Imperial College Business School

Forecasting Risk Premia of European Equities: A Neural Networks Approach

Quantitative Analysis Report

Gulshinder Gaddu

CID: 01448310

MSc Investment and Wealth Management

2018-2019 Academic Year

Client Specification

Our client, the United Kingdom based asset management firm Euclidean Capital is seeking to implement AI funds in its equities division after several consecutive quarters of underperformance. Investors have indicated disappointment at both performance and a lack of innovation and are considering withdrawing from the firm's equity funds. However, management industry and believes that expanding into the realm of machine learning and AI will assuage investors on both fronts. Management also recognises that an AI fund may be able to generate higher alpha by sourcing more investment opportunities than a traditional research analyst and by reducing human interference (and potentially overhead).

Having recognised the potential of AI to revolutionise the investment management industry, our client now aspires to become a global leader in the field of machine learning and AI funds. To this end our client has studied the noted working paper "Empirical Asset Pricing via Machine Learning" by Shihao Gu, Bryan Kelly and Dacheng Xiu (2019) and was impressed by the predictive power of artificial neural networks in an academic setting. Our client wishes to gradually implement AI funds across several equity markets using the neural network architectures and a similar methodology to that described in the paper.

Our role as Euclidean Capital's consultant is to evaluate and determine the feasibility of this project by analysing the performance of artificial neural networks in forecasting risk premia in the European equities market. To achieve we will use the neural network architectures described in the paper to forecast the risk premia of the Euro Stoxx 50's constituents individually in Python. We will then benchmark our neural networks' performance against the Huber Regression used in the paper as well as the XGBoost algorithm and analyse and compare the standard performance metrics of each. Finally, we are to provide a recommendation based upon our findings as to how the firm's first European AI equity fund should be implemented.

Note: Our client wishes to be able to replicate our methods and currently lacks access to CRSP or Compustat. Our client also ascribes to the view of semi strong market efficiency. Therefore, they ask that that computational resources be deployed as efficiently as possible as competitors will likely be searching for the same opportunities.

Table of Contents

Note: Please note that the main body of text is 3,108 words long. This is partly due to the need to explain XGBoost and partly to properly explain the data gathering process.

Data and Methodology

2.1 Data

Our client primarily trades in large cap, liquid securities and its AI funds will do so as well. For this reason we worked with the Euro Stoxx 50 index. As we could only use data sources that our client has access to all of our data was sourced from Bloomberg (with the exception of the size factors that were sourced from the Fama French website).

We constructed our set of regression targets by taking the index's current constituents' prices from 30/06/2000 to 28/06/2019 on a weekly basis. If any prices were omitted over this time period for a stock then the stock in question was removed from the project (for a full list of stocks used please see Appendix A). Our training data began on 29/06/2001 but we collected data from 1 calendar year earlier so as to include Jagdeesh and Titman's (1993) 1 year momentum as a feature. Whilst Gu, Kelly and Xiu (p.24, 2019) worked on a monthly basis we found this resulted in insufficient data to train our models. We were unable to obtain training data from before 29/06/2001 as there was insufficient data on the book to market ratio (a component of the linear benchmark and an important feature).

We were then able to calculate the weekly excess returns by solving:

$$ER_{i,t+1} = \frac{P_{i,t+1}}{P_{i,t}} - R_f \qquad \textbf{(2.1)}$$

Where $ER_{i,t+1}$ is the weekly excess return of stock *i* at time *t+1*, $P_{i,t}$ is the price of stock *i* at time *t* and R_f is the risk free rate (which we took to be the 1 week Euribor rate, a common proxy for the risk free rate in academia and business).

Having obtained our targets we then sought to obtain appropriate predictive features. We first looked at the comprehensive set of stock specific predictive characteristics composed by Green, Hand and Zhang (2016) and attempted to recreate as many of these as possible. In some situations we were able to treat missing data by applying the cross sectional median as per Gu, Kelly and Xiu's (p.24, 2019) methodology However, in other situations too much data was missing to do so and the feature was dropped from the project. The relative strength index (RSI) was also included as a stock specific feature as we felt that a short term metric would be beneficial given the shorter forecasting horizon.

We then attempted to augment our predictive features with a set of features pertaining to the overall index based on Welch and Goyal's (2008) macroeconomic features. However, due to insufficient coverage on the Euro Stoxx 50, we were unable to obtain the features as defined by Welch and Goyal and resorted to raw data from Bloomberg as a substitute (for a full list both stock specific and index features please see Appendix A).

2.2 Methodology

We sought two create two different sets of features from our predictive variables. The first set consisted of the raw predictive features listed in Appendix A. However, we also wished to determine whether out of sample performance could be improved by accounting for interactions between stock specific and index characteristics.

To achieve this we used the same model as Gu, Kelly and Xiu (p.24, 2019), to engineer the set of features:

$$z_{i,t} = x_t \otimes c_{i,t} \tag{2.2}$$

Where $c_{i,t}$ is a vector of stock specific characteristics for stock *i* at time *t* and x_t is a vector of index specific characteristics at time *t*. The interaction terms $z_{i,t}$ were then added to the raw predictive features to create a second set of covariates.

Our methods of fitting models and out of sample testing closely followed Gu, Kelly and Xiu's methodlogy (p.25, 2019) though we believed that our models would benefit from additional training. We chose to maintain the first 15 years for training and validation purposes and the last 4 years for out of sample testing. We fitted and evaluated our models in sample via walk forward validation using an expanding window to respect the temporal order of the data. Our training set was initially composed of the first year's data upon which a model was fitted onto with the validation set being the next year's data. The training dataset was then expanded to include the next year with the validation set being rolled 1 year forward and the process was repeated. Whilst we could have refitted models on a shorter time frame this was deemed too computationally expensive.

Furthermore, we applied Scikit-learn's StandardScalar to our Huber Regression and Neural Networks to prevent large feature values from dominating, (XGBoost was not scaled as decision trees are invariant to feature scaling (Ramadorai (2019)). We maintained a 4 week lookback period across models in the event there was a delay in the release of any of the predictive features, this also served to prevent look ahead bias.

Huber Regression

We elected for a simple linear regression to act as our first benchmark and followed Gu, Kelly and Xiu's (pp.4-5,10-11, 2019) reasoning. In keeping with sound econometric practice we desired a parsimonious system and so opted for a linear regression of lagged 1 month momentum, size and book to market ratio onto excess returns influenced by the model proposed by Lwellen (p.6, 2015). Whilst this model appears to be simple, raising concerns about underfitting, Lwellen (p.16, 2015) demonstrated that it exhibited respectable out of sample performance.

Gu, Kelly and Xiu (p.11, 2019) note that the use of the Ordinary Least Squares Regression (OLS) to estimate parameters is unsuitable for financial data that often has a heavy tails distribution. In an OLS regression the regressors $x_{i,t}$ are fitted onto the regressand y_t by selecting parameter values α and β that minimise the convex loss function:

$$\sum_t (y_t - \hat{y}_t)^2 = \sum_t \left(y_t - \alpha_t - \sum_i \beta_{i,t}\, x_{i,t}\right)^2 = u_t^2 \qquad \textbf{(3.1)}$$

Where $\hat{y}_t$is the fitted value. They note that the convexity of the loss function combined with the presence of numerous outliers can result in said outliers having a disproportionate degree of influence on parameter estimates. Huber (p.75, 1964) showed that the most robust loss function can be represented by:

$$H(u_t, \xi) \qquad \textbf{(3.2)}$$

$$where\ H(ut\ , \xi) = u_t^2\ when\ |u_t| \leq \xi$$

$$and\ H(u, t\ , \xi) = 2|u_t|\xi - \xi^2\ otherwise$$

Where the parameter ξ determines at what threshold a linear loss function is used. We used this loss function in conjunction with our linear system.

The implementations of our 3 techniques are provided in Appendix D in Python (the class StocksXGB and class StocksNN functions specifically) whilst information on hyperparameters is provided in Appendix B.

Extreme Gradient Boosting

XGBoost (Extreme Gradient Boosting) is a supervised machine learning technique that has been shown to perform extremely well on tabular data in Kaggle competitions (Chen and Guestrin (p.785, 2016)). For this reason it was selected as our second benchmark even though Gu, Kelly and Xiu (2019) did not implement it in their study. We felt it best to provide a more comprehensive background on XGBoost for our client as unlike the other regression techniques used in this project, it was not covered in Gu, Kelly and Xiu's (2019) paper.

Chen and Guestrin (p.785, 2016) define XGBoost as "a scalable machine learning system for tree boosting" and note it applies gradient boosting. Cisty and Soldanova (p.387, 2018) describe gradient boosting as "a forward learning ensemble model" (a model with strong predictive power that is created when several weaker models are combined in a step wise manner). A model is initially fitted onto the data, its precision is determined and the successive model will assign more weight to instances where its predecessor was incorrect in training. This process is repeated until the final model is deemed sufficiently accurate.

The primary model that XGBoost uses is the decision tree ensemble, tree-based methods according to Hastie Tibrashirani and Friedman (p.305, 2016) "partition the feature space into a set of rectangles and then fit a simple model on each one". Mathematically, Chen and Guestrin (p.786, 2016) note that we can represent the prediction from the tree ensemble method by:

$$\hat{y}_i = \sum_{k=1}^{K} f_k\,(x_i),\ f_k \in \mathcal{F} \tag{4.1}$$

$$\text{where } \hat{y}_{i,t} = \hat{y}_{i,t-1} + f_t(x_i)$$

Where the XGBoost documentation website ("Introduction to Boosted Trees- Decision Tree Ensembles", 2019) defines: "K as the number of trees, $\mathcal{F}$ as the space of all possible trees and f as a function in functional space $\mathcal{F}$". $\hat{y}_{i,t}$ can be considered as the model at training during round t. As with all supervised learning problems our model is optimised through the minimisation of an objective function which can be expressed as the sum of a loss function and a regularisation function:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_{i,t}) + \sum_{i=1}^{t} \Omega(f_i) \tag{4.2}$$

Using equation 4.1 we can express equation 4.2 as:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, (\hat{y}_{i,t-1} + f_t(x_i)) + \sum_{i=1}^{t} \Omega(f_i) \tag{4.3}$$

However, XGBoost's regularisation function can be written as (Chen and Guestrin (p.786, 2016)):

$$\Omega(f) = \gamma T + ½\, \lambda \sum_{j=1}^{T} ||w||_j^2 \qquad \textbf{(4.4)}$$

Where the XGBoost documentation website defines "T as the number of leaves" and "w as a vector of scores on leaves" (("Introduction to Boosted Trees-Model Complexity" (2019)). Therefore the first term in equation 4.4 penalises the number of leaves on a tree and the second l_2 term penalises leaf scores. The regularisation term for the XGBoost algorithm's l_2 penalty term makes it more regularised than a gradient boosted tree which reduces the likelihood of XGBoost overfitting (Chen and Guestrin (p.786, 2016)).

To determine the optimal leaf weight for a generalised loss function, Chen and Guestrin (p.786, 2016) take the second order Taylor Expansion of the objective function with respect to the loss function:

$$\mathcal{L} = \sum_i [l(y_i, \hat{y}_{i,(t-1))} + g_i f_t(x_i) + ½\, h_i f_t^2(x_i)] + \Omega(f_t) \qquad \textbf{(4.5)}$$

Where g_i and h_i are the first and second derivatives of the loss function with respect to $\hat{y}_{i,(t-1)}$. By first redefining a tree $f(x)$ as:

$$f_t(x) = w_{q(x)}, w \in \mathbb{R}^T \qquad \textbf{(4.6)}$$

We can then define q as "the structure of each tree that maps an example to the corresponding leaf index" (Chen and Guestrin (p.786, 2016)). By further defining I_j as "the set of indices of data points assigned to the j-th leaf" and letting $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$, ("Introduction to Boosted Trees-Structure Score" (2019)). Chen and Guestrin (p.787, 2016) show that the optimal leaf weight can be expressed as:

$$w^*_j = -\frac{G_j}{(H_j + \lambda)} \qquad \textbf{(4.7)}$$

This can then be substituted into the objective function to provide an optimal value which can be used to evaluate tree structures:

$$\mathcal{L}^* = -½\, \frac{\sum_{j=1}^{T} G_j^2}{(\mathrm{Hj} + \lambda) + \gamma \mathrm{T}} \qquad \textbf{(4.8)}$$

In practice it is not often feasible to evaluate all potential structures and a greedy algorithm is used instead. This greedy algorithm begins with a single leaf and continues to add branches to the tree until the loss reduction resulting from a split is zero (Chen and Guestrin (p.787, 2016)). The loss reduction can be expressed as:

$$\mathcal{L}_{\text{split}} = ½\left[\frac{G_L^2}{(H_L + \lambda)} + \frac{G_R^2}{(H_R + \lambda)} - \frac{(G_L + G_R)^2}{(H_L + H_R + \lambda)}\right] - \gamma \qquad \textbf{(4.9)}$$

The right hand side of equation 4.9 can be seen as the sum of the scores of the new left and right leaves minus the sum of the scores of the original leaf plus the penalty term for an additional leaf.

Artificial Neural Networks

5.1 Theoretical Framework

Zhang and Gupta (p.1340, 2003) note that an artificial neural network is composed of "an input layer, an output layer and one or more hidden layers". Within each of these layers lie neurons that receive information from neurons in the previous layer and apply an activation function generating an output. The input layer is the layer that receives the raw predictor variables as an input, the output layer is the layer that yields the final, observed output and the layers between them are the hidden layers.

We can represent this mathematically by defining the set of raw features that are initially passed into the input layer as x_k where $k \in 1,...,n_1$ where n_1 is the number of neurons on the input layer. Each neuron on the first hidden layer will receive a signal that Zhang and Gupta (p.1341, 2003) represent by the general expression:

$$y_j^{(l)} = \sum_{i=1}^{n_{l-1}} x_i w_{ij}^{(l-1)} + b_j^{(l-1)} \quad \textbf{(5.1)}$$

Where $y_j^{(l)}$ is the signal received by the j^{th} neuron on the l^{th} layer (the input layer is represented by $l = 1$), x_i is the signal generated by the i^{th} neuron on the $l-1^{th}$ layer and w and b are weighing and bias parameters that are to be optimised.

Walther (2019) notes that equation 5.1 is linear but we can induce non-linearity by applying an appropriate activation function f that can differ between layers. The process described above is then repeated every time a signal is passed from one layer to the next until the output layer generates the predicted value.

5.2 Architectures and Training

Our artificial neural networks were implemented using the Keras library. We used the same architectures that Gu, Kelly and Xiu (p.20,-2019) used in their study: "a single hidden layer containing 32 neurons (NN1), two hidden layers with 32 and 16 neurons respectively (NN2), three hidden layers with 32, 16 and 8 neurons respectively (NN3), 4 hidden layers containing 32, 16, 8 and 4 neurons respectively (NN4) and 5 hidden layers containing 32, 16, 8, 4 and 2 neurons respectively (NN5)". We also selected the same activation function (for its nonlinearity and simplicity), rectified linear unit (ReLU) defined as:

$$f(x) = max\{0, x\} \quad \textbf{(5.2)}$$

The weighing and bias parameters were optimised through the Adaptive Moment Estimation (Adam) algorithm. Kingma and Ba (p.1, 2014) define Adam as "an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments". We deemed standard gradient descent (altering parameter values in the direction that reduces loss) computationally inefficient, as Walter (2019) notes. Whilst Goodfellow, Bengio and Courville (p.147, 2016) argue that using a minibatch can yield a sufficiently accurate parameter estimate at the cost of considerably fewer computational resources (stochastic gradient descent). Gu, Kelly and Xiu (p.21, 2019) argue that as the gradient converges to zero so must the learning rate due to the low signal to noise ratio. In stochastic gradient descent, the learning rate remains constant whilst in Adam the learning rate decays over time. Whilst there are other potential optimisers to choose from (AdaGrad, SGD Nesterov etc), we were encouraged to use Adam after Kingma and Ba (p.7, 2014) demonstrated that it outperformed them.

To reduce variance in our predictions, we adopted a similar approach to Gu, Kelly and Xiu (p.22, 2019) and used an ensemble method in which we used 2 different initial random seeds to generate forecasts for each neural network and took an average across seeds to provide a final forecast. Whilst variance could be further reduced by using more seeds this required too much computational power to be feasible.

To prevent overfitting we also adopted the same approach as Gu, Kelly and Xiu (pp.21-22, 2019), an l_1 penalty applied to the weight parameters combined with the early stopping algorithm (which we selected for its simplicity and efficiency). Goodfellow, Bengio and Courville (pp.239-240, 2016) note that training error will decrease over time but after a point this will result in overfitting, increasing validation error. Early stopping addresses this by updating the parameter estimates of a model whenever the validation loss decreases. If there is no improvement in performance on the validation set after a predefined number of epochs then the algorithm terminates and the parameters with the best performance on the validation set are returned to be used for out of sample testing.

Performance Metrics

To compare the out of sample performance of our models, we used 4 metrics that are commonly employed when dealing with forecasting excess returns in a regression context. They are the mean-squared error (MSE), the sum of squared errors (SSE) and the out of sample R squared (meaned $\bar{R}^2_{OOS}$ and demeaned R_{OOS}^2). These performance metrics can be expressed mathematically on an individual stock basis as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{6.1}$$

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{6.2}$$

$$\bar{R}^2_{OOS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{6.3}$$

$$R^2_{OOS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i)^2} \tag{6.4}$$

Where $\bar{y}$ denotes the mean of y.

Results and Analysis

We report the overall performance metrics of each model for both the baseline set of covariates and the set of features that incorporates interactions between covariates. We also report the performance metrics on an individual basis for each stock in Appendix C as well as the optimal learning rates for XGBoost and neural networks determined in training.

