University of Auckland
Department of Accounting & Finance

FINANCE 788: Research Essay

Author: Connor McDowall
Supervisor: Dr Paul Geertsema

December 31, 2021

Abstract

Acknowledgements

Paul Geertsema

Declaration of Contribution

Contents

List of Figures

List of Tables

List of Equations

1 Introduction

2 Literature Review

Insert Literature Review - Very Brief, Only Double Spaced

2.1 History of Asset Pricing Theory

2.1.1 Optimisation Methodologies

Convexity is an important concept in optimisation

2.1.2 Machine Learning in Financial Contexts

A couple of recent publications highlight the increased application of machine learning algorithms in financial contexts. **corporate-culture** Gu et al (**eapvml**) explore the comparative use of machine learning in empirical asset pricing.

3 Research Intent

Insert Research Intent

4 Theory

4.1 Return predictability

Return predictability underlies asset pricing theory. **Insert**

4.2 Modelling, Loss, and Optimisation

We summarize the theory surrounding predictive modelling, loss functions, and optimisation algorithms. These functions train models by comparing predictions to realized observations using optimisation algorithms to minimize the loss function. We examine a linear model as our predictive model (**??**). Mean square error (**??**) and Gradient Descent (GD) are basic examples of a loss function and optimisation algorithm, respectively.

$$\hat{y} = mx_i + b \tag{1}$$

$$f(y,(mx_i + b)) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2 \tag{2}$$

Firstly, gradient descent takes the partial derivatives of the loss function, with the respect to the parameters in our predictive model. In our example, equations **??** and **??** are the partial derivatives for the mean square error loss function.

$$\frac{\partial f(y,(mx_i + b)}{\partial m} = \frac{1}{n}\sum_{i=1}^{n} -2x_i(y_i - (mx_i + b))^2 \tag{3}$$

$$\frac{\partial f(y,(mx_i + b)}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} -2(y_i - (mx_i + b))^2 \tag{4}$$

Secondly, the algorithm explores epochs, using a learning rate to update parameters to move in the opposite directions of the partial derivatives until settling in a local minima. This extrema is the optimisation of the loss function, quantifying the accuracy of the predicative model. Ordinary Least Squares (OLS) regressions is an extension of the linear model prevalent in asset pricing.

4.3 Ordinary Least Squares (OLS)

The OLS regression is the most prominent statistical model in asset pricing theory. Rosenfeld (**olsmf**) summarises OLS. The composition of the true OLS (**??**) model includes four components. Firstly, **X**, an n x k matrix of k independent variables for n observations. Secondly, **y**, an n x 1 vector of observation on the dependent variable. Thirdly, ϵ, an n x 1 vector of unexplained error. Lastly, θ, a k x 1 vector of parameters to be estimated.

$$y = X\theta + \epsilon \tag{5}$$

4.3.1 Estimation Criteria

The criteria to obtain the parameter estimate ($\hat{\theta}$) relies on the minimisation of the sum of squared residuals (**??**). We highlight the observed residuals (e) are distinct from unexplained disturbances (ϵ). Equation **??** derives residuals by taking the difference between observations based on parameter estimates.

$$\sum e_i^2 \tag{6}$$

$$e = y - X\hat{\theta} \tag{7}$$

Expanding the quadratic $e^T e$ after substituting in equation **??** leads to the alternative expression of the sum of squared residuals in equation **??**. Minimizing the sum of square residuals requires taking the partial derivative of equation **??** with respect to the estimated parameters (equation) using matrix differentiation (**??**). It is imperative X has full rank where all vectors in the matrix are linearly independent, validating both the presence of a positive definite matrix and minimum.

$$e^T e = y^T y - 2\hat{\theta}^T X^T y + \hat{\theta}^T X^T \hat{\theta} X \tag{8}$$

$$\frac{\partial e^T e}{\partial \hat{\theta}} = -2X^T y + 2X^T X\hat{\theta} = 0 \tag{9}$$

We find the expression for the Ordinary Least Squares (OLS) estimator (**??**) after rearranging equation **??** to normal form, utilizing inverse matrices to form identity matrices, and simplifying.

$$\begin{aligned} 2X^T X\hat{\theta} &= 2X^T y \\ (X^T X)^{-1}(X^T X)\hat{\theta} &= (X^T X)^{-1} X^T y \\ I\hat{\theta} &= (X^T X)^{-1} X^T y \\ \hat{\theta} &= (X^T X)^{-1}(X^T y) \end{aligned} \tag{10}$$

Therefore, we can use the OLS estimator to make predictions with OLS (**??**).

$$\hat{y} = X^T \hat{\theta} \tag{11}$$

4.3.2 Properties of OLS Estimators

There are six key properties in addition to the satisfaction in minimizing the summation of squared residuals.

1. The residuals are uncorrelated with the observed values of X i.e., $X^T e = 0$.
2. The sum of the residuals is zero i.e., $\sum e_i = 0$.
3. The sample mean of the residuals is zero i.e., $\bar{e} = \frac{\sum e_i}{n} = 0$.
4. The regression hyperplane passes through the means of observed values i.e., $\underset{=}{e} \frac{y - X\theta}{n} = 0$. Since $\bar{e} = 0$ assumed, it is implied $\bar{y} = \bar{x}\bar{\theta}$.
5. The residuals are uncorrelated with the predicted y i.e., $\hat{y} = X\hat{\theta}$, $\hat{y}^T e = (X\hat{\beta})^T e = b^T X^T e = 0$
6. The mean of $\hat{y}$ for the sample will equal the mean of the y.

4.3.3 The Gauss-Markov Theorem

However, OLS makes Gauss-Markov assumptions about the true model to make inferences regarding β from $\hat{\beta}$. The intention of the Gauss-Markov Theorem, conditional on the below assumptions, states the

OLS estimator is the best linear, unbiased, and efficient estimator:

$$
\begin{aligned}
y &= x\beta + \epsilon \\
E[\epsilon|X] &= 0 \qquad (12) \\
E(\epsilon\epsilon^T|X) &= \Omega = \sigma^2 I \qquad (13) \\
\epsilon|X\ & N[0, \sigma^T I] \text{ (hypothesis testing)}
\end{aligned}
$$

- X is an n x k matrix of full rank
- X must be generated randomly, or fixed, by a mechanism uncorrelated to disturbances.

Equation **??** implies $E(y) = X\beta$ as no observations of the independent variables convey any information about the expected values of the disturbances. Equation **??** captures homoskedasticity and no autocorrelation assumptions. Additionally, The theory underlying Ordinary Least Squares informs the common practice in minimising of the sum of least squares when evaluating prediction performance. The mathematical tractability, in accordance with the aforementioned assumption, frame our thinking surrounding the derivation of custom loss functions.

4.3.4 Research Intent

Minimisation of returns **Include examples on the minimisation of sum of the square errors does not contribute to maximising returns**

5 Data

Hou et al., (**hou2020replicating**) use an extensive data library to assess 452 anomalies across anomalies literature. Their analysis informs which abnormalities drive the cross section of expected returns. Most abnormalities fail under current standards of empirical finance when using a single hurdle test of absolute t-stat greater or equal to 1.96. Firstly, the paper finds economic fundamentals take precedence over trading frictions in explanatory power, statistical and economic significance. Secondly, micro-caps account for anomalies disproportionately, leading to NYSE breakpoints, value-weighted returns in both portfolio sorts and cross-sectional regressions with weighted least squares. Lastly, arguments in improving anomalies literature credibility follow a closer alignment to economic theory as the field persists to be statistical in nature. Overall, capital market efficiency is higher than expected. Jensen et al., **jensen2021there** use the above dataset to explore hierarchial bayesian models of alphas emphasising the joint behaviours of factors, and provide an alternative multiple testing adjustment, more powerful than common methods. Jensen et al., adapt the global dataset to focus only on one-month holding periods for all factors, only include most recent accounting data (quarterly or annually) and add 15 new factors. The exhaustive nature and accessibility of the global dataset makes it well-suited for exploring optimisation functions in neural-network construction.

