ST3233: Year 2020-2021

Final Project

Group Number: 12

Group Member 1: Nishanth Elango, A0184373Y

Group Member 2: Divakaran Haritha, A0187915N

Introduction

The dataset used in this project will be the weather statistics of Austin, Texas in USA. Taken from: https://www.kaggle.com/grubenm/austin-weather. The goal of this project is to predict the **average** temperature 2 days in advance(forecasting) by using linear regression. We will also be doing STL decomposition and trend analysis. Note that the temperature is in Farenheit. This project includes code references from chapter 2 and chapter 6 code snippets provided by Professor Thiery, Alexandre Hoang

In [273]:

```
%matplotlib inline
import pylab as plt
import numpy as np
import pandas as pd
import datetime
from statsmodels.tsa.seasonal import STL
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics import tsaplots
from statsmodels.tsa.api import SimpleExpSmoothing
import statsmodels
```

In [274]:

```
data = pd.read_csv("austin_weather.csv")
```

In [275]:

```
data.drop(data.head(1).index, inplace=True)
data.drop(['PrecipitationSumInches', 'Events'], axis=1, inplace=True)
data.dropna(inplace=True)
data.head()
```

Out[275]:

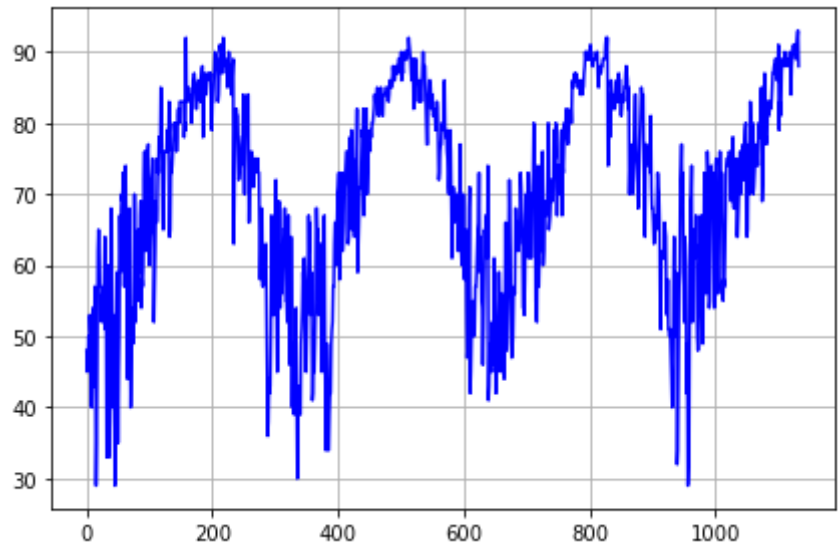
	Date	TempHighF	TempAvgF	TempLowF	DewPointHighF	DewPointAvgF	DewPointLowF	H
1	2013-12-22	56.0	48.0	39.0	43	36	28	
2	2013-12-23	58.0	45.0	32.0	31	27	23	
3	2013-12-24	61.0	46.0	31.0	36	28	21	
4	2013-12-25	58.0	50.0	41.0	44	40	36	
5	2013-12-26	57.0	48.0	39.0	39	36	33	

General Trend

We plot the graph of average temperature of the day vs number of datapoints to see the shape of the trend below.

In [276]:

```
plt.plot(data.values[:,2], "b-")
plt.grid(True)
plt.tight_layout()
```



STL Decomposition

In [277]:

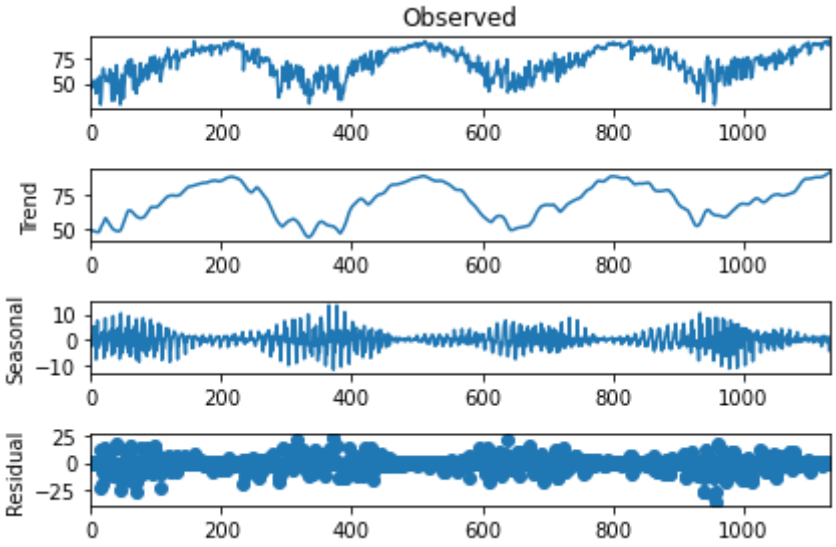
```
avg_temp = data["TempAvgF"][:].values
decomp = STL(avg_temp, period=12, robust=True ).fit()
```

