Model Comparison in Stock Return Forecasting

Ali Al-Jabri(aha2166)[1], Jose Gabriel Castillo(jgc2141)[2], Kaiwen Chen (kc3309)[3], Boyuan Tao (bt2524)[3], Tao Wang(tw2758)[4], Yutian Yao(yy3043)[3], and Zilu Yu(zy2379)[3]

[1]Columbia University, Teachers College, New York City, United States,
[2]Columbia University, School of General Studies, New York City, United States
[3]Columbia University, Graduate School of Arts and Sciences, New York City, United States
[4]Columbia University, School of Professional Studies, New York City, United States

ABSTRACT

Stock holders expect good prediction in stock markets since good prediction of return brings low risk. However, the accuracy of different models may vary. ARIMA, GARCH are traditional time series methods to forecast stock return, and SVM model is another model discussed. These models are applied on same stock in this project and testing their performance. They behave differently on this particular stock.

Key points: stock prediction, ARIMA, GARCH, time series model, SVM

Data

Coca-Cola's (NYSE: KO) stock prices in the period of 01/01/2016 to 01/01/2020 from Yahoo Finance[1]. Analyzed in R 3.6.2. and Python 3.8.

Introduction

The stock volatility is influenced by various economic conditions and factors. These factors are far complicated; therefore, the investors will expect pure analysis on stock values can help them to lower risk of investment. Then, it is expected to provide proper statistical models to predict future values of stocks. In order to evaluating goodness of models' behavior, the stock data in previous years is analyzed and subsequent data can reflects how models behave.

Two time series models where one has the ability to capture the non-constant volatility throughout the observations are used. These models are Autoregressive Integrated Moving Average (ARIMA) and Generalized Autoregressive Conditional Heterocedasticity (GARCH). These models will be used to fit the data where the best model will be used to forecast the future of forecasting market values[2]. GARCH is used to forecast the volatility of time series. Combined with ARIMA, it can give prediction about both the mean and variance of the stock return.

A support vector machine (SVM) is a a supervised machine learning model, which employs Support Vector Regression (SVR). SVM can take data as input, attempts to find and recognize patterns. The support vector machine algorithm comes from the maximal margin classifier. The maximal margin classifier uses the distance from a given decision boundary to classify an input. The greater the distance, or margin, the better the classifier is at handling the data[3]. In finance problem, the data from Yahoo can be scraped and used to create the features for models, which helps technical analysis.

Objectives and Plan

Based on the data of Coco-Cola from 01/01/2016 to 01/01/2020, the objective is to predict the return in the next 30 days, which is as the testing set in this project. Three models is expected to behave quite differently since they are based on different statistic theory. The models are implemented by computing program R and Python. According to different models, the return forecasting will be based on calculation on different features, for example, ARIMA on return and SVM on price.

Methods

ARIMA model

First, we need to check availability of data, which has been transformed. We first want to test if the daily log returns series is stationary so that it can be used for the ARIMA model. Then ARIMA function in package can be applied to get the parameters of model. Here, two functions are applied in order to get better choice of parameter. In order to check the selection of

parameters, the ACF and PACF function are applied to the training set of data. Order q of the MA process and order p of the AR process can be shown by the ACF plot and PACF plot respectively. After that, the prediction of price can be got and used to compare with the actual price.

GARCH model

Firstly AIC criteria is used to determine the optimal parameters for ARMA and GARCH models according to heat map. Then, we use qq-plot to identify how good the models are after applying the models on the training set. After that, we can get the prediction and compare it with real price.

SVM

Instead of using log returns, we found using daily stock prices is more suitable for SVM. The data, Coca-Cola stock prices, is again splitted in the same way as for the ARIMA model so that we will forecast for 30 days. Then we fitted our testing data to the model and obtained the prediction of stock prices with the functions in package.

Results

ARIMA model

After transforming the data to daily log return Figure 1, we applied an ADF test to our data. The resulting p-value is 0.01, smaller than 0.05, showing that the series is stationary and ready to be modeled. Then we splitted the data set at the break point of 976 so that we have a training set for 976 days and a testing set for 30 days. To choose the best parameters for the ARIMA model, we used the auto.arima function in R and found that (2, 0, 4) was the best based on AIC criteria Figure 2. In order to check the selection of our parameters, we applied the ACF and PACF function to the training set of data Figure 1.

