SKANNER

Year 4 Project Technical Specification

Author: Mateusz Koltun
Student ID: 11366981
Course: CASE4

Project Supervisor: **Yvette Graham**

Completion Date: 2019-05-29

Table of Contents

1. Glossary

- GUI - Graphical User Interface.
- MVC - Model-View-Controller is an architectural pattern.

2. Motivation

The ability to predict the change of stock market share prices is a crucial skill in stock investment. People attempt to do this on a daily basis since the beginning of the stock market. Stock changes are volatile due to the amount of factors and variables that need to be considered.

This application intends to assist the user with decisions of whether to invest in a particular stock. This is done by sentiment analysis on articles regarding that stock/company, which is indirectly correlated with how that stock/company will do based on people's opinions and thus, recommend whether to invest or not.

3. Design

The architectural structure of the application follows strictly Model-View-Controller (MVC) design pattern. This provides highly decoupled components, making it easier to maintain them. The data (models) is managed by controllers and displayed in views. It's a commonly used pattern when designing user interfaces in web development. Skanner, on the other hand is a pure Java application, built on top of JavaFX framework, which fully supports MVC architecture. JavaFX is an extension of Java Swing GUI library.

The Figure 2 below, shows the architecture of the application and how its modules are decoupled from each other. Each module has its own components. The model module, in general, is responsible for data manipulation. The view model is for displaying this data and finally, the controllers act like an interface between the two.

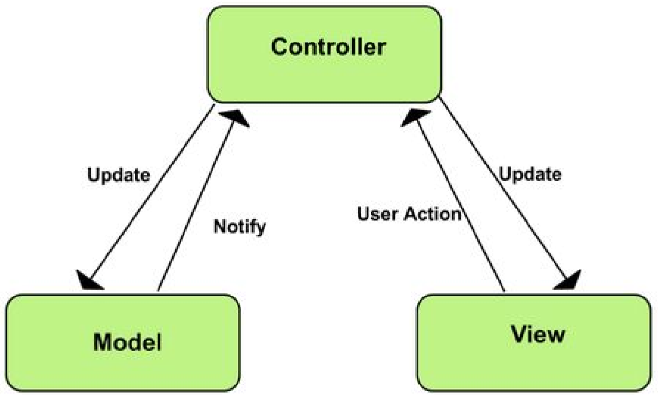


Figure 1. Model-View-Controller pattern.

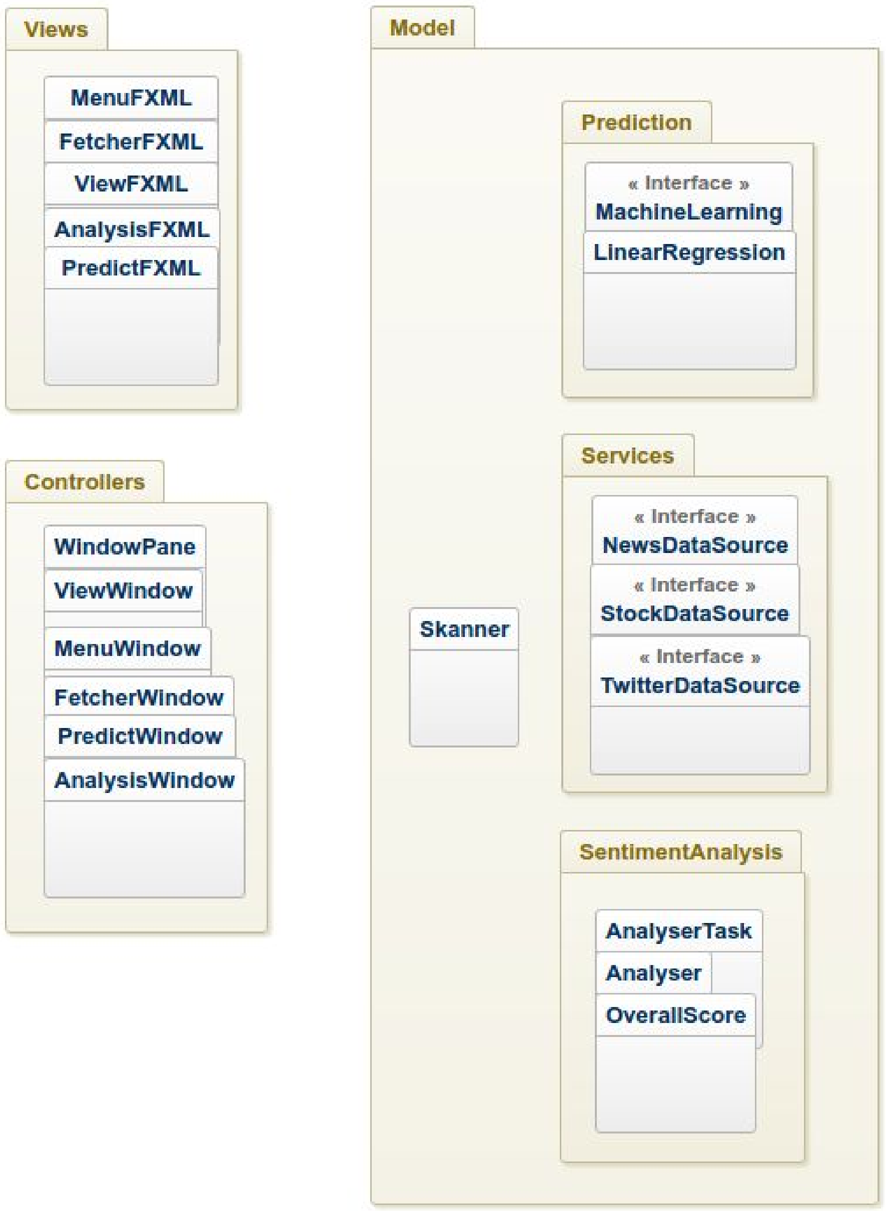


Figure 2. Model-View-Controller architecture

Skanner had been designed and developed as per functional specification. Numerous interfaces are used to obtain data which is then processed in order to derive the final result. Figures 3 and 4 show the flow of the data. External services are used to obtain articles, blogs, Tweets and historical stock data. Once data is obtained, the program pushes it via sentiment analysis and machine learning components.