In [278]:

```
S,T,E = decomp.seasonal, decomp.trend, decomp.resid
```

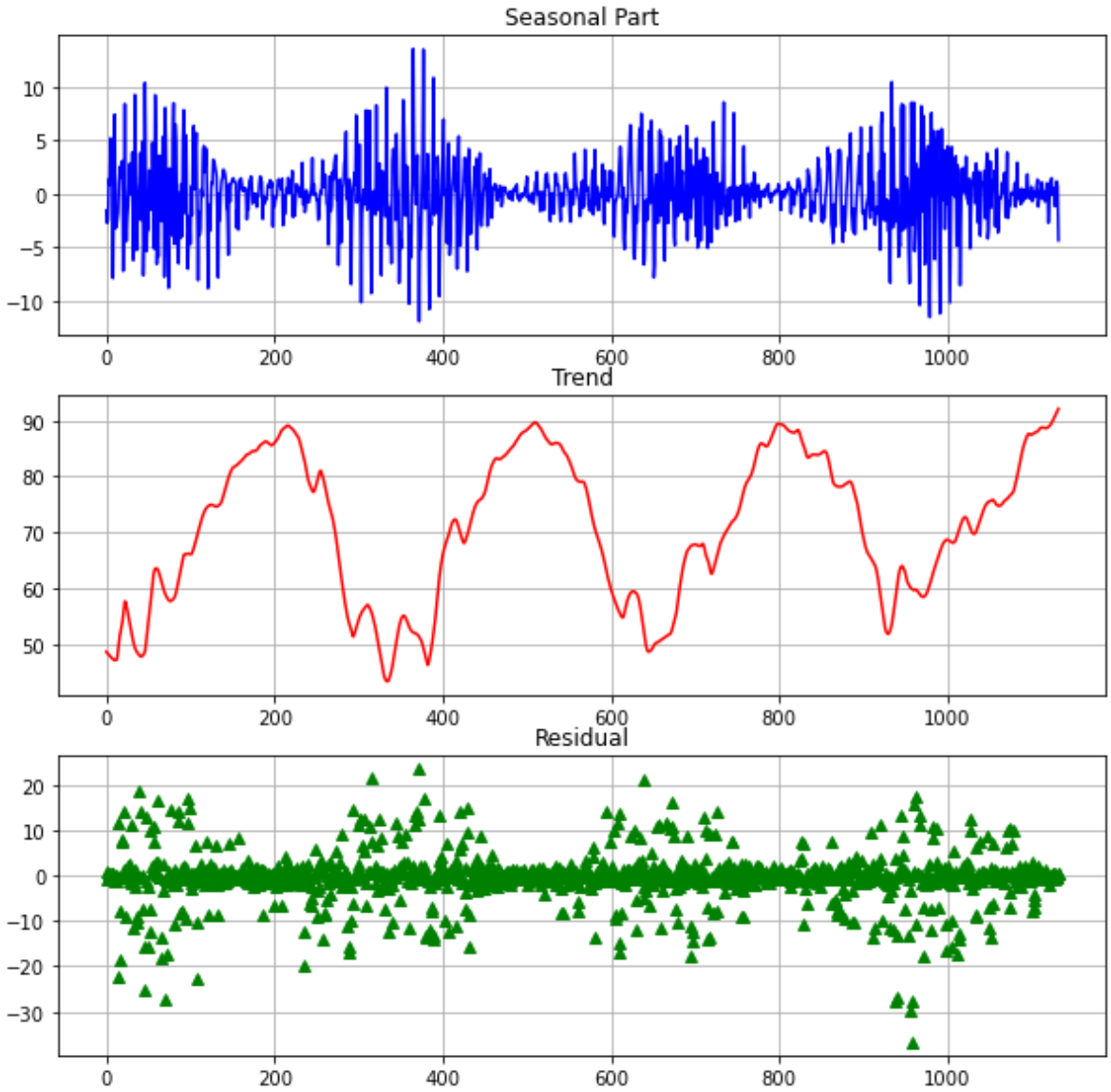
In [279]:

```
decomp.plot()
plt.show()
```



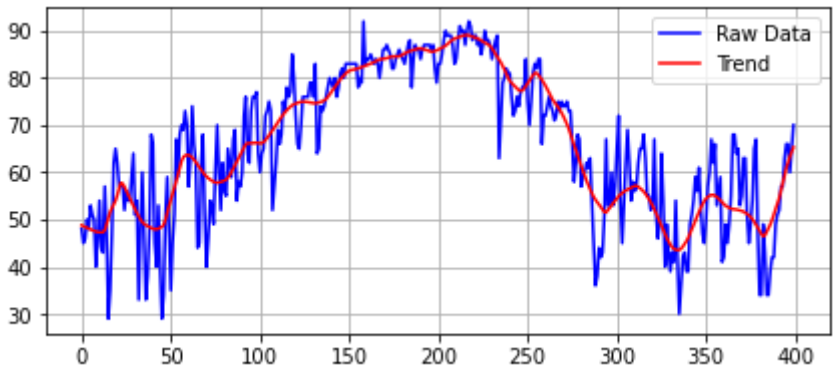
In [280]:

```
plt.figure(figsize=(10,10))
plt.subplot(3, 1, 1)
plt.plot(S, "b-")
plt.title("Seasonal Part")
plt.grid(True)

plt.subplot(3,1,2)
plt.plot(T, "r-")
plt.title("Trend")
plt.grid(True)


plt.subplot(3,1,3)
plt.plot(E, "g^")
plt.title("Residual")
plt.grid(True)
```



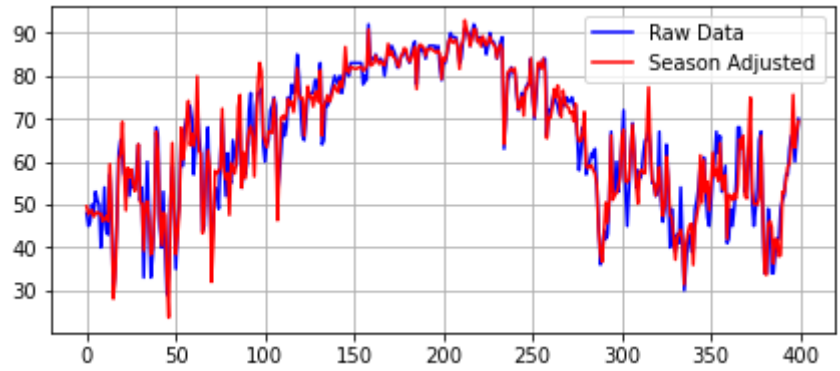
In [281]:

```
# let us superpose the trend estimation with the data
#let us plot only the first 400 values
T_max = 400
plt.figure(figsize=(7,3))
plt.plot(avg_temp[:T_max], "b-" , label="Raw Data")
plt.plot(T[:T_max], "r-", label="Trend")
plt.legend()
plt.grid(True)
```



In [282]:

```
plt.figure(figsize=(7,3))
plt.plot(avg_temp[:T_max], "b-" , label="Raw Data")
plt.plot((T+E)[:T_max], "r-", label="Season Adjusted")
plt.legend()
plt.grid(True)
```



Moving Average

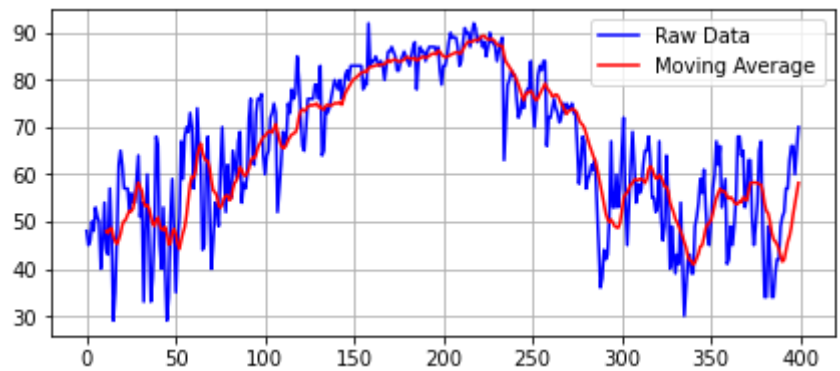
y = tsa.filters.filtertools.convolution_filter(x, filt, nsides = 1) is a built in function that produces a times series Y where $y[n] = filt[0] * x[n] + \ldots + filt[nfilt - 1] * x[n - nfilt + 1]$