Baseline Covariates							
	Huber Regression	**XGBoost**	**NN1**	**NN2**	**NN3**	**NN4**	**NN5**
MSE	10.67	10.98	18.96	12.90	11.71	11.20	11.22
SSE	80257.7	82552.2	142552.8	97005.9	88064.8	84264.9	84352.1
$\bar{R}_{OOS}^2$	0.002	-0.027	-0.786	-0.215	-0.103	-0.056	-0.057
R_{OOS}^2	0.003	-0.026	-0.786	-0.215	-0.103	-0.056	-0.057

Interaction between Covariates Incorporated							
	Huber Regression	**XGBoost**	**NN1**	**NN2**	**NN3**	**NN4**	**NN5**
MSE	10.67	11.68	14.34	12.72	10.81	10.92	10.88
SSE	80257.7	87820.5	107873.3	95639.2	81312.0	82093.5	81826.9
$\bar{R}_{OOS}^2$	0.002	-0.092	-0.352	-0.198	-0.019	-0.029	-0.025
R_{OOS}^2	0.003	-0.091	-0.351	-0.198	-0.019	-0.028	-0.025

Figure 1: Summary of findings

From Figure 1, it is apparent that the Huber Regression outperformed both the XGBoost and all neural network architectures regardless of the set of covariates employed across all performance measures. We also note that when only the baseline set of covariates was used, XGBoost outperformed all of the neural network structures across all performance metrics. However, when interactions between stock and index specific terms were incorporated then neural networks outperformed XGBoost across all all metrics from architectures of 3 hidden layers and deeper. The inclusion of interaction terms caused the out of sample forecasting power of XGBoost to drop suggesting overfitting.

Furthermore we note that out of sample improves as our neural networks become deeper up to a point after which performance deteriorates for both the baseline set of covariates (performance peaks at NN4) and when interactions are included (performance peaks at NN3).

Despite this, only the Huber Regression demonstrates a positive out of sample R squared (meaned or demeaned) overall ($\bar{R}_{OOS}^2$ = 0.002 and R_{OOS}^2 = 0.003). Whilst our client may not find this impressive, Campbell and Thompson (p.12, 2007) have demonstrated that an out of sample R squared as small as that given by the Huber Regression can still yield significant returns for an investor. In addition, whilst the overall out of sample R squared values for our neural networks were negative, Appendix C shows that a minority of stocks did indeed demonstrate small, positive out of sample R squared values for some of the neural network structures (NN3-NN5) when interactions between covariates were included.

Whilst we were unable to replicate the out of sample performance that Gu, Kelly and Xiu (2019) reported, we attribute this to a lack of data and coverage on the European equities market relative to the American market. Further features could therefore improve out of sample performance. We also note that better results could be obtained through the application of a Principal Components Analysis (PCA). However, this would cause some challenges in interpreting which features are important in forecasting returns which our client might find relevant.

Conclusion

Based our findings, we recommend that our client implement the Huber Regression to forecast excess returns in the European equities market. This simple model has been shown to outperform XGBoost and the neural network structures implemented by Gu, Kelly and Xiu (2019) regardless of the set of covariates used. Furthermore, despite it's simplicity, Gu, Kelly and Xiu (p.5 2019) note that it contains 3 of the most important predictive characteristics in financial literature.

However, if our client still wishes to implement neural networks to obtain alpha then we strongly recommend that they obtain as many features as possible pertaining not only to the specific stock but to the overall index and even the global macroeconomy as we attempted and consider interactions between these features as well. We further recommend that only the NN3 architecture should be employed and only on the minority of stocks that we found to generate a posivite out of sample R squared. However, we provide the caveat that this performance may not be repeated in the future.

Bibliography

Academic Papers

Campbell J., Thompson S., (2007), *Predicting Excess Stock Returns Out of Sample: Can Anything Beat the Historical Average?*, The Review of Financial Studies, Volume 21, Issue 4, July 2008, Pages 1509–1531, [Available from: https://dash.harvard.edu/handle/1/2622619.] Accessed on 12th September 2019.

Chen, T., Guestrin, C., (2016) *XGBoost: a scalable tree boosting system*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. [Available from: https://dl.acm.org/citation.cfm?doid=2939672.2939785.]

Accessed on 6th August 2019.

Green, J., Hand, J. and Zhang, F. (2016), *The Characteristics that Provide Independent Information about Average U.S. Monthly Stock Returns*. The Review of Financial Studies Volume 30, pp. 4389–4436. [Available from http://dx.doi.org/10.2139/ssrn.2262374.] Accessed on 7th July 2019.

Gu, S., Kelly, B. and Xiu, D., *Empirical Asset Pricing via Machine Learning* (2019). Chicago Booth Research Paper No. 18-04; 31st Australasian Finance and Banking Conference 2018; Yale ICF. Working Paper No. 2018-09. [Available from: http://dx.doi.org/10.2139/ssrn.3159577.] Accessed on 5th July 2019.

Huber, P. J., (1964), *Robust estimation of a location parameter*, Annals of Mathematical Statistics Volume 35 Number 1, pp. 73–101. [Available from: doi:10.1214/aoms/1177703732.] Accessed on 21st July 2019.

Jegadeesh N., Titman S., (1993), *Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency*. The Journal of Finance (Volume 48, Issue 1), pp. 65-91. [Available from https://doi.org/10.1111/j.1540-6261.1993.tb04702.x.] Accessed on 11th July 2019.

Kingma D., Ba J., (2014), *Adam: A method for stochastic optimization*, The International Conference on Learning Representations (ICLR), San Diego. [Available from: https://arxiv.org/abs/1412.6980v9 .] Accessed on 22nd July 2019.

Lewellen, Jonathan, (2015), *The cross-section of expected stock returns*, Critical Finance Review: Vol. 4: No. 1, pp. 1-44. [Available from http://dx.doi.org/10.1561/104.00000024.] Accessed on 21st July 2019.

Narasimhan Jegadeesh, Sheridan Titman, (1993), *Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency*, The Journal of Finance Vol. 48, No. 1, pp. 65-91.

Welch I., Goyal A., (2008), *A Comprehensive Look at The Empirical Performance of Equity Premium Prediction*, The Review of Financial Studies, Volume 21, Issue 4 , pp. 1455–1508. [Available from https://doi.org/10.1093/rfs/hhm014.] Accessed on 10th July 2019.

Zhang, Q. J. & Gupta, K. C. (20003), *Artificial Neural Networks for RF and Microwave Design—From Theory to Practice*, IEEE Transactions on microwave theory and techniques, Volume 51 No, 4, pp. (1339-1350). [Available from: https://ieeexplore.ieee.org/document/1193152.] Accessed on 21st July 2019.

Books

Cisty and Soldanova, (2018), Flow Prediction Versus Flow Simulation Using Machine Learning Algorithms, In: Perner P., (ed), *Machine Learning and Data Mining in Pattern Recognition, 14th International Conference, MLDM 2018 New York, NY, USA, July 15–19, 2018 Proceedings, Part II*, Leipzig, Germany, Springer, p. 387.

De Prado M. L., (2018), *Advances in Financial Machine Learning*, United Kingdom, Wiley Publications, United Kingdom, pp. 162,163-166.

Goodfellow I., Bengio Y. and Courville A., (2017), *Deep Learning*, Adaptive Computation and Machine Learning Series, Cambridge, Massachusetts, MIT Press pp. 147 & 239-240.

Hastie T., Tibshirani R., and Friedman J., (2016), *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer Series in Statistics, 2nd Edition, New York, United States of America, Springer, p. 305.

Lectures

Ramadorai T. (2019), *Lecture 1 – Introduction to Machine Learning*, [Lecture], Big Data in Finance 1, Imperial College Business School, 17th January 2019.

Ramachandram G. (2019), *Lecture 5 – Machine Learning in Finance*, [Lecture], Applied Trading Strategies, Imperial College Business School, 16th May 2019.

Walther A. (2019), *Lecture 4 – Deep Learning*, [Lecture], Big Data in Finance 2, Imperial College Business School, 17th May 2019.

Websites

Brownlee J., (2018), *How to Use the TimeseriesGenerator for Time Series Forecasting in Keras*, Available from: https://machinelearningmastery.com/how-to-use-the-timeseriesgenerator-for-time-series-forecasting-in-keras/ . [Accessed on 18th July 2019].

Kenneth R. French (n.d.), *Current Research Returns*. Available from: https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html. [Accessed on 15th July 2019].

Kolanovic and Krishnamacha (2017), *Big Data and AI Strategies Machine Learning and Alternative Data Approach to Investing*, Available from: https://faculty.sites.uci.edu/pjorion/files/2018/05/JPM-2017MachineLearningInvestments.pdf. [Accessed on 20th July 2019]. p.83. (Corporate research undertaken for JP Morgan).

https://machinelearningmastery.com/how-to-use-the-timeseriesgenerator-for-time-series-forecasting-in-keras/

Scikit-learn (2019), Huber Regression, In: Generalized Linear Models. Available from: https://scikit-learn.org/stable/modules/linear_model.html#huber-regression. [Accessed on 5th August 2019].

https://scikit-learn.org/stable/modules/linear_model.html#huber-regression

XGBoost documentation website (2019), Decision Tree Ensembles, In: *Introduction to Boosted Trees*. Available from: https://xgboost.readthedocs.io/en/latest/tutorials/model.html. [Accessed on 17th August 2019].

XGBoost documentation website (2019), Model Complexity, In: *Introduction to Boosted Trees*. Available from: https://xgboost.readthedocs.io/en/latest/tutorials/model.html. [Accessed on 17th August 2019].

XGBoost documentation website (2019), Structure Scores, In: *Introduction to Boosted Trees*. Available from: https://xgboost.readthedocs.io/en/latest/tutorials/model.html. [Accessed on 17th August 2019].

Appendix A: List of Stocks and Features used

We provide a complete list of stocks and features used in our study. We first provide the list of stocks, the relevant Bloomberg Tickers and the industries to which they belong:

Name	Bloomberg Ticker	Industry
Ahold Delhaize N.V.	AD NA Equity	Retail
Adidas A.G.	ADS GY Equity	Sportswear
Air Liquide S.A.	AI FP Equity	Chemical, Healthcare and Engineering
Allianz S.E.	ALV GY Equity	Financial
ASML Holding N.V.	ASML NA Equity	Technology
BASF S.E.	BAS GY Equity	Chemical
Bayer A.G.	BAYN GY Equity	Healthcare and Pharmaceuticals
Banco Bilbao S.A.	BBVA SQ Equity	Financial
BMW A.G.	BMW GY Equity	Automotive
Groupe Danone S.A.	BN FP Equity	Food and Beverages
BNP Paribas S.A.	BNP FP Equity	Financial
CRH Plc	CRH ID Equity	Construction
AXA	CS FP Equity	Financial
Daimler A.G.	DAI GY Equity	Automotive
Vinci S.A.	DG FP Equity	Construction
Deutsche Telekom A.G.	DTE GY Equity	Telecommunications
EssilorLuxottica S.A.	EL FP Equity	Pharmaceuticals
Enel S.p.A.	ENEL IM Equity	Utilities
Eni S.p.A.	ENI IM Equity	Oil and Gas
Total S.A.	FP FP Equity	Oil and Gas
Fresenius S.E.	FRE GY Equity	Healthcare
Société Générale S.A.	GLE FP Equity	Financial
Iberdrola S.A.	IBE SQ Equity	Utilities
ING Group N.V.	INGA NA Equity	Financial
Intesa Sanpaolo S.p.A.	ISP IM Equity	Financial
Kering S.A.	KER FP Equity	Luxury Goods
Louis Vuitton S.E.	MC FP Equity	Luxury Goods
Munich Re Group	MUV2 GY Equity	Financial
Nokia Corporation	NOKIA FH Equity	Telecommunications
L'Oreal S.A.	OR FP Equity	Cosmetics
Orange S.A.	ORA FP Equity	Telecommunications
Philips N.V.	PHIA NA Equity	Conglomorate
Safran S.A.	SAF FP Equity	Aerospace

Sanofi S.A.	SAN FP Equity	Pharmaceuticals
Banco Santander S.A.	SAN SQ Equity	Financial
SAP S.E.	SAP GY Equity	Technology
Siemens A.G.	SIE GY Equity	Conglomorate
Schneider Electric S.E.	SU FP Equity	Electronics
Telefónica S.A.	TEF SQ Equity	Telecommunications
Unilever	UNA NA Equity	Consumer Goods
Vivendi S.A.	VIV FP Equity	Media
Volkswagen A.G.	VOW3 GY Equity	Automotive

Figure A.1: List of stocks used in our study.

We then provide a list of stock specific features used in our project. 21 features were updated on a weekly basis with the remaining features being updated on a yearly basis. To account for nontrading days, we resampled our data on a weekly/yearly basis when appropriate. We took the view that as many features as possible should be employed even if there is a high degree of correlation between some of them. Due to the number of stocks and features, it was not feasible to construct a correlation matrix and remove highly correlated features for every stock, instead we depended on regularisation techniques to eliminate redundant features.

Whilst we looked at Green, Hand and Zhang's (2016) paper we also looked at the original author's work to understand how each feature was calculated and attempted to replicate it as closely as possible. We sought to provide as many features that acted to proxies for momentum, liquidity and volatility a possible as as Gu, Kelly and Xiu (p.30, 2019) found these to be the most salient in their study. We note the weekly features in figure A.2 below, how they were obtained, the relevant Bloomberg keys and the original influence.

Feature Name	Obtained	Bloomberg Keys	Influenced by
3 Day RSI	Directly from Bloomberg	RSI_3D	Author's addition
9 Day RSI	Directly from Bloomberg	RSI_9D	Author's addition
14 Day RSI	Directly from Bloomberg	RSI_14D	Author's addition
30 Day RSI	Directly from Bloomberg	RSI_30D	Author's addition
10 Day Volatility	Directly from Bloomberg	VOLATILITY_10D	Return Volatility (Ang et al. 2010)
20 Day Volatility	Directly from Bloomberg	VOLATILITY_20D	Return Volatility (Ang et al. 2010)
30 Day Volatility	Directly from Bloomberg	VOLATILITY_30D	Return Volatility (Ang et al. 2010)
60 Day Volatility	Directly from Bloomberg	VOLATILITY_60D	Return Volatility (Ang et al. 2010)
90 Day Volatility	Directly from Bloomberg	VOLATILITY_90D	Return Volatility (Ang et al. 2010)

1 Month Momentum	Price at t divided by Price at t-1	PX_LAST	1 Month Momentum (Jagdeesh and Titman 1993)
6 Month Momentum	Price at t divided by Price at t-1	PX_LAST	6 Month Momentum (Jagdeesh and Titman 1993)
12 Month Momentum	Price at t divided by Price at t-1	PX_LAST	12 Month Momentum (Jagdeesh 1990)
Ask Price	Directly from Bloomberg	PX_ASK	Bid-Ask Spread (Amihud & Mendelson 1989)
Bid Ask Spread	Bid Price - Ask Price	PX_BID, PX_ASK	Bid-Ask Spread (Amihud & Mendelson 1989)
Bid Price	Directly from Bloomberg	PX_BID	Bid-Ask Spread (Amihud & Mendelson 1989)
Book to Market Ratio	Inverse Market to Book Ratio	MARKET_CAPITALIZATION_TO_BV	Book to Market (Rosenberg et al. 1985)
Raw Beta	Directly from Bloomberg	BETA_RAW_OVERRIDABLE	Beta (Fama and Macbeth 1973)
Raw Beta squared	Squaring Raw Beta	BETA_RAW_OVERRIDABLE	Beta Squared (Fama and Macbeth 1973)
Trading Volume	Directly from Bloomberg	PX_VOLUME	Share Turnover (Datar et al. 1998)
Turnover	Directly from Bloomberg	TURNOVER	Share Turnover (Datar et al. 1998)

Figure A.2: List of features updated on a weekly basis used in our study.

Our 25 yearly features were updated at the end of each calender year and forward filled for the next year. Figure A.3 provides a comprehensive list of all yearly features used.

Feature Name	Obtained	Bloomberg Keys	Influenced by
% Change Sales to Inventories	% Change Sales to Inventories over 1 year	SALES_TO_INVENT	% Change in Sales to Inventory (Ou and Penman 1989)
Annual Equity Growth	Diference between total common equity over 1 year	TOT_COMMON_EQY	Growth in common shareholder equity (Richarson et al 2005)
Capital Expenditures and Inventories	Sum of Capital Expenditures and Inventories	CAPITAL_EXPEND, BS_INVENTORIES	Capital expenditures and inventories (Chen and Zhang 2010)
Cash Flow to Price	Inverse of Price to Cash Flow	PX_TO_CASH_FLOW	Cash Flow to Price (Desai et al 2004)
Cash Productivity (FCF)	Free Cash Flow/Cash and cash holdings	CF_FREE_CASH_FLOW, BS_CASH_NEAR_CASH_ITEM	Cash Productivity (Chandrashekar and Roa 2009)
Change in Shares Outstanding	Difference between shares outstanding over 1 year period	BS_SH_OUT	Chante in shares outstanding (Pontiff and Woodgate 2008)
Current Ratio	Directly from	CUR_RATIO	Current Ratio (Ou and Penman

	Bloomberg		1989)
Dividend Yield	Directly from Bloomberg	DIVIDEND_INDICATED_YIELD	Dividend to price (Litzenberger and Ramaswamy 1982)
Employee Growth	Directly from Bloomberg	EMPL_GROWTH	Employee Growth Rate (Bazdresch and Lin 2014)
Financial Leverage	Directly from Bloomberg	FNCL_LVRG	Leverage (Bhandari 1988)
Free Cash Flow to Debt	Directly from Bloomberg	FCF_TO_TOTAL_DEBT	Cash Flow to Debt (Ou and Penman 1989)
Gross Profitability	Gross Profit/Total Assets	GROSS_PROFIT, BS_TOT_ASSET	Gross Profitability (Novy-Marx 2013)
Growth in Capex	Directly from Bloomberg	TOT_CAP_EXPEND_GROWTH	Growth in capital expenditures (Anderson and Garcia-Feijoo 2006)
Price to Earnings Ratio	Directly from Bloomberg	PE_RATIO	Earnings to Price (Basu 1977)
Quick Ratio	Directly from Bloomberg	QUICK_RATIO	Quick Ratio (Ou and Penman 1989)
Return on Assets	Directly from Bloomberg	RETURN_ON_ASSET	Balakrishnan et al. 2010)
Return on Equity	Directly from Bloomberg	RETURN_COM_EQY	Return on Equity (Hou et al.2006)
Return on Invested Capital	Directly from Bloomberg	RETURN_ON_INV_CAPITAL	Return on Invested Capital (Brown and Rowe 2007)
Sales Growth	Directly from Bloomberg	SALES_GROWTH	Sales Growth (Lakonishok et al. 1994)
Sales to Accounts Receivables	Directly from Bloomberg	SALES_TO_ACCT_RCV	Sales to Receivables (Ou and Penman 1989)
Sales to Cash	Directly from Bloomberg	SALES_TO_CASH	Sales to Cash (Ou and Penman 1989)
Sales to Inventory	Directly from Bloomberg	SALES_TO_INVENT	Sales to Inventory (Ou and Penman 1989)
Sales to Price	Inverse of Price to Sales	PX_TO_SALES_RATIO	Sales to Price (Barbee et al 1996)
Total Capital	Directly from Bloomberg	BS_TOT_CAP	Organisational Capital (Eisfeldt and Papanikolaou 2013)
Total debt 1 year and 5 year growth	Directly from Bloomberg	TOTAL_DEBT_5_YEAR_GROWTH, TOTAL_DEBT_1_YEAR_GROWTH	Growth in long term debt (Richardson et al. 2005)

Figure A.3: List of features updated on a yearly basis used in our study.

