Proceedings of the 6th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/92/20241728

Using machine learning to predict Apple Inc.'s stock returns and develop a hedging strategy for investors

Yihang Han

Alliance Manchester Business School, the University of Manchester, Oxford Road, Manchester, M13 9PL, UK

hyh1651035453@outlook.com

Abstract. This report aims at investigating the relationship between the returns of Apple Inc. and the market returns by using the machine learning model. Nine months of historical financial data for Apple and market was utilized as the training data set, predicting the return of Apple in future three months. Within this research, my findings suggest that there is a significantly positive correlation between Apple's returns and the market returns, which indicates that it is acceptable to forecast Apple's performance through the market index, and any market fluctuation tends to have a pronounced impact on Apple. Based on the effective analysis of Apple's future returns, we can propose valid hedging strategies for investors to eliminate risks from both upward and downward price trends.

Keywords: Apple Inc, Linear Regression, Data Frame, Hedging, Cross-Validation.

1. Introduction

In today's complex financial market, it is inadequate to anticipate the trend of the stock by solely observing its recent fluctuation. Instead, the combination of artificial intelligence and advanced mathematical models is getting more essential for forecasting the possibility of stock in future days. In this research, I focus on using the machine learning model and fundamental programming commands in Python to estimate the future returns of Apple Inc. By examining the correlation between two stocks, the research provides insight about how market changes influence Apple's return. Due to the potential bias and inaccuracy within the model, it is also significant to assess its operating performance. A widely recognized metric for this is the R-Square value. In models related to the discussion of price-related factors, a higher R-Square always indicates the higher level of price efficiency [1]. In most cases, evaluating the R^2 would be a necessary step in model testing, ensuring the feasibility of our model and robustness of our findings throughout the whole research. In the following sections, we will discuss the methodology employed for data collection and analysis, the results from the machine learning model, the implications of findings, and potential solution for enhancing model performance in the future research.

2. Methodology

To evaluate the impact of market returns on Apple, I employed the linear regression model to analyze the relationship between the market return and Apple. Within the model, the slope and intercept of the line are principal parameters that we need for this prediction.

DOI: 10.54254/2755-2721/92/20241728

2.1. Creating Data Frame for Training Data

Initially, data regarding historical prices for Apple and S&P 500 were already prepared, then the data frame included price levels for training data set can be created. In this procedure, date ranged from 2018-01-02 to 2018-09-28 was selected, which can be achieved by just applying the *'getRange( )'* function, extracting a specific period from the entire database. Figure 1 illustrates the detail about how this process worked to print the result that we need.

```
def getRange(df, start_dt, end_dt):

    start_dt = pd.to_datetime(start_dt)
    end_dt = pd.to_datetime(end_dt)

    # check if the DataFrame's index is of type DatetimeIndex
    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    # find a date that is close to the start date
    date_before_start = df.index[df.index <= start_dt].max()

    if pd.isna(date_before_start):
        raise ValueError('no data available before the start date')

    # ensuring that the index is between two dates in the dataframe
    filtered_df = df.loc[date_before_start:end_dt]

    return filtered_df


train_data_price = getRange(data, start_dt, end_dt)

print(train_data_price[:])
```

```
            AAPL_Adj_Close  SPY_Adj_Close
Dt
2018-01-02        167.1999       260.1310
2018-01-03        167.1708       261.7763
2018-01-04        167.9473       262.8796
2018-01-05        169.8594       264.6314
2018-01-08        169.2285       265.1154
...                    ...            ...
2018-09-24        216.7654       285.3496
2018-09-25        218.1399       285.0848
2018-09-26        216.4021       284.2318
2018-09-27        220.8496       285.0260
2018-09-28        221.6252       285.0555

[188 rows x 2 columns]
```

Figure 1. Creating Training Data Frame

2.2. Creating Data Frame for Test Data

Similarly, the test data frame can be created under the same principle, by simply substituting the start and end date of training data with those of the test data, as shown in figure 2.

