JANUARY 2014

Groupon

QUARTERLY REPORT
Q4 2013

CONTENTS

IN A WORLD OF TECHNOLOGY, PEOPLE MAKE THE DIFFERENCE

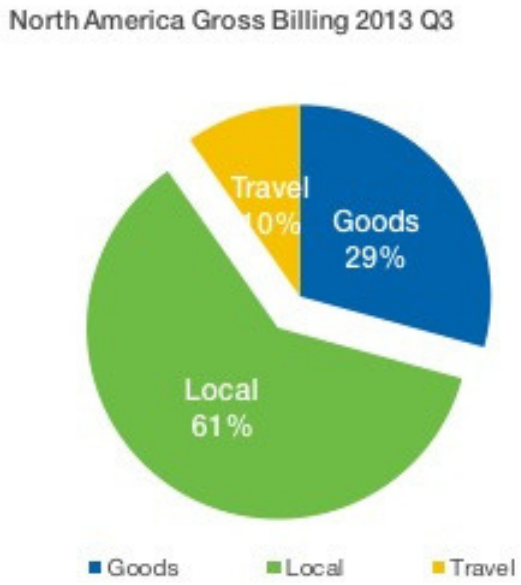
ABOUT THE COMPANY

OVERVIEW

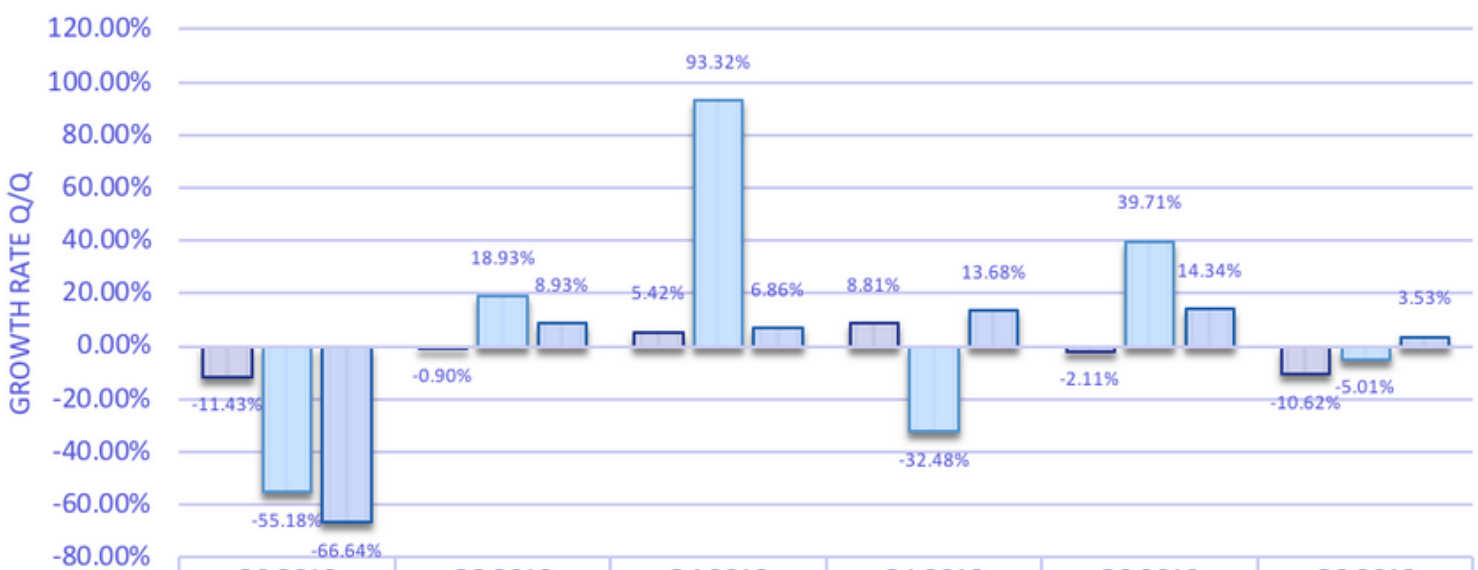
Groupon is a successful eCommerce platform and marketplace for deals, which has exhibited promised growth. Rising e-commerce spending on mobile devices boosts its profitability in the long run.

BUSINESS SEGMENTS

The company operates through three business segments, classified on geographic basis. However, Groupon has provided revenues based on categories. The company generates revenues through three categories: local (61% of the total billings in Q3 2013), goods (33%) and travel (13%)..



Financial Summary



	Q2 2012	Q3 2012	Q4 2012	Q1 2013	Q2 2013	Q3 2013
Local	-11.43%	-0.90%	5.42%	8.81%	-2.11%	-10.62%
Goods	-55.18%	18.93%	93.32%	-32.48%	39.71%	-5.01%
Travel	-66.64%	8.93%	6.86%	13.68%	14.34%	3.53%

For Q3 2013, North America, Groupon's largest geographic market, accounted for 50% of the total billings. Groupon recorded Gross Billings for North America of $665 million in Q3 2013, an increase of 20% over Q3 2012.

Based on the collected 2013 historical data set which represents Groupon's North America gross billings broken by segment: Local, Travel and Goods. The data gives us an insight for predicting the performance of Groupon's revenue in Q4 2013.

Recommendation: HOLD

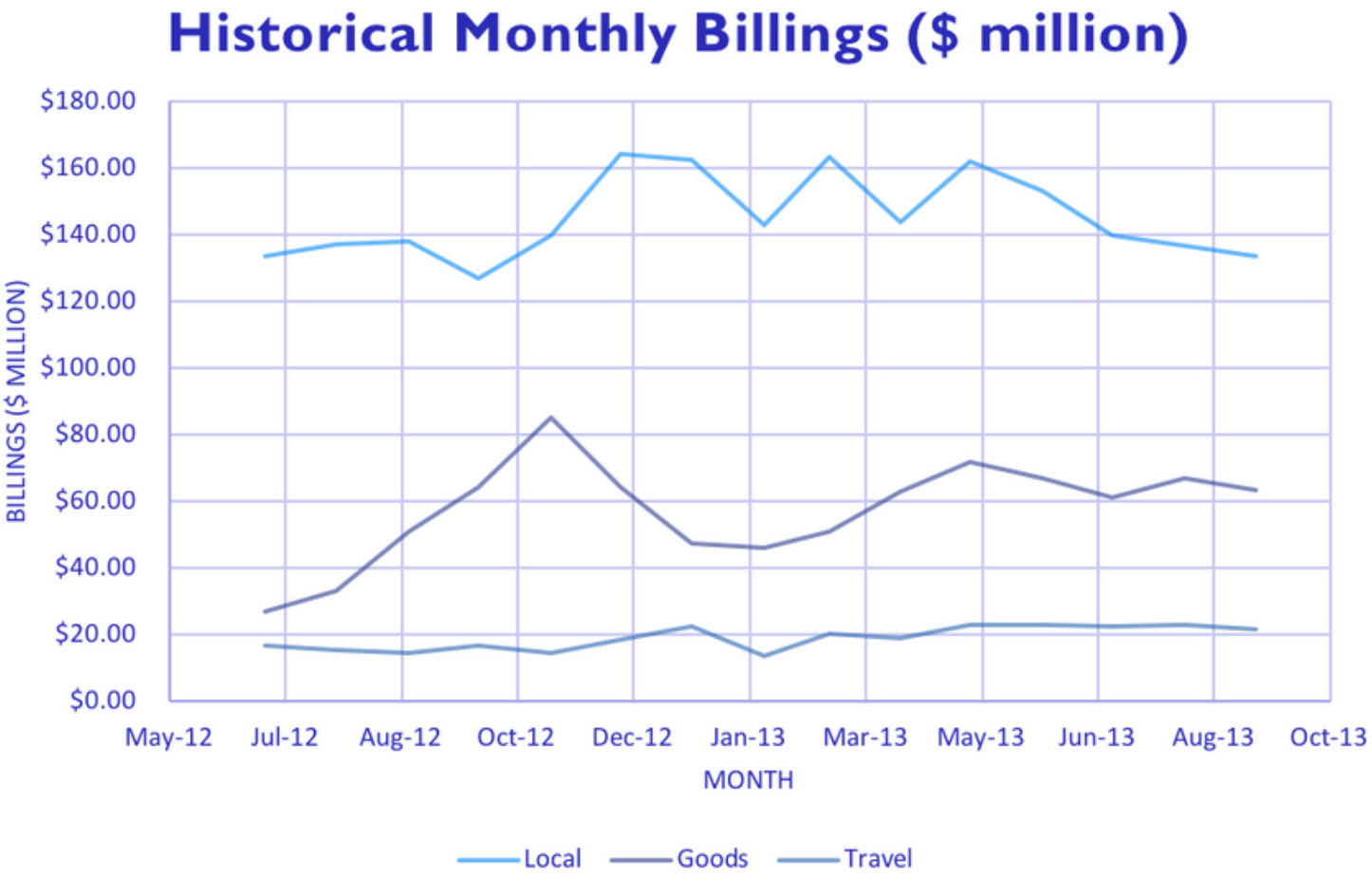
	Q3 2012	Q4 2012	Q1 2013	Q2 2013	Q3 2013	Q4 2013E
Local	$408.9	$431.1	$469.1	$459.2	$410.4	$450.8
Goods	$110.5	$213.7	$144.3	$201.6	$191.5	$282.2
Travel	$46.5	$49.7	$56.5	$64.6	$66.9	$70.6
Total	$565.9	$694.5	$669.9	$725.4	$668.8	$803.6

As shown in the table above, we could see that in the fourth quarter billings from North America is estimated to be $803.6 million, increased by 15.7% year over year. This shows the potential for growth, but it is not estimated to grow rapidly.

North America local gross billings came in at $450.8 million, rising by 4.5% year over year. Further, goods and local billings grew by 32.1% and 42.1% from the year-ago quarter, respectively. We could see billings were up compared to Q3 2012. However, compared to the past several quarters, there are still some uncertainties and fluctuations in the growth rates quarter to quarter with limited length of history for observation.

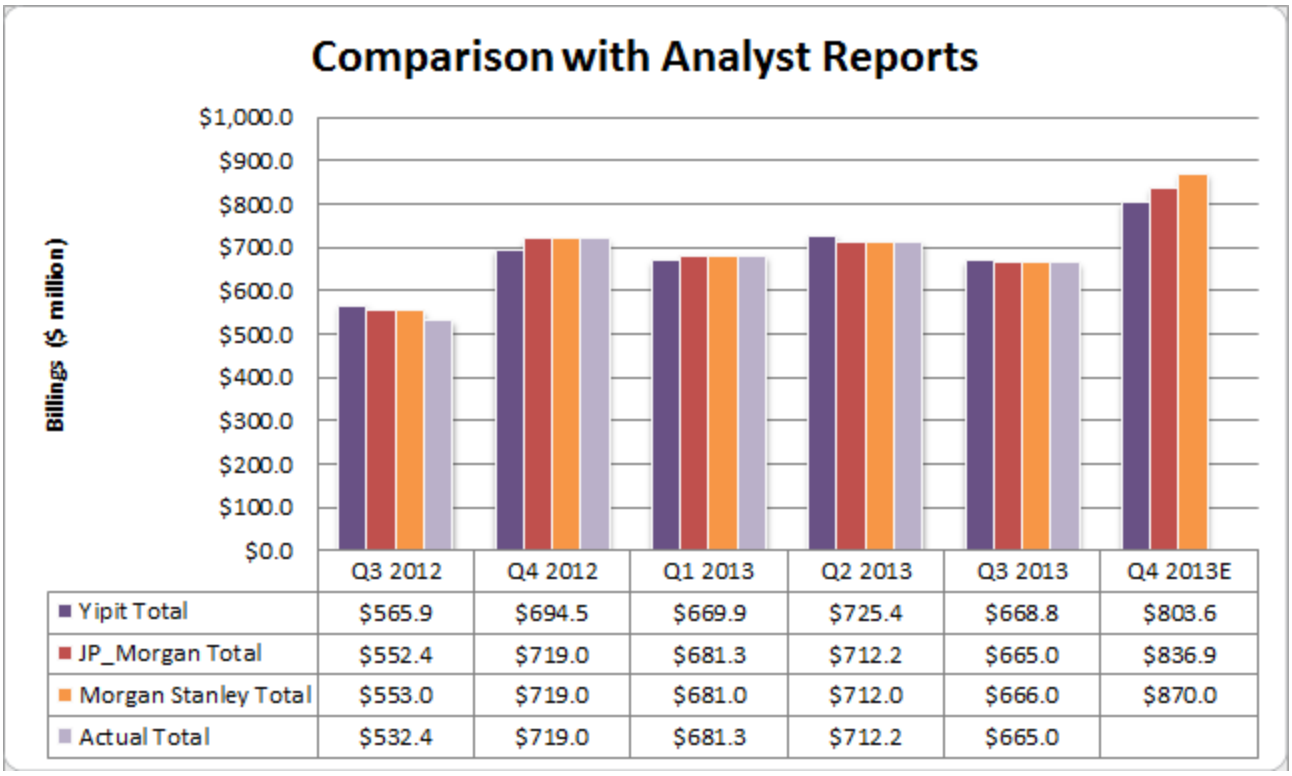
In addition, aggressive investments for mobile may drive operating costs higher. In addition, revenue growth has slowed down in the past years based on the trend shown in the history billings data. Uncertainties of the growth prospects may keep Groupon's pricing under tremendous pressure in the near term.
Although Groupon will expect positive results in the 2013 Q4, uncertainties of revenue growth and seasonality made my recommendation would stay Hold.

Seasonality



From the figure above, we could imply seasonality in sales. There are more sales in the second half of year where the last quarter usually is the peak of online sales. There are many important seasonal dates for online retail industry companies like Halloween, Thanksgiving, Black Friday and of course Christmas. These special days could drive people to spend more on shopping deals. Furthermore, there is actually a trend inside one week as well which I will state later in the next part.

Comparisons with analyst ratings



	Q3 2012	Q4 2012	Q1 2013	Q2 2013	Q3 2013	Q4 2013E
Yipit Total	$565.9	$694.5	$669.9	$725.4	$668.8	$803.6
JP_Morgan Total	$552.4	$719.0	$681.3	$712.2	$665.0	$836.9
Morgan Stanley Total	$553.0	$719.0	$681.0	$712.0	$666.0	$870.0
Actual Total	$532.4	$719.0	$681.3	$712.2	$665.0	

Most analysts were recommending this stock as a “buy”, with a few “hold” recommendations. There were higher estimates for 2013 Q4 billings for North America in their research reports which conclude overstated ratings for Groupon.

		Q3 2012	Q4 2012	Q1 2013	Q2 2013	Q3 2013	Q4 2013E
Yipit	Local	$408.9	$431.1	$469.1	$459.2	$410.4	$450.8
	Goods	$110.5	$213.7	$144.3	$201.6	$191.5	$282.2
	Travel	$46.5	$49.7	$56.5	$64.6	$66.9	$70.6
	Total	$565.9	$694.5	$669.9	$725.4	$668.8	$803.6
JP_Morgan	Local	$355.7	$430.3	$450.1	$450.5	$402.8	$490.1
	Goods	$152.1	$240.8	$165.4	$196.9	$194.6	$275.7
	Travel	$44.5	$47.9	$65.8	$64.9	$67.6	$71.0
	Total	$552.4	$719.0	$681.3	$712.2	$665.0	$836.9
Morgan Stanley	Local	$356.0	$430.0	$450.0	$450.0	$403.0	$508.0
	Goods	$152.0	$241.0	$165.0	$197.0	$195.0	$295.0
	Travel	$45.0	$48.0	$66.0	$65.0	$68.0	$67.0
	Total	$553.0	$719.0	$681.0	$712.0	$666.0	$870.0
Actual	Local	$335.7	$430.3	$450.1	$450.5	$402.8	
	Goods	$152.1	$240.8	$165.4	$196.9	$194.6	
	Travel	$44.5	$47.9	$65.8	$64.9	$67.6	
	Total	$532.4	$719.0	$681.3	$712.2	$665.0	

The data I used to estimate Q4 2013 was from YipitData estimates. They were for the most part the same as the actual data released from Groupon quarterly report shown above, except the actual amounts were always less than the estimates. In this case, my estimate would also be slightly above the actual amounts for billings.

Management indicated that the overall demand remains good and in line with its expectations, and that Q4 2013 should see positive seasonal tailwinds. Seasonality for the full year is playing as expected (4Q is the strongest quarter seasonally for Internet Commercial companies), with growth in Q3 expected to be stronger than that in the first half of the financial year FY2013.

The seasonality drove billings increase from Q3 2013 to Q4 2013, however the growth will be lower anticipated in Q1 2014. Overall, the billings estimates are following a bullish trend, but not as much growth is expected before.

