Walmart

December 11, 2018

1 Walmart Sales Forecasting

Yuha Yi University of Idaho Stats 517 - Statistical Learning and Predictive Modeling

Introduction The dataset contains historical sales data from 45 Walmart stores. Machine learning will be used to predict the sales of each stores using historical markdown data. Additionally, the effects of markdowns on certain holiday weeks that play a large part in weekly sales within the stores will be observed. Some important holidays include the super bowl, Christmas, and Thanksgiving/Black Friday to name a few. The research of this data observes how Walmart properly prepares during critical times of the year to maximize profit while achieving maximum customer satisfaction.

Variable Description The features (8191 x 12) file has some of the variables below Dept – the department number within the store Date – the week IsHoliday – whether the week is a special holiday week Temperature – The average temperature in the region Fuel_Price – Cost of fuel in the region CPI – The consumer price index Unemployment – The unemployment rate MarkDown(1-5) – Anonymized data related to promotional price markdowns from Walmart. Train.csv (422k x 5) contains Historical training data from 5/2/2010 to 26/10/2011, containing columns for Store, Dept, Date, Weekly_Sales and IsHoliday Test.csv (115k x 4) contains test data for 2/11/2012 to 26/7/2013, containing columns for Store, Department, Date and IsHoliday Stores file (45 x 3) has the 45 stores, the store size, and store type where Store type is an arbitrary value for the store type constiting of type A, B, and C

1.0.1 Process

The problems that this project attempts to solve is a real world application that all businesses look for. For regression, we seek the predicted weekly sales. The usage of this information can be used to allocate the right number of workers on busy weeks, know how much purchase to product for the holidays, and estimate overall income for the stores. This let us know which markdown selection is best during which time of year. The first half of the analysis uses regression to answer these questions For unsupervised learning (done in R) clustering is done for the stores (143 weeks x 45 stores) and an arima model is used for forecasting. To do this, an autocorrelation is ran through the store matrix to measure (in distance) the similarity of each store into a time series to view the similarities of the stores. They are then clustered into four individual time series. Association was attempted in this project however it did not end up successfully running, this it is not shown. However, if done properly it has the ability to tell us information such as given a certain department and week, how likely is it to have a markdown sale. This can give valuable information on when to purchase an item for the best price.

In [1]:
```
#!pip install pandas_profiling
```

In [2]:
```
import numpy as np
import pandas as pd
import plotly
import math
from numpy import *
import gc
import warnings
import os

from scipy.misc import imread
from scipy import sparse
import scipy.stats as ss
import matplotlib.pylab as plt
import seaborn as sns
import pandas_profiling
import seaborn as sns
import plotly.tools as tls
import plotly.graph_objs as go

from datetime import datetime

import statsmodels.api as sm
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.decomposition import PCA as sklearnPCA
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.cluster import SpectralClustering
from sklearn.mixture import GMM
from sklearn.cluster import AgglomerativeClustering
```

```
Store                3.000000      5.000000      11.000000      22.00000
Dept                 4.000000      7.000000      18.000000      37.00000
Weekly_Sales        59.974500    291.097000    2079.650000    7612.03000
Size             39690.000000  39910.000000   93638.000000  140167.00000
Temperature         27.310000     33.980000      46.680000      62.09000
Fuel_Price           2.653000      2.720000       2.933000       3.45200
MarkDown1          149.190000    375.200000    2240.270000    5347.45000
MarkDown2            1.950000      6.980000      41.600000     192.00000
MarkDown3            0.650000      1.650000       5.080000      24.60000
MarkDown4           28.760000    108.710000     504.220000    1481.31000
MarkDown5          715.520000   1070.830000    1878.440000    3359.45000
CPI                126.496258    128.823806     132.022667     182.31878
Unemployment         5.326000      5.965000       6.891000       7.86600

                         P75            P90            P95            P99  \
Store              33.000000      40.000000      43.000000      45.000000
Dept               74.000000      92.000000      95.000000      98.000000
Weekly_Sales    20205.852500   42845.673000   61201.951000  106479.586000
Size           202505.000000  204184.000000  206302.000000  219622.000000
Temperature        74.280000      83.580000      87.270000      92.810000
Fuel_Price          3.738000       3.917000       4.029000       4.202000
MarkDown1        9210.900000   15282.470000   21801.350000   41524.030000
MarkDown2        1926.940000    8549.740000   16497.470000   50366.600000
MarkDown3         103.990000     400.090000    1059.900000   63143.290000
MarkDown4        3595.040000    7871.420000   12645.960000   35785.260000
MarkDown5        5563.800000    8337.700000   11269.240000   27754.230000
CPI               212.416993     219.444244     221.941558     225.473509
Unemployment        8.572000       9.816000      12.187000      14.180000

                         MAX
Store              45.000000
Dept               99.000000
Weekly_Sales   693099.360000
Size           219622.000000
Temperature       100.140000
Fuel_Price          4.468000
MarkDown1       88646.760000
MarkDown2      104519.540000
MarkDown3      141630.610000
MarkDown4       67474.850000
MarkDown5      108519.280000
CPI               227.232807
Unemployment       14.313000
```

Data Explanation: Four files are given as the following, features (8191 x 12), stores (45 x 3), test (115k x 4), and train (422k x 5). The stores file contains the store number, size, and type. The store number is a unique identifier to differentiate all the stores, the size variable is measured in square feet, and the type indicates an arbitrary value of the store type consisting of type A, B, or C. The training file contains the store number, department number, the date (which week it is), the weekly sales, and a true/false variable of whether or not the week is considered as a holiday. The test file is identical to the train file without the weekly sales. The weekly sales and individual department sales must be predicted for each week for this file. The features file contains the following variables:

Store - a unique identifier for every Walmart store Dept – the department number within the store Date – the week IsHoliday – whether the week is a special holiday week Temperature – The average temperature in the region Fuel_Price – Cost of fuel in the region CPI – The consumer price index Unemployment – The unemployment rate Weekly_Sales - Sales for the given department within it's respective store MarkDown(1-5) – Anonymized data related to promotional price markdowns from Walmart.