As the parameters (p, d, q) were determined to be (2, 0, 4), ARIMA model can be bulit with training data. It produced the coefficients in Figure 2. When using ARIMA model with parameter (2, 0, 4), the forecasting price is plotted in Firgre 2, and accuracy of prediction is 53.33%.

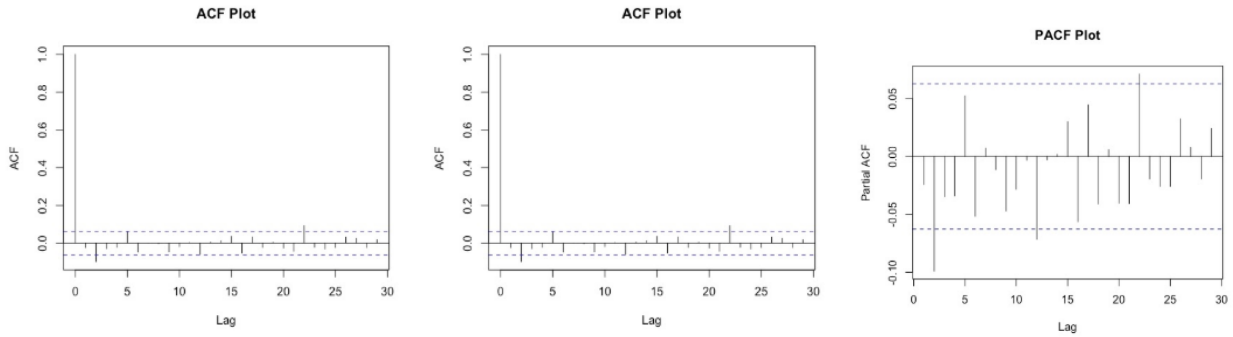

Figure 1. Log return is transformation of stock Price. ACF and PACF plots check test crtieria of parameters of the model.

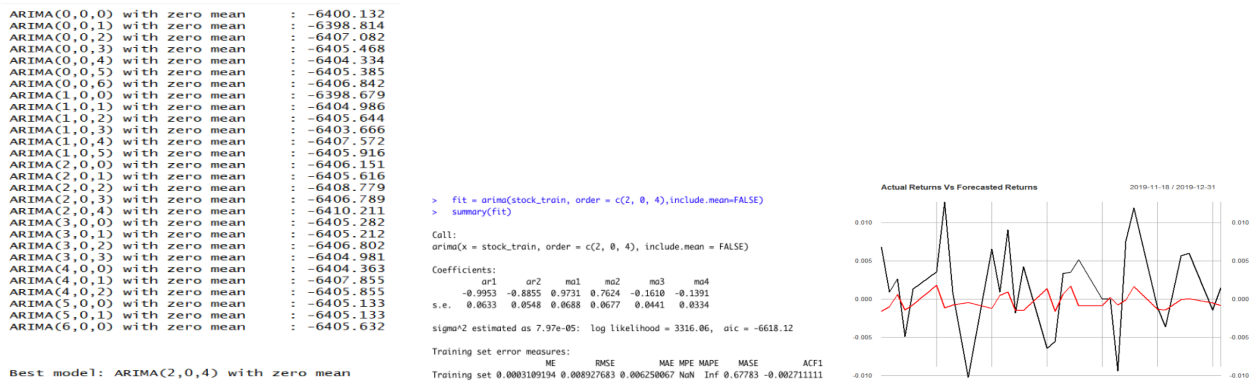

Figure 2. ARIMA function gives all criteria for selection. Predicted price is in red and actual price is in black.

GARCH model

Heat maps are shown below: After applying training set, we can see qq-plot in Figure 4, it fits the expected distribution well.

