Time Series in Stock Price Prediction

By Yirui Chen(Team 2)

I. Abstract

The objective of this project is to predict Apple daily close stock price for varied forecast horizons by applying basic time series forecasting along with Autoregressive Model, “naïve” forecast, and machine learning methods. This project also aims at optimizing these models and presenting a comparison of these models. As a result, this project shows that “naïve” forecast has the best results for prediction.

II. Reviews of Related Studies

In the financial field, stock price analysis is one of the most important studies and time series is often used for this analysis. Many methods have been proposed to forecast stock price such as fundamental analysis, technical analysis, machine learning methods, and so on. Fundamental analysts focus on the company that underlies the stock itself. They assess a company's past performance as well as the credibility of its accounts. Technical analysts do not care about any fundamentals of the company. They try to determine the future price of a stock based only on past price trends. For example, techniques such as the exponential moving average (EMA) are used. In recent years, machine learning has become increasingly prominent in all walks of life, prompting many traders to apply machine learning technology to the field, some of which have achieved good results.

III. Problem Definition on Investment Decisions

No matter what kind of work people do, there is need to make some kind of assessment of the future and make a chart based on it. Finance experts use forecasts to make financial plans. Investors invest their hard-earned money in stocks in the hope of a positive return. Since mastering future stock price movements is important for financial professionals and investors, we need to seek to investigate a good forecasting method or to develop a forecasting software for stock price to enhance investors’ confidence.

IV. Simple Baseline Models for Forecasting Stock Price

For simple baseline models, the data we used is Apple close price from 2009 to 2015. The data is split into 90% training set and 10% test set.

A. Persistence Model (“Naïve” Forecasting)

The main idea of persistence model (“naïve” forecasting) is to predict today’s price to be yesterday’s price, that is, use the immediately preceding value as a predictor of stock prices. It is a simple but useful benchmark approach which is widely used for testing other models’ performance.

B. Autoregressive Model (AR)

Autoregressive models operate under the premise that past values have an effect on current values. The notation AR(p) indicates an autoregressive model for order p. The AR(p) model is defined as

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

where φ_1 ,…, φ_p are the parameters of the model, c is a constant, and ε_t is white noise. In order to implement Autoregressive Model, we use lag() method in spark to generate p lag columns which stand for X_{t-i} for each X_t. Then, we use the transformed dataset with linear regression. The expectation is that the regression algorithm will figure out the autocorrelation coefficients from X_{t-i} to X_t.

C. Baseline Results

Model	1 Day sMAPE	1 Week sMAPE	2 Weeks sMAPE	1 Month sMAPE	4 Months sMAPE
Persistence (LAG1/MA1)	**1.27%**	**2.57%**	**3.59%**	**4.29%**	**8.99%**
AR(p)	1.29%	2.64%	3.60%	4.50%	9.64%
Model	**Used Data**	**Feature Transformation**	**Train/Test Split**	**Overfitting**	**Model Optimization**
Persistence	AAPL Stock Close	None, AAPL Close	90% Train, 10% Test	RMSE 1.22 vs. 1.8, 2.3 vs. 3.6, 3.73 vs. 4.82,4.56 vs 5.3, 10.6 vs. 13.7	N/A
AR(p)	AAPL Stock Close	None, AAPL Close	90% train (3-Fold CV), 10% test	RMSE 1.15 vs. 2, 2.5 vs. 4.1, 3.4 vs. 5.02, 4.78 vs 6.3,11.3 vs. 14.6	p (equal to prediction window)

Fig. 1 *Baseline Results*

As we can see, results are getting worse along with forecast horizons are getting longer. It is not surprised that shorter time horizons are often easier to predict with higher confidence. Unlike the short-term time series prediction, the long-term forecasting usually faces increased uncertainty from various sources. Next we would try more models to see if the results are improved over long forecast horizons or all forecast horizons.

V. Machine Learning Methods for Forecasting Stock Price

A. Feature Engineering

1) Used Data

In addition daily stock prices of Apple, we add twitter sentiment datasets, historical data of other stocks (S&P500) from 1980 to 2018 and news text from Reddit from 2009 to 2015 to improve prediction accuracy . Through the literature review, we know that many studies have shown that other factors should be considered in addition to the basic model. One of the most common and important of these factors is stock-related social media messages.