Deal with Data

Reading table from excel file and pull the raw data of Q4 2013 using python package pandas:

```
import pandas as pd
```

```
df_raw = pd.read_excel('Q4_2013_Groupon_North_America_Data_XLSX.xlsx', 'Q4 2013 Raw Data')
```

```
df_raw.head()
```

	Deal ID	Units Sold	Billings	Start Date	Deal URL	Segment	Inventory Type
0	gr-millevois-tire-service-center	0.0	0.0	2011-11-21	http://www.groupon.com/deals/gr-millevois-tire...	Local	Third - Party
1	gr-manakeesh-cafe-bakery	0.0	0.0	2011-11-21	http://www.groupon.com/deals/gr-manakeesh-cafe...	Local	Third - Party
2	gr-phoenix-salon-and-spa	0.0	0.0	2011-11-21	http://www.groupon.com/deals/gr-phoenix-salon-...	Local	Third - Party
3	gr-hands-in-motion	0.0	0.0	2011-11-21	http://www.groupon.com/deals/gr-hands-in-motion	Local	Third - Party
4	dc-fd2-bartending-college-allentown-reading	86.8	4253.2	2012-06-06	http://www.groupon.com/deals/dc-fd2-bartending...	Local	Third - Party

Separated dataframe df_raw by segments: df_Local, df_Goods, df_Travel. One more step to check if there are duplicates exist.

```
df_Local = df_raw[df_raw['Segment'] == 'Local']
df_Goods = df_raw[df_raw['Segment'] == 'Goods']
df_Travel = df_raw[df_raw['Segment'] == 'Travel']
```

```
df_Local[df_Local.duplicated('Deal ID')]
```

Deal ID	Units Sold	Billings	Start Date	Deal URL	Segment	Inventory Type

```
df_Goods[df_Goods.duplicated('Deal ID')]
```

Deal ID	Units Sold	Billings	Start Date	Deal URL	Segment	Inventory Type

```
df_Travel[df_Travel.duplicated('Deal ID')]
```

Deal ID	Units Sold	Billings	Start Date	Deal URL	Segment	Inventory Type

Deal with Data

```
Local_billings = df_Local['Billings'].sum()
print(Local_billings)
```

```
409222657.982
```

```
Goods_billings = df_Goods['Billings'].sum()
print(Goods_billings)
```

```
282245671.04132
```

```
Travel_billings = df_Travel['Billings'].sum()
print(Travel_billings)
```

```
70552062.12449999
```

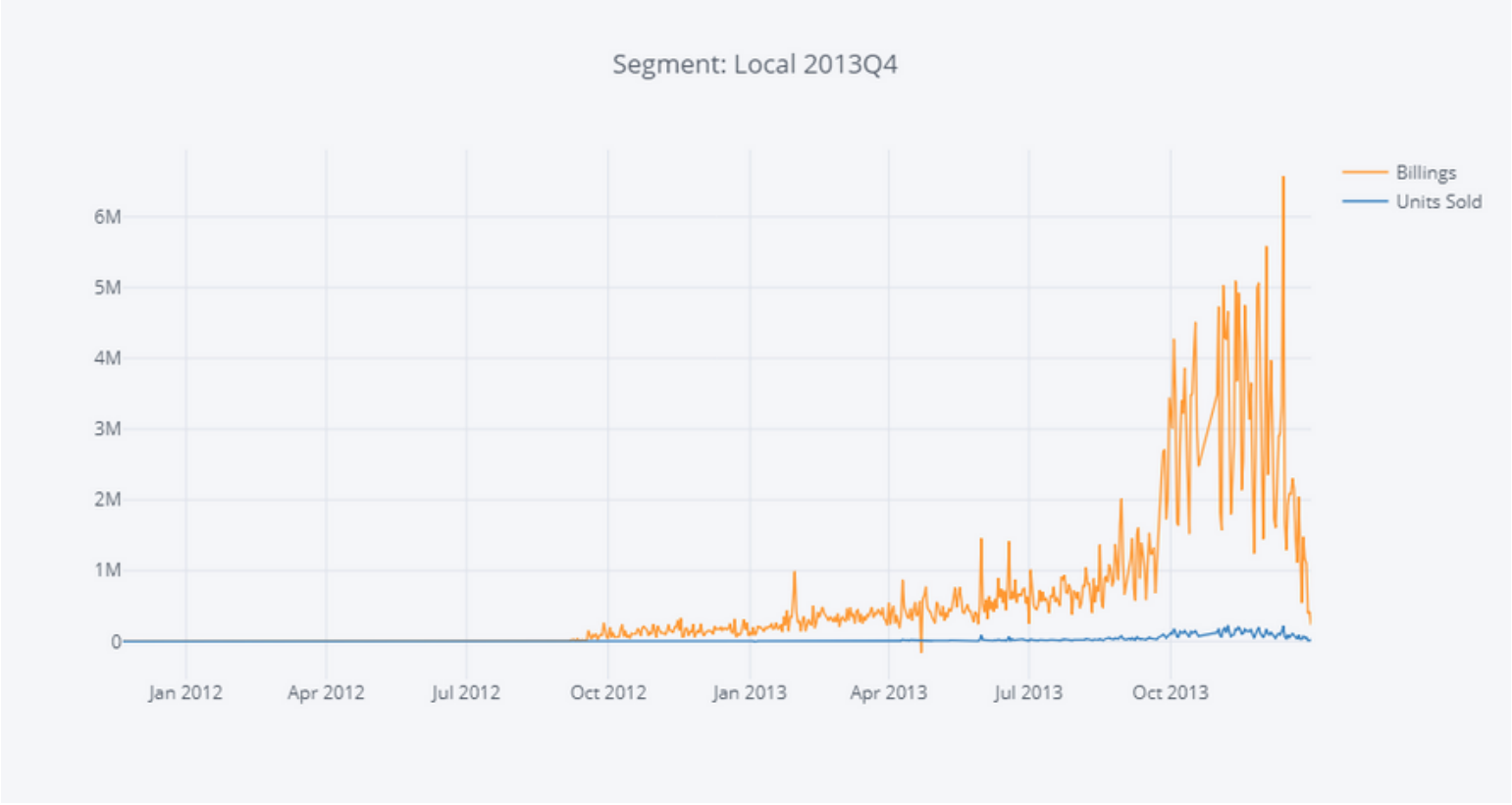
Then we could conclude by calculating sum values of each dataframe to get billings of each segment for Q4 2013. However, since Local billings data for the period from 2013-10-10 to 2013-10-20 is missing so we need to make some estimation for Local billings.

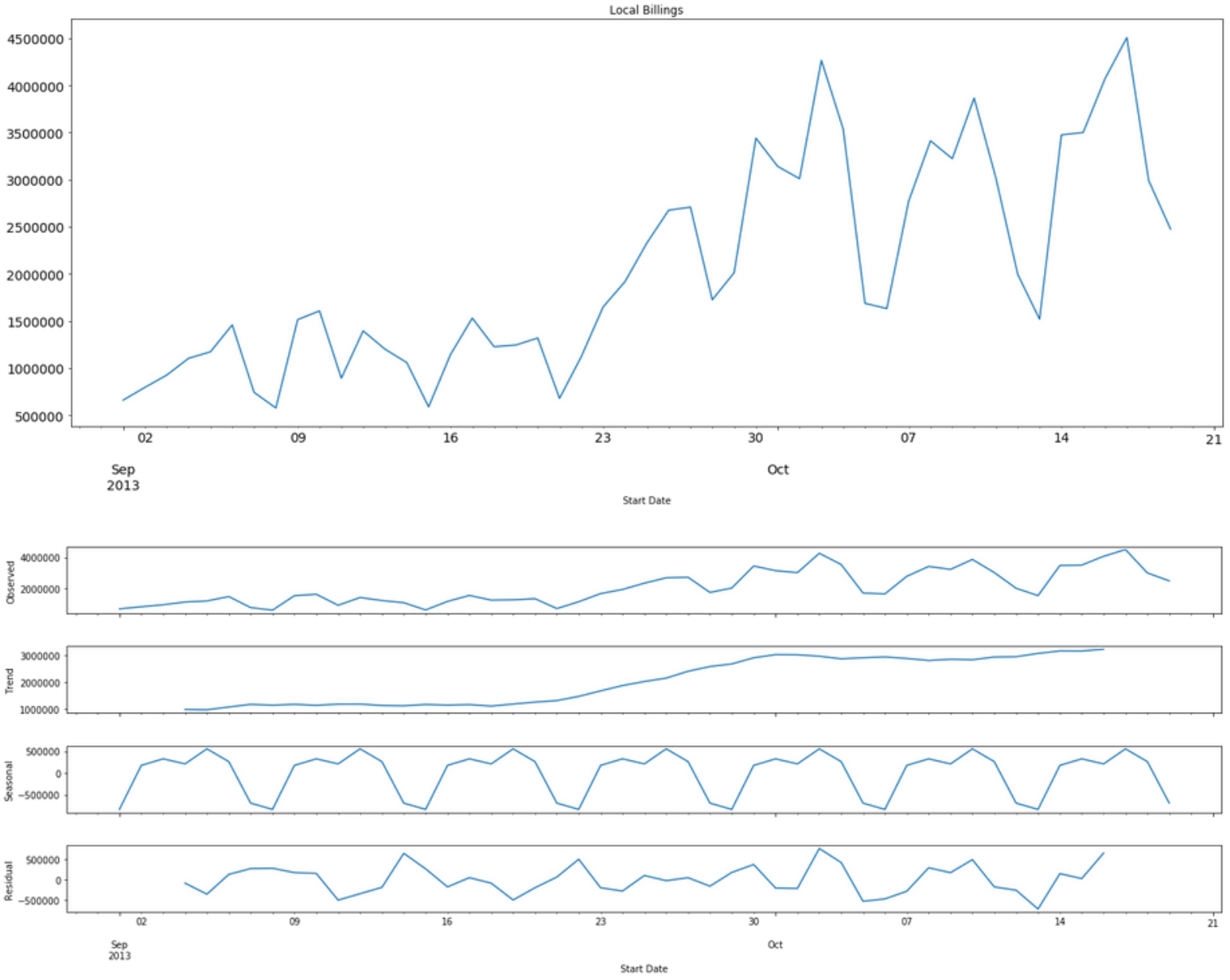
Estimate missing data

by creating seasonal ARIMA model using Python and Statsmodel

STEP 1
VISUALIZE DATA

As we visualize the local billings Q4 2013 data we can see there is both an upward trend in the data and there is seasonality to it.

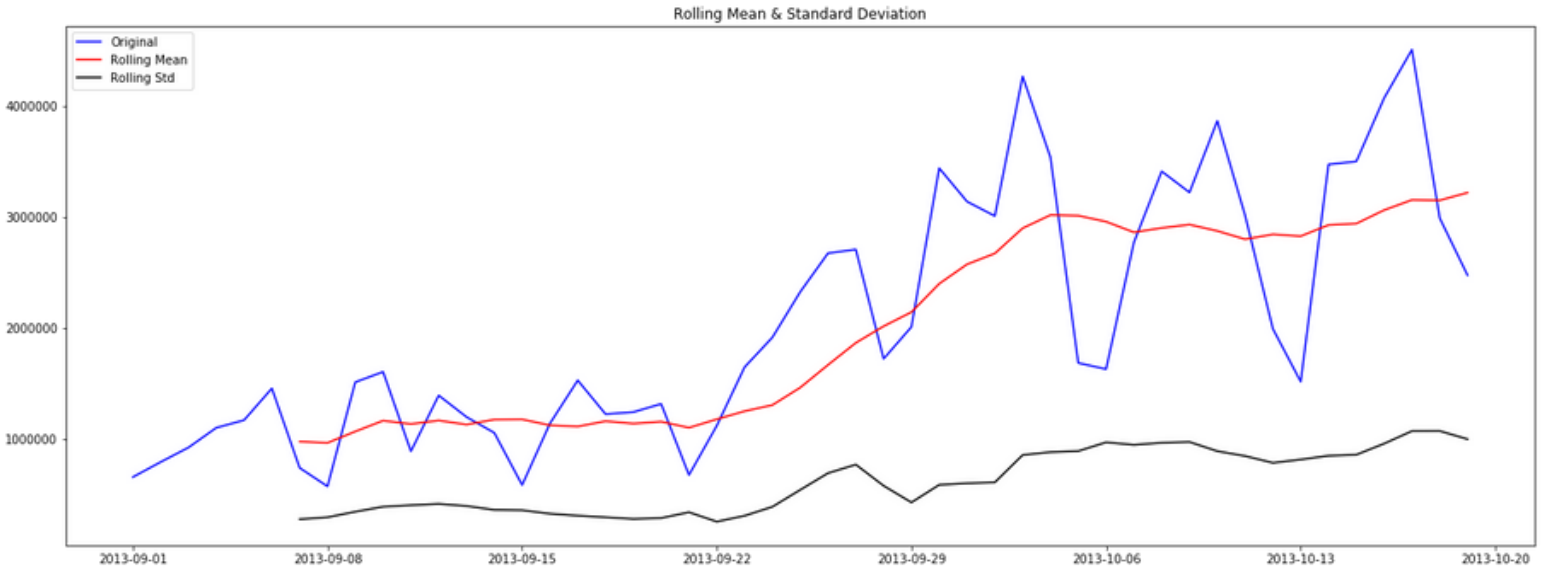




I choose neighbored period 2013-09-01 to 2013-12-15 to construct model for estimation for the missing data. When we decompose the series, the trend and seasonality become even more obvious.

STEP 2
STATIONARIZE THE DATA

To better examine the accuracy of stationary of data, I choose to use the Dickey-Fuller test.

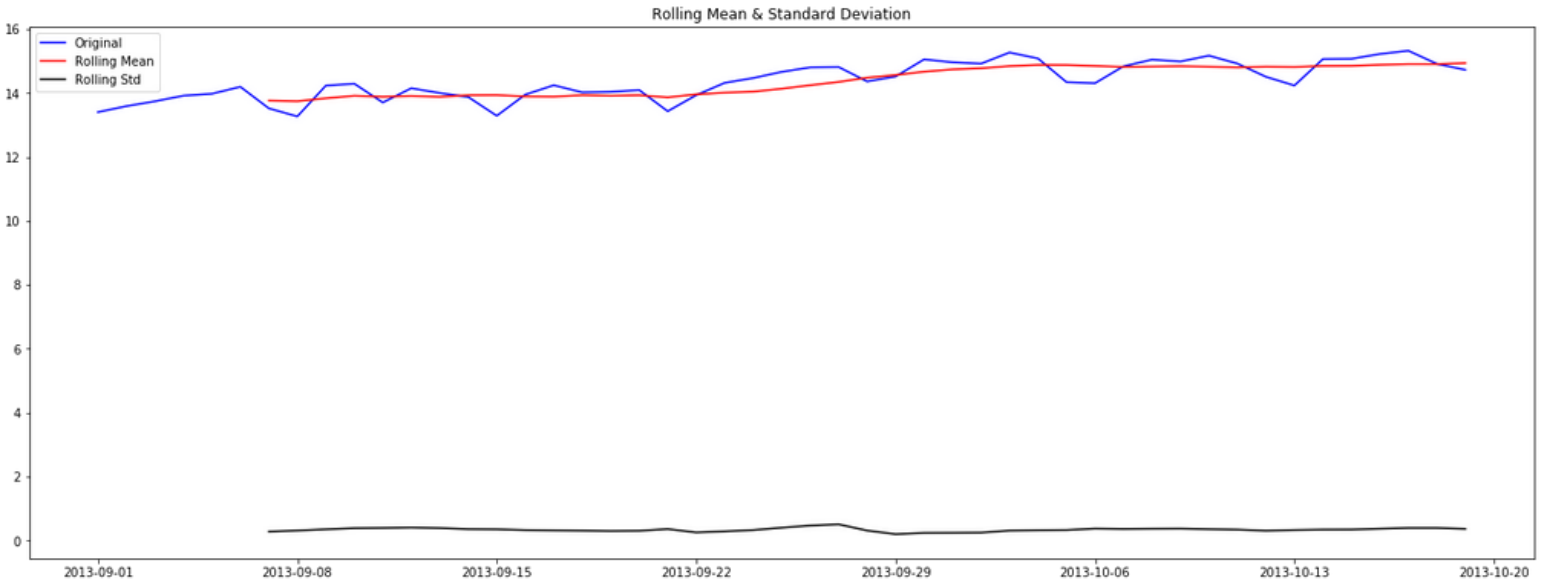


Results of Dickey-Fuller Test:
Test Statistic -0.493765
p-value 0.893227
#Lags Used 8.000000
Number of Observations Used 40.000000
Critical Value (1%) -3.605565
Critical Value (5%) -2.937069
Critical Value (10%) -2.606986
dtype: float64

We can easily see that the time series is not stationary, and our test_stationarity function confirms what we see.