5.1 Limitations

5.2 Summary Statistics

6 Methodology

6.1 Target Variable

6.2 Google Cloud Platform

6.3 Artificial Neural Networks

6.3.1

6.3.2 Configuration

6.3.3 Limitations

6.4 Tensorflow

6.4.1 Automatic Differientiation

6.5 Loss Functions & Performance Metrics

Table ?? emphasises the separation between training and validation datasets.

Variable	Description	$MSE(y,\hat{y})$	$HP(y,\hat{y})$
θ	Estimation Training	$\hat{\theta}_{MSE}$	$\hat{\theta}_{HP}$
λ	Validation	$\hat{\lambda}_{MSE}$	$\hat{\lambda}_{HP}$

Table 1: Objective (MSE: Mean Square Error, HP: Hedge Portfolio)

6.5.1 Mean Square Error (MSE)

Section ?? outlines advantages to Ordinary Least Squares. Subsequently, MSE serves as a baseline for loss function and performance metric comparisons. The following function (??) and partial derivative (??) describe Tensorflows's Mean Square Error implementation, both from in-built and custom contexts. Python classes describe equation ?? to enable Tensorflow's automatic differentiation capabilities, approximating the partial derivatives of the loss function (??) with numerical methods. Please note the use of Hadamard exponentiation ($x^{\circ n}$) as an element-wise operation.

$$f(y, X^T\hat{\theta}) = \frac{\vec{1}}{\vec{1}^T\vec{1}}(y - X^T\hat{\theta})^{\circ 2} \tag{14}$$

$$\frac{\partial f(y, X^T\hat{\theta})}{\partial\hat{\theta}} = \frac{\vec{1}}{\vec{1}^T\vec{1}}(-2(y - XT\hat{\theta})^{\circ 1}) \tag{15}$$

6.5.2 Hedge Portfolio

Hedge portfolios rely on monotonic ranking functions for optimisation as their monotonic nature preserves or reverses a given ordered set. The analysis cross-section of one-month lead portfolio excess returns using monotonic functions

$$R(y_{i,t}) \tag{16}$$

The ranking function (R($y_{i,t}$)) and thresholds (u,v) form subsets of long and short portfolios. Long (L) or Short (S) sets include excess returns conditioned on the associated monotonic ranking given a threshold, bound by the cardinality of the excess return vector ($|y|$). The subsequent truth sets mathematically express aforementioned time-series hedge portfolios.

$$
\begin{aligned}
&L = \{y_{i,t} | R(y_{i,t}) \leq u\} \\
&S = \{y_{i,t} | R(y_{i,t}) \geq v\} \\
&0 < u \leq |y| \\
&0 < v \leq |y| \\
&u < v
\end{aligned}
$$

Equation **??** describes hedge portfolio lead excess returns (H_t) at a given time (t).

$$H_t = \frac{1}{|L|} \sum_{i \epsilon L} y_{i,t} - \frac{1}{|S|} \sum_{i \epsilon S} y_{i,t} \tag{17}$$

Figure **??**) illustrates an approximate linear monotonic ranking function with a sample of 100 uniformly distributed excess returns between -10% and 10%. Boundary conditions u and v are set to 20 and 80 , respectively. Subsequently, excess returns above (below) the green (blue) dotted line belong to the long (L) (short (S)) set.

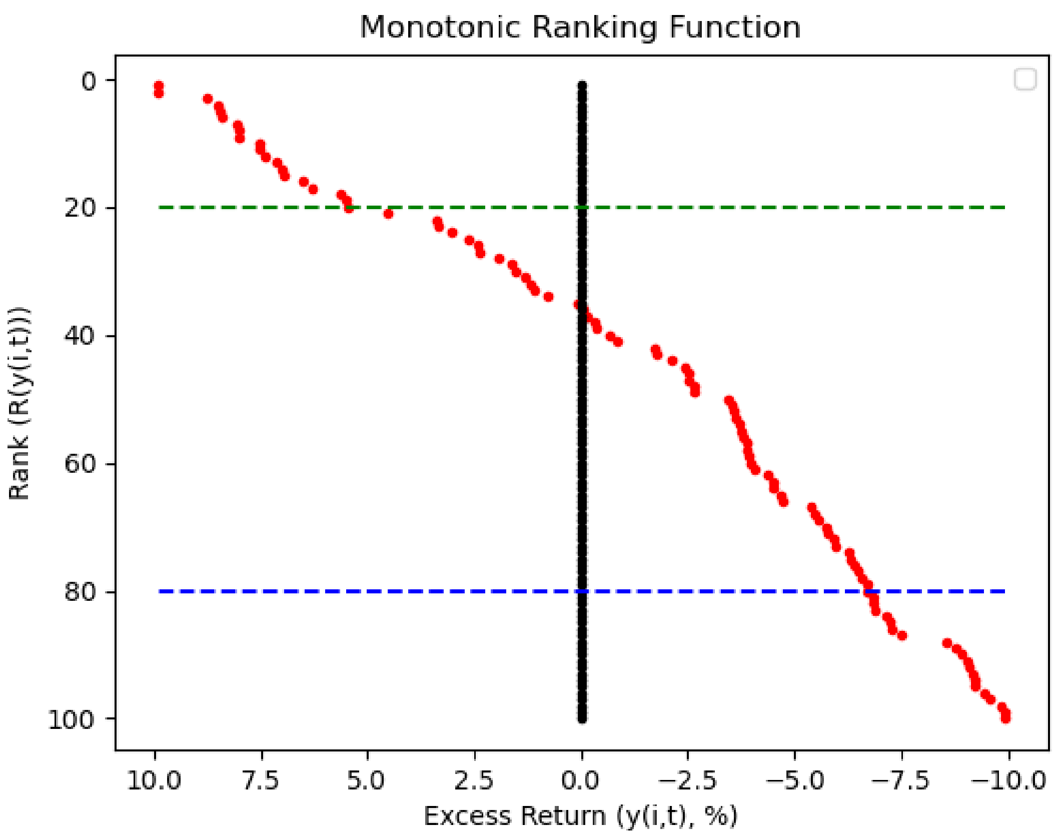

Figure 1: Approximate Linear Monotonic Ranking Function

The permutations in monotonic ranking functions, and subsequent hedge portfolios, are endless. This research essay develops a monotonic ranking function proportionally weighting one month lead excess returns (**??**). Therefore, equation **??** defines the loss function.

$$R(\hat{y}) = W \tag{18}$$

$$W := \frac{\hat{y}}{\vec{\mathbf{1}}\hat{y}}$$

$$\hat{y} = X^T\hat{\theta}$$

$$f_{\hat{\theta}}(X) = (\frac{X^T\hat{\theta}}{\vec{\mathbf{1}}X^T\hat{\theta}})^\top X^T\hat{\theta} \tag{19}$$

The above loss function is differentiable using symbolic mathematic as shown in equation **??**.

$$\frac{\partial f_{\hat{\theta}}(X)}{\partial\hat{\theta}} = \frac{\partial((\frac{X^T\hat{\theta}}{\vec{\mathbf{1}}X^T\hat{\theta}})^\top X^T\hat{\theta})}{\partial\hat{\theta}}$$

$$\frac{\partial(f_{\hat{\theta}}(X))}{\partial\hat{\theta}} = \frac{1}{(\hat{\theta}^\top X\vec{1})}XX^\top\hat{\theta} + \frac{1}{\vec{1}X^\top\hat{\theta}}XX^\top\hat{\theta} - \frac{1}{(\hat{\theta}^\top X\vec{1})^2}\hat{\theta}^\top XX^\top\hat{\theta}X\vec{1} \tag{20}$$