As mentioned we were unable to obtain or construct Goyal and Welch's (2008) macroeconomic features pertaining to the overall index due to insufficient coverage and data. However, we did take some of the index's overall features and added them to our stock specific data set as well as taking the tensor product of the these features with our stock specific features. In total we used 13 index specific features, therefore our basic set of features contained 49 terms (13+25+21) and the set of features that accounted for interactions contained 517 terms ($(13 \times 36) + 49$). Our index specific features (that were updated on a weekly basis) are noted in the figure below:

Feature Name	Bloomberg Code
Last Price	PX_LAST
Overridable Raw Beta	BETA_RAW_OVERRIDABLE
Price Earnings Ratio (P/E)	PE_RATIO
Price to Book Ratio	PX_TO_BOOK_RATIO
RSI 14 Day	RSI_14D
RSI 3 Day	RSI_3D
RSI 30 Day	RSI_30D
RSI 9 Day	RSI_9D
Volatility 10 Day	VOLATILITY_10D
Volatility 20 Day	VOLATILITY_20D
Volatility 30 Day	VOLATILITY_30D
Volatility 60 Day	VOLATILITY_60D
Volatility 90 Day	VOLATILITY_90D

Figure A.4: List of macroeconomic features used in our study.

We would also like to note that we considered using the global macroeconomic features used by Kolanovic and Krishnamacha (p. 83, 2017) in their 2017 research into Big Data and AI Strategies that they undertook for JP Morgan Securities Plc. However, we found that too much data was missing to be used over our time period. We have included the code used to create these features from Bloomberg data in Appendix D though we did not use it to generate any predictions. The function that did this (get_daily_additional_macro_predictors) is based off code created by Ramachandran (2019) and was included with his permission.

Appendix B: Hyperparameters and Settings

Huber Regression

Our Huber Regression was sourced from Scikit-learn and added an l_2 penalty term to the loss function to provide an objective function that penalised large numbers of β coefficients of great magnitude. When we ran our regressions we kept the parameters at their default settings of $\xi = 1.35$ (which is recommended by the Scikit-learn website for 95% statistical efficiency) and a tuning parameter (that controls the degree of regularisation) of 0.0001. Though Gu, Kelly and Xiu set a 99% statistical efficiency we did not observe any change on altering either of these hyperparameters.

XGBoost

When we ran the XGBoost algorithm from the XGBoost library, the maximum depth of a tree was set to 2 (the maximum value that Gu, Kelly and Xiu (2019) used for Gradient Boosted Trees), we did not use larger values as we did not wish for our trees to grow too deep and overfit. We set the number of estimators to 1000 to obtain a high degree of accuracy (this was also the maximum value set by Gu, Kelly and Xiu (2019) for Gradient Boosted Trees). The objective function was set to the squared error as we wished to disproportionately large errors. We primarily tuned the learning rate α using values of {0.001, 0.005, 0.01}. All other settings remained at default.

Feedforward Neural Networks

As mentioned in section 5, we applied an l_1 regularisation term to our weight parameters, with a tuning parameter of 0.001. We also used TimeseriesGenerators (recommended by Brownlee (2018)) from Keras to prepare the data and set a batch size of 32. The number of epochs was set to 100 and the loss function was set to the MSE. The range of values for the learning rate was the same as that of XGBoost. This was considered the most important hyperparameter. As Walter notes (2019), too high an α and the gradients may explode and too low an α would slow down training considerably (when we tried to run with lower α values the virtual machines crashed). Our early stopping algorithm was set to a patience threshold of 10 epochs.

Miscellaneous

As an aside we note that whilst walk forward validation is accepted industry practice for tuning hyperparameters when dealing with time series data, Marcos Lopez de Prado (p. 162, 2018) notes that it can lead to overfitting on the training data. De Prado has proposed alternatives (pp163-166, 2018) but at the time of writing these have yet to be adopted as standard practice. We also acknowledge that whilst tuning more hyperparameters could have yielded better results we were unable to do so due to computational constraints. Furthermore for the XGBoost and Feedforward Neural Networks, historical excess returns were fed as a feature. We note that at all points the temporal order of the data was respected (for more details please look at the class StocksXGB and class StocksNN functions in Appendix D).

Appendix C: Results

We provide the performance metrics for each stock for the Huber Regression and the XGBoost and neural networks for both sets of predictive features. For the XGBoost and Neural Networks, we also provide the learning rate that was found to be optimal in training and therefore used in testing.

Huber Regression				
Ticker	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.34	60.33	0.0038	0.0048
ADS GY Equity	28.21	5050.23	-0.0152	0.0050
AI FP Equity	7.03	1258.04	-0.0008	-0.0007
ALV GY Equity	20.74	3711.91	0.0033	0.0070
ASML NA	22.35	4000.40	0.0037	0.0078
BAS GY Equity	6.03	1078.78	-0.0153	-0.0144
BAYN GY	14.00	2505.30	-0.0212	-0.0148
BBVA SQ Equity	0.06	11.25	0.0058	0.0117
BMW GY Equity	9.39	1681.13	0.0025	0.0047
BN FP Equity	2.44	436.10	0.0027	0.0030
BNP FP Equity	4.16	744.52	0.0085	0.0090
CRH ID Equity	0.83	148.00	0.0198	0.0199
CS FP Equity	0.61	108.93	0.0120	0.0120
DAI GY Equity	5.65	1011.84	0.0030	0.0080
DG FP Equity	3.29	588.23	0.0029	0.0093
DTE GY Equity	0.20	36.50	-0.0184	-0.0183
EL FP Equity	9.35	1673.87	0.0057	0.0062
ENEL IM Equity	0.02	2.82	-0.0130	-0.0114
ENI IM Equity	0.20	35.09	0.0120	0.0121
FP FP Equity	1.67	299.40	-0.0010	-0.0004
FRE GY Equity	4.70	840.60	0.0012	0.0012
GLE FP Equity	2.87	513.80	-0.0003	0.0004
IBE SQ Equity	0.03	5.06	-0.0023	-0.0011
INGA NA Equity	0.18	32.77	-0.0005	0.0008
ISP IM Equity	0.01	2.08	-0.0167	-0.0157
KER FP Equity	127.16	22760.83	-0.0100	0.0051
MC FP Equity	47.23	8453.45	-0.0076	0.0000
MUV2 GY	20.36	3644.69	-0.0066	-0.0053
NOKIA FH	0.05	8.54	-0.0229	-0.0226
OR FP Equity	19.63	3514.00	0.0017	0.0042
ORA FP Equity	0.19	34.68	-0.0184	-0.0183
PHIA NA Equity	0.80	144.02	0.0069	0.0107
SAF FP Equity	5.42	970.49	-0.0009	0.0141
SAN FP Equity	5.65	1010.48	0.0120	0.0124
SAN SQ Equity	0.04	6.92	-0.0131	-0.0110

SAP GY Equity	4.91	878.82	-0.0111	-0.0054
SIE GY Equity	9.54	1706.91	0.0288	0.0290
SU FP Equity	4.32	773.78	0.0084	0.0085
TEF SQ Equity	0.11	19.18	-0.0202	-0.0138
UNA NA Equity	1.56	278.78	-0.0017	0.0012
VIV FP Equity	0.41	73.74	0.0115	0.0117
VOW3 GY	56.66	10141.48	-0.0091	-0.0065

Figure C.1: Performance metrics of Huber Regression when applied to individual stocks

XGBoost (No Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.001	0.47	83.32	-0.376	-0.374
ADS GY Equity	0.005	28.36	5076.62	-0.021	0.000
AI FP Equity	0.005	6.94	1241.76	0.012	0.012
ALV GY Equity	0.001	26.46	4736.82	-0.272	-0.267
ASML NA Equity	0.001	22.51	4028.85	-0.003	0.001
BAS GY Equity	0.001	5.97	1068.86	-0.006	-0.005
BAYN GY Equity	0.001	13.87	2482.79	-0.012	-0.006
BBVA SQ Equity	0.001	0.10	17.48	-0.544	-0.535
BMW GY Equity	0.001	10.01	1791.84	-0.063	-0.061
BN FP Equity	0.001	2.41	431.56	0.013	0.013
BNP FP Equity	0.001	3.95	707.76	0.057	0.058
CRH ID Equity	0.005	0.97	174.18	-0.154	-0.154
CS FP Equity	0.001	0.68	121.45	-0.102	-0.102
DAI GY Equity	0.001	5.70	1019.50	-0.005	0.001
DG FP Equity	0.001	3.28	587.20	0.005	0.011
DTE GY Equity	0.001	0.32	57.73	-0.611	-0.611
EL FP Equity	0.001	9.35	1674.25	0.005	0.006

ENEL IM Equity	0.005	0.03	4.67	-0.673	-0.671
ENI IM Equity	0.01	0.27	48.82	-0.375	-0.375
FP FP Equity	0.01	2.54	454.62	-0.520	-0.519
FRE GY Equity	0.001	4.75	850.01	-0.010	-0.010
GLE FP Equity	0.001	4.17	746.29	-0.453	-0.452
IBE SQ Equity	0.01	0.03	5.14	-0.017	-0.016
INGA NA Equity	0.001	0.29	52.62	-0.606	-0.604
ISP IM Equity	0.01	0.02	3.10	-0.520	-0.519
KER FP Equity	0.001	125.43	22451.76	0.004	0.019
MC FP Equity	0.001	45.86	8209.59	0.022	0.029
MUV2 GY Equity	0.001	23.96	4288.67	-0.184	-0.183
NOKIA FH Equity	0.001	0.09	16.04	-0.921	-0.921
OR FP Equity	0.001	20.25	3624.96	-0.030	-0.027
ORA FP Equity	0.001	1.65	295.26	-7.672	-7.670
PHIA NA Equity	0.001	0.91	162.69	-0.122	-0.118
SAF FP Equity	0.001	5.46	977.37	-0.008	0.007
SAN FP Equity	0.001	5.70	1020.47	0.002	0.003
SAN SQ Equity	0.005	0.07	13.39	-0.960	-0.956
SAP GY Equity	0.001	5.78	1033.79	-0.189	-0.183
SIE GY Equity	0.001	9.63	1723.03	0.020	0.020
SU FP Equity	0.001	4.46	799.12	-0.024	-0.024
TEF SQ Equity	0.005	0.30	52.91	-1.814	-1.796
UNA NA Equity	0.001	1.56	278.92	-0.002	0.001
VIV FP Equity	0.001	0.60	107.00	-0.434	-0.434

VOW3 GY Equity	0.001	56.03	10030.00	0.002	0.005

Figure C.2: Performance metrics of XGBoost when applied to individual stocks with no interaction terms.

XGBoost (Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.001	0.33	58.63	0.032	0.033
ADS GY Equity	0.001	28.04	5019.80	-0.009	0.011
AI FP Equity	0.001	7.05	1261.71	-0.004	-0.004
ALV GY Equity	0.001	31.24	5592.61	-0.502	-0.496
ASML NA Equity	0.001	22.68	4059.64	-0.011	-0.007
BAS GY Equity	0.001	6.03	1080.17	-0.017	-0.016
BAYN GY Equity	0.001	13.24	2369.49	0.034	0.040
BBVA SQ Equity	0.001	0.11	18.87	-0.667	-0.657
BMW GY Equity	0.001	9.32	1667.68	0.010	0.013
BN FP Equity	0.01	2.38	426.62	0.024	0.025
BNP FP Equity	0.001	4.38	784.74	-0.045	-0.045
CRH ID Equity	0.005	0.80	143.21	0.051	0.052
CS FP Equity	0.005	1.11	198.51	-0.800	-0.800
DAI GY Equity	0.001	6.00	1073.82	-0.058	-0.053
DG FP Equity	0.001	3.25	581.58	0.014	0.021
DTE GY Equity	0.001	0.30	53.90	-0.504	-0.504
EL FP Equity	0.005	9.39	1680.39	0.002	0.002
ENEL IM Equity	0.005	0.06	11.53	-3.136	-3.130

ENI IM Equity	0.005	0.21	38.02	-0.071	-0.071
FP FP Equity	0.001	1.62	290.82	0.028	0.028
FRE GY Equity	0.001	4.78	855.45	-0.016	-0.016
GLE FP Equity	0.005	3.92	702.31	-0.367	-0.366
IBE SQ Equity	0.01	0.04	6.90	-0.366	-0.364
INGA NA Equity	0.001	0.22	39.95	-0.220	-0.218
ISP IM Equity	0.005	0.02	2.69	-0.315	-0.314
KER FP Equity	0.001	149.01	26672.91	-0.184	-0.166
MC FP Equity	0.005	46.53	8329.71	0.007	0.015
MUV2 GY Equity	0.001	28.18	5043.64	-0.393	-0.391
NOKIA FH Equity	0.001	0.22	40.24	-3.821	-3.819
OR FP Equity	0.001	19.84	3550.60	-0.009	-0.006
ORA FP Equity	0.001	0.98	174.99	-4.139	-4.139
PHIA NA Equity	0.001	0.87	155.27	-0.071	-0.067
SAF FP Equity	0.005	5.64	1010.04	-0.042	-0.026
SAN FP Equity	0.001	5.70	1020.97	0.002	0.002
SAN SQ Equity	0.005	0.05	8.11	-0.186	-0.184
SAP GY Equity	0.001	4.81	861.42	0.009	0.014
SIE GY Equity	0.001	9.71	1737.82	0.011	0.011
SU FP Equity	0.001	4.39	786.37	-0.008	-0.008
TEF SQ Equity	0.005	0.14	24.85	-0.322	-0.313
UNA NA Equity	0.001	1.61	288.18	-0.036	-0.032
VIV FP Equity	0.001	0.55	98.42	-0.319	-0.319
VOW3 GY Equity	0.001	55.85	9997.95	0.005	0.008

Figure C.3: Performance metrics of XGBoost when applied to individual stocks with interaction terms.

NN1 (No Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.001	0.52	92.68	-0.585	-0.583
ADS GY Equity	0.001	43.45	7778.06	-0.484	-0.462
AI FP Equity	0.001	15.39	2755.48	-1.313	-1.313
ALV GY Equity	0.005	33.36	5972.21	-0.600	-0.598
ASML NA Equity	0.005	25.66	4592.98	-0.084	-0.081
BAS GY Equity	0.001	8.57	1533.60	-0.473	-0.472
BAYN GY Equity	0.001	15.94	2852.39	-0.210	-0.197
BBVA SQ Equity	0.001	0.20	35.96	-2.301	-2.278
BMW GY Equity	0.001	9.13	1634.83	-0.012	-0.011
BN FP Equity	0.001	3.58	640.48	-0.476	-0.476
BNP FP Equity	0.001	7.38	1321.43	-0.828	-0.824
CRH ID Equity	0.005	1.50	268.13	-0.733	-0.733
CS FP Equity	0.005	0.96	172.10	-0.579	-0.577
DAI GY Equity	0.001	7.37	1318.75	-0.366	-0.357
DG FP Equity	0.001	4.30	770.59	-0.292	-0.290
DTE GY Equity	0.001	0.68	121.49	-2.665	-2.664
EL FP Equity	0.001	13.61	2436.94	-0.517	-0.517
ENEL IM Equity	0.001	0.02	3.81	-0.327	-0.321
ENI IM Equity	0.001	0.88	157.12	-3.474	-3.473
FP FP Equity	0.001	3.12	558.53	-0.845	-0.845

FRE GY Equity	0.001	5.62	1005.65	-0.041	-0.039
GLE FP Equity	0.005	6.53	1168.85	-1.320	-1.313
IBE SQ Equity	0.001	0.03	6.11	-0.225	-0.223
INGA NA Equity	0.005	0.44	79.65	-1.475	-1.464
ISP IM Equity	0.001	0.03	4.79	-1.525	-1.519
KER FP Equity	0.001	332.25	59472.69	-1.616	-1.581
MC FP Equity	0.001	59.17	10591.27	-0.308	-0.302
MUV2 GY Equity	0.001	40.86	7314.45	-1.055	-1.053
NOKIA FH Equity	0.001	0.48	85.10	-9.309	-9.306
OR FP Equity	0.001	42.11	7537.99	-1.272	-1.268
ORA FP Equity	0.001	0.36	64.10	-1.023	-1.023
PHIA NA Equity	0.001	1.38	246.40	-0.683	-0.681
SAF FP Equity	0.001	7.42	1328.23	-0.349	-0.339
SAN FP Equity	0.001	5.68	1016.52	-0.122	-0.119
SAN SQ Equity	0.001	0.34	60.09	-8.315	-8.299
SAP GY Equity	0.001	13.88	2485.14	-1.852	-1.843
SIE GY Equity	0.005	11.69	2092.69	-0.205	-0.205
SU FP Equity	0.001	5.68	1016.21	-0.318	-0.318
TEF SQ Equity	0.001	0.80	143.55	-7.146	-7.063
UNA NA Equity	0.001	1.76	315.12	-0.171	-0.170
VIV FP Equity	0.001	5.41	967.58	-12.154	-12.151
VOW3 GY Equity	0.001	58.84	10533.13	-0.072	-0.071

Figure C.4: Performance metrics of NN1 when applied to individual stocks with no interaction terms.