2) Feature Extraction

We did a lot work on feature extraction to find factors that may affect the stock market to get good predictions.. As we can see in the above graphing there are a periodic weekly trend and a long term upwards so we could encode some time variables like day of week and month of year. Lag features are the classical way that time series forecasting problems are transformed into supervised learning problems so we add lag features by shifting the data by 1,2,…, 10 days. The window (rolling) statistics data calculated from the raw prices and the technical indicators giving information about the market tendency, volatility and momentum are also used as features. For instance, we use x-days moving average which stands for the average of the price of the last x days moving averages and moving Z-score as features.
All features and explanations are shown in **Table. 1**

Feature	Explanation
Time variables	Month of year, day of week, day of year
Moving average	X-days moving average
Moving variance	The variance within a moving window
Moving Z-score	The moving Z-score is the number of standard deviations each observation is away from the moving average
Differencing	Change in value since the prior period
Close_Diff_Percent	Rate of change percentage of differencing
Lag features	Shifting the data by 1,2 ,…, and 10 days .
News_sentiment	Subjectivity, objectivity, neutral, positive and negative scores of news in a certain day
Tf-idf bow features	Using bag of words model and Tf-idf to transform raw text in news into numerical features
Tickers	Historical data of other stocks (S&P500)

Table 1

3) Feature Selection

We add Chi-Square Selector which can be used to select n features with highest values for the test chi-squared statistic to Pipeline to do feature selection. In the process of model training, we can clearly notice eliminating noise features increases efficiency and effectiveness of models.

B. Model Selection

1) Linear Regression

Linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Formally, the model for multiple linear regression, given n observations, is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_p x_{ip} + \varepsilon_i$$

For i = 1,2, … n.

In the least-squares model, the best-fitting line for the observed data is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line.

2) Random Forest Regressor

A random forest is a meta that fits many decision trees on different subsamples of the data set and uses the average value to improve the prediction accuracy and control over fitting. Whether it is regression or classification, decision trees work on the same algorithm that in each step a variable that reduces the cost metrics is selected. In case of regression trees it is based on reduction in standard deviation done by splitting the node.

3) GBT Regressor

Boosting is a method of converting weak learners into strong learners. The first implementation of boosting was Adaptive Boosting. AdaBoost works by weighting the observations, putting more weight on instances that are difficult to classify and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns. Gradient boosting is a generalization of AdaBoost where the objective is to minimize the loss of the model by adding weak learners using a gradient descent like procedure.

C. Model Optimization

1) Methodology

To optimize our models, take the random forest as an example, we use the ParamGridBuilder utility to construct the parameter grid for max depth, num trees and subsampling rate. Then CrossValidator is used to select from a grid of parameters. To evaluate a particular parammap, CrossValidator caculates the average evaluation metric for the three models produced by fitting the Estimator on the three different (training, test) dataset pairs.

2) Metrics

To measure performance of the model, we use RMSE and sMAPE. RMSE is known as scale-dependent measures because its scale depends on the scale of the data which means it is not a suitable metric because the scale of close price of 1980 is different from 2018. SMAPE is the preferred metric for time series analysis because it is scale-independent. Unfortunately, metrics in RegressionEvaluator() in spark do not include sMAPE so the professor's recommendation is we could use RMSE as the metric when we implement cross validation and then use sMAPE after the best hyperparameters are selected for each model by using cross validation.

3) Features Importance

1 Day Window Top 5 Features	
Feature	**FI**
Close_aapl_Lag_1	0.225095
Low_aapl_Lag_1	0.158824
High_aapl_Lag_1	0.146731
Volume_aapl_ma_20	0.121023
Open_aapl_Lag_1	0.086223

Fig. 2 *1 Day Window Top 5 Features*

4 Month Window Top 5 Features	
Feature	**FI**
Open_aapl_ma_50	0.075808
Close_aapl_var_50	0.042087
Close_aapl_ma_50	0.036239
Volume_sp500_ma_50	0.024744
Volume_aapl_var_50	0.021343

Fig. 3 *4 Month Window Top 5 Features*

As we can see, the close price depends the close price of the previous day for small time frames whereas for longer forecast horizons the close price depends on macro values such as moving average, that is, the average price of the last fifty days.

VI. Results and Conclusions

A. All Results

Model	1 Day sMAPE	1 Week sMAPE	2 Weeks sMAPE	1 Month sMAPE	4 Months sMAPE
Persistence (LAG1/MA1)	**1.27%**	2.57%	3.59%	4.29%	8.99%
AR(p)	1.29%	2.64%	3.60%	4.50%	9.64%
Feature Engineered RF	2.52%	7.77%	8.705%	8.11%	**4.07%**
Feature Engineered GBT	2.75%	18.35%	12.01%	10.05%	**6.7%**
Feature Engineered RLR	1.41%	3.14%	4.62%	6.51%	18.30%