Our research Subsection **??** explains the theory supporting loss minimisation. Applying gradient descent methods to the product of the loss function and scaler of -1 transforms the minimisation to maximisation. This transformation leads to finding the argmax of maximisation function with respect to $\hat{\theta}$ (**??**). The aforementioned transformation is simply and suitable for exploration in the context of the research intent. More sophisticated methods exist for maximisation such as reinforcement learning (**??**).

$$\underset{\hat{\theta}}{\text{argmax}} : (\frac{X^T\hat{\theta}}{\vec{\mathbf{1}}X^T\hat{\theta}})^\top X^T\hat{\theta} \tag{21}$$

Conventional asset pricing methodologies persist in academic literature. The main contribution Hedge Portfolio Mean

The Capital Asset Pricing Model (CAPM)

Fama-French Three Factor Model (FF3)

Fama-French Five Factor Model (FF5) continues to inform asset pricing E. Fama & K. French produce **fama2004capital**

6.5.3 Sharpe Ratio

6.5.4 Information Ratio

6.6 Reinforcement learning

6.6.1 Dynamic Programming

6.6.2 Bellman's Algorithm

6.6.3 Q-Learning

7 Results

8 Discussion

9 Contributions

10 Conclusion

11 Appendix

11.1 Tables and Charts

	count	mean	std	min	25%	50%	75%	max
permno	2739928.0	5.405281e+04	2.782267e+04	10000.0000	2.651800e+04	5.715400e+04	8.018600e+04	9.343600e+04
permco	2739928.0	1.843974e+04	1.402881e+04	3.0000	7.702000e+03	1.640850e+04	2.321000e+04	5.766700e+04
crsp_shrcd	2739928.0	1.089520e+01	4.571000e-01	10.0000	1.100000e+01	1.100000e+01	1.100000e+01	1.200000e+01
crsp_exchcd	2739928.0	2.127400e+00	9.343000e-01	1.0000	1.000000e+00	3.000000e+00	3.000000e+00	3.000000e+00
sic	2692217.0	4.605936e+03	1.921398e+03	100.0000	3.271000e+03	4.011000e+03	6.036000e+03	9.999000e+03
ff49	2674304.0	3.037380e+01	1.341740e+01	1.0000	1.800000e+01	3.400000e+01	4.300000e+01	4.900000e+01
adjfct	2739928.0	2.838700e+00	1.267170e+01	0.0000	1.000000e+00	1.000000e+00	2.000000e+00	1.215000e+03
shares	2739928.0	6.078630e+01	2.852566e+02	0.0830	4.399000e+00	1.251900e+01	3.808200e+01	2.920640e+04
me	2739928.0	2.241254e+03	1.473073e+04	1.1708	4.367020e+01	1.565628e+02	7.167608e+02	2.255969e+06
me_company	2739928.0	2.283180e+03	1.527340e+04	1.1708	4.387450e+01	1.574086e+02	7.211363e+02	2.255969e+06
prc	2739928.0	2.876220e+01	6.488772e+02	0.0078	7.875000e+00	1.612500e+01	2.912500e+01	1.416000e+05
prc_local	2739928.0	2.876220e+01	6.488772e+02	0.0078	7.875000e+00	1.612500e+01	2.912500e+01	1.416000e+05
dolvol	2580622.0	3.282292e+08	2.520900e+09	0.0000	1.070786e+06	7.165154e+06	7.076108e+07	8.441730e+11
ret	2719460.0	1.640000e-02	1.672000e-01	-1.0000	-5.880000e-02	4.100000e-03	7.410000e-02	2.400000e+01
ret_local	2719460.0	1.640000e-02	1.672000e-01	-1.0000	-5.880000e-02	4.100000e-03	7.410000e-02	2.400000e+01
ret_exc	2719460.0	1.270000e-02	1.673000e-01	-1.0068	-6.250000e-02	7.000000e-04	7.060000e-02	2.399690e+01
ret_lag_dif	2739928.0	1.000000e+00	0.000000e+00	1.0000	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
ret_exc_lead1m	2732542.0	6.400000e-03	1.559000e-01	-1.0113	-6.560000e-02	-1.800000e-03	6.710000e-02	1.988170e+01
market_equity_rank_x	2739928.0	5.982920e+01	2.380660e+01	1.0000	4.000000e+01	6.000000e+01	8.000000e+01	9.950000e+01
enterprise_value_rank_x	2480615.0	5.845440e+01	2.501660e+01	1.0000	3.800000e+01	5.900000e+01	8.000000e+01	9.950000e+01
book_equity_rank_x	2452453.0	5.800700e+01	2.593820e+01	1.0000	3.800000e+01	5.900000e+01	8.000000e+01	9.950000e+01
assets_rank_x	2522907.0	5.751850e+01	2.635510e+01	1.0000	3.700000e+01	5.900000e+01	8.000000e+01	9.950000e+01
sales_rank_x	2509790.0	5.691950e+01	2.717080e+01	1.0000	3.600000e+01	5.900000e+01	8.000000e+01	9.950000e+01
net_income_rank_x	2517298.0	5.581200e+01	2.878360e+01	1.0000	3.300000e+01	6.000000e+01	8.000000e+01	9.950000e+01
bidask_x	2739928.0	1.289000e-01	3.351000e-01	0.0000	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
prc_high_x	2355383.0	2.540480e+01	2.608370e+01	0.1790	9.250000e+00	1.850000e+01	3.300000e+01	4.617600e+02
prc_low_x	2365005.0	2.211970e+01	2.325750e+01	0.0818	7.640000e+00	1.600000e+01	2.880000e+01	4.175300e+02
tvol_x	2580622.0	8.316484e+06	2.941295e+07	0.0000	9.875000e+04	5.510000e+05	3.923700e+06	6.485186e+08
div1m_me_x	2718102.0	1.300000e-03	3.700000e-03	0.0000	0.000000e+00	0.000000e+00	0.000000e+00	9.010000e-02
div3m_me_x	2718121.0	4.000000e-03	6.000000e-03	0.0000	0.000000e+00	0.000000e+00	6.700000e-03	1.164000e-01
div6m_me_x	2660395.0	8.100000e-03	1.170000e-02	0.0000	0.000000e+00	0.000000e+00	1.360000e-02	1.472000e-01
div12m_me_x	2548844.0	1.670000e-02	2.350000e-02	0.0000	0.000000e+00	3.800000e-03	2.780000e-02	4.015000e-01