NN1 (Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.001	0.66	118.05	-1.019	-1.016
ADS GY Equity	0.001	35.11	6285.43	-0.199	-0.182
AI FP Equity	0.001	7.38	1320.36	-0.109	-0.109
ALV GY Equity	0.001	44.84	8025.49	-1.150	-1.147
ASML NA Equity	0.001	24.84	4446.25	-0.049	-0.047
BAS GY Equity	0.001	7.21	1290.18	-0.240	-0.238
BAYN GY Equity	0.001	14.46	2588.32	-0.098	-0.086
BBVA SQ Equity	0.001	0.10	17.22	-0.581	-0.569
BMW GY Equity	0.001	9.97	1784.59	-0.105	-0.104
BN FP Equity	0.001	3.13	560.96	-0.292	-0.292
BNP FP Equity	0.001	5.92	1059.18	-0.465	-0.462
CRH ID Equity	0.001	1.16	208.15	-0.345	-0.345
CS FP Equity	0.001	0.92	164.49	-0.509	-0.508
DAI GY Equity	0.001	6.04	1081.44	-0.120	-0.113
DG FP Equity	0.001	3.94	705.55	-0.183	-0.181
DTE GY Equity	0.001	0.63	112.71	-2.400	-2.400
EL FP Equity	0.001	13.86	2480.79	-0.544	-0.544
ENEL IM Equity	0.001	0.03	4.63	-0.613	-0.606
ENI IM Equity	0.001	0.37	65.74	-0.872	-0.872
FP FP Equity	0.001	2.14	382.28	-0.263	-0.263

FRE GY Equity	0.001	5.50	984.95	-0.019	-0.017
GLE FP Equity	0.001	5.16	923.66	-0.833	-0.828
IBE SQ Equity	0.001	0.03	6.14	-0.230	-0.228
INGA NA Equity	0.001	0.49	86.96	-1.702	-1.691
ISP IM Equity	0.001	0.02	3.15	-0.658	-0.655
KER FP Equity	0.001	191.55	34286.98	-0.508	-0.488
MC FP Equity	0.001	53.18	9519.01	-0.176	-0.170
MUV2 GY Equity	0.001	34.66	6203.60	-0.743	-0.741
NOKIA FH Equity	0.001	0.19	33.98	-3.117	-3.115
OR FP Equity	0.001	24.87	4451.44	-0.341	-0.339
ORA FP Equity	0.001	0.30	53.48	-0.688	-0.688
PHIA NA Equity	0.001	1.87	334.28	-1.283	-1.280
SAF FP Equity	0.001	7.30	1306.54	-0.327	-0.317
SAN FP Equity	0.001	5.61	1004.68	-0.109	-0.106
SAN SQ Equity	0.001	0.10	18.15	-1.813	-1.808
SAP GY Equity	0.001	7.94	1422.09	-0.632	-0.627
SIE GY Equity	0.001	14.04	2512.50	-0.447	-0.447
SU FP Equity	0.001	4.37	782.40	-0.015	-0.015
TEF SQ Equity	0.001	0.17	30.95	-0.756	-0.739
UNA NA Equity	0.001	1.73	310.15	-0.153	-0.151
VIV FP Equity	0.001	3.16	566.14	-6.697	-6.695
VOW3 GY Equity	0.001	57.71	10330.33	-0.051	-0.050

Figure C.5: Performance metrics of NN1 when applied to individual stocks with interaction terms.

NN2 (No Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.01	0.33	59.95	-0.025	-0.024
ADS GY Equity	0.01	31.33	5608.43	-0.070	-0.054
AI FP Equity	0.01	7.64	1367.14	-0.148	-0.148
ALV GY Equity	0.001	61.46	11001.02	-1.947	-1.943
ASML NA Equity	0.01	24.52	4388.33	-0.036	-0.033
BAS GY Equity	0.005	6.77	1211.90	-0.164	-0.163
BAYN GY Equity	0.01	15.40	2756.18	-0.169	-0.157
BBVA SQ Equity	0.01	0.06	11.12	-0.021	-0.014
BMW GY Equity	0.01	9.27	1659.04	-0.027	-0.026
BN FP Equity	0.01	2.85	510.41	-0.176	-0.176
BNP FP Equity	0.01	4.57	818.33	-0.132	-0.129
CRH ID Equity	0.01	1.02	182.70	-0.181	-0.181
CS FP Equity	0.01	0.71	127.80	-0.172	-0.171
DAI GY Equity	0.01	5.80	1037.54	-0.075	-0.068
DG FP Equity	0.01	3.50	626.10	-0.050	-0.048
DTE GY Equity	0.001	0.33	59.72	-0.802	-0.801
EL FP Equity	0.005	13.58	2429.93	-0.513	-0.513
ENEL IM Equity	0.01	0.02	3.64	-0.268	-0.263
ENI IM Equity	0.01	0.19	34.82	0.008	0.009
FP FP Equity	0.01	1.75	313.20	-0.035	-0.035

FRE GY Equity	0.01	5.33	954.96	0.012	0.014
GLE FP Equity	0.01	3.64	652.10	-0.294	-0.291
IBE SQ Equity	0.01	0.03	5.08	-0.020	-0.018
INGA NA Equity	0.01	0.22	40.10	-0.246	-0.241
ISP IM Equity	0.01	0.01	2.48	-0.305	-0.302
KER FP Equity	0.01	143.47	25680.96	-0.130	-0.115
MC FP Equity	0.01	47.77	8551.68	-0.056	-0.051
MUV2 GY Equity	0.001	29.87	5347.07	-0.502	-0.501
NOKIA FH Equity	0.01	0.09	16.41	-0.988	-0.988
OR FP Equity	0.01	19.64	3515.19	-0.059	-0.058
ORA FP Equity	0.01	0.20	35.44	-0.119	-0.119
PHIA NA Equity	0.01	0.95	169.92	-0.160	-0.159
SAF FP Equity	0.01	5.84	1044.68	-0.061	-0.053
SAN FP Equity	0.001	5.53	990.56	-0.094	-0.090
SAN SQ Equity	0.01	0.04	6.75	-0.047	-0.045
SAP GY Equity	0.01	5.85	1047.29	-0.202	-0.198
SIE GY Equity	0.001	13.77	2465.25	-0.419	-0.419
SU FP Equity	0.01	4.36	780.50	-0.012	-0.012
TEF SQ Equity	0.01	0.11	20.48	-0.162	-0.150
UNA NA Equity	0.005	1.79	321.06	-0.193	-0.192
VIV FP Equity	0.001	1.13	201.88	-1.745	-1.744
VOW3 GY Equity	0.005	61.17	10948.78	-0.114	-0.113

Figure C.6: Performance metrics of NN2 when applied to individual stocks with no interaction terms.

NN2 (Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.001	0.570	101.948	-0.744	-0.741
ADS GY Equity	0.005	30.113	5390.211	-0.028	-0.013
AI FP Equity	0.005	7.465	1336.159	-0.122	-0.122
ALV GY Equity	0.001	47.819	8559.562	-1.293	-1.290
ASML NA Equity	0.005	24.258	4342.200	-0.025	-0.022
BAS GY Equity	0.005	6.101	1092.013	-0.049	-0.048
BAYN GY Equity	0.001	13.892	2486.655	-0.055	-0.044
BBVA SQ Equity	0.001	0.087	15.519	-0.425	-0.415
BMW GY Equity	0.001	9.899	1771.915	-0.097	-0.096
BN FP Equity	0.001	3.053	546.400	-0.259	-0.259
BNP FP Equity	0.001	5.452	975.930	-0.350	-0.347
CRH ID Equity	0.001	1.168	209.003	-0.351	-0.351
CS FP Equity	0.001	0.702	125.670	-0.153	-0.152
DAI GY Equity	0.001	6.335	1133.942	-0.174	-0.167
DG FP Equity	0.001	3.702	662.580	-0.111	-0.109
DTE GY Equity	0.001	0.438	78.421	-1.366	-1.365
EL FP Equity	0.001	12.325	2206.127	-0.373	-0.373
ENEL IM Equity	0.005	0.020	3.541	-0.235	-0.229
ENI IM Equity	0.005	0.222	39.765	-0.132	-0.132
FP FP Equity	0.001	2.592	464.024	-0.533	-0.533

FRE GY Equity	0.001	5.848	1046.833	-0.083	-0.081
GLE FP Equity	0.001	5.063	906.360	-0.799	-0.794
IBE SQ Equity	0.001	0.031	5.542	-0.111	-0.109
INGA NA Equity	0.01	0.259	46.389	-0.442	-0.435
ISP IM Equity	0.001	0.017	3.124	-0.646	-0.643
KER FP Equity	0.001	137.035	24529.314	-0.079	-0.065
MC FP Equity	0.001	52.387	9377.256	-0.158	-0.153
MUV2 GY Equity	0.001	42.639	7632.299	-1.144	-1.142
NOKIA FH Equity	0.005	0.202	36.185	-3.384	-3.382
OR FP Equity	0.005	18.192	3256.333	0.019	0.020
ORA FP Equity	0.005	0.191	34.154	-0.078	-0.078
PHIA NA Equity	0.001	1.180	211.179	-0.442	-0.441
SAF FP Equity	0.005	5.634	1008.470	-0.025	-0.017
SAN FP Equity	0.005	5.172	925.720	-0.022	-0.019
SAN SQ Equity	0.005	0.040	7.154	-0.109	-0.107
SAP GY Equity	0.001	6.857	1227.338	-0.408	-0.404
SIE GY Equity	0.001	11.087	1984.643	-0.143	-0.143
SU FP Equity	0.001	4.453	797.172	-0.034	-0.034
TEF SQ Equity	0.005	0.104	18.540	-0.052	-0.041
UNA NA Equity	0.001	1.625	290.788	-0.081	-0.079
VIV FP Equity	0.001	0.812	145.366	-0.976	-0.976
VOW3 GY Equity	0.001	59.260	10607.533	-0.079	-0.078

Figure C.7: Performance metrics of NN2 when applied to individual stocks with interaction terms.

NN3 (No Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.001	0.57	101.75	-0.740	-0.738
ADS GY Equity	0.005	35.69	6389.08	-0.219	-0.201
AI FP Equity	0.005	8.21	1469.85	-0.234	-0.234
ALV GY Equity	0.001	31.00	5549.86	-0.486	-0.485
ASML NA Equity	0.005	24.30	4349.38	-0.026	-0.024
BAS GY Equity	0.01	6.01	1075.86	-0.034	-0.032
BAYN GY Equity	0.01	14.11	2525.01	-0.071	-0.060
BBVA SQ Equity	0.01	0.08	14.47	-0.329	-0.319
BMW GY Equity	0.01	9.05	1620.72	-0.004	-0.002
BN FP Equity	0.01	2.59	464.25	-0.070	-0.070
BNP FP Equity	0.01	4.40	787.37	-0.089	-0.087
CRH ID Equity	0.01	0.91	163.29	-0.055	-0.055
CS FP Equity	0.01	0.70	125.07	-0.147	-0.146
DAI GY Equity	0.01	6.13	1097.57	-0.137	-0.129
DG FP Equity	0.01	3.57	639.67	-0.073	-0.071
DTE GY Equity	0.01	0.29	52.13	-0.573	-0.572
EL FP Equity	0.01	9.22	1649.68	-0.027	-0.027
ENEL IM Equity	0.01	0.02	3.31	-0.153	-0.148
ENI IM Equity	0.01	0.21	38.37	-0.093	-0.092
FP FP Equity	0.005	2.01	359.67	-0.188	-0.188

FRE GY Equity	0.01	5.47	979.47	-0.013	-0.012
GLE FP Equity	0.01	3.94	705.14	-0.400	-0.396
IBE SQ Equity	0.01	0.03	5.10	-0.022	-0.020
INGA NA Equity	0.005	0.43	77.68	-1.414	-1.404
ISP IM Equity	0.01	0.02	2.82	-0.487	-0.484
KER FP Equity	0.01	134.34	24046.51	-0.058	-0.044
MC FP Equity	0.01	48.63	8704.58	-0.075	-0.070
MUV2 GY Equity	0.001	29.08	5205.98	-0.462	-0.461
NOKIA FH Equity	0.01	0.88	158.01	-18.142	-18.136
OR FP Equity	0.01	19.01	3402.22	-0.025	-0.024
ORA FP Equity	0.01	0.19	34.35	-0.084	-0.084
PHIA NA Equity	0.01	1.30	233.41	-0.594	-0.592
SAF FP Equity	0.005	6.04	1081.38	-0.099	-0.090
SAN FP Equity	0.01	5.36	959.91	-0.060	-0.057
SAN SQ Equity	0.01	0.04	7.54	-0.169	-0.167
SAP GY Equity	0.01	5.00	895.09	-0.027	-0.024
SIE GY Equity	0.01	9.58	1715.25	0.012	0.012
SU FP Equity	0.01	4.59	822.41	-0.067	-0.067
TEF SQ Equity	0.01	0.12	20.84	-0.183	-0.171
UNA NA Equity	0.01	1.66	296.55	-0.102	-0.101
VIV FP Equity	0.001	0.67	120.78	-0.642	-0.642
VOW3 GY Equity	0.01	56.50	10113.46	-0.029	-0.028

Figure C.8: Performance metrics of NN3 when applied to individual stocks with no interaction terms.

NN3 (Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.005	0.33	59.63	-0.020	-0.018
ADS GY Equity	0.005	30.39	5440.40	-0.038	-0.023
AI FP Equity	0.005	6.65	1191.19	0.000	0.000
ALV GY Equity	0.01	21.09	3774.70	-0.011	-0.010
ASML NA Equity	0.01	23.71	4244.38	-0.002	0.001
BAS GY Equity	0.01	5.78	1034.23	0.006	0.008
BAYN GY Equity	0.01	13.26	2373.83	-0.007	0.004
BBVA SQ Equity	0.005	0.06	10.67	0.021	0.028
BMW GY Equity	0.005	9.20	1646.11	-0.019	-0.018
BN FP Equity	0.001	2.56	458.10	-0.055	-0.055
BNP FP Equity	0.005	4.05	724.21	-0.002	0.000
CRH ID Equity	0.01	0.86	154.75	0.000	0.000
CS FP Equity	0.005	0.62	111.39	-0.022	-0.021
DAI GY Equity	0.01	5.44	973.33	-0.008	-0.002
DG FP Equity	0.005	3.31	593.31	0.005	0.007
DTE GY Equity	0.01	0.19	33.65	-0.015	-0.015
EL FP Equity	0.005	9.28	1660.57	-0.034	-0.034
ENEL IM Equity	0.005	0.02	3.03	-0.056	-0.051
ENI IM Equity	0.01	0.20	35.15	-0.001	-0.001
FP FP Equity	0.005	1.74	310.98	-0.027	-0.027

FRE GY Equity	0.01	5.40	966.04	0.000	0.002
GLE FP Equity	0.005	2.90	518.54	-0.029	-0.026
IBE SQ Equity	0.001	0.03	6.23	-0.250	-0.248
INGA NA Equity	0.01	0.19	33.60	-0.044	-0.040
ISP IM Equity	0.01	0.01	2.09	-0.103	-0.100
KER FP Equity	0.01	130.78	23409.87	-0.030	-0.016
MC FP Equity	0.01	45.40	8126.91	-0.004	0.001
MUV2 GY Equity	0.005	23.45	4198.08	-0.179	-0.178
NOKIA FH Equity	0.001	0.08	13.44	-0.628	-0.628
OR FP Equity	0.01	18.65	3339.18	-0.006	-0.005
ORA FP Equity	0.005	0.18	31.99	-0.010	-0.010
PHIA NA Equity	0.01	0.85	151.86	-0.037	-0.036
SAF FP Equity	0.005	5.78	1033.78	-0.050	-0.042
SAN FP Equity	0.005	5.14	920.12	-0.016	-0.013
SAN SQ Equity	0.005	0.04	6.59	-0.021	-0.019
SAP GY Equity	0.005	5.08	909.32	-0.044	-0.040
SIE GY Equity	0.01	9.70	1736.86	0.000	0.000
SU FP Equity	0.01	4.23	757.89	0.017	0.017
TEF SQ Equity	0.005	0.11	19.22	-0.091	-0.080
UNA NA Equity	0.005	1.50	269.17	-0.001	0.001
VIV FP Equity	0.01	0.42	74.29	-0.010	-0.010
VOW3 GY Equity	0.001	55.61	9953.32	-0.013	-0.012

Figure C.9: Performance metrics of NN3 when applied to individual stocks with interaction terms.

NN4 (No Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.01	0.55	99.03	-0.694	-0.691
ADS GY Equity	0.005	32.16	5756.09	-0.098	-0.082
AI FP Equity	0.001	8.01	1434.63	-0.204	-0.204
ALV GY Equity	0.005	25.71	4602.28	-0.233	-0.231
ASML NA Equity	0.001	24.31	4351.53	-0.027	-0.024
BAS GY Equity	0.01	6.09	1090.83	-0.048	-0.047
BAYN GY Equity	0.001	13.79	2468.91	-0.047	-0.036
BBVA SQ Equity	0.01	0.07	13.33	-0.224	-0.215
BMW GY Equity	0.01	9.04	1618.76	-0.002	-0.001
BN FP Equity	0.01	2.61	466.72	-0.075	-0.075
BNP FP Equity	0.005	4.61	825.38	-0.142	-0.139
CRH ID Equity	0.005	0.97	173.28	-0.120	-0.120
CS FP Equity	0.005	0.74	131.60	-0.207	-0.206
DAI GY Equity	0.005	6.01	1075.93	-0.114	-0.107
DG FP Equity	0.01	3.63	649.58	-0.089	-0.087
DTE GY Equity	0.01	0.24	42.64	-0.286	-0.286
EL FP Equity	0.01	9.85	1762.47	-0.097	-0.097
ENEL IM Equity	0.005	0.02	3.48	-0.215	-0.209
ENI IM Equity	0.01	0.19	34.46	0.019	0.019
FP FP Equity	0.01	1.73	310.12	-0.025	-0.024

FRE GY Equity	0.01	5.45	975.57	-0.009	-0.008
GLE FP Equity	0.005	3.58	641.36	-0.273	-0.269
IBE SQ Equity	0.001	0.03	5.09	-0.021	-0.019
INGA NA Equity	0.01	0.20	35.99	-0.119	-0.114
ISP IM Equity	0.001	0.02	2.82	-0.485	-0.482
KER FP Equity	0.01	128.88	23070.26	-0.015	-0.001
MC FP Equity	0.01	48.55	8690.49	-0.073	-0.068
MUV2 GY Equity	0.01	22.01	3940.30	-0.107	-0.106
NOKIA FH Equity	0.001	0.16	29.40	-2.562	-2.560
OR FP Equity	0.005	20.28	3629.55	-0.094	-0.092
ORA FP Equity	0.01	0.17	30.46	0.038	0.038
PHIA NA Equity	0.01	0.82	147.21	-0.005	-0.004
SAF FP Equity	0.01	5.47	978.79	0.006	0.013
SAN FP Equity	0.01	5.33	953.57	-0.053	-0.050
SAN SQ Equity	0.01	0.04	7.41	-0.149	-0.147
SAP GY Equity	0.01	5.16	924.34	-0.061	-0.058
SIE GY Equity	0.01	9.70	1736.86	0.000	0.000
SU FP Equity	0.005	4.64	830.60	-0.077	-0.077
TEF SQ Equity	0.005	0.11	19.35	-0.098	-0.087
UNA NA Equity	0.01	1.56	279.61	-0.039	-0.038
VIV FP Equity	0.01	0.43	76.52	-0.040	-0.040
VOW3 GY Equity	0.005	57.81	10348.35	-0.053	-0.052

Figure C.10: Performance metrics of NN4 when applied to individual stocks with no interaction terms.