In [10]:
```
def cat_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.value_counts()],
                     index=['N', 'NMISS', 'ColumnsNames'])

cat_summary=train_cat.apply(lambda x: cat_summary(x))
cat_summary
```

Out[10]:
```
                                                          Date  \
N                                                       421570
NMISS                                                        0
ColumnsNames  2011-12-23    3027
2011-11-25    3021
2011-12-...


                                                          Type
N                                                       421570
NMISS                                                        0
ColumnsNames  A    215478
B    163495
C     42597
Name: Type...
```

In [11]:
```
#Similarly to above, but for the testing dataset
numeric_var_test=[key for key in dict(test.dtypes) if dict(test.dtypes)[key] in ['floa
cat_var_test=[key for key in dict(test.dtypes) if dict(test.dtypes)[key] in ['object']
# Train Numerical Data
test_num=test[numeric_var_test]

# Train Categorical Data
test_cat=test[cat_var_test]
print (numeric_var_test)
print (cat_var_test)
```

```
['Store', 'Dept', 'Size', 'Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3',
['Date', 'Type']
```

In [12]:
```python
num_summary=test_num.apply(lambda x: var_summary(x)).T
#num_summary.to_excel(writer,'Numeric_variable Summary',index=True)
num_summary.head()
```

Out[12]:

	N	NMISS	SUM	MEAN	MEDIAN
Store	115064.0	0.0	2.558817e+06	22.238207	22.000
Dept	115064.0	0.0	5.101883e+06	44.339524	37.000
Size	115064.0	0.0	1.570597e+10	136497.688921	140167.000
Temperature	115064.0	0.0	6.206760e+06	53.941804	54.470
Fuel_Price	115064.0	0.0	4.121070e+05	3.581546	3.606

	STD	VAR	MIN	P1	P5
Store	12.809930	1.640943e+02	1.000	1.000	3.000
Dept	30.656410	9.398155e+02	1.000	1.000	4.000
Size	61106.926438	3.734056e+09	34875.000	34875.000	39690.000
Temperature	18.724153	3.505939e+02	-7.290	11.440	23.980
Fuel_Price	0.239442	5.733244e-02	2.872	2.957	3.161

	P10	P25	P50	P75	P90
Store	5.000	11.000	22.000	33.000	40.000
Dept	7.000	18.000	37.000	74.000	92.000
Size	39910.000	93638.000	140167.000	202505.000	204184.000
Temperature	29.970	39.820	54.470	67.350	79.480
Fuel_Price	3.227	3.431	3.606	3.766	3.866

	P95	P99	MAX
Store	43.000	45.000	45.000
Dept	95.000	98.000	99.000
Size	206302.000	219622.000	219622.000
Temperature	83.820	92.140	101.950
Fuel_Price	3.951	4.079	4.125

In [13]:
```python
pandas_profiling.ProfileReport(train)
```

Out[13]:
```
<pandas_profiling.ProfileReport at 0x2540a54a1d0>
```

In [14]:
```python
df = pd.concat([train,test],axis=0) # Join train and test
```

```
C:\Users\yiyuh\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning:

Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False
```

In [15]: df.describe()

Out[15]:

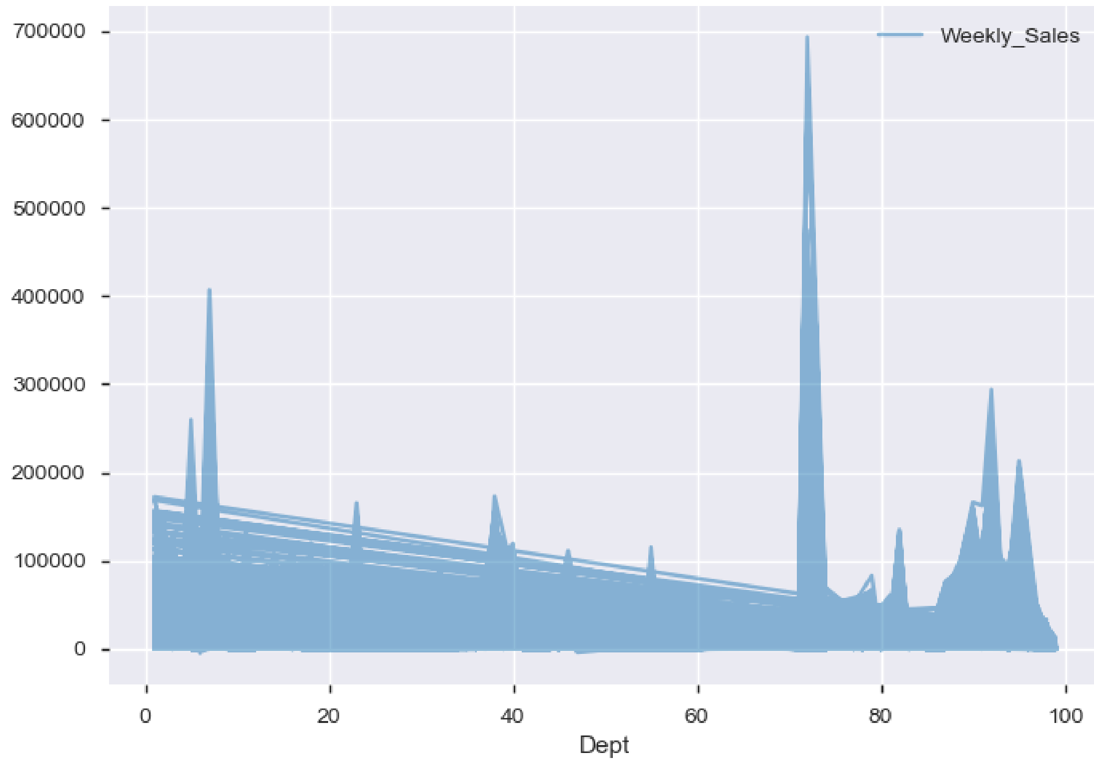
```
                CPI           Dept     Fuel_Price      MarkDown1  \
count  498472.000000  536634.000000  536634.000000  265596.000000
mean      172.090481      44.277301       3.408310    7438.004144
std        39.542149      30.527358       0.430861    9411.341379
min       126.064000       1.000000       2.472000   -2781.450000
25%       132.521867      18.000000       3.041000    2114.640000
50%       182.442420      37.000000       3.523000    5126.540000
75%       213.748126      74.000000       3.744000    9303.850000
max       228.976456      99.000000       4.468000  103184.980000

           MarkDown2      MarkDown3      MarkDown4      MarkDown5  \
count  197685.000000  242326.000000  237143.000000  266496.000000
mean     3509.274827    1857.913525    3371.556866    4324.021158
std      8992.047197   11616.143274    6872.281734   13549.262124
min      -265.760000    -179.260000       0.220000    -185.170000
25%        72.500000       7.220000     336.240000    1570.112500
50%       385.310000      40.760000    1239.040000    2870.910000
75%      2392.390000     174.260000    3397.080000    5012.220000
max    104519.540000  149483.310000   67474.850000  771448.100000

                Size          Store    Temperature   Unemployment  \
count  536634.000000  536634.000000  536634.000000  498472.000000
mean   136678.550960      22.208621      58.771762       7.791888
std     61007.711799      12.790580      18.678716       1.865076
min     34875.000000       1.000000      -7.290000       3.684000
25%     93638.000000      11.000000      45.250000       6.623000
50%    140167.000000      22.000000      60.060000       7.795000
75%    202505.000000      33.000000      73.230000       8.549000
max    219622.000000      45.000000     101.950000      14.313000

        Weekly_Sales
count  421570.000000
mean    15981.258123
std     22711.183519
min     -4988.940000
25%      2079.650000
50%      7612.030000
75%     20205.852500
max    693099.360000
```

