Quantitative Researcher Test

Ali Qadar

November 2019

1 Introduction

This supervised machine learning project aims to forecast future close returns using three years $(2015 - 2018)$ of historical data sourced from the Australian Stock Exchange (ASX).

1.1 Outline

In this quantitative researcher test, I will start with data processing where I will organize raw data files in independent dataframes corresponding to open-high-close-low (ohlc). The organized dataframes will then be transformed in feature sets which include close returns, high and low ratio, volatility, cumulative returns, and volume. In a preliminary forecasting analysis, I will predict time-series trends using close returns and volatility using linear regression.

To gain deeper insights of the financial time-series data and to determine patterns and statistical associations, will create various visualizations. These visualizations will help me understand the distribution of the data and abnormal trends in each time-series which can cause performance issues in later stages.

I will train regression based machine learning algorithms on data to best capture future trends. Prediction results will then be evaluated using mean-squared error and accuracy while comparing with regression algorithms. In a preliminary analysis, I have compared linear regression with multi output regressor with random forests, multi output regressor meta-estimator, and gradient boosting regressor. Additionally, a univariate analysis [3] is taken into account where feature sets such as close returns, high and low ratio, volume, and cumulative returns [4] corresponding to a single ticker code were stacked in a dataframe $\mathbf{X}$ of size $N \times p$ with p number of feature variables

and N times points. The response variables $\mathbf{y}$ of size $N \times 1$ was created from close returns for each ticker code.

1.2 Data description

A summary of end-of-day (EOD) data description is provided below:

- Raw data was in *.txt* files, each file had comma separated value followed the following headers:
  - Ticker - the three-digit unique identifier ASX ticker code
  - Date - date of trade information
  - Open - price per individual share at the beginning of the day's trade
  - High - highest price recorded per individual share during the day's trade
  - Low - lowest price recorded per individual share during the day's trade
  - Close - price per individual share at the end of the day's trade
  - Volume - number of shares traded during the day
- The provided data had 883 time-points from dates Jan 02 2015 to Jun 29 2018
- When sorted by date, data has a shape (883, 13865) including all the feature [Open, High, Close, Low, Volume]
- After basic data cleaning we get size, Open: $(883, 2754)$, High: $(883, 2755)$, Close: $(883, 2755)$, Low: $(883, 2754)$, and Volume: $(883, 2739)$.

2 Tasks

2.1 Data processing

2.1.1 Data conversion

In this report, I will be using Pandas library in Python and will be working with lists, dictionaries, and dataframes. To begin with, all data files were imported and placed in a large dataframe. After all data files were read, dataframe had the following format:
Ticker, Date, Open, High, Low, Close, Volume.

To convert the OHLC data in the required format, I used pivot function in Pandas to organize whole dataframe by setting index as Date and column as Ticker. At the same time, dates were converted to Pandas date format. After inspecting dataframes, I found there was a lot of data missing. To deal with missing values, I used linear interpolation in Pandas and filled all the gaps. Furthermore, there were many columns with constant values, these columns were excluded from the dataframe. As an example of what data look like, I have shown close price and volume for ticker code ABC in Fig.1.

2.1.2 Calculation of future close returns and close returns

Based on the provided formula, I calculated two different dataframe containing future close returns and 1 day close returns. I used shift method in Pandas to create forward (P^c_{t+1}) and backward (P^c_{t-1}) shift in time for each ticker code, where P^c_t is close price at time t. For sanity check, I compared close returns calculated by hand with Pandas built-in function pct change and found them consistent the one I calculated. Due to shift in time, the first row of each dataframe has been set NaN, therefore, I used Pandas fillna and set NaN to zero.

2.1.3 Calculating ratio of high and low

As required, I have calculated the high and low ratio (i.e., High/Low) in the same format for each ticker code each day.

2.2 Calculating cumulative daily returns

In addition to close returns, cumulative daily returns is a useful quantity in determining the value of an investment at regular intervals. Cumulative returns can be calculated by multiplying (1+ close returns) with the cumulative product of all previous values. It can be written as:

$$d_i = (1 + r_t).d_{t-1}, d_0 = 1 \tag{1}$$

In Pandas, I used .cumprod() function to calculate cumulative returns. Fig. 2 show cumulative returns for four different ticker code. It can be seen for example in AAC $0.8 invested in 2016-01 would be worth $1.4 in 2016-08.