NN4 (Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.01	0.35	62.13	-0.063	-0.061
ADS GY Equity	0.005	29.72	5319.91	-0.015	0.000
AI FP Equity	0.005	6.71	1201.46	-0.009	-0.009
ALV GY Equity	0.001	23.54	4213.42	-0.129	-0.127
ASML NA Equity	0.005	23.90	4278.29	-0.010	-0.007
BAS GY Equity	0.005	5.86	1049.24	-0.008	-0.007
BAYN GY Equity	0.01	13.84	2477.87	-0.051	-0.040
BBVA SQ Equity	0.001	0.10	17.84	-0.638	-0.626
BMW GY Equity	0.001	9.01	1613.64	0.001	0.002
BN FP Equity	0.01	2.55	456.74	-0.052	-0.052
BNP FP Equity	0.001	4.41	788.77	-0.091	-0.089
CRH ID Equity	0.001	0.87	155.78	-0.007	-0.007
CS FP Equity	0.005	0.67	120.60	-0.106	-0.105
DAI GY Equity	0.005	5.43	972.52	-0.007	-0.001
DG FP Equity	0.005	3.42	612.21	-0.027	-0.025
DTE GY Equity	0.01	0.19	34.06	-0.028	-0.028
EL FP Equity	0.01	9.07	1623.78	-0.011	-0.011
ENEL IM Equity	0.005	0.02	3.77	-0.316	-0.310
ENI IM Equity	0.005	0.20	35.29	-0.005	-0.005
FP FP Equity	0.005	1.72	308.10	-0.018	-0.018

FRE GY Equity	0.005	5.37	961.06	0.006	0.007
GLE FP Equity	0.001	3.45	617.39	-0.225	-0.222
IBE SQ Equity	0.005	0.03	5.17	-0.038	-0.036
INGA NA Equity	0.001	0.22	39.60	-0.231	-0.225
ISP IM Equity	0.01	0.01	1.95	-0.025	-0.023
KER FP Equity	0.005	129.25	23135.36	-0.018	-0.004
MC FP Equity	0.005	47.58	8515.95	-0.052	-0.047
MUV2 GY Equity	0.01	20.69	3704.13	-0.040	-0.040
NOKIA FH Equity	0.001	0.15	27.45	-2.325	-2.324
OR FP Equity	0.005	19.06	3411.63	-0.028	-0.027
ORA FP Equity	0.01	0.18	32.48	-0.025	-0.025
PHIA NA Equity	0.005	0.85	151.65	-0.036	-0.035
SAF FP Equity	0.001	5.47	979.08	0.005	0.013
SAN FP Equity	0.01	5.16	924.49	-0.021	-0.018
SAN SQ Equity	0.005	0.04	6.67	-0.034	-0.033
SAP GY Equity	0.001	5.32	953.07	-0.094	-0.090
SIE GY Equity	0.005	9.71	1737.66	0.000	0.000
SU FP Equity	0.005	4.31	772.00	-0.001	-0.001
TEF SQ Equity	0.01	0.10	17.19	0.025	0.035
UNA NA Equity	0.01	1.51	269.47	-0.002	0.000
VIV FP Equity	0.005	0.46	82.36	-0.120	-0.119
VOW3 GY Equity	0.005	58.11	10402.32	-0.058	-0.057

Figure C.11: Performance metrics of NN4 when applied to individual stocks with interaction terms.

NN5 (No Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.005	0.44	79.60	-0.361	-0.360
ADS GY Equity	0.01	30.66	5489.00	-0.047	-0.032
AI FP Equity	0.01	7.08	1267.71	-0.064	-0.064
ALV GY Equity	0.01	29.87	5346.90	-0.432	-0.430
ASML NA Equity	0.01	23.74	4250.17	-0.003	-0.001
BAS GY Equity	0.01	5.83	1042.87	-0.002	-0.001
BAYN GY Equity	0.01	13.56	2426.58	-0.029	-0.018
BBVA SQ Equity	0.01	0.08	13.65	-0.253	-0.244
BMW GY Equity	0.01	9.19	1645.89	-0.019	-0.018
BN FP Equity	0.01	2.62	469.13	-0.081	-0.081
BNP FP Equity	0.005	4.83	864.18	-0.196	-0.193
CRH ID Equity	0.005	0.93	165.64	-0.071	-0.070
CS FP Equity	0.005	0.65	115.68	-0.061	-0.060
DAI GY Equity	0.001	6.93	1239.64	-0.284	-0.276
DG FP Equity	0.01	3.35	599.13	-0.005	-0.003
DTE GY Equity	0.001	0.28	50.32	-0.518	-0.518
EL FP Equity	0.001	11.44	2047.07	-0.274	-0.274
ENEL IM Equity	0.005	0.02	3.02	-0.054	-0.049
ENI IM Equity	0.01	0.20	35.21	-0.003	-0.002
FP FP Equity	0.001	2.37	424.14	-0.401	-0.401

FRE GY Equity	0.005	5.40	965.82	0.001	0.003
GLE FP Equity	0.005	3.38	604.38	-0.200	-0.196
IBE SQ Equity	0.01	0.03	5.15	-0.032	-0.030
INGA NA Equity	0.01	0.21	37.95	-0.179	-0.174
ISP IM Equity	0.005	0.01	1.95	-0.029	-0.027
KER FP Equity	0.005	128.82	23058.61	-0.014	-0.001
MC FP Equity	0.005	49.71	8897.94	-0.099	-0.094
MUV2 GY Equity	0.01	20.05	3589.26	-0.008	-0.007
NOKIA FH Equity	0.01	0.12	21.23	-1.572	-1.571
OR FP Equity	0.01	19.87	3557.17	-0.072	-0.070
ORA FP Equity	0.01	0.21	37.40	-0.181	-0.181
PHIA NA Equity	0.01	0.95	170.93	-0.167	-0.166
SAF FP Equity	0.01	5.64	1010.33	-0.026	-0.019
SAN FP Equity	0.005	4.98	892.24	0.015	0.018
SAN SQ Equity	0.01	0.04	6.58	-0.020	-0.019
SAP GY Equity	0.01	4.89	875.03	-0.004	-0.001
SIE GY Equity	0.01	9.70	1737.11	0.000	0.000
SU FP Equity	0.01	4.31	771.42	-0.001	-0.001
TEF SQ Equity	0.005	0.11	19.41	-0.101	-0.090
UNA NA Equity	0.01	1.53	273.79	-0.018	-0.016
VIV FP Equity	0.001	1.06	189.44	-1.576	-1.575
VOW3 GY Equity	0.01	56.16	10053.40	-0.023	-0.022

Figure C.12: Performance metrics of NN5 when applied to individual stocks with no interaction terms.

NN5 (Interaction Terms)					
Ticker	**Learning Rate**	**MSE**	**SSE**	$\bar{R}_{OOS}^2$	R_{OOS}^2
AD NA Equity	0.005	0.35	62.77	-0.074	-0.072
ADS GY Equity	0.005	31.91	5712.21	-0.090	-0.074
AI FP Equity	0.005	6.66	1191.29	0.000	0.000
ALV GY Equity	0.005	20.90	3741.43	-0.002	-0.001
ASML NA Equity	0.005	23.97	4289.93	-0.012	-0.010
BAS GY Equity	0.005	5.83	1043.39	-0.002	-0.001
BAYN GY Equity	0.01	13.34	2388.14	-0.013	-0.002
BBVA SQ Equity	0.01	0.06	10.93	-0.003	0.004
BMW GY Equity	0.005	8.95	1602.63	0.008	0.009
BN FP Equity	0.005	2.67	478.27	-0.102	-0.102
BNP FP Equity	0.01	4.04	723.62	-0.001	0.001
CRH ID Equity	0.005	0.90	161.26	-0.042	-0.042
CS FP Equity	0.005	0.62	110.40	-0.013	-0.012
DAI GY Equity	0.01	5.54	992.25	-0.028	-0.021
DG FP Equity	0.005	3.31	592.69	0.006	0.008
DTE GY Equity	0.01	0.19	33.42	-0.008	-0.008
EL FP Equity	0.01	8.98	1607.56	-0.001	-0.001
ENEL IM Equity	0.01	0.02	3.06	-0.068	-0.063
ENI IM Equity	0.005	0.20	35.08	0.001	0.001
FP FP Equity	0.005	1.78	317.87	-0.050	-0.050

FRE GY Equity	0.005	5.42	969.36	-0.003	-0.001
GLE FP Equity	0.01	2.82	504.49	-0.001	0.001
IBE SQ Equity	0.005	0.03	5.08	-0.019	-0.018
INGA NA Equity	0.005	0.19	33.32	-0.035	-0.031
ISP IM Equity	0.01	0.01	1.92	-0.012	-0.010
KER FP Equity	0.005	132.48	23713.58	-0.043	-0.029
MC FP Equity	0.001	47.95	8583.82	-0.060	-0.055
MUV2 GY Equity	0.01	20.07	3591.85	-0.009	-0.008
NOKIA FH Equity	0.001	0.34	60.44	-6.323	-6.320
OR FP Equity	0.01	18.81	3366.89	-0.015	-0.013
ORA FP Equity	0.01	0.18	32.30	-0.020	-0.020
PHIA NA Equity	0.005	0.84	151.17	-0.032	-0.031
SAF FP Equity	0.005	5.50	984.37	0.000	0.008
SAN FP Equity	0.005	5.14	920.93	-0.017	-0.014
SAN SQ Equity	0.005	0.04	6.53	-0.013	-0.011
SAP GY Equity	0.01	4.91	878.15	-0.008	-0.005
SIE GY Equity	0.005	9.81	1756.70	-0.011	-0.011
SU FP Equity	0.005	4.32	772.76	-0.002	-0.002
TEF SQ Equity	0.005	0.11	19.47	-0.105	-0.094
UNA NA Equity	0.005	1.50	268.86	0.001	0.002
VIV FP Equity	0.005	0.42	74.63	-0.015	-0.014
VOW3 GY Equity	0.001	56.05	10032.14	-0.021	-0.020

Figure C.13: Performance metrics of NN5 when applied to individual stocks with interaction terms.

Appendix D

We present the Python code used in our project. Our code is modular and ran on Jupyter notebooks. We executed it on 2 Microsoft Azure Linux Virtual Machines on a pay as you go subscription (2 NC12 Promo machines each containing $2 \times$ K80 GPUs). It took approximately 3 hours to run for each neural network when using the raw set of features and 7 hours with the feature set containing interactions between stock specific and index specific terms. As previously noted in Appendix A the get_daily_additional_macro_predictors is virtually identical to the code written by Ramachandran (2019) however, the author gave his permission and it was not used to generate results. It was merely included in case our client wishes to use it in the future. We provide the functions below and a summary of what each one does. We first present the code used to prepare features and targets from the raw data and then present the code used to create, fit and test the models and obtain the performance metrics.

Data Preparation

```
import pandas as pd
import numpy as np
import os
```

Weekly features

Define a function that takes an Excel file and extract weekly features. The function returns a dictionary of 17 DataFrames. Each DataFrame includes the values of one feature for every single Stock.

```
def get_weekly_predictors(excel_file_path):

    # First obtain the betas
    df_beta = pd.read_excel(excel_file_path, "Raw Beta Hard weekly",
index_col=0, parse_dates=True).resample(
        'W').last().apply(lambda x: x.fillna(x.median()), axis=1)

    # Then the beta squareds
    df_beta_sq = pd.read_excel(excel_file_path, "Raw beta squared weekly
hard", index_col=0,

parse_dates=True).resample('W').last().apply(lambda x:
x.fillna(x.median()), axis=1)

    # Then the book to market
    df_market_to_book = pd.read_excel(excel_file_path, "Market to book hard
weekly", index_col=0,

parse_dates=True).resample('W').last().apply(lambda x:
x.fillna(x.median()),
```

```
axis=1)
    df_book_to_market = df_market_to_book.rdiv(1)

    # Then the volume
    df_volume = pd.read_excel(excel_file_path, "Volume hard weekly",
index_col=0,

parse_dates=True).resample('W').last().apply(lambda x:
x.fillna(x.median()), axis=1)

    # Then the turnover
    df_turn_over = pd.read_excel(excel_file_path, "Turnover hard weekly",
index_col=0,

parse_dates=True).resample('W').last().apply(lambda x:
x.fillna(x.median()), axis=1)

    # Then the individual volatilities
    df_volatility = pd.read_excel(excel_file_path, "10,20,30,60,90 day vol
hard", index_col=0,
                                  header=[0, 1], parse_dates=True)

    df_vol_10_day = df_volatility.loc[:, (slice(None), 'Volatility 10
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_vol_10_day.columns = df_vol_10_day.columns.droplevel(1)

    df_vol_20_day = df_volatility.loc[:, (slice(None), 'Volatility 20
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_vol_20_day.columns = df_vol_20_day.columns.droplevel(1)

    df_vol_30_day = df_volatility.loc[:, (slice(None), 'Volatility 30
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_vol_30_day.columns = df_vol_30_day.columns.droplevel(1)

    df_vol_60_day = df_volatility.loc[:, (slice(None), 'Volatility 60
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_vol_60_day.columns = df_vol_60_day.columns.droplevel(1)

    df_vol_90_day = df_volatility.loc[:, (slice(None), 'Volatility 90
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_vol_90_day.columns = df_vol_90_day.columns.droplevel(1)

    # Then get monthly bid and ask prices as well as the spread
    df_bid_ask = pd.read_excel(excel_file_path, "Bid ask hard weekly",
index_col=0, header=[0, 1], parse_dates=True)

    df_bid = df_bid_ask.loc[:, (slice(None), 'Bid
Price')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_bid.columns = df_bid.columns.droplevel(1)

    df_ask = df_bid_ask.loc[:, (slice(None), 'Ask
Price')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_ask.columns = df_ask.columns.droplevel(1)
```

```
    df_bid_ask_spread = df_bid.sub(df_ask)

    # Then the RSI
    rsi = pd.read_excel(excel_file_path, "RSI 3,9,14,30 week hard",
index_col=0, header=[0, 1], parse_dates=True)

    df_rsi_3_days = rsi.loc[:, (slice(None), 'RSI 3
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_rsi_3_days.columns = df_rsi_3_days.columns.droplevel(1)

    df_rsi_9_days = rsi.loc[:, (slice(None), 'RSI 9
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_rsi_9_days.columns = df_rsi_9_days.columns.droplevel(1)

    df_rsi_14_days = rsi.loc[:, (slice(None), 'RSI 14
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_rsi_14_days.columns = df_rsi_14_days.columns.droplevel(1)

    df_rsi_30_days = rsi.loc[:, (slice(None), 'RSI 30
Day')].resample('W').last().apply(
        lambda x: x.fillna(x.median()), axis=1)
    df_rsi_30_days.columns = df_rsi_30_days.columns.droplevel(1)

    return {'df_beta': df_beta,
            'df_beta_sq': df_beta_sq,
            'df_book_to_market': df_book_to_market,
            'df_volume': df_volume,
            'df_turn_over': df_turn_over,
            'df_vol_10_day': df_vol_10_day,
            'df_vol_20_day': df_vol_20_day,
            'df_vol_30_day': df_vol_30_day,
            'df_vol_60_day': df_vol_60_day,
            'df_vol_90_day': df_vol_90_day,
            'df_bid': df_bid,
            'df_ask': df_ask,
            'df_bid_ask_spread': df_bid_ask_spread,
            'df_rsi_3_days': df_rsi_3_days,
            'df_rsi_9_days': df_rsi_9_days,
            'df_rsi_14_days': df_rsi_14_days,
            'df_rsi_30_days': df_rsi_30_days}
```