Model	Used Data	Feature Transformation	Train/test Split	Overfitting	Model Optimization
Persistence	AAPL Stock	None, AAPL Close	90% Train, 10% Test	RMSE 1.22 vs. 1.8, 2.3 vs. 3.6, 3.73 vs. 4.82,4.56 vs 5.3, 10.6 vs. 13.7	N/A
AR(p)	AAPL Stock	None, AAPL Close	90% train (3-Fold CV), 10% test	RMSE 1.15 vs. 2, 2.5 vs. 4.1, 3.4 vs. 5.02, 4.78 vs 6.3,11.3 vs. 14.6	p
Feature Engineered RF	AAPL + S&P	All attributes, time, lags, moving averages, differences, sign, variance, zscores etc.	90% train (3-Fold CV), 10% test	RMSE: 1.9 vs. 3.8, 2.9 vs. 11.11, 3.2 vs 12.6, 3.7 vs 12.2, 5.9 vs 6.6	maxDepth, numTrees, subsamplingRate [3,25,0.8]
Feature Engineered GBT	AAPL + S&P	All attributes, time, lags, moving averages, differences, sign, variance, zscores etc.	90% train (3-Fold CV), 10% test	RMSE: 2.5 vs 4.2, 3.4 vs 22.9 ,3.71 vs 15.84, 4.2 vs 14.14, 6.4 vs 9.2	maxDepth, maxIter, subsamplingRate [5,30,0.7]
Feature Engineered RLR	AAPL + S&P	All attributes, time, lags, moving averages, differences, sign, variance, zscores	90% train (3-Fold CV), 10% test	RMSE: 1.6 vs 2.2, 3.4 vs 4.6, 4.6 vs 6.8, 5.7 vs 9.7, 11.35 vs 24.7	Regularization + Elastic Net [0.01,0.05]

Fig.4 *All Results without News*

Model	1 Day sMAPE	1 Week sMAPE	2 Weeks sMAPE	1 Month sMAPE	4 Months sMAPE
Feature Engineered RF + News	3.42%	8.67%	9.91%	10.12%	5.22%
Feature Engineered GBT + News	2.99%	22.32%	13.21%	14.11%	8.4%
Feature Engineered RLR + News	1.71%	3.89%	5.91%	8.60%	21.10%

Fig.5 *Machine Learning Models with News Data*

As we can see, news data did not improve models because the news data we used is not specific enough to Apple. We would use better news data such as tweets about #aapl on twitter in the future work. Models are overfitting which are probably due to poor hyperparameters tuning because we could not give Databricks a big enough grid to search over which is also a limitation we run into when we work with Databricks.

In all, we can draw conclusions that:

(a) Simple models performed better than regression trees, and "naive" forecasting performed best. As I said before, classical approaches should be used as a baseline when evaluating any machine learning and deep learning methods for time series forecasting because sometimes simple methods outperform sophisticated methods.

(b) As the prediction horizon gets longer, the results get worse.

B. Recommendation to Improve Results

(a) We would use better news data such as tweets about #aapl on twitter to do sentiment analysis for stock price prediction in the future work.

(b) Use Fundamental Analysis such as unit of sales for Apple Product. As I said in the related work part, fundamental analysts are concerned with the company that underlies the stock itself and they evaluate a company's past performance as well as the credibility of its accounts.

C. Challenges and Mentor’s Recommendations

We hope to try more robust models as much as we can such as Artificial Neural Network(ANN) and Xgboost. Unfortunately, Spark only offers algorithms about ANN for classification whereas we are supposed to apply regression algorithm to stock prediction. Also, unlike gradient boosting algotithm, xgboost library is difficult to import in Databricks and we have found many people have the same problem with Xgboost. We asked our mentor for help and we were recommended to use Scala because these algorithms are not supported in Pyspark. Given we have did so much so far, we decided not to change our programming language to Scala but focus on optimizing the models we can implement in Pyspark. Also, our mentor recommended we should use some external datasets to improve prediction accuracy such as news data and he acknowledged it is a bit of a challenge to get financial news as the gold standard for news feeds tends to be Thomson Reuters which are paid for services.

References

[1] Shynkevich, Yauheniya, et al. "Forecasting price movements using technical indicators: Investigating the impact of varying input window length." Neurocomputing 264 (2017): 71-88.

[2] Long, W., Lu, Z., & Cui, L. (2019). Deep learning-based feature engineering for stock price movement prediction. *Knowledge-Based Systems*, *164*, 163-173.

[3] Deng, S., Zhang, N., Zhang, W., Chen, J., Pan, J. Z., & Chen, H. (2019). Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network.

[4] Xu, S. Y., & Berkely, C. U. (2014). Stock price forecasting using information from Yahoo finance and Google trend. *UC Brekley*.

Appendix

AR_Model:

```
from pyspark.sql import SQLContext, Window
from pyspark.sql.functions import *
from pyspark.ml.regression import LinearRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler, VectorSlicer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import abs, sqrt
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler, Standa
rdScaler, OneHotEncoderEstimator, PCA, VectorSlicer
from pyspark.ml.regression import DecisionTreeRegressor, GBTRegressor, RandomForestR
egressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import pandas as pd

seed = 26

#Parameters
lags = 80
pred_window = 80

total_mape = []
total_rmse = []

#read in data
sqlContext = SQLContext(sc)
df = sqlContext.sql("SELECT aapl.date, aapl.close AS close1 FROM aapl WHERE YEAR(aap
l.Date) >= 2009 and YEAR(aapl.Date) <= 2015")

lag_col_names = ["close1"]

#generate lag columns
for lag_num in range(1,lags+1):
    lagtest = lag("close1",lag_num,0).over(Window.partitionBy().orderBy("date"))
    df = df.withColumn("lag{}".format(lag_num),lagtest)
    lag_col_names.append("lag{}".format(lag_num))