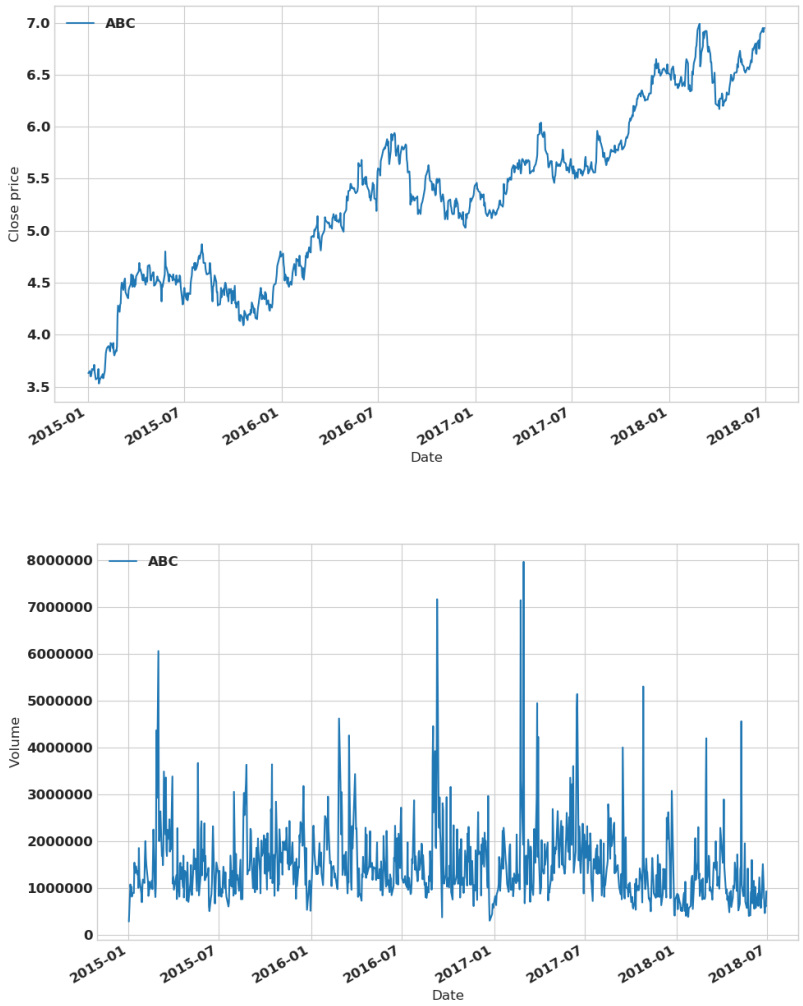


Figure 1: Close price and volume for ticker code ABC

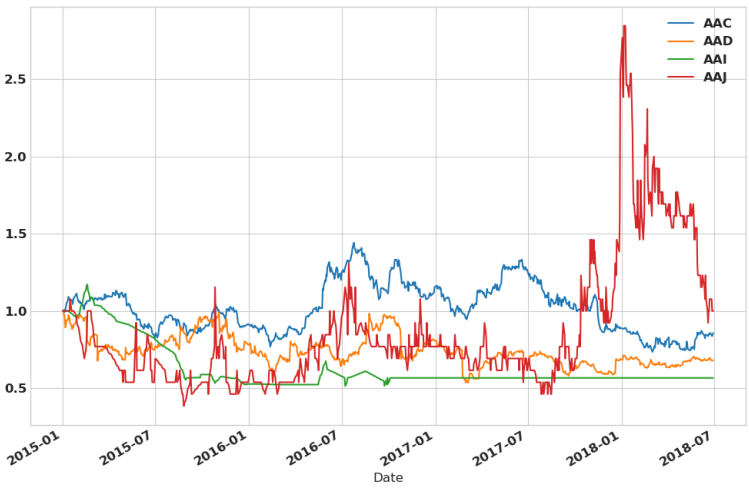


Figure 2: Cumulative returns for multiple ticker codes.

