

Final project on Big Data Analytics course in Harvard Extension course  
The topic name is “**Cassandra and Time Series analysis**”

Student name: **Galina Alperovich**

May, 2017

---

## Abstract

One of the most common use cases for Cassandra NoSQL is tracking time-series data. The reason for this is Cassandra’s data model which is an excellent fit for handling data in sequence regardless of datatype or size. Cassandra has highly fast writing speed, built-in replication across nodes and high availability with no single point of failure. Since NoSQL databases developed in a different way than traditional RDMS, you will not find ad-hoc quires for joining, grouping and other standard operations from SQL.

The purpose of this project is to demonstrate how Cassandra can be used for financial time series analysis and explain why it is natural for Cassandra to work with sequential data. Also we present lightweight web application where the user can select one of the 3000 US companies and play with the time series representing the stock data. The user will be able to draw the series, discover basic statistics, aggregate daily data in a different way and make forecast in real time. The application is fully developed in Python, with Cassandra NoSQL database for data storing, Bokeh framework for interactive Python visualization and Prophet framework for fast and automatic time series forecasting recently open-sourced by Facebook.

The web application is hosted on AWS and available online. Full step-by-step documentation from project start to deployment is provided. You can reproduce the same application from scratch.

Application URL: <http://34.202.35.11:5006/CassandraTime>

Application on GitHub: <https://github.com/galinaalperovich/CassandraTime/>

YouTube video (2 min): <https://youtu.be/9UKfyR77bG8>

YouTube video (15 min): <https://youtu.be/4VBh6UQd6z8>

# Cassandra time!

Final project on [Big Data Analytics course](#)

The topic: “Cassandra and Time Series”

Student name: **Galina Alperovich**

May, 2017

This is an example of small application on Time Series analysis with **Cassandra database** for data storing, **Bokeh framework** for visualization and **Prophet framework** for time series forecasting. As a data source **Open Wiki database of 3000 US companies** from Quandl web service has been used. Everything is implemented in Python.

This application is a final project on **Big Data Analytics course** in Harvard Extension school. The teacher is Zoran B. Djordjevic, student name Galina Alperovich.

1 step: Choose the company

Facebook, Inc.

Predict

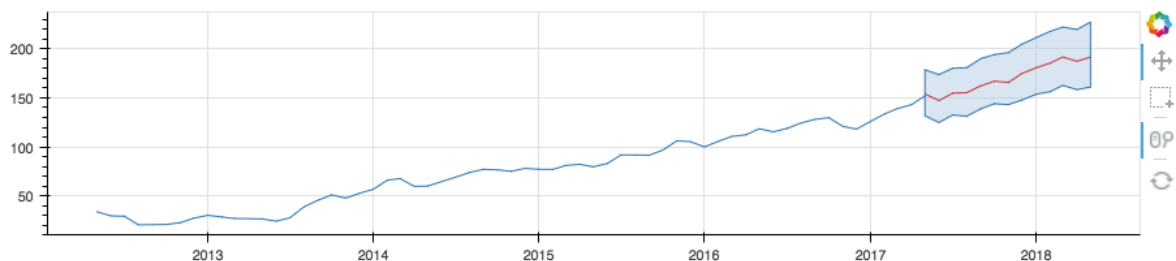
2 step: Choose the indicator

adj\_close

Click if you want to see the prediction for the time series

3 step: How to aggregate?

Monthly



## Summary statistics

#	count	mean	std	min	25%	50%	75%	max
0	61	76	38	20	39	77	106	152

## Original data

#	date	adj_close	adj_high	adj_low	adj_open	adj_volume	close	ex_dividen	high	low	openp	split_ratio	volume
0	2012 Ma...	33.7850...	34.5764...	31.5278...	32.2681...	1532603...	0	1	33.7850...	34.5764...	31.5278...	32.2681...	1532603...
1	2012 Ju...	29.5246...	30.2716...	28.8663...	29.5565...	3388354...	0	1	29.5246...	30.2716...	28.8663...	29.5565...	3388354...

# Table of Contents

<i>Why Cassandra for Time Series</i>	4
Patterns for time series data	4
<i>Shortly about keys</i>	7
<i>Web application specification</i>	8
<b>Installation</b>	<b>9</b>
<i>Install Cassandra</i>	9
<i>Install Python libraries</i>	9
<i>Install Python IDE</i>	10
<i>Download data</i>	10
<b>Cassandra queries</b>	<b>11</b>
<i>Create and activate keyspace</i>	11
<i>Create tables</i>	12
<i>Load csv files to tables</i>	13
<b>Web application</b>	<b>13</b>
<i>Run application locally</i>	13
<i>Discover the features</i>	15
Company name	16
Indicator name	16
Aggregation method	17
Time series graph	17
Summary statistics	18
Original data table	18
Prediction button	18
<b>What's inside the script</b>	<b>19</b>
<i>Quick explanation of main functions</i>	20
<b>Deployment to AWS</b>	<b>21</b>
<i>Prepare your EC2 instance</i>	21
<i>Install libraries</i>	25
<i>Run application remotely</i>	25
<b>Conclusion</b>	<b>26</b>
<b>Reference</b>	<b>26</b>

# Why Cassandra for Time Series

Cassandra NoSQL database is an excellent fit for time series data, it's widely used for storing the data that follows the time series pattern: logs, sensor data, financial data, performance metrics, user activity, and so on.