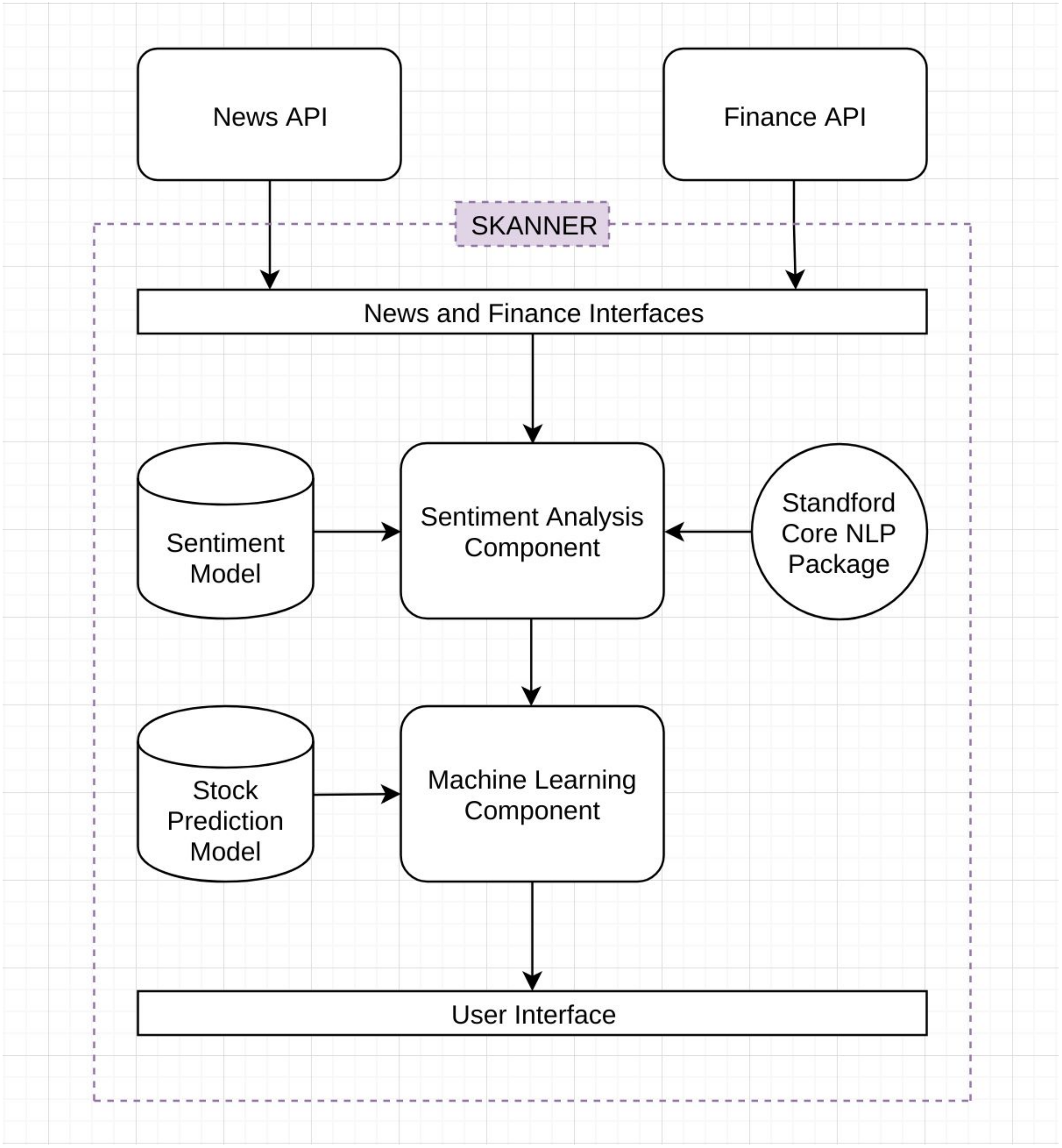

News API
Finance API
SKANNER
News and Finance Interfaces
Sentiment Model
Sentiment Analysis Component
Standford Core NLP Package
Stock Prediction Model
Machine Learning Component
User Interface

Figure 3. System Architecture Diagram

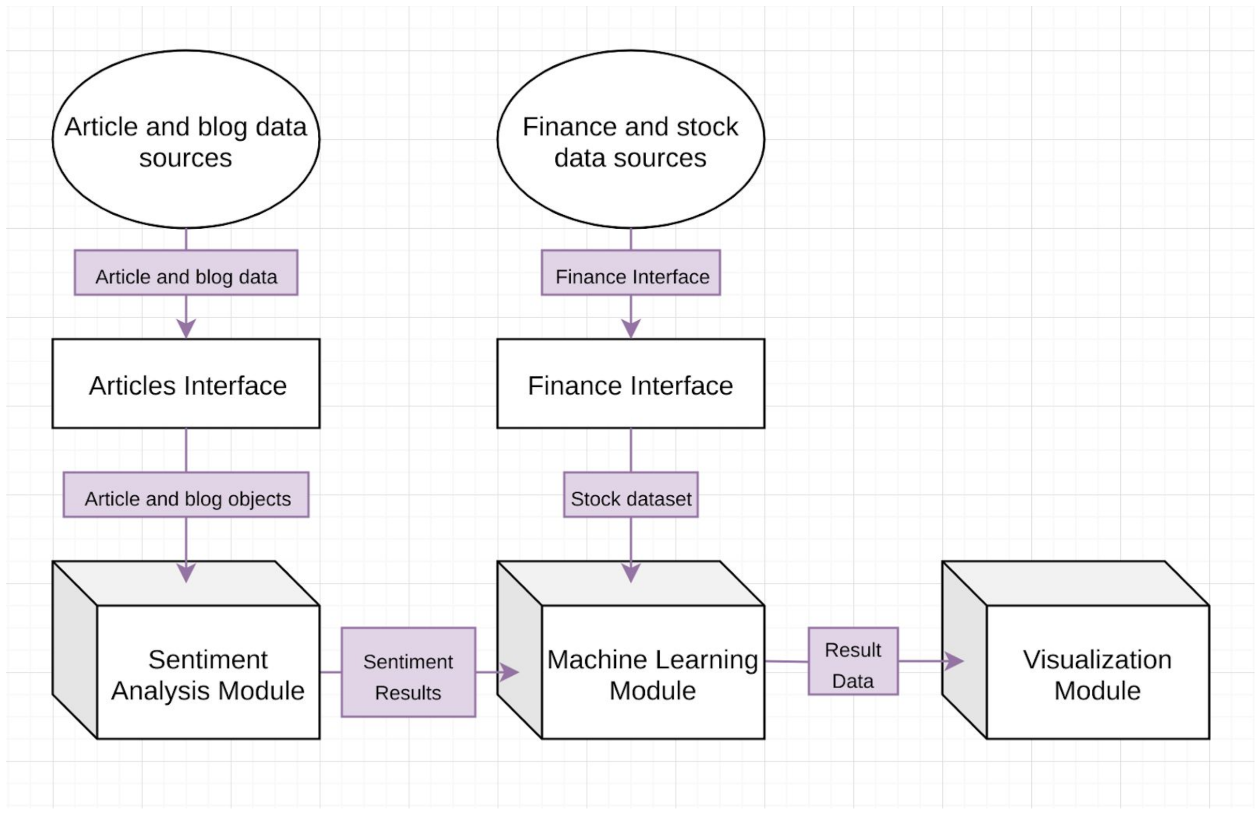


Figure 4. Data Flow Diagram

In order to develop highly decoupled system, various design patterns and software engineering techniques were used, such as:

- Singletons
- Observers
- Callbacks
- Object Oriented principles
    - Inheritance
    - Abstraction
    - Interfaces

Following the above principles allowed for clearer code, which is easier to understand and maintain. In the Figure 5, we can see the relationship between major classes. Log, Config and Skanner are singleton objects. Only one instance of each is created and are used in various ways. Log is used for logging purposes and most of the classes uses it. Config on the other hand is used by primary objects, such as Skanner, Services and some of the Window classes. However, it's important for it to be a singleton, because this way it ensures that the same configuration is used across the app. Skanner is the most important class in this diagram, it implements interface which is used by Controllers (Window objects) to perform computation and other manipulation on data. It's also crucial for it to be a singleton, because it's the class which has an access to data.

Observers are mainly used to update GUI when any changes are made to data. Article and Tweet feeds in the View Window use observers to update the list views accordingly.

Callbacks is another software engineering technique used commonly with asynchronous tasks. In this case it's not different. Services in Skanner are acting like async jobs in most cases, and callbacks where necessary to update fields and attributes of other classes upon finishing their job.

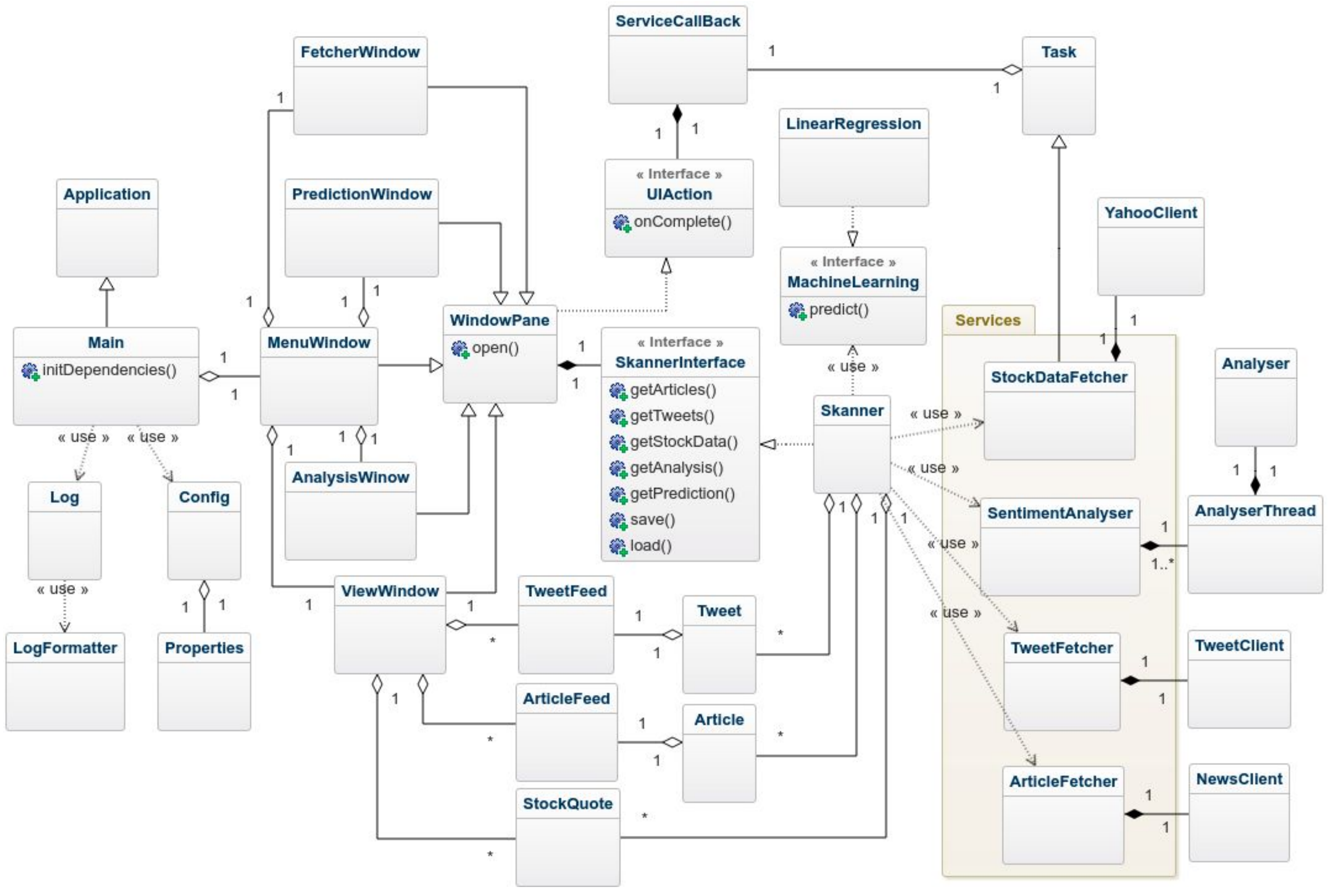


Figure 5. Class Diagram

4. Implementation

Graphical User Interface

As mentioned before, JavaFX framework was used to implement GUI. It consists of five in total separate windows (stages).

