In [59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [60]:
```
# read the main dataset about weather data from csv
df = pd.read_csv('MET Office Weather Data.csv',encoding='utf-8')

# filter the dataframe in the range of year 2000 to 2019
df = df[(df['year']>=2000) & (df['year']<=2019)]
df['year'] = df['year'].astype(int)
df['month'] = df['month'].astype(int)
df['dateInt']=df['year'].astype(str) + df['month'].astype(str) +'1'
df['date'] = pd.to_datetime(df['dateInt'], format='%Y%m%d')
df = df.drop(['dateInt'],axis=1)

df.head()
```

Out[60]:

	year	month	tmax	tmin	af	rain	sun	station	date
708	2000	1	7.8	3.6	3.0	54.4	51.9	aberporth	2000-01-01
709	2000	2	8.6	4.3	1.0	107.0	96.3	aberporth	2000-02-01
710	2000	3	9.3	4.7	0.0	28.8	138.5	aberporth	2000-03-01
711	2000	4	10.3	4.2	0.0	87.6	159.9	aberporth	2000-04-01
712	2000	5	14.4	8.3	0.0	58.4	230.9	aberporth	2000-05-01

In [61]:
```
# shows the general info of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8116 entries, 708 to 37042
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   year     8116 non-null   int64
 1   month    8116 non-null   int64
 2   tmax     7671 non-null   float64
 3   tmin     7690 non-null   float64
 4   af       7690 non-null   float64
 5   rain     7703 non-null   float64
 6   sun      5584 non-null   float64
 7   station  8116 non-null   object
 8   date     8116 non-null   datetime64[ns]
dtypes: datetime64[ns](1), float64(5), int64(2), object(1)
memory usage: 634.1+ KB
```

In [62]:
```
# check duplicated value, no duplicated values is found
df[df.duplicated()]
```

Out[62]:

	year	month	tmax	tmin	af	rain	sun	station	date

In [63]:
```
# check if null value exist, i.e. 2532 null values have been found
# in attribute 'sun'
df.isnull().sum()
```

Out[63]:
```
year          0
month         0
tmax        445
tmin        426
af          426
rain        413
sun        2532
station       0
date          0
dtype: int64
```

In [64]:
```
# fill the missing values in numerical variables
# with the mean value of that column
df['sun'] = df['sun'].fillna(df['sun'].mean())
df['tmax'] = df['tmax'].fillna(df['tmax'].mean())
df['tmin'] = df['tmin'].fillna(df['tmin'].mean())
df['af'] = df['af'].fillna(df['af'].mean())
df['rain'] = df['rain'].fillna(df['rain'].mean())
```

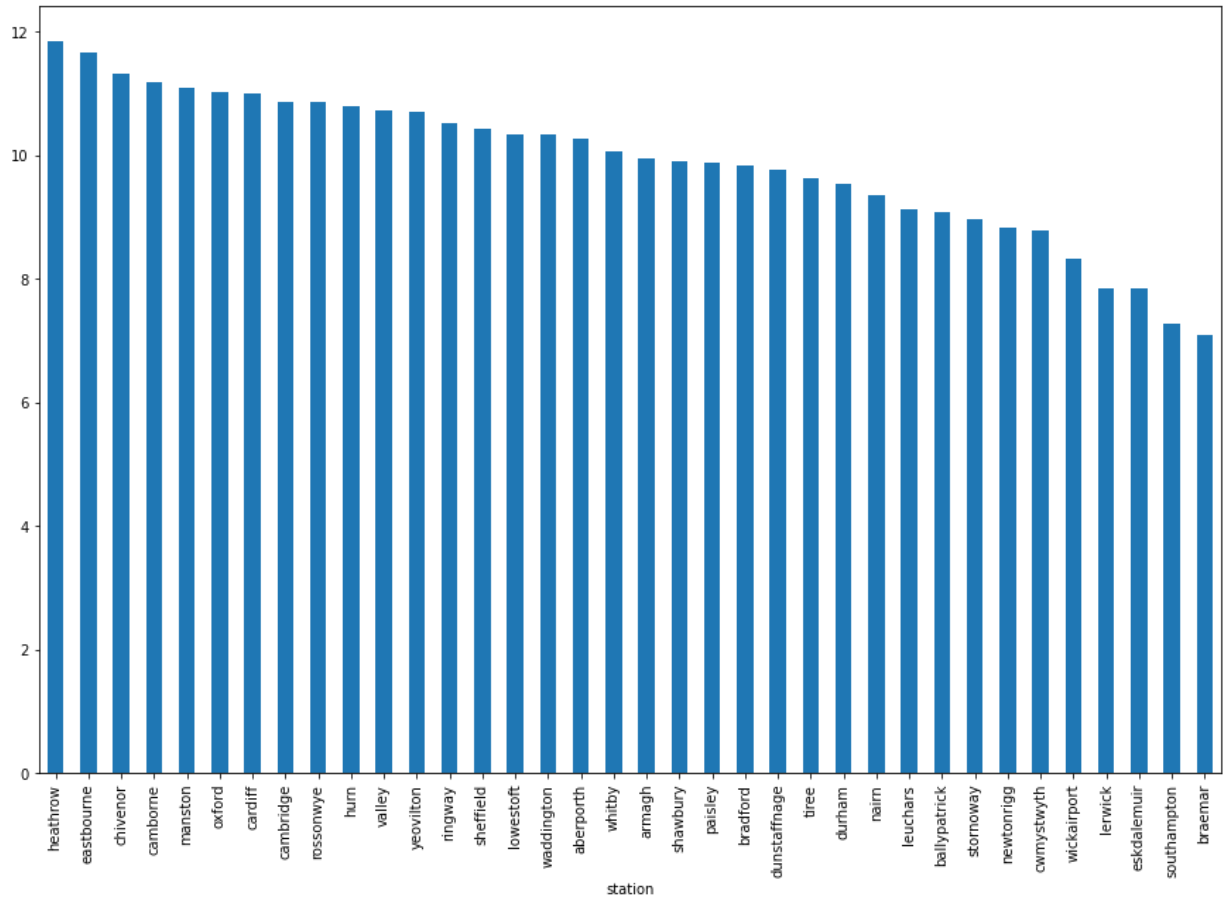
In [65]:
```
# Add an attribute mean temperature which is calculated from the
# average of the mean daily maximum and mean daily minimum temperature
df['tmean'] = (df['tmax']+df['tmin'])/2
```

In [66]:
```
df.isnull().sum()   # We have successfully replaced all missing values
```

Out[66]:
```
year       0
month      0
tmax       0
tmin       0
af         0
rain       0
sun        0
station    0
date       0
tmean      0
dtype: int64
```

In [67]:
```
# Explortary data analysis - mean temperature pattern according to stations
station_tmean = df.groupby('station')['tmean'].mean()
plt.rcParams['figure.figsize'] = 15,10
station_tmean.nlargest(36).plot(kind='bar')
```

Out[67]:
```
<AxesSubplot:xlabel='station'>
```



In [68]:
```
# using groupby to calculate average value among all cities in monthly basis
df = df.groupby(['date']).agg({'year':'mean','month':'mean','tmax':'mean','tm
df.head()
```