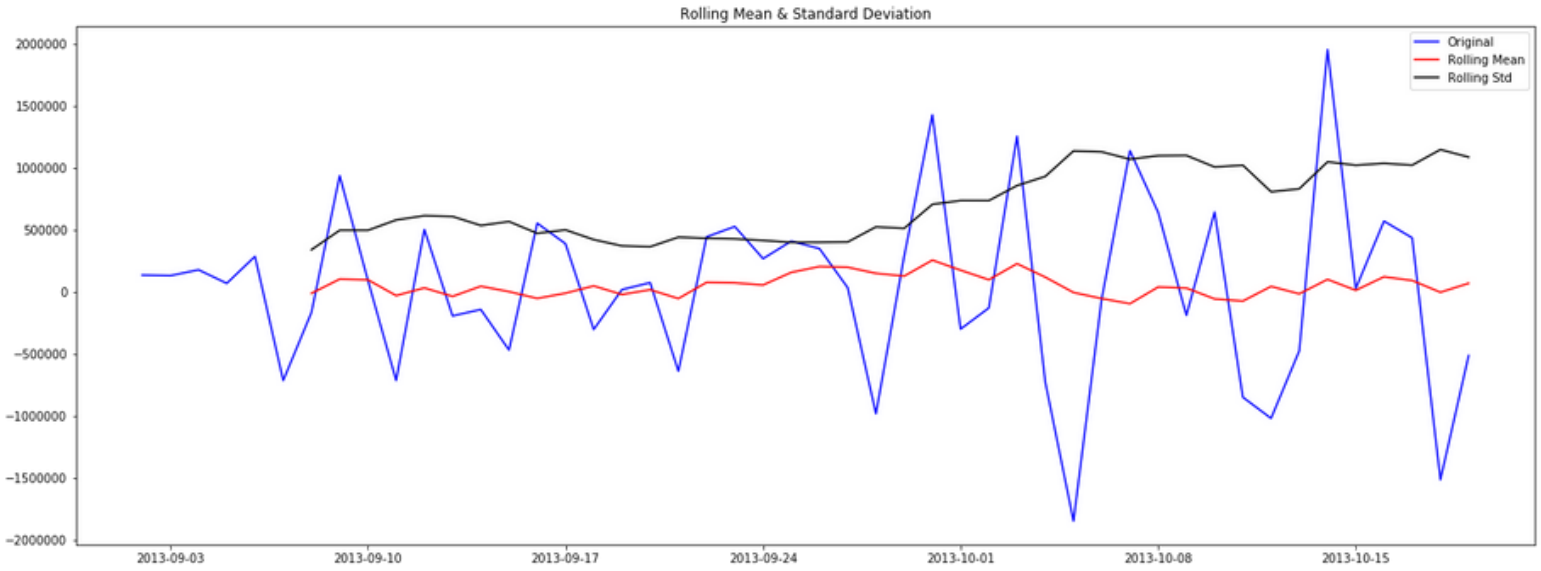
So now we need to transform the data.
The following are the next few steps I will do in order to make our series more stationary:
1. Logarithmic
2. First Difference
3. Seasonal Difference
4. Seasonal Adjustment

Take the log of the series and test for stationarity in python
data.Billings_log = data.Billings.apply(lambda x: np.log(x))
test_stationarity(data.Billings_log)



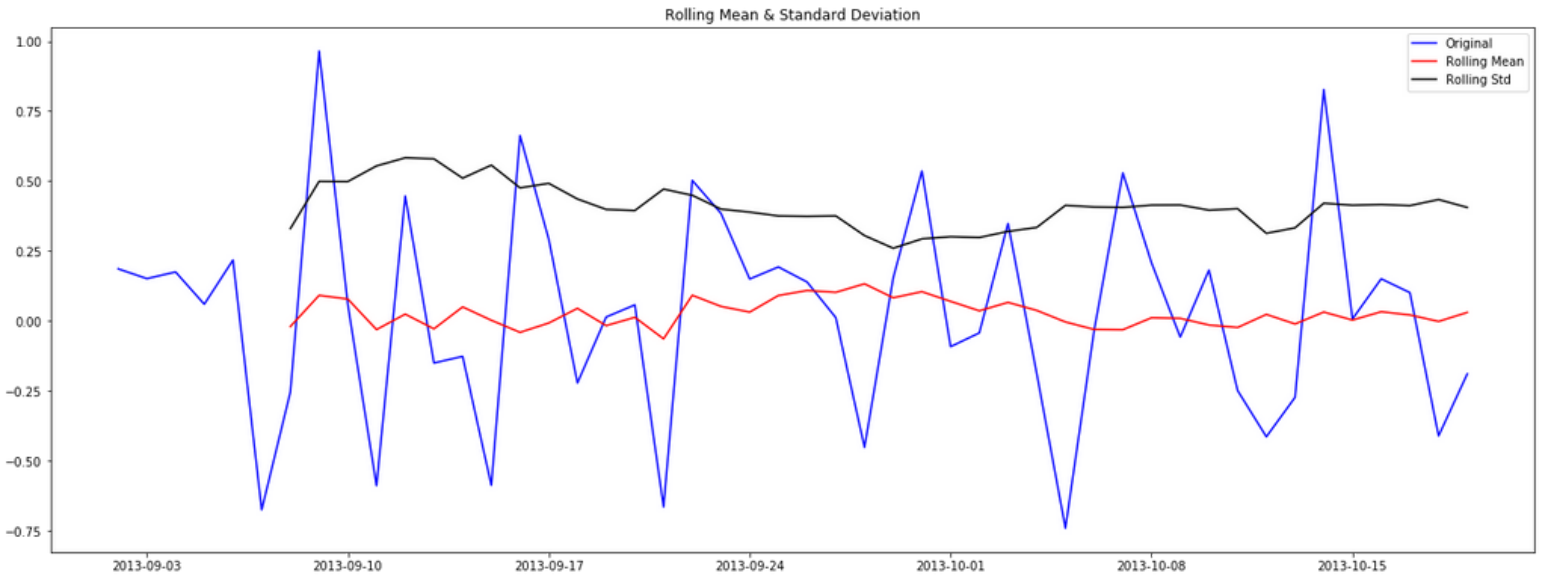
Results of Dickey-Fuller Test:
Test Statistic -0.676594
p-value 0.852720
#Lags Used 8.000000
Number of Observations Used 40.000000
Critical Value (1%) -3.605565
Critical Value (5%) -2.937069
Critical Value (10%) -2.606986
dtype: float64

Take first_difference
data['first_difference'] = data.Billings - data.Billings.shift(1)
test_stationarity(data.first_difference.dropna(inplace=False))



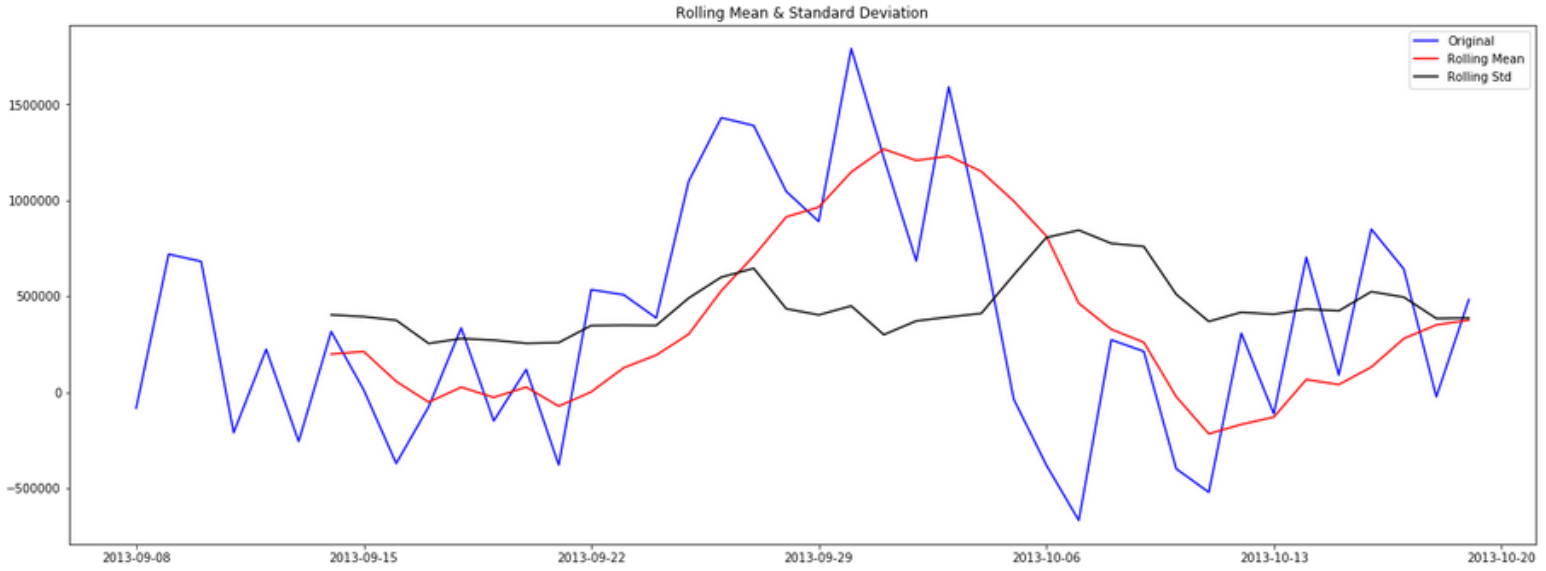
Results of Dickey-Fuller Test:
Test Statistic -1.771613
p-value 0.394643
#Lags Used 7.000000
Number of Observations Used 40.000000
Critical Value (1%) -3.605565
Critical Value (5%) -2.937069
Critical Value (10%) -2.606986
dtype: float64

Take log_first_difference
data['log_first_difference'] = data.Billings_log - data.Billings_log.shift(1)
test_stationarity(data.log_first_difference.dropna(inplace=False))



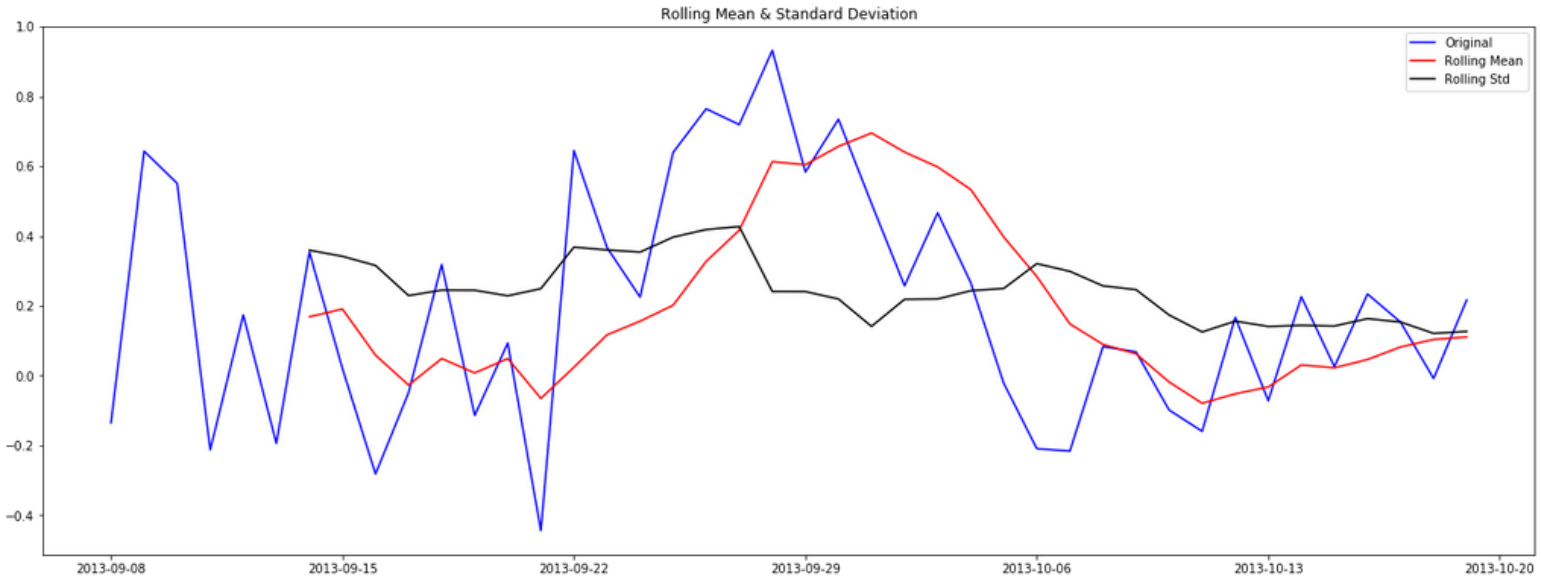
Results of Dickey-Fuller Test:
Test Statistic -1.831369
p-value 0.365003
#Lags Used 7.000000
Number of Observations Used 40.000000
Critical Value (1%) -3.605565
Critical Value (5%) -2.937069
Critical Value (10%) -2.606986
dtype: float64

Take seasonal_difference
data['seasonal_difference'] = data.Billings - data.Billings.shift(7)
test_stationarity(data.seasonal_difference.dropna(inplace=False))
We take 7 days as the seasonal lag.



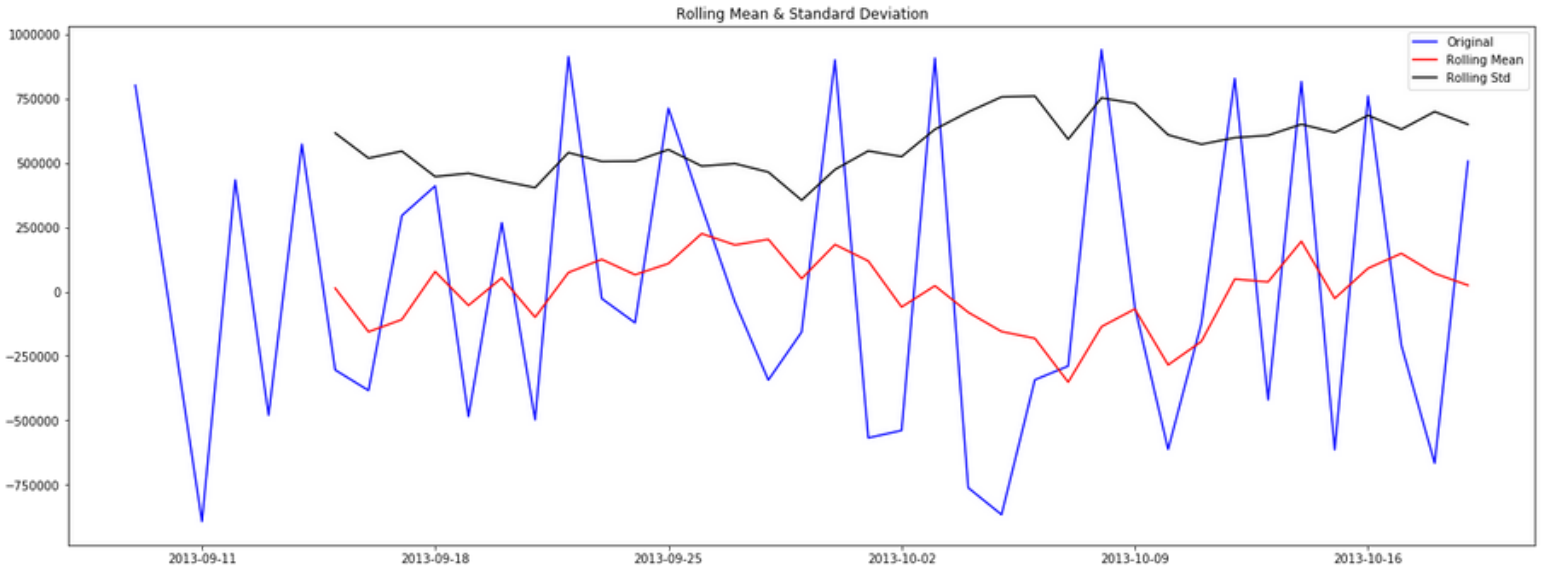
Results of Dickey-Fuller Test:
Test Statistic -3.247368
p-value 0.017401
#Lags Used 8.000000
Number of Observations Used 33.000000
Critical Value (1%) -3.646135
Critical Value (5%) -2.954127
Critical Value (10%) -2.615968
dtype: float64

Take log_seasonal_difference
data['log_seasonal_difference'] = data.Billings_log - data.Billings_log.shift(7)
test_stationarity(data.log_seasonal_difference.dropna(inplace=False))