In [16]:

```
train_corr=pd.DataFrame(train.corr())
train_corr.to_excel(writer,'Train_Data Corr',index=True)
train_corr.head()
```

Out[16]:

```
                 Store      Dept  Weekly_Sales  IsHoliday      Size  \
Store         1.000000  0.024004     -0.085195  -0.000548 -0.182881
```



Data Exploration: The first variable to note is the weekly sales. The average is at 15981 in US dollars. There has been a week where they lost profit with the weekly sale value at -4988.9. Despite most items being at a lower price when there is a holiday within the week, sales are still higher on average when it is a holiday. It seems to be that Wal-Mart is efficient in marking down the right products at the perfect amount while still making profit. That observation seems a whole different project. The maximum sales are from black Friday and Christmas. Store types A and B are similar, however type C has significantly lower amount of weekly sales. The correlation between the variables are tested using the pearson and spearmen methods, visualized with heatmaps below (Figure1, Figure2). The correlation between the size of the store and weekly sales shows to be promising (Figure 3). A general summary of the variables can be seen in the table below. More date variables are adjusted, creating new month variables and adding their holidays within the months. This helps generalize the time frame in which the sales performed in. To help with the projection of the sales for each department in each store, the markdown variables must be observed to understand which department is being marked down during the given holiday.

```
In [25]: print (train.isnull().sum())
         print ("******************************")
         print (test.isnull().sum())
```

```
Store                0
Dept                 0
Date                 0
```

Converting Categorical Variable 'IsHoliday' into Numerical Variable

```
In [38]: type_mapping = {False: 0, True: 1}
         for dataset in train_test_data:
             dataset['IsHoliday'] = dataset['IsHoliday'].map(type_mapping)
```

Creating Extra Holiday Variable. If that week comes under extra holiday then 1(=Yes) else 2(=No)

Making New Holiday Variable Based on Given Data

```
In [39]: train['Super_Bowl'] = np.where((train['Date']==datetime(2010, 2, 12)) | (train['Date'
         train['Labour_Day'] = np.where((train['Date']==datetime(2010, 9, 10)) | (train['Date'
         train['Thanksgiving'] = np.where((train['Date']==datetime(2010, 11, 26)) | (train['Da
         train['Christmas'] = np.where((train['Date']==datetime(2010, 12, 31)) | (train['Date'
         #.......................................................................
         test['Super_Bowl'] = np.where((test['Date']==datetime(2010, 2, 12)) | (test['Date']==
         test['Labour_Day'] = np.where((test['Date']==datetime(2010, 9, 10)) | (test['Date']==
         test['Thanksgiving'] = np.where((test['Date']==datetime(2010, 11, 26)) | (test['Date'
         test['Christmas'] = np.where((test['Date']==datetime(2010, 12, 31)) | (test['Date']==
```

Altering the isHoliday value depending on these new holidays

```
In [40]: train['IsHoliday']=train['IsHoliday']|train['Super_Bowl']|train['Labour_Day']|train['
         test['IsHoliday']=test['IsHoliday']|test['Super_Bowl']|test['Labour_Day']|test['Thanks

In [41]: print (train.Christmas.value_counts())
         print (train.Super_Bowl.value_counts())
         print (train.Thanksgiving.value_counts())
         print (train.Labour_Day.value_counts())
```

```
0    415624
1      5946
Name: Christmas, dtype: int64
0    412675
1      8895
Name: Super_Bowl, dtype: int64
0    415611
1      5959
Name: Thanksgiving, dtype: int64
0    412709
1      8861
Name: Labour_Day, dtype: int64
```

```
In [42]: print (test.Christmas.value_counts())
         print (test.Super_Bowl.value_counts())
         print (test.Thanksgiving.value_counts())
         print (test.Labour_Day.value_counts())
```

see a significant spike only within the two weeks before the super bowl event. A lagged variable for each of the holidays is needed to run an accurate model for this issue. In opposition, black Friday sales are only for a single day.

1.) Linear Regression

```
In [49]: clf = LinearRegression()
         clf.fit(train_X, train_y)
         y_pred_linear=clf.predict(test_X)
         acc_linear=round( clf.score(train_X, train_y) * 100, 2)
         print ('score:'+str(acc_linear) + ' percent')
```

```
score:8.86 percent
```

```
In [51]: #roc_auc_score(train_X, y_pred_linear)
```

2.) Random Forest

This algorithm takes a significant amount of time and may crash processing machine

```
In [52]: #clf = RandomForestRegressor(n_estimators=100)
         #clf.fit(train_X, train_y)
         #y_pred_rf=clf.predict(test_X)
         #acc_rf= round(clf.score(train_X, train_y) * 100, 2)
         #print ("Accuracy: %i %% \n"%acc_rf)
```

The result of Random Forest is at 99.77%

3.) Decision Tree

```
In [53]: clf=DecisionTreeRegressor()
         clf.fit(train_X, train_y)
         y_pred_dt= clf.predict(test_X)
         acc_dt = round( clf.score(train_X, train_y) * 100, 2)
         print (str(acc_dt) + ' percent')
```

```
100.0 percent
```

. . . . **Impressive** . . . . It's interesting to see how much of a difference the classifier makes. The Random forest and the decision tree approach 100 while the linear regression only gets an 8% accuracy. I believe the next step is to use unsupervised machine learning, clustering

Walmart Project

Yuha Yi

December 11, 2018

```
setwd("C:\\Users\\yiyuh\\Documents\\College\\Fall 2018\\Stat 517 - Machine Learning\\Final Project - Sta
#install.packages("reshape")
source('data_prep.R')
```

