97	81.464378	84.059310	85.172495	
24	25	84	83.017940	86.647448	88.720747	
25	26	89	83.116146	86.117958	87.304523	
26	27	59	83.704532	86.694367	87.813166	
27	28	88	81.234078	81.155493	79.169216	
28	29	69	81.910671	82.524395	81.818451	
29	30	72	80.619603	79.819516	77.972916	
30	31	84	79.757643	78.255613	76.181041	
31	32	92	80.181879	79.404490	78.526729	
32	33	85	81.363691	81.923592	82.568710	
33	34	95	81.727322	82.538874	83.298097	

In [10]:

```
# Graph for exponential smoothing
graph = sns.lineplot(data=pd.melt(weekly_sales_cut_exp, ['week']), x="week", y="value",
graph.set_title("Exponential Smoothing Forecasts", size=20)
graph.legend(loc=4, prop={'size': 14})
```

Out[10]:

```
<matplotlib.legend.Legend at 0x1e974a1ae20>
```



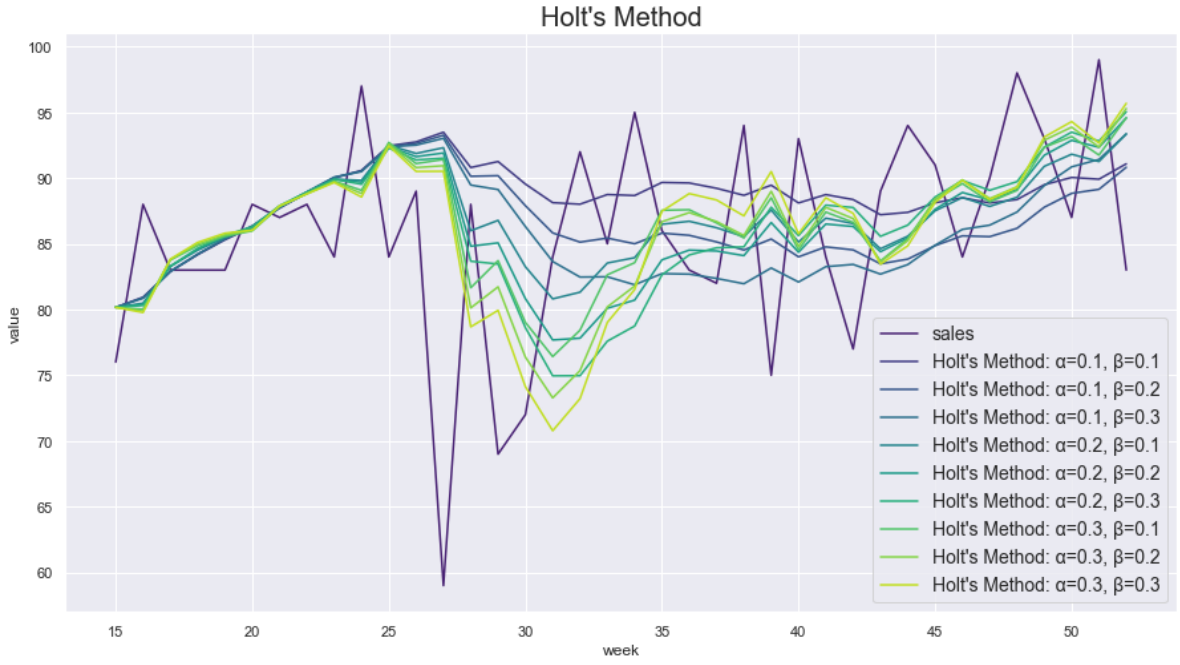
Holt's Method (Double Exponential smoothing)

In [11]:

```
from statsmodels.tsa.holtwinters import Holt

def add_holt(df, alpha, beta):
    model = Holt(df.sales, initialization_method="estimated")
    fit = model.fit(smoothing_level=alpha, smoothing_trend=beta, optimized=False)
    fc = fit.fittedvalues
    df.insert(loc=len(df.columns), column=f"Holt's Method: α={alpha.round(1)}, β={beta


for a in np.array([0.1, 0.2, 0.3]):
    for b in np.array([0.1, 0.2, 0.3]):
        add_holt(weekly_sales_cut_holt, alpha=a, beta=b)

graph = sns.lineplot(data=pd.melt(weekly_sales_cut_holt, ['week']), x="week", y="value
graph.set_title("Holt's Method", size=20)
graph.legend(loc=4, prop={'size': 14})
```

Out[11]:

```
<matplotlib.legend.Legend at 0x1e974bc0040>
```



Error Measures

To compare the different forecasting methods we will take a loot at three different error measures, namely MAD, RMSE and MAPE.