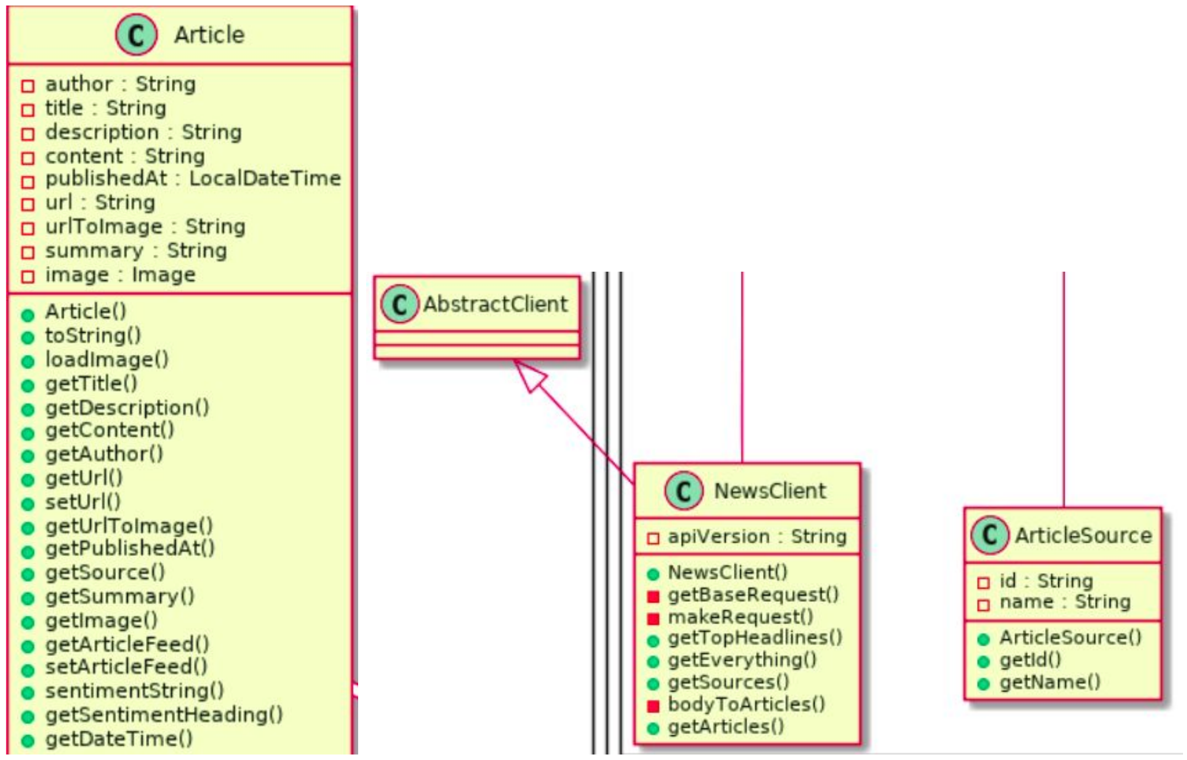
1) Menu Window - allows to choose a function for the application to execute.

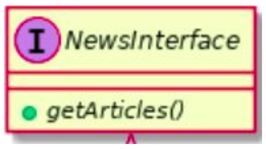
2) Fetch Window - contains parameters used to specify how much of data should be obtained.
3) Analysis Window - small information window to update the progress on analysis.
4) View Window - is used primarily to display obtained data.
5) Prediction Window - place where final result is shown.

Data source

News Client

Three different types of data is obtained by Skanner. First are news, articles and blogs. I had to implement from scratch REST client for NewsAPI service (https://newsapi.org/) because they don't provide one for Java. HTTP calls are powered by Unirest library. Following classes are included in this package.