```

```
#remove null
#df = df.dropna()

#prepare for model
lag_col_names.pop(0) #remove close from name vector

featureassembler = VectorAssembler(inputCols=lag_col_names,outputCol="features")
output = featureassembler.transform(df)
finalized_data = output.select("date","features","close1")

#PREDICTION WINDOW
finalized_data = finalized_data.withColumn("Close_aapl_Window",lead("close1",pred_window-1,None).over(Window.orderBy("date")))
finalized_data = finalized_data.drop("close1")
finalized_data = finalized_data.withColumnRenamed("Close_aapl_Window","close1")
finalized_data = finalized_data.withColumnRenamed("close1","label")
finalized_data = finalized_data.dropna()


#split into test/train maintaining integrity of time series data
#train_data,test_data = finalized_data.randomSplit([0.7,0.3],seed)
finalized_data = finalized_data.withColumn("rank",percent_rank().over(Window.partitionBy().orderBy("date")))
train_data = finalized_data.where("rank <= .9").drop("rank")
test_data = finalized_data.where("rank > .9").drop("rank")

#fit regression model
regressor = LinearRegression(featuresCol="features",labelCol="label")

paramGrid = ParamGridBuilder().addGrid(regressor.regParam, [0,0.01])\
                              .build()

crossval = CrossValidator(estimator=regressor,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator().setMetricName("rmse"),
                          numFolds=3)

cvModel = crossval.fit(train_data)

#PRINT RESULTS
params = [{p.name: v for p, v in m.items()} for m in cvModel.getEstimatorParamMaps()]

cv_results = pd.DataFrame.from_dict([
```

```
    {cvModel.getEvaluator().getMetricName(): metric, **ps}
    for ps, metric in zip(params, cvModel.avgMetrics)
])

test_results = evaluateModel(cvModel,test_data)
print(cv_results)
ExtractFeatureImp(cvModel.bestModel.coefficients, test_results, "features").head(100)
```

MA1_Model:

```
from pyspark.sql import SQLContext, Window
from pyspark.sql.functions import *
from pyspark.ml.regression import LinearRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler, VectorSlicer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import abs, sqrt
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler, StandardScaler, OneHotEncoderEstimator, PCA, VectorSlicer
from pyspark.ml.regression import DecisionTreeRegressor, GBTRegressor, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import pandas as pd

#Parameters
ma_order = 1

#Metrics
total_mape = []
total_rmse = []
total_smape = []

#read in data
sqlContext = SQLContext(sc)
df = sqlContext.sql("SELECT * FROM aapl WHERE YEAR(aapl.date) >= 2009 and YEAR(aapl.date) <= 2015")

#forecast using ma
df = df.withColumn("close_ma", avg(col("close")).over(Window.rowsBetween(-ma_order,-1)))

```

```
30.
31. for window in [1,5,10,20,80]:
32.   df = df.withColumn("Close_Actual_Window_{}".format(window),lead("close",window-
    1,None).over(Window.orderBy("Date")))
33.
34. for window in [1,5,10,20,80]:
35.   df = df.withColumn("squared_error_Window_{}".format(window), pow((col("Close_Actua
    l_Window_{}".format(window)) - col("close_ma")),2))
36.   df = df.withColumn("s_abs_percentage_error_Window_{}".format(window),(abs(col("clo
    se_ma") -
    col("Close_Actual_Window_{}".format(window)))/((col("Close_Actual_Window_{}".format(
    window)) + col("close_ma"))/2))*100)
37.
38. df.show()
39.
40. df = df.withColumn("rank",percent_rank().over(Window.partitionBy().orderBy("date")))

41. train_data = df.where("rank <= .9").drop("rank")
42. test_data = df.where("rank > .9").drop("rank")
43.
44. for window in [1,5,10,20,80]:
45.   total_rmse.append(test_data.select(sqrt(mean(col("squared_error_Window_{}".format(
    window))))).collect())
46.   total_smape.append(test_data.select(mean(col("s_abs_percentage_error_Window_{}".fo
    rmat(window)))).collect())
47.
48. print(total_mape)
49. print(total_rmse)
50. print(total_smape)
```

