Course Project: Big Data Concepts

Name: Sai Teja Burla

Student ID: 2001058983

Imports

In [1]:
```python
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.functions import *
from pyspark.sql.functions import to_date
from pyspark.sql import SQLContext

import pymongo
from pymongo import MongoClient

import pandas as pd
import matplotlib.pyplot as plt

from prophet import Prophet
from prophet.plot import plot_plotly, plot_components_plotly
import plotly.offline as py
import plotly.graph_objects as go
```

Initialise a Spark Session

In [2]:
```python
spark = SparkSession.builder.appName("MStocks").getOrCreate()
```

Connection to MongoDB

In [3]:
```python
mongodb_url = "mongodb+srv://saiteja:L4KMI01na9a7h5WD@c1.c1bek3y.mongodb.net/test"
client = MongoClient(mongodb_url)
```

In [4]:
```python
db = client.get_database('Microsoft_Stock')
collection = db.get_collection('Dataset')
cursor = collection.find()
main_list = []
for document in cursor:
    main_list.append([document['Date'], document['Open'], document['High'], document['Low'], document['Close'], document['Volume']])
microsoft = pd.DataFrame(main_list, columns = ['Date', 'Open', 'High', 'Low', 'Close', 'Volume'])
```

In [5]:
```python
collection_train = db.get_collection('Train')
cursor = collection_train.find()
main_list = []
for document in cursor:
    main_list.append([document['Date'], document['Close']])
train_data = pd.DataFrame(main_list, columns = ['Date', 'Close'])
```

In [6]:
```python
collection_test = db.get_collection('Test')
cursor = collection_test.find()
main_list = []
for document in cursor:
    main_list.append([document['Date'], document['Close']])
test_data = pd.DataFrame(main_list, columns = ['Date', 'Close'])
```

Spark Dataframes for Full, Train and Test

In [7]:
```python
df = spark.createDataFrame(microsoft)
train = spark.createDataFrame(train_data.astype(str))
test = spark.createDataFrame(test_data.astype(str))
```

Schema for all dataframes

In [8]:
```python
df.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
```

In [9]:
```python
train.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Close: string (nullable = true)
```

In [10]:
```python
test.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Close: string (nullable = true)
```

Visualizing the main dataframe

In [11]:
```python
df.show()
```

```
+---------------+-----+-----+-----+-----+---------+
|           Date| Open| High|  Low|Close|   Volume|
+---------------+-----+-----+-----+-----+---------+
| 4/1/2015 16:00| 40.6|40.76|40.31|40.72| 36865322|
| 4/2/2015 16:00|40.66|40.74|40.12|40.29| 37487476|
| 4/6/2015 16:00|40.34|41.78|40.18|41.55| 39223692|
| 4/7/2015 16:00|41.61|41.91|41.31|41.53| 28809375|
| 4/8/2015 16:00|41.48|41.69|41.04|41.42| 24753438|
| 4/9/2015 16:00|41.25|41.62|41.25|41.48| 25723861|
|4/10/2015 16:00|41.63|41.95|41.41|41.72| 28022002|
|4/13/2015 16:00| 41.4|42.06|41.39|41.76| 30276692|
|4/14/2015 16:00| 41.8|42.03|41.39|41.65| 24244382|
|4/15/2015 16:00|41.76|42.46|41.68|42.26| 27343581|
|4/16/2015 16:00|41.95|42.34|41.82|42.16| 22509652|
|4/17/2015 16:00|41.67|41.74|41.16|41.62| 42387608|
|4/20/2015 16:00|41.73|43.17|41.68|42.91| 46057733|
|4/21/2015 16:00| 43.0|43.15|42.53|42.64| 26013844|
|4/22/2015 16:00|42.67|43.13|42.55|42.99| 25064273|
|4/23/2015 16:00|42.85|43.61| 42.8|43.34| 46309530|
|4/24/2015 16:00|45.66|48.14|45.65|47.87|130933665|
|4/27/2015 16:00|47.23|48.13|47.22|48.03| 59248172|
|4/28/2015 16:00|47.78|49.21| 47.7|49.16| 60730778|
|4/29/2015 16:00|48.72|49.31| 48.5|49.06| 47804562|
+---------------+-----+-----+-----+-----+---------+
only showing top 20 rows
```

Checking for Missing values

In [12]:
```python
def count_missings(df,sort=True):
    ndf = df.select([F.count(F.when(F.isnan(c) | F.isnull(c), c)).alias(c) for (c,c_type) in df.dtypes]).toPandas()

    if len(ndf) == 0:
        print("There are no any missing values!")
        return None

    if sort:
        return ndf.rename(index={0: 'count'}).T.sort_values("count",ascending=False)

    return ndf
```

In [13]:
```python
count_missings(df)
```

Out[13]:

	count
Date	0
Open	0
High	0
Low	0
Close	0
Volume	0

Changing Dates column type to date

In [14]:
```python
df.withColumn('Date', to_date('Date'))
```

Out[14]: `DataFrame[Date: date, Open: double, High: double, Low: double, Close: double, Volume: bigint]`