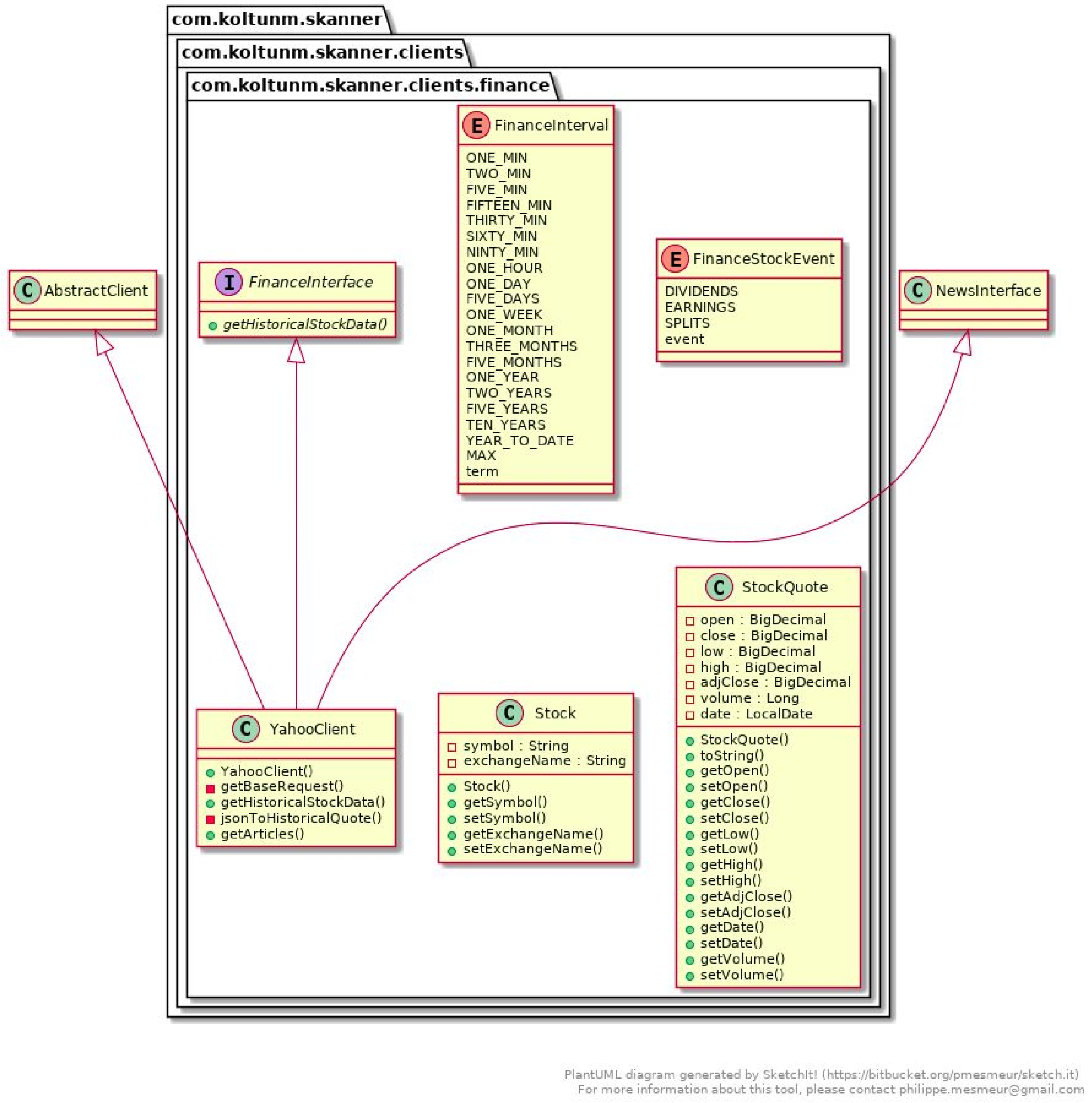




Finance Client

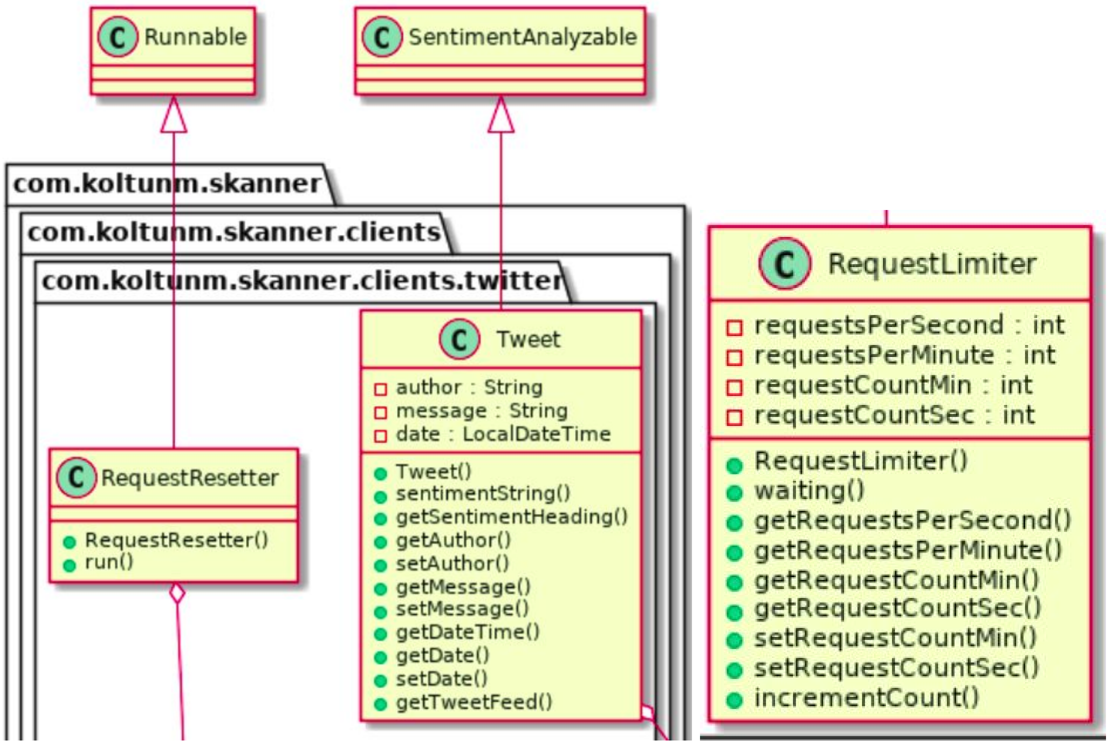
Same as with the news client, the finance client had to be implemented entirely as well. It's also powered by Unirest library to make HTTP requests.

FINANCE's Class Diagram



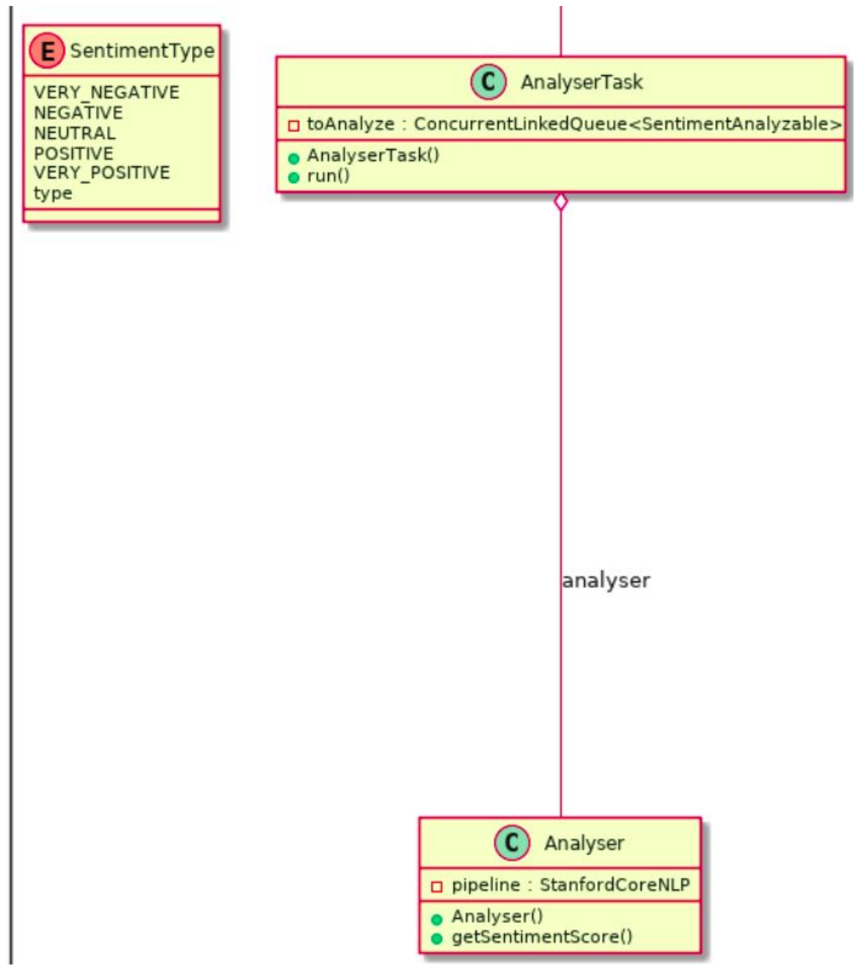
Twitter Client

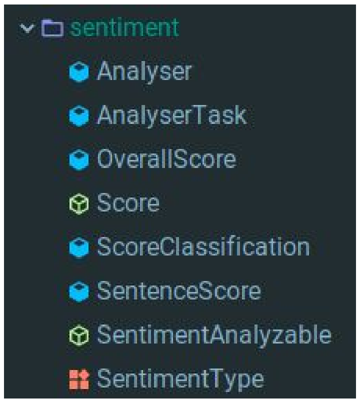
For this interface I used an unofficial client which had been already implemented - Twtter4j. However I had to create request limiter, because the requests are throttled to 30 requests per minute and 10 requests per second.