	count	mean	std	min	25%	50%	75%	max
chcsho_1m_x	2720001.0	3.200000e-03	2.550000e-02	-0.1168	0.000000e+00	0.000000e+00	0.000000e+00	1.096800e+00
chcsho_3m_x	2681179.0	1.240000e-02	6.180000e-02	-0.1424	0.000000e+00	0.000000e+00	3.300000e-03	1.686700e+00
chcsho_6m_x	2624125.0	2.810000e-02	1.189000e-01	-0.1880	0.000000e+00	9.000000e-04	1.070000e-02	3.832600e+00
chcsho_12m_x	2514147.0	6.190000e-02	2.297000e-01	-0.2696	0.000000e+00	4.700000e-03	3.390000e-02	8.477000e+00
eqnpo_1m_x	2718435.0	-1.500000e-03	2.310000e-02	-0.6801	-0.000000e+00	0.000000e+00	0.000000e+00	1.263000e-01
eqnpo_3m_x	2677912.0	-6.200000e-03	5.200000e-02	-0.9973	-1.800000e-03	0.000000e+00	8.000000e-03	1.696000e-01
eqnpo_6m_x	2618619.0	-1.350000e-02	8.900000e-02	-1.5754	-7.400000e-03	0.000000e+00	1.640000e-02	2.788000e-01
eqnpo_12m_x	2504936.0	-2.670000e-02	1.474000e-01	-2.2489	-2.450000e-02	0.000000e+00	3.340000e-02	4.743000e-01
ret_1_0_x	2541516.0	1.490000e-02	1.481000e-01	-0.7242	-6.120000e-02	7.900000e-03	7.690000e-02	2.176500e+00
ret_2_0_x	2521767.0	2.960000e-02	2.125000e-01	-0.8327	-8.110000e-02	1.480000e-02	1.176000e-01	3.342500e+00
ret_3_0_x	2503682.0	4.400000e-02	2.649000e-01	-0.8864	-9.610000e-02	2.270000e-02	1.506000e-01	5.000000e+00
ret_3_1_x	2502019.0	2.870000e-02	2.108000e-01	-0.8310	-8.140000e-02	1.440000e-02	1.167000e-01	3.342500e+00
ret_6_0_x	2447794.0	8.830000e-02	3.970000e-01	-0.9396	-1.267000e-01	4.500000e-02	2.336000e-01	8.555600e+00
ret_6_1_x	2446030.0	7.230000e-02	3.553000e-01	-0.9171	-1.184000e-01	3.700000e-02	2.059000e-01	8.411800e+00
ret_9_0_x	2393988.0	1.336000e-01	5.093000e-01	-0.9721	-1.466000e-01	6.750000e-02	3.069000e-01	9.857100e+00
ret_9_1_x	2392087.0	1.168000e-01	4.700000e-01	-0.9555	-1.414000e-01	5.930000e-02	2.812000e-01	9.273700e+00
ret_12_0_x	2341375.0	1.813000e-01	6.179000e-01	-0.9783	-1.593000e-01	9.080000e-02	3.773000e-01	1.301590e+01
ret_12_1_x	2339380.0	1.635000e-01	5.789000e-01	-0.9728	-1.558000e-01	8.200000e-02	3.514000e-01	1.223080e+01
ret_12_7_x	2337747.0	7.050000e-02	3.478000e-01	-0.9055	-1.163000e-01	3.610000e-02	2.015000e-01	8.509400e+00
ret_18_1_x	2239551.0	2.625000e-01	7.812000e-01	-0.9850	-1.710000e-01	1.321000e-01	4.926000e-01	2.048480e+01
ret_24_1_x	2145964.0	3.596000e-01	9.260000e-01	-0.9890	-1.717000e-01	1.837000e-01	6.267000e-01	1.484620e+01
ret_24_12_x	2142652.0	1.821000e-01	6.037000e-01	-0.9678	-1.493000e-01	9.260000e-02	3.714000e-01	1.345160e+01
ret_36_1_x	1976435.0	5.673000e-01	1.234400e+00	-0.9935	-1.548000e-01	2.964000e-01	8.916000e-01	1.914000e+01
ret_36_12_x	1972590.0	3.838000e-01	9.482000e-01	-0.9864	-1.546000e-01	2.006000e-01	6.490000e-01	1.702520e+01
ret_48_12_x	1821582.0	5.938000e-01	1.256400e+00	-0.9918	-1.358000e-01	3.161000e-01	9.172000e-01	1.811810e+01
ret_48_1_x	1826053.0	7.976000e-01	1.577300e+00	-0.9965	-1.285000e-01	4.175000e-01	1.176300e+00	1.772000e+01
ret_60_1_x	1691563.0	1.064400e+00	2.014800e+00	-0.9985	-9.170000e-02	5.486000e-01	1.492300e+00	2.754720e+01
ret_60_12_x	1686573.0	8.258000e-01	1.611700e+00	-0.9960	-1.096000e-01	4.364000e-01	1.200000e+00	2.063640e+01
ret_60_36_x	1680619.0	3.857000e-01	9.340000e-01	-0.9860	-1.429000e-01	2.072000e-01	6.479000e-01	1.808570e+01
seas_1_1an_x	2426517.0	1.420000e-02	1.421000e-01	-0.6705	-6.040000e-02	7.600000e-03	7.560000e-02	1.823500e+00
seas_1_1na_x	1870192.0	1.490000e-02	4.360000e-02	-0.2355	-7.800000e-03	1.280000e-02	3.460000e-02	3.871000e-01
seas_2_5an_x	1599992.0	1.520000e-02	6.790000e-02	-0.2970	-2.260000e-02	1.180000e-02	4.810000e-02	6.337000e-01
at_gr1_x	2426455.0	2.641000e-01	9.239000e-01	-0.7398	4.800000e-03	9.050000e-02	2.391000e-01	3.163840e+01
ca_gr1_x	2184566.0	3.206000e-01	1.336600e+00	-0.8313	-3.830000e-02	9.400000e-02	2.815000e-01	4.636900e+01
nca_gr1_x	2183067.0	3.950000e-01	1.682300e+00	-0.8737	-1.530000e-02	8.250000e-02	2.844000e-01	5.781320e+01
lt_gr1_x	2408077.0	3.042000e-01	9.791000e-01	-0.8021	-2.990000e-02	8.560000e-02	2.894000e-01	1.783760e+01