In [283]:

```
avg_temp_MA = statsmodels.tsa.filters.filtertools.convolution_filter(avg_temp, filt=np.ones
```

In [284]:

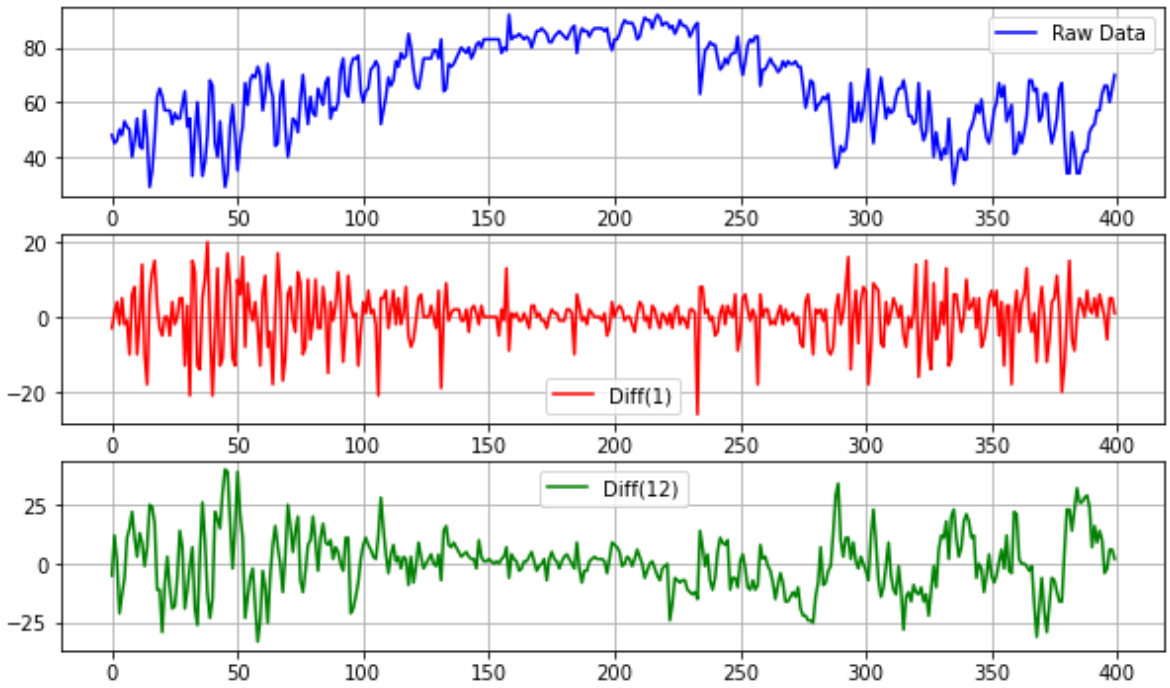
```
plt.figure(figsize=(7,3))
plt.plot(avg_temp[:T_max], "b-" , label="Raw Data")
plt.plot(avg_temp_MA[:T_max], "r-", label="Moving Average")
plt.legend()
plt.grid(True)
plt.savefig("trend_MA.png", dpi=200)
```



Getting rid of a trend: Differentiation

In [285]:

```
plt.figure(figsize=(10,6))
#Let us plot only the T_max first values
data_diff_1 = avg_temp[1:] - avg_temp[:-1]
data_diff_12 = avg_temp[12:] - avg_temp[:-12]

plt.subplot(3,1,1)
plt.plot(avg_temp[:T_max], "b-", label="Raw Data" )
plt.legend()
plt.grid(True)

plt.subplot(3,1,2)
plt.plot(data_diff_1[:T_max] , "r-", label="Diff(1)" )
plt.legend()
plt.grid(True)

plt.subplot(3,1,3)
plt.plot(data_diff_12[:T_max] , "g-", label="Diff(12)" )
plt.legend()
plt.grid(True)
```



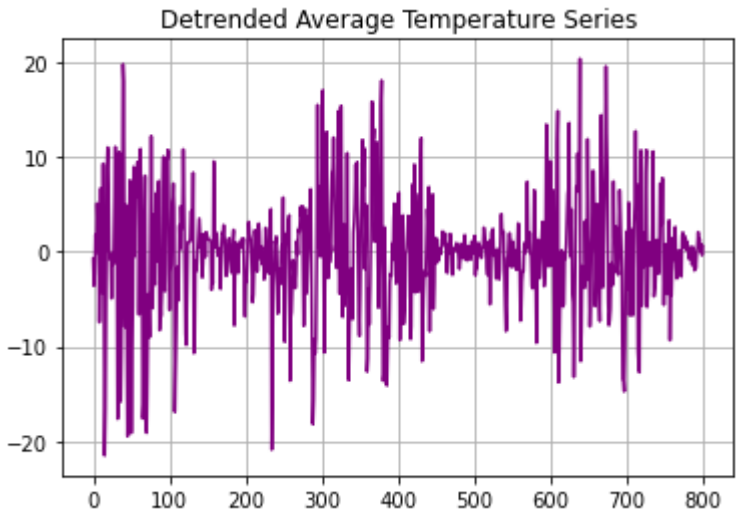
Visualizing a seasonal pattern

In [286]:

```
#Removing trend from the raw data
data_detrend = avg_temp - T
T_max = 800
plt.plot(data_detrend[:T_max], "purple")
plt.grid(True)
plt.title("Detrended Average Temperature Series")
```