Out[68]:

	date	year	month	tmax	tmin	tmean	af	rain	sun
0	2000-01-01	2000.0	1.0	7.900000	2.155556	5.027778	7.500000	69.790659	76.511835
1	2000-02-01	2000.0	2.0	8.855556	2.980556	5.918056	4.555556	102.646215	93.037600
2	2000-03-01	2000.0	3.0	10.497222	3.938889	7.218056	3.694444	48.658333	114.609822
3	2000-04-01	2000.0	4.0	11.090973	3.928571	7.509772	3.371429	106.980000	140.606211
4	2000-05-01	2000.0	5.0	15.565714	7.277143	11.421429	0.171429	66.200000	194.596852

In [69]:
```
# Reading from another Excel in order to extract values of total energy
# consumption in accordance to each month, which is our target attribute
with pd.ExcelFile('Electricity_ODS.xls') as xls:
    df_energy = pd.read_excel(xls, '5_3')
# Select appropraite columns and rows from the dataframe
df_energy = df_energy.iloc[:,[0,1,2]]
df_energy = df_energy[4:-1]
# Rename column name
df_energy = df_energy.rename(columns={'5 ELECTRICITY':'year','Unnamed: 1':'mo
# Filter the dataframe in the range of year 2000 to 2019
df_energy = df_energy[(df_energy['year']>=2000) & (df_energy['year']<=2019)].
df_energy
```

Out[69]:

	year	month	energy_total
0	2000	January	6.6136
1	2000	February	6.3047
2	2000	March*	6.9653
3	2000	April	5.7452
4	2000	May	5.554
...	...	...	...
235	2019	August	3.311
236	2019	September	3.3959
237	2019	October	4.0072
238	2019	November	4.3475
239	2019	December	4.2403

240 rows × 3 columns

In [70]:

```
# insert the target attribute energy_total into the main dataset
# in respect to each month from 2000 to 2019
df['energy_total'] = df_energy['energy_total']
df.head()
```

Out[70]:

	date	year	month	tmax	tmin	tmean	af	rain	sun
0	2000-01-01	2000.0	1.0	7.900000	2.155556	5.027778	7.500000	69.790659	76.511835
1	2000-02-01	2000.0	2.0	8.855556	2.980556	5.918056	4.555556	102.646215	93.037600
2	2000-03-01	2000.0	3.0	10.497222	3.938889	7.218056	3.694444	48.658333	114.609822
3	2000-04-01	2000.0	4.0	11.090973	3.928571	7.509772	3.371429	106.980000	140.606211
4	2000-05-01	2000.0	5.0	15.565714	7.277143	11.421429	0.171429	66.200000	194.596852

In [71]:

```
df = df.astype({'energy_total':'float64'})
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 240 entries, 0 to 239
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   date          240 non-null    datetime64[ns]
 1   year          240 non-null    float64       
 2   month         240 non-null    float64       
 3   tmax          240 non-null    float64       
 4   tmin          240 non-null    float64       
 5   tmean         240 non-null    float64       
 6   af            240 non-null    float64       
 7   rain          240 non-null    float64       
 8   sun           240 non-null    float64       
```

```
 9   energy_total  240 non-null    float64
dtypes: datetime64[ns](1), float64(9)
memory usage: 18.9 KB
```

In [72]:

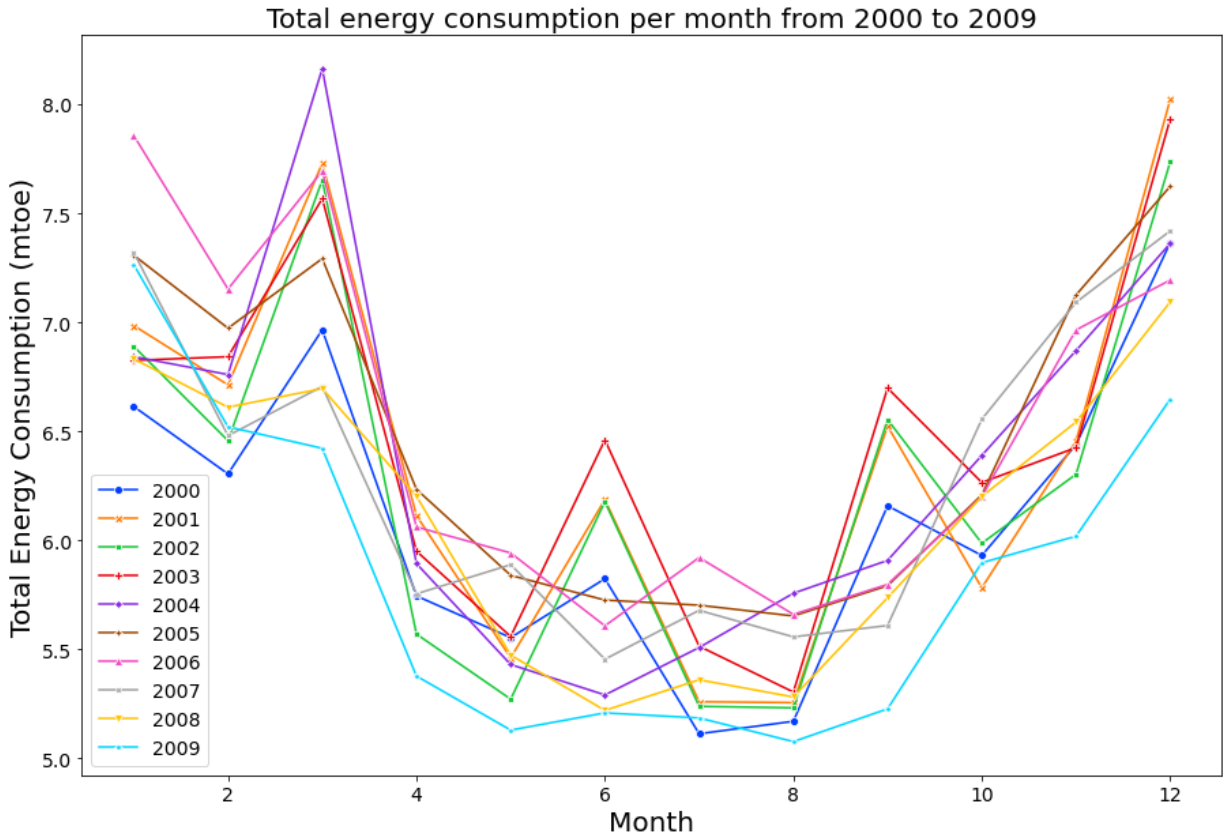
```
# check data staistic info, no abnormal value is found
df.describe().T.round(2)
```

Out[72]:

	count	mean	std	min	25%	50%	75%	max
year	240.0	2009.50	5.78	2000.00	2004.75	2009.50	2014.25	2019.00
month	240.0	6.50	3.46	1.00	3.75	6.50	9.25	12.00
tmax	240.0	13.39	4.39	3.67	9.49	13.23	17.48	22.94
tmin	240.0	6.55	3.64	-2.58	3.43	6.19	10.04	13.19
tmean	240.0	9.97	3.99	0.54	6.45	9.61	13.80	18.06
af	240.0	3.03	3.70	0.00	0.28	1.38	4.58	21.18
rain	240.0	78.77	29.76	19.14	57.08	74.29	94.49	185.09
sun	240.0	123.02	38.26	53.61	87.08	123.66	154.18	221.95
energy_total	240.0	5.60	1.09	3.23	4.97	5.58	6.42	8.16