	count	mean	std	min	25%	50%	75%	max
cl_gr1_x	2190296.0	2.996000e-01	8.898000e-01	-0.8494	-6.490000e-02	1.114000e-01	3.701000e-01	1.634630e+01
ncl_gr1_x	2075342.0	9.926000e-01	5.509500e+00	-1.0000	-1.023000e-01	3.970000e-02	3.376000e-01	1.990000e+02
be_gr1_x	2311345.0	3.178000e-01	1.301000e+00	-0.9166	5.900000e-03	9.660000e-02	2.271000e-01	3.373330e+01
debt_gr1_x	2158693.0	7.838000e-01	4.707200e+00	-1.0000	-1.456000e-01	1.900000e-02	3.292000e-01	1.090000e+02
sale_gr1_x	2362404.0	2.228000e-01	6.711000e-01	-0.9960	5.000000e-03	1.032000e-01	2.478000e-01	1.370570e+01
cogs_gr1_x	2358805.0	2.142000e-01	6.122000e-01	-0.9619	-4.700000e-03	1.032000e-01	2.613000e-01	1.190030e+01
sga_gr1_x	1997437.0	1.844000e-01	3.963000e-01	-1.0000	1.340000e-02	1.044000e-01	2.389000e-01	6.765800e+00
opex_gr1_x	2387208.0	1.949000e-01	4.470000e-01	-0.7668	7.900000e-03	1.058000e-01	2.505000e-01	7.187400e+00
capx_gr1_x	2147147.0	6.016000e-01	2.183000e+00	-1.3370	-2.236000e-01	1.144000e-01	6.251000e-01	3.425000e+01
inv_gr1_x	1910333.0	2.595000e-01	9.931000e-01	-1.0000	-6.850000e-02	8.260000e-02	2.909000e-01	1.698080e+01
at_gr3_x	2114339.0	9.104000e-01	2.670800e+00	-0.8797	8.870000e-02	3.426000e-01	8.167000e-01	6.899070e+01
ca_gr3_x	1898998.0	9.832000e-01	3.187300e+00	-0.9099	2.890000e-02	3.230000e-01	8.289000e-01	7.748590e+01
nca_gr3_x	1897746.0	1.592100e+00	6.786800e+00	-0.9628	4.280000e-02	3.455000e-01	1.005000e+00	1.792615e+02
lt_gr3_x	2091277.0	1.135900e+00	3.376000e+00	-0.8936	3.580000e-02	3.474000e-01	9.457000e-01	5.633890e+01
cl_gr3_x	1906078.0	9.845000e-01	2.656400e+00	-0.9194	9.000000e-03	3.652000e-01	9.754000e-01	4.535460e+01
ncl_gr3_x	1803330.0	4.168200e+00	2.242620e+01	-1.0000	-1.231000e-01	2.914000e-01	1.285200e+00	8.323333e+02
be_gr3_x	1998122.0	1.009400e+00	3.275200e+00	-0.9384	7.210000e-02	3.326000e-01	7.902000e-01	6.699660e+01
debt_gr3_x	1882647.0	3.622500e+00	2.086590e+01	-1.0000	-2.165000e-01	2.251000e-01	1.145100e+00	4.310000e+02
sale_gr3_x	2063618.0	8.605000e-01	2.814400e+00	-1.0000	7.210000e-02	3.286000e-01	7.527000e-01	8.620390e+01
cogs_gr3_x	2052669.0	7.935000e-01	2.179500e+00	-1.0000	4.870000e-02	3.267000e-01	7.894000e-01	4.537560e+01
sga_gr3_x	1713690.0	6.540000e-01	1.324200e+00	-1.0000	9.470000e-02	3.366000e-01	7.294000e-01	2.400000e+01
opex_gr3_x	2073541.0	7.171000e-01	1.625000e+00	-0.8979	7.650000e-02	3.367000e-01	7.689000e-01	2.833740e+01
capx_gr3_x	1846897.0	1.692700e+00	5.902400e+00	-1.2088	-2.368000e-01	3.214000e-01	1.355700e+00	1.128462e+02
cash_gr1a_x	2396920.0	1.480000e-02	1.380000e-01	-1.1898	-1.600000e-02	2.800000e-03	3.520000e-02	8.303000e-01
inv_gr1a_x	2351255.0	1.250000e-02	5.090000e-02	-0.3723	-7.000000e-04	7.000000e-04	2.250000e-02	2.978000e-01
rec_gr1a_x	2363716.0	2.190000e-02	6.430000e-02	-0.4405	-2.700000e-03	1.190000e-02	4.270000e-02	3.340000e-01
ppeg_gr1a_x	2178200.0	5.240000e-02	1.039000e-01	-0.8431	8.900000e-03	3.670000e-02	8.330000e-02	5.756000e-01
lti_gr1a_x	2205853.0	5.400000e-03	4.060000e-02	-0.4964	0.000000e+00	0.000000e+00	1.100000e-03	3.478000e-01
intan_gr1a_x	2110874.0	1.080000e-02	6.690000e-02	-0.9608	-7.000000e-04	0.000000e+00	1.700000e-03	5.336000e-01
debtst_gr1a_x	2395084.0	3.900000e-03	6.220000e-02	-0.5236	-5.000000e-03	0.000000e+00	1.320000e-02	4.847000e-01
ap_gr1a_x	2267822.0	1.460000e-02	4.890000e-02	-0.2766	-3.900000e-03	6.100000e-03	2.540000e-02	2.945000e-01
txp_gr1a_x	2057276.0	9.000000e-04	1.130000e-02	-0.0902	-9.000000e-04	0.000000e+00	2.200000e-03	9.250000e-02
debtlt_gr1a_x	2411829.0	1.770000e-02	9.970000e-02	-0.6085	-1.080000e-02	0.000000e+00	3.540000e-02	5.760000e-01
txditc_gr1a_x	2135161.0	2.300000e-03	1.280000e-02	-0.1302	0.000000e+00	0.000000e+00	4.800000e-03	8.330000e-02
coa_gr1a_x	2167569.0	3.450000e-02	1.005000e-01	-0.7908	-4.200000e-03	2.200000e-02	7.140000e-02	4.923000e-01
col_gr1a_x	2191221.0	1.980000e-02	6.480000e-02	-0.4855	-5.500000e-03	1.350000e-02	4.240000e-02	3.834000e-01

	count	mean	std	min	25%	50%	75%	max
cowc_gr1a_x	2146736.0	1.440000e-02	8.680000e-02	-0.6052	-1.810000e-02	9.000000e-03	4.750000e-02	4.185000e-01
ncoa_gr1a_x	2185140.0	4.890000e-02	1.438000e-01	-1.8841	-5.500000e-03	2.970000e-02	9.040000e-02	7.494000e-01
ncol_gr1a_x	2174709.0	6.300000e-03	3.310000e-02	-0.3605	-1.100000e-03	1.900000e-03	1.180000e-02	3.338000e-01
nncoa_gr1a_x	2147813.0	4.270000e-02	1.424000e-01	-1.8841	-9.700000e-03	2.500000e-02	8.290000e-02	7.692000e-01
oa_gr1a_x	2167557.0	8.310000e-02	2.025000e-01	-2.5884	-3.400000e-03	6.800000e-02	1.668000e-01	8.176000e-01
ol_gr1a_x	2174709.0	2.620000e-02	8.090000e-02	-0.6433	-4.900000e-03	2.070000e-02	5.460000e-02	5.422000e-01
fna_gr1a_x	2497393.0	5.700000e-03	6.030000e-02	-0.7055	0.000000e+00	0.000000e+00	0.000000e+00	6.896000e-01
fnl_gr1a_x	2418391.0	2.150000e-02	1.353000e-01	-1.2296	-1.620000e-02	1.000000e-04	5.400000e-02	1.130300e+00
nfna_gr1a_x	2418391.0	-1.580000e-02	1.552000e-01	-1.1078	-5.900000e-02	-9.000000e-04	2.760000e-02	1.384100e+00
gp_gr1a_x	2387365.0	3.580000e-02	1.161000e-01	-0.8663	-2.200000e-03	2.080000e-02	7.290000e-02	1.372100e+00
ebitda_gr1a_x	2390711.0	9.700000e-03	9.740000e-02	-0.8685	-1.050000e-02	9.300000e-03	3.840000e-02	1.237100e+00
ebit_gr1a_x	2392217.0	5.200000e-03	9.760000e-02	-0.8536	-1.310000e-02	6.700000e-03	3.280000e-02	1.345400e+00
ope_gr1a_x	2056758.0	9.400000e-03	1.005000e-01	-0.9869	-1.390000e-02	1.090000e-02	3.950000e-02	1.233300e+00
ni_gr1a_x	2402691.0	8.000000e-04	1.303000e-01	-1.6889	-1.340000e-02	3.900000e-03	2.430000e-02	2.739400e+00
nix_gr1a_x	2402691.0	6.000000e-04	1.422000e-01	-1.8549	-1.540000e-02	3.800000e-03	2.570000e-02	2.791300e+00
dp_gr1a_x	2309627.0	3.900000e-03	1.560000e-02	-0.3935	-0.000000e+00	2.500000e-03	7.500000e-03	1.932000e-01
fincf_gr1a_x	2053075.0	1.220000e-02	2.465000e-01	-2.0255	-5.480000e-02	2.700000e-03	7.330000e-02	1.485100e+00
ocf_gr1a_x	2334713.0	1.000000e-04	1.397000e-01	-0.9941	-4.190000e-02	2.900000e-03	4.640000e-02	1.151200e+00
fcf_gr1a_x	2181931.0	-7.300000e-03	1.637000e-01	-1.1368	-6.050000e-02	-4.000000e-04	5.020000e-02	1.202900e+00
nwc_gr1a_x	2164316.0	2.640000e-02	1.763000e-01	-1.4272	-2.650000e-02	1.650000e-02	7.240000e-02	9.090000e-01
eqnetis_gr1a_x	2052797.0	1.170000e-02	2.127000e-01	-1.9975	-1.000000e-02	0.000000e+00	1.380000e-02	1.207600e+00
dltnetis_gr1a_x	2373431.0	-3.100000e-03	1.313000e-01	-0.7874	-2.580000e-02	0.000000e+00	2.250000e-02	7.003000e-01
dstnetis_gr1a_x	2290818.0	7.000000e-04	8.970000e-02	-0.8063	-1.090000e-02	0.000000e+00	1.870000e-02	7.197000e-01
dbnetis_gr1a_x	2374474.0	-2.600000e-03	1.670000e-01	-1.0269	-4.130000e-02	0.000000e+00	4.330000e-02	1.017900e+00
netis_gr1a_x	2052412.0	8.700000e-03	2.717000e-01	-2.0764	-6.040000e-02	1.700000e-03	7.550000e-02	1.539900e+00
eqnpo_gr1a_x	2047069.0	-1.040000e-02	2.148000e-01	-1.1821	-1.480000e-02	0.000000e+00	1.310000e-02	1.940900e+00
tax_gr1a_x	2398103.0	3.100000e-03	2.840000e-02	-0.2157	-3.800000e-03	1.000000e-03	1.140000e-02	2.047000e-01
eqbb_gr1a_x	1893504.0	1.700000e-03	3.370000e-02	-0.3806	0.000000e+00	0.000000e+00	3.000000e-04	2.809000e-01
eqis_gr1a_x	2000469.0	1.360000e-02	2.117000e-01	-2.0255	-2.500000e-03	0.000000e+00	5.700000e-03	1.226200e+00
div_gr1a_x	2382722.0	1.100000e-03	1.270000e-02	-0.2183	0.000000e+00	0.000000e+00	1.200000e-03	2.439000e-01
eqpo_gr1a_x	1891334.0	2.900000e-03	4.380000e-02	-0.4620	-1.000000e-04	0.000000e+00	4.100000e-03	3.915000e-01
capx_gr1a_x	2184434.0	7.400000e-03	5.440000e-02	-0.4868	-7.300000e-03	2.300000e-03	1.940000e-02	4.471000e-01
be_gr1a_x	2311289.0	4.620000e-02	1.699000e-01	-2.0718	1.600000e-03	3.510000e-02	8.970000e-02	8.561000e-01
cash_gr3a_x	2081646.0	2.960000e-02	1.755000e-01	-2.5781	-1.260000e-02	9.500000e-03	6.320000e-02	9.052000e-01
inv_gr3a_x	2033267.0	2.900000e-02	8.700000e-02	-0.6971	0.000000e+00	6.800000e-03	5.550000e-02	4.115000e-01
rec_gr3a_x	2047864.0	4.970000e-02	1.082000e-01	-0.7795	1.400000e-03	3.280000e-02	8.960000e-02	4.887000e-01