DOI: 10.54254/2755-2721/92/20241728

```
test_start_dt = "2018-10-01"
test_end_dt = "2018-12-31"
test_data_price = None

def getRange(df, start_dt, end_dt):

     # YOUR CODE HERE
    start_dt = pd.to_datetime(start_dt)
    end_dt = pd.to_datetime(end_dt)

    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    date_before_start = df.index[df.index <= start_dt].max()

    if pd.isna(date_before_start):
        raise ValueError('no data available before the start date')

    filtered_df = df.loc[date_before_start:end_dt]

    return filtered_df


test_data_price = getRange(data, test_start_dt, test_end_dt)

print(test_data_price[:])
```

```
            AAPL_Adj_Close  SPY_Adj_Close
Dt
2018-10-01        223.1175       286.0458
2018-10-02        225.1006       285.8791
2018-10-03        227.8398       286.0359
2018-10-04        223.8342       283.8004
2018-10-05        220.2016       282.2119
...                    ...            ...
2018-12-24        144.6565       231.1158
2018-12-26        154.8435       242.7929
2018-12-27        153.8386       244.6569
2018-12-28        153.9174       244.3412
2018-12-31        155.4050       246.4814

[63 rows x 2 columns]
```

Figure 2. Creating Test Data Frame

2.3. Calculating Returns for Training Set

Subsequently, returns between two adjacent days can be evaluated by using the above prepared price data,

$$r_{AAPL} = \frac{P_t^{AAPL} - P_{t-1}^{AAPL}}{P_{t-1}^{AAPL}},$$

where r_{AAPL} is the return of Apple, P_t^{AAPL} is the current price and P_{t-1}^{AAPL} is the price before the current day.

I introduced the *'filterData( )'* function, which conducts the similar purpose to *'getRange( )'*, removing the unnecessary data from the database, then, the *'getReturns( )'* function was utilized to calculate Apple's returns. Specifically, within the *'getReturns( )'* function, the *'pct_change( )'* method calculates the percentage change between the current and prior price in the Pandas data frame. In addition, the *'dropna( )'* method removes any rows or columns containing missing values *'NaN'*. These basic commands make it available to compute returns over those specific dates. Figure 3 shows the overview of the expected returns for stocks, accompanied with the necessary code for return calculation.

DOI: 10.54254/2755-2721/92/20241728

```
def filterData(df, start_dt, end_dt):

    start_dt = pd.to_datetime(start_dt)
    end_dt = pd.to_datetime(end_dt)

    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    date_before_start = df.index[df.index <= start_dt].max()

    if pd.isna(date_before_start):
        raise ValueError('No data available before the start date')

    filtered_df = df.loc[date_before_start:end_dt]
    return filtered_df


def getReturns(df):

    returns_df = df.pct_change().dropna()
    return returns_df


filtered_data = filterData(train_data_price, start_dt, end_dt)


train_data_ret = getReturns(filtered_data)

print(train_data_ret[:])
```

```
            AAPL_Ret   SPY_Ret
Dt
2018-01-03 -0.000174  0.006325
2018-01-04  0.004645  0.004215
2018-01-05  0.011385  0.006664
2018-01-08 -0.003714  0.001829
2018-01-09 -0.000115  0.002263
...              ...       ...
2018-09-24  0.014380 -0.003322
2018-09-25  0.006341 -0.000928
2018-09-26 -0.007966 -0.002992
2018-09-27  0.020552  0.002794
2018-09-28  0.003512  0.000103

[187 rows x 2 columns]
```

Figure 3. Evaluating the Training Data Returns

2.4. Removing the Target and Transforming Test Data Returns

Given that both Apple and market prices were included in the training data set, it would be misleading to predict Apple's future returns by looking at the historical Apple prices. The target variable, which is known as Apple's return in this research, should not be included in the independent variables. Therefore, information regarding Apple's past returns must be separated from the regression, ensuring that the forecast of Apple is solely based on the historical market index, i.e. the model only learnt from the market index to make the further prediction. Figure 4 illustrates how this process of separation works.

```
tickerAttr = ticker + "_Ret"

X_train, y_train =  train_data_ret.drop(columns=[tickerAttr]), train_data_ret[[ tickerAttr ]]
```