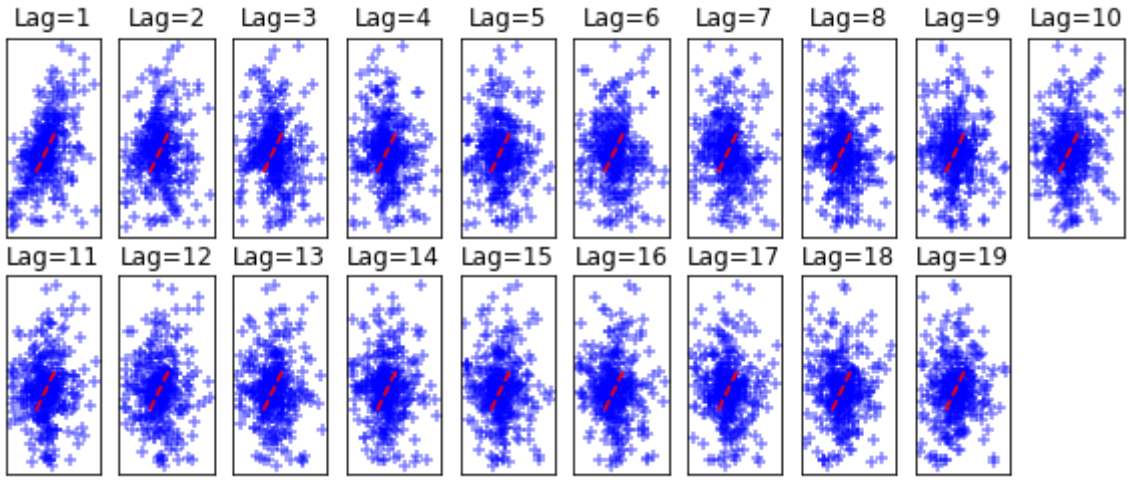
Out[286]:

```
Text(0.5, 1.0, 'Detrended Average Temperature Series')
```



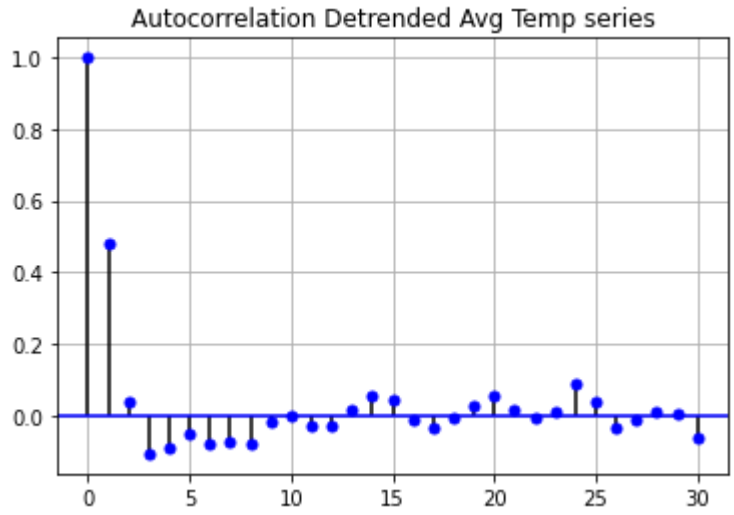
In [287]:

```
#Let us construct scatter plots for lag in between lag=1 and lag=12
plt.figure(figsize=(10,4))
T_max = len(avg_temp)
for lag in np.arange(1,20):
    plt.subplot(2,10,lag)
    plt.scatter(data_detrend[350:(T_max-365)], data_detrend[lag+350:(T_max-365+lag)], colo
    plt.plot([-4,4], [-4,4], "r--")  #add a red diagonal
    plt.title(f"Lag={lag}")
    plt.xticks([]) #hide x-label for clarity
    plt.yticks([]) #hide y-label for clarity
```



In [288]:

```
#ACF - Auto Correlation Function for the detrended time series
fig = tsaplots.plot_acf(data_detrend, lags=30, alpha=None, title="Autocorrelation Detrende
plt.grid(True)
```



Here we notice that there is a significant spike at a lag of 1 and much lower spikes for the subsequent lags. Thus, an AR(1) model would likely be feasible for this data set.

Naive model

This naive model will be a baseline to compare our other approaches to. This model will simply consist of predicting that the temparture in 2 days = the temperature now.

In [289]:

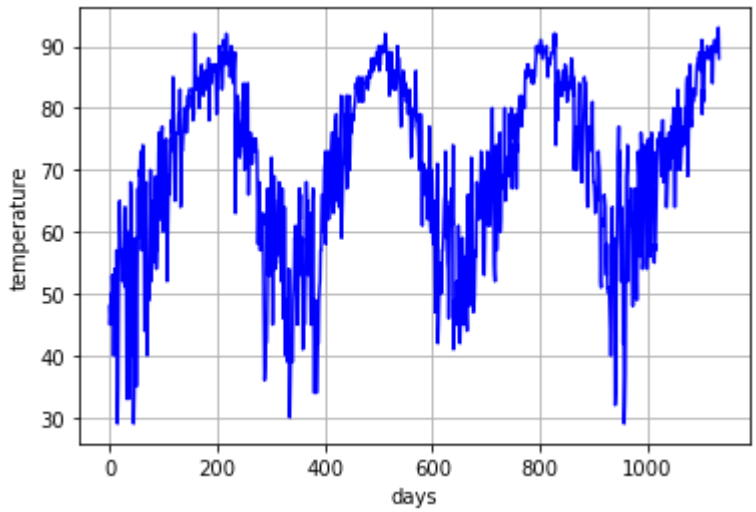
```
avg_temp = data["TempAvgF"][:].values
avg_temp
```

Out[289]:

```
array([48., 45., 46., ..., 92., 93., 88.])
```

In [290]:

```
plt.plot(avg_temp, "b-")
plt.xlabel("days")
plt.ylabel("temperature")
plt.grid(True)
```



In [291]:

```
#total number of days
n_tot = len(avg_temp)
n_test = int(n_tot * 0.3)
n_train = n_tot - n_test
print(f"n_tot, n_test, n_train = {n_tot, n_test, n_train}")
```

```
n_tot, n_test, n_train = (1133, 339, 794)
```

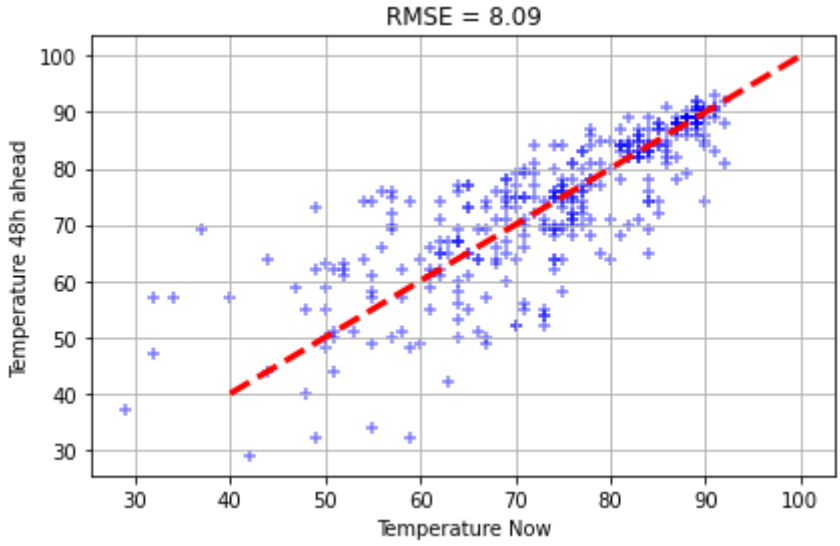
In [292]:

```
#subtracting values from 2 days ahead
errors = avg_temp[:-2] - avg_temp[2:]

#computing RMSE on the last 30% of data
RMSE_naive = np.sqrt(np.mean( errors[-n_test:]**2 ))
print(f"RMSE_naive={RMSE_naive: 2.2f}")
```

```
RMSE_naive= 8.09
```