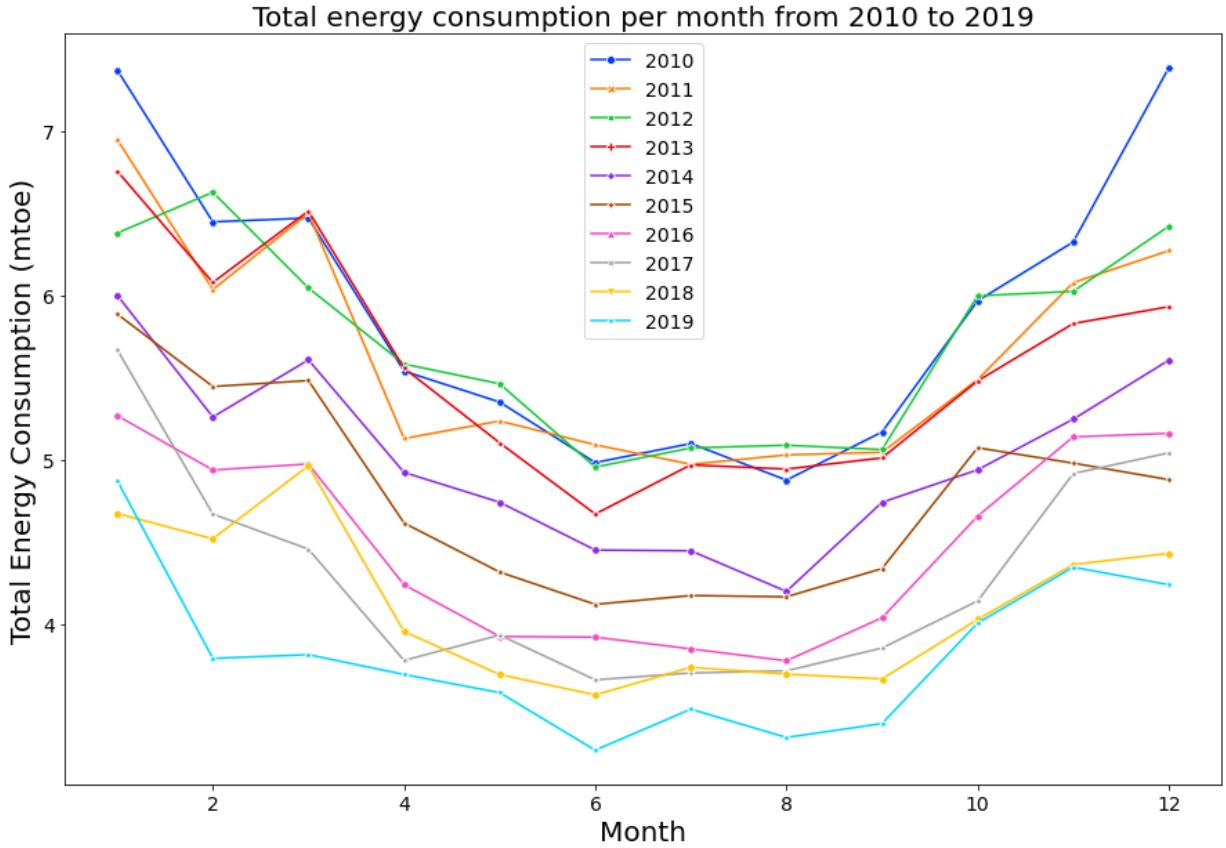
In [73]:

```
# divide the dataframe into two by year
df_1 = df[df['date'].dt.year<=2009]
df_2 = df[df['date'].dt.year>2009]

# Explortary data analysis
# Use seaborn to plot the linechart of total energy consumption
# per month from 2000 to 2009
sns.lineplot(x=df['date'].dt.month,y='energy_total',data=df_1,ci=None, hue=df
plt.title('Total energy consumption per month from 2000 to 2009', fontsize=20
plt.legend(range(2000,2010),fontsize=14)
plt.xlabel('Month',fontsize=20)
plt.ylabel('Total Energy Consumption (mtoe)',fontsize=20)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



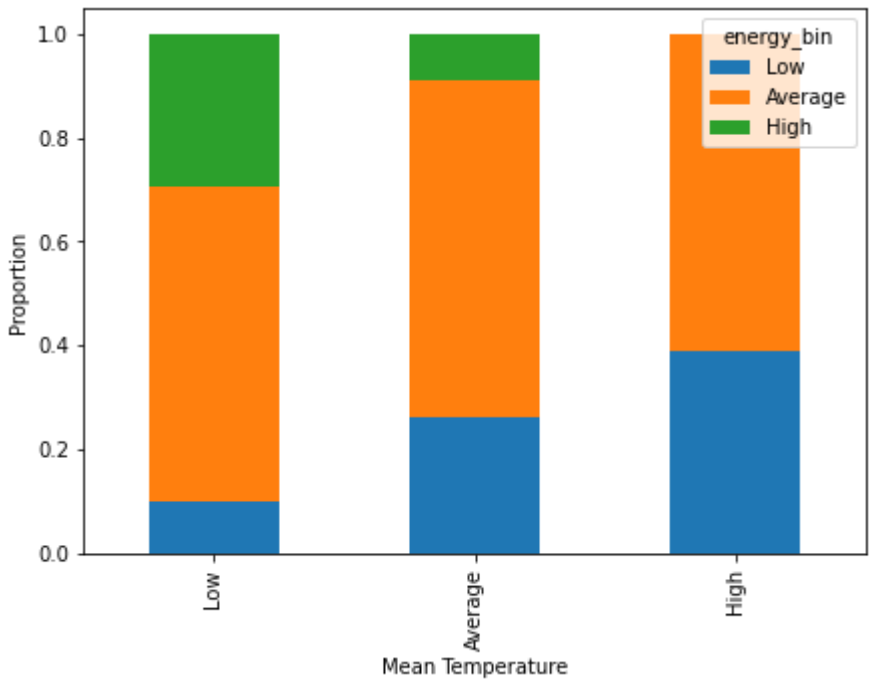
In [74]:

```
# plot the linechart of total energy consumption per month from 2010 to 2019
sns.lineplot(x=df['date'].dt.month,y='energy_total',data=df_2,ci=None, hue=df
plt.title('Total energy consumption per month from 2010 to 2019', fontsize=20
plt.legend(range(2010,2020),fontsize=14)
plt.xlabel('Month',fontsize=20)
plt.ylabel('Total Energy Consumption (mtoe)',fontsize=20)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



