Web Traffic Forecasting

1. Data Exploration

1.1 Import Libraries

In [47]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pandas import read_csv, datetime, Series

from pandas.plotting import autocorrelation_plot

import os
import matplotlib.pyplot as plt
from matplotlib import pyplot
import re
from math import sqrt
%matplotlib inline


from sklearn.metrics import mean_squared_error, r2_score, median_absolute_erro
import statsmodels.api as sm
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from pyramid.arima import auto_arima

from __future__ import print_function

np.random.seed(12345)

print(os.listdir("./"))
```

```
['.ipynb_checkpoints', '99_plot_arima.ipynb', 'all', 'Arima-WebTrafficExploration-ARIMA.ipynb', 'data', 'io_plot_arima.ipynb', 'pics', 'pll_plot_arima.ipynb', 'ws_plot_arima.ipynb']
```

1.2 Import Data

Read data from processed.csv which was created from above train_1.csv using the code above. The above code is creating new features based off the first column "Page" which contains more details about the wiki page.

In [48]:
```
df = pd.read_csv('./all/processed.csv').fillna(0)
print("Number of wiki pages is",df.shape[0]) #gives number of row count
print("Number of days is",df.shape[1]) #gives number of col count
```

```
Number of wiki pages is 145063
Number of days is 557
```

- PageName: Page Name
- Lang: Page Language
- Project: Wikipedia project (e.g. wikipedia, wikimedia etc)
- Access: type of access (e.g. desktop)
- Agent: type of agent (e.g. spider).
- Page: contains all the information together. In other words, each article name has the following format: 'name_project_access_agent' (e.g. 'AKB48_zh.wikipedia.org_all-access_spider').

1.3. Data cleaning: daily views of individual pages

We randomly sample 100 wiki pages and organize them in order of the mean views. The idea is to make prediction for pages which have more number of views.

In [49]:
```
df['Avg'] = df.iloc[:,2:-6].mean(axis=1)
dfs = df.sample(n=100, random_state=2)
dfs = dfs.drop(['Unnamed: 0','PageName','Access', 'Agent', 'Avg','Lang','Proje
dfs.set_index('Page', inplace=True)
dfs = dfs.T
dfs = dfs.reindex(dfs.mean().sort_values().index, axis=1) #ascending order of
dfs.describe()
dfs.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\dfs.c
```

In [50]:
```
dfs.head()
```

Out[50]:

Page	恶魔少爷别吻我_zh.wikipedia.org_all-access_spider	User:Anasserrihani_commons.wikimedia.org_all-access_spider	User:Rxy_www.mediawil w
2015-07-01	0.0	0.0	
2015-07-02	0.0	0.0	
2015-07-03	0.0	0.0	
2015-07-04	0.0	0.0	
2015-07-05	0.0	0.0	

5 rows × 100 columns

2. Data Transformation

2.1 Removing outliers

We need to remove outliers from the time series because they result in artifical trends in data which in our case is usually because of some current events causing huge number of people to view a wiki page. For example, while exploring the dataset I found a wiki aritcle that had <100 views on all days expect one when it had 100,000+ hits. On further investigation, I found that the wiki page was that of an obscure pop artist, who had been caught in a drug related police investigation on that day resulting in huge traffic to his wiki page. ITs unrealistic to expect the model to predict such anomalies as they may not repeat on a periodic basis. It is best to remove such data points in order to get more prediction for web traffic.

In order to determine the outliers, an upper and lower bound is defined and any value outside that band is considered an outlier. The bounds are a combination of mean absolute error and deviation of the data, both of which are calculated around the rolling mean with frequency 7 days. The outliers are replaced by the upper or lower bound.

2.2 Normalization

In order to have a consistent range of data and be able to compare results between different time series, each time series is normalized between 0 and 1. The formula for normalizing the ith element of a series is

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

The following function called "RemoveOutliers" removes outliers as well as normalizes the data and returns a series with outlier removed.

In [51]:
```
def RemoveOutliers(input, window, plot_intervals=False, scale=1, plot_anomalie

    """
        series - dataframe with timeseries
        window - rolling window size
        plot_intervals - show confidence intervals
        plot_anomalies - show anomalies

    """
    series = input.copy()
    rolling_mean = series.rolling(window=window).mean()
    #print(rolling_mean)
    plt.figure(figsize=(15,5))
    plt.title("Moving average\n window size = {}".format(window))
    plt.plot(rolling_mean, "g", label="Rolling mean trend")

    # Plot confidence intervals for smoothed values
    if plot_intervals:
        mae = mean_absolute_error(series[window:], rolling_mean[window:])
        deviation = np.std(series[window:] - rolling_mean[window:])
        lower_bond = rolling_mean - (mae + scale * deviation)
        upper_bond = rolling_mean + (mae + scale * deviation)
        plt.plot(upper_bond, "r--", label="Upper Bound")
        plt.plot(lower_bond, "r--", label="Lower Bound")

        # Having the intervals, find abnormal values
        if plot_anomalies:
            #series[(series<lower_bond)|(series>upper_bond)] = upper_bond
            anomalies = []
            anomalies = series[(series<lower_bond)|(series>upper_bond)]
            series[series<lower_bond] = lower_bond
            series[series>upper_bond] = upper_bond
            #anomalies = pd.DataFrame(index=series.index)
            #anomalies[series<lower_bond] = series[series<lower_bond]
            #anomalies[series<lower_bond] = series.loc[series<lower_bond]
            #anomalies[series>upper_bond] = series[series>upper_bond]
            #print(anomalies)
            plt.plot(anomalies, "ro", markersize=10, label="Anomalies")

    plt.plot(series[window:], label="Actual values")
    plt.legend(loc="upper left")
    plt.grid(True)

    series=(series-series.min())/(series.max()-series.min()) #normalize data b
    series[series==0]=10**-6  #replace zero by by a non-zero low value
    return series
```

3. Methodology

3.1 Stationary series

A time series is **stationary** if all of its statistical properties— mean, variance, autocorrelations, etc. —are constant in time. Thus, it has no trend, no heteroscedasticity, and a constant degree of “wiggliness.” **Stationary series** has no trend or seasonality. White noise is an example of stationary data.

Dickey-Fuller Test can be used to determine if a series is stationary. If t-statistics is less than critical value, a series is considered to be stationary.

Null Hypothesis (H0): time series is non-stationary and has a unit root, against Alternate Hypothesis (H1): time series is stationary and does not have a unit root

The more negative test statistic, the more likely we are to reject the null hypothesis (we have a stationary dataset)

The following transformations can be used to stationarize the data:

- Logarithmic : Convert multiplicative pattern to additive pattern
- First Difference: to stationarize a series with strong trend
- Seasonal Difference: to remove gross features of seasonality

In [62]:

```
def AdFullerTest(data):
    dftest = adfuller(data, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags

    print("Results of Dickey-Fuller Test")
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
    #if t-stats < critical, series is stationary
    return
```

3.2 Auto-Correlation and Partial Correlation Function

Correlation measures the extent of linear relationship between two vairables. **Auto-correlation** measures the linear relationship between lagged values of a time series. **Partial Auto-correlation** function measures the linear relationship between lagged values of a time series after removing the affect of other lagged values. The Auto Correlation and Partial Auto-correlation can be seen using the ACF and PACF plots.

In [53]:

```
def PlotCorrelationFunc(data, length):
    plot_acf(data, lags = length)
    pyplot.show()
    plot_pacf(data, lags = length)
    pyplot.show()
    return
```

3.3 SARIMA

SARIMA includes autoregressive models, moving average models and seasoanlity. **Autoregressive models** forecast the variable of interest using a linear combination of the past values of the variable. **Moving average model** uses psat forecast erris in a regressive model. The equations for AR and MA models are as follows:

$$\text{AR}(p)\text{: } y_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \cdots + \varphi_p y_{t-p} + \varepsilon_t$$

$$\text{MA}(q)\text{: } y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

p	order of the autoregressive part
q	order of the moving-average part
d	degree of first differencing

ARIMA (p,d,q):

$$y'_t = c + \underbrace{\varphi_1 y'_{t-1} + \cdots + \varphi_p y'_{t-p}}_{\textit{autoregressive}} + \underbrace{\theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}}_{\textit{moving-average}} + \varepsilon_t$$

$$(1 - \varphi_1 B - \cdots - \varphi_p B^p)(1 - B)^d y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t$$

B : backshift notation; $(B)y_{t-1} = y_t$

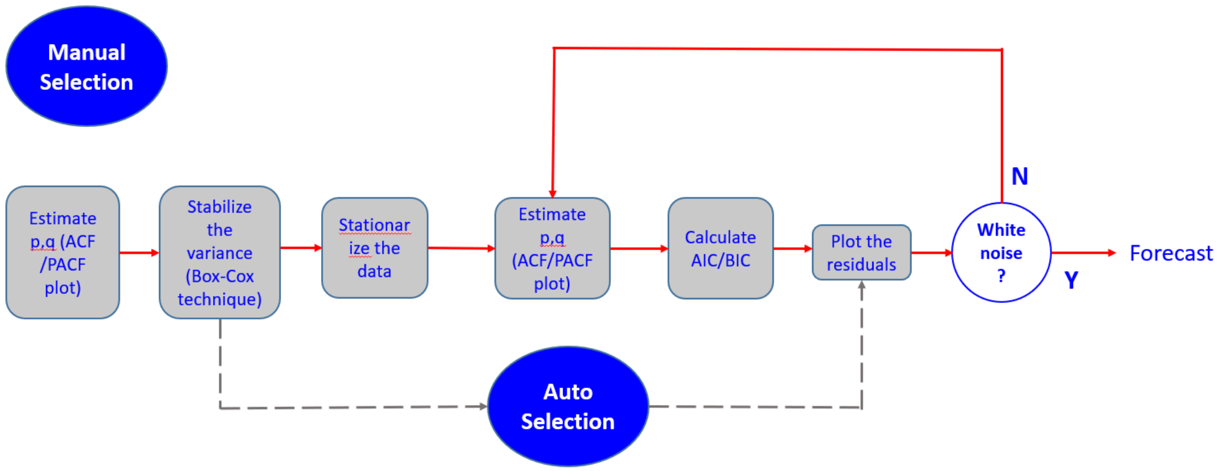
SARIMA is AR and MA models accounting for seasonality and can be written as follows:

$$\text{SARIMA}(p,d,q)(P,D,Q)_m$$

$$(1 - \varphi_1 B - \cdots - \varphi_p B^p)(1 - \phi_1 B^m - \cdots - \phi_P B^{mP})(1 - B)^d (1 - B^m)^D y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q)(1 + \Theta_1 B^m + \cdots + \Theta_Q B^{mQ})\varepsilon_t$$

P	order of the seasonal autoregressive part
D	degree of first differencing of the seasonal part
Q	order of the seasonal moving-average part
M	season period length of the series
p	order of the autoregressive part
d	degree of first differencing
q	order of the moving-average part

The following workflow shows how to find the optimum parameters and make forecasts.



The following function automates the process of finding the optimum parameters for SARIMA and make predictions:

In [54]:
```
def ARIMAStepWisePrediction(data,seas_step):
    #https://www.alkaline-ml.com/pmdarima/quickstart.html#auto-arima-example
    #https://github.com/Vijayganeshsrinivasan/Forecasting-using-ARIMA-method/

    prediction = pd.Series([])

    len = int(126/seas_step)
    for i in range(0,len): #20
        j = 21+seas_step*i
        print('(i,j) = ', i,j)
        stepwise_model = auto_arima(data[:j], start_p=0, start_q=0,max_p=4, ma
        order_in = stepwise_model.order
        seasonal_order_in = stepwise_model.seasonal_order
        #print('\nLeast AIC: '+ str(stepwise_model.aic()))
        #print('Least BIC: '+ str(stepwise_model.bic()))
        #print('order: '+ str(order_in))
        #print('seasonal order: '+ str(seasonal_order_in))

        mod = sm.tsa.statespace.SARIMAX(data[:j], trend='n', order=order_in,
        results = mod.fit()
        ##print(results.summary())
        ##print('\nPlotting Diagnostics')
        ##results.plot_diagnostics(figsize=(20, 14))
        predict_temp = results.forecast(seas_step)
        prediction= pd.concat([prediction, predict_temp])

    return(prediction)
```

3.4 Predictions and RMSE

In [55]:

```
def PlotARIMAPrediction(data, WebsiteName):
    plt.figure(figsize=(10, 5))
    #dfsn_io.index = pd.to_datetime(dfsn_io.index)

    labels = ['actual', 'prediction:7', 'prediction:14','prediction:21']
    colors=['r','g','b','m','c']

    xi = [i for i in range(0, len(dfsn_io.date))]

    plt.axes().xaxis.set_major_locator(plt.MaxNLocator(10))

    plt.plot(data.date, data.views,'-',color=colors[0],label=labels[0])
    plt.plot(predict7.date, predict7.views,'--',color=colors[1],label=labels[1
    plt.plot(predict14.date, predict14.views,'--',color=colors[2],label=labels
    plt.plot(predict21.date, predict21.views,'--',color=colors[3],label=labels


    plt.xlabel('time')
    plt.ylabel('views')
    plt.title('total daily views of "{}" \n using SARIMA\n' .format(WebsiteNam
    #plt.legend()
    plt.legend(loc="upper right")
    plt.grid(True)
    plt.show()
```

In [23]:


```
def PlotARIMAPrediction(data, WebsiteName):
    plt.figure(figsize=(10, 5))
    #dfsn_io.index = pd.to_datetime(dfsn_io.index)

    labels = ['actual', 'prediction:7', 'prediction:14','prediction:21']
    colors=['r','g','b','m','c']

    xi = [i for i in range(0, len(dfsn_io.date))]

    plt.axes().xaxis.set_major_locator(plt.MaxNLocator(10))

    plt.plot(data.date, data.views,'-',color=colors[0],label=labels[0])
    plt.plot(predict7.date, predict7.views,'--',color=colors[1],label=labels[1
    plt.plot(predict14.date, predict14.views,'--',color=colors[2],label=labels
    plt.plot(predict21.date, predict21.views,'--',color=colors[3],label=labels


    plt.xlabel('time')
    plt.ylabel('views')
    plt.title('total daily views of "{}" \n using SARIMA\n' .format(WebsiteNam
    #plt.legend()
    plt.legend(loc="upper right")
    plt.grid(True)
    plt.show()
```

In [21]:
```
def PlotARIMA_RMSE(data, predict7, predict14, predict21, WebsiteName):

    df_pred = data.join(predict7.set_index('date'), on='date', rsuffix='_7')
    df_pred = df_pred.join(predict14.set_index('date'), on='date', rsuffix='_1
    df_pred = df_pred.join(predict21.set_index('date'), on='date', rsuffix='_2
    df_pred[21:28]

    ARIMA_RMSE = pd.DataFrame({'arima_7d': [], 'arima_14d': [], 'arima_21d': [

    for i in range(3,20):
        ARIMA_RMSE = ARIMA_RMSE.append({'arima_7d': sqrt(mean_squared_error(df

    labels = ['arima_7d', 'arima_14d', 'arima_21d']
    colors=['r','g','b','m','c']

    for i in range(0,3):
        plt.plot(ARIMA_RMSE.iloc[:,i],'o-',color=colors[i],label=labels[i])