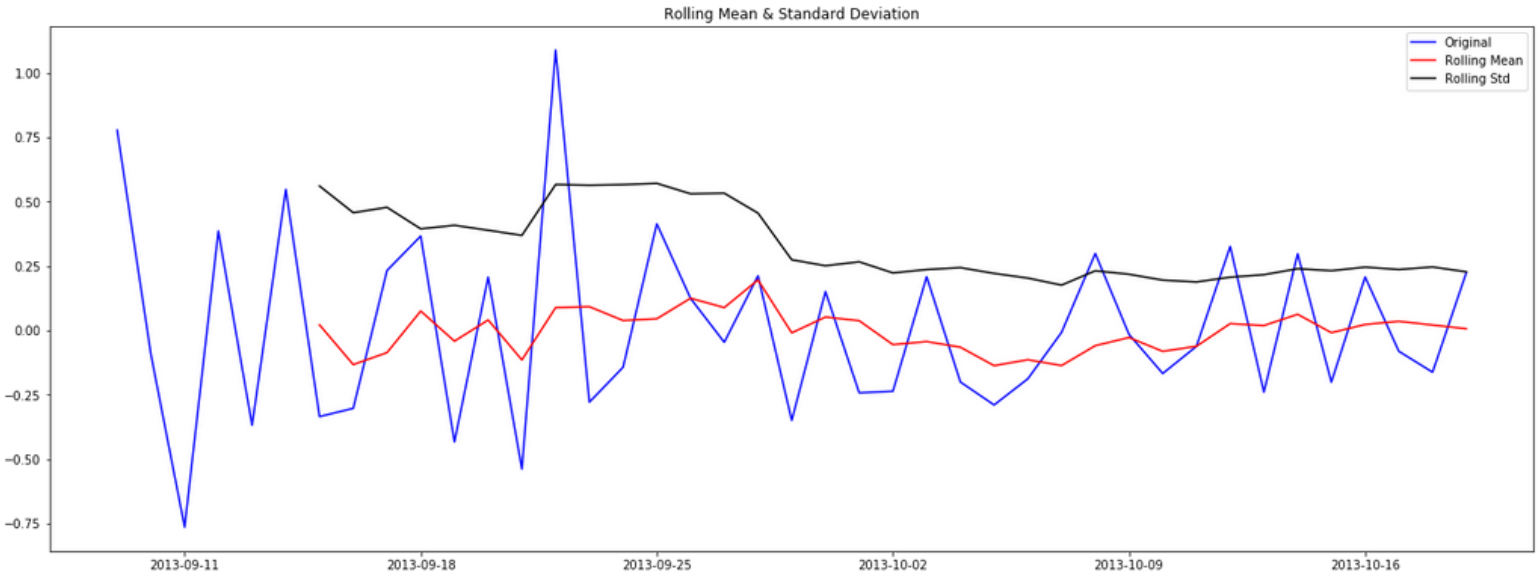
Results of Dickey-Fuller Test:
Test Statistic -2.713955
p-value 0.071654
#Lags Used 8.000000
Number of Observations Used 33.000000
Critical Value (1%) -3.646135
Critical Value (5%) -2.954127
Critical Value (10%) -2.615968
dtype: float64

Take seasonal_first_difference
data['seasonal_first_difference'] = data.first_difference - data.first_difference.shift(test_stationarity(data.seasonal_first_difference.dropna(inplace=False))



Results of Dickey-Fuller Test:
Test Statistic -8.999948e+00
p-value 6.538471e-15
#Lags Used 0.000000e+00
Number of Observations Used 4.000000e+01
Critical Value (1%) -3.605565e+00
Critical Value (5%) -2.937069e+00
Critical Value (10%) -2.606986e+00
dtype: float64

Take log_seasonal_first_difference
data['log_seasonal_first_difference'] = data.log_first_difference - data.log_first_difference.test_stationarity(data.log_seasonal_first_difference.dropna(inplace=False))

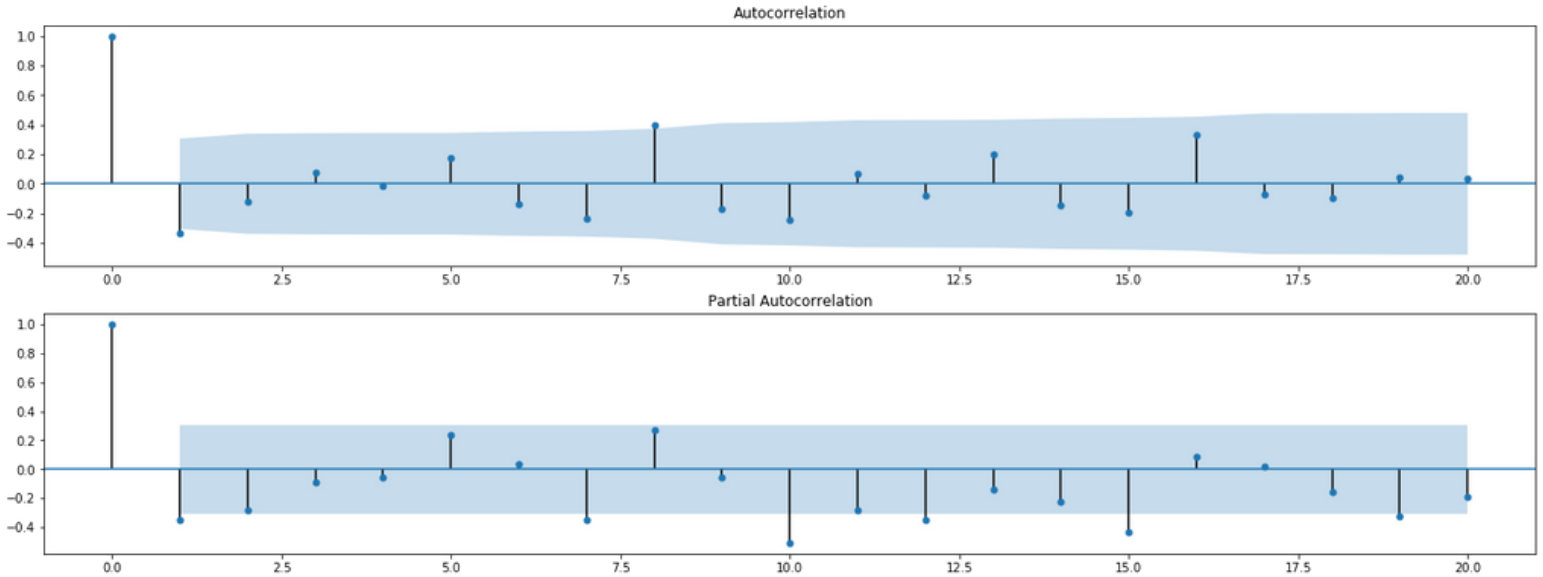


Results of Dickey-Fuller Test:
Test Statistic -2.755552
p-value 0.064909
#Lags Used 9.000000
Number of Observations Used 31.000000
Critical Value (1%) -3.661429
Critical Value (5%) -2.960525
Critical Value (10%) -2.619319
dtype: float64

As you can see by the p-value, taking the seasonal first difference has made our data stationary. By doing this differencing for the log values didn't make the data any more stationary. So we choose use the seasonal first difference series which more stationary.

STEP 3
PLOT THE ACF AND PACF CHARTS

We find the optimal parameters by looking at the autocorrelation and partial autocorrelation graphs.



STEP 4
BUILD UP MODEL

Make the parameters for model ((0,1,0)x(0,1,1,7),

Statespace Model Results

Dep. Variable:	Billings	No. Observations:	49
Model:	SARIMAX(0, 1, 0)x(0, 1, 1, 7)	Log Likelihood	-599.473
Date:	Tue, 04 Sep 2018	AIC	1202.946
Time:	21:39:39	BIC	1206.373
Sample:	09-01-2013	HQIC	1204.194
	- 10-19-2013		
Covariance Type:	opg		

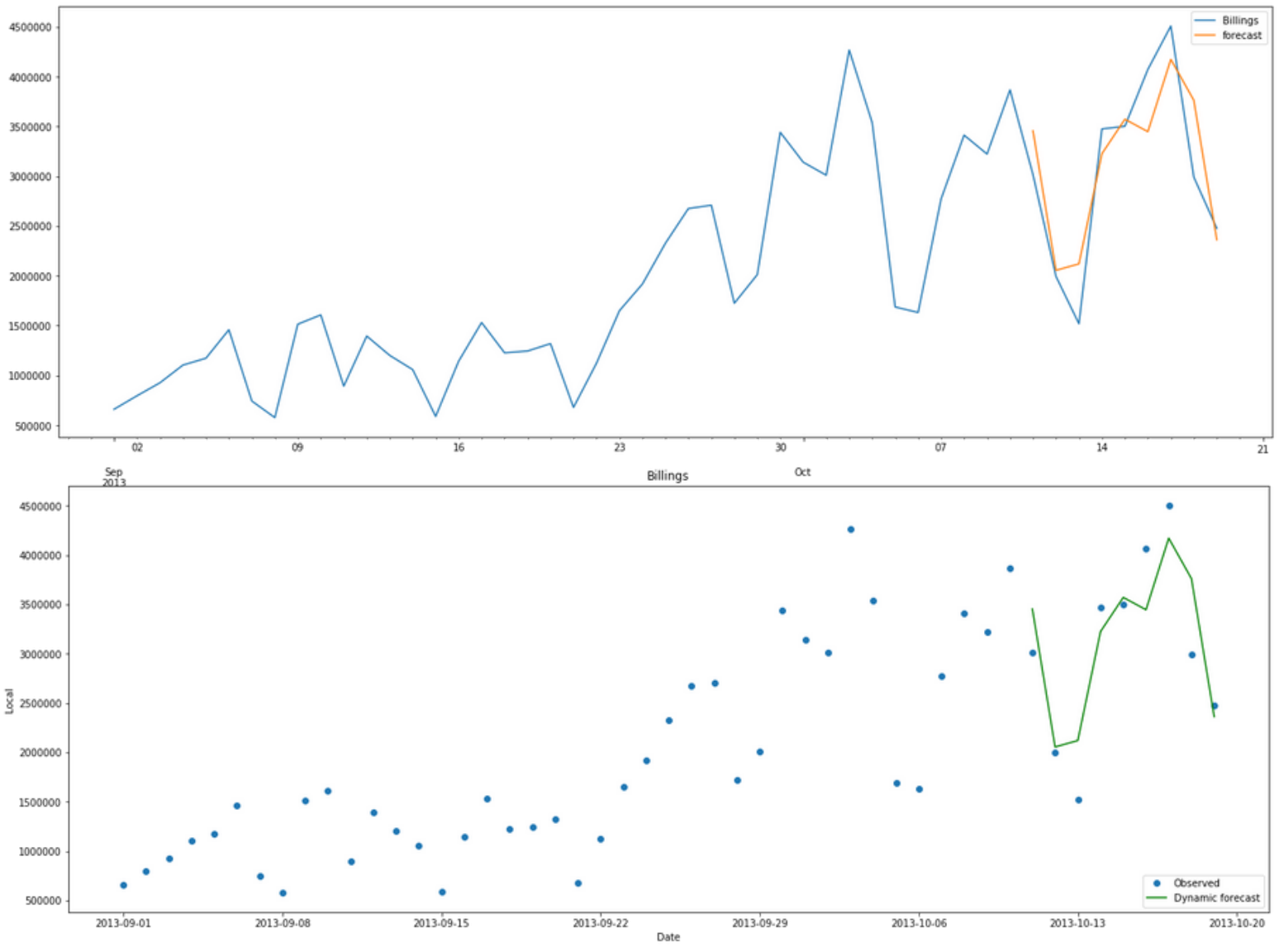
	coef	std err	z	P>|z|	[0.025	0.975]
ma.S.L7	-0.4188	0.265	-1.581	0.114	-0.938	0.101
sigma2	3.789e+11	1.9e-13	2e+24	0.000	3.79e+11	3.79e+11

Ljung-Box (Q):	69.66	Jarque-Bera (JB):	1.29
Prob(Q):	0.00	Prob(JB):	0.52
Heteroskedasticity (H):	1.32	Skew:	0.11
Prob(H) (two-sided):	0.62	Kurtosis:	2.16

STEP 5
ESTIMATION FOR MISSING DATA

Now that we have a model built, we want to use it to make estimate our missing data.First I am using the model to forecast for
time periods that we already have data for, so we can understand how accurate are the forecasts.

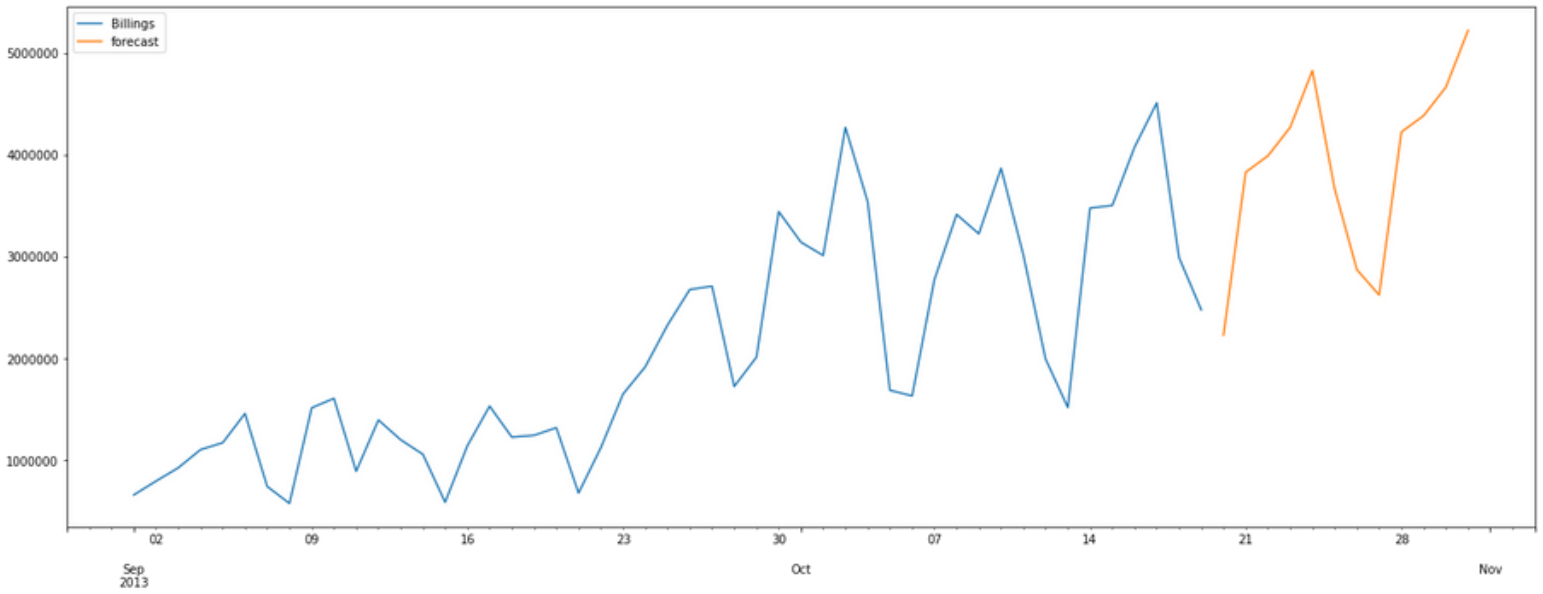
```
data['forecast'] = results.predict(start = 40, end= 49, dynamic= True)
data[['Billings', 'forecast']].plot(figsize=(22, 8))
```



STEP 5
ESTIMATION FOR MISSING DATA

Now I will have use the predict function to create forecast values for these newly added time periods and plot them.

```
data['forecast'] = results.predict(start = 49, end = 60, dynamic= True)
data[['Billings', 'forecast']].plot(figsize=(22, 8))
```

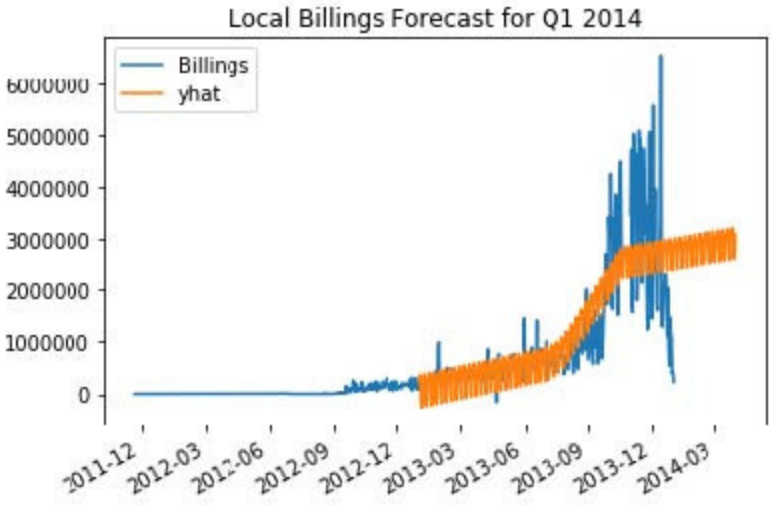



```
data['forecast']['2013-10-20':'2013-10-30'].sum()
= 41557108.592442185
```