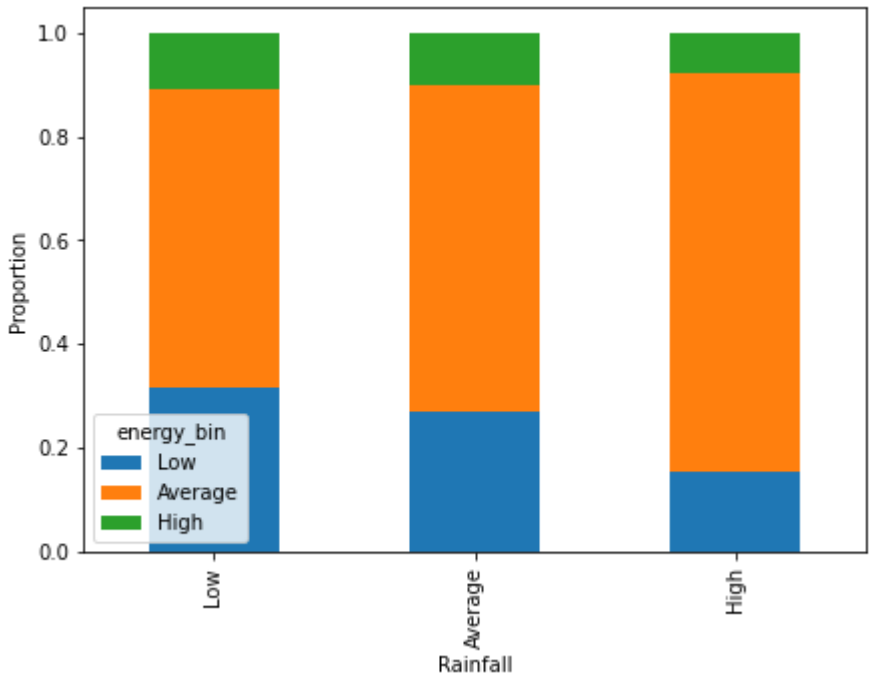
In [75]:
```
# Make bins and analysis relationship between amount of mean temperature
# and energy consumption
bins_energy=[3,5,7,9]
bins_tmean=[0,6,12,18]
group=['Low','Average','High']
df['energy_bin']=pd.cut(df['energy_total'],bins_energy,labels=group)
df['tmean_bin']=pd.cut(df['tmean'],bins_tmean,labels=group)
energy_bin=pd.crosstab(df['tmean_bin'],df['energy_bin'])
energy_bin.div(energy_bin.sum(1).astype(float),axis=0).plot(kind='bar',stacke
plt.xlabel('Mean Temperature')
plt.ylabel('Proportion')
```

Out[75]:
```
Text(0, 0.5, 'Proportion')
```



In [76]:
```
# Make bins and analysis relationship between the amount of rainfall and ener
bins_rain=[0,60,120,190]
group=['Low','Average','High']
df['rain_bin']=pd.cut(df['rain'],bins_rain,labels=group)
rain_bin=pd.crosstab(df['rain_bin'],df['energy_bin'])
rain_bin.div(rain_bin.sum(1).astype(float),axis=0).plot(kind='bar',stacked=Tr
plt.xlabel('Rainfall')
plt.ylabel('Proportion')
```

Out[76]:
```
Text(0, 0.5, 'Proportion')
```



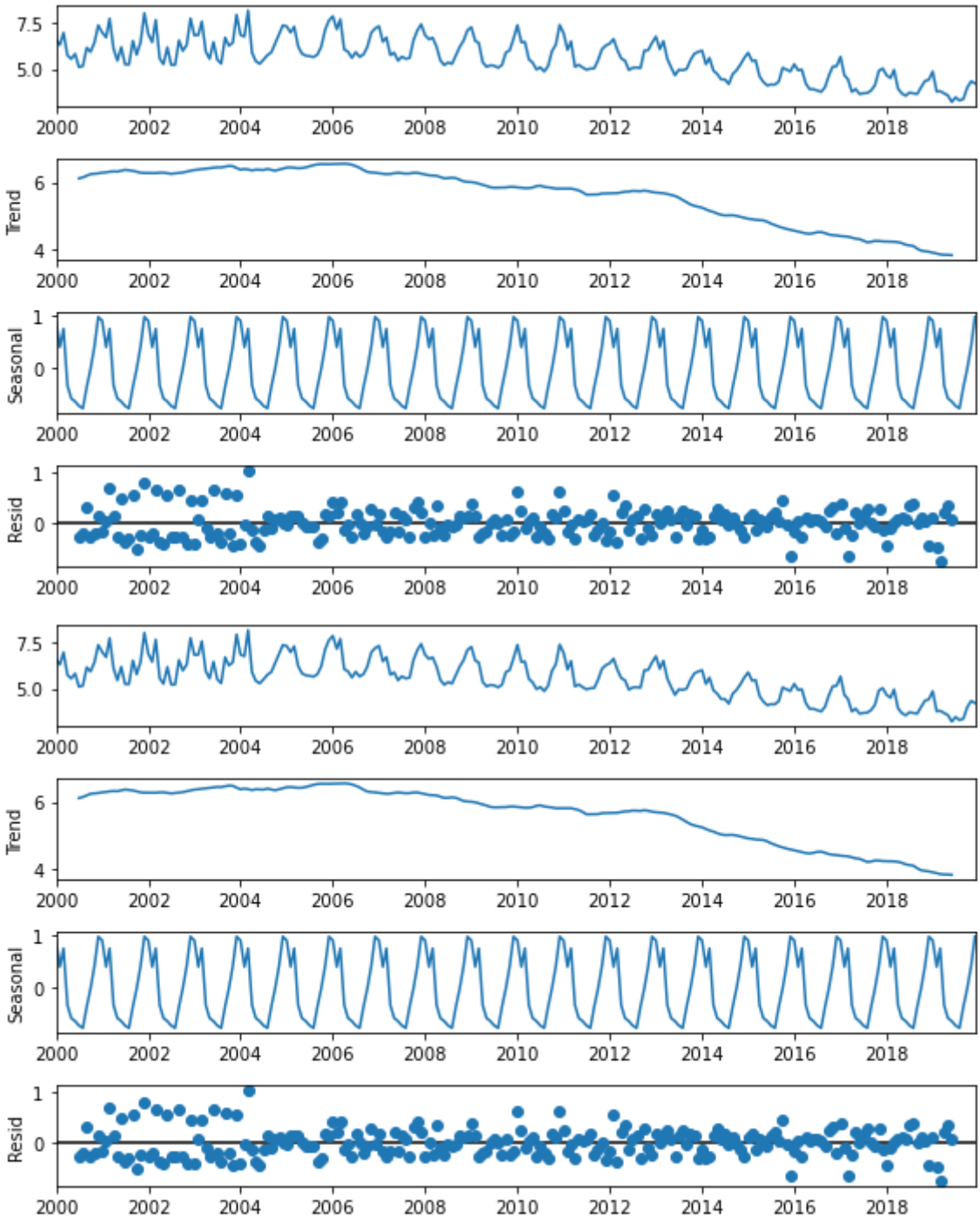
In [77]:
```
# drop the bins which we created for the exploration part
df = df.drop(['energy_bin','tmean_bin','rain_bin'], axis=1)
```