3 Exploratory Data Analysis

In this section, I will explore data to gain an understanding of the underlying structure of the data. I will use various visualizations for this analysis. For example, a histogram for data distribution, qqplot, correlation and scatter matrices, and probability plot. To begin with, I will check for any missing values in OHLC dataframes. Upon inspection I found that there were no missing values, this is due to interpolation that was performed at the first step after creating OHLC timeseries. I used histogram to examine data distribution, which is shown in Fig.3. According to Fig.3, data for ticker code ABC is symmetrical and normally distributed, most of the daily changes center around 0.0 with a small amount of skew to the right.
A statistical summary of the data was found using describe() method, which shows the mean of 0.000817 and a standard deviation of 0.012740 and 75 percent of the points fall below 0.007. The percentiles give us 95% confidence interval which means over the last three years (2015-2018) 95% of the days stock will fall in the range -0.02 to 0.02. Fig. 4 shows a comparison of return distribution for a number of stocks.
Correlation measures the strength and direction linear relationship between

two variables. In time-series analysis, the correlation matrix is a useful measure to visual how multiple stocks correlate positively or negatively. For example, in Fig.5 (left) tickers from 2500 to onward exhibit high values, this can be seen in Fig. 5 (right).

A common issue in timeseries analysis is the presence of outliers. Mostly mean and the standard is used to remove or replace outliers. The problem with using mean and standard deviation is that they are sensitive to outliers, other metrics such as median and inter-quartile are more robust. Almost all methods in robust statistic domain do not remove outliers completely but they replace it with a robust metric. Based on these arguments, I decided to use the median and applied it on my dataframe to check the difference in timeseries distribution. Fig. 6 shows an example of the timeseries where the outlier was replaced with the median. However, we can see that after removing outliers values of datapoints has been decreased which can affect our analysis especially in cases where covariance calculation is required. Therefore, I decided not to replace outliers.

To compare the daily change in close returns between stocks, I used a scatter matrix from Pandas to observe combinations of stocks shown in Fig. 7. It can be seen in scatter matrix there is very little correlation between selected stocks.

To further gain insight into the structure of timeseries, I used the moving average window to smooth out short-term fluctuations and to highlight long-term trends. I calculated a rolling window with its period set to 5 for ABC in year 2017 shown in Fig.8. I also calculated the mean average absolute deviation which demonstrates how far on average sample values from overall mean shown in Fig. 8.

It is also useful to check stock volatility to assess the change in variance over a specific period. Generally, high volatility implies riskier the investment [4]. For example, Fig. 9 shows higher volatility (i.e., riskier investment) for AAJ compared with AAC which is most stable among all selected stocks.

Lastly, I calculate the rolling correlation of close returns to demonstrate how correlation change over time. Fig.10(a) show rolling correlation between ticker code ABC and ABP. One issue here is simply calculating does not capture changes in the volatility of two investments. For I carried out least squares regression on two stocks ABC and ABP. The resultant coefficient vector show in Fig.10(b) shows changes in volatility between stocks.

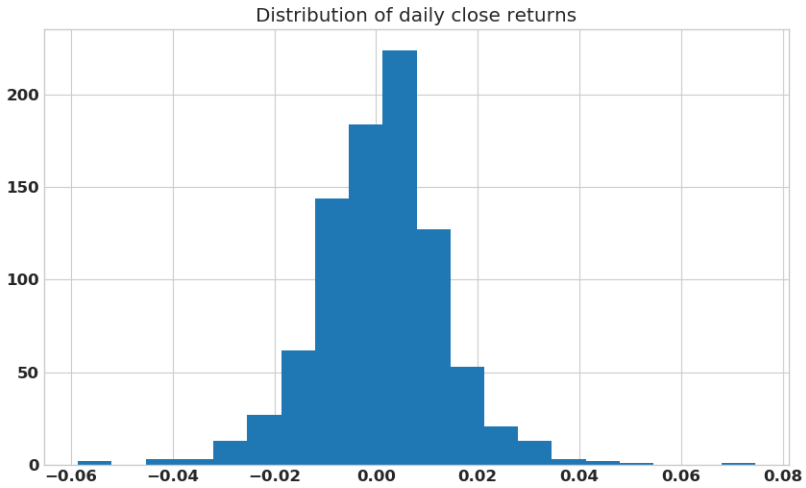


Figure 3: Distribution of daily close returns for ticker ABC.