    plt.xlabel('week')
    plt.ylabel('RMSE')
    plt.title('RMSE for total daily views of "'+ WebsiteName + '" wiki \n usir
    plt.legend()
    plt.show()
```

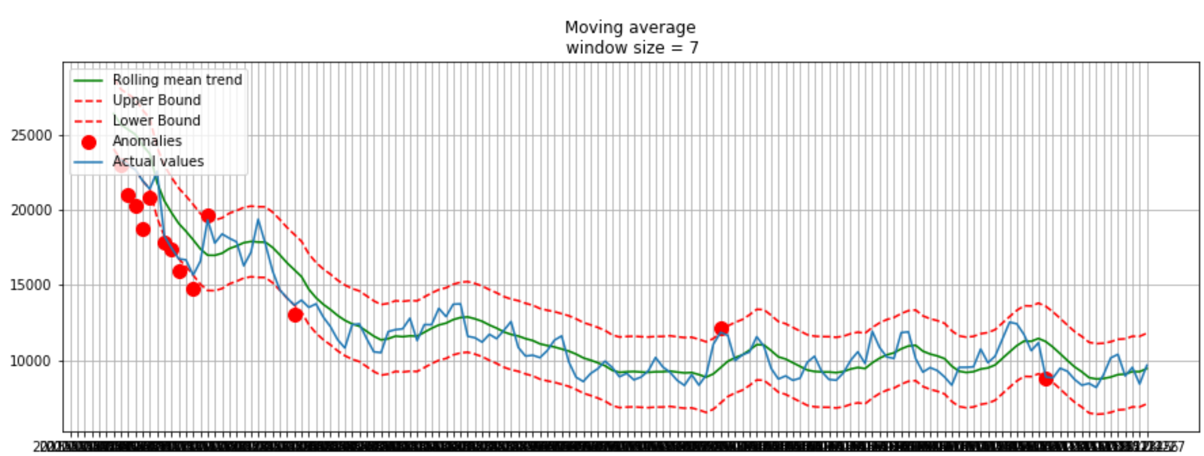
4. Results

4.1. Inside Out

We use SARIMA to make predictions for the views of the "Inside Out" wiki:

The following figure shows data with outliers removed for total daily views of russian wiki pages. The red dots are the outliers, the green line is the rolling mean average, the broken lines are upper and lower bounds while the blue line is the data with outliers removed.

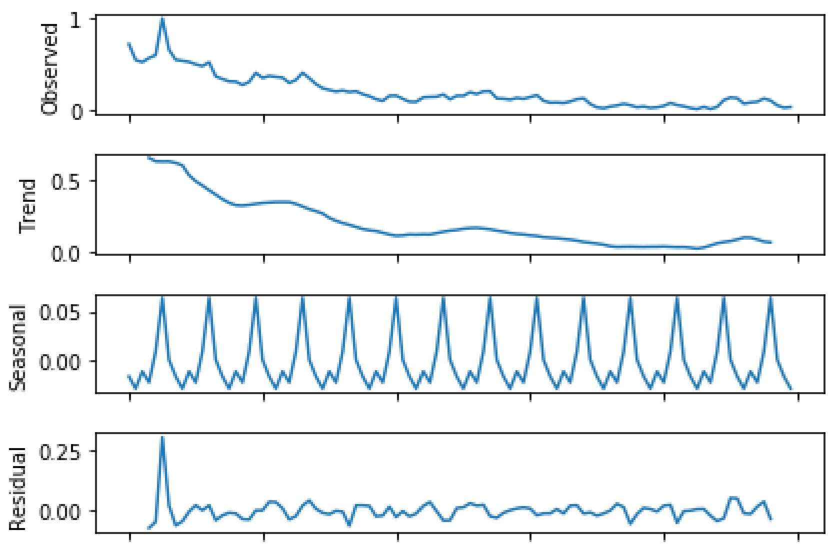
In [57]:
```
dfsn_io = RemoveOutliers(dfs.iloc[:150,98], 7, plot_intervals=True, plot_anoma

dfsn_io.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\i
```



Next the series is decomposed to show the trend, seasonality and residue in the data:

In [58]:
```
plt.figure(figsize=(500, 500))
sm.tsa.seasonal_decompose(dfsn_io[:100], freq=7).plot()
plt.show()
```

```
<Figure size 36000x36000 with 0 Axes>
```



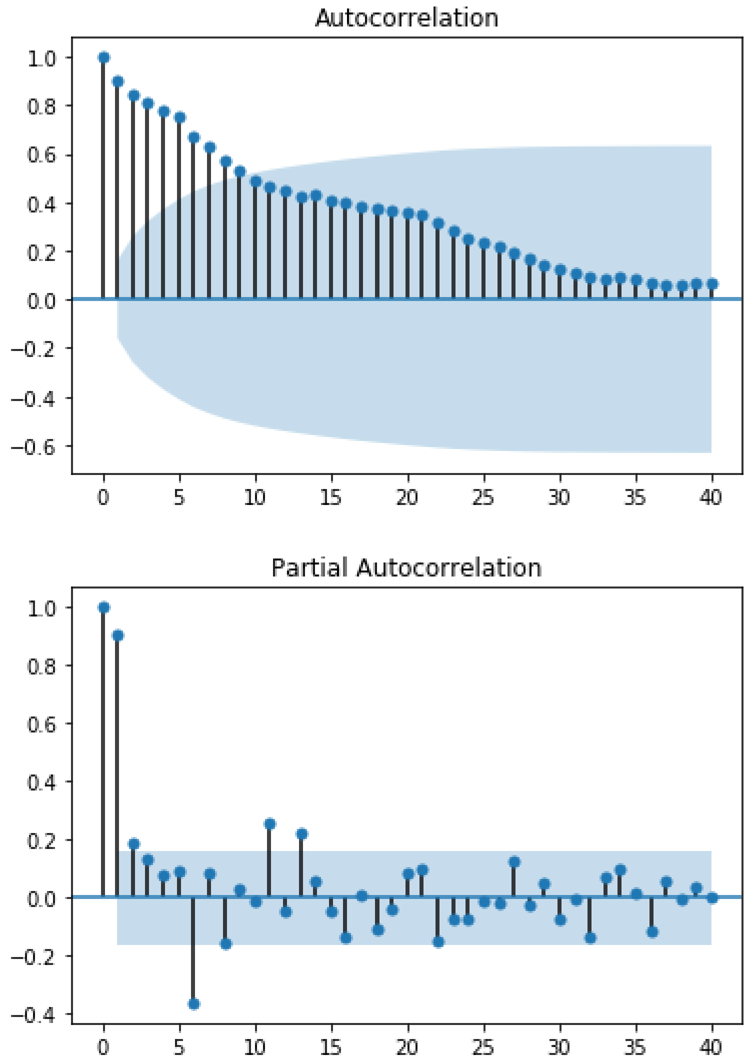
We run the Dickey-Fuller Test to determine if is stationary.

In [63]:
```
AdFullerTest(dfsn_io[:56])
```

```
Results of Dickey-Fuller Test
Test Statistic                 -2.461673
p-value                         0.125080
#Lags Used                      8.000000
Number of Observations Used    47.000000
Critical Value (1%)            -3.577848
Critical Value (5%)            -2.925338
Critical Value (10%)           -2.600774
dtype: float64
```

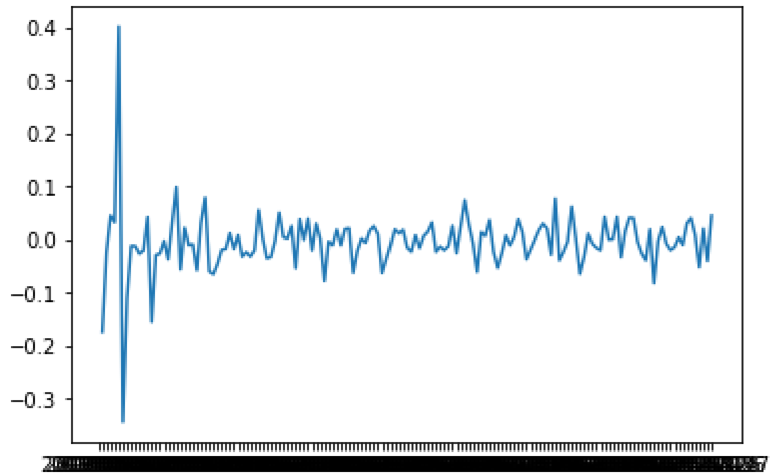
Since t-stats > critical value, series is not stationary. This is also evident from the ACF plot which is decaying slowly.

In [29]:
```
PlotCorrelationFunc(dfsn_io, 40)
```



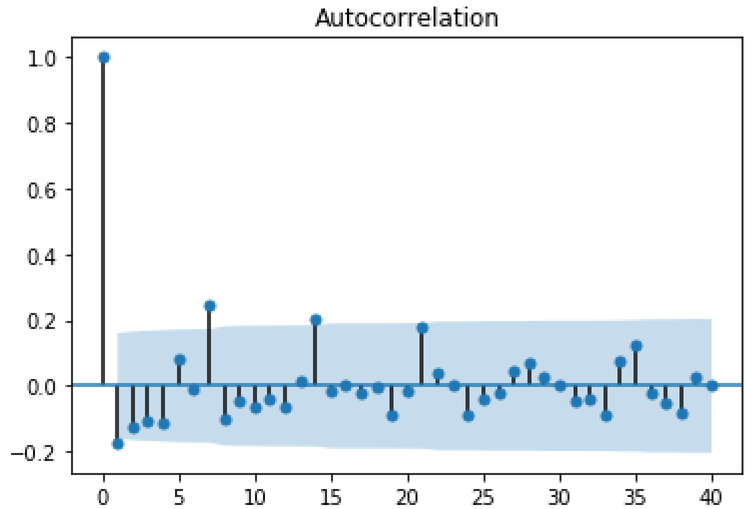
The PACF plot shows that there are no significant spikes after 2. So p = 2 is a good guess. To make the series stationary, we take the first difference and plot the data.

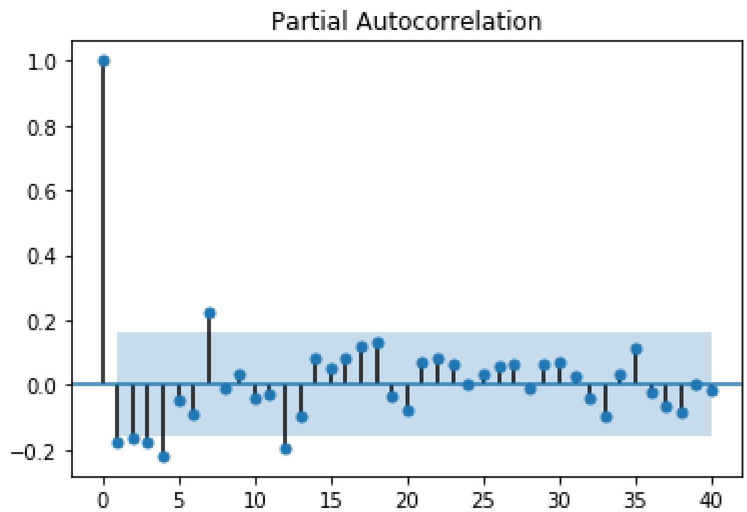
In [60]:
```
dfsn_io_diff = dfsn_io - dfsn_io.shift()
plt.plot(dfsn_io_diff)
dfsn_io_diff.dropna(inplace=True)
```



This series looks like white noise. Next we look at the ACF and PACF plots

In [31]:
```
PlotCorrelationFunc(dfsn_io_diff, 40)
```





The ACF shows that there is a strong correlation with every 7th value. So the order of seasonality is 7. Dickey Fuller test is carried out on the differenced series.

In [64]:
```
AdFullerTest(dfsn_io_diff[:56])
```