Time series decomposition

In [78]:
```
# separate other attributes from the predicting attribute into X
X = df.drop(['date','energy_total'],axis=1)
# separate the predicting attribute into y for model training
y = df['energy_total']
```

In [79]:
```
# import statsmodels for time series decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
energy_dec = pd.DataFrame(y.values,index=df['date'],columns=['energy_total'])
plt.rcParams['figure.figsize'] = 8,5
result = seasonal_decompose(energy_dec)
result.plot()
```

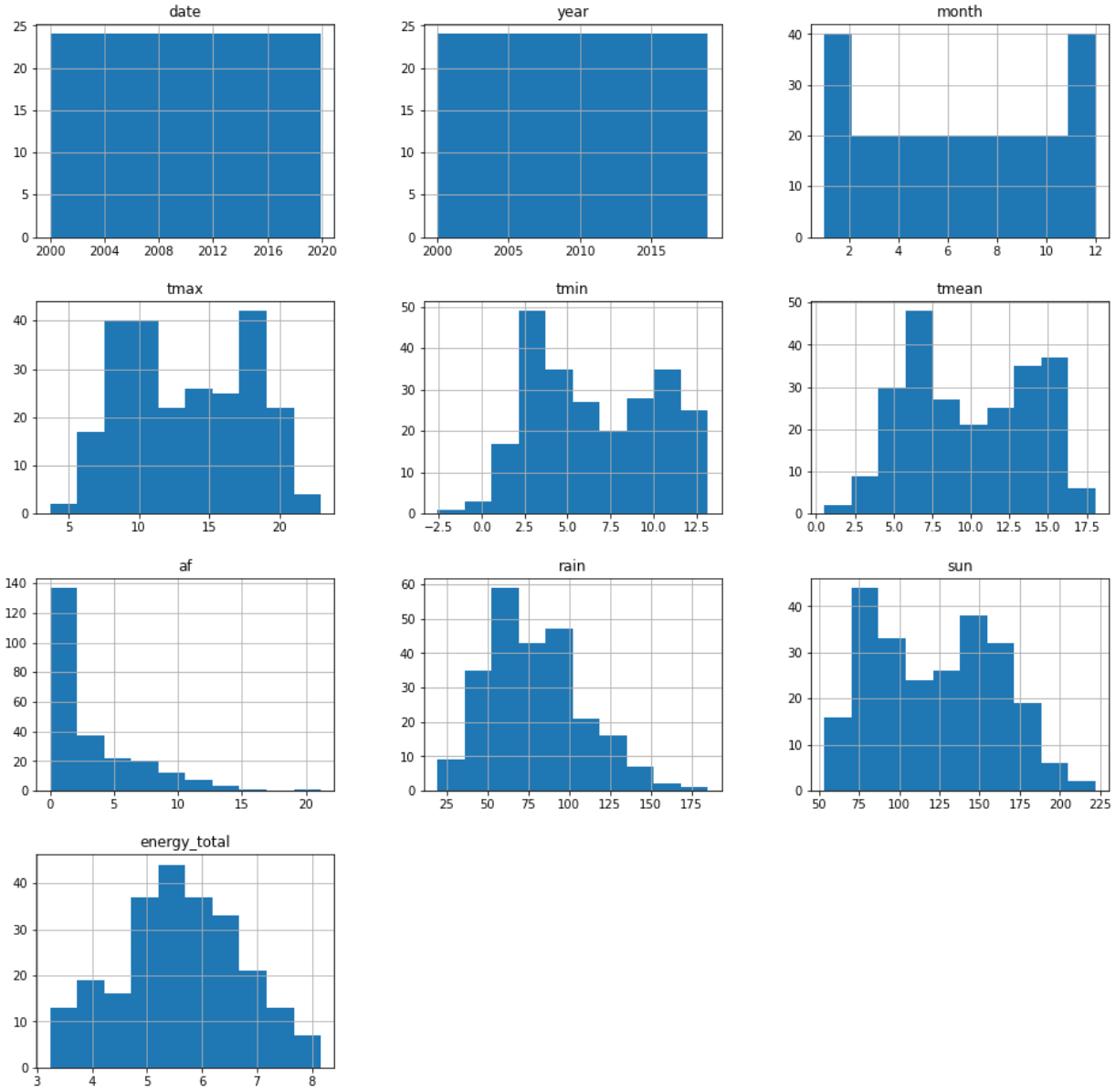
Out[79]:



Data Preprocessing - Scaling and Log Transformation

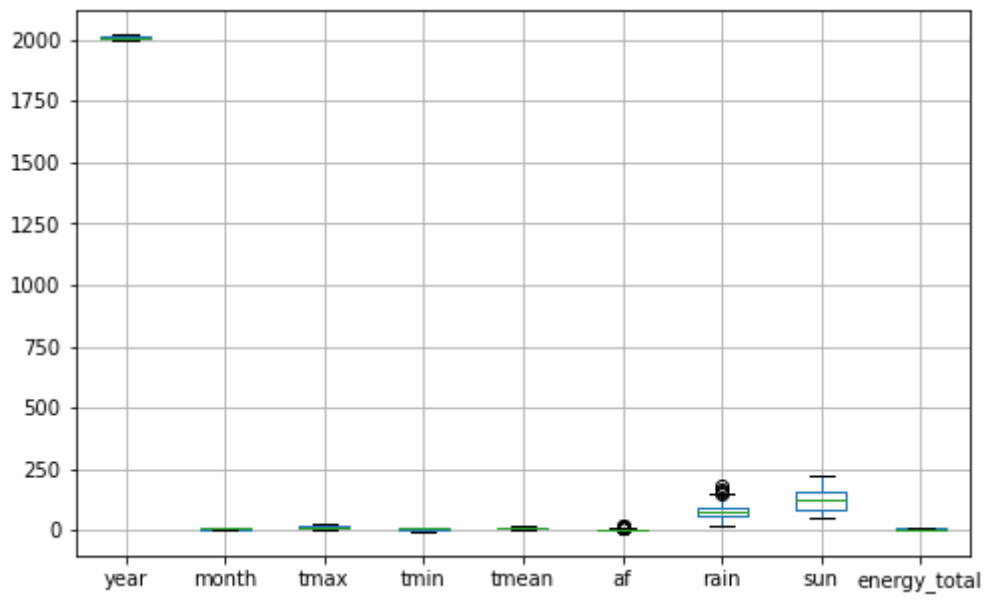
In [80]:
```
df.hist(figsize=(16, 16))        # showed skewed distributoin in af and rain
```