Writing the data to Cassandra is very fast, it can hold millions of rows per seconds (hundreds of rows comparing to SQL databases). When writing data to Cassandra, data is **sorted and written sequentially** to disk. Natural pattern for time series is retrieving data by some keys and then by range. With Cassandra you get a fast and efficient access due to minimal disk seeks.

## Patterns for time series data

Let's imagine we have financial data: **symbol name (for example 'AAPL' or 'GOOG')**, **timestamp** and **stock closing price**. We got used to imagine it as a normal 3 columns table.

symbol_id	timestamp	closing_price
'AAPL'	'2013-04-03 00:00:00'	156.3
'AAPL'	'2013-04-04 00:00:00'	157.8
....	....	....

The important thing is that when we create table in Cassandra, it keeps it in a very different manner, not 3 columns table. How the table looks and on which node of the cluster it is stored depend on how we defined PRIMARY KEY statement when we create the table. We will explain the concept of keys a bit later, now let's see how Cassandra keeps data.

### Pattern 1: Single stock symbol per row

Let's imagine we have financial data: **symbol name**, **timestamp** and **stock closing price**.

To create this kind of table in Cassandra you need such query:

```
CREATE TABLE stocks (  
  symbol_id text,  
  event_time timestamp,  
  closing_price float,  
  PRIMARY KEY (symbol_id, event_time)  
);
```

This query will create table which look as follows: one long row for every **symbol\_id**, so we will have very wide table with a row for every symbol and a column for every **timestamp**. The first key in the Primary Key clause is partition key, the second key is clustering key (we will discuss the difference later).

'AAPL'	'2013-04-03 00:00:00'	'2013-04-04 00:00:00'	....
	156.3	157.8	....
'GOOG'	'2013-04-03 00:00:00'	'2013-04-04 00:00:00'	....
	240.3	247.8	....
...	...	...	...

Every row is independent, so the timestamps may be different for different stock symbol. The reading can be done as follows:

```
SELECT * FROM stocks
WHERE symbol_id ='AAPL'
AND event_time > '2013-04-03 00:00:00'
AND event_time < '2013-04-05 00:00:00';
```

## Pattern 2: Partitioning

In the first case we had only one row for one **symbol\_id**. In case we have highly frequent data (for example we have the closing price for every second) we may exceed the number of columns (2 billions). In this situation we can add another key and have **one row for every pair**. If we exceeded again – we can add more keys and so one. This process is called **partitioning**. We just need to keep the number of (rows x columns) for one partition less than 2 billion, this is Cassandra's limitation.

Let's imagine we have similar stock data but for every seconds. Let's add a new column **date** and do partitioning by this column together with **symbol\_id**.

```
CREATE TABLE stocks (
  symbol_id text,
  date text,
  event_time timestamp,
  closing_price float,
  PRIMARY KEY ((symbol_id, date), event_time)
);
```

You see the change in PRIMARY KEY line? Now we have a new type of portion (**symbol\_id, date**) compounded with the two components. After this we will get a table with the row for every pair (symbol\_id, date) and columns for every timestamp (every second in our case).

	'00:00:01'	'00:00:02'	....
('AAPL', '2013-04-03')	156.3	156.9	....
	'06:00:01'	'06:00:01'	....
('AAPL', '2013-04-04')	157.0	156.4	....
...	...	...	...

Retrieving data from this table will look as follows:

```
SELECT *
FROM stocks
WHERE symbol_id ='AAPL'
AND date = '2013-04-03'
AND event_time > '00:00:00'
AND event_time < '12:00:00';
```

Now we see how naturally time series fits to the Cassandra data model. Cassandra is scalable (you can easily add new nodes to your cluster), it automatically replicates your data. All nodes are independent so even if one node dies the others work and database will never fail. Also the Cassandra's model allows to write and retrieve data exactly as time series needed.

## Shortly about keys

Let's discuss the concept of **primary, partition and clustering keys**. It's very important to understand since it is related to how Cassandra writes and reads the data.

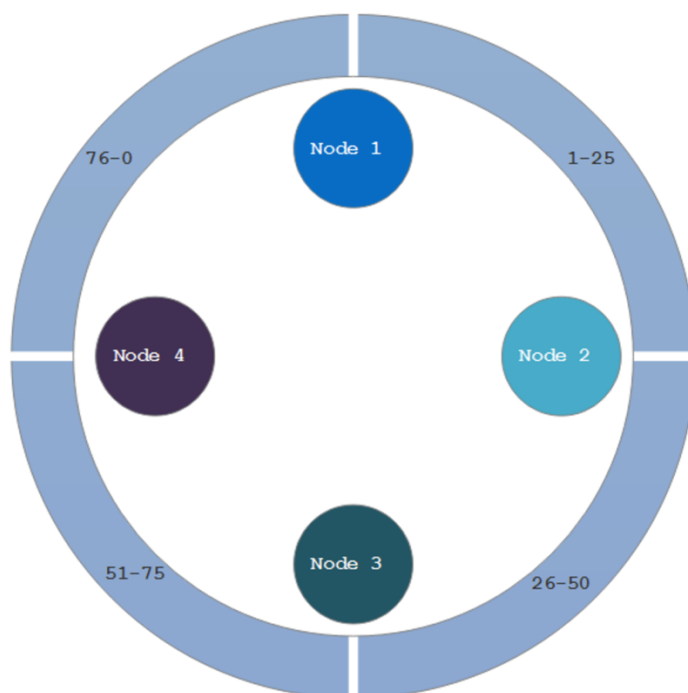
```
PRIMARY KEY ((symbol_id, date), event_time)
```