	count	mean	std	min	25%	50%	75%	max
ppeg_gr3a_x	1890568.0	1.277000e-01	2.118000e-01	-2.1282	3.190000e-02	1.080000e-01	2.163000e-01	9.231000e-01
lti_gr3a_x	1864897.0	1.290000e-02	7.040000e-02	-0.6566	0.000000e+00	0.000000e+00	8.800000e-03	4.683000e-01
intan_gr3a_x	1784074.0	2.520000e-02	1.171000e-01	-1.7938	-0.000000e+00	0.000000e+00	2.360000e-02	6.632000e-01
debtst_gr3a_x	2078323.0	8.500000e-03	7.970000e-02	-0.8315	-6.500000e-03	3.000000e-04	2.440000e-02	5.514000e-01
ap_gr3a_x	1936459.0	3.440000e-02	8.510000e-02	-0.4973	-3.000000e-04	1.600000e-02	4.880000e-02	4.801000e-01
txp_gr3a_x	1751204.0	1.900000e-03	1.400000e-02	-0.0976	-1.200000e-03	0.000000e+00	4.400000e-03	1.079000e-01
debtlt_gr3a_x	2098723.0	4.090000e-02	1.579000e-01	-1.1700	-1.120000e-02	1.060000e-02	1.011000e-01	7.496000e-01
txditc_gr3a_x	1843283.0	6.200000e-03	2.480000e-02	-0.2172	0.000000e+00	0.000000e+00	1.330000e-02	1.273000e-01
coa_gr3a_x	1880953.0	7.660000e-02	1.701000e-01	-1.4412	6.100000e-03	6.190000e-02	1.549000e-01	6.791000e-01
col_gr3a_x	1907173.0	4.420000e-02	9.650000e-02	-0.9653	4.300000e-03	3.750000e-02	8.380000e-02	4.559000e-01
cowc_gr3a_x	1861920.0	3.210000e-02	1.338000e-01	-1.0405	-2.130000e-02	2.260000e-02	9.140000e-02	5.604000e-01
ncoa_gr3a_x	1899708.0	1.091000e-01	2.575000e-01	-4.5815	1.230000e-02	1.026000e-01	2.250000e-01	8.112000e-01
ncol_gr3a_x	1887939.0	1.640000e-02	5.970000e-02	-0.5782	-0.000000e+00	9.000000e-03	3.080000e-02	4.104000e-01
nncoa_gr3a_x	1861492.0	9.300000e-02	2.474000e-01	-3.9391	1.200000e-03	8.690000e-02	2.030000e-01	8.094000e-01
oa_gr3a_x	1880920.0	1.840000e-01	3.641000e-01	-5.1474	4.560000e-02	2.082000e-01	3.829000e-01	9.247000e-01
ol_gr3a_x	1887939.0	6.020000e-02	1.295000e-01	-1.1795	1.270000e-02	5.900000e-02	1.138000e-01	6.233000e-01
fna_gr3a_x	2302373.0	1.560000e-02	8.920000e-02	-1.1421	0.000000e+00	0.000000e+00	0.000000e+00	7.162000e-01
fnl_gr3a_x	2105333.0	4.560000e-02	2.040000e-01	-1.8999	-1.910000e-02	2.600000e-02	1.304000e-01	8.753000e-01
nfna_gr3a_x	2105333.0	-3.150000e-02	2.282000e-01	-1.3255	-1.318000e-01	-2.310000e-02	4.440000e-02	2.048000e+00
gp_gr3a_x	2074121.0	7.850000e-02	1.870000e-01	-1.2858	4.200000e-03	5.550000e-02	1.554000e-01	1.274100e+00
ebitda_gr3a_x	2079592.0	2.410000e-02	1.330000e-01	-1.0362	-8.600000e-03	2.410000e-02	7.360000e-02	1.478800e+00
ebit_gr3a_x	2081034.0	1.490000e-02	1.346000e-01	-1.1637	-1.460000e-02	1.620000e-02	6.010000e-02	1.985300e+00
ope_gr3a_x	1772515.0	2.290000e-02	1.350000e-01	-1.1140	-1.410000e-02	2.540000e-02	7.260000e-02	1.382600e+00
ni_gr3a_x	2095331.0	5.500000e-03	1.607000e-01	-2.0040	-1.480000e-02	8.900000e-03	4.110000e-02	3.365400e+00
nix_gr3a_x	2095331.0	5.200000e-03	1.722000e-01	-2.2144	-1.670000e-02	8.800000e-03	4.270000e-02	3.330500e+00
dp_gr3a_x	1998657.0	9.200000e-03	2.780000e-02	-0.6566	5.000000e-04	7.400000e-03	1.760000e-02	3.627000e-01
ocf_gr3a_x	2026157.0	1.030000e-02	1.536000e-01	-0.9623	-3.950000e-02	1.100000e-02	6.680000e-02	1.459300e+00
fcf_gr3a_x	1875380.0	-2.300000e-03	1.806000e-01	-0.9594	-6.520000e-02	3.500000e-03	6.430000e-02	1.668700e+00
nwc_gr3a_x	1880705.0	5.470000e-02	2.333000e-01	-3.1433	-2.400000e-02	4.470000e-02	1.438000e-01	9.475000e-01
dltnetis_gr3a_x	2057295.0	-7.000000e-03	1.381000e-01	-0.9437	-3.150000e-02	0.000000e+00	2.360000e-02	8.602000e-01
dstnetis_gr3a_x	1975805.0	-1.000000e-04	7.960000e-02	-0.7776	-1.420000e-02	0.000000e+00	1.680000e-02	6.541000e-01
dbnetis_gr3a_x	2058325.0	-7.400000e-03	1.681000e-01	-1.2437	-4.610000e-02	0.000000e+00	4.140000e-02	1.075700e+00
tax_gr3a_x	2090131.0	6.500000e-03	3.600000e-02	-0.2190	-4.800000e-03	2.700000e-03	1.970000e-02	2.106000e-01
div_gr3a_x	2069485.0	2.200000e-03	1.420000e-02	-0.2110	0.000000e+00	0.000000e+00	4.200000e-03	2.609000e-01
capx_gr3a_x	1877910.0	1.340000e-02	6.720000e-02	-0.6838	-6.700000e-03	6.500000e-03	3.240000e-02	3.679000e-01
capx_at_x	2305667.0	6.630000e-02	7.300000e-02	-0.0305	1.920000e-02	4.470000e-02	8.570000e-02	6.092000e-01