```
Results of Dickey-Fuller Test
Test Statistic                 -4.107944
p-value                         0.000940
#Lags Used                     11.000000
Number of Observations Used    44.000000
Critical Value (1%)            -3.588573
Critical Value (5%)            -2.929886
Critical Value (10%)           -2.603185
dtype: float64
```

t-stats < critical value implying that series is stationary. This is also evident from the ACF plot decays rapidly. Based on this, A good model for SARIMA may be:

(p,d,q)(P,D,Q)*m : (2,1,0)(1,1,1)*7

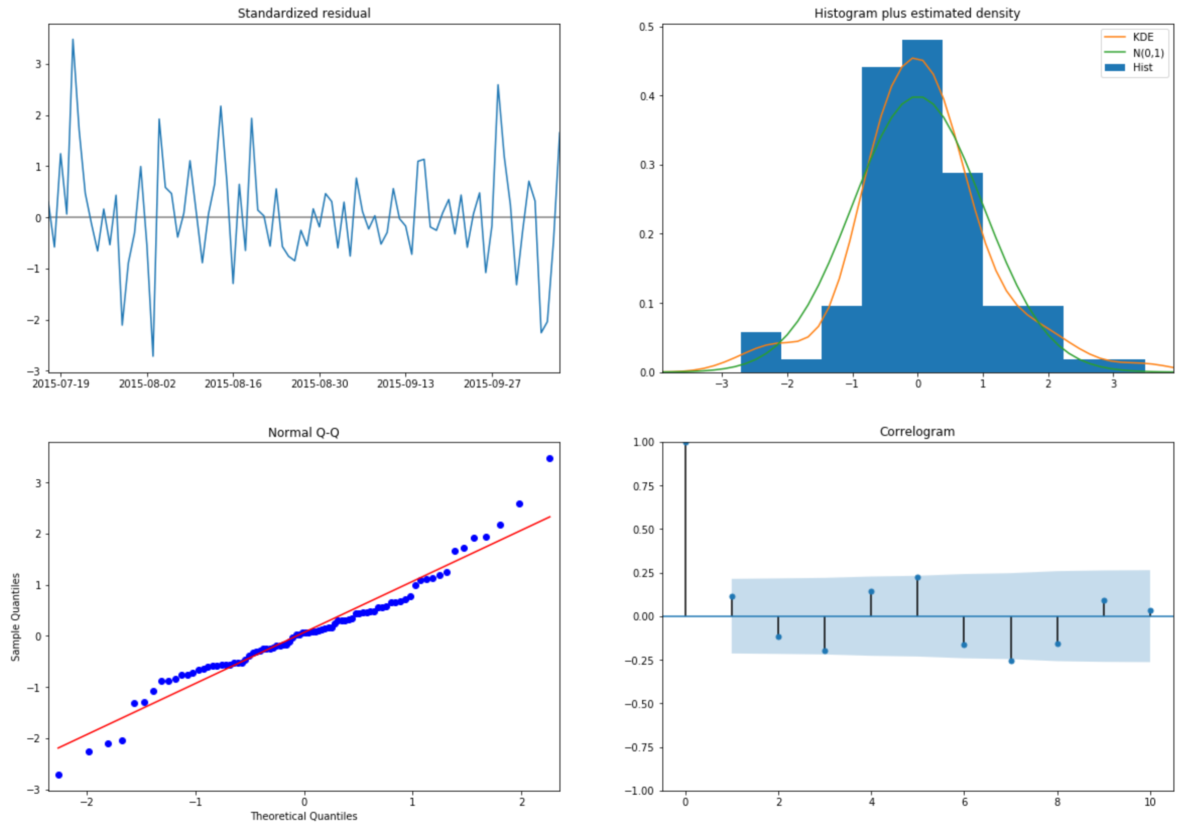
The following code runs SARIMA for difference value of p,d,q to determine the optimum values:

In [65]:

```
#https://www.alkaline-ml.com/pmdarima/quickstart.html#auto-arima-example
#https://github.com/Vijayganeshsrinivasan/Forecasting-using-ARIMA-method/blob/

stepwise_model = auto_arima(dfsn_io[:100], start_p=0, start_q=0,
                            max_p=4, max_q=3, m=7, start_P=0,
                            seasonal=True,d=1, D=1, trace=True, error_action='
                            suppress_warnings=True,stepwise=True,)
order_in = stepwise_model.order
seasonal_order_in = stepwise_model.seasonal_order
print('\nLeast AIC: '+ str(stepwise_model.aic()))
print('Least BIC: '+ str(stepwise_model.bic()))
print('order: '+ str(order_in))
print('seasonal order: '+ str(seasonal_order_in))

# order=order_in,  seasonal_order=seasonal_order_in,

mod = sm.tsa.statespace.SARIMAX(dfsn_io[:100], trend='n', order=(0,1,0),  seas
results = mod.fit()
print(results.summary())

print('\nPlotting Diagnostics')
results.plot_diagnostics(figsize=(20, 14))
plt.show()

predict = results.forecast(70)
predict.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\i
#plt.plot(predict)
#plt.plot(dfsn_io)
```

```
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=-256.553, BIC=-24
8.988, Fit time=0.731 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-253.971, BIC=-24
8.928, Fit time=0.361 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-256.256, BIC=-24
6.168, Fit time=0.812 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=-256.164, BIC=-24
6.077, Fit time=1.520 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=-255.262, BIC=-24
5.175, Fit time=1.646 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 2, 7); AIC=-239.651, BIC=-22
9.564, Fit time=1.778 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 2, 7); AIC=-254.083, BIC=-24
1.474, Fit time=5.265 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=-255.554, BIC=-24
5.467, Fit time=0.552 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fit
time=nan seconds
Total fit time: 12.674 seconds

Least AIC: -256.55287049608586
Least BIC: -248.98750476493873
order: (0, 1, 0)
seasonal order: (0, 1, 1, 7)

c:\programdata\miniconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:
171: ValueWarning: No frequency information was provided, so inferred frequenc
```