We see 3 different columns in a key clause. Here the definition for different parts of this statement:

**((symbol\_id, date), event\_time)** – all this structure is called **Primary key**. Multi-column primary key is called **Compound Key**, we have 2 elements of the key: the pair (symbol\_id, date) with two columns and one single column event\_time.

**(symbol\_id, date)** - **the first element** of Primary key is called **Partition key**. In our case the first element is a pair (symbol\_id, date). This element may be very complicated and consists of many columns. The partition key helps to choose the node in Cassandra's cluster to store associated data. Cassandra calculates the hash of this key and after that send to corresponding node. Every node corresponds to some hash value range.

For example, on the picture below you see the cluster with 4 nodes. First node corresponds to the hash value from 1 to 25, second for the hash from 26 to 50, and so one. So **Partition key** helps to keep related data on the same node.



**event\_time** - Each additional column that is added to the Primary Key clause is called a **Clustering Key**. A clustering key is responsible for sorting data within the partition. By default is sorted in ascending order, but you can change it when create the table:

```
CREATE TABLE stocks (  
  symbol_id text,  
  date text,  
  event_time timestamp,  
  closing_price float,  
  PRIMARY KEY ((symbol_id, date), event_time)  
)  
WITH CLUSTERING ORDER BY (event_time DESC);
```

Creating the Primary key is a tricky task and fully depends on **select** queries you need to do for the problem.

## Web application specification

As a practical example for the Final project we want to build small web application which satisfies the following requirements:

- Everything is written in Python
- Cassandra NoSQL as a database
- Big dataset of time series for analysis
- Interactive plots for visualization
- Features: visualize time series, show summary statistics, show original data, ability to aggregate daily data to monthly, quarterly, annually.
- Forecast the time series in real time.

The rest of this document is devoted to step-by-step instruction how to implement such application from very beginning to deployment on AWS.



# Installation

## Steps:

1. Install Cassandra
2. Install needed Python libraries
3. Install Python IDE
4. Download data

## Install Cassandra

If you have brew installer, it's easy to install Cassandra. Insert the following commands to console:

```
pip install cql
brew install cassandra
cp /usr/local/Cellar/cassandra/<VERSION>/homebrew.mxcl.cassandra.plist
~/Library/LaunchAgents/
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.cassandra.plist

cqlsh
```

If everything is OK you will see cqlsh console. If not, try to read [official documentation](#).

## Install Python libraries

The following libraries we will actively use:

- **cassandra-driver** – for executing queries for Cassandra from Python
- **bokeh** – [Bokeh](#) framework for interactive Python visualization
- **pandas and numpy** – for working and manipulation with data
- **fbprophet** – [Prophet](#) is a superfast framework for automatic time series forecasting.

These libraries can be installed with the standard pip installer. Also it's highly recommended to install [conda](#), [create separate virtual environment](#) to keep it clean and install all possible libraries with conda, not pip.

```
pip install cassandra-driver
pip install bokeh (conda has it)
pip install numpy (conda has it)
pip install pandas (conda has it)
pip install fbprophet #this command will take a while
```

# Install Python IDE

I was using the [PyCharm Community Edition IDE](#). It is free and very powerful.

## Download data

Steps:

1. Register on [Quandl web service](#), it is a service which provides with many different time series data. There are both free and paid datasets. We are using free dataset.
2. Download [Wiki Price dataset](#). It is a collection of time series stock data from 3000 US companies for 40 years and until now.

The screenshot shows the Wiki Prices website interface. At the top, there's a dark blue header with the 'WIKI Prices' logo and navigation links for 'DATA' and 'DOCUMENTATION'. Below the header, there's a table of stock data with columns: TICKER, DATE, OPEN, HIGH, LOW, CLOSE, VOLUME, EX-DIVIDEND, SPLIT\_RATIO, and ADJ\_OPEN. The table lists data for ticker 'A' from 1999-11-18 to 1999-12-16. To the right of the table, there's a sidebar with a 'BULK Download - Export the full database' section. This section includes a 'REQUEST' dropdown menu with options for 'CSV', 'JSON', and 'XML'. Below this is a 'GENERATE .ZIP FILE' button. A note at the bottom of the sidebar states: 'NOTE: You can filter the data table to export smaller slices of this database.'

3. Unzip the archive, you will get the datasets around **1.8 GB**.
4. Rename the file to **WIKI\_PRICE.csv**

Now we have original dataset with stocks. One additional dataset we will need is a table with correspondence between symbol name (also called the ticker, for example 'AAPL') and the official company name ('Apple Inc.').