	count	mean	std	min	25%	50%	75%	max
spi_at_x	2376699.0	-1.010000e-02	4.960000e-02	-1.3123	-2.700000e-03	0.000000e+00	0.000000e+00	1.961000e-01
xido_at_x	2513016.0	-5.000000e-04	1.800000e-02	-0.4152	0.000000e+00	0.000000e+00	0.000000e+00	1.762000e-01
nri_at_x	2375825.0	-1.080000e-02	6.070000e-02	-1.5759	-4.600000e-03	0.000000e+00	0.000000e+00	2.675000e-01
gp_sale_x	2468341.0	8.440000e-02	3.062100e+00	-124.7476	2.080000e-01	3.345000e-01	5.045000e-01	9.763000e-01
ebitda_sale_x	2470375.0	-3.073000e-01	4.409900e+00	-171.6176	5.970000e-02	1.272000e-01	2.277000e-01	7.373000e-01
ebit_sale_x	2470818.0	-3.840000e-01	4.578500e+00	-185.0447	3.170000e-02	8.990000e-02	1.721000e-01	6.154000e-01
pi_sale_x	2473639.0	-4.469000e-01	4.876400e+00	-184.2990	1.190000e-02	7.260000e-02	1.445000e-01	7.101000e-01
ni_sale_x	2474362.0	-4.693000e-01	4.796100e+00	-184.2990	7.200000e-03	4.550000e-02	9.440000e-02	5.566000e-01
nix_sale_x	2472905.0	-4.745000e-01	4.848700e+00	-184.2990	6.200000e-03	4.620000e-02	9.640000e-02	6.508000e-01
ocf_sale_x	2414346.0	-3.439000e-01	3.755000e+00	-140.2577	-1.520000e-02	5.800000e-02	1.448000e-01	1.412300e+00
fcf_sale_x	2267091.0	-5.418000e-01	4.134400e+00	-125.9694	-1.053000e-01	-1.100000e-03	6.670000e-02	1.210500e+00
gp_at_x	2503159.0	3.011000e-01	2.895000e-01	-1.2660	1.023000e-01	2.659000e-01	4.563000e-01	1.412300e+00
ebitda_at_x	2505194.0	7.710000e-02	1.992000e-01	-2.1076	2.950000e-02	1.080000e-01	1.699000e-01	5.122000e-01
ebit_at_x	2506116.0	4.100000e-02	1.986000e-01	-2.1142	1.820000e-02	7.130000e-02	1.269000e-01	4.730000e-01
fi_at_x	2185678.0	1.660000e-02	2.114000e-01	-2.6041	2.010000e-02	6.410000e-02	9.800000e-02	3.716000e-01
cop_at_x	2259456.0	1.333000e-01	1.925000e-01	-1.1882	3.940000e-02	1.365000e-01	2.302000e-01	1.940400e+00
ni_at_x	2514966.0	-5.000000e-03	2.045000e-01	-2.8828	3.400000e-03	3.510000e-02	7.410000e-02	3.332000e-01
ope_be_x	2108352.0	1.569000e-01	5.427000e-01	-8.8149	9.490000e-02	2.136000e-01	3.261000e-01	3.725100e+00
ni_be_x	2444347.0	-1.990000e-02	5.962000e-01	-10.7541	1.720000e-02	9.500000e-02	1.504000e-01	1.450500e+00
nix_be_x	2444347.0	-2.270000e-02	6.187000e-01	-11.9515	1.490000e-02	9.590000e-02	1.526000e-01	1.558300e+00
ocf_be_x	2375509.0	4.150000e-02	5.350000e-01	-7.2459	-3.990000e-02	1.089000e-01	2.199000e-01	4.068700e+00
fcf_be_x	2219533.0	-1.352000e-01	6.520000e-01	-9.8959	-2.117000e-01	-4.000000e-03	1.206000e-01	2.895100e+00
gp_bev_x	2404319.0	6.940000e-01	1.236500e+00	-11.0645	2.172000e-01	4.625000e-01	8.366000e-01	1.753110e+01
ebitda_bev_x	2406313.0	5.730000e-02	1.310800e+00	-38.6063	9.750000e-02	1.837000e-01	2.972000e-01	3.290900e+00
ebit_bev_x	2406990.0	-2.510000e-02	1.386000e+00	-41.0563	5.220000e-02	1.282000e-01	2.282000e-01	2.800000e+00
fi_bev_x	2116451.0	-8.600000e-02	1.345800e+00	-38.5103	4.190000e-02	9.910000e-02	1.608000e-01	2.274200e+00
cop_bev_x	2188818.0	3.139000e-01	8.344000e-01	-8.9448	8.920000e-02	2.259000e-01	4.111000e-01	1.607970e+01
gp_ppen_x	2466653.0	2.766900e+00	6.510900e+00	-130.5385	4.559000e-01	1.518900e+00	3.353000e+00	1.035052e+02
ebitda_ppen_x	2468488.0	-1.134000e-01	1.280070e+01	-558.0000	1.689000e-01	4.726000e-01	1.116300e+00	3.389320e+01
fcf_ppen_x	2270795.0	-8.658000e-01	1.104610e+01	-423.4211	-3.778000e-01	-1.180000e-02	3.338000e-01	3.272670e+01
fincf_at_x	2181057.0	6.050000e-02	2.270000e-01	-0.9085	-4.100000e-02	1.800000e-03	8.120000e-02	1.643700e+00
netis_at_x	2180970.0	2.900000e-02	2.576000e-01	-1.3681	-4.860000e-02	0.000000e+00	5.940000e-02	1.592800e+00
eqnetis_at_x	2181226.0	5.680000e-02	1.918000e-01	-0.3507	-8.000000e-04	6.000000e-04	1.520000e-02	1.488800e+00
eqis_at_x	2142004.0	7.050000e-02	1.912000e-01	-0.1034	0.000000e+00	3.200000e-03	2.280000e-02	1.535600e+00
dbnetis_at_x	2487875.0	-2.120000e-02	1.573000e-01	-1.3624	-3.980000e-02	-8.000000e-04	2.270000e-02	6.456000e-01
dltnetis_at_x	2487184.0	-2.430000e-02	1.364000e-01	-1.2268	-3.180000e-02	-2.200000e-03	1.200000e-03	5.184000e-01