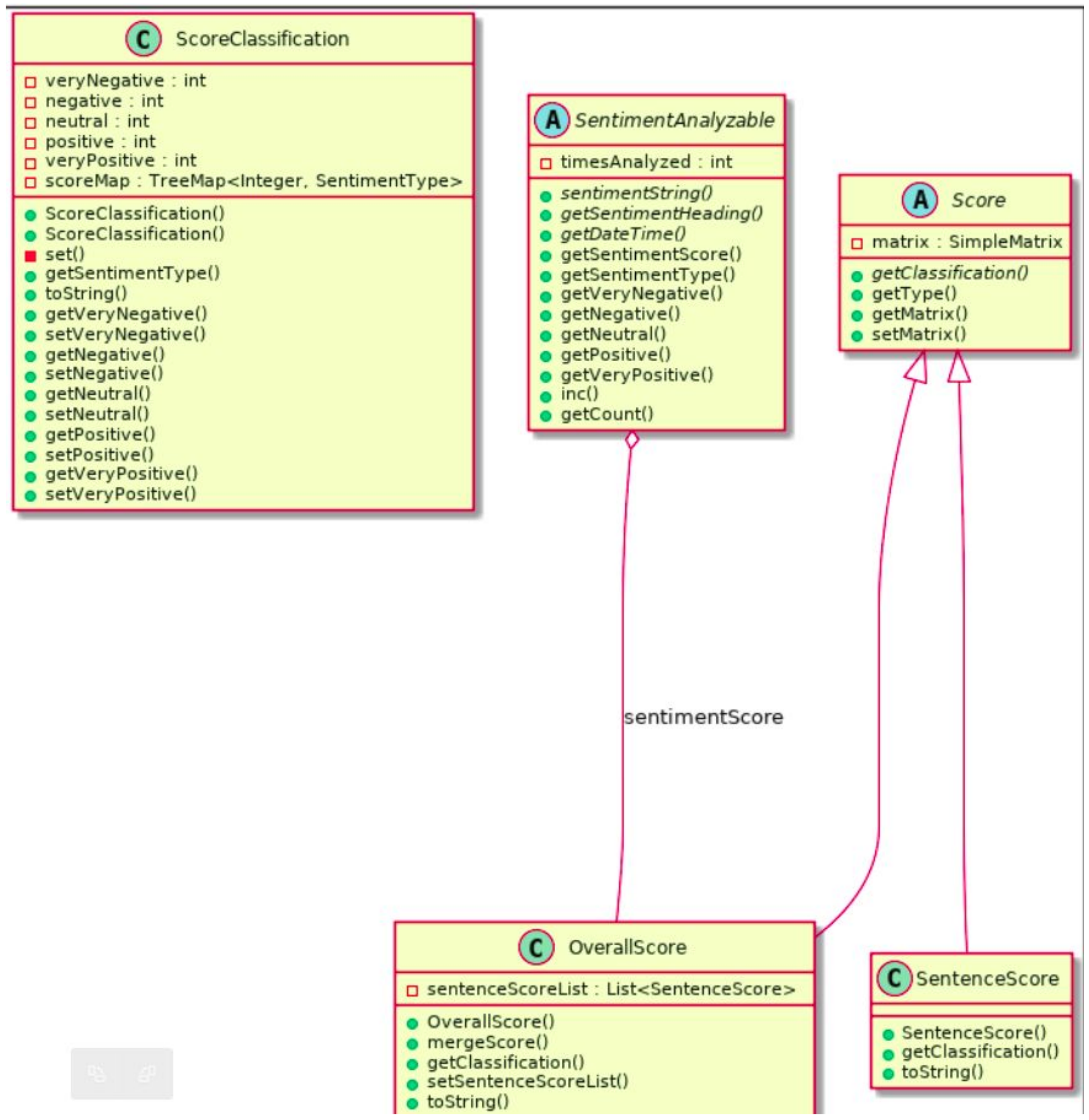


Sentiment Analysis

Sentiment analysis is a huge part of this project. For this purpose I used Stanford NLP package under MIT license. They provide language models and advanced algorithms, however I still had to implement support classes. Scoring system based on matrices had to be created. Sentiment analysis is obtained per sentences. The overall score is the average of all scores in the text. Analysis is performed on Tweets and content summary of articles.

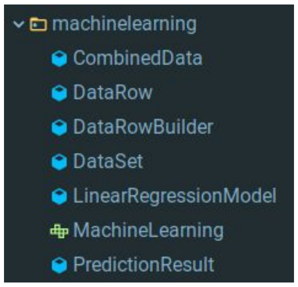

E SentimentType
VERY_NEGATIVE
NEGATIVE
NEUTRAL
POSITIVE
VERY_POSITIVE
type
C AnalyserTask
toAnalyze : ConcurrentLinkedQueue<SentimentAnalyzable>
AnalyserTask()
run()
analyser
C Analyser
pipeline : StanfordCoreNLP
Analyser()
getSentimentScore()


sentiment
Analyser
AnalyserTask
OverallScore
Score
ScoreClassification
SentenceScore
SentimentAnalyzable
SentimentType

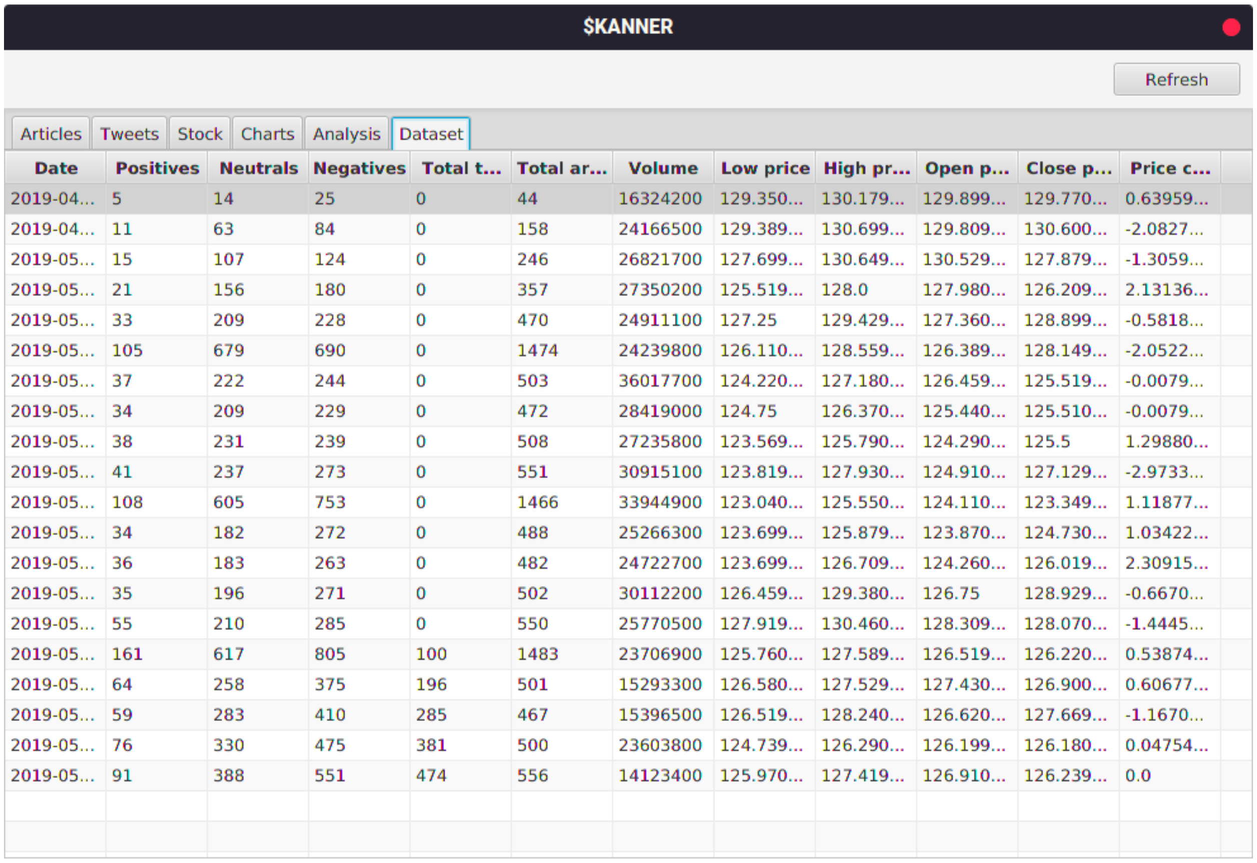


Machine learning and prediction

The purpose of Skanner is to predict the price of the stock in the near future. The classifier model is trained on a dataset which is constructed from the combination of sentiment analysis of the articles and tweets and historical stock quota. Linear Regression classifier is used for prediction of the price change in percent.



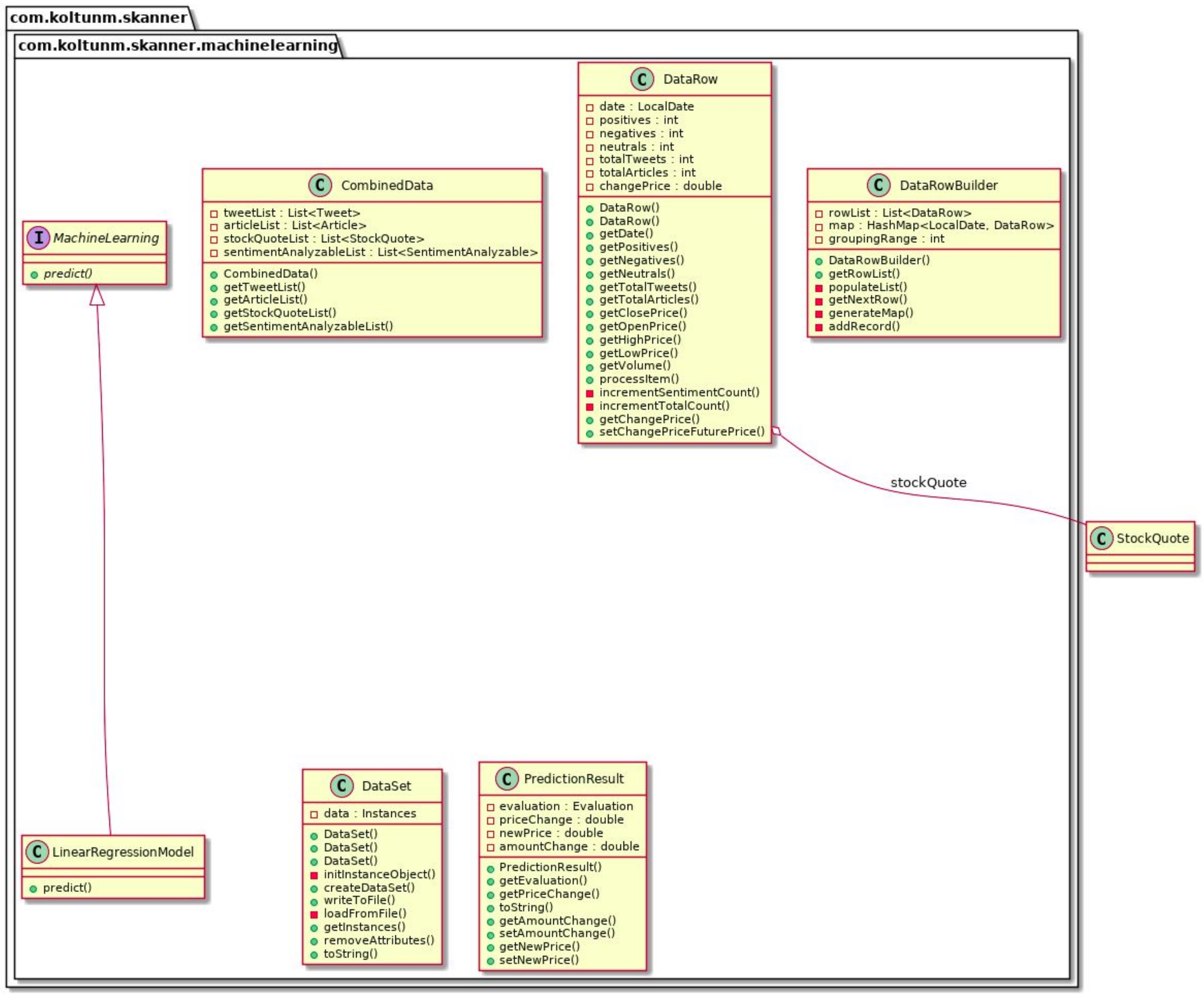
```
machinelearning
    CombinedData
    DataRow
    DataRowBuilder
    DataSet
    LinearRegressionModel
    MachineLearning
    PredictionResult
```