Converting the spark dataframe to pandas dataframe for Model Creation

In [15]:
```python
sqlcontext = SQLContext(spark)
```

```
/usr/local/spark/python/pyspark/sql/context.py:112: FutureWarning:

Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
```

In [16]:
```python
train = train.toPandas()
train['Date'] = pd.to_datetime(train['Date'])
train['Close'] = train['Close'].astype(float)
train.set_index('Date')
train
```

Out[16]:

	Date	Close
0	2015-04-01	40.72
1	2015-04-02	40.29
2	2015-04-06	41.55
3	2015-04-07	41.53
4	2015-04-08	41.42
...	...	...
1192	2019-12-24	157.38
1193	2019-12-26	158.67
1194	2019-12-27	158.96
1195	2019-12-30	157.59
1196	2019-12-31	157.70

1197 rows × 2 columns

In [17]:
```python
test = test.toPandas()
test['Date'] = pd.to_datetime(test['Date'])
test['Close'] = test['Close'].astype(float)
test.set_index('Date')
test
```

Out[17]:

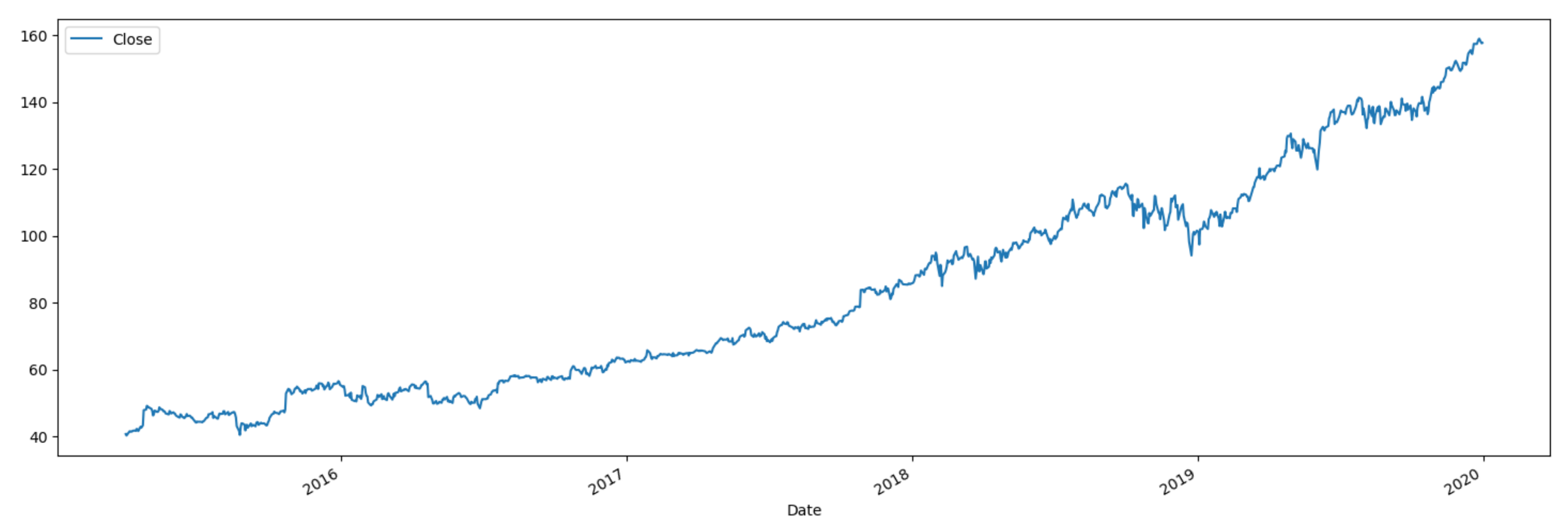
	Date	Close
0	2020-01-02	160.62
1	2020-01-03	158.62
2	2020-01-06	159.03
3	2020-01-07	157.58
4	2020-01-08	160.09
...	...	...
309	2021-03-25	232.34
310	2021-03-26	236.48
311	2021-03-29	235.24
312	2021-03-30	231.85
313	2021-03-31	235.77

314 rows × 2 columns

Visualizing the Train Data

In [18]:
```python
train.plot(x='Date', y='Close', figsize=(18,6))
```

Out[18]: `<Axes: xlabel='Date'>`



Renaming Train Column Names

We do this since we are going to make of a module called Prophet and that module only recognises these names for column names.