Out[80]:
```
array([[<AxesSubplot:title={'center':'date'}>,
        <AxesSubplot:title={'center':'year'}>,
        <AxesSubplot:title={'center':'month'}>],
       [<AxesSubplot:title={'center':'tmax'}>,
        <AxesSubplot:title={'center':'tmin'}>,
        <AxesSubplot:title={'center':'tmean'}>],
       [<AxesSubplot:title={'center':'af'}>,
        <AxesSubplot:title={'center':'rain'}>,
        <AxesSubplot:title={'center':'sun'}>],
       [<AxesSubplot:title={'center':'energy_total'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



In [81]:
```
df.boxplot()        # showed a large scale of values in rain and sun
```

Out[81]: <AxesSubplot:>



In [82]:
```
df.var().round(2)        # showed high variance in sun and rain
```

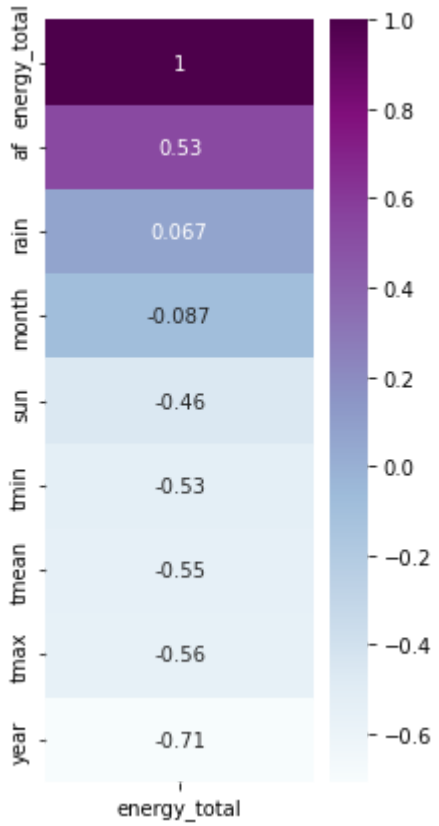
```
/var/folders/v0/sn6863fj5437h78v71__0rcm0000gn/T/ipykernel_3816/2829177571.py:
1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise TypeEr
ror.  Select only valid columns before calling the reduction.
  df.var().round(2)       # showed high variance in sun and rain
```

Out[82]:
```
year              33.39
month             11.97
tmax              19.23
tmin              13.28
tmean             15.93
af                13.68
rain             885.77
sun             1464.09
energy_total       1.19
dtype: float64
```

In [83]:
```
# perform scaling for all independent variables, excluding year and month
from sklearn.preprocessing import StandardScaler
to_scale = ['tmax','tmin','tmean','af','rain','sun']
ss = StandardScaler()
ss.fit(df[to_scale])
df[to_scale] = pd.DataFrame(ss.transform(df[to_scale]),columns=to_scale)
```

In [84]:
```
 # perform normalisation using log transformation for each variable
 # not normally distributed
from sklearn.preprocessing import PowerTransformer
pt=PowerTransformer()
df[['af','rain']]=pd.DataFrame(pt.fit_transform(df[['af','rain']]),columns=['
```

In [85]:
```
# Visualise correlation matrix between target attribute and other attributes
corr=df.corr()
corr_energy = corr['energy_total']
corr_energy = np.asarray(corr_energy).reshape(9,1)
corr_energy = pd.DataFrame(corr_energy,index=corr.columns,columns=['energy_to
corr_energy=corr_energy.sort_values('energy_total',ascending=False)
plt.rcParams['figure.figsize'] = 3,7
sns.heatmap(corr_energy,annot=True,cmap="BuPu")
plt.show()
```



In [86]:
```
# import necessary libraries for kbest and mutual_info_regression
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.feature_selection import mutual_info_regression

# replace 0 in case of any infinite and null value to avoid error in followin
df = df.replace((np.inf,-np.inf,np.nan,-np.nan),0).reset_index(drop=True)
```

In [87]:
```
# call the k-best method and pass X and y as inputs,
# f_regression is used as a parameter here
kbest = SelectKBest(f_regression, k = 'all')
ordered_features = kbest.fit(X,y)
```

In [88]:
```
# use the k-best method to find the most highest predictive features of 'y'
# save the scores into a dataFrame
df_scores = pd.DataFrame(ordered_features.scores_, columns=['Score'])
# save the feature names of those scores into a dataFrame
df_columns = pd.DataFrame(X.columns, columns = ['Feature_name'])
# combine the two dataFrames
feature_rank = pd.concat([df_scores,df_columns],axis=1)
# rank the features by score - based on the f_regression scoring function
feature_rank.nlargest(11,'Score')
```

Out[88]:

	Score	Feature_name
0	239.377612	year
2	108.567035	tmax
4	102.872496	tmean
3	92.976102	tmin
5	91.843839	af
7	63.495258	sun

	Score	Feature_name
1	1.823019	month
6	1.449326	rain

In [89]:
```
# use mutual_info_regression as a alternative method to predict the relations
mu_ifo = mutual_info_regression(X,y)
# save the features names into a series
mu_data = pd.Series(mu_ifo, index = X.columns)
mu_data.sort_values(ascending=False)
```

Out[89]:
```
year     0.556604
af       0.349827
tmean    0.305327
tmax     0.288459
month    0.265695
tmin     0.249815
sun      0.232660
rain     0.103892
dtype: float64
```

Main part - Data Analytics Models

In [90]:
```
# import library to implement data analytics models,
# which include polynomial regression and SVR
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
```

In [91]:
```
# initialise blocking time series splitter as a model performance measure
class BlockingTimeSeriesSplit():
    def __init__(self, n_splits):
        self.n_splits = n_splits

    def get_n_splits(self, X, y, groups):
        return self.n_splits

    def split(self, X, y=None, groups=None):
        n_samples = len(X)
        k_fold_size = n_samples // self.n_splits
        indices = np.arange(n_samples)

        margin = 0
        for i in range(self.n_splits):
            start = i * k_fold_size
            stop = start + k_fold_size
            mid = int(0.5 * (stop - start)) + start
            yield indices[start: mid], indices[mid + margin: stop]

btscv = BlockingTimeSeriesSplit(n_splits=10)
```

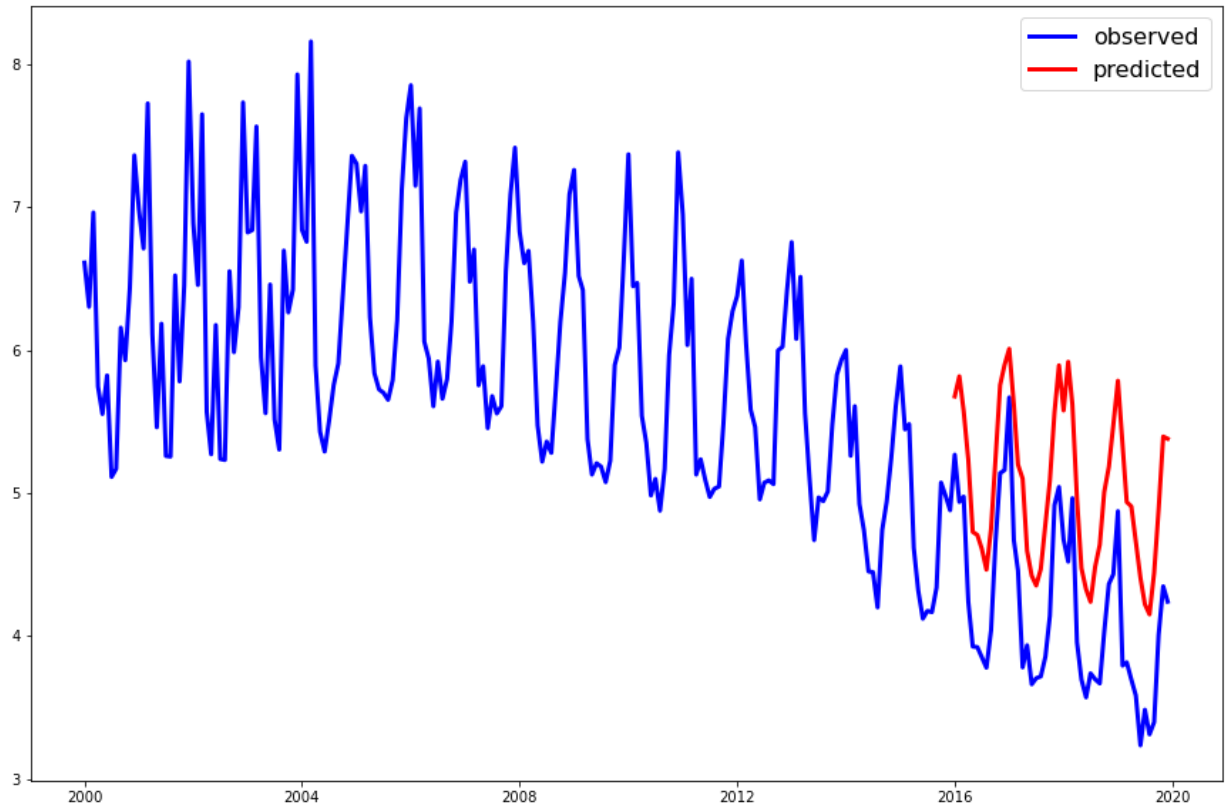
In [92]:
```
# initialise Time series splitter as a model performance measure
tscv = TimeSeriesSplit(n_splits=10)
```