Date	Positives	Neutrals	Negatives	Total t...	Total ar...	Volume	Low price	High pr...	Open p...	Close p...	Price c...
2019-04...	5	14	25	0	44	16324200	129.350...	130.179...	129.899...	129.770...	0.63959...
2019-04...	11	63	84	0	158	24166500	129.389...	130.699...	129.809...	130.600...	-2.0827...
2019-05...	15	107	124	0	246	26821700	127.699...	130.649...	130.529...	127.879...	-1.3059...
2019-05...	21	156	180	0	357	27350200	125.519...	128.0	127.980...	126.209...	2.13136...
2019-05...	33	209	228	0	470	24911100	127.25	129.429...	127.360...	128.899...	-0.5818...
2019-05...	105	679	690	0	1474	24239800	126.110...	128.559...	126.389...	128.149...	-2.0522...
2019-05...	37	222	244	0	503	36017700	124.220...	127.180...	126.459...	125.519...	-0.0079...
2019-05...	34	209	229	0	472	28419000	124.75	126.370...	125.440...	125.510...	-0.0079...
2019-05...	38	231	239	0	508	27235800	123.569...	125.790...	124.290...	125.5	1.29880...
2019-05...	41	237	273	0	551	30915100	123.819...	127.930...	124.910...	127.129...	-2.9733...
2019-05...	108	605	753	0	1466	33944900	123.040...	125.550...	124.110...	123.349...	1.11877...
2019-05...	34	182	272	0	488	25266300	123.699...	125.879...	123.870...	124.730...	1.03422...
2019-05...	36	183	263	0	482	24722700	123.699...	126.709...	124.260...	126.019...	2.30915...
2019-05...	35	196	271	0	502	30112200	126.459...	129.380...	126.75	128.929...	-0.6670...
2019-05...	55	210	285	0	550	25770500	127.919...	130.460...	128.309...	128.070...	-1.4445...
2019-05...	161	617	805	100	1483	23706900	125.760...	127.589...	126.519...	126.220...	0.53874...
2019-05...	64	258	375	196	501	15293300	126.580...	127.529...	127.430...	126.900...	0.60677...
2019-05...	59	283	410	285	467	15396500	126.519...	128.240...	126.620...	127.669...	-1.1670...
2019-05...	76	330	475	381	500	23603800	124.739...	126.290...	126.199...	126.180...	0.04754...
2019-05...	91	388	551	474	556	14123400	125.970...	127.419...	126.910...	126.239...	0.0

Figure 12. Dataset for Microsoft stock.

MACHINELEARNING's Class Diagram

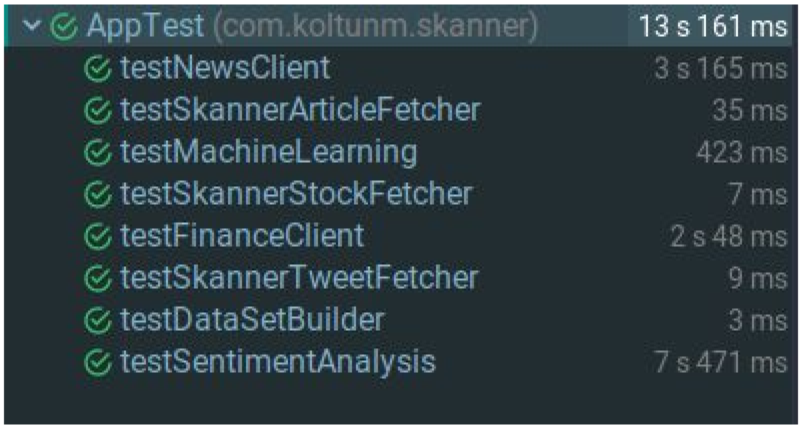


PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

5. Evaluation and Testing

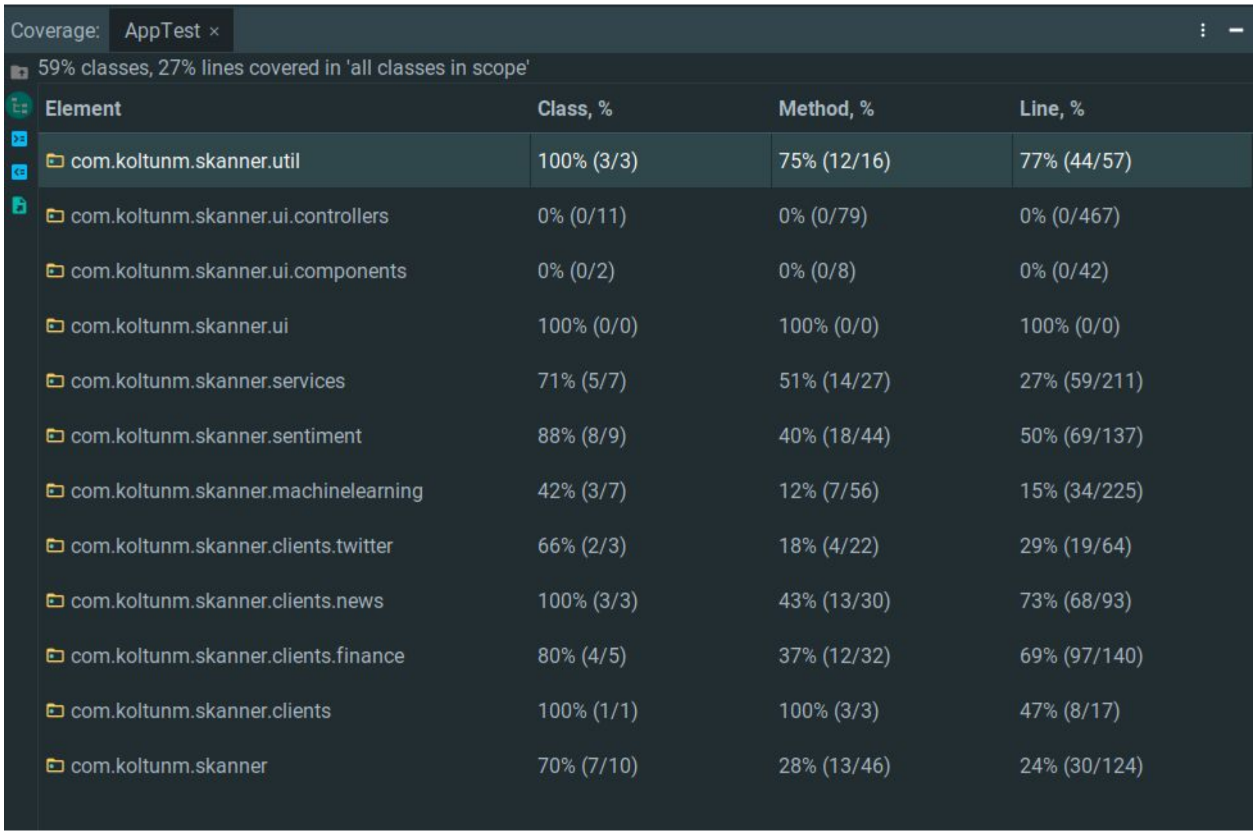
Unit and Integration Tests

The



```
AppTest (com.koltunm.skanner)    13 s 161 ms
    testNewsClient               3 s 165 ms
    testSkannerArticleFetcher    35 ms
    testMachineLearning          423 ms
    testSkannerStockFetcher      7 ms
    testFinanceClient            2 s 48 ms
    testSkannerTweetFetcher      9 ms
    testDataSetBuilder           3 ms
    testSentimentAnalysis        7 s 471 ms
```

Figure. Integration and Unit tests



Coverage: AppTest ×

59% classes, 27% lines covered in 'all classes in scope'