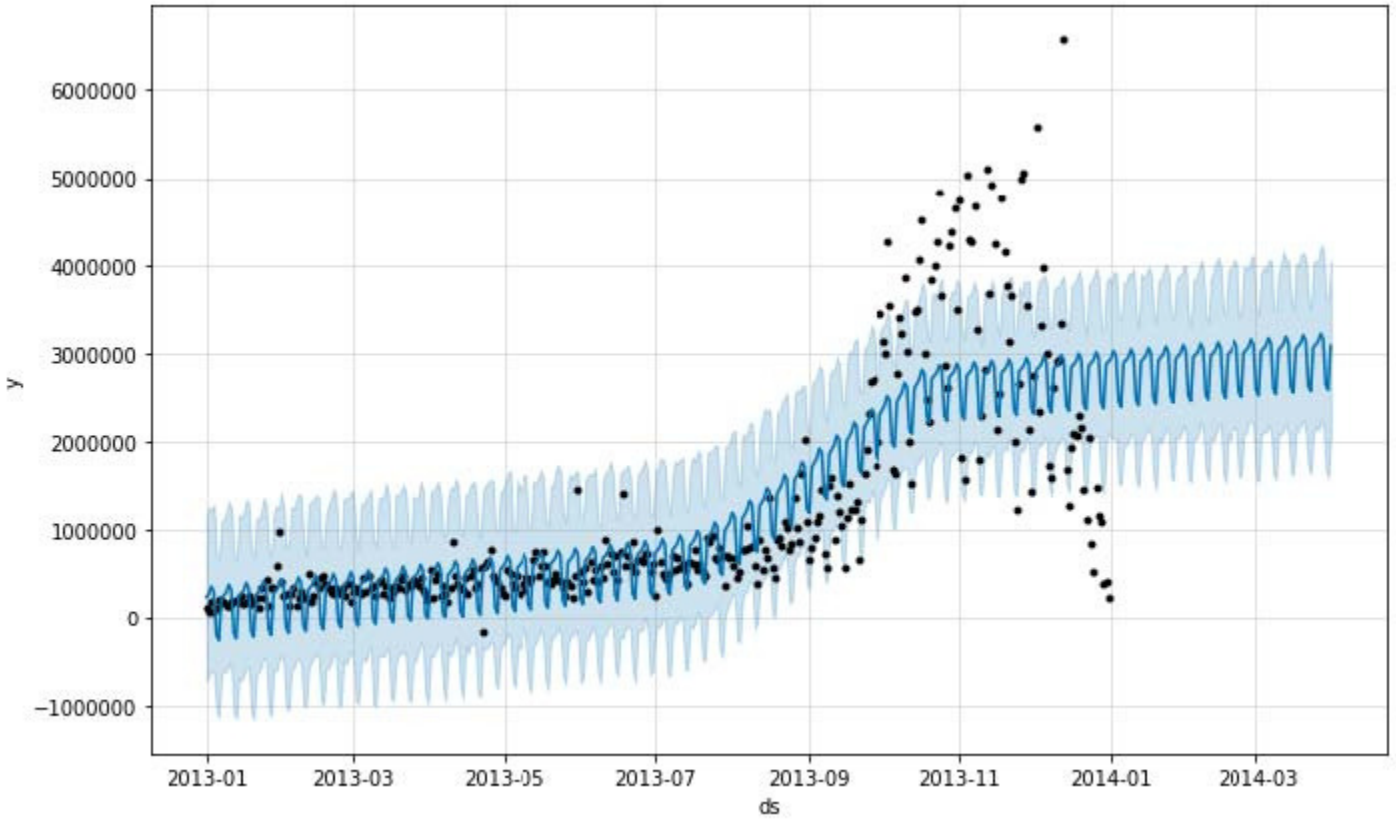
Finally, we got the estimation for missing period data from 2013-10-20 to 2013-10-30 which is about $41.56 million.

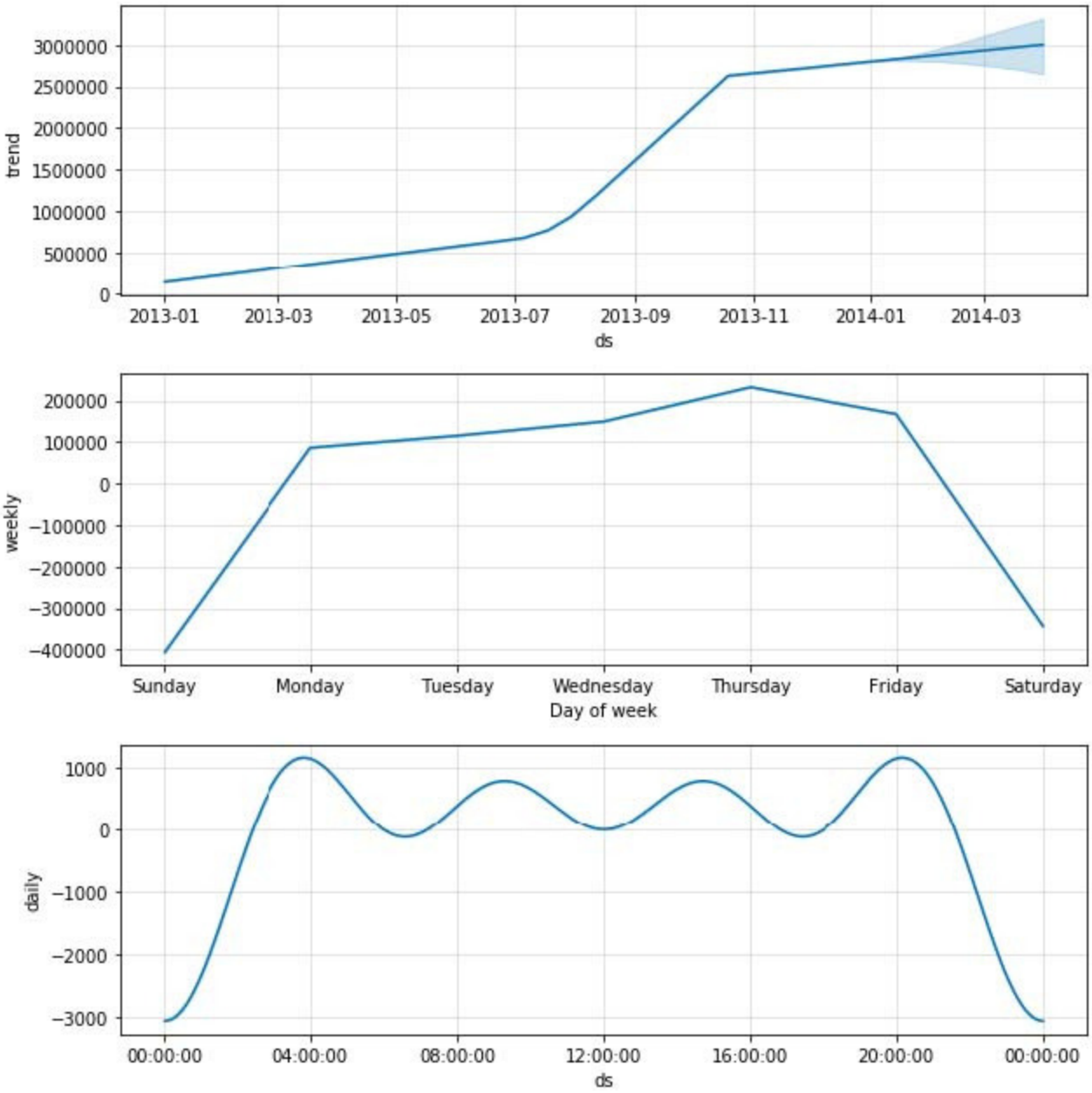
Combined with the billings for local previously which is $409.22 million, the final result for 2013 Q4 Local billings will be $450.78 million.

STEP 6
PREDICT TREND IN NEXT QUARTER
BY USING PROPHET



Moreover, after filling in missing data, I used Prophet which is robust to missing data and shifts in the trend, and typically handles outliers well. It Implements a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.





From the graphs, we could see there is also a seasonality within a week. Also there is an uptrend in the future Q1 2014 with a lower growth rate which confirms the lower growth potential in the future.

PART II
DE-DUPLICATE OF MAR 2015

For identifying duplicates in the 2015 March data, I found that in the url of each coupon the Deal ID will be the second latest element between '/'. The two urls with the same Deal ID should be actually directed to the same webpage which is considered as a duplicate deal.

The first step is to extract Deal ID from each url as the following:

```
import re
import pandas as pd
```

```
df = pd.read_excel('Mar-15_groupon.be_Deals.xlsx', 'Deals')
```

```
Deal_ID = [i.split('/')[-2] for i in list(df['Deal URL'])]
```

Furthermore, I found that even two different Deal IDs could lead to a duplicate deal if they are only different with some numbers in between or after the ID. Hence, I applied regular expression which only extract the alphabets. Then we get Deal ID regular in expression format.

```
Deal_IDre = ["".join(re.findall("[a-z]+", i)) for i in Deal_ID]
```

```
df['Deal IDre'] = Deal_IDre
```

```
sortdf = df.groupby(['Deal IDre', 'Price'])['Quantity Sold'].unique().reset_index()
```

```
quantity = [max(i) for i in sortdf['Quantity Sold']]
```

```
sortdf['Quantity Sold'] = quantity
```

```
sortdf.head()
```

	Deal IDre	Price	Quantity Sold
0	a	19.99	22
1	a	36.99	5
2	aacarwash	39.00	78
3	aalstballooningcompany	29.99	10
4	abluxconcept	119.00	1

```
sortdf['Billings'] = sortdf['Price'] * sortdf['Quantity Sold']
```

```
Total_Billings = sortdf['Billings'].sum()/float(1000000)
```

```
Total_Billings
```

```
24.2383472
```

The last step is to sort the data frame grouped by Deal ID and price. Here I choose to group both ID and price is based on assuming that different price would be a different deal.
One thing needs to be mentioned is that I chose the maximum number for the quantity sold when there are duplicates for a deal. Since each time a deal is bought by consumer, all the duplicated web page will add a record.

Finally, we can calculate the sum of Billings column which is about $24.24 millions for Mar 2015.

Thank you
for your
time.

finn

September 4, 2018

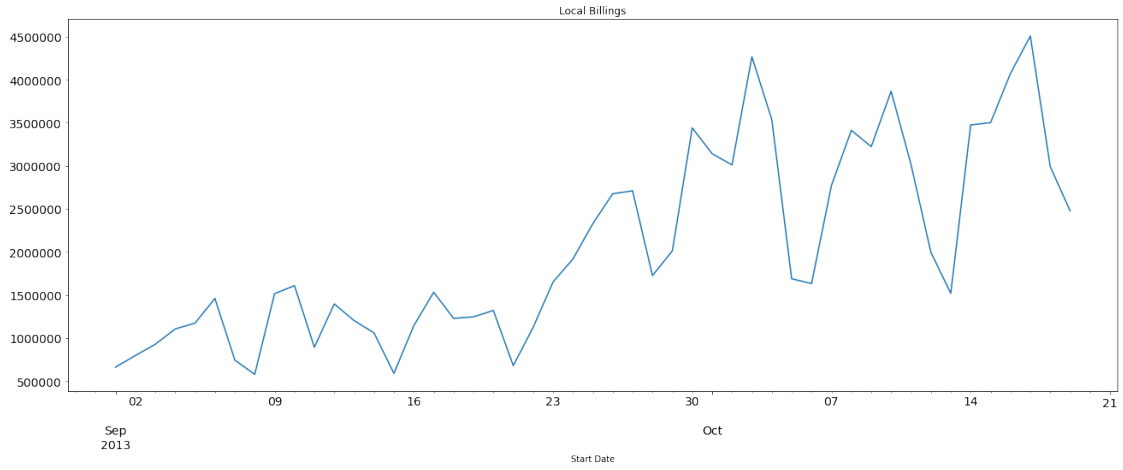
In [1]:
```
import pandas as pd
df_raw = pd.read_excel('Q4_2013_Groupon_North_America_Data_XLSX.xlsx', 'Q4 2013 Raw Da
df_Local = df_raw[df_raw['Segment'] == 'Local']
df_Goods = df_raw[df_raw['Segment'] == 'Goods']
df_Travel = df_raw[df_raw['Segment'] == 'Travel']
```

In [2]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from dateutil.relativedelta import relativedelta
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [3]:
```
Local_Billings = df_Local.groupby('Start Date')['Start Date', 'Billings'].sum()
```

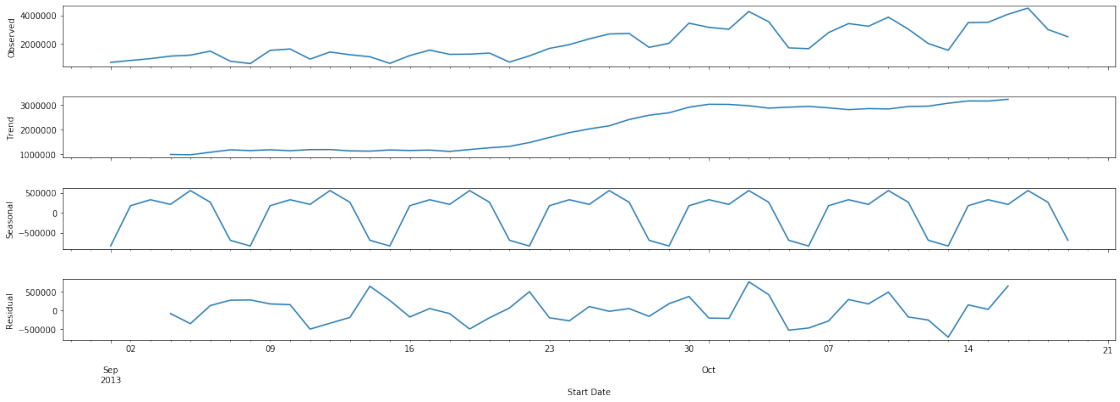
In [4]:
```
data = Local_Billings['2013-09-01':'2013-10-19']
```

In [5]:
```
data.Billings.plot(figsize=(22,8), title= 'Local Billings', fontsize=14)
plt.savefig('Local_Billings.png', bbox_inches='tight')
```



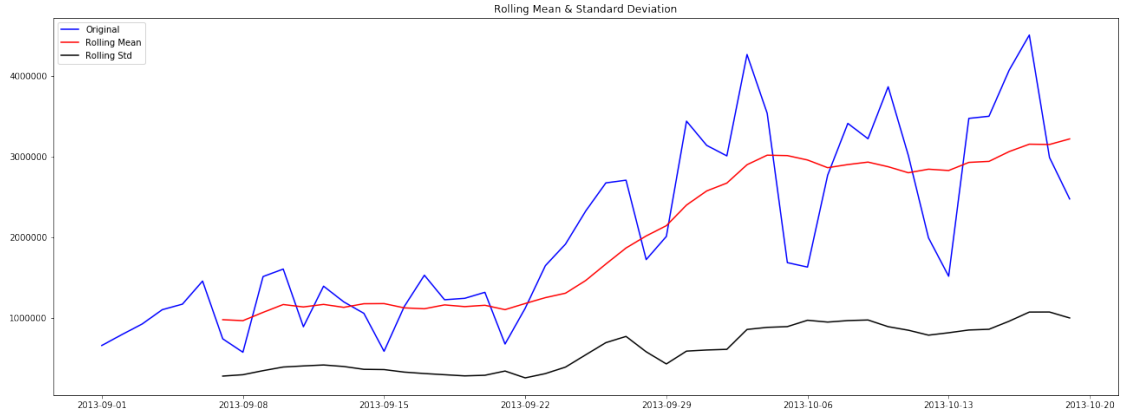
In [6]:
```
decomposition = seasonal_decompose(data.Billings, freq=7)
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(22, 8)
```

```
<Figure size 432x288 with 0 Axes>
```



In [7]:
```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(window=7).mean()
    rolstd = timeseries.rolling(window=7).std()

    #Plot rolling statistics:
    fig = plt.figure(figsize=(22, 8))
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

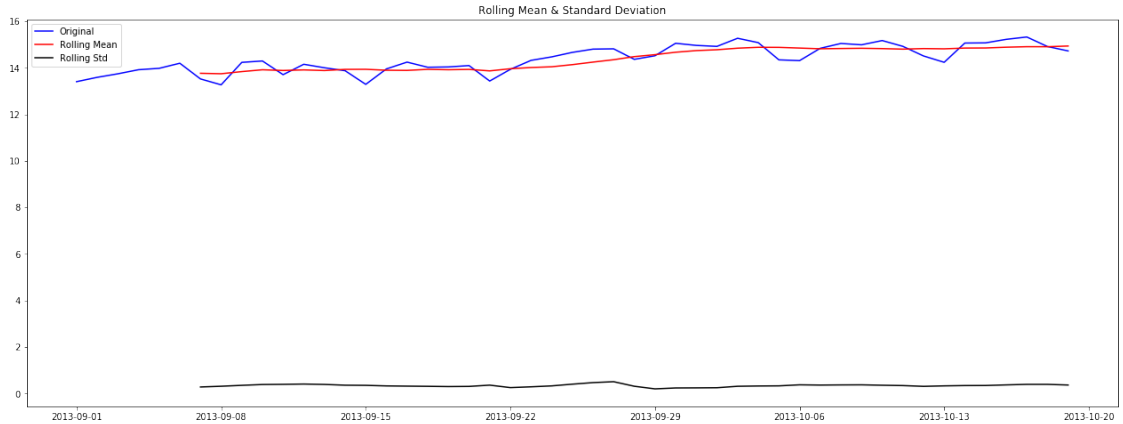
In [8]: test_stationarity(data.Billings)