Figure 4. Separating the Feature from the Target

Code in figure 4 splits the predictor *'X_train'* from *'y_train'*, where *'X_train'* is the predictor, SPY, and *'y_train'* is the target variable, Apple. Then, the test-data returns can be acquired by replicating the above-mentioned process of return evaluation, as shown in figure 5 below.

```
test_start_dt = '2018-10-02'
test_end_dt = '2018-12-31'

test_data_price = getRange(data, test_start_dt, test_end_dt)

test_data_ret = getReturns(test_data_price)

tickerAttr = 'AAPL_Adj_Close'

X_test = test_data_ret.drop(columns=[tickerAttr])
y_test = test_data_ret[[tickerAttr]]

print(test_data_ret[:])
```

```
            AAPL_Ret   SPY_Ret
Dt
2018-10-03  0.012169  0.000548
2018-10-04 -0.017581 -0.007815
2018-10-05 -0.016229 -0.005597
2018-10-08 -0.002318  0.000000
2018-10-09  0.013854 -0.001459
...              ...       ...
2018-12-24 -0.025874 -0.026423
2018-12-26  0.070422  0.050525
2018-12-27 -0.006490  0.007677
2018-12-28  0.000512 -0.001290
2018-12-31  0.009665  0.008759

[61 rows x 2 columns]
```