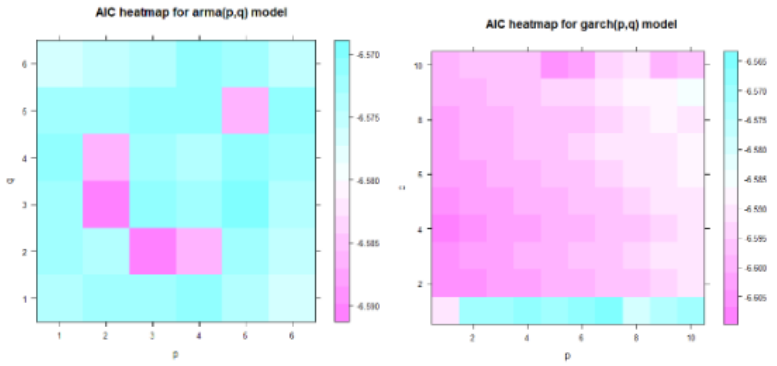

Figure 3. The optimal model is ARMA(2, 3) + GARCH(1, 4).

```
Error Analysis:
         Estimate  Std. Error  t value Pr(>|t|)
mu     8.662e-05   6.387e-05    1.356   0.1750
ar1   -3.476e-02   1.286e-01   -0.270   0.7869
ar2    7.043e-01   9.786e-02    7.198 6.13e-13 ***
ma1    4.591e-02   1.336e-01    0.344   0.7312
ma2   -7.870e-01   8.647e-02   -9.102  < 2e-16 ***
ma3   -3.173e-02   3.997e-02   -0.794   0.4272
omega  4.396e-07   5.315e-07    0.827   0.4082
alpha1 8.221e-02   1.867e-02    4.404 1.06e-05 ***
beta1  1.000e-08   9.855e-02    0.000   1.0000
beta2  2.742e-01   1.661e-01    1.651   0.0987 .
beta3  1.000e-08   9.343e-02    0.000   1.0000
beta4  6.446e-01   1.516e-01    4.253 2.11e-05 ***
```

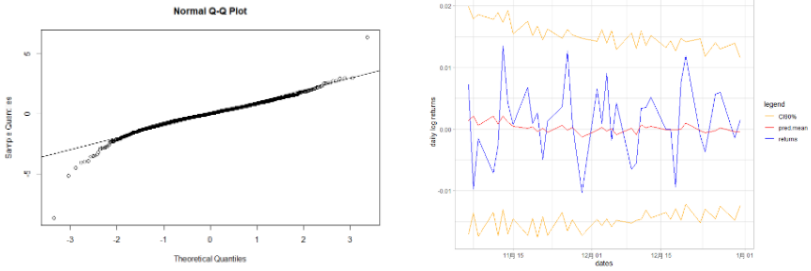

Figure 4. The prediction accuracy is calculated to be 52.6%.

SVM

We used radial basis function when building the SVR and fitted the model as follows:

```
In [18]: svr_rbf = SVR(kernel='rbf', C=1000, gamma=0.1)
         svr_rbf.fit(x_train, y_train)

Out[18]: SVR(C=1000, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.1,
             kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

In [20]: svm_confidence = svr_rbf.score(x_test, y_test)
         print('svm_confidence:', svm_confidence)
         svr_rbf.predict(x_test) # predicted prices

         svm_confidence: 0.9159966788664171
```

From the plot of forecasting (red line) and actual (grey line) stock prices in Figure 5, there are largely overlapped showing a relatively high level of accuracy.

Conclusion

From ARIMA plot in Figure 2, it is observed that generally the forecasting returns (red line) move in the same direction of the actual returns(black line). However, the ARIMA(2, 0, 4) model seems to be lacking when it comes to forecasting the exact returns. To examine the performance of our ARIMA model, the accuracy rate of its forecast shows the model is said to be accurate either the model forecasts positive return when the actual return is positive or it forecasts negative return when the actual return is negative since accuracy ratio is 53.33%. In order to diagnose this model more precisely, ACF and residual can be checked Figure 6. Forecasting returns (red line) does not follow the trend of actual returns well, although its accuracy is close to ARIMA model. Although the fluctuation of price is not as large as the actual price in ARIMA model, the direction and the trend are approximately matching, so it may also be a good model for stockholder to make trade decision.

ARIMA Model and GARCH models are better to predict the stock price than stock return. Support Vector Machines (SVM), however, works perfect for stock return forecasting.