Steps:

1. We will download the Excel file from [this link](#). To download the file you can just click [this link](#). This file is the information about 158 000 Yahoo stock ticker symbols.
2. Open the downloaded file **Yahoo Ticker Symbols - Jan 2016.xlsx**
3. Select and copy 'ticker' and 'name' columns from the first sheet 'Stock' and paste them to the new text file.

#### 4. Save the file as **Tickers.csv**

We will need the company name for usability purposes. It's not obvious which symbol name corresponds to which company. So we will give the ability to choose company name rather than the symbol name. After converting **company name** → **ticker name** we will get time series from our database.

So now we have two data files: **WIKI\_PRICES.csv** and **Tickers.csv**. Also we have all libraries and tools being installed for further work.

## Cassandra queries

We will have to execute several types of query for Cassandra:

1. Create and activate keyspace.
2. Create two tables, one for Wiki Prices and another for Tickers info.
3. Load corresponding csv files to newly created tables

Run **cqlsh** command in your terminal. All queries from this section will be made there.

### Create and activate keyspace

1. **This query creates our working keyspace:**

```
CREATE KEYSPACE wiki_price_keyspace  
WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': '2' };
```

2. **This query use created keyspace:**

```
USE wiki_price_keyspace;
```

## Create tables

3. This query creates table for Wiki Prices. You can see the structure of the main table:

```
CREATE TABLE wikiprice (  
    ticker text,  
    date timestamp,  
    openp float,  
    high float,  
    low float,  
    close float,  
    volume float,  
    ex_dividend float,  
    split_ratio float,  
    adj_open float,  
    adj_high float,  
    adj_low float,  
    adj_close float,  
    adj_volume float ,  
    PRIMARY KEY (ticker, date));
```

**Blue columns** are the main ones because we will execute select queries by them, that's why we put them as a Primary key. **ticker** is a symbol of the stock, **date** is a date stamp corresponding to the stock indicators. **Green columns** are different indicators which we will analyse. The most interesting is **adj\_close**. More about this and others financial indicators you can read for example [here](#).

We create primary key with the ticker and date because in future we will do queries by these fields. Partition key is a **ticker**, clustering key is a **date**. In the first section we explained what's the difference between them.

4. This query creates table for Tickers info.

```
CREATE TABLE tickers (  
    ticker text,  
    name text,
```

```
PRIMARY KEY (ticker)
```

```
);
```

## Load csv files to tables

These two queries will take some time, about 5-10 minutes because there are a lot of rows in WIKI\_PRICE.csv file (several millions) and Tickers.csv (27K rows).

### 5. This load data from Tickers.csv file to tickers table:

```
COPY tickers FROM 'Tickers.csv' WITH HEADER = true AND DELIMITER='\t';
```

### 6. This query load data from WIKI\_PRICES.csv file to wikiprice table:

```
COPY wikiprice FROM 'WIKI_PRICES.csv' WITH HEADER = true;
```

This query will take some time to execute.

After these 6 queries we will have two Cassandra tables: **wikiprice** table with all information about stocks and **tickers** table with information about company name. Now we are ready to retrieve data from them. In further sections we will do it from Python, not from cqlsh.

## Web application

### Run application locally

If you downloaded the project folder, you will find the folder CassandraTime/, it has the following structure:

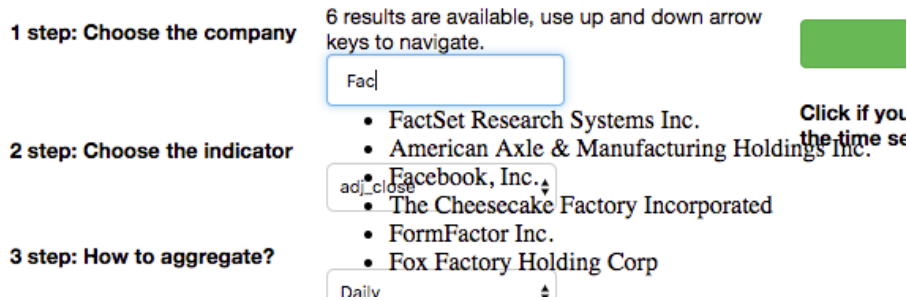
**CassandraTime/**

- **app.py**
- **description.html**
- **utils.py**
- **awesomplete\_input.coffee**
- **awesomplete\_input.py**

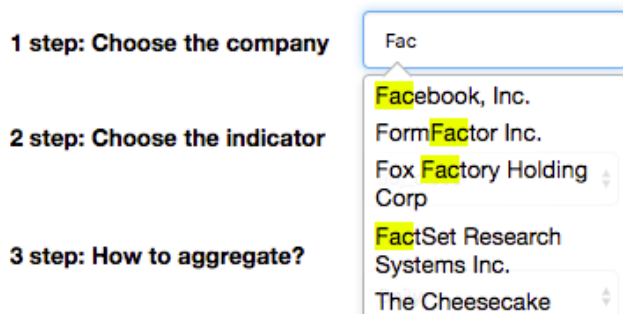
**awesomeplete\_input** - related files (.coffee and .py) are taken from this [GitHub thread](#). It is an add-on for nice auto-completion for Bokeh Input widget. Original widget is pretty ugly.