4 Forecasting

To implement forecasting in this situation where I have multiple response variables, I will use multiple linear regression and other regressors available in sklearn library to predict multiple correlated variables. In will carry two different experiments, in the first one I will predict 30-day historical volatility of close returns. The first step is to prepare target variables. The first in data preparation is to concatenate multiple variables in a single dataframe object. Next step is to create feature variables, for that, I used the following formula to create five feature variables:

$$\frac{v_{t-1}}{v_{t-k}} \times v_{t-1}, \qquad k = 1, \cdots, 5 \tag{2}$$

where v_t is the volatility at time t. We use yesterday's volatility adjusted by the ratio of its recent values [1]. These newly created features are combined with data columns and some missing values were dropped by setting a threshold using .dropna() and others were filled using interpolation. The reason for setting threshold is that with a lot of missing values, timepoints reduce largely and therefore performance degrades. Target variables $\mathbf{y}$ and $\mathbf{X}$ are then created from the concatenated data. Before proceeding to model building, I scaled data in range 0 and 1 using MinMaxScaler to avoid the

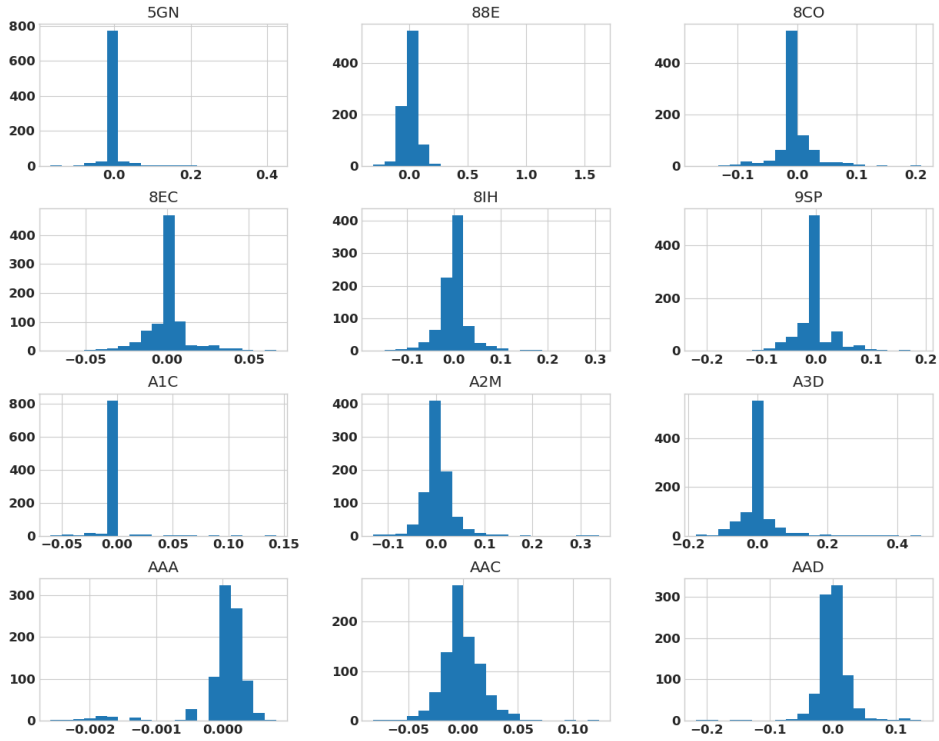


Figure 4: Data distribution of multiple ticker codes.

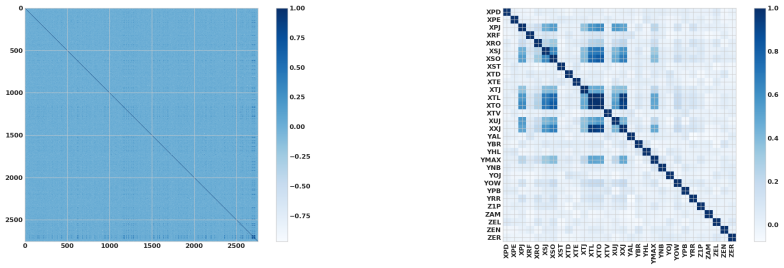


Figure 5: Correlation matrix of all close returns (left) and a zoomed version of the most correlated close returns (right).

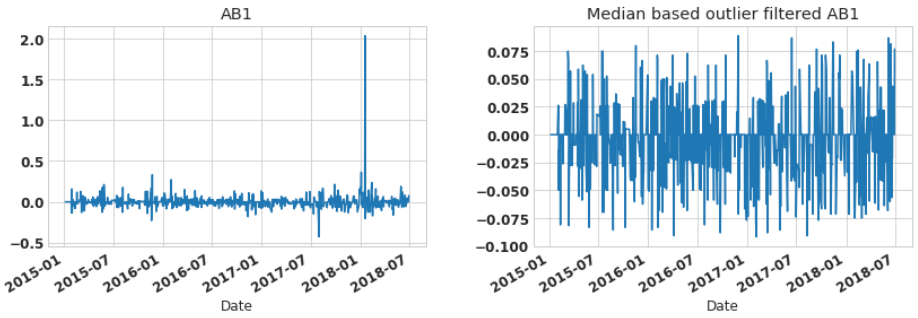


Figure 6: illustration of median based outlier removal.