```
y D will be used.
  % freq, ValueWarning)
c:\programdata\miniconda3\lib\site-packages\statsmodels\tsa\statespace\represe
ntation.py:375: FutureWarning: Using a non-tuple sequence for multidimensional
indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which w
ill result either in an error or a different result.
  return matrix[[slice(None)]*(matrix.ndim-1) + [0]]

                                          Statespace Model Results
==============================================================================
========================================
Dep. Variable:     Inside_Out_(2015_film)_en.wikipedia.org_desktop_all-agents
No. Observations:                  100
Model:                                       SARIMAX(0, 1, 0)x(0, 1, 1, 7)
Log Likelihood                 158.516
Date:                                                     Wed, 26 Jun 2019
AIC                           -313.032
Time:                                                             17:17:34
BIC                           -308.170
Sample:                                                         07-01-2015
HQIC                          -311.078
                                                              - 10-08-2015
Covariance Type:                                                       opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L7       -0.1544      0.056     -2.739      0.006      -0.265      -0.044
sigma2         0.0013      0.000      8.006      0.000       0.001       0.002
==============================================================================
=====
Ljung-Box (Q):                      52.21   Jarque-Bera (JB):
11.40
Prob(Q):                             0.09   Prob(JB):
0.00
Heteroskedasticity (H):              0.73   Skew:
0.34
Prob(H) (two-sided):                 0.41   Kurtosis:
4.67
==============================================================================
=====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex
-step).

Plotting Diagnostics
```



The code proposes the following model parameters:

(p,d,q)(P,D,Q)*m : (0,1,0)(0,1,1)*7

which is close to the values determined from analyzing the ACF, PACF plots and Dickey-Fuller Test.

We run the ARIMAStepWisePrediction to make prediction with different step sizes of 7,14 and 21.

In [57]:

```
predict7 = ARIMAStepWisePrediction(dfsn_io,7)
predict7.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [60]:

```
predict14 = ARIMAStepWisePrediction(dfsn_io,14)
predict14.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
(i,j) =  0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.073 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.424 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.219 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.122 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.338 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.456 seconds
```

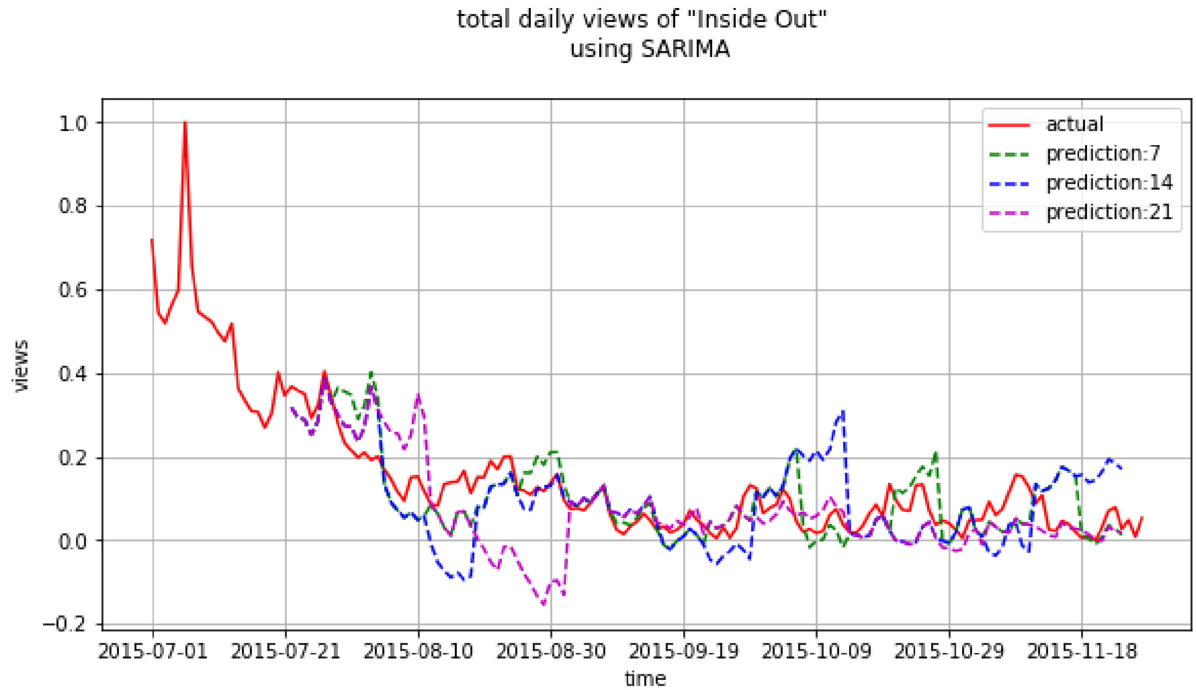
In [65]:

```
predict21 = ARIMAStepWisePrediction(dfsn_io,21)
predict21.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
(i,j) =  0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.393 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.207 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.125 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.325 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.463 seconds
```

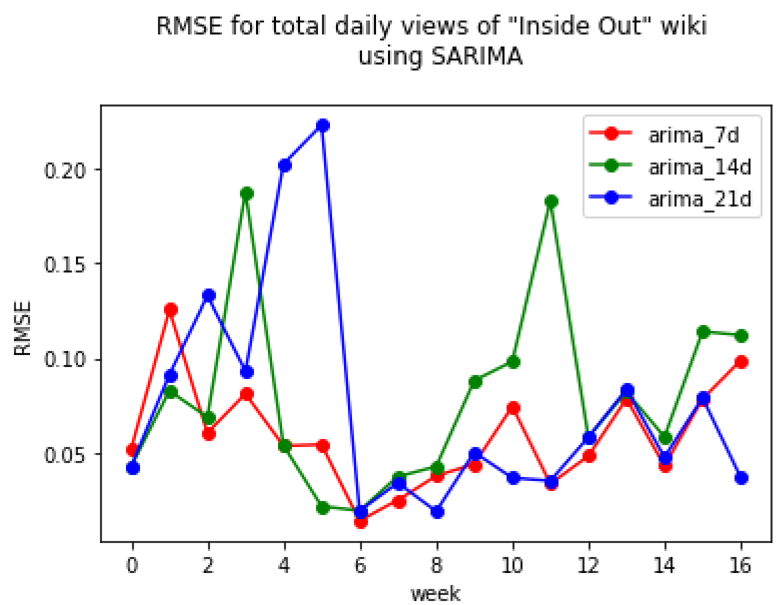
Let's plot the predictions:

In [66]:
```
dfsn_io = pd.read_csv('.\data\io\dfsn_io.csv',header=None,names=['date', 'view
predict7 = pd.read_csv('.\data\io\predict7.csv',header=None,names=['date', 'vi
predict14 = pd.read_csv('.\data\io\predict14.csv',header=None,names=['date', '
predict21 = pd.read_csv('.\data\io\predict21.csv', header=None,names=['date',
PlotARIMAPrediction(dfsn_io, "Inside Out")
```



Let's plot the RMSE for the predictions:

In [67]:
```
PlotARIMA_RMSE(dfsn_io, predict7, predict14, predict21, "Inside Out")
```



The RMSE errors from prediction with different step sizes are as follows:

RMSE over 18 weeks for IO	
7d	0.06
14d	0.09
21d	0.10

We run similar models for three other web pages:

Sanada Maru (japanese TV shows), Pretty Little Liars and William Shakespeare

and get the following results

RMSE over 18 weeks for SM	
7d	0.24
14d	0.25
21d	0.36

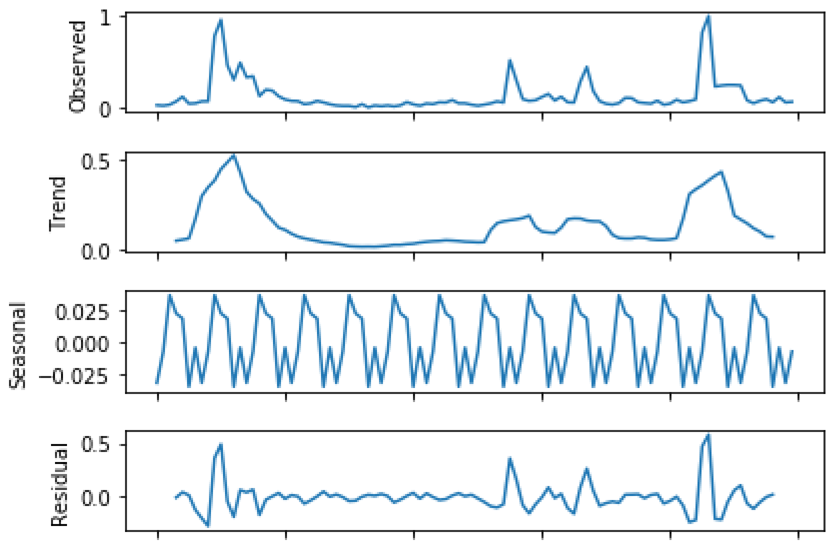
RMSE over 18 weeks for PLL	
7d	0.13
14d	0.21
21d	0.48