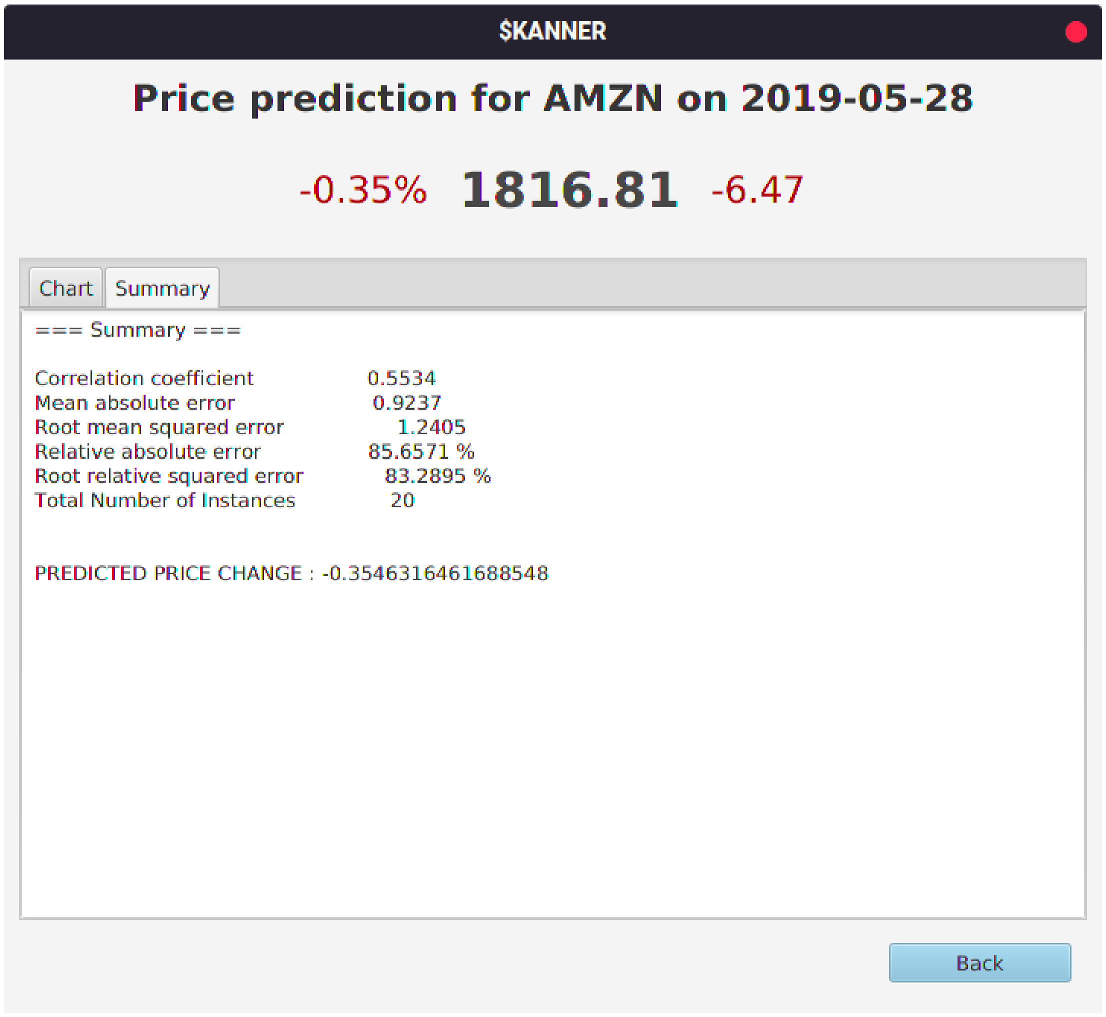
Element	Class, %	Method, %	Line, %
com.koltunm.skanner.util	100% (3/3)	75% (12/16)	77% (44/57)
com.koltunm.skanner.ui.controllers	0% (0/11)	0% (0/79)	0% (0/467)
com.koltunm.skanner.ui.components	0% (0/2)	0% (0/8)	0% (0/42)
com.koltunm.skanner.ui	100% (0/0)	100% (0/0)	100% (0/0)
com.koltunm.skanner.services	71% (5/7)	51% (14/27)	27% (59/211)
com.koltunm.skanner.sentiment	88% (8/9)	40% (18/44)	50% (69/137)
com.koltunm.skanner.machinelearning	42% (3/7)	12% (7/56)	15% (34/225)
com.koltunm.skanner.clients.twitter	66% (2/3)	18% (4/22)	29% (19/64)
com.koltunm.skanner.clients.news	100% (3/3)	43% (13/30)	73% (68/93)
com.koltunm.skanner.clients.finance	80% (4/5)	37% (12/32)	69% (97/140)
com.koltunm.skanner.clients	100% (1/1)	100% (3/3)	47% (8/17)
com.koltunm.skanner	70% (7/10)	28% (13/46)	24% (30/124)

Figure. Coverage of Integration and Unit tests

Evaluation

Case	Prediction	Actual price change	Difference (off by)
AMAZON	-0.35%	+0.72%	1.07%
APPLE	-1.13%	-0.41%	0.72%
NETFLIX	-0.22%	+0.11%	0.33%
MICROSOFT	-0.47%	-0.63%	0.16%

Case 1 AMAZON.COM, Inc.



Price prediction for AMZN on 2019-05-28

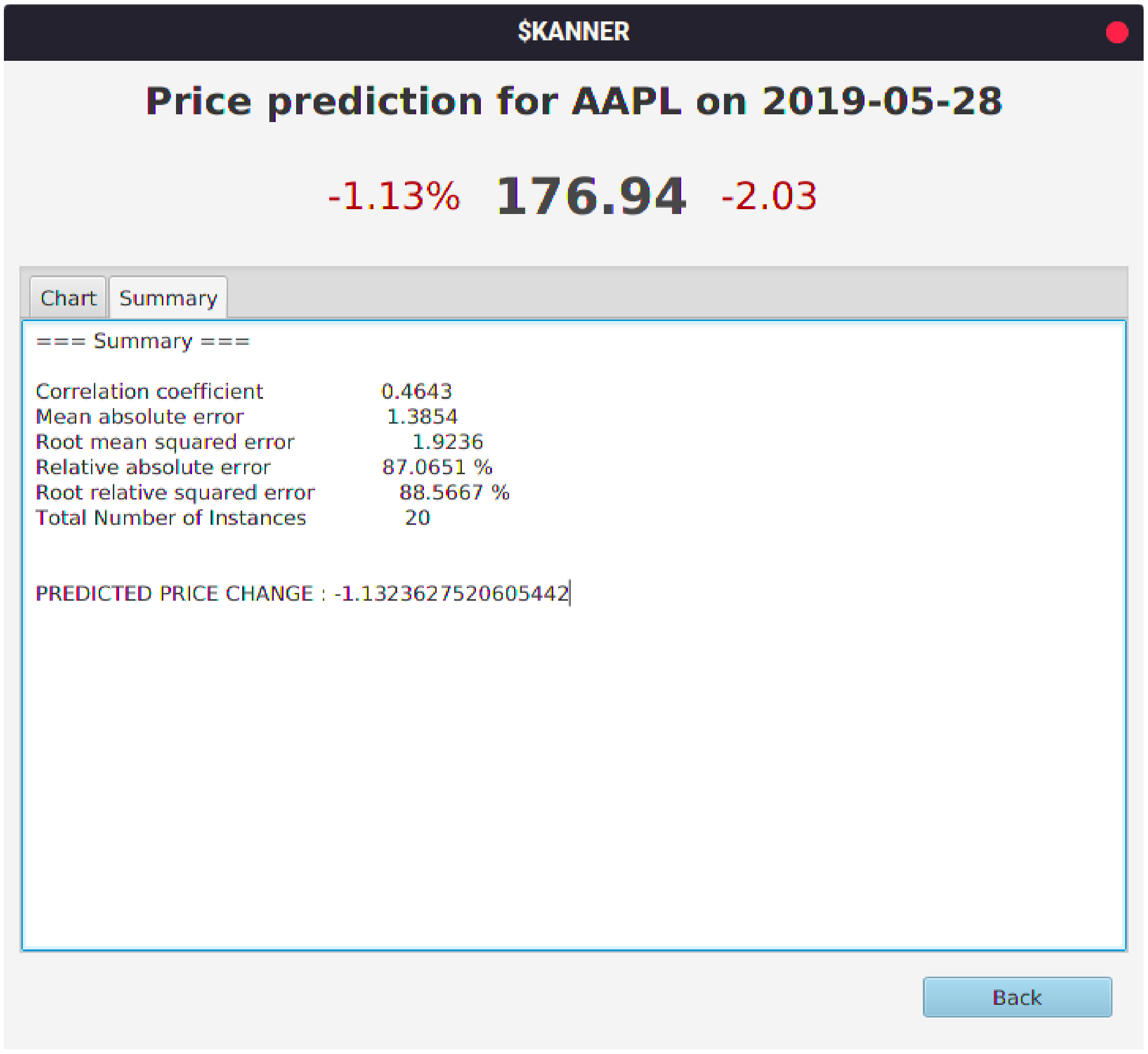
-0.35% 1816.81 -6.47

```
=== Summary ===

Correlation coefficient                 0.5534
Mean absolute error                     0.9237
Root mean squared error                   1.2405
Relative absolute error                 85.6571 %
Root relative squared error               83.2895 %
Total Number of Instances                20


PREDICTED PRICE CHANGE : -0.3546316461688548
```

Case 2 APPLE Inc.