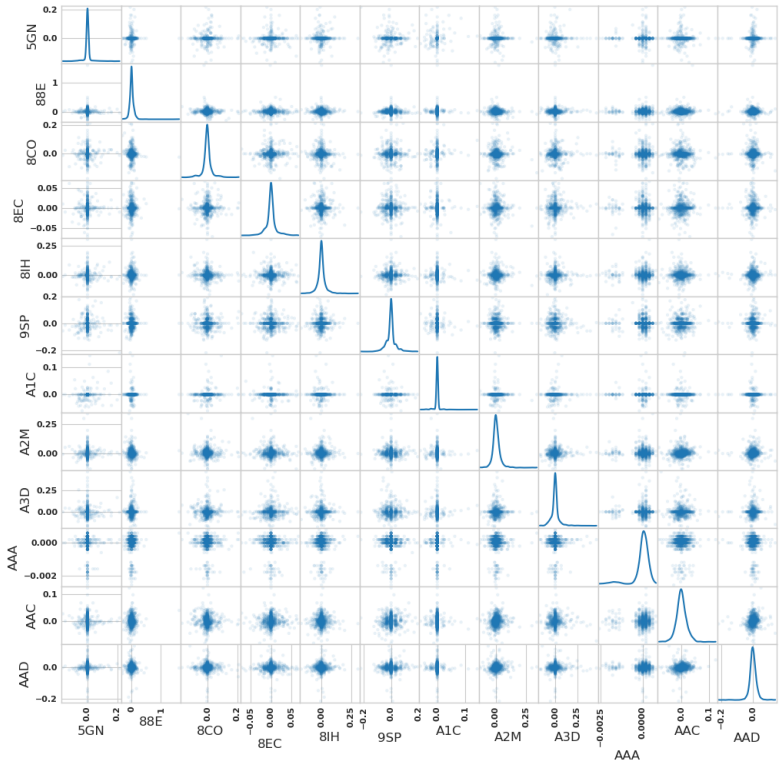
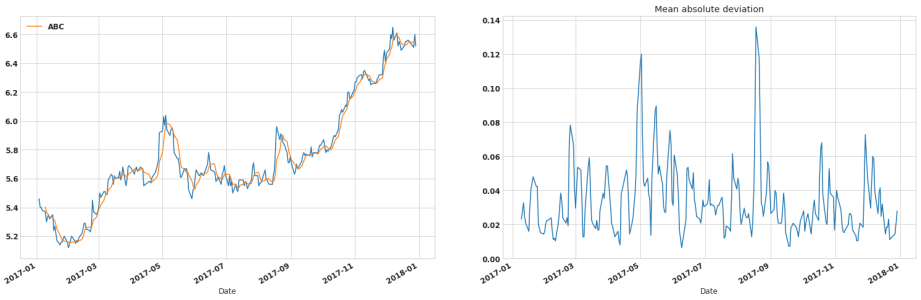


Figure 7: Shows comparison of daily close returns between stocks.



(a) (b)