RMSE over 18 weeks for WS	
7d	0.12
14d	0.14
21d	0.18

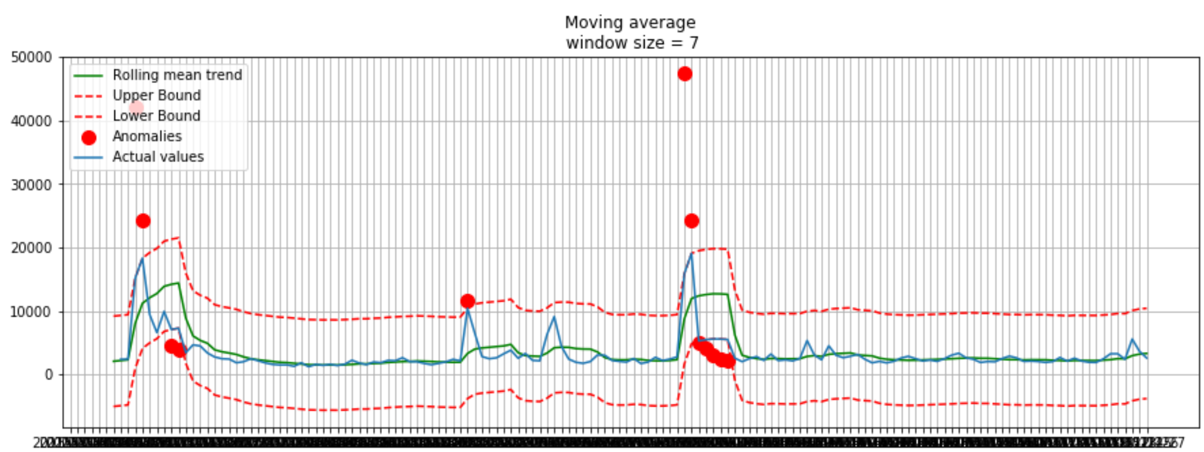
4.2. Sanada Maru (japanese TV show)

In [23]:
```
plt.figure(figsize=(500, 500))
sm.tsa.seasonal_decompose(dfsn_99[:100], freq=7).plot()
plt.show()
```

<Figure size 36000x36000 with 0 Axes>



In [15]:
```
dfsn_99 = RemoveOutliers(dfs.iloc[:150,99], 7, plot_intervals=True, plot_anoma
dfsn_99.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\9
```



In [57]:

```
predict7 = ARIMAStepWisePrediction(dfsn_99,7)
predict7.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [57]:

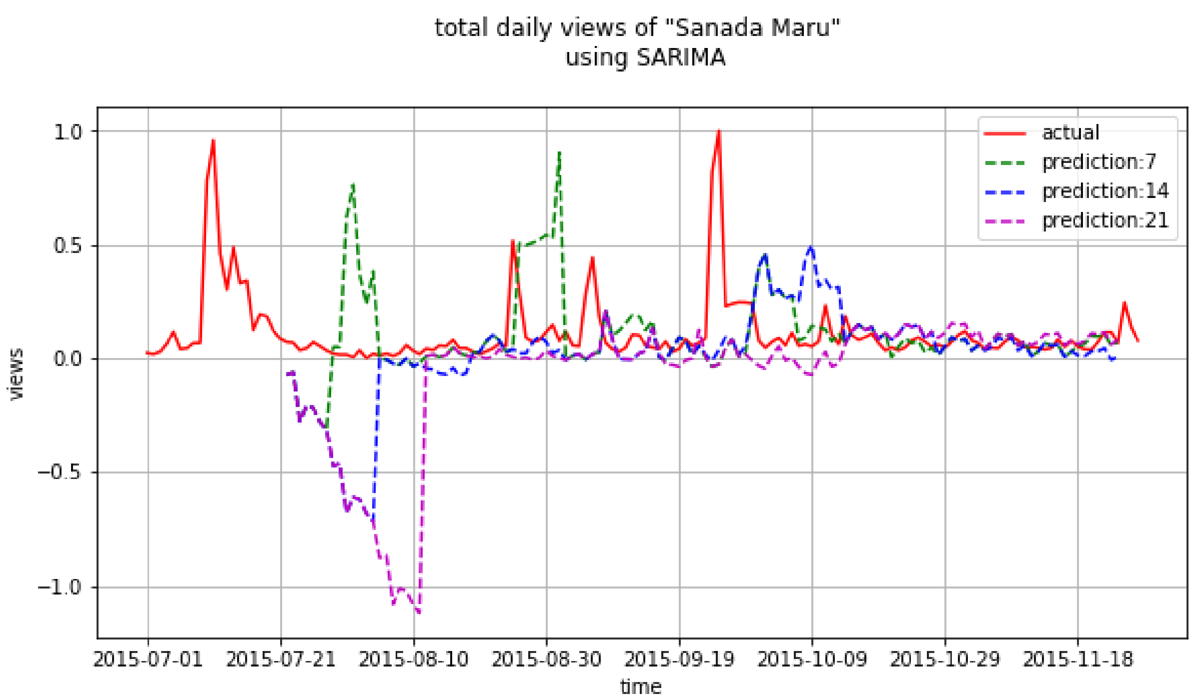
```
predict14 = ARIMAStepWisePrediction(dfsn_99,14)
predict14.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

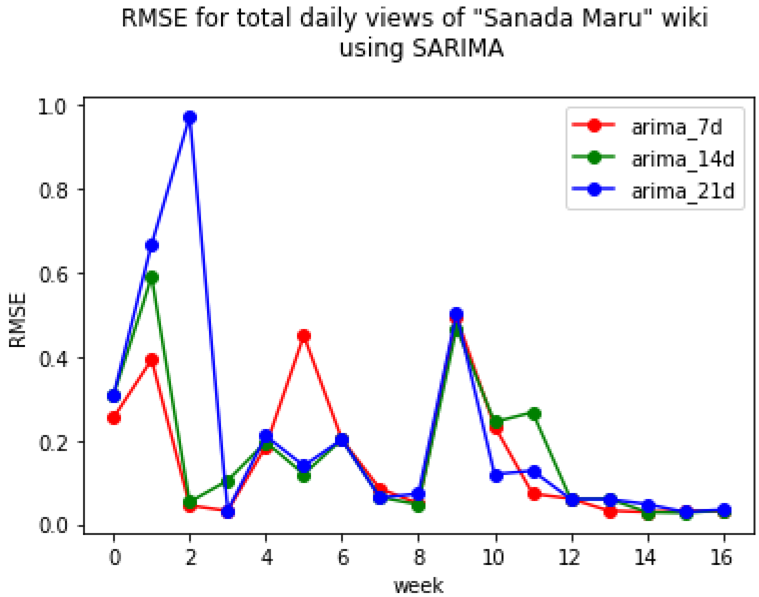
In [57]:
```
predict21 = ARIMAStepWisePrediction(dfsn_99,21)
predict21.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [38]:
```
dfsn_99 = pd.read_csv('.\data\99\dfsn_99.csv',header=None,names=['date', 'view
predict7 = pd.read_csv('.\data\99\predict7.csv',header=None,names=['date', 'vi
predict14 = pd.read_csv('.\data\99\predict14.csv',header=None,names=['date', '
predict21 = pd.read_csv('.\data\99\predict21.csv', header=None,names=['date',