Figure 5. Transformation of Test Data Returns

DOI: 10.54254/2755-2721/92/20241728

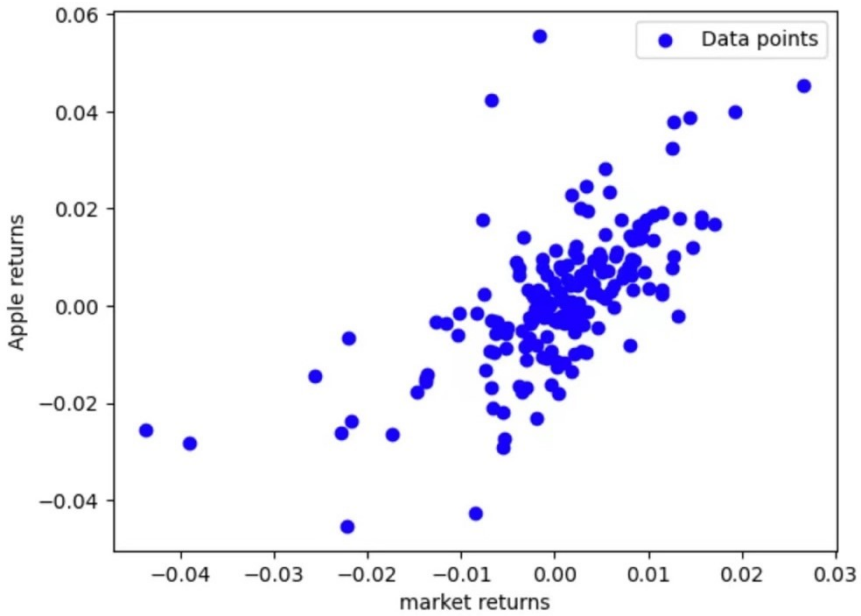

Figure 6. Scatter Diagram for Training Sample Returns

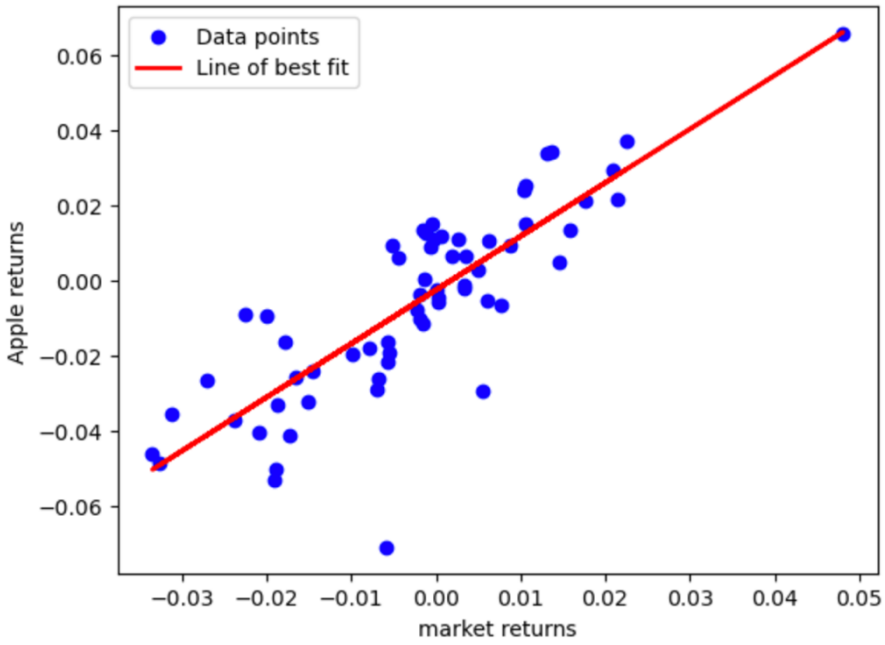

Figure 7. Scatter Diagram for Test Sample Returns

In figure 6, daily returns for Apple Inc. and the market primarily range between 0 and 0.02. However, in the test sample, which is shown in figure 7, both two stocks exhibit worse performance with an increased distribution of negative returns. This trend suggests that the market returns are likely to be negative in the future. If we regard this market index as a benchmark for predicting Apple's returns, a negative return for Apple can finally be anticipated.

While the discussion of returns here is not directly related to the analysis of following machine learning model, our ultimate purpose is to scheme strategies for investors. Therefore, the analysis of return here serves as an indication of the hedging strategy discussed in Part 3 below. If the expectation

DOI: 10.54254/2755-2721/92/20241728

of future Apple's return is negative and potential investors are concerned about further decline in stock prices, purchasing a put option on Apple can be considered. It allows the stock to be sold at a relatively higher predetermined price when they suffer from a continuous drop in stock value.

2.5. Introduction of the Linear Regression Model

After calculating the returns within the test period from '2018-10-03' to '2018-12-31', we are going to design a linear regression model which incorporates parameters related to the historical return of market and expected return of Apple.

$$r^t_{AAPL} = \beta_0 + \beta_{SPY} * r^t_{SPY} + \varepsilon^t_{AAPL}$$

Two functions are worth mentioning in this step, *'createModel( )'* and *'regress( )'*. In simple terms, *'createModel( )'* is utilized to construct the linear model required in the analysis, *'regress( )'* is employed to speculate the slope and intercept of the regression based on both index returns and ticker returns. Detailed code is presented in figure 6 below. From the regression analysis, we conclude that the intercept β_0 approximates zero and the slope coefficient β_{SPY} equals to 1.07, indicating that for every one percent change in market return, Apple's return tends to change by 1.07%. Furthermore, the t-value is 11.57, which is dramatically higher than the critical values at any significance level, suggesting a statistically significant relationship between the market return and Apple's return within the model. In conclusion, S&P 500 index can be treated as a significant predictor of Apple's return.

Table 1. Parameter Values within the Regression

TABLE 1	
BETA_0	*0*
BETA_SPY	*1.07*
T-VALUE	*11.571*

```
#
def createModel():

    return LinearRegression()
#
def regress(model, X, y):

    X = X.values
    y = y.values

    model.fit(X,y)

    beta_0 = model.intercept_[0]
    beta_SPY = model.coef_[0][0]

    return beta_0, beta_SPY

tickerAttr = 'AAPL_Adj_Close'
X_train = train_data_ret.drop(columns = [tickerAttr])
y_train = train_data_ret[[tickerAttr]]

regr = createModel()

beta_0, beta_SPY = regress(regr, X_train, y_train)
t_value = calculate_t_value(regr, X_train, y_train, beta_SPY)

ticker = 'AAPL'
beta_0_rounded = round(beta_0, 2)
beta_SPY_rounded = round(beta_SPY, 2)

print(f"{ticker}: beta_0={beta_0_rounded}, beta_SPY={beta_SPY_rounded}, t-value={t_value:.3f}")
```