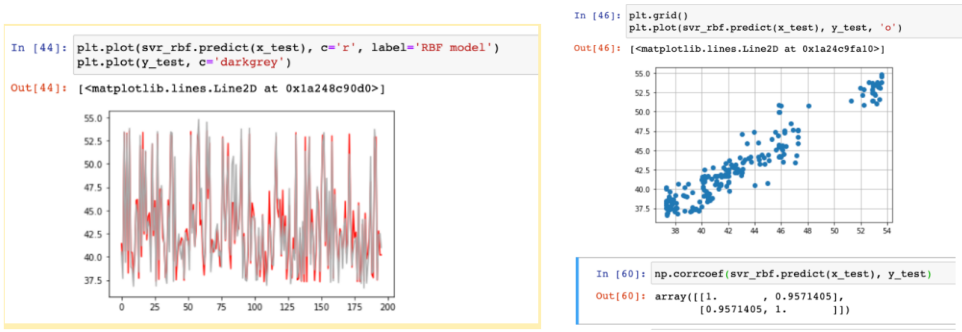

Figure 5. An alternative way we used to access the performance of SVM is to check for the correlation between predicted prices and actual prices. From the plot, we observe strong positive correlation (0.9571) between them.

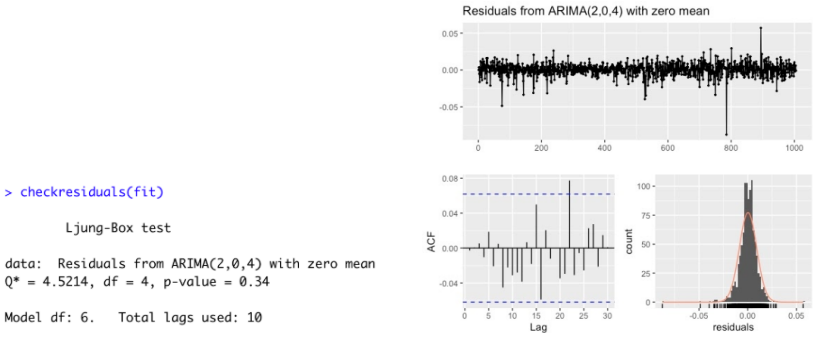

Figure 6. Residual diagnostics on the model should, ideally, satisfy the properties that they are uncorrelated and have zero mean. Also, the large p-value from the Ljung-Box test and the plots indicate that residuals of our model do behave like white noise and show no obvious correlation.

References

1. Coca-cola company (the) (ko) stock historical prices and data (2020).
2. Miswan, N. H. *Modelling and forecasting volatile data by using ARIMA and GARCH models* (2013).
3. Cao, L.-J. & Tay, F. E. H. Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Transactions on neural networks* **14**, 1506–1518 (2003).

Attachment

ARIMA code

```
library(quantmod)
library(tseries)
library(timeSeries)
library(forecast)
library(xts)

# Pull data from Yahoo finance
getSymbols('KO', from='2016-01-01', to='2020-01-01')
head(KO)

# Select the closing price
stock_p <- KO[,4]
head(stock_p)

# Daily log Return
stock_r <- periodReturn(stock_p, period = "daily", type = "log")
head(stock_r)
dim(stock_r)

# Plot log returns
plot(stock_r, type='l', main='log returns plot')


# Conduct ADF test on log returns
print(adf.test(stock_r))

# the small P-value indicate the log returns are stationary

# Split the dataset in two parts - training and testing
breakpoint = floor(nrow(stock_r)*(.971))

train = stock_r[c(1:breakpoint),]


# use the auto.arima to choose the best ARIMA Parameters
fit1 = auto.arima(train,d=0,max.p=10,max.q=10,max.order=6,ic="aic",
                  seasonal=FALSE,stepwise=FALSE,trace=TRUE,
                  approximation=FALSE,allowdrift=FALSE,allowmean=FALSE)


# ARIMA(2,0,4) fits the best usign AIC criteria




# Apply the ACF and PACF functions
par(mfrow = c(2,1))
acf.stock = acf(stock_r[c(1:breakpoint),], main='ACF Plot')
pacf.stock = pacf(stock_r[c(1:breakpoint),], main='PACF Plot')



# Initialzing an xts object for Actual log returns
Actual_series = xts(0,as.Date("2019-11-15","%Y-%m-%d"))

# Initialzing a dataframe for the forecasted return series
forecasted_series = data.frame(Forecasted = numeric())
```

```
for  (n in breakpoint:(nrow(stock_r)-1)) {

  stock_train = stock_r[1:n, ]
  stock_test = stock_r[(n+1):nrow(stock_r), ]

  # Summary of the ARIMA model using the determined (p,d,q) parameters
  fit = arima(stock_train, order = c(2, 0, 4),include.mean=FALSE)
  summary(fit)


  # Forecasting the log returns
  arima.forecast = forecast(fit, h = 1,level=95)
  summary(arima.forecast)


  # Creating a series of forecasted returns for the forecasted period
  forecasted_series = rbind(forecasted_series,arima.forecast$mean[1])
  colnames(forecasted_series) = c("Forecasted")

  # Creating a series of actual returns for the forecasted period
  Actual_return = stock_r[(n+1),]
  Actual_series = c(Actual_series,xts(Actual_return))
  rm(Actual_return)

}


# plotting the forecast
par(mfrow=c(1,1))
plot(arima.forecast, main = "ARIMA Forecast")


# Adjust the length of the Actual return series
Actual_series = Actual_series[-1]

# Create a time series object of the forecasted series
forecasted_series = xts(forecasted_series,index(Actual_series))

# Create a plot of the two return series - Actual versus Forecasted
plot(Actual_series,type='l',main='Actual Returns Vs Forecasted Returns',
col = "black")
lines(forecasted_series, lwd=2,col='red')

# Create a table for the accuracy of the forecast
comparsion = cbind(Actual_series,forecasted_series)
comparsion$Accuracy =
sign(comparsion$Actual_series)==sign(comparsion$Forecasted)
print(comparsion)

# Create a table for the accuracy
table(comparsion$Accuracy)


# the accuracy percentage
Accuracy_percentage = (sum(comparsion$Accuracy == 1)/
length(comparsion$Accuracy))*100
print(Accuracy_percentage)
```