PlotARIMAPrediction(dfsn_99, "Sanada Maru")
```



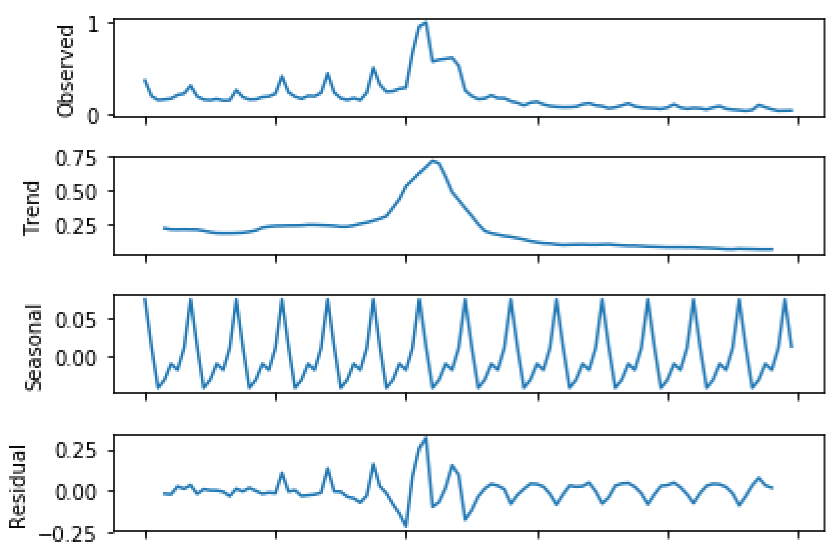
In [39]:
```
PlotARIMA_RMSE(dfsn_99, predict7, predict14, predict21, "Sanada Maru")
```



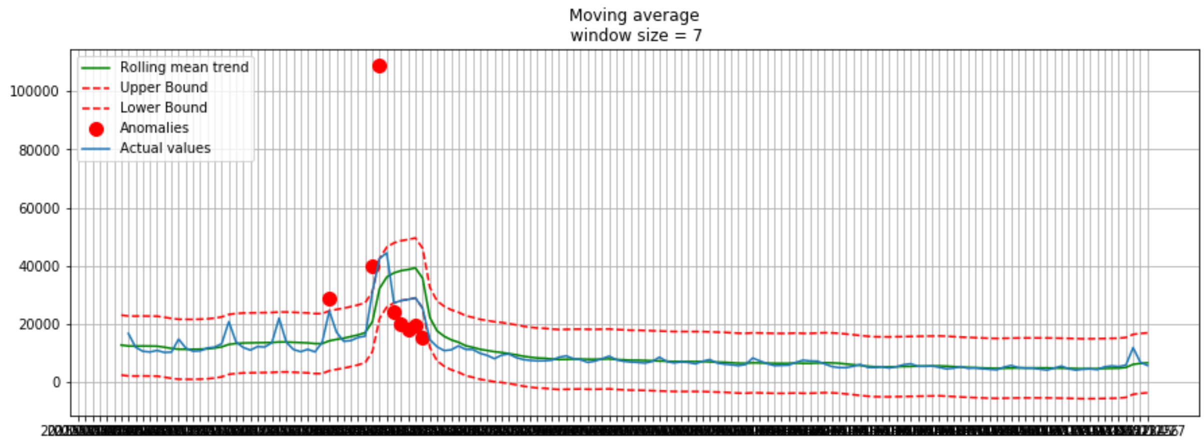
4.3. Pretty Little Liars

In [24]:
```
plt.figure(figsize=(500, 500))
sm.tsa.seasonal_decompose(dfsn_pll[:100], freq=7).plot()
plt.show()
```

```
<Figure size 36000x36000 with 0 Axes>
```



In [16]:
```
dfsn_pll = RemoveOutliers(dfs.iloc[:150,97], 7, plot_intervals=True, plot_anom
dfsn_pll.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\
```



In [57]:
```
predict7 = ARIMAStepWisePrediction(dfsn_pll,7)
predict7.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
Total fit time: 1.751 seconds
```

In [57]:

```
predict14 = ARIMAStepWisePrediction(dfsn_pll,14)
predict14.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

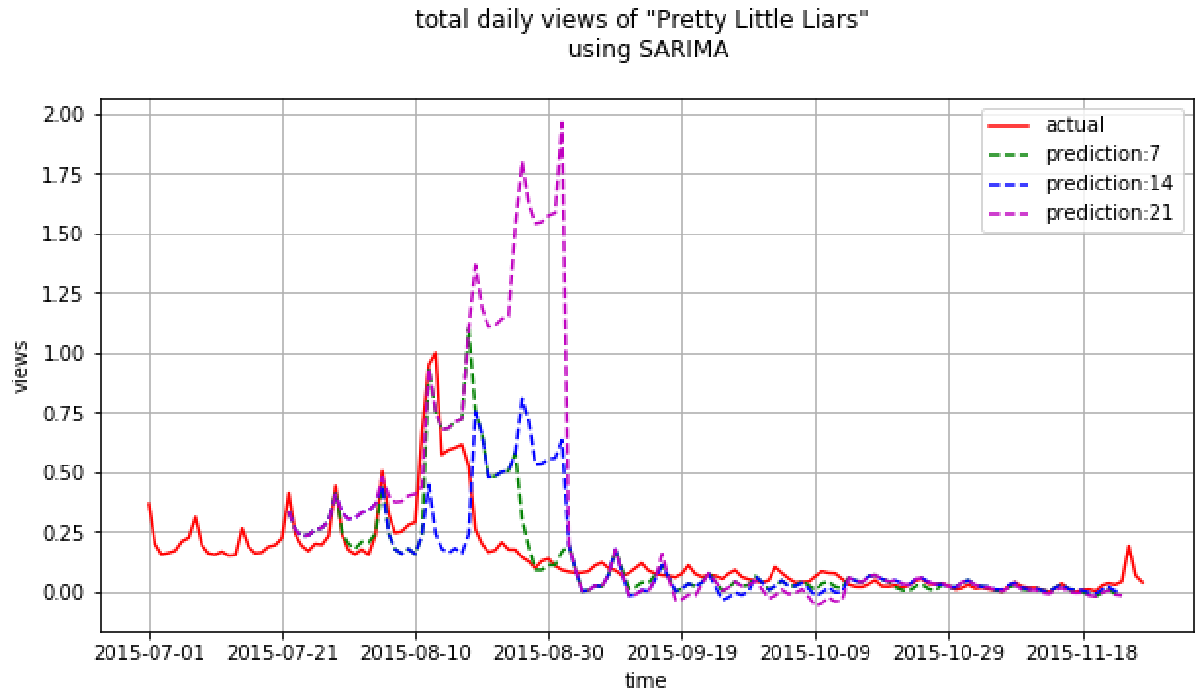
```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [57]:

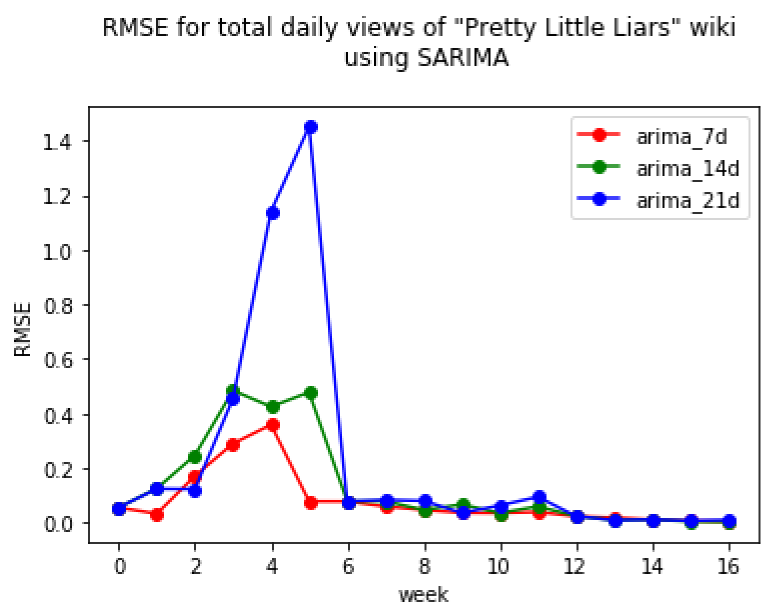
```
predict21 = ARIMAStepWisePrediction(dfsn_pll,21)
predict21.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [40]:
```
dfsn_pll = pd.read_csv('.\data\pll\dfsn_pll.csv',header=None,names=['date', '\
predict7 = pd.read_csv('.\data\pll\predict7.csv',header=None,names=['date', '\
predict14 = pd.read_csv('.\data\pll\predict14.csv',header=None,names=['date',
predict21 = pd.read_csv('.\data\pll\predict21.csv', header=None,names=['date',

PlotARIMAPrediction(dfsn_pll, "Pretty Little Liars")
```