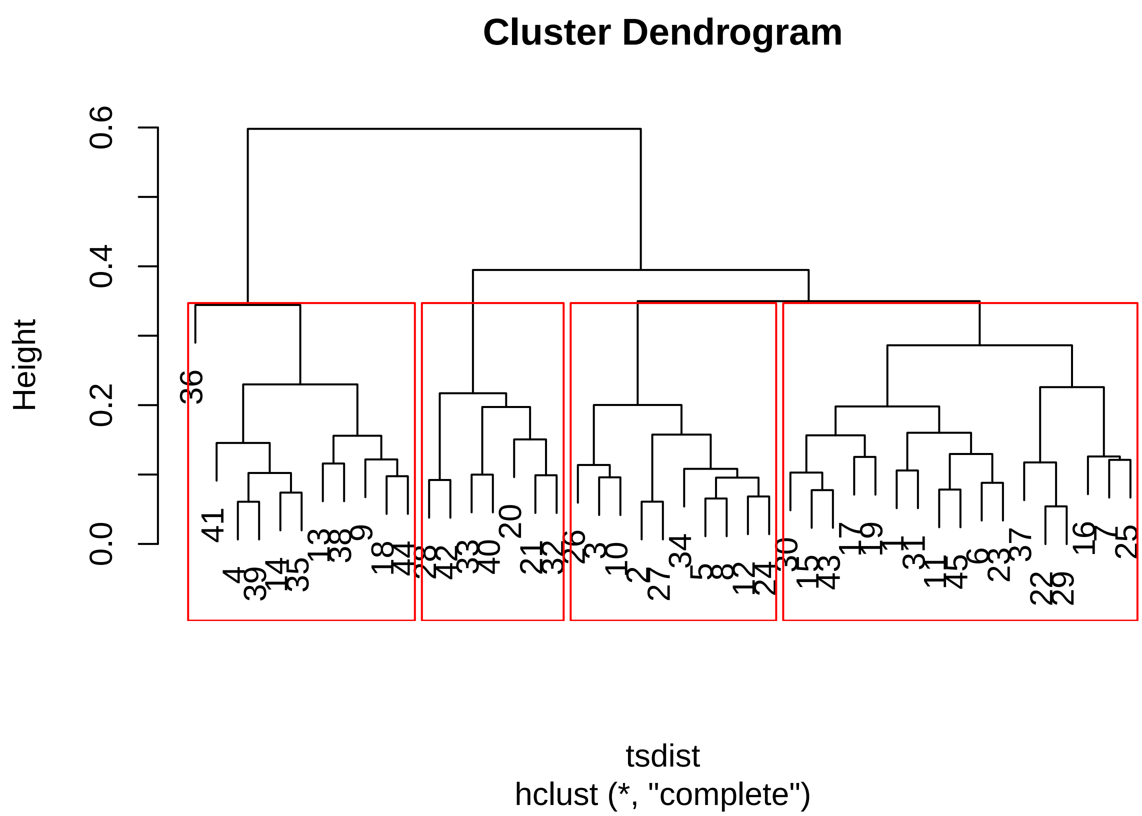
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
source('forecast.R')
```

```
##
## Attaching package: 'reshape'

## The following objects are masked from 'package:reshape2':
##
##     colsplit, melt, recast

## The following object is masked from 'package:dplyr':
##
##     rename

## Loading required package: wmtsa

## Loading required package: pdc

## Loading required package: cluster
```

```
source('clustering.R')
train <- read.csv("train.csv")
test <- read.csv("test.csv")
store.matrix <- reshape.by.stores(train)
```

```
#Perform and plot hierarchical clustering based on dissimilarity computation of weekly sales vs stores
tsdist<-calculate.ts.dist(store.matrix)
hc<-hclust(tsdist)
plot(hc)

#Upon visual inspection of the cluster plot, I decide to cluster the data into 4 clusters
rect.hclust(hc,k=4)
```




```
clust.vec <- cutree(hc,k=4)
clust.vec[hc$order]
```

```
## 36 41  4 39 14 35 13 38  9 18 44 28 42 33 40 20 21 32 26  3 10  2 27 34  5
##  3  3  3  3  3  3  3  3  3  3  3  4  4  4  4  4  4  4  2  2  2  2  2  2  2
##  8 12 24 30 15 43 17 19  1 31 11 45  6 23 37 22 29 16  7 25
##  2  2  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

```
#temp remove date column from store matrix
store.matrix.wodate <- store.matrix[,-1]
```

```
##Creating clusters
cluster1 <- store.matrix.wodate[,clust.vec==1]
cluster2 <- store.matrix.wodate[,clust.vec==2]
cluster3 <- store.matrix.wodate[,clust.vec==3]
cluster4 <- store.matrix.wodate[,clust.vec==4]

##Force clusters in a ts() object
cluster1.ts <-ts(rowMeans(cluster1),frequency=52)
cluster2.ts <-ts(rowMeans(cluster2),frequency=52)
cluster3.ts <-ts(rowMeans(cluster3),frequency=52)
cluster4.ts <-ts(rowMeans(cluster4),frequency=52)
```

```
### Time Series Forecasting
library(tseries)
#Test for stationarity by performing ADF test
adf.test(cluster1.ts, alternative='stationary') #Dickey-Fuller = -5.279, Lag order = 5, p-value = 0.01
```

```
## Warning in adf.test(cluster1.ts, alternative = "stationary"): p-value
## smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  cluster1.ts
## Dickey-Fuller = -5.279, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
adf.test(cluster2.ts, alternative='stationary') #Dickey-Fuller = -5.2943, Lag order = 5, p-value = 0.01
```

```
## Warning in adf.test(cluster2.ts, alternative = "stationary"): p-value
## smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  cluster2.ts
## Dickey-Fuller = -5.2943, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

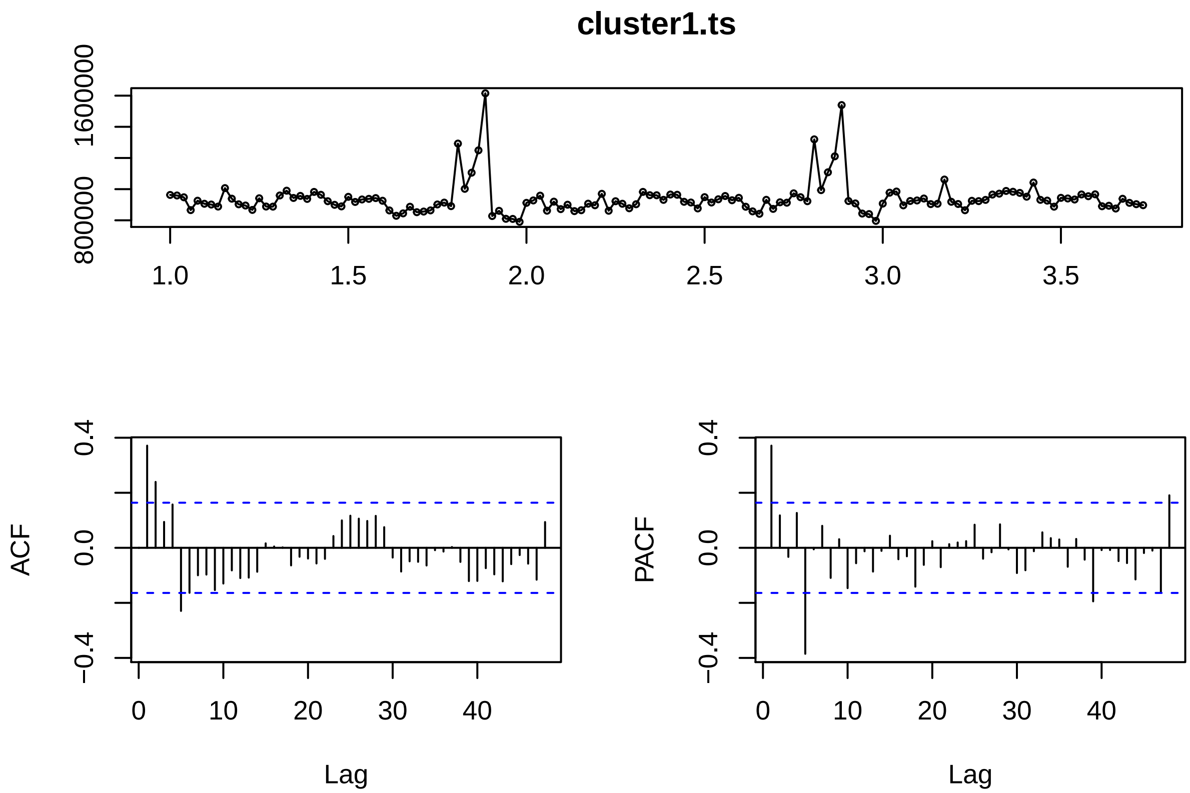
```
adf.test(cluster3.ts, alternative='stationary') #Dickey-Fuller = -5.3377, Lag order = 5, p-value = 0.01
```