Figure 8: a) Moving average b) mean absolute deviation

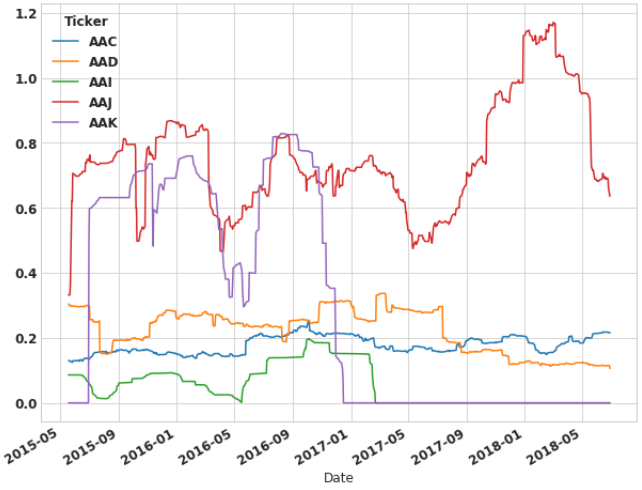
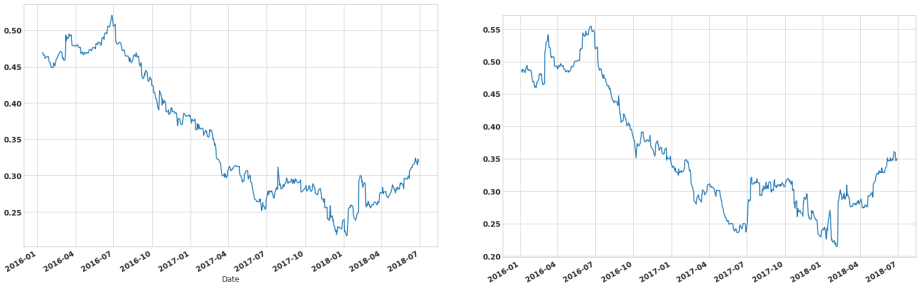


Figure 9: Shows volatility of the stocks.



(a) (b)

Figure 10: a) Rolling correlation b) Least square regression of close returns.

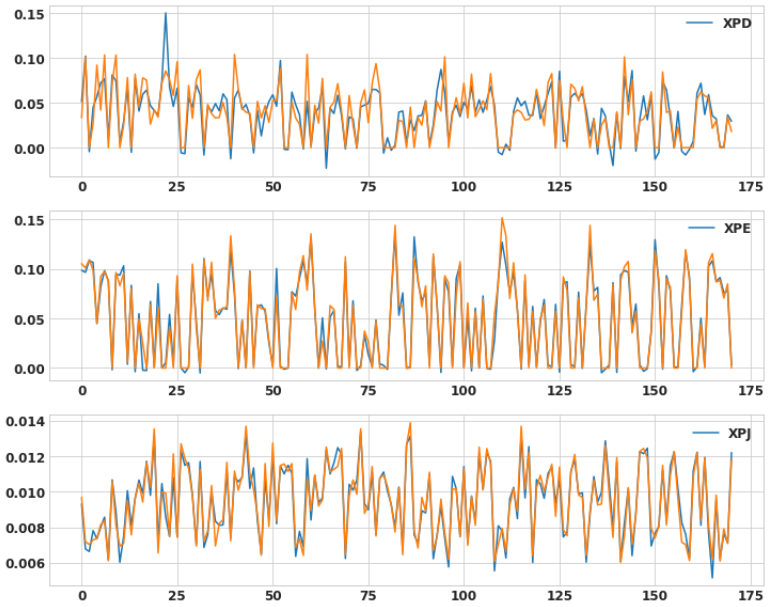


Figure 11: Prediction result using linear regression model.

heavy computational cost and to standardize influential variables that may affect overall performance. Data is then split in training and testing sets having 80 and 20 percent of the data, respectively.

After this step, I proceed to the model specification. I used to LinearRegression to train my training sets and calculated the score on test sets. I achieved a score of .8639 with the total mean squared error of 0.0021 regression model. The output is shown in Fig. 11. To compare these results with other regressors, I used MultiOutputRegressor with RandomForest [3] and MultiOutputRegressor. MultiOutputRegressor maps each target per regressor without exploiting the correlation between targets. Random forest is used as it natively supports MultiOuput regression. Random forest is useful in this due to its predictive power and control of over-fitting by training on sub-samples of the dataset. To assess overfitting while training on these models, I used a cross-validation score on training sets. The result shows a score of 0.9449 using MultiOutputRegressor with RandomForest and 0.9366 using RandomForest. I included an additional model GradientBoostingRegressor (GBR) [2] to compare the above results and found an accuracy score of 0.9686 which is slightly better compared with the above two models. I then calculated mean square error (MSE) for each algorithm and identi-

fied ticker code with minimum MSE, evidently, GBR outperformed others achieving lowest MSE and highest prediction accuracy.
In my second experiment, I will forecast future returns. Here, I created a feature set using the following features:
Close, High/Low, Close returns, Volume, Cumulative returns.
The response variable contains close returns. In this approach, I recursively read each ticker code and forecasted returns for a few future dates. I replaced rows with futures dates to NaN in close returns and appended future returns in forecasted columns. I then compared accuracy and MSE against several algorithms involving linear regression, K-nearest Neighbor [3], RandomForest, Multilayer Perceptron, and GradientBoostingRegressors.
Results show that linear regression has achieved minimum MSE of 0.002 and while KNN with MSE of 0.003 is the second best. Result of linear regression and KNN are shown in Fig. 12.