```
Results of Dickey-Fuller Test:
Test Statistic                  -0.493765
p-value                          0.893227
#Lags Used                       8.000000
Number of Observations Used     40.000000
Critical Value (1%)             -3.605565
Critical Value (5%)             -2.937069
Critical Value (10%)            -2.606986
dtype: float64
```

In [9]:
```
data.Billings_log = data.Billings.apply(lambda x: np.log(x))
test_stationarity(data.Billings_log)
```

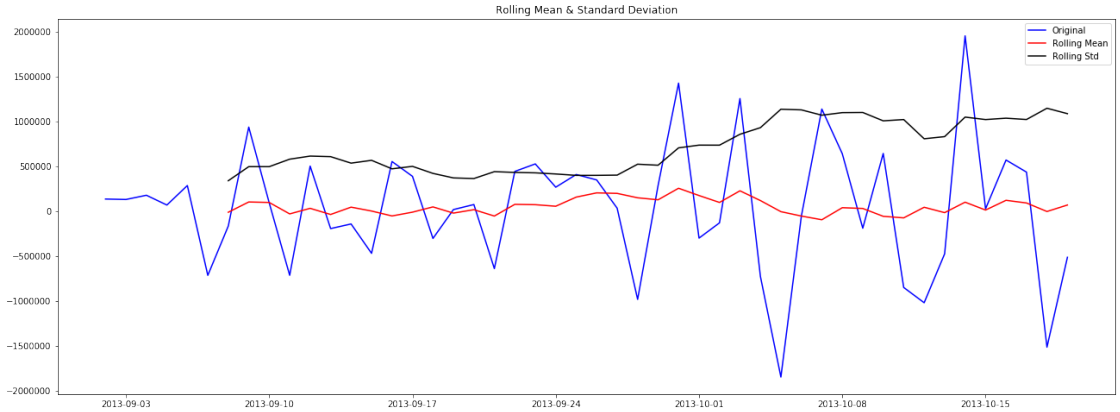
```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: UserWarning: Pandas
  """Entry point for launching an IPython kernel.
```



```
Results of Dickey-Fuller Test:
Test Statistic                  -0.676594
p-value                          0.852720
#Lags Used                       8.000000
Number of Observations Used     40.000000
Critical Value (1%)             -3.605565
Critical Value (5%)             -2.937069
Critical Value (10%)            -2.606986
dtype: float64
```

```
In [10]: data['first_difference'] = data.Billings - data.Billings.shift(1)
         test_stationarity(data.first_difference.dropna(inplace=False))
```

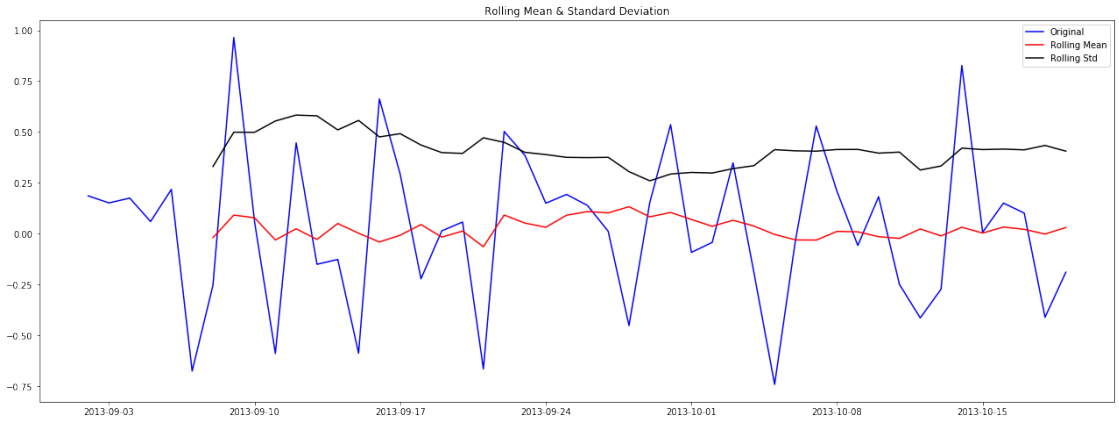
```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarr
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.771613
p-value                          0.394643
#Lags Used                       7.000000
Number of Observations Used     40.000000
Critical Value (1%)             -3.605565
Critical Value (5%)             -2.937069
Critical Value (10%)            -2.606986
dtype: float64
```

```
In [11]: data['log_first_difference'] = data.Billings_log - data.Billings_log.shift(1)
         test_stationarity(data.log_first_difference.dropna(inplace=False))
```

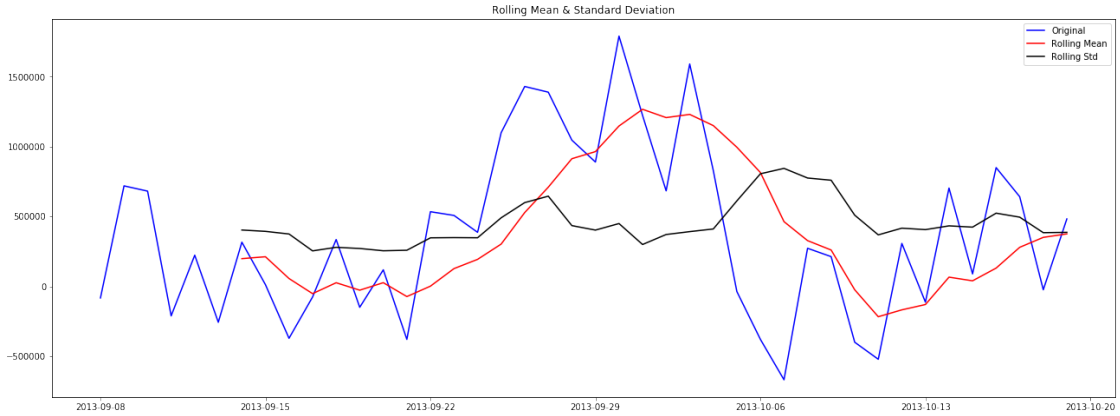
```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```




```
Results of Dickey-Fuller Test:
Test Statistic                 -1.831369
p-value                         0.365003
#Lags Used                      7.000000
Number of Observations Used    40.000000
Critical Value (1%)            -3.605565
Critical Value (5%)            -2.937069
Critical Value (10%)           -2.606986
dtype: float64
```

```
In [12]: data['seasonal_difference'] = data.Billings - data.Billings.shift(7)
         test_stationarity(data.seasonal_difference.dropna(inplace=False))
```

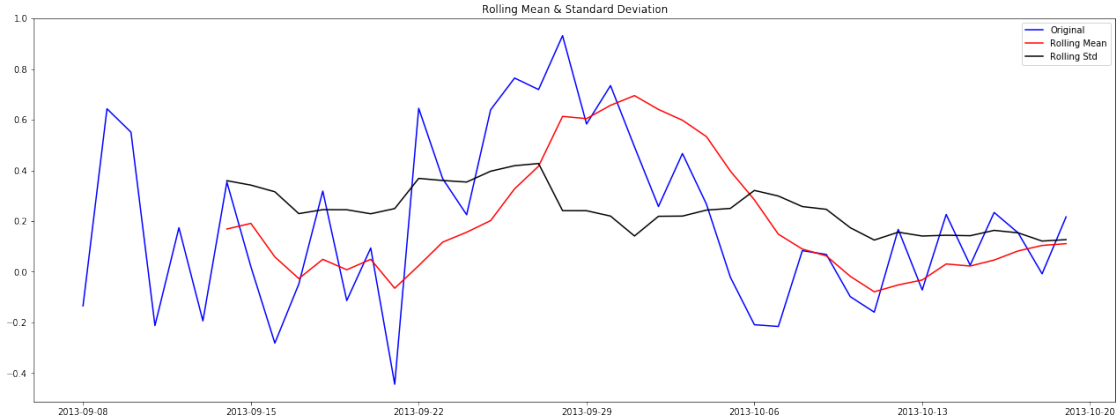
```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -3.247368
p-value                          0.017401
#Lags Used                       8.000000
Number of Observations Used     33.000000
Critical Value (1%)             -3.646135
Critical Value (5%)             -2.954127
Critical Value (10%)            -2.615968
dtype: float64
```

```
In [13]: data['log_seasonal_difference'] = data.Billings_log - data.Billings_log.shift(7)
         test_stationarity(data.log_seasonal_difference.dropna(inplace=False))
```

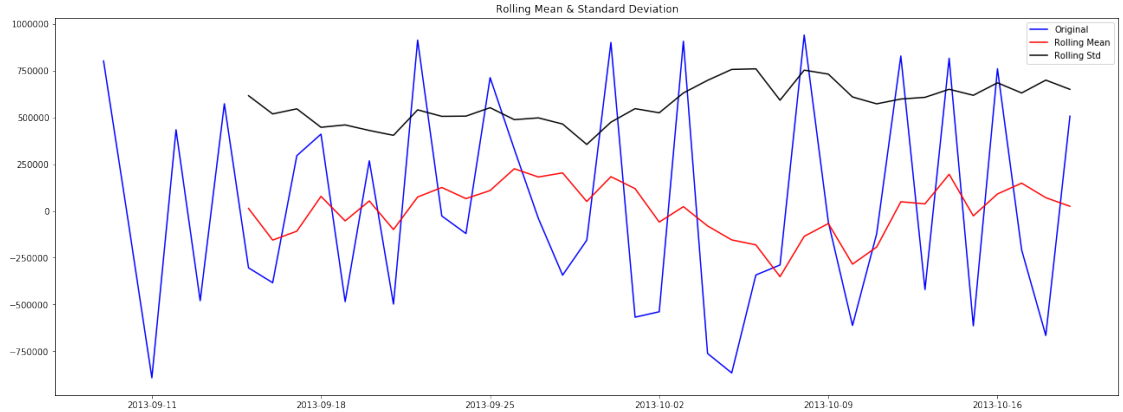
```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarr
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```



```
Results of Dickey-Fuller Test:
Test Statistic                  -2.713955
p-value                          0.071654
#Lags Used                       8.000000
Number of Observations Used     33.000000
Critical Value (1%)             -3.646135
Critical Value (5%)             -2.954127
Critical Value (10%)            -2.615968
dtype: float64
```

```
In [14]: data['seasonal_first_difference'] = data.first_difference - data.first_difference.shi
         test_stationarity(data.seasonal_first_difference.dropna(inplace=False))
```

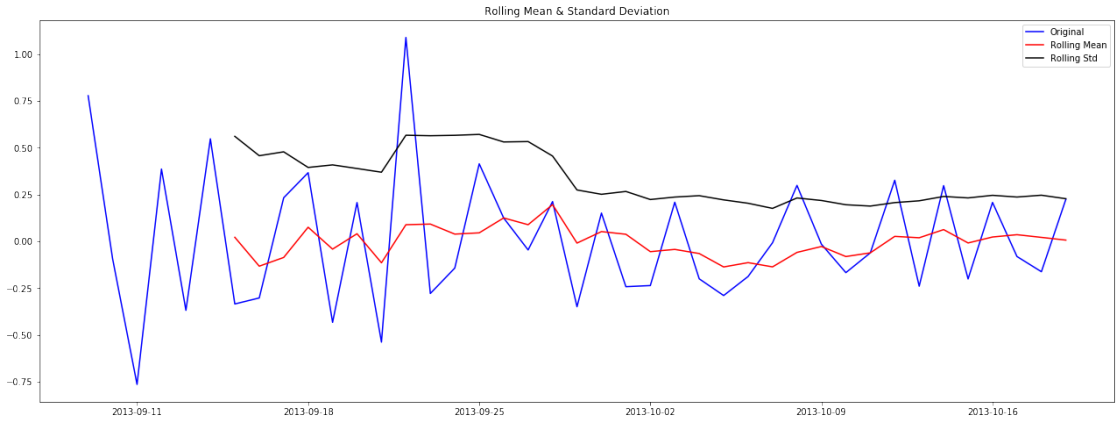
```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarr
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```




```
Results of Dickey-Fuller Test:
Test Statistic                -8.999948e+00
p-value                        6.538471e-15
#Lags Used                     0.000000e+00
Number of Observations Used    4.000000e+01
Critical Value (1%)           -3.605565e+00
Critical Value (5%)           -2.937069e+00
Critical Value (10%)          -2.606986e+00
dtype: float64
```

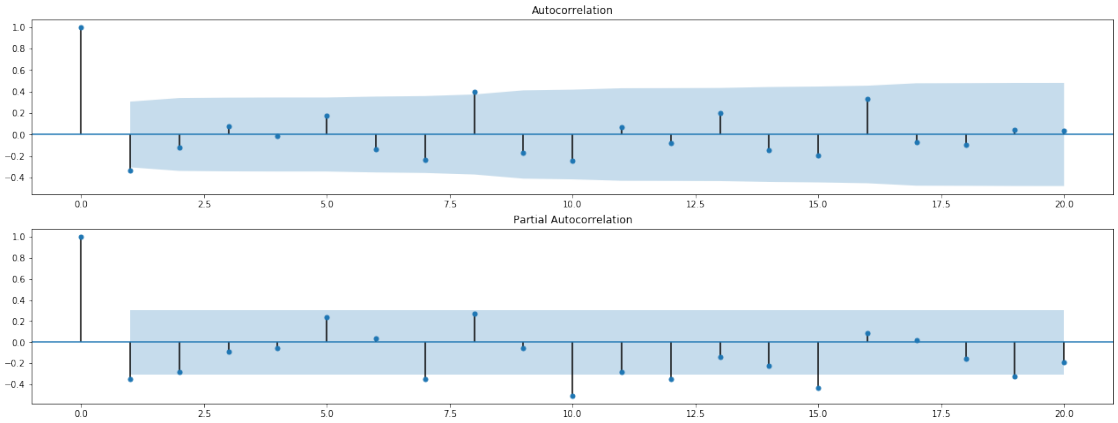
In [15]:
```
data['log_seasonal_first_difference'] = data.log_first_difference - data.log_first_di
test_stationarity(data.log_seasonal_first_difference.dropna(inplace=False))
```

```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -2.755552
p-value                          0.064909
#Lags Used                       9.000000
Number of Observations Used     31.000000
Critical Value (1%)             -3.661429
Critical Value (5%)             -2.960525
Critical Value (10%)            -2.619319
dtype: float64
```

In [16]:
```
fig = plt.figure(figsize=(22,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(data.seasonal_first_difference.iloc[8:], lags=20, ax=ax
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(data.seasonal_first_difference.iloc[8:], lags=20, ax=a
```



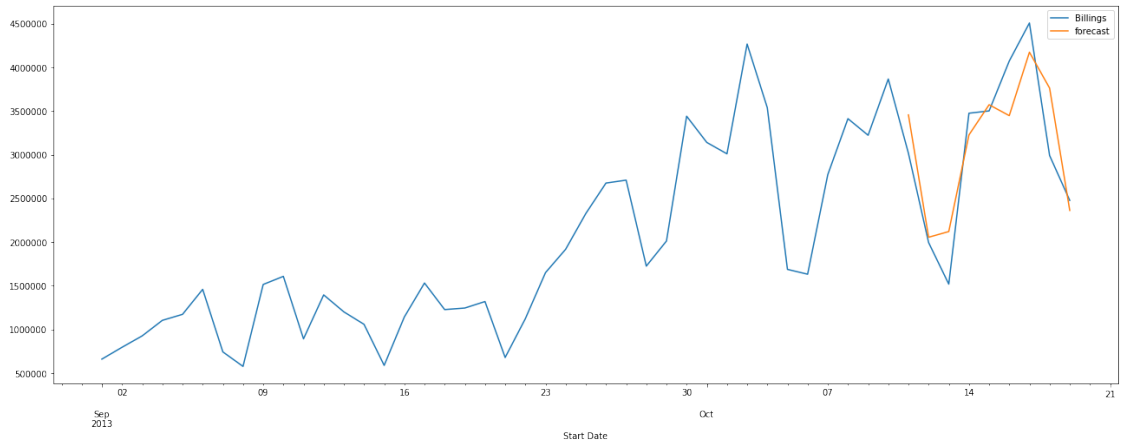

```
In [17]: mod = sm.tsa.statespace.SARIMAX(data.Billings, trend='n', order=(0,1,0), seasonal_orde
         results = mod.fit()
         print (results.summary())
```

```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                            Billings   No. Observations:                   49
Model:             SARIMAX(0, 1, 0)x(0, 1, 1, 7)   Log Likelihood                -599.473
Date:                           Tue, 04 Sep 2018   AIC                           1202.946
Time:                                   21:39:39   BIC                           1206.373
Sample:                                 09-01-2013   HQIC                          1204.194
                                      - 10-19-2013
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L7       -0.4188      0.265     -1.581      0.114      -0.938       0.101
sigma2      3.789e+11    1.9e-13      2e+24      0.000    3.79e+11    3.79e+11
===================================================================================
Ljung-Box (Q):                       69.66   Jarque-Bera (JB):                 1.29
Prob(Q):                              0.00   Prob(JB):                         0.52
Heteroskedasticity (H):               1.32   Skew:                             0.11
Prob(H) (two-sided):                  0.62   Kurtosis:                         2.16
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.76e+40. Standard er


/Users/vicky/anaconda3/lib/python3.6/site-packages/statsmodels/tsa/base/tsa_model.py:171: Value
  % freq, ValueWarning)
```