**Just compare:**

Standart **AutocompleteInput** :(



Custom **AwesomepleteInput** from [one GitHub user](#) :)



**app.py** is a main file which we will run  
**description.html** is made for task description text  
**utils.py** – custom useful functions

To run the application locally you need to type the following query in you terminal (in the project's directory):

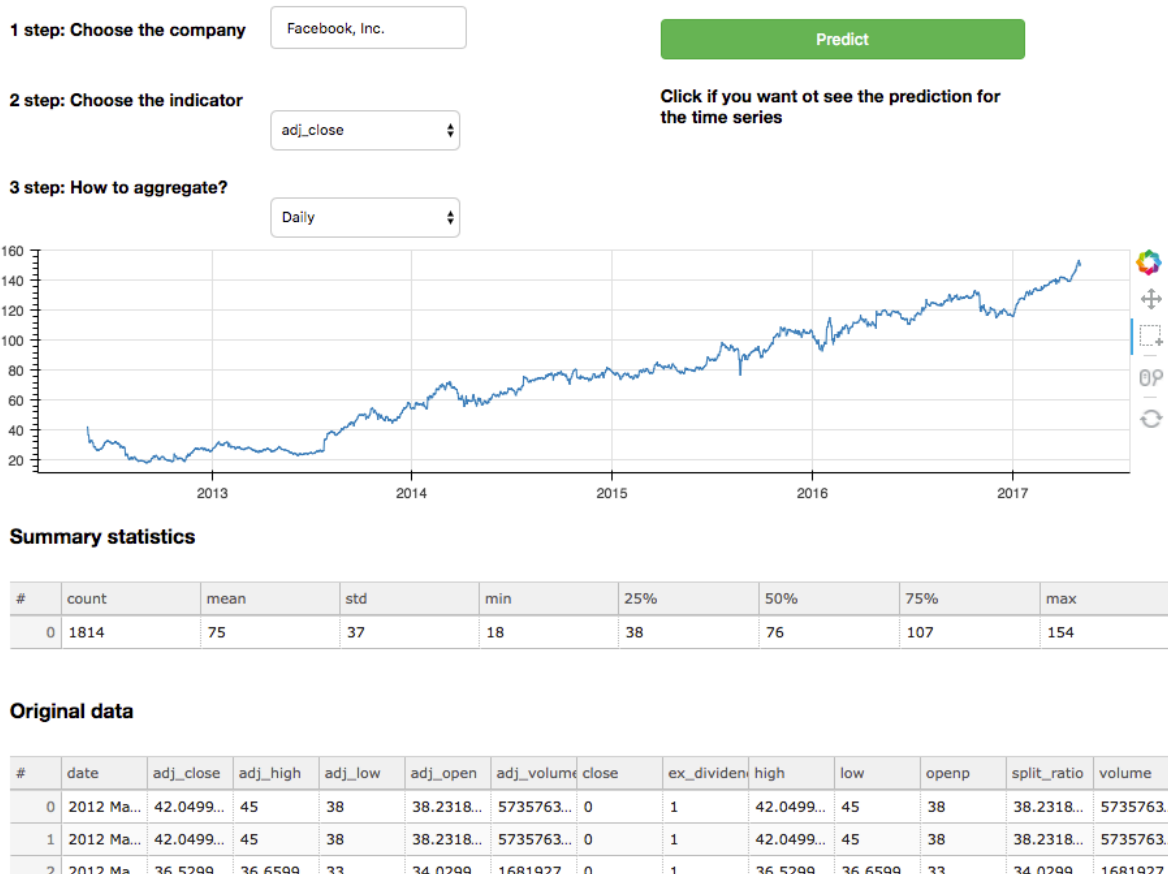
```
bokeh serve --show app.py
```

If everything goes OK you was navigated to the browser and you are able to see the page 'Stocks' by URL <http://localhost:5006/app>

## TIME SERIES ANALYSIS WITH CASSANDRA

This is an example of small application on Time Series analysis with **Cassandra database** for data storing, **Bokeh framework for visualization** and **Prophet framework** for time series forecasting. As a data source **Open Wiki database of 3000 US companies** from Quandl web service has been used. Everything is implemented in Python.

This application is a final project on **Big Data Analytics course** in Harvard Extension school. The teacher is Zoran B. Djordjevic, student name Galina Alperovich.



## Discover the features

Before we move on to the code structure, let's play a bit with the app and see what it can do. The page consists of **6 main sections**:

1. Title and description of the task
2. Three selectors: for company name, indicator name and aggregation method (there are default values but you can change it)
3. Button 'Predict'
4. Time series plot
5. Summary statistics table
6. Original data table for the company

Let's explore the behavior of interactive parts of the page.

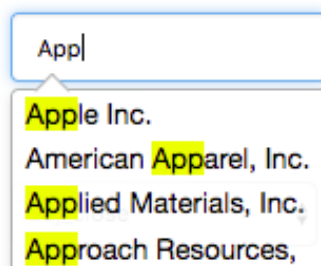
## Company name

The first thing is to select the **company name**. The default value is Facebook Inc. Try to play with auto-completion input widget and choose different company name. Don't be surprised if you don't find the company you want, the list of companies is limited (there are 3000 US companies).