News Data Feature Engineering:

```
1.  from pyspark.sql import SQLContext, Window
2.  from pyspark.sql.functions import *
3.  from pyspark.sql.types import DoubleType
4.  from pyspark.ml.regression import LinearRegression
5.  from pyspark.ml.linalg import Vectors
6.  from pyspark.ml.feature import VectorAssembler, Imputer
7.  from pyspark.ml.evaluation import RegressionEvaluator
8.  from  pyspark.sql.functions import abs, sqrt
9.  from pyspark.sql.functions import concat, col, lit
10. from pyspark.ml.feature import StopWordsRemover, Tokenizer, RegexTokenizer, HashingT
    F, IDF
11.
```

```
#TFIDF BOW FEATURES

df_news = sqlContext.sql("SELECT Date, Top1,Top2,Top25 FROM combined_news_djia_csv")


num_word_features = 2000

#news data only goes to july 2016
df_news = sqlContext.sql("SELECT * FROM combined_news_djia_csv")
df_news = df_news.select("Date",concat(col("Top1"), lit(" "), col("Top2"), lit(" "), col("Top3"), lit(" "), col("Top4"), lit(" "), col("Top5"), lit(" "), col("Top6"), lit(" "), col("Top7"), lit(" "), col("Top8"), lit(" "), col("Top9"), lit(" "), col("Top10"), lit(" "), col("Top11"), lit(" "), col("Top12"), lit(" "), col("Top13"), lit(" "), col("Top14"), lit(" "), col("Top15"), lit(" "), col("Top16"), lit(" "), col("Top17"), lit(" "), col("Top18"), lit(" "), col("Top19"), lit(" "), col("Top20"), lit(" "), col("Top21"), lit(" "), col("Top22"), lit(" "), col("Top23"), lit(" "), col("Top24"), lit(" "), col("Top25")).alias("all_text_dirty"))

df_news = df_news.withColumn("all_text_1",regexp_replace(col("all_text_dirty"), "['\"]", ""))
df_news = df_news.withColumn("all_text",expr("substring(all_text_1, 2, length(all_text_1)+1)"))


df_news = df_news.dropna()

tokenizer = Tokenizer(inputCol="all_text", outputCol="words")
wordsData = tokenizer.transform(df_news)

remover = StopWordsRemover(inputCol="words", outputCol="wordsFil")
wordsDataFil = remover.transform(wordsData)

hashingTF = HashingTF(inputCol="wordsFil", outputCol="rawFeatures", numFeatures=num_word_features)
featurizedData = hashingTF.transform(wordsDataFil)
# alternatively, CountVectorizer can also be used to get term frequency vectors

idf = IDF(inputCol="rawFeatures", outputCol="news_features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)

#df_news = rescaledData.select("Date","news_features")

"""imputer = Imputer(
```

```
    inputCols=[column for column in df_news.columns if column not in ["Date"]],
    outputCols=["{}_cleaned".format(c) for c in [column for column in df_news.columns if column not in ["Date"]]],
    strategy = "median"
)

df_news = imputer.fit(df_news).transform(df_news)
df_news = df_news.dropna()"""

#df_news  = df_news.select([col for col in df_news.columns if ("_cleaned" in col) or (col in time_cols)])

df_news = rescaledData.select("news_features")

#NEWS SENTIMENT DATA (FROM KAGGLE AND VADER)
lags = 10

df_news_sent = sqlContext.sql("SELECT Date, Subjectivity, Objectivity, Positive, Neutral, Negative FROM news_sentiment")

#GENERATE LAGS (News might take time to take effect)
for lag_num in range(1,lags+1):
    df_news_sent = df_news_sent.withColumn("Subjectivity_Lag_{}".format(lag_num),avg(col("Subjectivity")).over(Window.rowsBetween(-lag_num,-lag_num)))
    df_news_sent = df_news_sent.withColumn("Objectivity_Lag_{}".format(lag_num),avg(col("Objectivity")).over(Window.rowsBetween(-lag_num,-lag_num)))
    df_news_sent = df_news_sent.withColumn("Positive_Lag_{}".format(lag_num),avg(col("Positive")).over(Window.rowsBetween(-lag_num,-lag_num)))
    df_news_sent = df_news_sent.withColumn("Neutral_Lag_{}".format(lag_num),avg(col("Neutral")).over(Window.rowsBetween(-lag_num,-lag_num)))
    df_news_sent = df_news_sent.withColumn("Negative_Lag_{}".format(lag_num),avg(col("Negative")).over(Window.rowsBetween(-lag_num,-lag_num)))

drop_list = ["Subjectivity","Objectivity","Positive","Neutral","Negative"]
df_news_sent  = df_news_sent.select([column for column in df_news_sent.columns if column not in drop_list])

"""imputer = Imputer(
    inputCols=[column for column in df_news_sent.columns if column not in ["Date"]],

    outputCols=["{}_clean".format(c) for c in [column for column in df_news_sent.columns if column not in ["Date"]]],
    strategy = "median"
)
```

```
df_news_sent = imputer.fit(df_news_sent).transform(df_news_sent)
"""
#df_news_sent = df_news_sent.select([col for col in df_news_sent.columns if ("_clean
" in col) or (col in ["Date"])])

df_news_sent = df_news_sent.dropna()
df_news_sent.show(5)
```