Figure 8. Calculations Within the Linear Regression

DOI: 10.54254/2755-2721/92/20241728

2.6. Evaluating the Feasibility of Our Model

However, some potential risks may be overlooked as the model has only one test set, therefore, we need to assess the model performance after acquiring results from the regression. Firstly, the method of Cross Validation is considered, which estimates the generalization ability of the predictive model and prevents the overfitting [2]. The initial dataset is divided into five groups, from Fold 1 to Fold 5. The model is trained on four out of five subsets and tested on the remaining one subset; different subset is used as the test set each time. For example, in the first iteration, the model is trained on Fold 1 to Fold 4 and tested on Fold 5, the procedure is ended after five times repeat. During this step, we can obtain five performance indicators, i.e. the accuracy of modelling for each iteration. Then, we can use these five indicators to compute an Average Cross Validation Score, which can be interpreted as R-square, indicating how well the model fits the observed database. From the model assessment, the average cross validation score for Apple is presented as a low value of 0.32, which is less persuasive. Nevertheless, in the field of finance, where human behaviors can influence a lot, a lower R^2 is still acceptable. Another reason for this lower value is due to the simplicity of the model, a linear regression with only one independent variable always fails to capture the complexity of the real-world data, as a result, an underfitting can be expected from our model.

```
from sklearn.model_selection import cross_val_score

cross_val_avg = 0 # average score of cross validation
k = 5             # 5-fold cross validation

def compute_cross_val_avg(model, X, y, k):
    '''
    Compute the average score of k-fold cross validation

    Parameters
    ----------
    model: An sklearn model

    X: DataFrame
    - Index returns

    y: DataFrame
    - Ticker returns

    k: Scalar number
    - k-fold cross validation

    Returns
    --------
    The average, across the k iterations, of the score
    '''
    # Perform cross-validation and return the mean score
    scores = cross_val_score(model, X, y, cv=k)
    return scores.mean()

# Assuming regr, X_train, y_train, and ticker are defined elsewhere in your code
cross_val_avg = compute_cross_val_avg(regr, X_train, y_train, 5)
print("{t:s}: Avg cross val score = {sc:3.2f}".format(t=ticker, sc=cross_val_avg))
```

Figure 9. Cross-Validation Evaluation

In addition, we can evaluate the Root of Mean Square Error (RMSE), which measures the difference between the observed value and expected value. RMSE is agreed to be an outstanding tool to test normally distributed errors [3]. We are looking forward to finding a low value of RMSE, as the lower the RMSE is, the smaller the gap between the prediction and actual value is.

$$RMSE = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}\ ,$$

where n is the number of observations, y_i is the observed value, and $\hat{y}_i$ is the expected value.

In our model, the in-sample and out-of-sample RMSE are 0.011 and 0.015 respectively, these two values are quite low, suggesting that the model captures the trend in the training set and predicts the future data evolution effectively.

DOI: 10.54254/2755-2721/92/20241728

```
from sklearn.metrics import mean_squared_error

rmse_in_sample = 0 # in sample loss
rmse_out_sample = 0 # out of sample performance

# Predicted  in-sample returns of AAPL using SPY index
aapl_predicted_in_sample = regr.predict(X_train)
# Predicted out-of-sample returns of AAPL using SPY index
aapl_predicted_out_sample = regr.predict(X_test)

def computeRMSE( target, predicted ):

    mse = mean_squared_error(target, predicted)
    rmse = np.sqrt(mse)
    return rmse

rmse_in_sample = computeRMSE(y_train, aapl_predicted_in_sample)
rmse_out_sample = computeRMSE(y_test, aapl_predicted_out_sample)

print("In Sample Root Mean squared error: {:.3f}".format( rmse_in_sample ) )
print("Out of Sample Root Mean squared error: {:.3f}".format( rmse_out_sample ) )
```

Figure 10. RMSE Evaluation

Table 2. Values of R-Square and RMSE

TABLE 2

AVG CROSS VAL SCORE	0.32
IN-SAMPLE RMSE	0.011
OUT-OF-SAMPLE RMSE	0.015

3. Hedging Strategies and Hedging Returns

Finally, we can hedge these two stocks based on the previously calculated returns. To assess whether Apple outperforms its expected performance, we can make a difference between the actual Apple return and the predicted Apple return which is represented by the market proxy.

$$r'_{AAPL} = r^t_{AAPL} - \beta_{SPY} * r^t_{SPY},$$

when $r'_{AAPL} > 0$, a gain can be expected from the portfolio.

Specifically, we can observe from figure 10, for example, on the first day, the portfolio return is 0.01, indicating that Apple outperformed the market index, when investors are confident about Apple's future price, a call option can be taken into consideration, allowing them to purchase Apple's stock in a lower strike price even though its price rises in the future. In contrast, the portfolio return is calculated to be negative on the second day. However, a loss does not necessarily explain an ineffective hedging strategy, as the primary objective of hedging is to mitigate risk rather than to generate profits.