```
# create and transform independent features into polynomial features
poly = PolynomialFeatures(degree=2,include_bias=False)
poly_features = poly.fit_transform(X)

# split data into train and test data without shuffle
x_train,x_test,y_train,y_test = train_test_split(X,y, test_size = 0.2, shuffl

def regression(model,x,y):
    """ split into training and testing data, train the model with training d
    plot a comparison graph between observed and predicted values,
    and evaluate the corresponding regression models """
    x_train,x_test,y_train,y_test = train_test_split(x,y, test_size = 0.2, sh
    model.fit(x_train,y_train)
    # run model to predict y value
    y_pred = model.predict(x_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    # the first performance measure - RMSE
    print('RMSE is: ', rmse.round(3))
    # the second performance measure - R2
    print('Accuracy is: ', model.score(x_test,y_test).round(3))
    # the third performance measure - time series cross validation
    score = cross_val_score(model,x_train,y_train,cv=tscv,scoring='r2')
    print('Time Series Cross validation Accuaracy: ', score.mean().round(3))
    # the fourth performance measure - blocked cross validation
    score2 = cross_val_score(model,x_train,y_train,cv=btscv,scoring='r2')
    print('Blocking Time Series Cross validation Accuaracy: ', score2.mean().
    # plot a line chart to evaluate the goodness of fit between observed and
    x = df['date']
    y = df['energy_total']
    plt.plot(x,y,color='blue',linewidth=3)
    plt.plot(x[-48:],y_pred,color='red',linewidth=3)
    plt.legend(['observed','predicted'], fontsize=16)
```

In [93]:

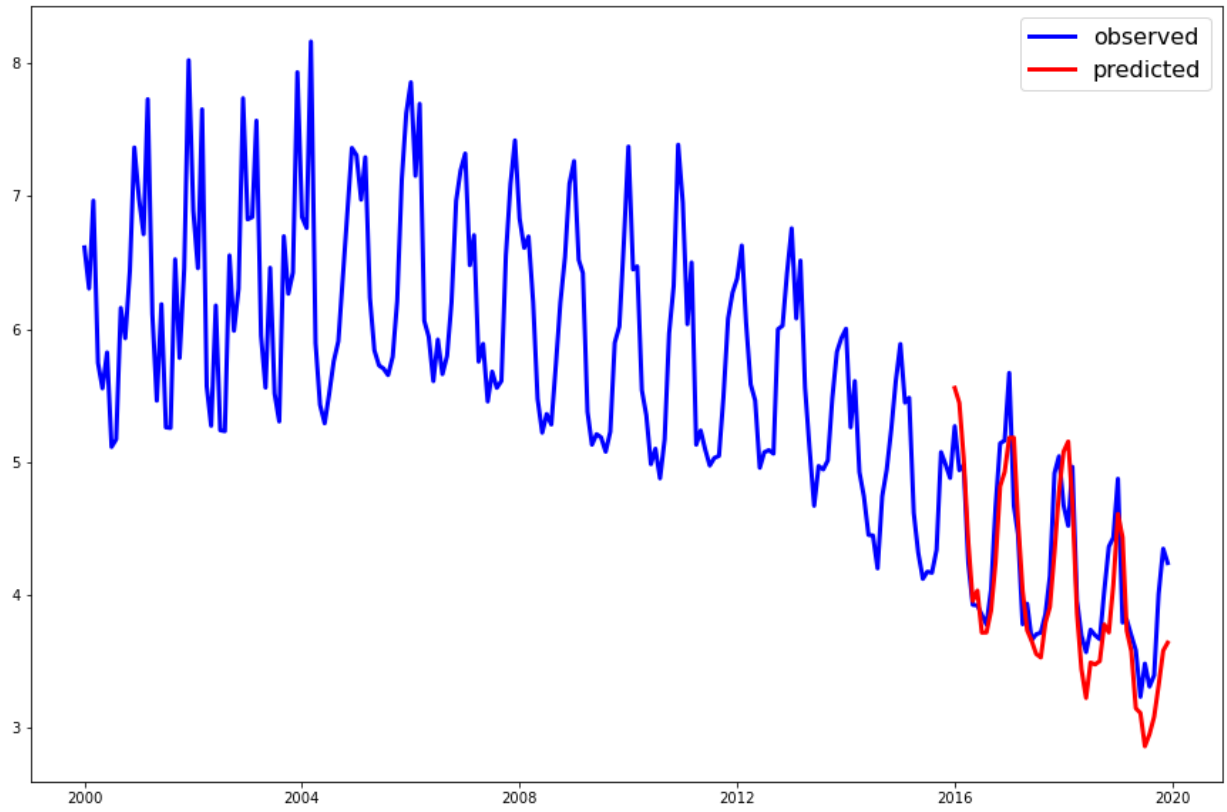
```
# create and test for Linear regression - poor result (discarded in report)
plt.rcParams['figure.figsize'] = 15,10
LG_model = LinearRegression()
regression(LG_model,X,y)
```