Price prediction for AAPL on 2019-05-28

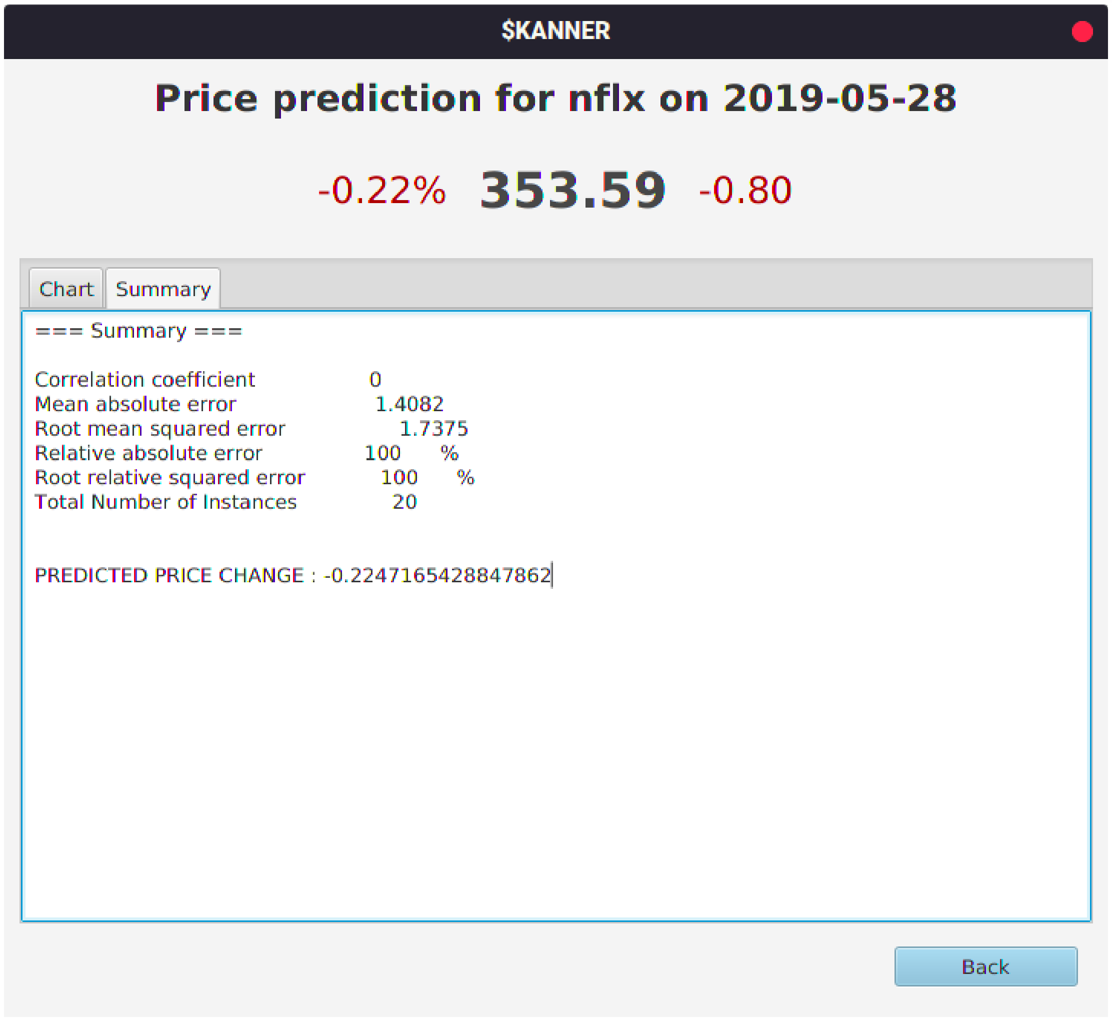
-1.13% 176.94 -2.03

```
=== Summary ===

Correlation coefficient                  0.4643
Mean absolute error                      1.3854
Root mean squared error                  1.9236
Relative absolute error                 87.0651 %
Root relative squared error             88.5667 %
Total Number of Instances               20


PREDICTED PRICE CHANGE : -1.1323627520605442
```

Case 3 Netflix, Inc.



Price prediction for nflx on 2019-05-28

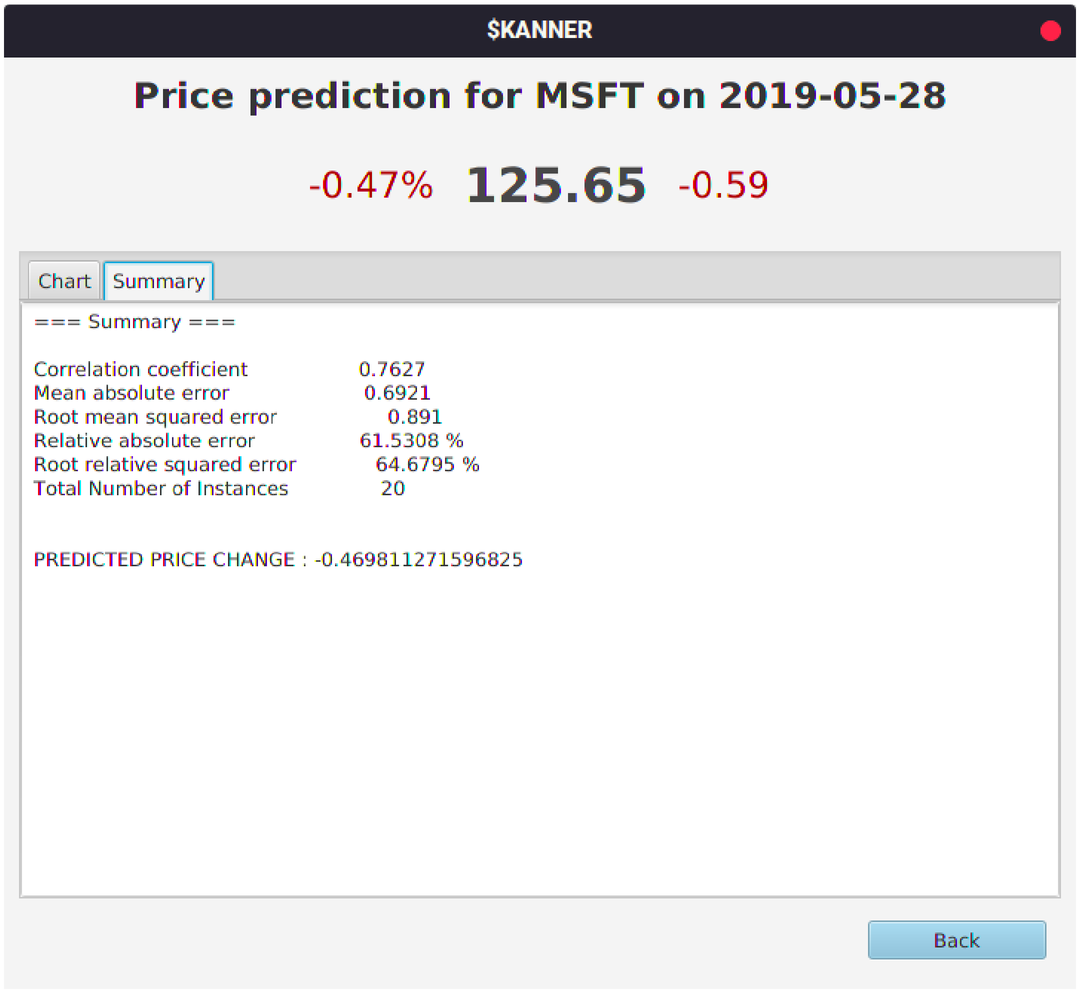
-0.22% 353.59 -0.80

```
=== Summary ===

Correlation coefficient                  0
Mean absolute error                      1.4082
Root mean squared error                      1.7375
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances                20


PREDICTED PRICE CHANGE : -0.2247165428847862
```

Case 4 Microsoft Corporation



Price prediction for MSFT on 2019-05-28

-0.47% 125.65 -0.59

```
=== Summary ===

Correlation coefficient                  0.7627
Mean absolute error                      0.6921
Root mean squared error                  0.891
Relative absolute error                 61.5308 %
Root relative squared error             64.6795 %
Total Number of Instances               20


PREDICTED PRICE CHANGE : -0.469811271596825
```