**Effect on selection change:** when you select another company and click somewhere on the page (artefact from Bokeh) or press Enter, you will see that the data is updated on the graph, summary statistics and Original data table.

**1 step: Choose the company**

**2 step: Choose the indicator**



Updated values come exactly from **Cassandra**! The query is generated on the backend and executed after you change the selection.

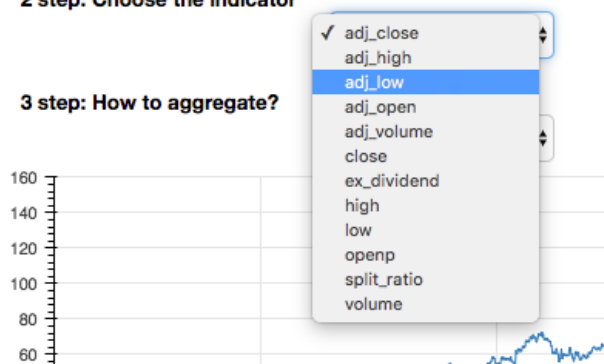
## Indicator name

Select an **indicator** from the dropdown list. Basically this is column names from wikiprice table. Default value is adj\_close, you can choose another indicator.

**Effect on selection change:** when you select another indicator, you will see that the data is updated on the graph and summary statistics. Original data table stays the same because it corresponds to company name.

**2 step: Choose the indicator**

**3 step: How to aggregate?**





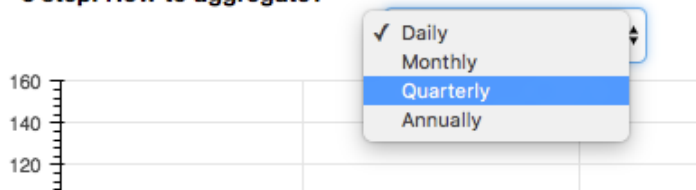
## Aggregation method

You may see stock data aggregated daily, monthly, quarterly or annually. It is easy to do because pandas Series data structure handles it by the nature.

```
ts.resample('D') .fillna('nearest')
ts.resample('AS') .fillna('nearest') # 'S' means start from beginning of year
ts.resample('MS') .fillna('nearest')
ts.resample('QS') .fillna('nearest')
```

**Effect on selection change:** when you select aggregation method, you will see that the data is updated on the graph and summary statistics. Original data table stays the same.

### 3 step: How to aggregate?



## Time series graph

It is an interactive time series graph. You can zoom and move the series in order to see smaller intervals and navigate across the time axes. This widget is not 100% intuitive but after some time you get used to it and it's OK. We were choosing between Bokeh and Plotly and finally decided to choose Bokeh because of this nice widget.



## Summary statistics

It's simple summary statistics which describe your data. They include count, mean, standard deviation, min, max and 3 quartiles.

#	count	mean	std	min	25%	50%	75%	max
0	1814	75	37	18	38	76	107	154

It came from pandas:

```
data.describe()
```

## Original data table

It is original data loaded after you choose the company name. You may see there all indicators and corresponding date. It's possible to sort values by clicking on the header of the table.

#	date ▼	adj_close	adj_high	adj_low	adj_open	adj_volume	close	ex_dividen	high	low	openp	split_ratio	volume
1811	2017 Ma...	153.600...	153.600...	151.339...	151.800...	21755088	0	1	153.600...	153.600...	151.339...	151.800...	21755088
1810	2017 Ma...	153.339...	153.440...	151.660...	152.779...	21255364	0	1	153.339...	153.440...	151.660...	152.779...	21255364
1809	2017 Ma...	151.740...	152.570...	151.419...	152.460...	24669160	0	1	151.740...	152.570...	151.419...	152.460...	24669160
1808	2017 Apr...	151.740...	152.570...	151.419...	152.460...	24669160	0	1	151.740...	152.570...	151.419...	152.460...	24669160
1807	2017 Apr...	149.5	151.529...	149.070...	150.25	30280372	0	1	149.5	151.529...	149.070...	150.25	30280372
1806	2017 Apr...	149.5	151.529...	149.070...	150.25	30280372	0	1	149.5	151.529...	149.070...	150.25	30280372
1805	2017 Apr...	146.669...	147.75	146.139...	147.699...	10561420	0	1	146.669...	147.75	146.139...	147.699...	10561420
1804	2017 Apr...	147.089...	147.589...	146.089...	146.559...	11324335	0	1	147.089...	147.589...	146.089...	146.559...	11324335
1803	2017 Apr...	145.789...	147.149...	145.789...	146.490...	17684832	0	1	145.789...	147.149...	145.789...	146.490...	17684832
1802	2017 Apr...	144.960...	145.673...	144.339...	145.470...	14397448	0	1	144.960...	145.673...	144.339...	145.470...	14397448

## Prediction button

This is the most interesting feature here. As you may guess if you click on it you will see the forecasting for current company name, indicator time series and aggregated with the current method. For daily data it takes several seconds to predict, so please wait a little bit. If prediction is finished you will see the red line on the graph, also you will see the confidence corridor.

1 step: Choose the company

Predict

2 step: Choose the indicator

Click if you want ot see the prediction for the time series

3 step: How to aggregate?