In [293]:

```
plt.scatter(avg_temp[n_train:-2], avg_temp[n_train+2:], color="blue", marker="+", alpha=0.!
plt.plot([40,100], [40,100], "r--", lw=3)
plt.xlabel("Temperature Now")
plt.ylabel("Temperature 48h ahead")
plt.grid(True)
plt.title(f"RMSE = {RMSE_naive:2.2f}")

plt.tight_layout()
```



Observation:

The naive model has a root mean squared error of 8.09 which is extremely high. This shows us that the naive model of predicting temperature 2 days ahead is not a good one.

Using the N previous days Average Temperature data

We would like to fit a model of the type

$$T_{t+2} \approx \alpha_0 T_t + \alpha_1 T_{t-1} + \ldots + \alpha_0 T_{t-10}.$$

That is a simple **linear regression**. We will keep 70% of data to train our model, and save the last 30% to evaluate our model.

In [294]:

```
#n_train = 794, 70% for training, 30% for test
avg_temp_train = avg_temp[:n_train]
avg_temp_test = avg_temp[n_train:]
```

In [295]:

```
def linear_regression(N = 10):
    T = 400
    # append all the covariates in a list
    covariates = []
    for k in range(N+1):
        covariates.append(avg_temp[T-k:n_tot-2-k])
    # concatenate the list of vectors in a big matrix
    X_all = np.column_stack(covariates)
    # store the true values (i.e 2 days ahead)
    y_all= avg_temp[T+2:n_tot]

    #test/test set
    X_train = X_all[0:400]
    y_train = y_all[0:400]
    X_test = X_all[400:]
    y_test = y_all[400:]


    return X_train, y_train, X_test, y_test
```

In [296]:

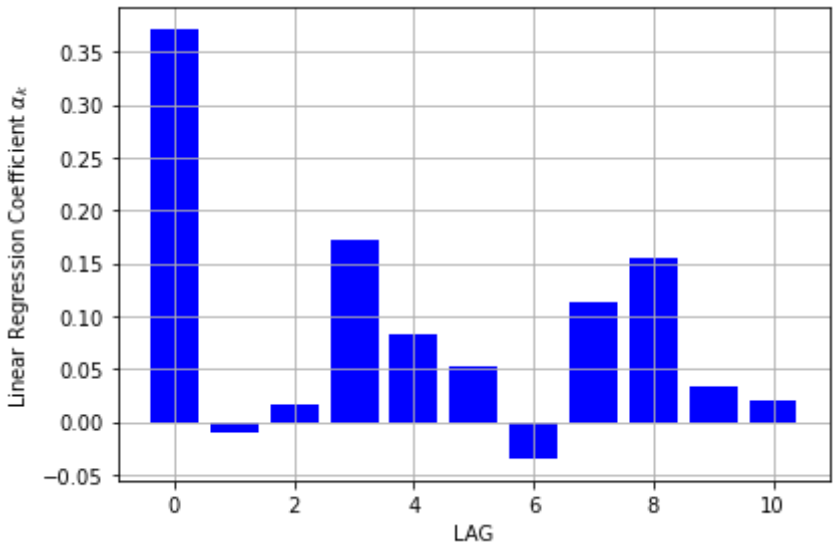
```
#fit a linear regression
from sklearn.linear_model import LinearRegression
```

In [297]:

```
N = 10
X_train, y_train, X_test, y_test = linear_regression(N)
lin_reg = LinearRegression().fit(X_train, y_train)
```

In [298]:

```
plt.bar(range(N+1), lin_reg.coef_, color="blue")
plt.grid(True)
plt.xlabel("LAG")
plt.ylabel(r"Linear Regression Coefficient $\alpha_k$")

plt.tight_layout()
```

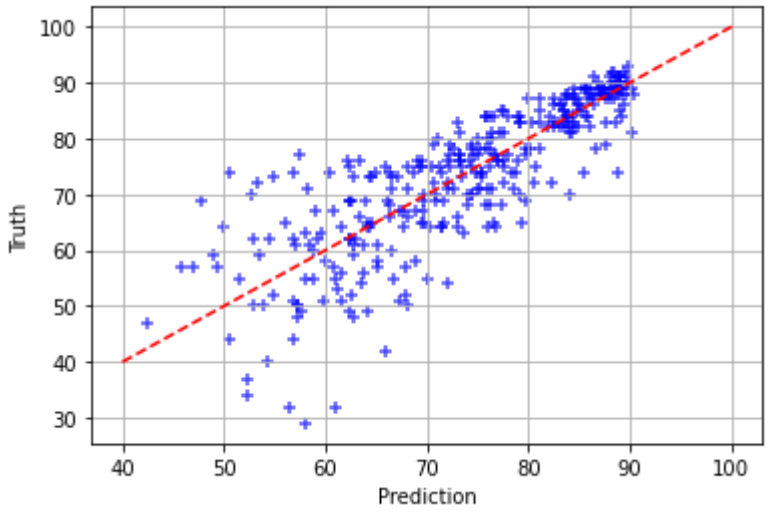


In [299]:

```
# make predictions
y_test_pred = lin_reg.predict(X_test)
```

In [300]:

```
plt.scatter(y_test_pred, y_test, marker="+", color="blue", alpha=0.7)
plt.plot([40,100], [40,100], "r--")
plt.xlabel("Prediction")
plt.ylabel("Truth")
plt.grid(True)
```



In [301]:

```
errors = y_test - y_test_pred
#Let us compute the RMSE on the last 30% of data
RMSE = np.sqrt(np.mean( errors**2 ))
print(f"RMSE_LINREG={RMSE: 2.2f}")
```

```
RMSE_LINREG= 7.31
```

In [302]:

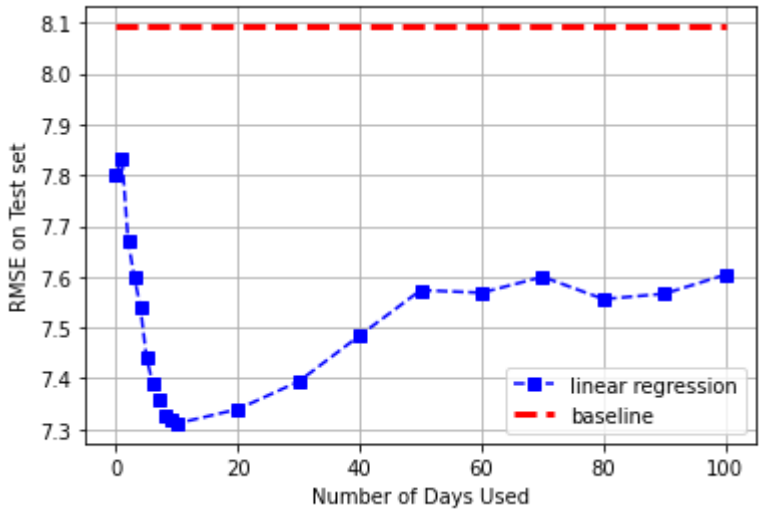
```
#checking for multiple days
days_list = [0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100]
RMSE_list = []
for N in days_list:
    X_train, y_train, X_test, y_test = linear_regression(N)
    lin_reg = LinearRegression().fit(X_train, y_train)
    y_test_pred = lin_reg.predict(X_test)
    errors = y_test - y_test_pred
    RMSE = np.sqrt(np.mean( errors**2 ))
    RMSE_list.append(RMSE)
```

In [303]:

```
plt.plot(days_list, RMSE_list, "b--s", label="linear regression")
plt.plot(days_list, [RMSE_naive for _ in days_list], "r--", lw=3, label="baseline")
plt.legend()

plt.grid(True)
plt.xlabel("Number of Days Used")
plt.ylabel("RMSE on Test set")
```

Out[303]:

```
Text(0, 0.5, 'RMSE on Test set')
```



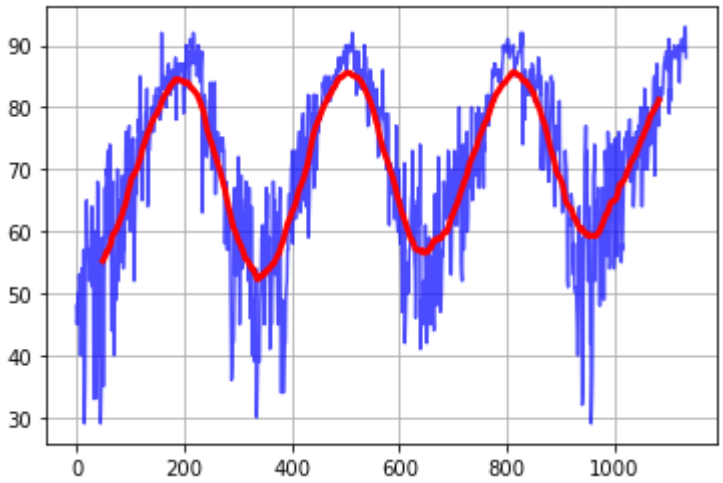
Using the average of previous year around the same month

In [304]:

```
#Two-sided moving average
smoothing_filter = np.ones(100)/100.
MA_temp = statsmodels.tsa.filters.filtertools.convolution_filter(avg_temp, filt=smoothing_
```

In [305]:

```
plt.plot(avg_temp, "b-", alpha=0.7)
plt.plot(MA_temp, "r-", lw=3)
plt.grid(True)
```



In [306]:

```
np.sqrt(np.mean((avg_temp[n_train:] - MA_temp[n_train-365:-365])**2 ) )
```

Out[306]:

13.836748226188343

In [307]:

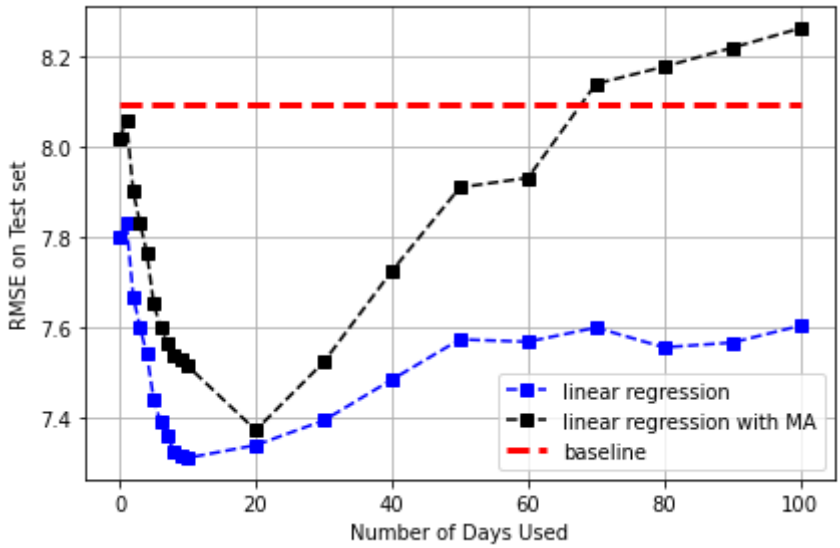
```
# now, let us do a linear regression, using only the average temperature data of the last N
# the average temp of last year

def linear_regression_with_MA(N = 10):
    T = 400

    # append all the covariates in a list
    covariates = []
    for k in range(N+1):
        covariates.append(avg_temp[T-k:n_tot-2-k])
    # also use the average the same day but previous year
    covariates.append(MA_temp[T-365:n_tot-2-365])
    # concatenate the list of vectors in a big matrix
    X_all = np.column_stack(covariates)
    # store the true values
    y_all= avg_temp[T+2:n_tot]

    #test/test set, select 20 to 420 as there are nan values in the first few X_all
    X_train = X_all[20:400]
    y_train = y_all[20:400]
    X_test = X_all[420:]
    y_test = y_all[420:]
    return X_train, y_train, X_test, y_test
```

In [308]:

```
#checking for multiple days
days_list = [0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80, 90, 100]
RMSE_ma_list = []
for N in days_list:
    X_train, y_train, X_test, y_test = linear_regression_with_MA(N)
    lin_reg = LinearRegression().fit(X_train, y_train)
    y_test_pred = lin_reg.predict(X_test)
    errors = y_test - y_test_pred

    RMSE_ma = np.sqrt(np.mean( errors**2 ))
    RMSE_ma_list.append(RMSE_ma)
```