Stock Feature Engineering and Advanced Models:

```
# Databricks notebook source
from pyspark import keyword_only  ## < 2.0 -> pyspark.ml.util.keyword_only
from pyspark.ml import Transformer
from pyspark.ml.param.shared import HasInputCol, HasOutputCol, Param
from pyspark.sql.functions import udf
from pyspark.sql.types import ArrayType, StringType

from pyspark.sql import SQLContext, Window
from pyspark.sql.functions import *
from pyspark.sql.types import DoubleType, StringType
from pyspark.ml.regression import LinearRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler, Imputer
from pyspark.ml.evaluation import RegressionEvaluator
from  pyspark.sql.functions import abs, sqrt
import pandas as pd

#CUSTOM TRANSFORMERS
class StockFeatureCreator(Transformer):

    @keyword_only
    def __init__(self, lags, pred_window, ma_windows, tickers):
        self._paramMap = {}

        #: internal param map for default values
        self._defaultParamMap = {}

        #: value returned by :py:func:`params`
        self._params = None

        # Copy the params from the class to the object
        self._copy_params()
```

```
33.
34.         self.lags = lags
35.         self.pred_window = pred_window
36.         self.ma_windows = ma_windows
37.         self.inputCols = []
38.         self.tickers = tickers
39.         self.outputCols = []
40.
41.     def getOutputCols(self):
42.
43.         return self.outputCols
44.
45.     def build(self):
46.
47.         #GENERATE ALL COLUMN NAMES
48.         for tick in self.tickers:
49.           self.inputCols.append("Open_"+tick)
50.           self.inputCols.append("High_"+tick)
51.           self.inputCols.append("Low_"+tick)
52.           self.inputCols.append("Volume_"+tick)
53.           self.inputCols.append("Close_"+tick)
54.           self.inputCols.append("Volume_"+tick)
55.
56.         #Lags
57.         for lag_num in range(1,self.lags+1):
58.           for feature in self.inputCols:
59.             self.outputCols.append("{}_Lag_{}".format(feature,lag_num))
60.
61.         #MA windows
62.         for ma_length in self.ma_windows:
63.           for feature in self.inputCols:
64.             self.outputCols.append("{}_ma_{}".format(feature,ma_length))
65.             self.outputCols.append("{}_var_{}".format(feature,ma_length))
66.             self.outputCols.append("{}_Z_{}".format(feature,ma_length))
67.
68.         for lag_num in range(1,self.lags+1):
69.           for tick in self.tickers:
70.             self.outputCols.append("Open_Close_{}_Ratio_Lag_{}".format(tick,lag_num)
    )
71.
72.         for lag_num in range(1,int((self.lags)/3)):
73.           for feature in self.inputCols:
74.             self.outputCols.append("{}_Diff_{}".format(feature,lag_num))
75.             self.outputCols.append("{}_Diff_Percent_{}".format(feature,lag_num))
```

```
            self.outputCols.append("{}_Diff_{}_Sign".format(feature,lag_num))
            self.outputCols.append("{}_Rolling_Sign_{}".format(feature,lag_num))

        #self.outputCols = [s + "_imputed" for s in self.outputCols]

    def _transform(self, dataset):

        #LOG VOLUME
        for tick in self.tickers:
          dataset = dataset.withColumn("Volume_{}_float".format(tick), log(col("Volume_{}".format(tick)).cast(DoubleType()))) #log because large value
          dataset = dataset.drop("Volume_{}".format(tick))
          dataset = dataset.withColumnRenamed("Volume_{}_float".format(tick),"Volume_{}".format(tick))

        #GENERATE TIME COLUMNS
        dataset = dataset.withColumn("Day_Of_Week",dayofweek("Date"))
        dataset = dataset.withColumn("Month",month("Date"))
        dataset = dataset.withColumn("Quarter",quarter(col("Date")))
        dataset = dataset.withColumn("Week_Of_Year",weekofyear(col("Date")))
        #dataset = dataset.withColumn("Day_Of_Year",dayofyear(col("Date")))
        dataset = dataset.withColumn("Day_Of_Month",dayofmonth(col("Date")))
        #dataset = dataset.withColumn("Year",year(col("Date")))

        time_cols = ["Date",
                     "Day_Of_Week",
                      "Month",
                      "Quarter",
                      "Week_Of_Year",
                      "Day_Of_Month"]

         #GENERATE LAG COLUMNS
         for lag_num in range(1,self.lags+1):
           for feature in self.inputCols:
             dataset = dataset.withColumn("{}_Lag_{}".format(feature,lag_num),
                                          lag(feature,lag_num,None).over(Window.orderBy("Date")))

         #SIMPLE MOVING AVERAGES, MOVING VARIANCE AND Z SCORE
         for ma_length in self.ma_windows:
           for feature in self.inputCols:
             dataset = dataset.withColumn("{}_ma_{}".format(feature,ma_length), avg(col(feature)).over(Window.rowsBetween(-ma_length,-1)))
```

```
115.             dataset = dataset.withColumn("{}_var_{}".format(feature,ma_length), var
    iance(col(feature)).over(Window.rowsBetween(-ma_length,-1)))
116.             dataset = dataset.withColumn("{}_Z_{}".format(feature,ma_length), (col(
    "{}_Lag_1".format(feature)) - col("{}_ma_{}".format(feature,ma_length)))/(col("{}_va
    r_{}".format(feature,ma_length))))
117.
118.         #OPEN/CLOSE RATIO
119.         for lag_num in range(1,self.lags+1):
120.           for tick in self.tickers:
121.             dataset = dataset.withColumn("Open_Close_{}_Ratio_Lag_{}".format(tick,l
    ag_num),col("Open_{}_Lag_{}".format(tick,lag_num))/col("Close_{}_Lag_{}".format(tick
    ,lag_num)))
122.
123.         #DIFFERENCING, PERCENT CHANGE, SIGN AND ROLLING SUM
124.         for lag_num in range(1,int((self.lags)/3)):
125.           for feature in self.inputCols:
126.             dataset = dataset.withColumn("{}_Diff_{}".format(feature,lag_num), when
    (isnull(col("{}_Lag_{}".format(feature,lag_num)) - col("{}_Lag_{}".format(feature,la
    g_num+1))), 0).otherwise(col("{}_Lag_{}".format(feature,lag_num)) - col("{}_Lag_{}".
    format(feature,lag_num+1))))
127.             dataset = dataset.withColumn("{}_Diff_Percent_{}".format(feature,lag_nu
    m), when(isnull((col("{}_Lag_{}".format(feature,lag_num)) - col("{}_Lag_{}".format(f
    eature,lag_num+1)))/col("{}_Lag_{}".format(feature,lag_num+1))), 0).otherwise((col("
    {}_Lag_{}".format(feature,lag_num)) - col("{}_Lag_{}".format(feature,lag_num+1)))/co
    l("{}_Lag_{}".format(feature,lag_num+1))))
128.             dataset = dataset.withColumn("{}_Diff_{}_Sign".format(feature,lag_num),
    when(col("{}_Diff_{}".format(feature,lag_num)) > 0, 1.0).otherwise(-1.0))
129.             dataset = dataset.withColumn("{}_Rolling_Sign_{}".format(feature,lag_nu
    m), sum(col("{}_Diff_{}_Sign".format(feature,lag_num))).over(Window.rowsBetween(-
    (lag_num+1),-1)))
130.
131.         #IMPUTE VALUES
132.         """imputer = Imputer(inputCols=[column for column in dataset.columns if col
    umn not in time_cols],
133.                              outputCols=["{}_imputed".format(c) for c in [column for
    column in dataset.columns if column not in time_cols]],
134.                              strategy = "median")
135.         dataset = imputer.fit(dataset).transform(dataset)
136.         dataset = dataset.dropna()"""
137.
138.         #Drop Columns
139.         drop_list_fil = [col for col in self.inputCols if col != "Close_aapl"]
140.         drop_list = [col for col in drop_list_fil]
141.
```

```
142.         dataset = dataset.select([col for col in dataset.columns if col not in drop
    _list])
143.
144.         #PREDICTION WINDOW
145.         dataset = dataset.withColumn("Close_aapl_Window",lead("Close_aapl",self.pre
    d_window-1,None).over(Window.orderBy("Date")))
146.         dataset = dataset.drop("Close_aapl")
147.         dataset = dataset.withColumnRenamed("Close_aapl_Window","Close_aapl")
148.         dataset  = dataset.withColumnRenamed("Close_aapl","label")
149.
150.         #DROP NULL CREATED BY
151.         dataset = dataset.dropna()
152.
153.         #REMOVE NON IMPUTED COLUMNS
154.         #dataset  = dataset.select([col for col in dataset.columns if ("_imputed" i
    n col) or (col in time_cols)])
155.
156.         #self.outputCols = dataset.columns
157.
158.         return dataset
159.
160.
161. # COMMAND ----------
162.
163. #FUNCTIONS
164. def evaluateModel(model,data):
165.   pred_results = model.transform(data)
166.
167.   pred_results = pred_results.withColumn("squared_error",pow((col("label") - col("p
    rediction")),2))
168.   pred_results = pred_results.withColumn("s_abs_percentage_error",(abs(col("predict
    ion") - col("label"))/((col("label") + col("prediction"))/2))*100)
169.
170.   total_rmse = pred_results.select(sqrt(mean(col("squared_error")))).collect()
171.   total_smape = pred_results.select(mean(col("s_abs_percentage_error"))).collect()

172.
173.   #print("Train RMSE: ", model.bestModel.summary.rootMeanSquaredError)
174.   print("Test RMSE: ", total_rmse)
175.   print("Test sMAPE: ", total_smape)
176.
177.   return pred_results
178.
179. def ExtractFeatureImp(featureImp, dataset, featuresCol):
```

```
    list_extract = []
    for i in dataset.schema[featuresCol].metadata["ml_attr"]["attrs"]:
        list_extract = list_extract + dataset.schema[featuresCol].metadata["ml_attr"]["attrs"][i]
    varlist = pd.DataFrame(list_extract)
    varlist['score'] = varlist['idx'].apply(lambda x: featureImp[x])
    return(varlist.sort_values('score', ascending = False))


# COMMAND ----------

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler, StandardScaler, OneHotEncoderEstimator, PCA, VectorSlicer, ChiSqSelector
from pyspark.ml.regression import DecisionTreeRegressor, GBTRegressor, RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import pandas as pd

time_cols = ["Date",
             "Day_Of_Week",
             "Month",
             "Quarter",
             "Week_Of_Year",
             "Day_Of_Month"]

#READ DATA
sqlContext = SQLContext(sc)
df = sqlContext.sql("""
                    SELECT aapl.Date, aapl.Open as Open_aapl, aapl.High as High_aapl, aapl.Low as Low_aapl, aapl.Close as Close_aapl, aapl.Volume as Volume_aapl,
                    sp500_csv.Open as Open_sp500, sp500_csv.High as High_sp500, sp500_csv.Low as Low_sp500, sp500_csv.Close as Close_sp500, sp500_csv.Volume as Volume_sp500
                    FROM aapl
                    JOIN sp500_csv on aapl.Date = sp500_csv.Date
                    WHERE YEAR(aapl.Date) >= 2009 and YEAR(aapl.Date) <= 2015
                    """)

#msft.Open as Open_msft, msft.High as High_msft, msft.Low as Low_msft, msft.Close as Close_msft, msft.Volume as Volume_msft
#JOIN msft on aapl.Date = msft.Date