You can select different series and try to predict their future :)

Forecasting is made with very cool library [Prophet](#) developed by core data scientist team from Facebook, it was recently open-sourced. As they said on the webpage

*“It is based on an additive model where non-linear trends are fit with yearly and weekly seasonality, plus holidays. It works best with daily periodicity data with at least one year of historical data. Prophet is robust to missing data, shifts in the trend, and large outliers.”*

### The main benefits of Prophet are:

- It's really fast, we have both fitting and prediction for every time series in real-time. And even for daily data for the last 40 years it is very fast.
- It finds the model automatically. Usually to predict the time series you need to find the best model with trial and errors. We were trying to implement simple model selection with **statsmodels** library but it is very slow and not accurate. Also custom ARIMA models take more time for fitting.
- It's extremely easy to use. Basically several lines of code and you have nice prediction with the confidence interval for a new time series.
- The prediction is quite accurate (compared and run on past data).

On this stage you already know the features of the application and you guess how it works inside.
---

## What's inside the script

Our primary file would be **app.py**. Also some functions would be in **utils.py**.

If you quickly read the script from top to bottom, you will see the following structure:

1. Imports
2. Constants which we use across all methods
3. Small methods for updating/loading/etc
4. Handlers which track if the chooser changed the value (company name, indicator or aggregator)
5. Data sources for interactive widgets
6. Choosers for company name, indicator or aggregator
7. Sections for description, prediction button, plot and tables
8. Layout building

Several principles of interactive widgets with [Bokeh](#):

1. There are many elements which you can insert into layout: plots, tables, input text, choosers, tabs, etc. All of them have similar properties. All of them have callback method which takes the function that does (updates) something when the value is change. We have callbacks which track if any of the chooser changed its value (3 handlers)
2. Popular way how to pass new data to interactive widget and update how it looks is using the **source**. You can think about it as a box which is linked with the widget and where you can take and put new data. After data in the source is updated the widget (plot/table/etc) is changed. We use ColumnDataSource for every interactive parts (time series graph, summary table, original table)
3. One function which updates the graph is a bit longer than others. It is “Predict” button handler. After that we update the label “Please wait”. When we get the data and update the graph we change that label to “<empty>”.
4. Text fields on the layout is simply made with html file, for example **description.html**.
5. As we said before we use AwesomepleteInput widget in order to get nice and clean completion for a company name.

Try read the code starting from the part where data sources are created. When you get to the point where we create the callbacks for every chooser it’s easy to read:

```
company_chooser.on_change('value', company_change_handler)
indicator_chooser.on_change('value', indicator_change_handler)
aggregator_chooser.on_change('value', aggregator_select_handler)
```

After that just read what are `company_change_handler()`, `indicator_change_handler()` and `aggregator_select_handler()` methods. All other methods are invoked from there.

## Quick explanation of main functions

### **get\_cassandra\_session()**

Returns Cassandra session object. The result after execution of queries would be automatically converted to pandas DataFrame with the help of pandas factory  
`session.row_factory = pandas_factory`

### **load\_ticker\_data(ticker)**

Load data from Cassandra. Ticker parameter is a stock symbol. Returns the pandas DataFrame with all available indicators time series.

### **get\_ticker\_by\_company(company\_name)**

Convert symbol to company name.

**predict(series)**

It takes pandas Series and return DataFrame with prediction. It uses Prophet framework for automatic prediction. Firstly it fits the data, then it makes the prediction.

## Handlers

Handlers are listening changes from selectors and buttons.

```
company_change_handler()  
indicator_change_handler()  
aggregator_select_handler()
```

```
update_button_predict()
```

After the click on this button you see the prediction. This prediction made in this function. It calls predict function from utils.py

```
future = predict(current_series)
```

# Deployment to AWS

## Prepare your EC2 instance

Steps:

1. Create AWS account
2. Create EC2 instance

### Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.


**Launch Instance**

3. Choose **Bitnami Cassandra** from the AWS Marketplace

## Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications user community, or the AWS Marketplace; or you can select one of your own AMIs.

Quick Start
My AMIs
AWS Marketplace
Community AMIs
Categories
All Categories
Software Infrastructure (7)
Business Software (6)
Operating System
Clear Filter



**Cassandra powered by Bitnami**


★★★★★ (0) | 3.9-1-r28 on Ubuntu 14.04.3 [Previous versions](#) | Sold by [Bitnami](#)

**\$0.00/hr for software + AWS usage fees**

Linux/Unix, Ubuntu 14.04.3 | 64-bit Amazon Machine Image (AMI) | Updated:

Bitnami Cassandra is a pre-configured, ready to run image for running open source distributed database management system designed for

[More info](#)



**Azul Zing on RHEL**

★★★★★ (0) | 2016Q3 [Previous versions](#) | Sold by [Azul Systems](#)

**\$10.00/mo + \$0.269 to \$3.468/hr for software + AWS usage fees**

- Choose t2.large instance, it's minimal possible instance to install and run the application.

<input type="checkbox"/>	General purpose	t2.small	1	2	
<input type="checkbox"/>	General purpose	t2.medium	2	4	
<input checked="" type="checkbox"/>	General purpose	t2.large	2	8	
<input type="checkbox"/>	General purpose	t2.xlarge	4	16	
<input type="checkbox"/>	General purpose	t2.2xlarge	8	32	

- Finish creating the instance by clicking “Review and launch”
- Be sure that you save **.pem file** for accessing the EC2 instance through SSH.