In [309]:

```
plt.plot(days_list, RMSE_list, "b--s", label="linear regression")
plt.plot(days_list, RMSE_ma_list, "k--s", label="linear regression with MA")

plt.plot(days_list, [RMSE_naive for _ in days_list], "r--", lw=3, label="baseline")

plt.legend()

plt.grid(True)
plt.xlabel("Number of Days Used")
plt.ylabel("RMSE on Test set")

plt.tight_layout()
```

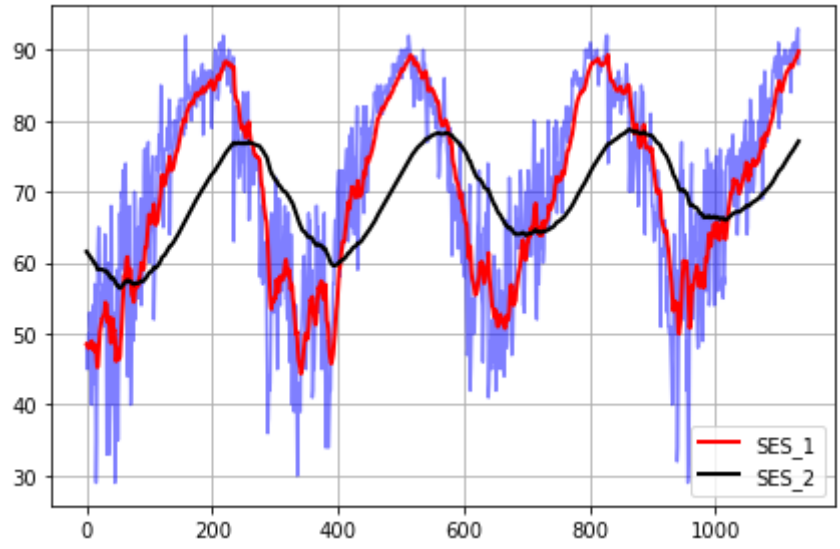


Exponential smoothing of Covariates

In [310]:

```
plt.plot(avg_temp, "b-", alpha=0.5)
SES_1 = SimpleExpSmoothing(avg_temp).fit(smoothing_level=0.1,optimized=True).fittedvalues
SES_2 = SimpleExpSmoothing(avg_temp).fit(smoothing_level=0.01,optimized=True).fittedvalues

plt.plot(SES_1, "r-", lw=2, label="SES_1")
plt.plot(SES_2, "k-", lw=2, label="SES_2")

plt.grid(True)
plt.legend()

plt.tight_layout()
```

```
C:\Users\harit\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:73
1: RuntimeWarning: invalid value encountered in greater_equal
  loc = initial_p >= ub
```



In this dataset, smoothing level of 0.1 seems to be a better fit than smoothing level of 0.01.

In [311]:

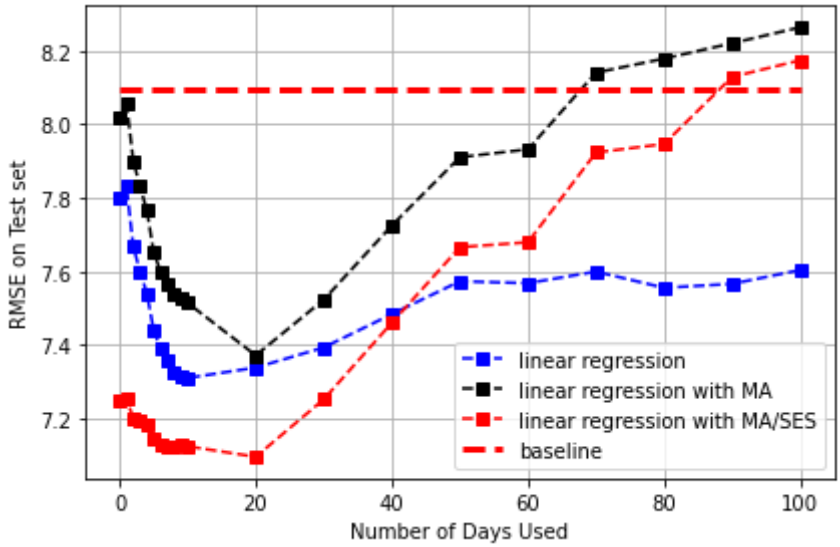
```
def linear_regression_with_MA_SES(N = 10):
    T = 400

    # append all the covariates in a list
    covariates = []
    for k in range(N+1):
        covariates.append(avg_temp[T-k:n_tot-2-k])
    # also use the average the same day but previous year
    covariates.append(MA_temp[T-365:n_tot-2-365])
    covariates.append(SES_1[T-365:n_tot-2-365])
    covariates.append(SES_2[T-365:n_tot-2-365])

    # concatenate the list of vectors in a big matrix
    X_all = np.column_stack(covariates)
    # store the true values
    y_all= avg_temp[T+2:n_tot]

    #test/test set, select 20 to 420 as there are NaN values in the first few X_all
    X_train = X_all[20:400]
    y_train = y_all[20:400]
    X_test = X_all[400:]
    y_test = y_all[400:]

    return X_train, y_train, X_test, y_test
```

In [312]:

```
#checking for multiple days
days_list = [0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80, 90, 100]
RMSE_ma_ses_list = []
for N in days_list:
    X_train, y_train, X_test, y_test = linear_regression_with_MA_SES(N)
    lin_reg = LinearRegression().fit(X_train, y_train)
    y_test_pred = lin_reg.predict(X_test)
    errors = y_test - y_test_pred
    RMSE_ma_ses = np.sqrt(np.mean( errors**2 ))
    RMSE_ma_ses_list.append(RMSE_ma_ses)
```

In [313]:

```
plt.plot(days_list, RMSE_list, "b--s", label="linear regression")
plt.plot(days_list, RMSE_ma_list, "k--s", label="linear regression with MA")
plt.plot(days_list, RMSE_ma_ses_list, "r--s", label="linear regression with MA/SES")


plt.plot(days_list, [RMSE_naive for _ in days_list], "r--", lw=3, label="baseline")

plt.legend()

plt.grid(True)
plt.xlabel("Number of Days Used")
plt.ylabel("RMSE on Test set")

plt.tight_layout()
```



Using average of previous year's other covariates

In [314]:

```
#let us do a two-sided moving average
smoothing_filter = np.ones(100)/100.

#taking covariates from DewPointHighF to WindGustMPH and storing it in float array
float_array = data.values[:, 4:]

row_mask = (float_array != '-').all(axis=1) #some rows contain hyphenated data removing the
float_array = float_array[row_mask,:]
float_array = float_array.astype(np.float) #converting strings to float as other covariates
print(float_array)
data.head()
```

```
[[43. 36. 28. ... 16.  6. 25.]
 [31. 27. 23. ...  8.  3. 12.]
 [36. 28. 21. ... 12.  4. 20.]
 ...
 [72. 64. 55. ... 12.  4. 17.]
 [70. 68. 63. ... 13.  4. 20.]
 [66. 61. 54. ... 12.  4. 20.]]
```