In [12]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_per
```

In [13]:

```
weekly_sales_cut_forecasts = pd.merge(left=weekly_sales_cut_avg, right=weekly_sales_cut
weekly_sales_cut_forecasts = pd.merge(left=weekly_sales_cut_forecasts, right=weekly_sa

methods = list(weekly_sales_cut_forecasts.columns[1:])
weekly_sales_cut_forecasts = np.array(weekly_sales_cut_forecasts).transpose()
sales_cut = weekly_sales_cut_forecasts[0]
sales_cut = np.tile(sales_cut, (len(methods),1))
weekly_sales_cut_forecasts = weekly_sales_cut_forecasts[1:]
```

In [14]:

```
MAE = list()
RMSE  = list()
MAPE = list()

for sales, forecast in zip(sales_cut, weekly_sales_cut_forecasts):
    sales = sales[~np.isnan(forecast)]
    forecast = forecast[~np.isnan(forecast)]
    MAE.append(mean_absolute_error(y_true=sales, y_pred=forecast))
    RMSE.append(mean_squared_error(y_true=sales, y_pred=forecast, squared=False))
    MAPE.append(mean_absolute_percentage_error(y_true=sales, y_pred=forecast))

error_measures = pd.DataFrame([MAE,RMSE, MAPE],columns=methods, index=["MAE", "RMSE",
try:
    error_measures.to_excel("ErrorMeasure.xlsx")
except:
    pass
error_measures
```

Out[14]:

	moving_average_n=2	moving_average_n=3	moving_average_n=4	moving_average_n=5	r
MAE	6.986111	6.809524	6.875000	7.224242	
RMSE	8.957074	8.854198	8.708832	9.252289	
MAPE	0.084760	0.083165	0.084438	0.088669	

3 rows × 27 columns

After we calculated the error measures for every method we have to find the best method to use for our future forecasts.

In [15]:

```
print("Best Forecast per error measure:")
print(error_measures.idxmin(axis=1))
```

```
Best Forecast per error measure:
MAE     Holt's Method: α=0.2, β=0.1
RMSE            exp_smoothing_α=0.2
MAPE    Holt's Method: α=0.2, β=0.1
dtype: object
```

I would recommend to use a exponential smoothing forecast with an alpha of 0.2, since it has the best error measures in 2 categories and is very close to exp_smoothing_α=0.3 for the MAE, which is best for this error measure.

c) Evaluation of Forecast Model

To evaluate our forecast model from b) we will test it on the new data and evaluate it.

In [16]:

```
weekly_sales_contd = pd.read_csv("Task_4_weekly_sales_TurboPlasmaXPL_cont'd.csv", name
weekly_sales_contd.insert(loc=0, column='week', value=range(53, len(weekly_sales_contd
weekly_sales_contd.head()
```

Out[16]:

	week	sales
0	53	89
1	54	79
2	55	65
3	56	68
4	57	70

In [17]:

```
weekly_sales_cut_contd = pd.concat([weekly_sales_cut, weekly_sales_contd], axis=0).res
weekly_sales_cut_contd.tail()
```

Out[17]:

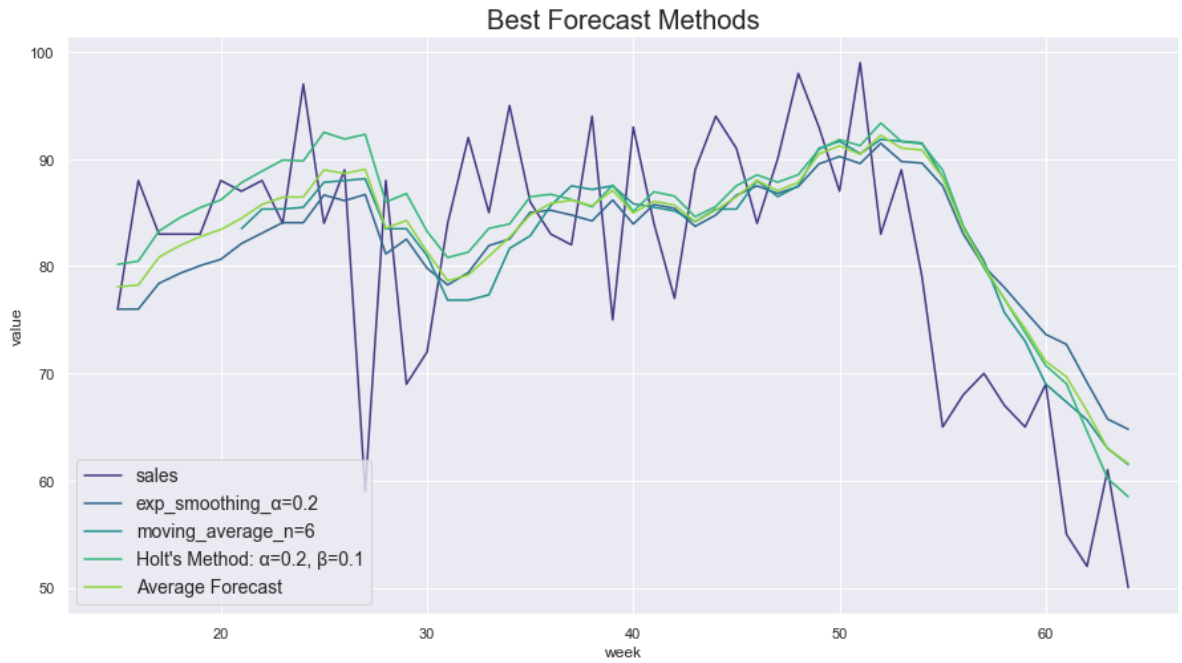
	week	sales
45	60	69
46	61	55
47	62	52
48	63	61
49	64	50

In [18]:

```
# adding the 3 best Forecasts
add_exponential_smoothing_forecast(df=weekly_sales_cut_contd, a=np.float64(0.2))
add_rolling_average_forecast(df=weekly_sales_cut_contd, n=6)
add_holt(df=weekly_sales_cut_contd, alpha=np.float64(0.2), beta=np.float64(0.1))

# adding a average Forecast
weekly_sales_cut_contd["Average Forecast"] = weekly_sales_cut_contd.drop(["week", "sale


graph = sns.lineplot(data=pd.melt(weekly_sales_cut_contd, ['week']), x="week", y="value
graph.set_title("Best Forecast Methods", size=20)
graph.legend(loc=3, prop={'size': 14})
```

Out[18]:

```
<matplotlib.legend.Legend at 0x1e97501f3d0>
```



In [19]:

```
1 weekly_sales_cut_contd.tail(12)
```

Out[19]:

	week	sales	exp_smoothing_α=0.2	moving_average_n=6	Holt's Method: α=0.2, β=0.1	Average Forecast
38	53	89	89.779314	91.666667	91.642082	91.029354
39	54	79	89.623451	91.500000	91.412332	90.845261
40	55	65	87.498761	88.333333	88.980285	88.270793
41	56	68	82.999009	83.666667	83.755042	83.473573
42	57	70	79.999207	80.500000	79.859747	80.119651
43	58	67	77.999366	75.666667	76.946316	76.870783
44	59	65	75.799492	73.000000	73.816645	74.205379
45	60	69	73.639594	69.000000	70.736575	71.125390
46	61	55	72.711675	67.333333	69.037787	69.694265
47	62	52	69.169340	65.666667	64.598002	66.478003
48	63	61	65.735472	63.000000	60.194213	62.976562
49	64	50	64.788378	61.500000	58.487298	61.591892

Error Measures for new data only

In [20]:

```
forecasts = weekly_sales_cut_contd.tail(12).drop("week", axis=1)
methods = list(forecasts.columns[1:])
forecasts = np.array(forecasts).transpose()
sales = weekly_sales_cut_contd.tail(12).sales
sales = np.tile(sales, (len(methods),1))
forecasts = forecasts[1:]

# TODO add bias and tracking signal (Slide 18)
MAE = list()
RMSE  = list()
MAPE = list()
BIAS = list()
Tracking_Signal = list()

for sales, forecast in zip(sales, forecasts):
    sales = sales[~np.isnan(forecast)]
    forecast = forecast[~np.isnan(forecast)]
    MAE.append(mean_absolute_error(y_true=sales, y_pred=forecast))
    RMSE.append(mean_squared_error(y_true=sales, y_pred=forecast, squared=False))
    MAPE.append(mean_absolute_percentage_error(y_true=sales, y_pred=forecast))
    BIAS.append(sum(forecast-sales))
    Tracking_Signal.append(sum(forecast-sales)/mean_absolute_error(y_true=sales, y_pre


error_measures = pd.DataFrame([MAE, RMSE, MAPE, BIAS, Tracking_Signal],columns=methods
                              index=["MAE", "RMSE", "MAPE", "Bias", "Tracking Signal"]
try:
    error_measures.to_excel("ErrorMeasuresForecast.xlsx")
except:
    pass
error_measures
```

Out[20]:

	exp_smoothing_α=0.2	moving_average_n=6	Holt's Method: α=0.2, β=0.1	Average Forecast
MAE	11.645255	10.069444	10.089825	10.556742
RMSE	13.081046	11.824234	11.866154	12.173395
MAPE	0.189662	0.160820	0.159833	0.169371
Bias	139.743059	120.833333	119.466324	126.680905
Tracking Signal	12.000000	12.000000	11.840277	12.000000

In [21]:

```
print("Best Forecast per error measure (only new data considered):")
print(error_measures.idxmin(axis=1))
```

```
Best Forecast per error measure (only new data considered):
MAE                        moving_average_n=6
RMSE                       moving_average_n=6
MAPE               Holt's Method: α=0.2, β=0.1
Bias               Holt's Method: α=0.2, β=0.1
Tracking Signal    Holt's Method: α=0.2, β=0.1
dtype: object
```