```
## Warning in adf.test(cluster3.ts, alternative = "stationary"): p-value
## smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  cluster3.ts
## Dickey-Fuller = -5.3377, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

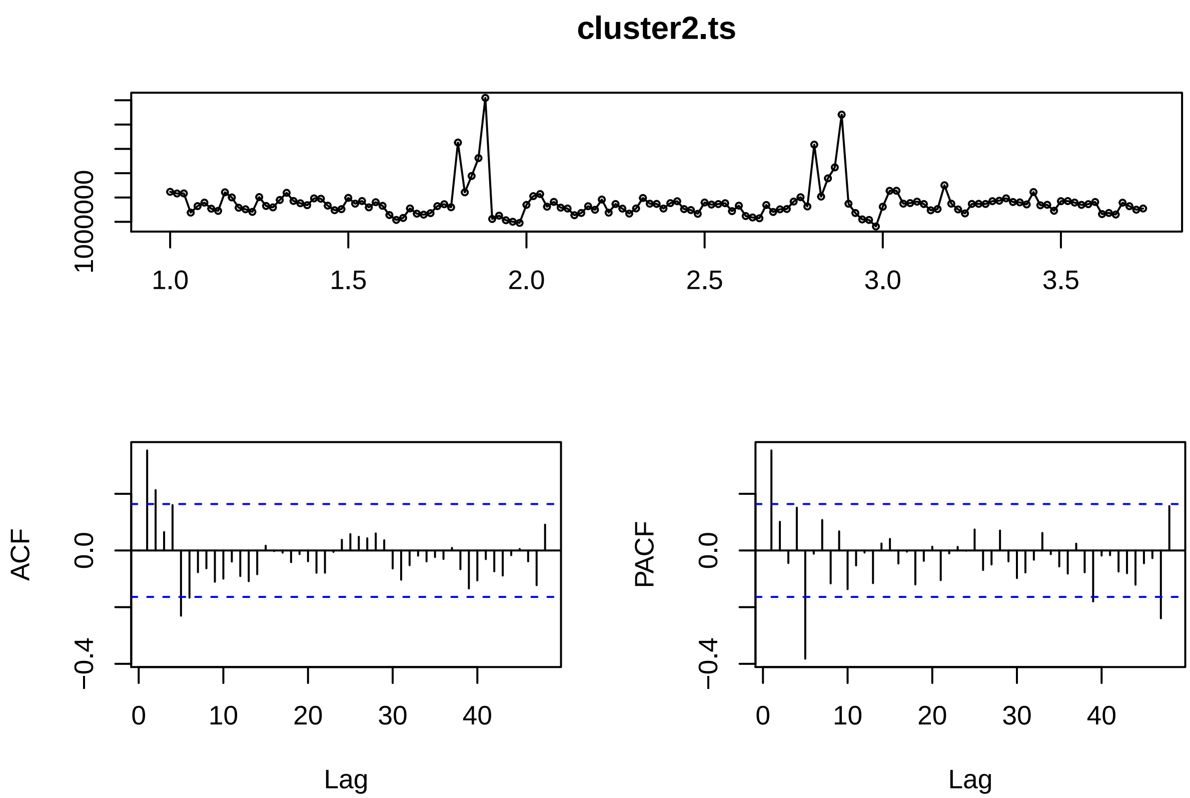
```
adf.test(cluster4.ts, alternative='stationary') #Dickey-Fuller = -5.1801, Lag order = 5, p-value = 0.01
```

```
## Warning in adf.test(cluster4.ts, alternative = "stationary"): p-value
## smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  cluster4.ts
## Dickey-Fuller = -5.1801, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

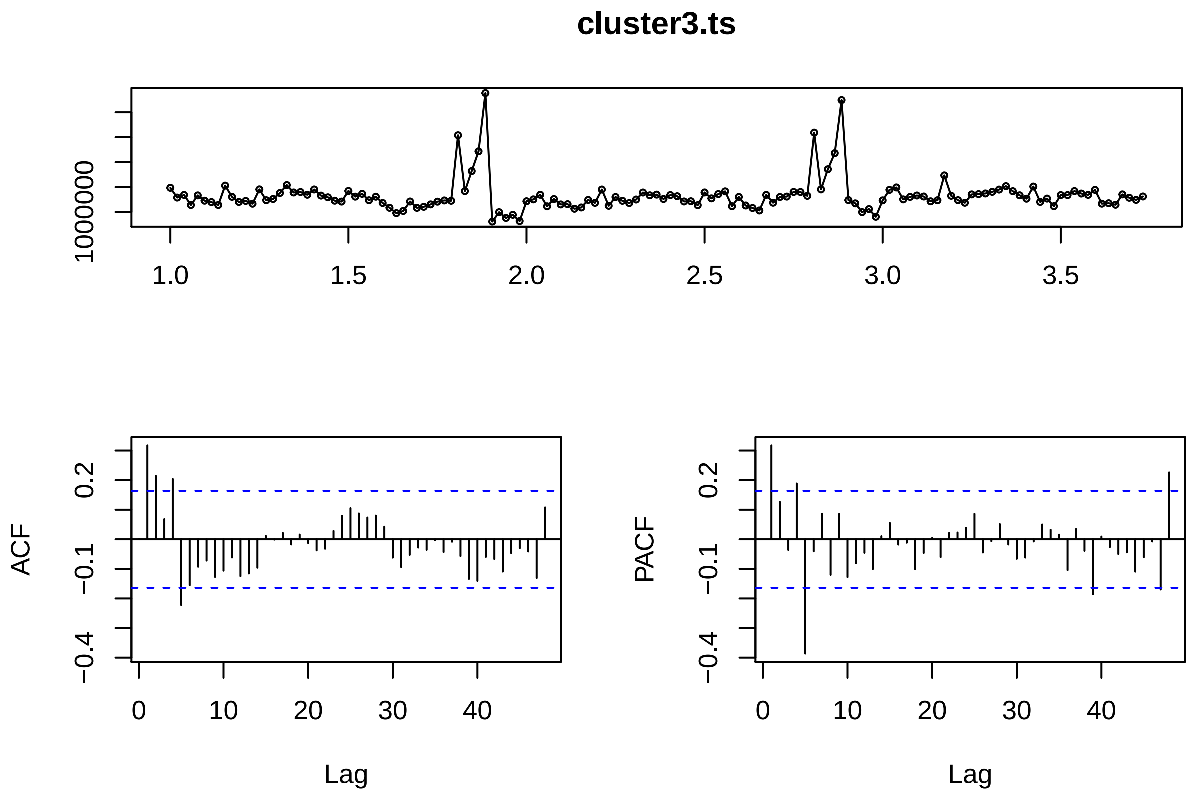
```
#To get an estimate coefficients for AR and MA, plot the ACF and PACF curve for each cluster
#The PACF and ACF lag orders which cross the confidence boundaries, are candidates for AR and MA coeffi
tsdisplay(cluster1.ts)
```



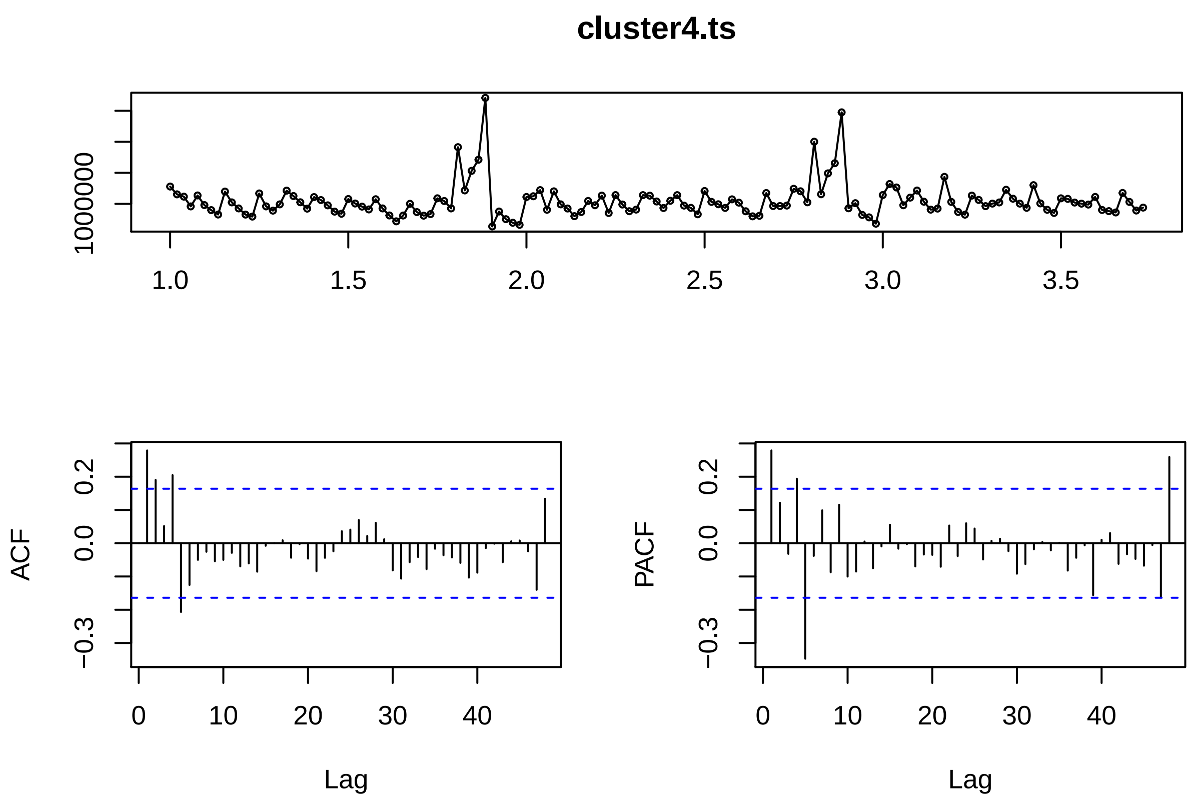