In [19]:
```python
train.columns = ['ds', 'y']
```

Prophet Model

In [20]:
```python
m = Prophet(yearly_seasonality=True)
m.fit(train)
future = m.make_future_dataframe(periods = 730)
forecast = m.predict(future)
```

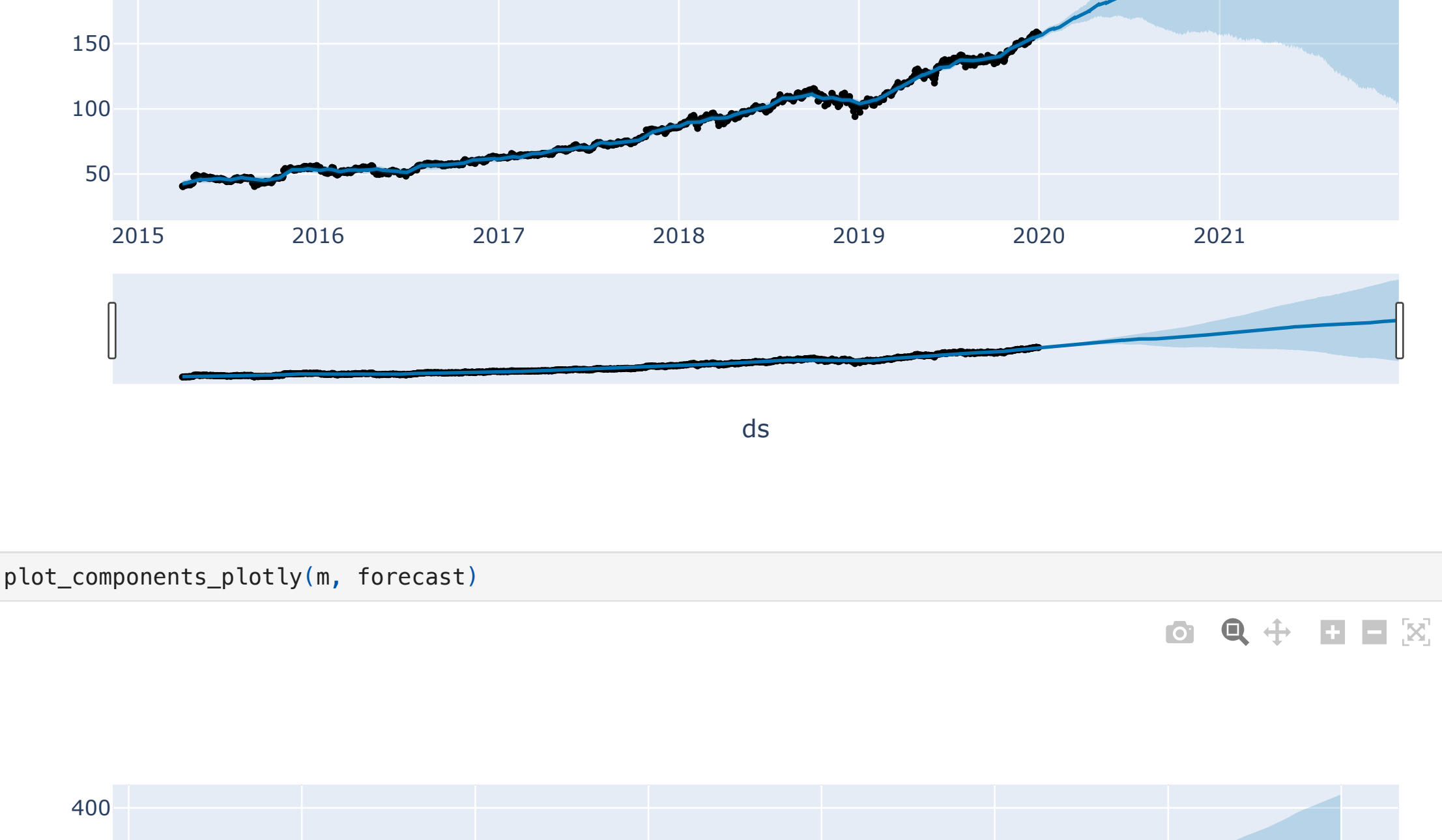
```
02:05:12 - cmdstanpy - INFO - Chain [1] start processing
02:05:13 - cmdstanpy - INFO - Chain [1] done processing
```

In [21]:
```python
forecast.head()
```

Out[21]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	weekly	weekly_lower	weekly_upper	yearly	yearly_lower	ye
0	2015-04-01	40.980206	39.904605	44.793344	40.980206	40.980206	1.431615	1.431615	1.431615	0.085557	0.085557	0.085557	1.346058	1.346058	
1	2015-04-02	41.001201	39.987824	44.908337	41.001201	41.001201	1.523028	1.523028	1.523028	0.150855	0.150855	0.150855	1.372173	1.372173	
2	2015-04-06	41.085180	40.289567	45.197853	41.085180	41.085180	1.609674	1.609674	1.609674	0.064705	0.064705	0.064705	1.544969	1.544969	
3	2015-04-07	41.106175	40.391785	45.299746	41.106175	41.106175	1.727544	1.727544	1.727544	0.120561	0.120561	0.120561	1.606983	1.606983	
4	2015-04-08	41.127170	40.314963	45.406599	41.127170	41.127170	1.762549	1.762549	1.762549	0.085557	0.085557	0.085557	1.676991	1.676991	

In [22]:
```python
py.init_notebook_mode()
fig = plot_plotly(m, forecast)
py.iplot(fig)
```



In [23]:
```python
plot_components_plotly(m, forecast)
```