Annual features

Define a function that takes an Excel file and extract annual features. The function returns a dictionary of 24 DataFrames. Each DataFrame includes the values of one feature for every single Stock.

```
def get_annual_predictors(excel_file_path):

    # First get the cash to debt figures for each company for every year.
First load the tab
    # Then fill in NA values with the cross sectional median as per Gu
Kelly and Xiu
```

```
    df_cash_to_debt = pd.read_excel(excel_file_path, "Cash flow to debt
hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')


    #  The load the cash productivity tab and treat the free cash flow and
cash and cash holdings figures.
    df_cash_prod = pd.read_excel(excel_file_path, "Cash Productivity Hard",
index_col=0,
                                 header=[0, 1], parse_dates=True)

    df_fcf = df_cash_prod.loc[:, (slice(None), 'Free Cash
Flow')].resample('Y').last().apply(
        lambda x: x.fillna(x.median()), axis=1).fillna(method='ffill')
    df_fcf.columns = df_fcf.columns.droplevel(1)

    df_cash = df_cash_prod.loc[:, (slice(None), 'Cash and Cash
Equivalents')].resample('Y').last().apply(
        lambda x: x.fillna(x.median()), axis=1).fillna(method='ffill')
    df_cash.columns = df_cash.columns.droplevel(1)

    df_cash_prod = df_fcf.div(df_cash)

    # Then load the cash flow to price tab
    df_price_to_cash_flow = pd.read_excel(excel_file_path, "Price to Cash
Flow Hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')
    df_cash_flow_to_price = df_price_to_cash_flow.rdiv(1)

    # Then the change in outstanding shares
    df_no_shares = pd.read_excel(excel_file_path, "Change in shares
outstanding ha", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')
    df_change_in_shares = df_no_shares - df_no_shares.shift(1)
    df_change_in_shares =
df_change_in_shares.drop(df_change_in_shares.index[0])

    # Then the current ratio
    df_current_ratio = pd.read_excel(excel_file_path, "Current Ratio Hard",
index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')

    # Then the dividend yield
    df_div_yield = pd.read_excel(excel_file_path, "Dividend Yield Hard",
index_col=0,
```

```
parse_dates=True).resample('Y').last().apply(
        lambda x: x.fillna(x.median()), axis=1).fillna(method='ffill')

    # Then the annual common equity growth
    df_tot_eq = pd.read_excel(excel_file_path, "Annual common equity growth
ha", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')
    df_an_eq_growth = df_tot_eq - df_tot_eq.shift(1)
    df_an_eq_growth = df_an_eq_growth.drop(df_an_eq_growth.index[0])

    # Then the price to earnings ratio
    df_price_ear = pd.read_excel(excel_file_path, "Price to Earnings Ratio
Hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')

    # Then the gross profitability
    df_gross_pro = pd.read_excel(excel_file_path, "Gross Profitability
Hard", index_col=0,
                                 header=[0, 1], parse_dates=True)

    df_gross_profit = df_gross_pro.loc[:, (slice(None), 'Gross
Profit')].resample('Y').last().apply(
        lambda x: x.fillna(x.median()), axis=1).fillna(method='ffill')
    df_gross_profit.columns = df_gross_profit.columns.droplevel(1)

    df_total_assets = df_gross_pro.loc[:, (slice(None), 'Total
Assets')].resample('Y').last().apply(
        lambda x: x.fillna(x.median()), axis=1).fillna(method='ffill')
    df_total_assets.columns = df_total_assets.columns.droplevel(1)

    df_gros_pro_rat = df_gross_profit.div(df_total_assets)

    # Then the 1 year growth in capex
    df_cap_ex_growth = pd.read_excel(excel_file_path, "Growth in capital
exp hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')

    # Then the 1 year employee growth
    df_employee_growth = pd.read_excel(excel_file_path, "Employee Growth
hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')

    # Then for capital expenditures and inventories
```

```
    df_cap_ex_inve = pd.read_excel(excel_file_path, "Capital Expenditures
and inv ha", index_col=0,
                                   header=[0, 1], parse_dates=True)

    df_cap_ex = df_cap_ex_inve.loc[:, (slice(None), 'Capital
Expenditures')].resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')
    df_cap_ex.columns = df_cap_ex.columns.droplevel(1)

    df_inventories = df_cap_ex_inve.loc[:, (slice(None),
'Inventories')].resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')
    df_inventories.columns = df_inventories.columns.droplevel(1)

    df_cap_inv = df_cap_ex.add(df_inventories)

    # Then financial leverage
    df_leverage = pd.read_excel(excel_file_path, "Leverage Hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then quick ratio
    df_quick_ratio = pd.read_excel(excel_file_path, "Quick Ratio hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then total capital
    df_total_capital = pd.read_excel(excel_file_path, "Total Capital hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then the return on assets
    df_roa = pd.read_excel(excel_file_path, "Return on Assets hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then the return on equity
    df_roe = pd.read_excel(excel_file_path, "Return on Equity hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then the return on invested capital
    df_roi = pd.read_excel(excel_file_path, "Return on invested capital
hard", index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then the sales to inventory ratio
    df_sales_to_inv = pd.read_excel(excel_file_path, "Sales to inventories
hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),
```

```
axis=1).fillna(method='ffill')

    # Then the sales to accounts receivables
    df_sales_to_acc = pd.read_excel(excel_file_path, "Sales to accounts
receivables h", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')

    # Then the sales to price
    df_price_to_sales = pd.read_excel(excel_file_path, "Price to sales
hard", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')
    df_sales_to_price = df_price_to_sales.rdiv(1)

    # Then the sales growth
    df_sales_grow = pd.read_excel(excel_file_path, "Sales Growth Hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then sales to cash
    df_sales_cash = pd.read_excel(excel_file_path, "Sales to Cash hard",
index_col=0, parse_dates=True).resample(
        'Y').last().apply(lambda x: x.fillna(x.median()),
axis=1).fillna(method='ffill')

    # Then the sales to inventory % change
    df_sales_inv_per = pd.read_excel(excel_file_path, "Sales to inventories
hard %", index_col=0,

parse_dates=True).resample('Y').last().apply(lambda x:
x.fillna(x.median()),

axis=1).fillna(method='ffill')

    df_sales_inv_perc = ((df_sales_inv_per.div(df_sales_inv_per.shift(1)))
- 1) * 100
    df_sales_inv_perc = df_sales_inv_perc.drop(df_sales_inv_perc.index[0])

    return {'df_cash_to_debt': df_cash_to_debt,
            'df_cash_prod': df_cash_prod,
            'df_cash_flow_to_price': df_cash_flow_to_price,
            'df_change_in_shares': df_change_in_shares,
            'df_current_ratio': df_current_ratio,
            'df_div_yield': df_div_yield,
            'df_an_eq_growth': df_an_eq_growth,
            'df_price_ear': df_price_ear,
            'df_gros_pro_rat': df_gros_pro_rat,
            'df_employee_growth': df_employee_growth,
            'df_cap_ex_growth': df_cap_ex_growth,
            'df_cap_inv': df_cap_inv,
            'df_leverage': df_leverage,
            'df_quick_ratio': df_quick_ratio,
```

```
            'df_total_capital': df_total_capital,
            'df_roa': df_roa,
            'df_roe': df_roe,
            'df_roi': df_roi,
            'df_sales_to_inv': df_sales_to_inv,
            'df_sales_to_acc': df_sales_to_acc,
            'df_sales_to_price': df_sales_to_price,
            'df_sales_grow': df_sales_grow,
            'df_sales_cash': df_sales_cash,
            'df_sales_inv_perc': df_sales_inv_perc}
```

Macro features

Define a function that takes an Excel file and extract Macro features. The function returns one DataFrame, which will be used for all stocks. (This function was applied to the Eurostoxx data)

```
def get_macro_predictors(excel_file_path, period):

    sheet = None
    resample_by = None

    if period == 'daily':
        sheet = 'Eurostoxx data daily hard'
        resample_by = 'D'
    elif period == 'weekly':
        sheet = 'Eurostoxx data weekly hard'
        resample_by = 'W'
    elif period == 'monthly':
        sheet = 'Eurostoxx data monthly hard'
        resample_by = 'M'

    df_macro = pd.read_excel(excel_file_path, sheet, index_col=0, 
parse_dates=True).resample(
        resample_by).last().drop(columns=['BEst Div Yld']).apply(lambda x: 
x.fillna(x.interpolate()),
                                                                          axis=1)

    # Read in the price to book ratio and then invert it
    df_macro['Price to Book Ratio'] = df_macro['Price to Book 
Ratio'].rdiv(1)

    # Then the price to earnings ratio
    df_macro['Price Earnings Ratio (P/E)'] = df_macro['Price Earnings Ratio 
(P/E)'].rdiv(1)

    return df_macro
```

Euribor features

Define a function that takes an Excel file and extract Euribor features. The function returns one DataFrame, which will be used for all stocks:

```
def get_euribor_rates(excel_file_path, period):

    sheet = None
```

```
    resample_by = None
    drop_col = None

    if period == 'weekly':
        sheet = '1 week Euribor Hard'
        resample_by = 'W'
        drop_col = ['EUR001W Index']
    elif period == 'monthly':
        sheet = '1 month Euribor Hard'
        resample_by = 'M'
        drop_col = ['EUR001M Index']

    df_euribo = pd.read_excel(excel_file_path, sheet, index_col=0,
parse_dates=True).resample(
        resample_by).last().drop(columns=drop_col).apply(lambda x:
x.fillna(x.interpolate()),
                                                                 axis=1)

    return df_euribo
```

Additional Macro features

Define a function that takes an Excel file and extract Additional Macro features. The function returns one DataFrame, which will be used for all stocks. This code was taken from Ramachanrdan (2019) with his permission but not used due to lack of data.

```
def get_daily_additional_macro_predictors(excel_file_path):

    # Dataframe of Spot prices. Remove all zero values
    df_spot = pd.read_excel(excel_file_path, "FX SPOT HARD",
parse_dates=True, index_col=0)
    df_spot = df_spot[(df_spot != 0).all(axis=1)]

    # Calculating spot returns to be further used in calculating 2M
realized volatilities
    # TODO: by default the result is written back to the edge of the window
but we can make it at the center
    # TODO: why shifting by 1?
    df_returns = df_spot.pct_change().dropna()
    df_realized_vol_2m = (df_returns.rolling(window=22 * 2).std() *
np.sqrt(252)).shift(1).dropna()
    df_realized_vol_2m.columns = [col + ' Vol2M' for col in
df_realized_vol_2m.columns]

    # Calculating 1W change in realized Volatilities
    # TODO: the shift changed from 3 to 5
    df_1w_vol_per_change = (df_realized_vol_2m /
df_realized_vol_2m.shift(5) - 1).dropna()
    df_1w_vol_per_change.columns = [col + ' 1W' for col in
df_1w_vol_per_change.columns]

    # Calculating 1month change in realized Volatilities
    df_1m_vol_per_change = (df_realized_vol_2m /
df_realized_vol_2m.shift(22) - 1).dropna()
    df_1m_vol_per_change.columns = [col + ' 1M' for col in
df_1m_vol_per_change.columns]

    # join Volatilite, 1W change in vols and 1M change in realized vols
```

```
    df_main =
df_realized_vol_2m.join(df_1w_vol_per_change).join(df_1m_vol_per_change).dr
opna()

    for sheet in ["ATM VOLS HARD", "3M 25D RR HARD"]:
        df = pd.read_excel(excel_file_path, sheet, parse_dates=True,
index_col=0).dropna(axis=1)
        df = df[(df != 0).all(axis=1)]
        df_main = df_main.join(df.shift(1)).dropna()

    # looping through sheets to calculate 1week and 1month change
    # and joining them with df_main
    for sheet in ["FX SPOT HARD", "ATM VOLS HARD", "3M 25D RR HARD", "3M
DEPOSIT RATES HARD", "10Y YIELD HARD",
                  "EQUITY INDICES HARD", "COMDTY HARD", "CREDIT SPREADS
HARD", "IMM POSITIONING HARD"]:
        df = pd.read_excel(excel_file_path, sheet, parse_dates=True,
index_col=0).dropna(axis=1)
        df = df[(df != 0).all(axis=1)]

        df_1w_per_change = (df / df.shift(5) - 1).dropna()
        df_1w_per_change.columns = [col + ' 1W' for col in
df_1w_per_change.columns]
        df_main = df_main.join(df_1w_per_change.shift(1)).dropna()

        df_1m_per_change = (df / df.shift(22) - 1).dropna()
        df_1m_per_change.columns = [col + ' 1M' for col in
df_1m_per_change.columns]
        df_main = df_main.join(df_1m_per_change.shift(1)).dropna()

    # Remove all zero values
    df_easi = pd.read_excel(excel_file_path, "JPM EASI HARD",
parse_dates=True, index_col=0).dropna(axis=1)
    df_easi = df_easi[(df_easi != 0).all(axis=1)]

    # JPM EASI is an index value between -100 to +100, so we have divided
by total
    # range (200) to find out change in 1W and 1M
    df_easi_1w = ((df_easi - df_easi.shift(5)) / 200).dropna()
    df_easi_1w.columns = [col + ' 1W' for col in df_easi_1w.columns]
    df_main = df_main.join(df_easi_1w.shift(1)).dropna()

    df_easi_1m = ((df_easi - df_easi.shift(22)) / 200).dropna()
    df_easi_1m.columns = [col + ' 1M' for col in df_easi_1m.columns]
    df_main = df_main.join(df_easi_1m.shift(1)).dropna()

    df_main.to_csv(os.path.join(os.getcwd(), 'data',
'Daily_Additional_Macro_Processed.csv'))

    return df_main
```

Save Engineered Features Per Stock

Define a function that creates one CSV file per stock. The CSV file includes all engineered features (and the target), for a certain period of time (daily, monthly, annually)

```
def save_csv_per_stock(df_returns,
                       df_excess_return,
                       df_monthly_momentum,
                       df_6_months_momentum,
```

```
                       df_12_months_momentum,
                       features_df_dict,
                       df_macro_features,
                       df_additional_macro_features,
                       annual_features_df_dict,
                       tensor_product,
                       period):
    """
    Join all given features for a given period for all stocks
    Save each stock results in a csv file
    """
    # loop over stocks
    for stock in df_returns.columns:
        print(f'Processing stock {stock}...')

        df = pd.DataFrame(columns=pd.MultiIndex(levels=[[], []],
                                                codes=[[], []],
                                                names=['data',
'features']))

        save_in_sub_dir = f'{period}_features'

        # loop over dict items {'predictor name': predictor_DataFrame}
        for k, v in features_df_dict.items():
            df['stock_features', k] = v[stock]

        # inner join with macro
        if df_macro_features is not None:
            # make a copy and add a column level so we can join
            df_macro_features_temp = df_macro_features.copy()
            df_macro_features_temp.columns =
pd.MultiIndex.from_product([['macro_features'],

df_macro_features_temp.columns])
            df = df.join(df_macro_features_temp).apply(lambda x:
x.fillna(x.median()), axis=0)

        if df_monthly_momentum is not None:
            # make a copy and add a column level so we can join
            df_monthly_momentum_temp = df_monthly_momentum.copy()
            df_monthly_momentum_temp[stock].name = ('momentum_features',
'momentum_1M')
            df = df.join(df_monthly_momentum_temp[stock], how='inner')

        if df_6_months_momentum is not None:
            # make a copy and add a column level so we can join
            df_6_months_momentum_temp = df_6_months_momentum.copy()
            df_6_months_momentum_temp[stock].name = ('momentum_features',
'momentum_6M')
            df = df.join(df_6_months_momentum_temp[stock], how='inner')

        if df_12_months_momentum is not None:
            # make a copy and add a column level so we can join
            df_12_months_momentum_temp = df_12_months_momentum.copy()
            df_12_months_momentum_temp[stock].name = ('momentum_features',
'momentum_12M')
            df = df.join(df_12_months_momentum_temp[stock], how='inner')

        # inner join with additional macro
        if df_additional_macro_features is not None:
            # make a copy and add a column level so we can join
```

```
            df_additional_macro_features_temp =
df_additional_macro_features.copy()
            df_additional_macro_features_temp.columns =
pd.MultiIndex.from_product([['additional_macro_features'],

df_additional_macro_features_temp.columns])
            df = df.join(df_additional_macro_features_temp).apply(lambda x:
x.fillna(x.median()), axis=0)

        # augment with annual features if given
        # loop over dict items {'predictor name': predictor_DataFrame}
        if annual_features_df_dict is not None:
            df_annual = pd.DataFrame()

            for k, v in annual_features_df_dict.items():
                df_annual[k] = v[stock]

            # add a column level so we can join
            df_annual.columns =
pd.MultiIndex.from_product([['annual_features'], df_annual.columns])

            # re-sampling annual to daily/weekly results in NAs for all
days/weeks but the last
            # inner join
            if period == 'daily':
                df =
df.join(df_annual.resample('D').last().fillna(method='ffill')).dropna()
            elif period == 'weekly':
                df =
df.join(df_annual.resample('W').last().fillna(method='ffill')).dropna()
            elif period == 'monthly':
                df =
df.join(df_annual.resample('M').last().fillna(method='ffill')).dropna()

        if tensor_product:
            # returns a series of lists
            s_tensor_product = df.apply(lambda s:
np.kron(s[['stock_features', 'momentum_features', 'annual_features']],

s[['macro_features']]), axis=1)
            # convert series of lists to df
            df_tensor_product =
pd.DataFrame.from_dict(dict(zip(s_tensor_product.index,
s_tensor_product.values))).T

            # add a column level so we can join
            df_tensor_product.columns =
pd.MultiIndex.from_product([['tensor_product'], df_tensor_product.columns])
            df = df.join(df_tensor_product).apply(lambda x:
x.fillna(x.median()), axis=0)

        # make a copy and add a column level so we can join
        df_returns_temp = df_returns.copy()
        df_returns_temp[stock].name = ('returns', 'return')
        df = df.join(df_returns_temp[stock]).apply(lambda x:
x.fillna(x.median()), axis=0)

        if df_excess_return is not None:
            # make a copy and add a column level so we can join
            df_excess_return_temp = df_excess_return.copy()
```

```
            df_excess_return_temp[stock].name = ('returns',
'excess_return')
            df = df.join(df_excess_return_temp[stock]).dropna(axis=0)

        # make sure there's a sub directory to save results to
        save_in = os.path.join(os.getcwd(), 'data', save_in_sub_dir)
        if not os.path.exists(save_in):
            os.makedirs(save_in)

        df.to_csv(os.path.join(save_in, f'{stock}.csv'))
```

Data Wrangling

All Excel files paths

```
macro_features_file_path = os.path.join(os.getcwd(), 'data',
r'Macro_Features.xlsx')
daily_features_file_path = os.path.join(os.getcwd(), 'data',
'Daily_Features.xlsx')
daily_additional_macro_features_file_path = os.path.join(os.getcwd(),
'data', r'Daily_Additional_Macro.xlsx')
daily_prices_file_path = os.path.join(os.getcwd(), 'data',
'Daily_Prices.csv')
weekly_prices_file_path = os.path.join(os.getcwd(), 'data',
'Weekly_Prices.csv')
weekly_euribor_file_path = os.path.join(os.getcwd(), 'data',
'Euribor_Rates.xlsx')
weekly_features_file_path = os.path.join(os.getcwd(), 'data',
'Weekly_Features.xlsx')
monthly_prices_file_path = os.path.join(os.getcwd(), 'data',
'Monthly_Prices.csv')
monthly_euribor_file_path = os.path.join(os.getcwd(), 'data',
'Euribor_Rates.xlsx')
monthly_features_file_path = os.path.join(os.getcwd(), 'data',
'Monthly_Features.xlsx')
annual_features_file_path = os.path.join(os.getcwd(), 'data',
'Annual_Features.xlsx')
```