GARCH code

```
library(fGarch)
library(quantmod)
library(lattice)
library(ggplot2)

# get data
getSymbols('KO', from='2015-01-01', to='2020-01-01')
head(KO)
KO.price <- KO$KO.Close

# get return
KO.returns <- periodReturn(KO.price, period = "daily", type = "log")

# Splitting into train and test sets.
KO.split = round(nrow(KO.returns)*(.97))
train = KO.returns[c(1:KO.split),]
test = KO.returns[c((KO.split+1):length(KO.returns)),]

# find optimal parameter for mean model using AIC
AICS.arma<-matrix(nrow = 6,ncol = 6)
for (i in c(1:6)){
  for (j in c(1:6)){
    formular.i.j<-c("~arma(",i,",",j,")+garch(1,1)",sep = "")
    as.formula(formular.i.j)
    fit.i.j <- garchFit(as.formula(formular.i.j),data = train,trace =
FALSE)
    fits.slot<-slot(fit.i.j,"fit")
    AICS.arma[i,j]<-fits.slot$ics["AIC"]
    print(c(i,j))
  }
}

levelplot(AICS.arma,xlab = "p",ylab = "q",main = "AIC heatmap for arma(p,q)
model")

c(which.min(AICS.arma)%%6,floor(which.min(AICS.arma)/6)+1)
min(AICS.arma)

# arma(2,3) is optimal acording to AIC. Next find optimal parameters for
GARCH(p',q') model

AICS<-matrix(nrow = 10,ncol = 10)
for (i in c(1:10)){
  for (j in c(1:10)){
    formular.i.j<-c("~arma(2,3)+garch(",i,",",j,")",sep = "")
    as.formula(formular.i.j)
    fit.i.j <- garchFit(as.formula(formular.i.j),data = train,trace =
FALSE)
    fits.slot<-slot(fit.i.j,"fit")
    AICS[i,j]<-fits.slot$ics["AIC"]
    print(c(i,j))
  }
}

levelplot(AICS,xlab = "p",ylab = "q",main = "AIC heatmap for garch(p,q)
model")

c(which.min(AICS)%%10,floor(which.min(AICS)/10)+1)
```

```
min(AICS)

# AIC shows optimal parameter for garch is (). Next do prediction using the
optimal parameters above.

opt.fit <- garchFit(~arma(2,3)+garch(1,4),train,trace = FALSE)
summary(opt.fit)
prediction <- c()
for (i in c(0:((length(KO.returns)-KO.split)-1))){
  fit.i <- garchFit(~arma(2,4)+garch(1,4),KO.returns[1:(KO.split+i)],trace
= FALSE)
  predict.i <- predict(fit.i,1)
  prediction <- rbind(prediction,predict.i)
}

pred<-prediction$meanForecast
pred.u.5<-prediction$meanForecast+prediction$standardDeviation*qnorm(0.95)
pred.d.5<-prediction$meanForecast+prediction$standardDeviation*qnorm(0.05)

data.prediction <- data.frame(dates = index(test),
                              daily.returns = test$daily.returns,
                              pred = pred,
                              pred.u = pred.u,
                              pred.d = pred.d)

# plot of predicted mean and predicted 90% CI boundaries
ggplot(data = data.prediction,aes(x = dates))+
  geom_line(aes(y = daily.returns,color = "returns"))+
  geom_line(aes(y = pred,color = "pred.mean"))+
  geom_line(aes(y = pred.u,color = "CI90%"))+
  geom_line(aes(y = pred.d,color = "CI90%"))+
  scale_color_manual(values = c( "returns" = "blue",
                                 "pred.mean" = "red",
                                 "CI90%" = "orange"))+
  labs(x = "dates",
       y = "daily log returns",
       color = "legend")+
  theme_light()

pred.accu <-
sign(data.prediction$daily.returns)==sign(data.prediction$pred)
accu<-sum(pred.accu)/length(pred.accu)
accu

# other plots of the result
# ACF
plot(fit.i,which = 4)
# QQ-plot of standardized residuals on train set
plot(fit.i,which = 13)
# mannually plot the QQ-plot
fit.residuals<-residuals(fit.i,standardize = TRUE)
qqnorm(fit.residuals)
qqline(qnorm(c(1:length(fit.residuals))/
length(fit.residuals)),fit.residuals)
# QQ-plot of standardized residuals on test set
residuals.standardized<-scale((data.prediction$daily.returns-
data.prediction$pred)/prediction$standardDeviation)
qqnorm(residuals.standardized)
```

```
qqline(qnorm(c(1:length(residuals.standardized))/
length(residuals.standardized)),residuals.standardized)
```