```
RMSE is:  0.896
Accuracy is:  -1.349
Time Series Cross validation Accuaracy:  0.47
Blocking Time Series Cross validation Accuaracy:  -13.423
```



In [94]:
```
# create and test for Polynomial regression - good result
regression(LG_model,poly_features,y)
```

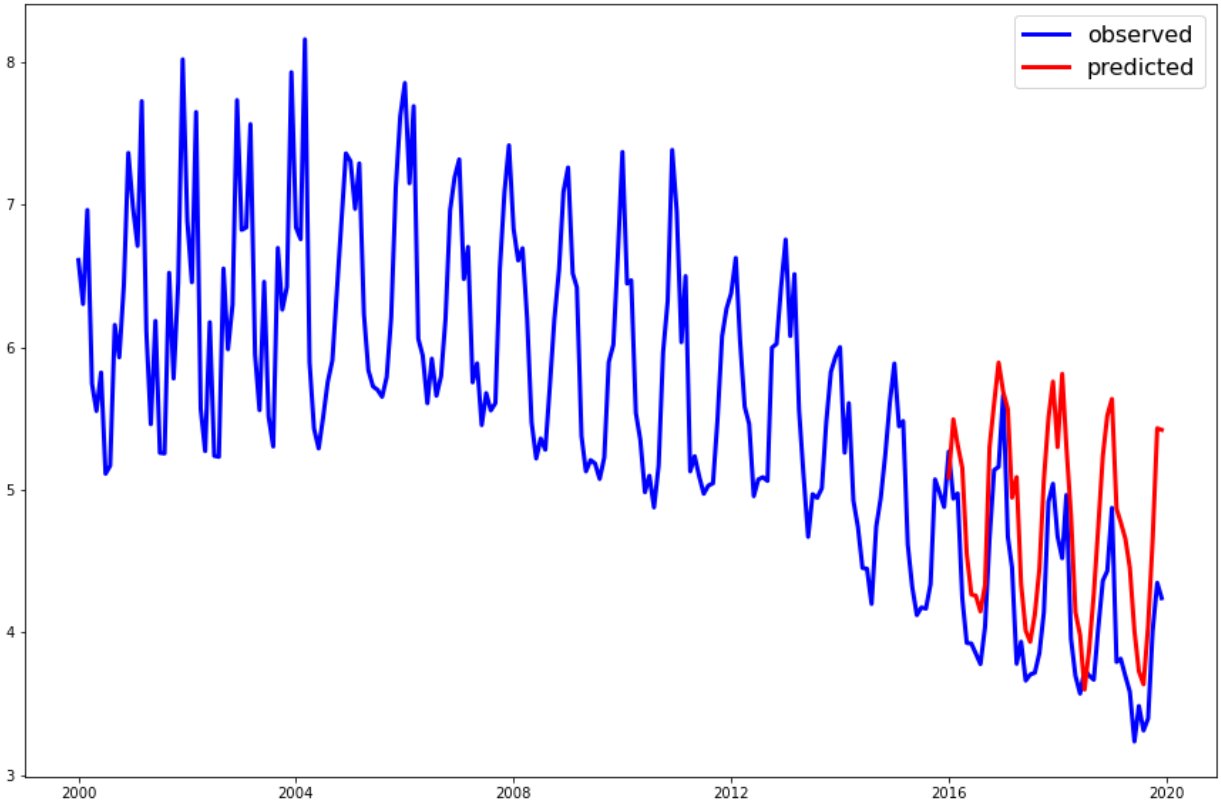
```
RMSE is:  0.36
Accuracy is:  0.621
Time Series Cross validation Accuaracy:  -11.025
Blocking Time Series Cross validation Accuaracy:  -15.244
```



In [95]:
```
# create and test for SVR  -- good result
```

```
SVR_model = SVR(kernel='linear')
regression(SVR_model,X,y)
```

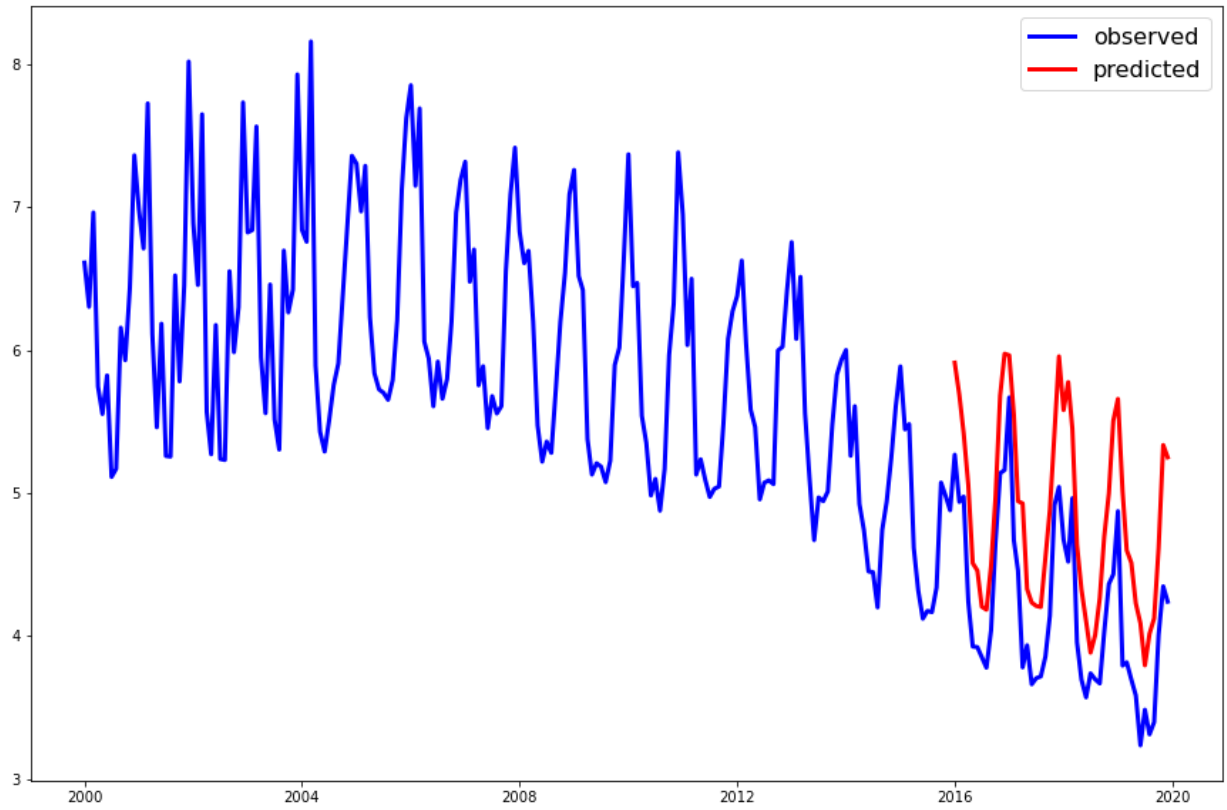
```
RMSE is:  0.698
Accuracy is:  -0.426
Time Series Cross validation Accuaracy:  0.347
Blocking Time Series Cross validation Accuaracy:  -0.033
```



In [96]:

```
# test for another kernel on SVR - shows similar result when a low gamma is s
SVR_model = SVR(kernel='rbf', C= 100, gamma= 0.0001)
regression(SVR_model,X,y)
```

```
RMSE is:  0.705
Accuracy is:  -0.453
Time Series Cross validation Accuaracy:  0.41
Blocking Time Series Cross validation Accuaracy:  0.354
```


observed
predicted
8
7
6
5
4
3
2000
2004
2008
2012
2016
2020