Select an existing key pair or create a new key pair

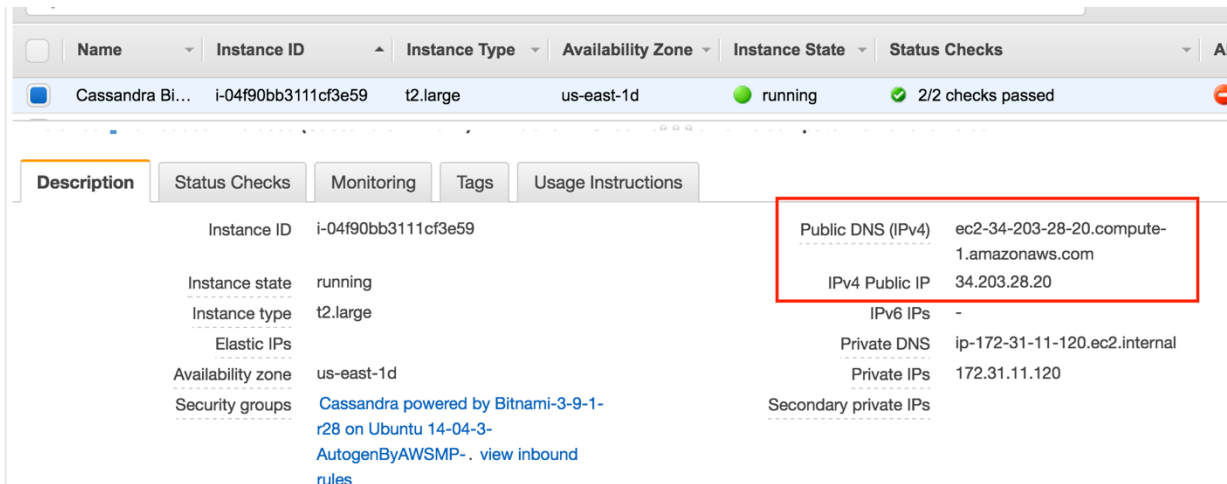
A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair

Key pair name
my\_pair
Download Key Pair

7. Your EC2 instance is ready! Look at its public DNS (or public IP) and save it somewhere, we will need it later.



8. Move .pem file to the folder which is one level higher of the project folder CassandraTime/ and go there. Put csv files (WIKI\_PRICES.csv and Tickers.csv ) to CassandraTime/ folder as well.

9. SSH login to your EC2 instance:

```
ssh -i bitnami_cassandra.pem bitnami@34.202.35.11
```

After @ you put your public DNS or public IP. bitnami\_cassandra.pem is a key file.

10. After you checked the connection is OK, logged out and copy WIKI\_PRICES.csv, Tickers.csv and project folder CassandraTime to the instance. You can do it with the following query:

```
scp -r -i bitnami_cassandra.pem CassandraTime/ bitnami@34.202.35.11
```

After this command all your code and data will be on EC2 instance.

11. To login to Cassandra on your remote EC2 instance, you need randomly generated password after Bitnami has started. You can find it in your instances dashboard (you will need it only one time):





15. After you created all tables and populated them from csv files, we can install all requirements on EC2 instance from “Installation” section.

## Install libraries

Installing requirements on EC2 was a bit more complicated for me than on my local machine. I have Mac OS, and there is Linux on EC2. Some libraries I already had and Linux didn't. You might need additional queries (depending in your OS and packages you already have). Here is a full list, please follow the order.

1. [Install Miniconda](#)
2. `sudo apt-get install python3-tk`
3. `sudo apt-get install tk-dev`
4. **`conda install gcc`**
5. `conda install pandas`
6. `conda install numpy`
7. `conda install bokeh`
8. `pip install cassandra-driver`
9. `pip install pystan`
10. `pip install fbprophet`
11. **`conda install -c bokeh nodejs`**

After you installed all requirements, you would be able to run the application.

## Run application remotely

### Steps:

1. Rename app.py to main.py (it is necessary since we will start the application in a bit different way, other way didn't work). Be sure you are in the project folder  
CassandraTime/
2. Run command in a terminal

```
bokeh serve . --allow-websocket-origin=34.202.35.11:5006
```

Insert your public IP to this query.

After that you can navigate to **34.202.35.11:5006/CassandraTime** and see the application (replace to your public IP address).

# Conclusion

In the Final project we showed an example of the application which executes time series analysis and uses Cassandra NoSQL database. Also we explained the basics of Cassandra and why this database is a good choice for time series application.

If you have any question regarding this work, please contact [galya.alperovich@gmail.com](mailto:galya.alperovich@gmail.com)

# Reference

- [1]. [Article “Why Apache Cassandra for time series?”](#)
- [2]. [Bokeh documentation on server creation](#)
- [3]. [Article “Top open sourced databases for time series”](#)
- [4]. [Article “Cassandra Partitioning and Clustering keys explained”](#)
- [5]. [Advanced time series with Cassandra](#)
- [6]. [Comparison table of open sourced databases for time series](#)