DOI: 10.54254/2755-2721/92/20241728

```
hedged_series = pd.DataFrame()

def compute_hedged_series(model, X, y):

    # Predict the returns using the model
    predicted_returns = model.predict(X)

    # Compute the hedged returns (residuals)
    hedged_returns = y - predicted_returns

    return hedged_returns

hedged_series = compute_hedged_series(regr, X_test, y_test)

hedged_series = helper.renamePriceToRet(hedged_series)
hedged_series = hedged_series[start_dt:]
print(hedged_series[:])
```

```
            portfolio_ret
Dt
2018-10-03       0.010534
2018-10-04      -0.010290
2018-10-05      -0.011305
2018-10-08      -0.003367
2018-10-09       0.014362
...                   ...
2018-12-24       0.001273
2018-12-26       0.015457
2018-12-27      -0.015732
2018-12-28       0.000840
2018-12-31      -0.000731
```

Figure 11. Hedging Returns

4. Limitation and Further Research

Overall, the model demonstrates a great power of prediction, making it a reliable tool to analyze the hedging strategies within our context. However, some deficiencies still exist in the model, affecting the robustness of the prediction in the real-world applications, and consequently, interfering investors in their decision-making. Firstly, considering solely on the market index could lead to underfitting and biased estimation. Market index contains more comprehensive information from a wide range of society and industries, whereas some factors are insignificant and unrelated to Apple's operations [4]. Besides, our linear model with only one independent parameter would be difficult to capture more complicated dynamics in the financial market.

In the future research, we can enhance our model from two perspectives. It is crucial to incorporate additional relevant factors, such as the quality of product, chip-development, customer preference, etc. After this alteration, the model would be rather complex, so there is a further requirement for multi-faceted evaluation within the model [5]. Therefore, the method like Mean Absolute Error (MAE) and bootstrap can be considered. Moreover, as stock returns can be systematically influenced by the macroeconomic conditions, variables like the inflation, purchasing power and interest rate can be included into the regression [6]. Additionally, more advanced machine learning models, for example, the neural network and random forest could be applied to deal with the complex interactions among real-world parameters.

5. Conclusion

In this research, we have analyzed the correlation between market return and Apple's return. Our model suggests that market return serves as a good estimator of Apple's performance. However, even though evidence shows that the predictability of our model is quite convinced in identifying the relationship between stocks, the introduction of additional macroeconomic factors that directly relevant to the current financial market environment could yield much robust estimations [7]. Lastly, it is important for investors to be risk-averse and maintain a well-diversified portfolio, holding multiple when stocks simultaneously can help to alleviate risks, as the stability and growth in another technology stock can offset the slump in Apple's stock. In this context, having more specific knowledge about an investor's profile would allow for the development of more effective hedging strategy.

DOI: 10.54254/2755-2721/92/20241728

References

[1] Pagano, M.S. and Schwartz, R.A. (2003). A closing call's impact on market quality at Euronext Paris. Journal of Financial Economics, 68(3), pp.439–484. doi:https://doi.org/10.1016/s0304-405x(03)00073-4.

[2] Yoshua Bengio and Yves Grandvalet (2004). No Unbiased Estimator of the Variance of K-Fold Cross-Validation. *Journal of Machine Learning Research*, 5, pp.1089–1105.

[3] Bates, S.S., Hastie, T. and Tibshirani, R. (2021). Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association*, pp.1–22. doi:https://doi.org/10.1080/01621459.2023.2197686.

[4] COCHRANE, J.H. (2011). Presidential Address: Discount Rates. *The Journal of Finance*, [online] 66(4), pp.1047–1108. doi:https://doi.org/10.1111/j.1540-6261.2011.01671.x.

[5] Chai, T. and Draxler, R.R. (2014). Root mean square error (RMSE) or mean absolute error (MA E)? *Geoscientific Model Development Discussions*, 7(1), pp.1525–1534. doi:https://doi.org/10.5194/gmdd-7-1525-2014.

[6] Fama, E.F. and French, K.R. (1988). Dividend yields and expected stock returns. *Journal of Financial Economics*, 22(1), pp.3–25. doi:https://doi.org/10.1016/0304-405x(88)90020-7.

[7] Engle, R.F., Ghysels, E. and Sohn, B. (2013). Stock Market Volatility and Macroeconomic Fund amentals. *Review of Economics and Statistics*, 95(3), pp.776–797. doi:https://doi.org/10.1162/rest_a_00300.