```

```
217.
218. #BUILD FEATURES
219. #Pred windows 1, 5, 10, 20, 60
220. stockcreator = StockFeatureCreator(lags = 10, pred_window = 1, ma_windows = [3,5,10
    ,20,50,80,100], tickers = ["aapl","sp500"])
221. stockcreator.build()
222.
223. #ONE HOT CATEGORICAL DATE FEATURES
224. inputs = [s for s in time_cols if s not in ["Date"]]
225. encoder = OneHotEncoderEstimator(inputCols=inputs, outputCols=[s + "_Vec" for s in
    time_cols if s not in ["Date"]])
226.
227. #VECTOR ASSEMBLER
228. features = stockcreator.getOutputCols() + encoder.getOutputCols() #getOutoutCols re
    turning empty list
229. features = [col for col in features if (col != "label") or (col not in time_cols)]

230. featureassembler = VectorAssembler(inputCols=features,outputCol="features")
231.
232. #SPLIT
233. finalized_data = df.withColumn("rank",percent_rank().over(Window.partitionBy().orde
    rBy("Date")))
234. train_data = finalized_data.where("rank <= .9").drop("rank")
235. test_data = finalized_data.where("rank > .9").drop("rank")
236.
237. #FEATURE SELECTION
238. selector = ChiSqSelector(numTopFeatures=300, featuresCol="features",
239.                           outputCol="selectedFeatures", labelCol="label")
240.
241. '''''num_features = 50
242.
243. fs_estimator = RandomForestRegressor(labelCol="label", featuresCol="all_features")
244. pipe1 = Pipeline(stages=[stockcreator, encoder, featureassembler, fs_estimator])
245. model = pipe1.fit(train_data)
246. df2 = model.transform(test_data)
247.
248. varlist = ExtractFeatureImp(model.stages[-
    1].featureImportances, df2, "all_features")
249. varidx = [x for x in varlist['idx'][0:num_features]]
250. slicer = VectorSlicer(inputCol="all_features", outputCol="features", indices=varidx
    )
251. df2.unpersist()'''
252.
253. #FIT MODEL
```

```
estimator = GBTRegressor(labelCol="label", featuresCol="selectedFeatures") #can be changed to RF, or RLR - make sure to change param grid for each model

paramGrid = ParamGridBuilder().addGrid(estimator.maxDepth, [2,3,5]).addGrid(estimator.subsamplingRate, [0.8,0.7]).addGrid(estimator.maxIter, [20,30,50]).addGrid(selector.numTopFeatures, [200,300,400]).build()

pipe2 = Pipeline(stages=[stockcreator, encoder, featureassembler, selector, estimator])

crossval = CrossValidator(estimator=pipe2,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator().setMetricName("rmse"),
                          numFolds=3)

cvModel = crossval.fit(train_data)

#PRINT RESULTS
params = [{p.name: v for p, v in m.items()} for m in cvModel.getEstimatorParamMaps()]

pd.DataFrame.from_dict([
    {cvModel.getEvaluator().getMetricName(): metric, **ps}
    for ps, metric in zip(params, cvModel.avgMetrics)
])


# COMMAND ----------

# MAGIC %md Example Runs (Not Including all tests for all models)

# COMMAND ----------

# MAGIC %md 1 Day GBT 300 Features

# COMMAND ----------

results = evaluateModel(cvModel.bestModel,train_data)
ExtractFeatureImp(cvModel.bestModel.stages[-1].featureImportances, results, "selectedFeatures")

# COMMAND ----------

# MAGIC %md 4 Month GBT 300 Features
```

```
292.
293. # COMMAND ----------
294.
295. results = evaluateModel(cvModel.bestModel,test_data)
296. ExtractFeatureImp(cvModel.bestModel.stages[-
    1].featureImportances, results, "selectedFeatures")
297.
298. # COMMAND ----------
299.
300. results.select("features","label","prediction").show()
301.
302. # COMMAND ----------
303. #pandas_results = results.select("label","prediction").toPandas()
304. #display(pandas_results)
```