```
tsdisplay(cluster2.ts)
```




```r
tsdisplay(cluster3.ts)
```

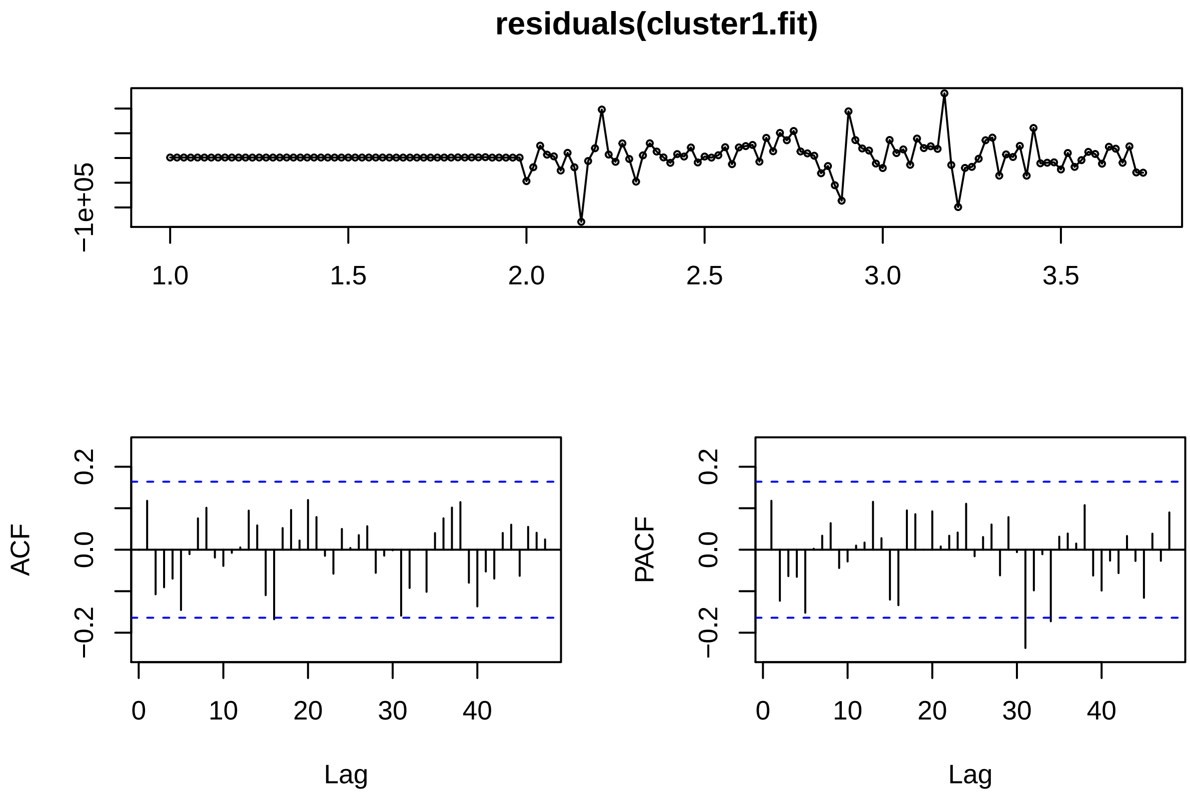



```
tsdisplay(cluster4.ts)
```

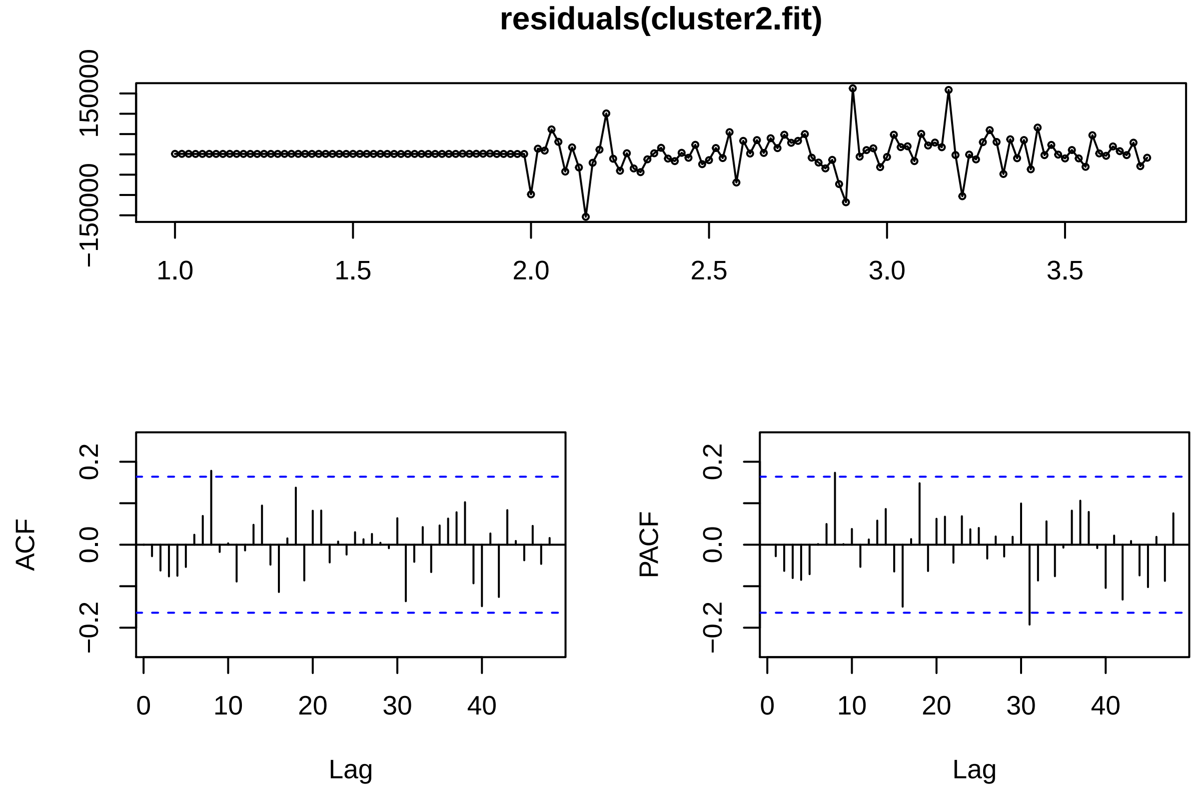



```r
#It is observed that all 4 clusters have a clear seasonal pattern for period length of 52 weeks.
#Hence, the seasonal order for ARIMA modeling will be defaulted to 'seasonal= list(order = c(0,1,0), pe
#To find the optimal pdq coeffecients for the trend component, run the following function for each clus