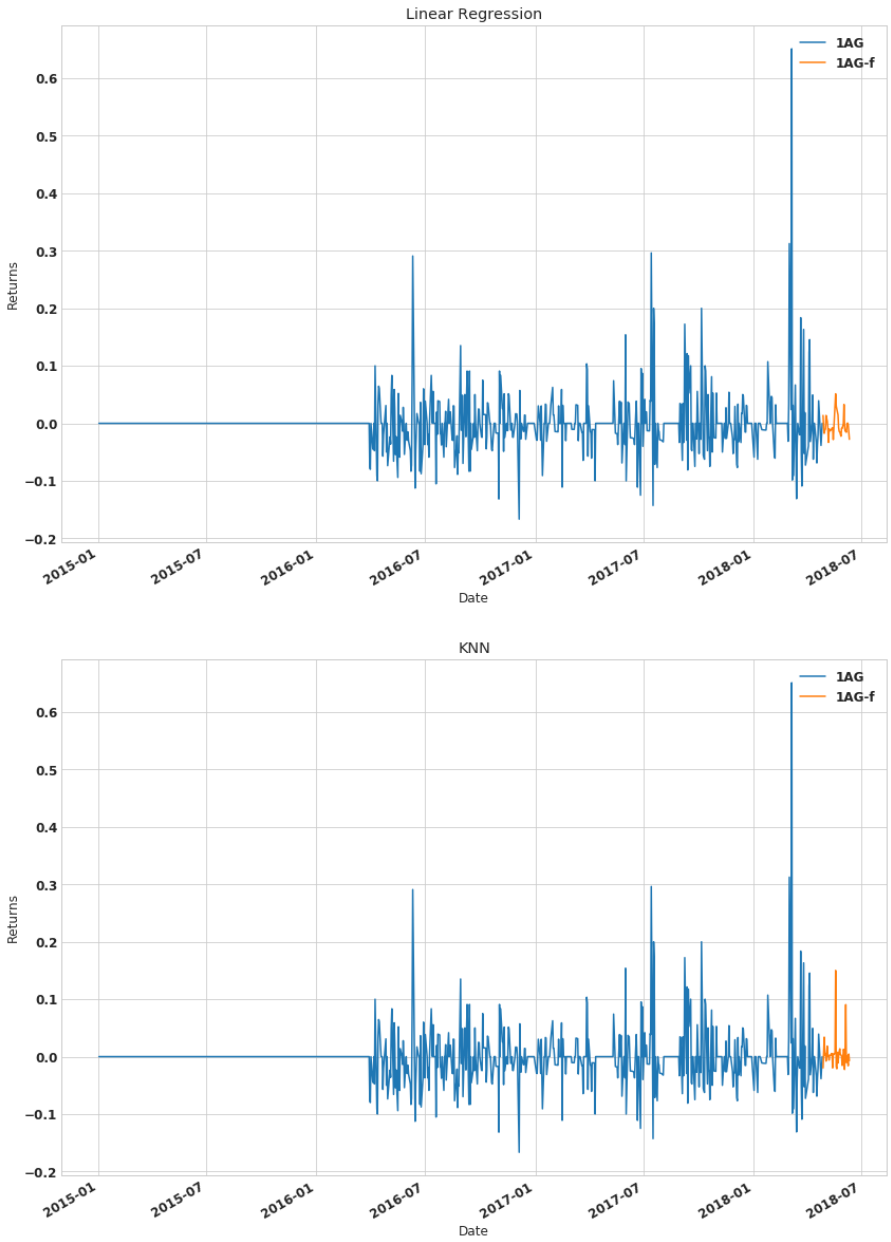


Figure 12: Shows forecast of future close returns using linear regression (left) and KNN (right).

References

[1] Nigel Da Costa Lewis. *Deep time series forecasting with Python : an intuitive introduction to deep learning for applied time series modeling.* CreateSpace Independent Publishing Platform, 2016.

[2] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367 – 378, 2002. Nonlinear Methods and Data Mining.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[4] Heydt Michael. *Mastering Python for Finance.* Packt Publishing, 2015.