Weekly returns and features for each Stock as CSV

```
df_weekly_prices = pd.read_csv(weekly_prices_file_path, index_col=0,
parse_dates=True).dropna(
    axis='columns').resample('W').last()

df_weekly_returns = df_weekly_prices.apply(lambda s: s -
s.shift(1)).dropna()
df_weekly_momentum_1m = df_weekly_prices.apply(lambda s: s / s.shift(4) -
1).dropna()
df_weekly_momentum_6m = df_weekly_prices.apply(lambda s: s / s.shift(24) -
1).dropna()
df_weekly_momentum_12m = df_weekly_prices.apply(lambda s: s / s.shift(48) -
1).dropna()

df_weekly_euribor = get_euribor_rates(weekly_euribor_file_path,
period='weekly')
```

```
df_weekly_excess_return =
df_weekly_returns.subtract(df_weekly_euribor.iloc[:, 0] / 100, axis=0)
df_dict_weekly_features = get_weekly_predictors(weekly_features_file_path)
df_weekly_macro_features = get_macro_predictors(macro_features_file_path,
'weekly')
df_weekly_additional_macro_features =
df_daily_additional_macro_features.resample('W').last().dropna()


save_csv_per_stock(df_weekly_returns,
                   df_weekly_excess_return,
                   df_weekly_momentum_1m,
                   df_weekly_momentum_6m,
                   df_weekly_momentum_12m,
                   df_dict_weekly_features,
                   df_weekly_macro_features,
                   df_weekly_additional_macro_features,
                   df_dict_annual_features,
                   tensor_product=True,
                   period='weekly')
```

Modelling

```
import os
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.linear_model import LinearRegression, HuberRegressor
from sklearn.metrics import mean_squared_error, r2_score
import xgboost as xgb
from glob import glob
from tqdm import tqdm
import tensorflow as tf

tf.logging.set_verbosity(tf.logging.ERROR)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

from keras.layers import Input, Dense, LSTM, BatchNormalization, Flatten
from keras.models import Model
from keras.initializers import glorot_uniform
from keras.preprocessing.sequence import TimeseriesGenerator
from keras import optimizers, regularizers, activations
from keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau, TensorBoard
import keras.backend as k
```

NN model with 1 Dense layer

```
def nn1(input_shape_0, input_shape_1, activation, seed,
kernel_regularizer):
    input_tensor = Input(shape=(input_shape_0, input_shape_1),
                         dtype='float32',
                         name='input_tensor')
```

```
    dense_1 = Dense(units=32,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_1')(input_tensor)

    dense_1_flatten = Flatten(name='dense_1_flattened')(dense_1)

    # add a regression layer
    output_tensor = Dense(units=1,
                          activation=None,
                          name='output_tensor')(dense_1_flatten)

    # specify input and output
    return Model(input_tensor, output_tensor)
```

NN model with 2 Dense layers

```
def nn2(input_shape_0, input_shape_1, activation, seed,
kernel_regularizer):
    input_tensor = Input(shape=(input_shape_0, input_shape_1),
                         dtype='float32',
                         name='input_tensor')

    dense_1 = Dense(units=32,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_1')(input_tensor)

    dense_2 = Dense(units=16,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_2')(dense_1)

    dense_2_flatten = Flatten(name='dense_2_flatten')(dense_2)

    # add a regression layer
    output_tensor = Dense(units=1,
                          activation=None,
                          name='output_tensor')(dense_2_flatten)

    # specify input and output
    return Model(input_tensor, output_tensor)
```

NN model with 3 Dense layers

```
def nn3(input_shape_0, input_shape_1, activation, seed,
kernel_regularizer):
    input_tensor = Input(shape=(input_shape_0, input_shape_1),
                         dtype='float32',
                         name='input_tensor')

    dense_1 = Dense(units=32,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_1')(input_tensor)
```

```
    dense_2 = Dense(units=16,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_2')(dense_1)

    dense_3 = Dense(units=8,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_3')(dense_2)

    dense_3_flatten = Flatten(name='dense_2_flatten')(dense_3)

    # add a regression layer
    output_tensor = Dense(units=1,
                          activation=None,
                          name='output_tensor')(dense_3_flatten)

    # specify input and output
    return Model(input_tensor, output_tensor)
```

NN model with 4 Dense layers

```
def nn4(input_shape_0, input_shape_1, activation, seed,
kernel_regularizer):
    input_tensor = Input(shape=(input_shape_0, input_shape_1),
                         dtype='float32',
                         name='input_tensor')

    dense_1 = Dense(units=32,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_1')(input_tensor)

    dense_2 = Dense(units=16,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_2')(dense_1)

    dense_3 = Dense(units=8,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_3')(dense_2)

    dense_4 = Dense(units=4,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_4')(dense_3)

    dense_4_flatten = Flatten(name='dense_4_flatten')(dense_4)

    # add a regression layer
    output_tensor = Dense(units=1,
                          activation=None,
```

```
                          name='output_tensor')(dense_4_flatten)

    # specify input and output
    return Model(input_tensor, output_tensor)
```

NN model with 5 Dense layers

```
def nn5(input_shape_0, input_shape_1, activation, seed,
kernel_regularizer):
    input_tensor = Input(shape=(input_shape_0, input_shape_1),
                         dtype='float32',
                         name='input_tensor')

    dense_1 = Dense(units=32,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_1')(input_tensor)

    dense_2 = Dense(units=16,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_2')(dense_1)

    dense_3 = Dense(units=8,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_3')(dense_2)

    dense_4 = Dense(units=4,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_4')(dense_3)

    dense_5 = Dense(units=4,
                    activation=activation,
                    kernel_regularizer=kernel_regularizer,
                    kernel_initializer=glorot_uniform(random.seed(seed)),
                    name='dense_5')(dense_4)

    dense_5_flatten = Flatten(name='dense_5_flatten')(dense_5)

    # add a regression layer
    output_tensor = Dense(units=1,
                          activation=None,
                          name='output_tensor')(dense_5_flatten)

    # specify input and output
    return Model(input_tensor, output_tensor)
```

Define a class for Neural Network

```
class StocksNN:
    def __init__(self, features_file_path, period, lookback, step,
features, target):
        """
```

```
        :param features_file_path: path to the csv file that includes the
features
        :param period: daily or weekly
        :param lookback: number of time-steps in the past to use for
predicting the next time-step
        :param step: number of steps for sampling
        :param features: a list of the features to use in the model
        :param target: return or excess_return
        """

        self.period = period
        self.lookback = lookback
        self.step = step
        self.target = target

        # make sure there's a sub directory to save results to
        self.save_in = os.path.join(os.getcwd(), 'results',
f'{self.period}_features')
        if not os.path.exists(self.save_in):
            os.makedirs(self.save_in)

        # concatenate a unique string for this model
        file_name = os.path.basename(features_file_path)
        file_root, file_ext = os.path.splitext(file_name)
        self.unique_string = '_'.join(file_root.lower().split())

        # -------------------------------------------------- READ DATA
        # read data and choose only features wanted for this model from the
multi-indexed header
        self.df = pd.read_csv(features_file_path, index_col=[0], header=[0,
1], parse_dates=True)
        self.df = self.df[features].droplevel(level=0, axis=1)

        # drop either return or excess_return
        if self.target == 'return':
            self.df = self.df.drop('excess_return', axis=1)
        elif self.target == 'excess_return':
            self.df = self.df.drop('return', axis=1)

        # -------------------------------------------------- SELECT DATA
INDICES FOR SPLITTING
        # split data 80/20 for training/testing without shuffling to keep
time order
        # keep target as a feature in data
        # the initial training data will be split later into n splits
        self.X_train_init, self.X_test_init = train_test_split(self.df,
test_size=0.20, shuffle=False)

        # select a number of splits equals to the number of years in
training data
        self.n_split = len(set(self.X_train_init.index.year))

        # --------------------------------------------------- DATA SCALING
        # fit scaler to training data only (including target as it will be
a feature as well)
        # select target
        self.scaler = StandardScaler()
        self.X_train_init = self.scaler.fit_transform(self.X_train_init)
        self.y_train_init = self.X_train_init[:, -1]
```

```
        # transform test data (including target as it will be a feature as
well)
        # select target
        self.X_test = self.scaler.transform(self.X_test_init)
        self.y_test = self.X_test[:, -1]

    # ------------------------------------------------- BUILD & COMPILE
MODEL
    def build_and_compile_model(self, model_function, model_name,
activation, optimizer, seed, kernel_regularizer):
        self.input_shape_0 = self.lookback // self.step
        self.input_shape_1 = self.X_train_init.shape[1]

        self.model = model_function(self.input_shape_0, self.input_shape_1,
activation, seed, kernel_regularizer)

        self.model.compile(optimizer=optimizer,
                           loss='mse')

        # append unique string
        self.unique_string =
f'{self.unique_string}_{model_name}_{round(k.eval(self.model.optimizer.lr),
3)}'

    # ------------------------------------------------- WALK FORWARD
VALIDATION
    def walk_forward_validation(self, batch_size, epochs, n_splits=None):

        self.batch_size = batch_size

        # if n_splits is not provided, use number of years in training data
as above
        if n_splits is None:
            n_splits = self.n_split

        # split initial training data for training/validation
        # forward walking validation
        tscv = TimeSeriesSplit(n_splits=n_splits)
        eval_scores = list()
        for train_index, val_index in tscv.split(self.X_train_init):
            X_train, X_val = self.X_train_init[train_index],
self.X_train_init[val_index]
            y_train, y_val = self.y_train_init[train_index],
self.y_train_init[val_index]

            # ----------------------------------------- CREATE GENERATORS
            # generator output is a list of batches
            # each batch is a tuple of (samples, targets)
            train_gen = TimeseriesGenerator(X_train,
                                            y_train,
                                            length=self.lookback,
                                            sampling_rate=self.step,
                                            stride=1,
                                            batch_size=self.batch_size)

            val_gen = TimeseriesGenerator(X_val,
                                          y_val,
                                          length=self.lookback,
                                          sampling_rate=self.step,
                                          stride=1,
                                          batch_size=self.batch_size)
```

```
            # ----------------------------------------- FIT & EVALUATE
            # interrupt training when improvement stops
            # continually save the model during training (only current
best)
            # reduce learning rate when the validation loss has stopped
improving
            callbacks_list = [EarlyStopping(monitor='val_loss',
                                            patience=5),

ModelCheckpoint(filepath=os.path.join(self.save_in,
f'{self.unique_string}.h5'),
                                              monitor='val_loss',
                                              save_best_only=True),
                              ReduceLROnPlateau(monitor='val_loss',
                                                factor=0.1,
                                                patience=10)
                             ]

            self.model.fit_generator(train_gen,
                                     steps_per_epoch=len(train_gen),
                                     epochs=epochs,
                                     validation_data=val_gen,
                                     validation_steps=len(val_gen),
                                     callbacks=callbacks_list,
                                     verbose=False)

            # get eval metric and append to list
            eval_score = self.model.evaluate_generator(val_gen,
steps=len(val_gen))
            eval_scores.append(eval_score)

        # average evaluation scores
        return pd.Series(np.mean(eval_scores), index=['mse'])

    def predict_on_test_data_nn(self):
        self.test_gen = TimeseriesGenerator(self.X_test,
                                            self.y_test,
                                            length=self.lookback,
                                            sampling_rate=self.step,
                                            stride=1,
                                            batch_size=self.batch_size)

        self.predictions = self.model.predict_generator(self.test_gen,

steps=len(self.test_gen))

        # inverse transform predictions to un-normalize
        # first, repeat as many times as the transformer expects
        self.predictions = np.repeat(self.predictions, self.df.shape[1],
axis=1)
        self.predictions =
self.scaler.inverse_transform(self.predictions)[:, -1]

        # get test data using the generator to get corresponding targets
        # each batch is a tuple of numpy arrays (samples, targets), get the
targets as a list comprehension
        # convert list of numpy arrays into a numpy vector, reshape into 1D
array
        self.true = np.concatenate([batch[1] for batch in self.test_gen],
axis=0).reshape(-1, 1)
```

```
        # inverse transform targets to un-normalize
        # first, repeat as many times as the transformer expects
        self.true = np.repeat(self.true, self.df.shape[1], axis=1)
        self.true = self.scaler.inverse_transform(self.true)[:, -1]

        # index was lost when scaling. Get the index from self.X_test_init
        # however, we will loose few of the last samples in self.test_gen.
        # use the length of self.true to get the exact number of test
samples used
        return pd.DataFrame({'true': self.true, 'predictions':
self.predictions},

index=self.X_test_init.index[:self.true.shape[0]])

    def predict_on_test_data_lewellen(self, n_splits=None):

        df_size_factors = None

        # if n_splits is not provided, use number of years in training data
as above
        if n_splits is None:
            n_splits = self.n_split

        if self.period == 'daily':
            df_size_factors = pd.read_csv(os.path.join(os.getcwd(), 'data',
'Daily_F_F_Research_Data_Factors.csv'),
                                          index_col=0,
                                          parse_dates=True)
        elif self.period == 'weekly':
            df_size_factors = pd.read_csv(os.path.join(os.getcwd(), 'data',
'Weekly_F_F_Research_Data_Factors.csv'),
                                          index_col=0,

parse_dates=True).resample('W').last()

        # select only indexes in stocks
        df_size_factors =
df_size_factors[df_size_factors.index.isin(self.df.index)]

        # select target and shift by 1 time-step and drop first row
        # concatenate features and remove first row
        y = self.df[self.target].shift(self.lookback).dropna()
        X = pd.concat([df_size_factors['SMB'],
self.df['df_book_to_market'], self.df['momentum_1M']],
                      axis=1, join='inner').iloc[self.lookback:]

        # rows should have the same order as the first train_test_split
        X_train, X_test, y_train, true_lewellen = train_test_split(X, y,
test_size=0.20, shuffle=False)

        # fit scaler to training data only
        # transform test data
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        # create a Huber regressor
        model = HuberRegressor(alpha=0.0001)

        # split initial training data for training/validation
```

```
        # forward walking validation
        tscv = TimeSeriesSplit(n_splits=n_splits)
        eval_scores = list()
        for train_index, val_index in tscv.split(self.X_train_init):
            X_train, X_val = self.X_train_init[train_index],
self.X_train_init[val_index]
            y_train, y_val = self.y_train_init[train_index],
self.y_train_init[val_index]

            # fit Huber regressor
            model = HuberRegressor(alpha=0.0001)
            model.fit(X_train, y_train)

        # predict on test data
        predictions_lewellen = model.predict(X_test)

        # create df with all time-steps. Select only indexes that are in
the self.X_test_init
        df = pd.DataFrame({'true': true_lewellen, 'predictions':
predictions_lewellen})

        return
df[df.index.isin(self.X_test_init.index[:self.true.shape[0]])]
```

Define a class for XGBoost

```
class StocksXGB:
    def __init__(self, features_file_path, period, lookback, step,
features, target):
        """
        :param features_file_path: path to the csv file that includes the
features
        :param period: daily or weekly
        :param lookback: number of time-steps in the past to use for
predicting the next time-step
        :param step: number of steps for sampling
        :param features: a list of the features to use in the model
        :param target: return or excess_return
        """

        self.period = period
        self.lookback = lookback
        self.step = step
        self.target = target

        # make sure there's a sub directory to save results to
        self.save_in = os.path.join(os.getcwd(), 'results',
f'{self.period}_features')
        if not os.path.exists(self.save_in):
            os.makedirs(self.save_in)

        # concatenate a unique string for this model
        file_name = os.path.basename(features_file_path)
        file_root, file_ext = os.path.splitext(file_name)
        self.unique_string = '_'.join(file_root.lower().split())

        # ------------------------------------------------- READ DATA
        # read data and choose only features wanted for this model from the
multi-indexed header
```

```
        self.df = pd.read_csv(features_file_path, index_col=[0], header=[0,
1], parse_dates=True)
        self.df = self.df[features].droplevel(level=0, axis=1)

        # drop either return or excess_return
        if self.target == 'return':
            self.df = self.df.drop('excess_return', axis=1)
        elif self.target == 'excess_return':
            self.df = self.df.drop('return', axis=1)

        # ------------------------------------------------- FEATURES
EXTRACTIONS
        # extract features to infer time
        # this's important in xgb as it selects samples randomly so time-
order is lost
        self.df['month'] = self.df.index.month
        self.df['quarter'] = self.df.index.quarter
        self.df['year'] = self.df.index.year

        if self.period == 'daily':
            self.df['dayofyear'] = self.df.index.dayofyear
            self.df['dayofmonth'] = self.df.index.day
        elif self.period == 'weekly':
            self.df['weekofyear'] = self.df.index.weekofyear

        # shift return by lookback so the target is not the current return
but in the future
        # this is similar to what the keras generator does for the NN model
        # the old return will be used as a normal feature now (similar to
the NN model as well)
        self.df['return_future'] =
self.df[self.target].shift(self.lookback)

        # ------------------------------------------------- SPLIT DATA
        # split data 80/20 for training/testing without shuffling to keep
time order
        # keep target as a feature in data
        # the initial training data will be split later into n splits
        self.X_train_init, self.X_test_init =
train_test_split(self.df.dropna(), test_size=0.20, shuffle=False)

        # select a number of splits equals to the number of years in
training data
        self.n_split = len(set(self.X_train_init.index.year))

        # trees don't require scaling or centering of data
        self.y_train_init = self.X_train_init['return_future'].values
        self.X_train_init = self.X_train_init.drop(['return_future'],
axis=1).values

        self.y_test = self.X_test_init['return_future'].values
        self.X_test = self.X_test_init.drop(['return_future'],
axis=1).values

    # ------------------------------------------------- BUILD & COMPILE
MODEL
    def build_model(self, objective, max_depth, learning_rate,
n_estimators, seed):
        self.max_depth = max_depth
        self.learning_rate = learning_rate
        self.n_estimators = n_estimators
```

```
        self.xgb_model = xgb.XGBRegressor(objective=objective,
                                          max_depth=self.max_depth,
                                          learning_rate=self.learning_rate,
                                          n_estimators=self.n_estimators,
                                          random_state=seed)

        # append unique string
        self.unique_string =
f"{self.unique_string}_xgb_{self.max_depth}_{round(self.learning_rate,
3)}_{self.n_estimators}"

    # ------------------------------------------------- WALK FORWARD
VALIDATION
    def walk_forward_validation(self, n_splits=None):

        # if n_splits is not provided, use number of years in training data
as above
        if n_splits is None:
            n_splits = self.n_split

        # split initial training data for training/validation
        # forward walking validation
        tscv = TimeSeriesSplit(n_splits=n_splits)
        eval_scores = list()
        for train_index, val_index in tscv.split(self.X_train_init):
            X_train, X_val = self.X_train_init[train_index],
self.X_train_init[val_index]
            y_train, y_val = self.y_train_init[train_index],
self.y_train_init[val_index]
            # print(f'X_train: {len(X_train)}', f'X_val: {len(X_val)}')

            # ---------------------------------------- FIT & EVALUATE
            self.xgb_model.fit(X_train,
                               y_train.ravel(),
                               eval_set=[(X_val, y_val)],
                               eval_metric=['rmse'],
                               verbose=False)

            # get the last tree eval metrics and append to list
            # square rmse to get mse
            eval_score =
self.xgb_model.evals_result_['validation_0']['rmse'][-1] ** 2
            eval_scores.append(eval_score)

            # average evaluation scores
            return pd.Series(np.mean(eval_scores), index=['mse'])

    def predict_on_test_data(self):
        predictions = self.xgb_model.predict(self.X_test)

        # index was lost when scaling. Get the index from self.X_test_init
        return pd.DataFrame({'true': self.y_test, 'predictions':
predictions}, index=self.X_test_init.index)
```