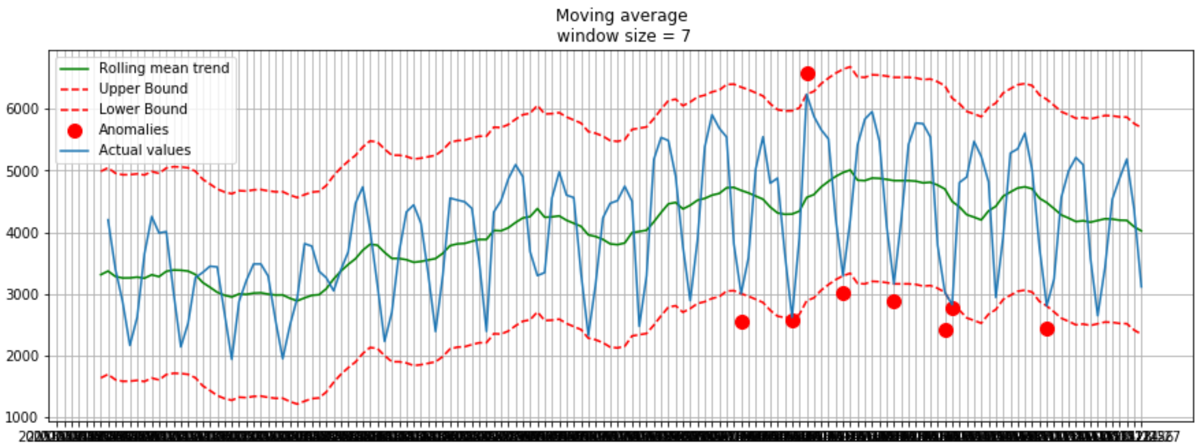


In [41]:
```
PlotARIMA_RMSE(dfsn_pll, predict7, predict14, predict21, "Pretty Little Liars'
```



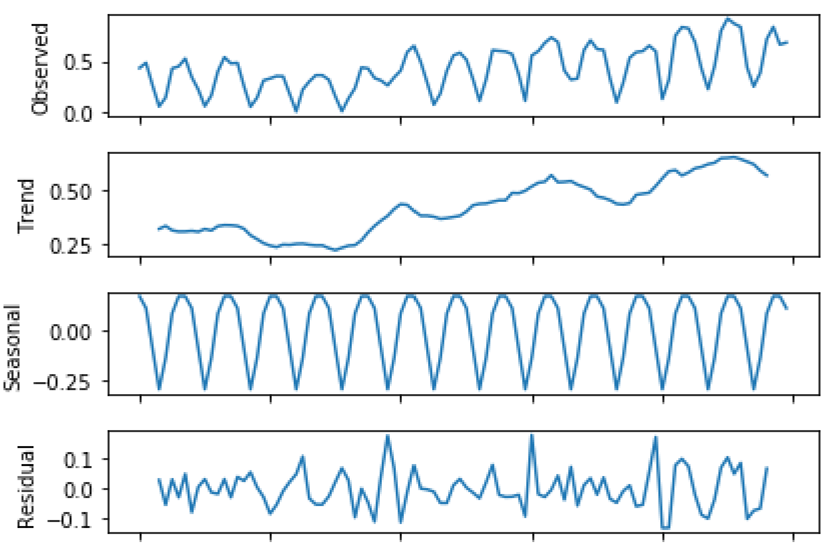
4.4. William Shakespeare

In [17]:
```
dfsn_ws = RemoveOutliers(dfs.iloc[:150,95], 7, plot_intervals=True, plot_anoma
dfsn_ws.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\w
```



In [25]:
```
plt.figure(figsize=(500, 500))
sm.tsa.seasonal_decompose(dfsn_ws[:100], freq=7).plot()
plt.show()
```

<Figure size 36000x36000 with 0 Axes>



In [57]:

```
predict7 = ARIMAStepWisePrediction(dfsn_ws,7)
predict7.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data\
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [57]:

```
predict14 = ARIMAStepWisePrediction(dfsn_ws,14)
predict14.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

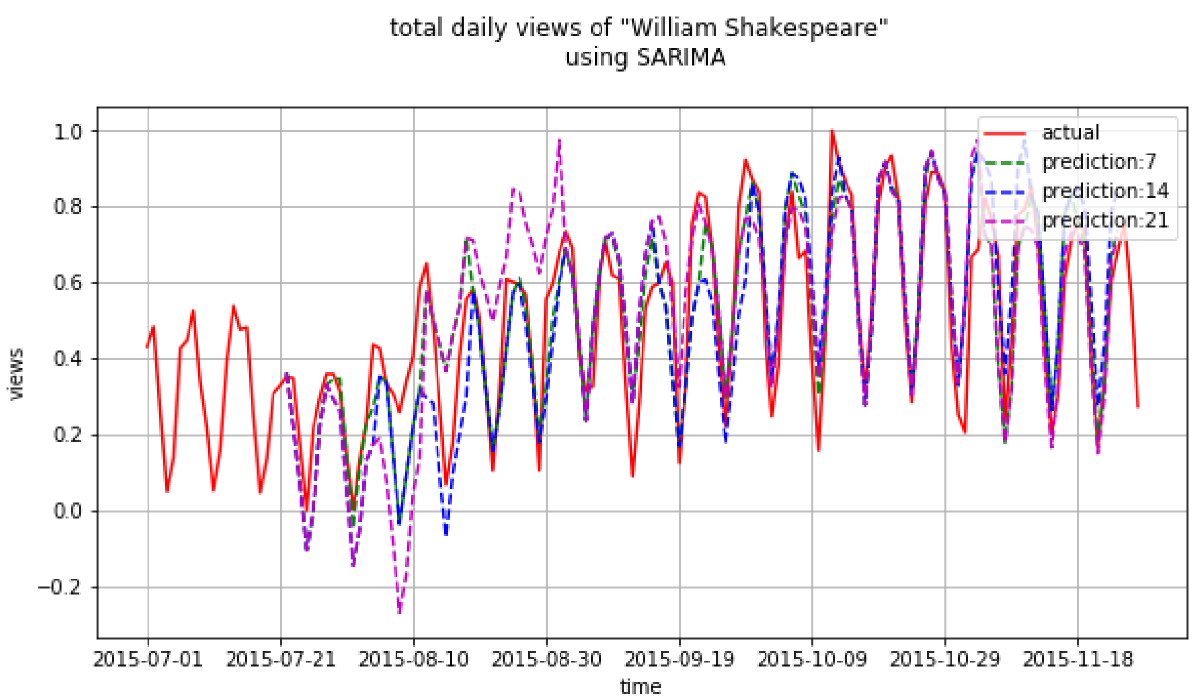
In [57]:

```
predict21 = ARIMAStepWisePrediction(dfsn_ws,21)
predict21.to_csv('C:\ML Projects\WebTrafficPrediction\FinalCodeforWebsite\data
```

```
0 21
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-12.070, BIC=-1
0.940, Fit time=0.075 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-8.285, BIC=-6.0
26, Fit time=0.452 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 0, 7); AIC=-10.090, BIC=-8.
395, Fit time=0.196 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(1, 1, 1, 7); AIC=nan, BIC=nan, Fi
t time=nan seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 1, 0, 7); AIC=-10.201, BIC=-8.
506, Fit time=0.118 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-10.252, BIC=-8.
557, Fit time=0.354 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1, 0, 7); AIC=-9.846, BIC=-7.5
87, Fit time=0.537 seconds
```

In [34]:

```
dfsn_ws = pd.read_csv('.\data\ws\dfsn_ws.csv',header=None,names=['date', 'view
predict7 = pd.read_csv('.\data\ws\predict7.csv',header=None,names=['date', 'vi
predict14 = pd.read_csv('.\data\ws\predict14.csv',header=None,names=['date', '
predict21 = pd.read_csv('.\data\ws\predict21.csv', header=None,names=['date',

PlotARIMAPrediction(dfsn_ws, "William Shakespeare")
```



In [35]:
```
1 PlotARIMA_RMSE(dfsn_ws, predict7, predict14, predict21, "William Shakespeare")
```