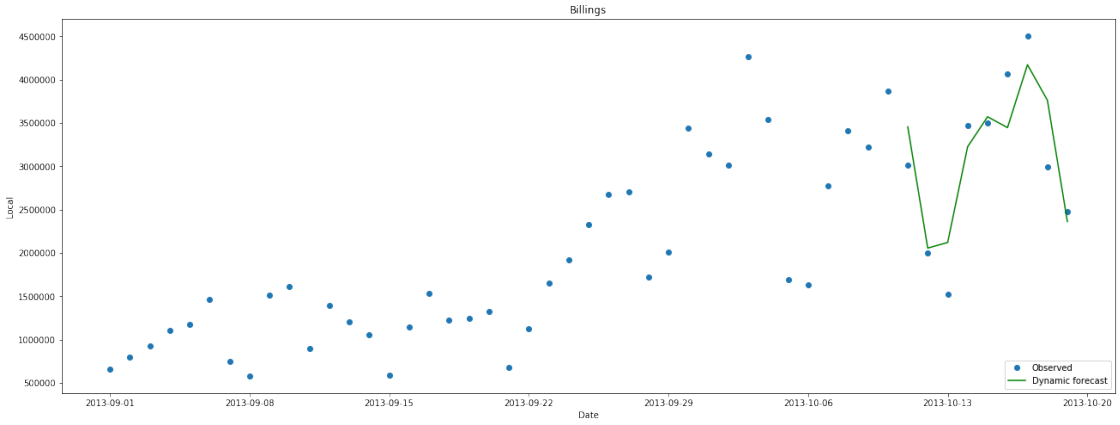
In [18]:
```
data['forecast'] = results.predict(start = 40, end= 49, dynamic= True)
data[['Billings', 'forecast']].plot(figsize=(22, 8))
```

```
/Users/vicky/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x10e279a58>



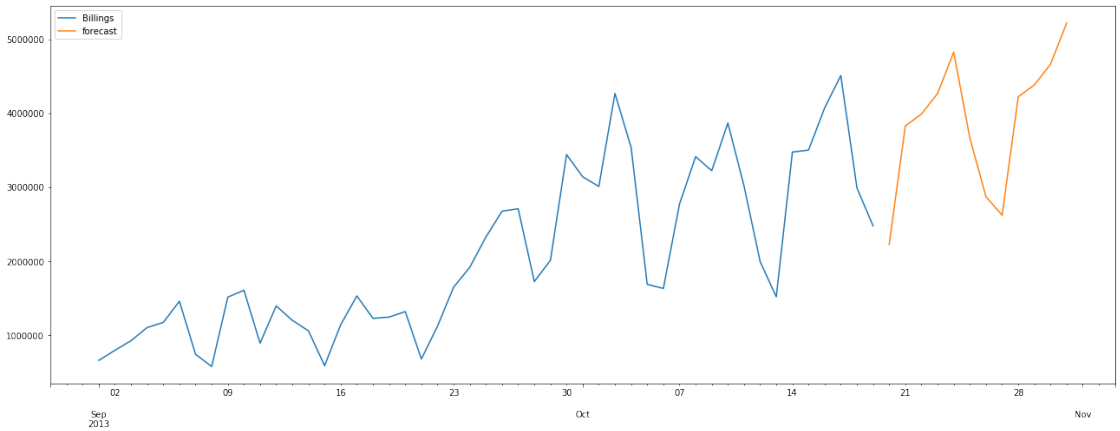
In [19]:
```
npredict = data.Billings['2013-09-01':].shape[0]
fig, ax = plt.subplots(figsize=(22,8))
npre = 7
ax.set(title='Billings', xlabel='Date', ylabel='Local')
ax.plot(data.index[-npredict-npre+1:], data.iloc[-npredict-npre+1:]['Billings'], 'o',
ax.plot(data.index[-npredict-npre+1:], data.iloc[-npredict-npre+1:]['forecast'], 'g',
legend = ax.legend(loc='lower right')
legend.get_frame().set_facecolor('w')
#plt.savefig('ts_predict_compare.png', bbox_inches='tight')
```




```
In [20]: start  =  datetime.datetime.strptime("2013-10-20", "%Y-%m-%d")
         date_list = [start + relativedelta(days=x) for x in range(0,20)]
         future = pd.DataFrame(index=date_list, columns= data.columns)
         data = pd.concat([data, future])
```

```
In [21]: data['forecast'] = results.predict(start = 49, end = 60, dynamic= True)
         data[['Billings', 'forecast']].plot(figsize=(22, 8))
         #plt.savefig('ts_predict_future.png', bbox_inches='tight')
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x10f722a20>
```




```
In [22]: data
```

```
Out[22]:                Billings  first_difference  log_first_difference  \
         2013-09-01  6.607904e+05               NaN                   NaN
         2013-09-02  7.963350e+05      1.355445e+05              0.186583
         2013-09-03  9.268435e+05      1.305085e+05              0.151765
```

2013-09-04	1.104587e+06	1.777438e+05	0.175442
2013-09-05	1.173147e+06	6.855923e+04	0.060218
2013-09-06	1.459212e+06	2.860653e+05	0.218207
2013-09-07	7.433991e+05	-7.158128e+05	-0.674419
2013-09-08	5.773694e+05	-1.660297e+05	-0.252751
2013-09-09	1.515168e+06	9.377988e+05	0.964799
2013-09-10	1.607966e+06	9.279752e+04	0.059443
2013-09-11	8.929169e+05	-7.150488e+05	-0.588232
2013-09-12	1.395849e+06	5.029323e+05	0.446765
2013-09-13	1.201757e+06	-1.940918e+05	-0.149718
2013-09-14	1.059236e+06	-1.425211e+05	-0.126237
2013-09-15	5.890799e+05	-4.701564e+05	-0.586742
2013-09-16	1.143040e+06	5.539596e+05	0.662884
2013-09-17	1.531653e+06	3.886131e+05	0.292656
2013-09-18	1.227678e+06	-3.039750e+05	-0.221223
2013-09-19	1.245592e+06	1.791426e+04	0.014487
2013-09-20	1.319770e+06	7.417796e+04	0.057847
2013-09-21	6.793485e+05	-6.404214e+05	-0.664078
2013-09-22	1.123031e+06	4.436824e+05	0.502652
2013-09-23	1.650251e+06	5.272196e+05	0.384896
2013-09-24	1.918048e+06	2.677974e+05	0.150381
2013-09-25	2.327159e+06	4.091115e+05	0.193340
2013-09-26	2.675751e+06	3.485919e+05	0.139582
2013-09-27	2.709123e+06	3.337194e+04	0.012395
2013-09-28	1.725309e+06	-9.838143e+05	-0.451219
2013-09-29	2.012469e+06	2.871604e+05	0.153956
2013-09-30	3.440883e+06	1.428414e+06	0.536366
...	...	...	...
2013-10-10	3.866735e+06	6.439296e+05	0.182158
2013-10-11	3.015991e+06	-8.507440e+05	-0.248482
2013-10-12	1.994842e+06	-1.021149e+06	-0.413364
2013-10-13	1.519141e+06	-4.757016e+05	-0.272420
2013-10-14	3.475041e+06	1.955900e+06	0.827461
2013-10-15	3.501416e+06	2.637561e+04	0.007561
2013-10-16	4.072212e+06	5.707955e+05	0.151019
2013-10-17	4.507996e+06	4.357840e+05	0.101666
2013-10-18	2.991175e+06	-1.516821e+06	-0.410186
2013-10-19	2.477089e+06	-5.140854e+05	-0.188582
2013-10-20	NaN	NaN	NaN
2013-10-21	NaN	NaN	NaN
2013-10-22	NaN	NaN	NaN
2013-10-23	NaN	NaN	NaN
2013-10-24	NaN	NaN	NaN
2013-10-25	NaN	NaN	NaN
2013-10-26	NaN	NaN	NaN
2013-10-27	NaN	NaN	NaN
2013-10-28	NaN	NaN	NaN
2013-10-29	NaN	NaN	NaN

```
2013-10-30           NaN               NaN                   NaN
2013-10-31           NaN               NaN                   NaN
2013-11-01           NaN               NaN                   NaN
2013-11-02           NaN               NaN                   NaN
2013-11-03           NaN               NaN                   NaN
2013-11-04           NaN               NaN                   NaN
2013-11-05           NaN               NaN                   NaN
2013-11-06           NaN               NaN                   NaN
2013-11-07           NaN               NaN                   NaN
2013-11-08           NaN               NaN                   NaN

            seasonal_difference  log_seasonal_difference  \
2013-09-01                  NaN                      NaN
2013-09-02                  NaN                      NaN
2013-09-03                  NaN                      NaN
2013-09-04                  NaN                      NaN
2013-09-05                  NaN                      NaN
2013-09-06                  NaN                      NaN
2013-09-07                  NaN                      NaN
2013-09-08        -8.342101e+04                -0.134954
2013-09-09         7.188332e+05                 0.643262
2013-09-10         6.811222e+05                 0.550940
2013-09-11        -2.116704e+05                -0.212734
2013-09-12         2.227027e+05                 0.173813
2013-09-13        -2.574545e+05                -0.194111
2013-09-14         3.158372e+05                 0.354070
2013-09-15         1.171051e+04                 0.020080
2013-09-16        -3.721286e+05                -0.281835
2013-09-17        -7.631301e+04                -0.048622
2013-09-18         3.347608e+05                 0.318386
2013-09-19        -1.502573e+05                -0.113892
2013-09-20         1.180125e+05                 0.093672
2013-09-21        -3.798877e+05                -0.444169
2013-09-22         5.339511e+05                 0.645225
2013-09-23         5.072110e+05                 0.367236
2013-09-24         3.863953e+05                 0.224961
2013-09-25         1.099482e+06                 0.639524
2013-09-26         1.430159e+06                 0.764619
2013-09-27         1.389353e+06                 0.719168
2013-09-28         1.045960e+06                 0.932027
2013-09-29         8.894385e+05                 0.583331
2013-09-30         1.790632e+06                 0.734801
...                         ...                      ...
2013-10-10        -4.001325e+05                -0.098469
2013-10-11        -5.220921e+05                -0.159657
2013-10-12         3.069616e+05                 0.167091
2013-10-13        -1.134224e+05                -0.072006
2013-10-14         7.030876e+05                 0.226054
```

2013-10-15	8.868560e+04	0.025655
2013-10-16	8.494065e+05	0.233934
2013-10-17	6.412609e+05	0.153442
2013-10-18	-2.481629e+04	-0.008262
2013-10-19	4.822471e+05	0.216519
2013-10-20	NaN	NaN
2013-10-21	NaN	NaN
2013-10-22	NaN	NaN
2013-10-23	NaN	NaN
2013-10-24	NaN	NaN
2013-10-25	NaN	NaN
2013-10-26	NaN	NaN
2013-10-27	NaN	NaN
2013-10-28	NaN	NaN
2013-10-29	NaN	NaN
2013-10-30	NaN	NaN
2013-10-31	NaN	NaN
2013-11-01	NaN	NaN
2013-11-02	NaN	NaN
2013-11-03	NaN	NaN
2013-11-04	NaN	NaN
2013-11-05	NaN	NaN
2013-11-06	NaN	NaN
2013-11-07	NaN	NaN
2013-11-08	NaN	NaN

	seasonal_first_difference	log_seasonal_first_difference	\
2013-09-01	NaN	NaN	
2013-09-02	NaN	NaN	
2013-09-03	NaN	NaN	
2013-09-04	NaN	NaN	
2013-09-05	NaN	NaN	
2013-09-06	NaN	NaN	
2013-09-07	NaN	NaN	
2013-09-08	NaN	NaN	
2013-09-09	802254.2235	0.778216	
2013-09-10	-37711.0285	-0.092321	
2013-09-11	-892792.6300	-0.763674	
2013-09-12	434373.1045	0.386547	
2013-09-13	-480157.1195	-0.367925	
2013-09-14	573291.6745	0.548182	
2013-09-15	-304126.7055	-0.333991	
2013-09-16	-383839.1325	-0.301915	
2013-09-17	295815.6160	0.233213	
2013-09-18	411073.8190	0.367009	
2013-09-19	-485018.0770	-0.432278	
2013-09-20	268269.7655	0.207565	
2013-09-21	-497900.2370	-0.537842	

2013-09-22	913838.8050	1.089394
2013-09-23	-26740.0170	-0.277989
2013-09-24	-120815.7665	-0.142276
2013-09-25	713086.4460	0.414563
2013-09-26	330677.6475	0.125095
2013-09-27	-40806.0180	-0.045452
2013-09-28	-343392.8820	0.212860
2013-09-29	-156521.9950	-0.348696
2013-09-30	901194.0185	0.151470
...	...	...
2013-10-10	-612539.6850	-0.166649
2013-10-11	-121959.6600	-0.061187
2013-10-12	829053.7765	0.326748
2013-10-13	-420384.0220	-0.239098
2013-10-14	816509.9795	0.298060
2013-10-15	-614401.9965	-0.200399
2013-10-16	760720.9340	0.208279
2013-10-17	-208145.6090	-0.080492
2013-10-18	-666077.2180	-0.161704
2013-10-19	507063.3720	0.224782
2013-10-20	NaN	NaN
2013-10-21	NaN	NaN
2013-10-22	NaN	NaN
2013-10-23	NaN	NaN
2013-10-24	NaN	NaN
2013-10-25	NaN	NaN
2013-10-26	NaN	NaN
2013-10-27	NaN	NaN
2013-10-28	NaN	NaN
2013-10-29	NaN	NaN
2013-10-30	NaN	NaN
2013-10-31	NaN	NaN
2013-11-01	NaN	NaN
2013-11-02	NaN	NaN
2013-11-03	NaN	NaN
2013-11-04	NaN	NaN
2013-11-05	NaN	NaN
2013-11-06	NaN	NaN
2013-11-07	NaN	NaN
2013-11-08	NaN	NaN

	forecast
2013-09-01	NaN
2013-09-02	NaN
2013-09-03	NaN
2013-09-04	NaN
2013-09-05	NaN
2013-09-06	NaN

```
2013-09-07           NaN
2013-09-08           NaN
2013-09-09           NaN
2013-09-10           NaN
2013-09-11           NaN
2013-09-12           NaN
2013-09-13           NaN
2013-09-14           NaN
2013-09-15           NaN
2013-09-16           NaN
2013-09-17           NaN
2013-09-18           NaN
2013-09-19           NaN
2013-09-20           NaN
2013-09-21           NaN
2013-09-22           NaN
2013-09-23           NaN
2013-09-24           NaN
2013-09-25           NaN
2013-09-26           NaN
2013-09-27           NaN
2013-09-28           NaN
2013-09-29           NaN
2013-09-30           NaN
...                  ...
2013-10-10           NaN
2013-10-11           NaN
2013-10-12           NaN
2013-10-13           NaN
2013-10-14           NaN
2013-10-15           NaN
2013-10-16           NaN
2013-10-17           NaN
2013-10-18           NaN
2013-10-19           NaN
2013-10-20  2.227903e+06
2013-10-21  3.827660e+06
2013-10-22  3.987888e+06
2013-10-23  4.267343e+06
2013-10-24  4.824704e+06
2013-10-25  3.663981e+06
2013-10-26  2.871038e+06
2013-10-27  2.621852e+06
2013-10-28  4.221609e+06
2013-10-29  4.381838e+06
2013-10-30  4.661292e+06
2013-10-31  5.218653e+06
2013-11-01           NaN
```

```
2013-11-02           NaN
2013-11-03           NaN
2013-11-04           NaN
2013-11-05           NaN
2013-11-06           NaN
2013-11-07           NaN
2013-11-08           NaN

[69 rows x 8 columns]
```

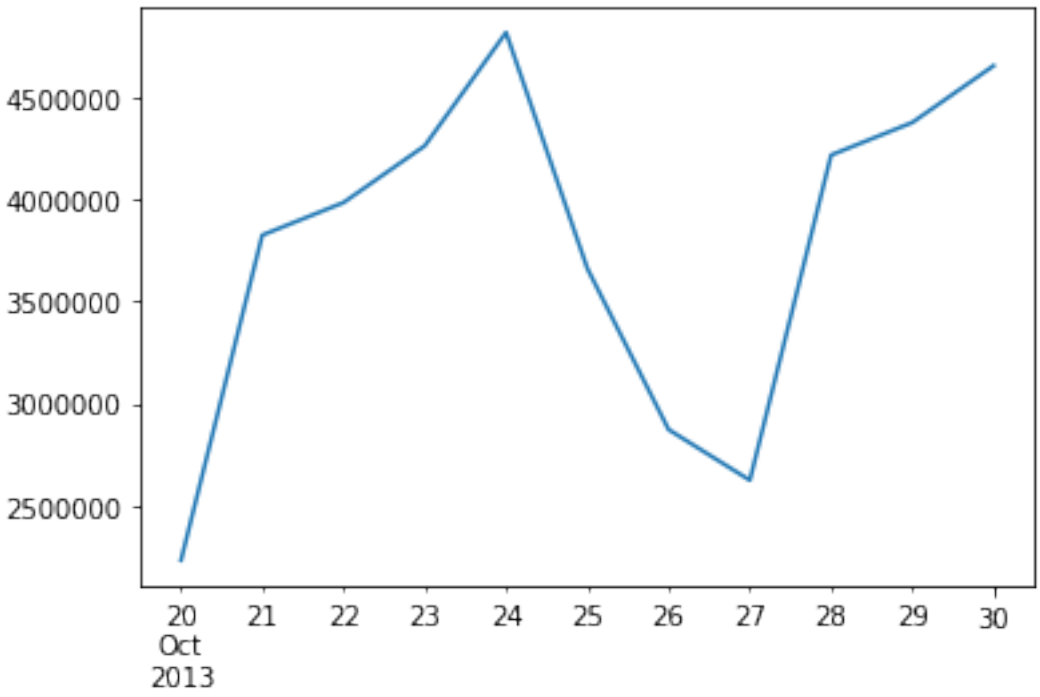
```
In [23]: data['forecast']['2013-10-20':'2013-10-30'].sum()
```