Out[314]:

	Date	TempHighF	TempAvgF	TempLowF	DewPointHighF	DewPointAvgF	DewPointLowF	H
1	2013-12-22	56.0	48.0	39.0	43	36	28	
2	2013-12-23	58.0	45.0	32.0	31	27	23	
3	2013-12-24	61.0	46.0	31.0	36	28	21	
4	2013-12-25	58.0	50.0	41.0	44	40	36	
5	2013-12-26	57.0	48.0	39.0	39	36	33	

In [315]:

```
# let us compute the MA of all the covariates
austin_data_MA = np.copy(float_array)
for k in range(float_array.shape[1]):
    austin_data_MA[:,k] = statsmodels.tsa.filters.filtertools.convolution_filter(float_arra
```

In [316]:

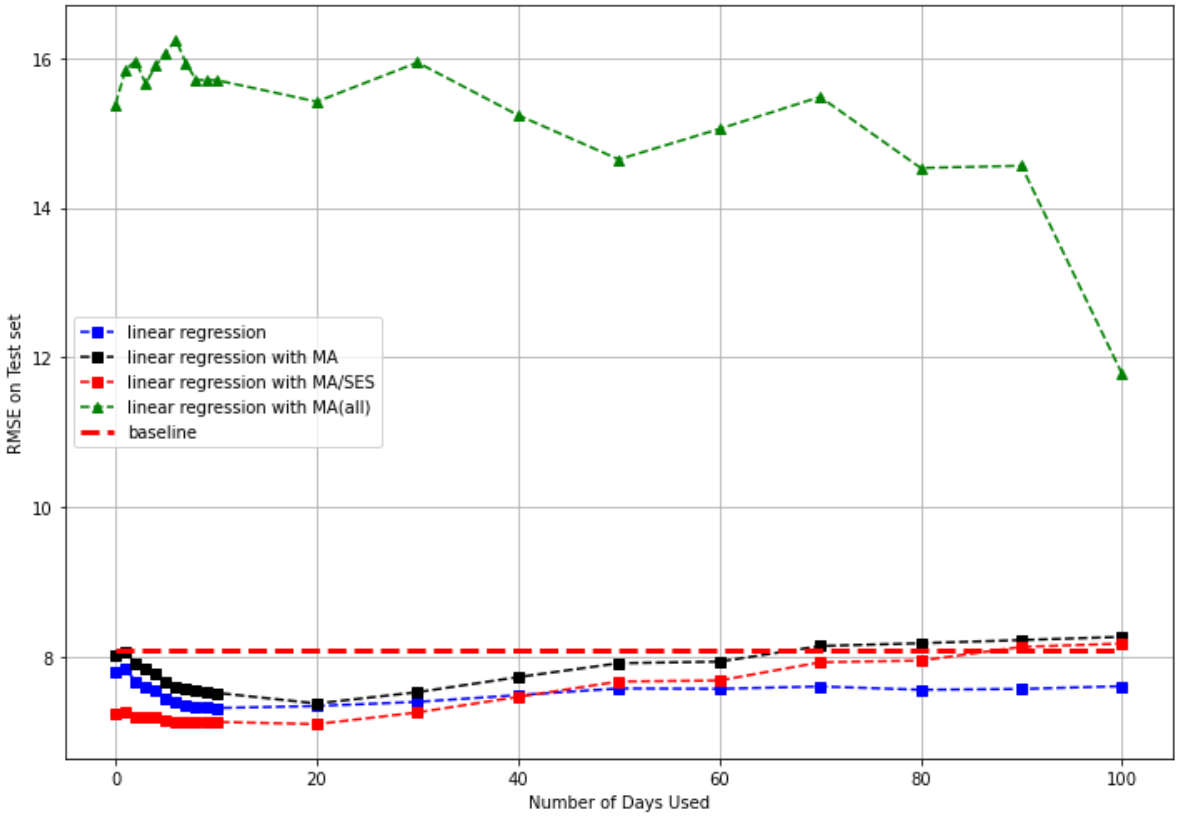
```
def linear_regression_with_MA_all(N = 10):
    T = 400

    # append all the covariates in a list
    covariates = []
    for k in range(N+1):
        covariates.append(avg_temp[T-k:n_tot-2-k])
    # also use the average the same day but previous year for all the covariates
    for k in range(austin_data_MA.shape[1]):
        covariates.append(austin_data_MA[T-365:n_tot-2-365,k])


    # concatenate the list of vectors in a big matrix
    X_all = np.column_stack(covariates)
    # store the true values
    y_all= avg_temp[T+2:n_tot]

    #test/test set
    X_train = X_all[20:400]
    y_train = y_all[20:400]
    X_test = X_all[400:]
    y_test = y_all[400:]

    return X_train, y_train, X_test, y_test
```

In [317]:

```
#checking for multiple days
days_list = [0,1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100]
RMSE_ma_all_list = []
for N in days_list:
    X_train, y_train, X_test, y_test = linear_regression_with_MA_all(N)
    lin_reg = LinearRegression().fit(X_train, y_train)
    y_test_pred = lin_reg.predict(X_test)
    errors = y_test - y_test_pred
    RMSE_ma_all = np.sqrt(np.mean( errors**2 ))
    RMSE_ma_all_list.append(RMSE_ma_all)
```

In [318]:

```
plt.figure(figsize=(10,7))
plt.plot(days_list, RMSE_list, "b--s", label="linear regression")
plt.plot(days_list, RMSE_ma_list, "k--s", label="linear regression with MA")
plt.plot(days_list, RMSE_ma_ses_list, "r--s", label="linear regression with MA/SES")
plt.plot(days_list, RMSE_ma_all_list, "g--^", label="linear regression with MA(all)")



plt.plot(days_list, [RMSE_naive for _ in days_list], "r--", lw=3, label="baseline")

plt.legend()

plt.grid(True)
plt.xlabel("Number of Days Used")
plt.ylabel("RMSE on Test set")

plt.tight_layout()
plt.savefig("jena_LR_final.pdf")
```



Conclusion: Out of all the models, we conclude that the method where we use exponential smoothing of covariate(avg temp) of the previous year aroud the same month, gives the most accurate prediction with the lowest mean squared error when the number of days used is less than 40.