SVM code


```
{
  "cells": [
    {
      "cell_type": "code",
      "execution_count": 2,
      "metadata": {},
      "outputs": [
        {
          "name": "stderr",
          "output_type": "stream",
          "text": [
            "/opt/anaconda3/lib/python3.7/site-packages/pandas_datareader/
compat/__init__.py:7: FutureWarning: pandas.util.testing is deprecated. Use
the functions in the public API at pandas.testing instead.\n",
            "  from pandas.util.testing import assert_frame_equal\n"
            ]
        }
        ],
      "source": [
        "import numpy as np\n",
        "import pandas as pd\n",
        "import pandas_datareader as dr\n",
        "from matplotlib import pyplot as plt\n",
        "%matplotlib inline\n",
        "import datetime as dt\n",
        "from sklearn.linear_model import LinearRegression\n",
        "from sklearn.svm import SVR\n",
        "from sklearn.model_selection import train_test_split\n"
        ]
    },
    {
      "cell_type": "code",
      "execution_count": 30,
      "metadata": {},
      "outputs": [
        {
          "data": {
            "text/html": [
              "<div>\n",
              "<style scoped>\n",
              "    .dataframe tbody tr th:only-of-type {\n",
              "        vertical-align: middle;\n",
              "    }\n",
              "\n",
              "    .dataframe tbody tr th {\n",
              "        vertical-align: top;\n",
              "    }\n",
              "\n",
              "    .dataframe thead th {\n",
              "        text-align: right;\n",
              "    }\n",
              "</style>\n",
              "<table border=\"1\" class=\"dataframe\">\n",
              "  <thead>\n",
              "    <tr style=\"text-align: right;\">\n",
              "      <th></th>\n",
              "      <th>High</th>\n",
              "      <th>Low</th>\n",
```

```
"      <th>Open</th>\n",
"      <th>Close</th>\n",
"      <th>Volume</th>\n",
"      <th>Adj Close</th>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>Date</th>\n",
"      <th></th>\n",
"      <th></th>\n",
"      <th></th>\n",
"      <th></th>\n",
"      <th></th>\n",
"      <th></th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
```