```
Out[23]: 41557108.592442185
```

```
In [115]: estimate = data['forecast']['2013-10-20':'2013-10-30']
```

```
In [140]: estimate.plot()
```

```
Out[140]: <matplotlib.axes._subplots.AxesSubplot at 0x10ed30198>
```




```
In [24]: from fbprophet import Prophet
```

```
In [141]: dataf = Local_Billings['2013-01-01':'2013-12-31']
```

```
In [142]: dataf.head()
```

```
Out[142]:                Billings
          Start Date
          2013-01-01  133671.4000
          2013-01-02   87383.1355
          2013-01-03  199138.1000
          2013-01-04   89798.5000
          2013-01-05  115552.3750

In [143]: est = estimate.reset_index()
          est.columns=['ds', 'y']

In [144]: dataf.reset_index(inplace=True)
          dataf.columns = ['ds', 'y']

In [145]: dataf.set_index('ds', inplace=True)
          dataf.head()

Out[145]:                       y
          ds
          2013-01-01  133671.4000
          2013-01-02   87383.1355
          2013-01-03  199138.1000
          2013-01-04   89798.5000
          2013-01-05  115552.3750

In [146]: est.set_index('ds', inplace=True)
          est.head()

Out[146]:                        y
          ds
          2013-10-20  2.227903e+06
          2013-10-21  3.827660e+06
          2013-10-22  3.987888e+06
          2013-10-23  4.267343e+06
          2013-10-24  4.824704e+06

In [147]: dataf1 = pd.concat([dataf, est])

In [148]: dataforecast = dataf1.reset_index()

In [149]: model = Prophet(daily_seasonality=True)
          model.fit(dataforecast)

INFO:fbprophet.forecaster:Disabling yearly seasonality. Run prophet with yearly_seasonality=Tru
/Users/vicky/anaconda3/lib/python3.6/site-packages/pystan/misc.py:399: FutureWarning: Conversio
  elif np.issubdtype(np.asarray(v).dtype, float):


Out[149]: <fbprophet.forecaster.Prophet at 0x10f360be0>
```

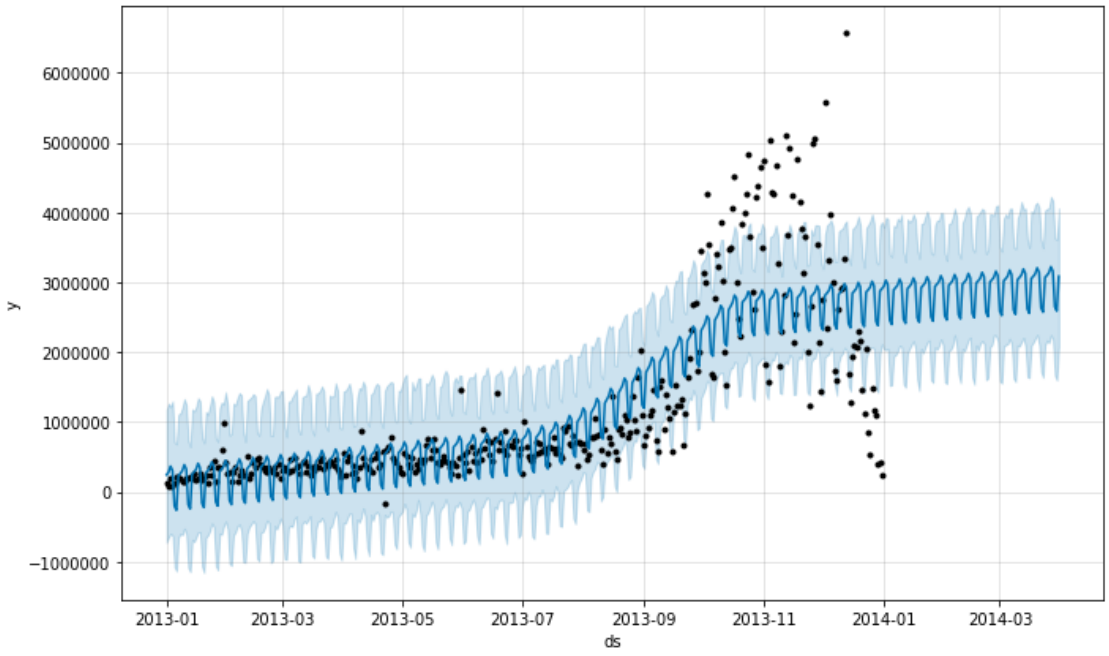
```
In [150]: future = model.make_future_dataframe(periods=90)
          future.tail()
```

```
Out[150]:            ds
          450 2014-03-27
          451 2014-03-28
          452 2014-03-29
          453 2014-03-30
          454 2014-03-31
```

```
In [151]: forecast = model.predict(future)
          forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

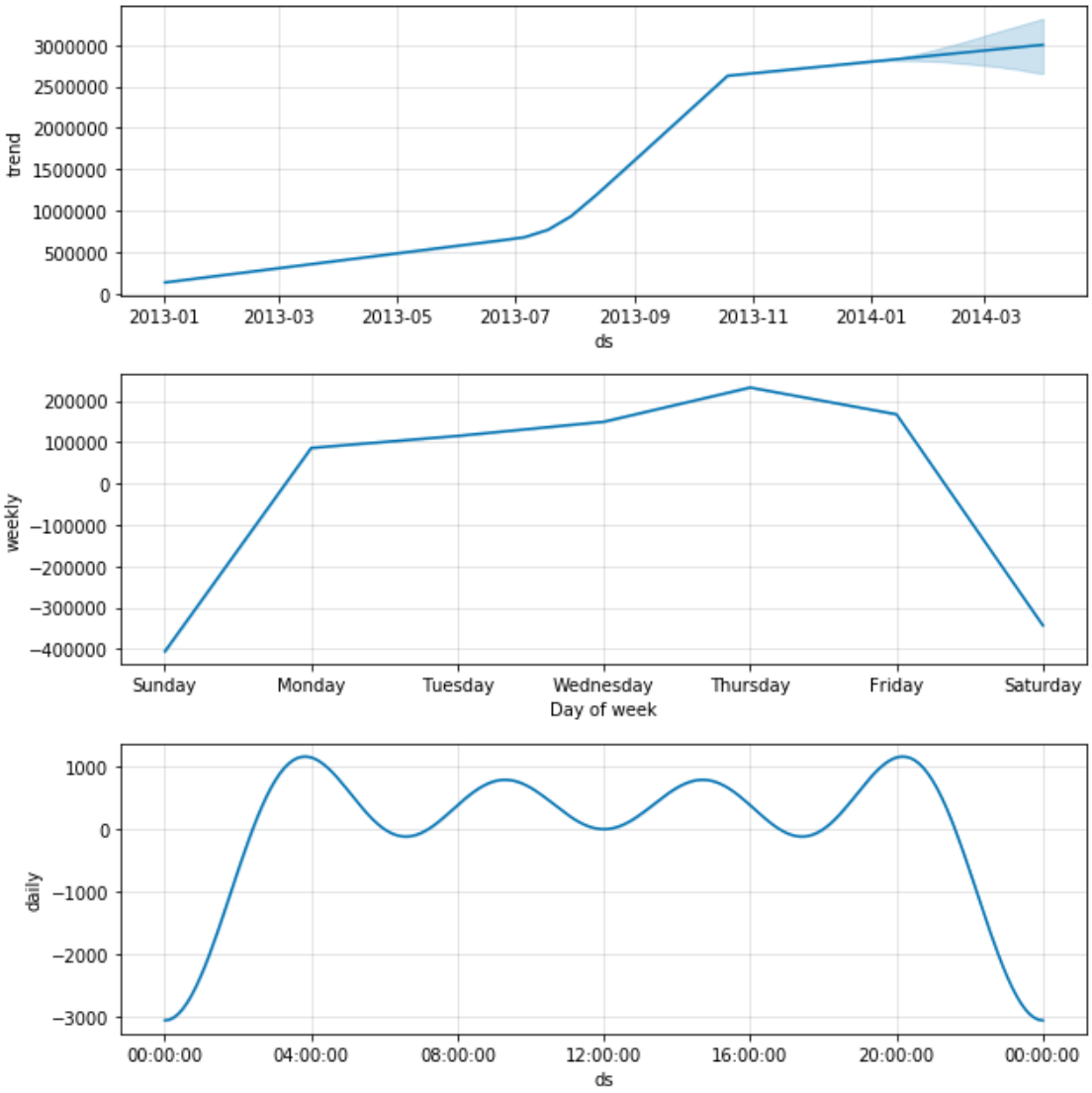
```
Out[151]:            ds          yhat    yhat_lower    yhat_upper
          450 2014-03-27  3.221339e+06  2.244154e+06  4.226177e+06
          451 2014-03-28  3.158876e+06  2.182102e+06  4.167847e+06
          452 2014-03-29  2.650727e+06  1.669018e+06  3.630408e+06
          453 2014-03-30  2.589856e+06  1.598655e+06  3.613669e+06
          454 2014-03-31  3.084363e+06  2.091375e+06  4.059499e+06
```

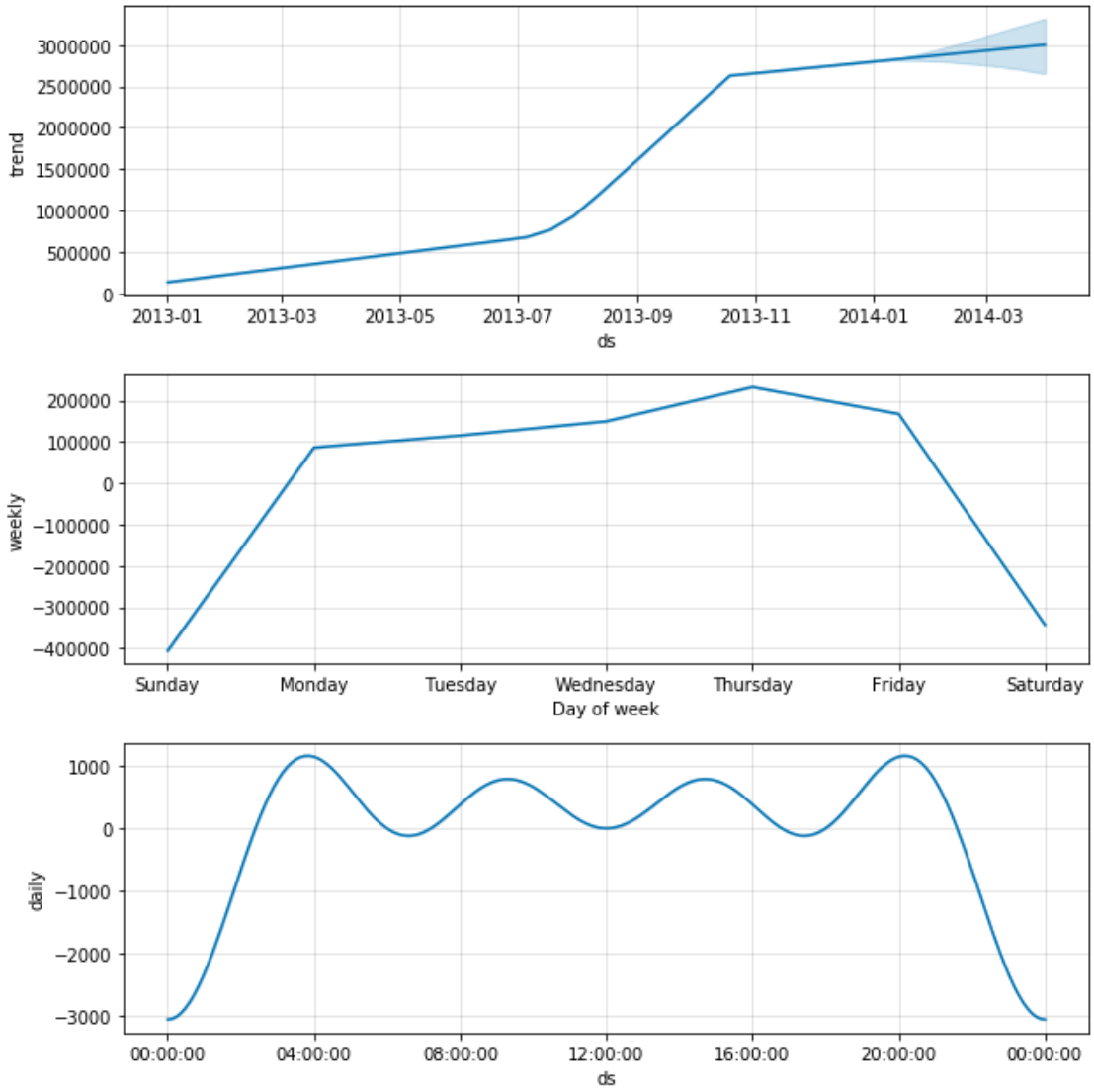
```
In [152]: fig1 = model.plot(forecast)
```



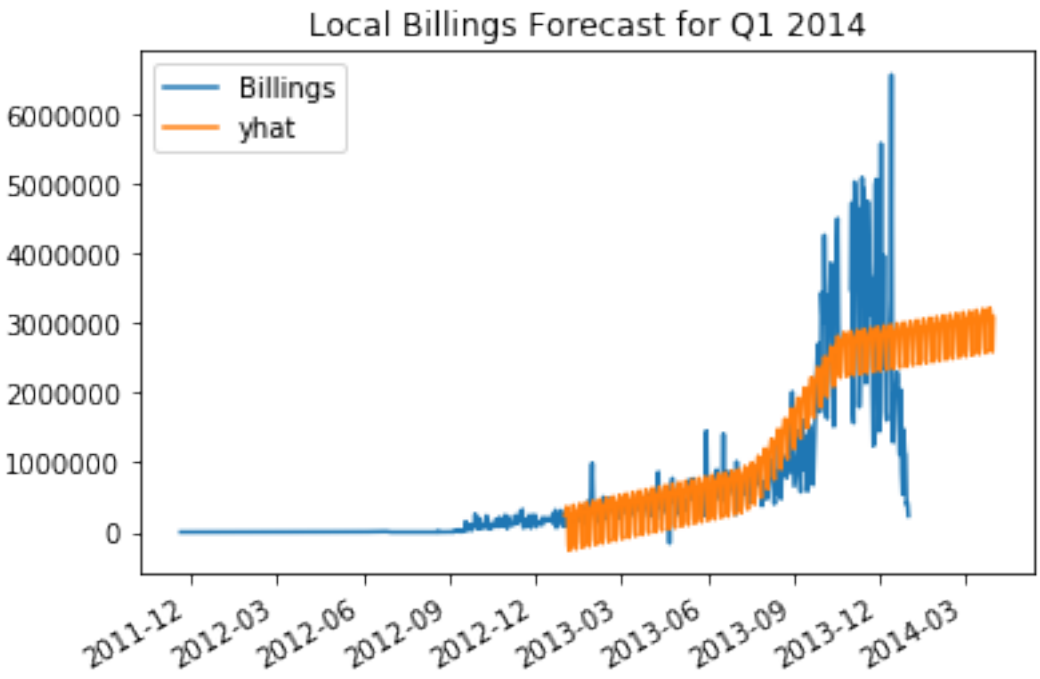

```
In [153]: model.plot_components(forecast)
```

```
Out[153]:
```


trend
3000000
2500000
2000000
1500000
1000000
500000
0
2013-01
2013-03
2013-05
2013-07
2013-09
2013-11
2014-01
2014-03
ds
weekly
200000
100000
0
−100000
−200000
−300000
−400000
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week
daily
1000
0
−1000
−2000
−3000
00:00:00
04:00:00
08:00:00
12:00:00
16:00:00
20:00:00
00:00:00
ds




```
In [154]: df = pd.concat([Local_Billings, forecast.set_index('ds')['yhat']], axis=1)

In [166]: df.plot(title= 'Local Billings Forecast for Q1 2014', fontsize=10)

Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x1c246c8a58>
```


Local Billings Forecast for Q1 2014
Billings
yhat
6000000
5000000
4000000
3000000
2000000
1000000
0
2011-12
2012-03
2012-06
2012-09
2012-12
2013-03
2013-06
2013-09
2013-12
2014-03