#manually try out combinations of p,d,q
cluster1.fit<-Arima(cluster1.ts,order=c(1,0,1), seasonal = list(order = c(0,1,0), period = 52), include
cluster2.fit<-Arima(cluster2.ts,order=c(1,0,2), seasonal = list(order = c(0,1,0), period = 52), include
cluster3.fit<-Arima(cluster3.ts,order=c(1,0,1), seasonal = list(order = c(0,1,0), period = 52), include
cluster4.fit<-Arima(cluster4.ts,order=c(1,0,1), seasonal = list(order = c(0,1,0), period = 52), include
```

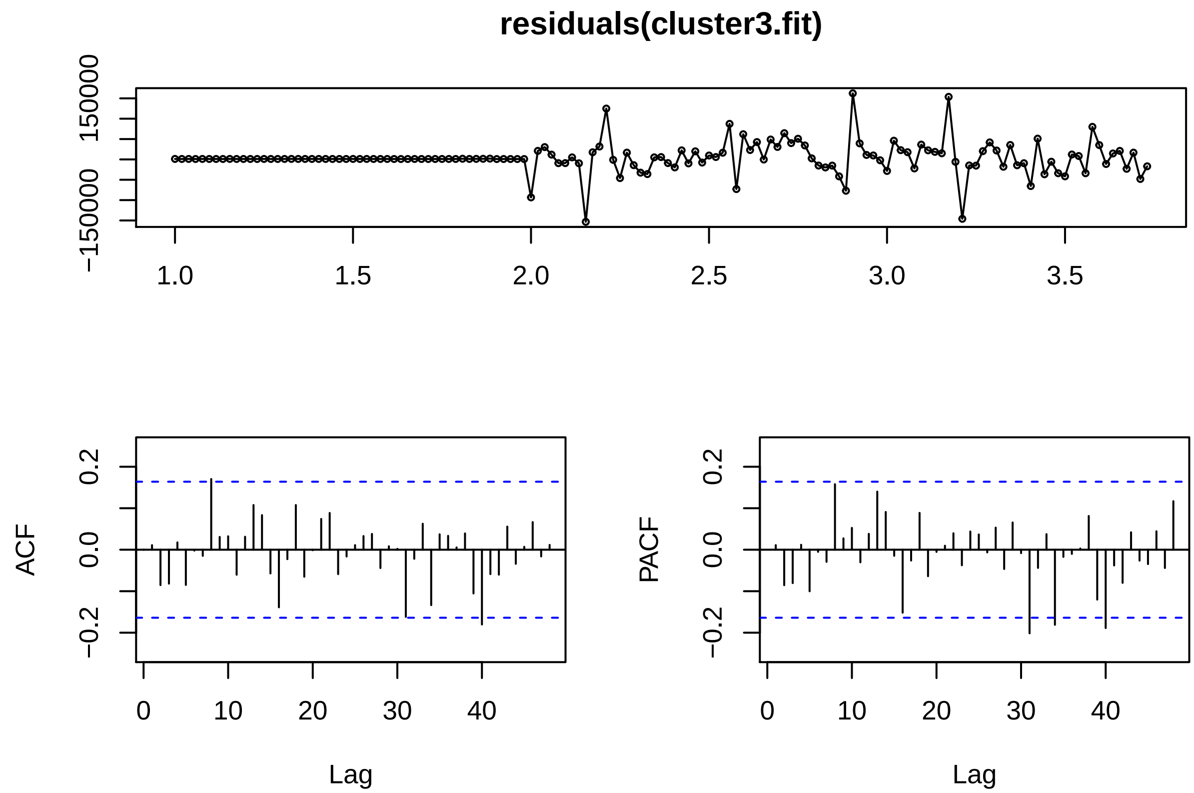
```r
### Evaluating forecast accuracy
#
#
# Visually check the fit of the arima model by plotting the ACF, PACF graph of the residuals
# Residuals which fall within the confidence boundaries suggest a good fit
tsdisplay(residuals(cluster1.fit))
```



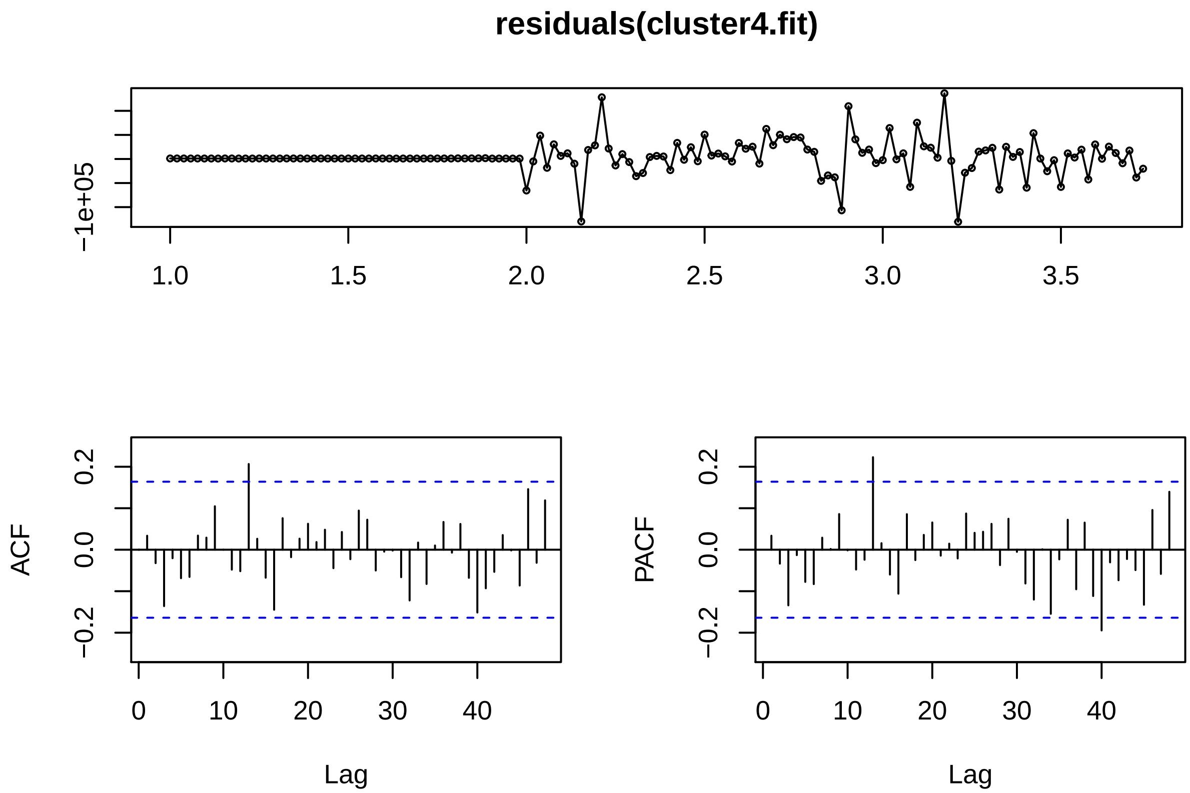

```r
tsdisplay(residuals(cluster2.fit))
```




```
tsdisplay(residuals(cluster3.fit))
```




```
tsdisplay(residuals(cluster4.fit))
```




```
#The mean absolute percentage error turns out to be
#5.837927 for cluster 1
#5.824512 for cluster 2
#5.570019 for cluster 3 and
#6.833386 for cluster 4
```