Test Results: scores and plots

Calculate predicted vs true scores for test data

```
def calc_predictions_avg_test_scores(true, predictions):
```

```
    """
    Calculate test scores between true and predicted returns
    :param true: true returns
    :param predictions: predicted returns
    :return: Pandas Series
    """
    r2_score_modified = 1 - (np.sum((true - predictions) ** 2)) /
(np.sum(true ** 2))
    r2_score_orig = r2_score(true, predictions)
    mse = mean_squared_error(true, predictions)
    sse = np.sum((predictions - true) ** 2)

    return pd.Series([r2_score_modified, r2_score_orig, mse, sse],
                     index=['r2_modified', 'r2_original', 'mse', 'sse'])
```

Plot predicted vs true returns for test data

```
def plot_predictions_avg_stock(df_models_avg_true_best,
df_models_avg_pre_best, df_benchmarks, save_in, unique_string):
    """
    Plot the averaged predicted returns vs true for model, and lewellen per
stock
    :param df_predictions_avg: Pandas DataFrame
    :param save_in: directory to save figure
    :param unique_string: to add to the figure name
    :return:
    """
    start_time = df_models_avg_true_best.index[0]
    end_time = df_models_avg_true_best.index[-1]
    stocks = set(df_models_avg_true_best.columns.get_level_values(1))

    custom_lines = [Line2D([0], [0], color='b', lw=2),
                    Line2D([0], [0], color='r', lw=2)]

    # loop over unique stocks found in the second level
    for stock in stocks:
        fig, ax = plt.subplots(2, 2, figsize=(12, 6))
        fig.suptitle(f'{stock} avg. Predictions', fontsize=12)

        ax[0, 0].plot(df_models_avg_true_best['NN', stock], 'b',
label='Actual')
        ax[0, 0].plot(df_models_avg_pre_best['NN', stock], 'r',
label='Predicted')
        ax[0, 0].set_xticks([ax[0, 0].get_xticks()[0], ax[0,
0].get_xticks()[-1]], minor=False)
        ax[0, 0].set_xticklabels([start_time.strftime("%b %Y"),
end_time.strftime("%b %Y")])
        ax[0, 0].set_title(f"NN, lr={df_models_avg_true_best['NN',
stock].columns.values}", fontsize=10)

        ax[0, 1].plot(df_models_avg_true_best['XGB', stock], 'b',
label='Actual')
        ax[0, 1].plot(df_models_avg_pre_best['XGB', stock], 'r',
label='Predicted')
        ax[0, 1].set_xticks([ax[0, 1].get_xticks()[0], ax[0,
1].get_xticks()[-1]], minor=False)
        ax[0, 1].set_xticklabels([start_time.strftime("%b %Y"),
end_time.strftime("%b %Y")])
        ax[0, 1].set_title(f"XGBoost, lr={df_models_avg_true_best['XGB',
stock].columns.values}", fontsize=10)
```

```
        ax[1, 0].plot(df_benchmarks['lewellen', stock, 'true'], 'b',
label='Actual')
        ax[1, 0].plot(df_benchmarks['lewellen', stock, 'predictions'], 'r',
label='Predicted')
        ax[1, 0].set_xticks([ax[1, 0].get_xticks()[0], ax[1,
0].get_xticks()[-1]], minor=False)
        ax[1, 0].set_xticklabels([start_time.strftime("%b %Y"),
end_time.strftime("%b %Y")])
        ax[1, 0].set_title('Lewellen', fontsize=10)

        ax[1, 1].legend(custom_lines, ['Actual', 'Predicted'],
loc='center')
        ax[1, 1].axis('off')

        plt.savefig(os.path.join(save_in,
f'{unique_string}_predictions_avg_{stock}.png'),
                    orientation='portrait',
                    format='png')
        plt.close()
def plot_predictions_avg_all_stocks(df_models_avg_true_best,
df_models_avg_pre_best, df_benchmarks, save_in, unique_string):
    """
    Plot the averaged predicted returns vs true for model, and lewellen for
all stocks
    :param df_predictions_avg: Pandas DataFrame
    :param save_in: directory to save figure
    :param unique_string: to add to the figure name
    :return:
    """
    start_time = df_models_avg_true_best.index[0]
    end_time = df_models_avg_true_best.index[-1]
    stocks = set(df_models_avg_true_best.columns.get_level_values(1))

    fig, ax = plt.subplots(2, 2, figsize=(12, 6))
    fig.suptitle('Avg. Predictions all Stocks', fontsize=12)

    ax[0, 0].set_title('NN', fontsize=10)
    ax[0, 1].set_title('XGBoost', fontsize=10)
    ax[1, 0].set_title('Lewellen', fontsize=10)

    # loop over unique stocks found in the second level
    for stock in stocks:
        ax[0, 0].plot(df_models_avg_true_best['NN', stock], 'b',
label='Actual')
        ax[0, 0].plot(df_models_avg_pre_best['NN', stock], 'r',
label='Predicted')

        ax[0, 1].plot(df_models_avg_true_best['XGB', stock], 'b',
label='Actual')
        ax[0, 1].plot(df_models_avg_pre_best['XGB', stock], 'r',
label='Predicted')

        ax[1, 0].plot(df_benchmarks['lewellen', stock, 'true'], 'b',
label='Actual')
        ax[1, 0].plot(df_benchmarks['lewellen', stock, 'predictions'], 'r',
label='Predicted')

    ax[0, 0].set_xticks([ax[0, 0].get_xticks()[0], ax[0, 0].get_xticks()[-
1]], minor=False)
```

```
    ax[0, 0].set_xticklabels([start_time.strftime("%b %Y"),
end_time.strftime("%b %Y")])

    ax[1, 0].set_xticks([ax[1, 0].get_xticks()[0], ax[1, 0].get_xticks()[-
1]], minor=False)
    ax[1, 0].set_xticklabels([start_time.strftime("%b %Y"),
end_time.strftime("%b %Y")])

    ax[0, 1].set_xticks([ax[0, 1].get_xticks()[0], ax[0, 1].get_xticks()[-
1]], minor=False)
    ax[0, 1].set_xticklabels([start_time.strftime("%b %Y"),
end_time.strftime("%b %Y")])

    # add fake line to add one legend only
    custom_lines = [Line2D([0], [0], color='b', lw=2),
                    Line2D([0], [0], color='r', lw=2)]

    ax[1, 1].legend(custom_lines, ['Actual', 'Predicted'], loc='center')
    ax[1, 1].axis('off')

    plt.savefig(os.path.join(save_in,
f'{unique_string}_predictions_avg_all_stocks.png'),
                orientation='portrait',
                format='png')
    plt.close()
```

Main script

- Try 2 seeds and 3 learning rates (i.e. 6 combinations)
- For each combination:
    - Fit nn model (lstm1, nn1, nn2, nn3, nn4, nn5)
    - Get walk forward validation evaluation metrics
    - Make predictions on test data
    - Fit XGBoost model
    - Get walk forward validation evaluation metrics
    - Make predictions on test data
    - Make predictions on test data using Lewellen benchmark

```
stocks_csv_dir_name = os.path.join(os.getcwd(), 'data', 'weekly_features')
stocks_csv_files_full_path_list = glob(os.path.join(stocks_csv_dir_name,
'*.csv'))

features = ['stock_features', 'macro_features', 'momentum_features',
'annual_features', 'tensor_product', 'returns']

period = 'weekly'
lookback = 12
step = 1
seeds = [1, 42]

learning_rates = [0.001, 0.005, 0.01]

save_in = os.path.join(os.getcwd(), 'results', f'{period}_features')

# ---------------------------------------------- NN ARGS
model_function = nn5
```

```
model_name = 'nn5'

activation = activations.relu
kernel_regularizer = regularizers.l1(0.001)

batch_size = 32
epochs = 100
unique_string = f'{model_name}_relu_adam_{epochs}'

# ------------------------------------------------ XGB ARGS
objective = 'reg:squarederror'
max_depth = 2
n_estimators = 1000

unique_string = f"{unique_string}_xgb_{max_depth}_{n_estimators}"

# ---------------------------------------------------------------------------
-----------------------------------------
# create empty MultiIndexed DataFrames to save all results for all stocks
df_models = pd.DataFrame(columns=pd.MultiIndex(levels=[[], [], [], [], []],
                                               codes=[[], [], [], [], []],
                                               names=['model', 'stock',
'lr', 'seed', 'true_pre']))
df_benchmarks = pd.DataFrame(columns=pd.MultiIndex(levels=[[], [], []],
                                                   codes=[[], [], []],
                                                   names=['benchmark',
'stock', 'true_pre']))
df_models_eval_metrics = pd.DataFrame(columns=pd.MultiIndex(levels=[[], [],
[]],
                                                            codes=[[], [],
[]],
                                                            names=['data',
'stock', 'lr']))
df_models_avg_best_all_stocks = pd.DataFrame()

# loop over each stock
# for NN and XGB, loop over each learning rate and seed
# for lewellen, no need to loop over learning rates and seeds
# train a new model, get predictions, then save results in the master
DataFrame
for stocks_csv_file_full_path in tqdm(stocks_csv_files_full_path_list):
        file_name = os.path.basename(stocks_csv_file_full_path)
        file_root, _ = os.path.splitext(file_name)
        print(f'\nProcessing {file_name}...')

        # loop over learning rates
        for learning_rate in tqdm(learning_rates):
            print(f'\nlearning rate {learning_rate}...')
            optimizer = optimizers.adam(lr=learning_rate)

            # loop over seeds
            for seed in tqdm(seeds):
                print(f'\nseed {seed}...')

                # ------------------------------------------- NN MODEL ---
-----------------------------------------
                print('\nFitting NN model...')
                # create NN object
                stock_nn = StocksNN(stocks_csv_file_full_path,
                                    period,
                                    lookback,
```

```
                                step,
                                features,
                                'excess_return')
                stock_nn.build_and_compile_model(model_function,
                                             model_name,
                                             activation,
                                             optimizer,
                                             seed,
                                             kernel_regularizer)

                df_models_eval_metrics['NN', f'{file_root}', learning_rate]
= stock_nn.walk_forward_validation(
                    batch_size, epochs)

                df_nn_predictions = stock_nn.predict_on_test_data_nn()
                df_models['NN', f'{file_root}', learning_rate, seed,
'true'] = df_nn_predictions['true']
                df_models['NN', f'{file_root}', learning_rate, seed,
'predictions'] = df_nn_predictions['predictions']

                # ---------------------------------------------- XGB MODEL --
-----------------------------------------
                print('\nFitting XGB model...')
                # create XGB object
                stock_xgb = StocksXGB(stocks_csv_file_full_path,
                                      period,
                                      lookback,
                                      step,
                                      features,
                                      'excess_return')
                stock_xgb.build_model(objective,
                                      max_depth,
                                      learning_rate,
                                      n_estimators,
                                      seed)

                df_models_eval_metrics['XGB', f'{file_root}',
learning_rate] = stock_xgb.walk_forward_validation()

                df_xgb_predictions = stock_xgb.predict_on_test_data()
                df_models['XGB', f'{file_root}', learning_rate, seed,
'true'] = df_xgb_predictions['true']
                df_models['XGB', f'{file_root}', learning_rate, seed,
'predictions'] = df_xgb_predictions[
                    'predictions']

        # -------------------------------------------------- LEWELLEN
        print('Predicting lewellen...')
        df_lewellen_pre = stock_nn.predict_on_test_data_lewellen()
        df_benchmarks['lewellen', f'{file_root}', 'true'] =
df_lewellen_pre['true']
        df_benchmarks['lewellen', f'{file_root}', 'predictions'] =
df_lewellen_pre['predictions']
```

- Average predictions across seeds
- Choose learning rate with the best evaluation scores on the validation set
- Calculate evaluation scores on the test set (using chosen learning rate)
- Save as csv files
- Plot results

```
# --------------------------------------------------- BENCHMARK -----------
-----------------------------------------
# save predictions
df_benchmarks.to_csv(os.path.join(save_in,
f'{unique_string}_benchmarks.csv'))

# calculate and save test scores
df_benchmarks_scores = df_benchmarks.dropna().groupby(level=[0, 1],
axis=1).apply(
    lambda s: calc_predictions_avg_test_scores(s.iloc[:, 0], s.iloc[:, 1]))
df_benchmarks_scores.to_csv(os.path.join(save_in,
f'{unique_string}_benchmarks_scores.csv'))

# ----------------------------------------------------- MODELS ------------
-----------------------------------------
# save each stock predictions
df_models = df_models.dropna(axis=0)
df_models.to_csv(os.path.join(save_in, f'{unique_string}_models.csv'))

# average per stock predictions across all seeds
df_models_avg = df_models.groupby(level=[0, 1, 2, 4], axis=1).mean()
df_models_avg.to_csv(os.path.join(save_in,
f'{unique_string}_models_avg.csv'))

# calculate test scores per stock averaged predictions
df_models_avg_scores = df_models_avg.groupby(level=[0, 1, 2],
axis=1).apply(
    lambda s: calc_predictions_avg_test_scores(s.iloc[:, 0], s.iloc[:, 1]))
df_models_avg_scores.to_csv(os.path.join(save_in,
f'{unique_string}_models_avg_scores.csv'))

# select best learning rate per stock (minimum averaged 'mse' on validation
set)
df_models_eval_metrics_best = df_models_eval_metrics[
    df_models_eval_metrics.groupby(level=[0, 1],
axis=1).idxmin().loc['mse']]
df_models_eval_metrics_best.to_csv(os.path.join(save_in,
f'{unique_string}_models_eval_metrics_best.csv'))

# select metrics for best learning rate per stock (using previous best
learning rates on validation)
df_models_avg_scores_best =
df_models_avg_scores[df_models_eval_metrics_best.columns]
df_models_avg_scores_best.to_csv(os.path.join(save_in,
f'{unique_string}_models_avg_scores_best.csv'))

# select true/pre values for best learning rate per stock (using previous
best learning rates on validation)
# separate true from predictions to get rid of the forth level
df_models_avg_true = df_models_avg.loc[:, (slice(None), slice(None),
slice(None), 'true')]
df_models_avg_true.columns = df_models_avg_true.columns.droplevel(3)
df_models_avg_true_best =
df_models_avg_true[df_models_eval_metrics_best.columns]
df_models_avg_true_best.to_csv(os.path.join(save_in,
f'{unique_string}_models_avg_true_best.csv'))

df_models_avg_pre = df_models_avg.loc[:, (slice(None), slice(None),
slice(None), 'predictions')]
df_models_avg_pre.columns = df_models_avg_pre.columns.droplevel(3)
```

```
df_models_avg_pre_best =
df_models_avg_pre[df_models_eval_metrics_best.columns]
df_models_avg_pre_best.to_csv(os.path.join(save_in,
f'{unique_string}_models_avg_pre_best.csv'))

# plot per stock
plot_predictions_avg_stock(df_models_avg_true_best,
                           df_models_avg_pre_best,
                           df_benchmarks,
                           save_in,
                           unique_string)
# plot all stocks
plot_predictions_avg_all_stocks(df_models_avg_true_best,
                           df_models_avg_pre_best,
                           df_benchmarks,
                           save_in,
                           unique_string)

# concatenate averaged predictions for all stocks vertically
# get rid of the third level
df_models_avg_true_best.columns =
df_models_avg_true_best.columns.droplevel(2)
df_models_avg_true_best_all_stocks =
df_models_avg_true_best.groupby(level=[0], axis=1).apply(
    lambda df: df.values.flatten())

df_models_avg_pre_best.columns =
df_models_avg_pre_best.columns.droplevel(2)
df_models_avg_pre_best_all_stocks =
df_models_avg_pre_best.groupby(level=[0], axis=1).apply(
    lambda df: df.values.flatten())

df_models_avg_best_all_stocks['NN'] = calc_predictions_avg_test_scores(
    df_models_avg_true_best_all_stocks.loc['NN'],
    df_models_avg_pre_best_all_stocks.loc['NN'])

df_models_avg_best_all_stocks['XGB'] = calc_predictions_avg_test_scores(
    df_models_avg_true_best_all_stocks.loc['XGB'],
    df_models_avg_pre_best_all_stocks.loc['XGB'])

df_benchmarks_all_stocks = df_benchmarks.groupby(level=[0, 2],
axis=1).apply(
    lambda df: df.values.flatten())

df_models_avg_best_all_stocks['lewellen'] =
calc_predictions_avg_test_scores(
    df_benchmarks_all_stocks.loc['lewellen', 'true'],
    df_benchmarks_all_stocks.loc['lewellen', 'predictions'])

df_models_avg_best_all_stocks.to_csv(os.path.join(save_in,
f'{unique_string}_models_avg_best_all_stocks.csv'))
```