	count	mean	std	min	25%	50%	75%	max
dstnetis_at_x	2428021.0	3.500000e-03	6.050000e-02	-0.4789	-5.100000e-03	0.000000e+00	1.130000e-02	4.836000e-01
eqnpo_at_x	2177364.0	-4.470000e-02	1.949000e-01	-1.4673	-1.110000e-02	8.000000e-04	2.020000e-02	4.462000e-01
eqbb_at_x	2059717.0	1.250000e-02	3.500000e-02	-0.0026	0.000000e+00	0.000000e+00	5.300000e-03	4.018000e-01
div_at_x	2500964.0	1.160000e-02	2.170000e-02	0.0000	0.000000e+00	1.900000e-03	1.660000e-02	3.183000e-01
oaccruals_at_x	2261617.0	-1.580000e-02	1.522000e-01	-2.2637	-7.200000e-02	-1.830000e-02	4.760000e-02	6.719000e-01
oaccruals_ni_x	2260635.0	-5.853000e-01	6.180500e+00	-71.4418	-1.208700e+00	-2.712000e-01	6.967000e-01	8.515790e+01
taccruals_at_x	2240180.0	-3.100000e-02	2.045000e-01	-2.4802	-9.100000e-02	-1.180000e-02	4.930000e-02	1.294200e+00
taccruals_ni_x	2238904.0	-1.448100e+00	8.683400e+00	-131.5096	-1.516600e+00	-1.946000e-01	7.622000e-01	6.728570e+01
noa_at_x	2142866.0	6.816000e-01	4.649000e-01	-1.1515	4.896000e-01	6.884000e-01	8.418000e-01	1.038840e+01
be_bev_x	2368048.0	1.343100e+00	2.666700e+00	0.0326	5.543000e-01	8.086000e-01	1.190400e+00	6.053070e+01
debt_bev_x	2416506.0	4.732000e-01	6.162000e-01	0.0000	1.399000e-01	3.804000e-01	6.012000e-01	1.276120e+01
cash_bev_x	2397575.0	8.357000e-01	3.110100e+00	0.0000	3.800000e-02	1.245000e-01	4.276000e-01	8.007360e+01
pstk_bev_x	2418755.0	2.720000e-02	1.704000e-01	0.0000	0.000000e+00	0.000000e+00	0.000000e+00	7.089400e+00
debtlt_bev_x	2412477.0	3.446000e-01	4.482000e-01	0.0000	5.390000e-02	2.671000e-01	4.815000e-01	9.026500e+00
debtst_bev_x	2403343.0	1.233000e-01	2.903000e-01	0.0000	3.200000e-03	3.390000e-02	1.172000e-01	5.633000e+00
int_debt_x	1959042.0	1.258000e-01	3.153000e-01	0.0000	5.310000e-02	7.610000e-02	1.063000e-01	7.750000e+00
int_debtlt_x	1874541.0	3.393000e-01	1.552500e+00	0.0000	6.360000e-02	9.400000e-02	1.485000e-01	4.145000e+01
ebitda_debt_x	2242375.0	2.161600e+00	2.312980e+01	-362.2105	1.666000e-01	3.959000e-01	9.501000e-01	5.562212e+02
profit_cl_x	2270271.0	4.298000e-01	1.566600e+00	-11.9038	2.114000e-01	5.648000e-01	1.016300e+00	6.155300e+00
ocf_cl_x	2269486.0	5.390000e-02	1.456200e+00	-14.9568	-1.363000e-01	2.183000e-01	5.993000e-01	5.976400e+00
ocf_debt_x	2189764.0	1.253200e+00	1.968000e+01	-264.1167	-7.590000e-02	1.564000e-01	5.185000e-01	4.307215e+02
cash_lt_x	2487462.0	7.781000e-01	2.113200e+00	0.0000	4.150000e-02	1.312000e-01	5.084000e-01	2.990910e+01
inv_act_x	2124755.0	2.719000e-01	2.276000e-01	0.0000	4.860000e-02	2.538000e-01	4.448000e-01	9.113000e-01
rec_act_x	2130411.0	3.499000e-01	2.071000e-01	0.0000	1.990000e-01	3.479000e-01	4.754000e-01	9.455000e-01
debtst_debt_x	2235158.0	2.916000e-01	3.181000e-01	0.0000	3.900000e-02	1.578000e-01	4.582000e-01	1.000000e+00
cl_lt_x	2271050.0	5.408000e-01	2.822000e-01	0.0172	3.033000e-01	5.188000e-01	7.861000e-01	1.000000e+00
debtlt_debt_x	2251637.0	7.215000e-01	3.158000e-01	0.0000	5.637000e-01	8.571000e-01	9.724000e-01	1.000000e+00
lt_ppen_x	2467297.0	1.413180e+01	4.095230e+01	0.0809	1.032300e+00	2.019600e+00	5.768200e+00	7.630447e+02
debtlt_be_x	2439883.0	7.140000e-01	1.464700e+00	0.0000	3.360000e-02	3.025000e-01	7.618000e-01	2.225160e+01
opex_at_x	2503218.0	9.413000e-01	8.196000e-01	0.0029	3.295000e-01	7.872000e-01	1.304500e+00	7.158500e+00
nwc_at_x	2253296.0	2.724000e-01	2.457000e-01	-0.7924	8.520000e-02	2.536000e-01	4.349000e-01	9.547000e-01
debt_at_x	2514980.0	2.331000e-01	2.095000e-01	0.0000	5.090000e-02	1.957000e-01	3.591000e-01	1.428700e+00
debt_be_x	2444508.0	9.825000e-01	1.972300e+00	0.0000	9.520000e-02	4.426000e-01	1.023800e+00	3.440000e+01
ebit_int_x	2038745.0	1.266250e+01	1.784445e+02	-3702.0000	1.253300e+00	4.003000e+00	1.124330e+01	3.302250e+03
inv_days_x	2394275.0	8.869850e+01	1.683021e+02	0.0000	9.009300e+00	5.392190e+01	1.091676e+02	3.574195e+03
rec_days_x	2403668.0	3.602296e+02	9.967740e+02	0.0000	3.863530e+01	5.827670e+01	8.822010e+01	7.354934e+03

	count	mean	std	min	25%	50%	75%	max
ap_days_x	2314657.0	1.459695e+03	7.489965e+03	0.7812	2.587680e+01	4.209780e+01	7.865320e+01	1.412089e+05
cash_conversion_x	1836443.0	1.256743e+02	2.122532e+02	0.0000	4.172550e+01	8.193360e+01	1.398610e+02	3.521431e+03
cash_cl_x	2262167.0	1.419800e+00	3.231200e+00	0.0000	1.124000e-01	3.726000e-01	1.177400e+00	3.650000e+01
caliq_cl_x	2241081.0	2.487700e+00	3.827100e+00	0.0581	9.004000e-01	1.378900e+00	2.376600e+00	4.066670e+01
ca_cl_x	2252774.0	3.162200e+00	3.912700e+00	0.0824	1.372500e+00	2.102000e+00	3.307100e+00	4.119530e+01
inv_turnover_x	1990611.0	1.861590e+01	4.951140e+01	0.0438	2.956600e+00	5.130900e+00	1.205000e+01	7.307939e+02
at_turnover_x	2482416.0	1.084900e+00	9.318000e-01	0.0000	3.768000e-01	9.269000e-01	1.525100e+00	9.298300e+00
rec_turnover_x	2400338.0	1.234110e+01	2.636800e+01	0.0000	4.039600e+00	6.187900e+00	9.236800e+00	2.787135e+02
ap_turnover_x	2229997.0	1.163840e+01	1.238900e+01	-0.1258	4.826800e+00	8.918500e+00	1.434510e+01	1.336129e+02
sale_bev_x	2408388.0	2.269200e+00	2.923100e+00	0.0000	7.623000e-01	1.580300e+00	2.598800e+00	3.887110e+01
sale_be_x	2437063.0	2.732600e+00	3.718300e+00	0.0000	9.001000e-01	1.758000e+00	3.096000e+00	5.438940e+01
div_ni_x	1963756.0	3.126000e-01	5.775000e-01	0.0000	0.000000e+00	1.650000e-01	4.135000e-01	1.293670e+01
sale_nwc_x	2017664.0	9.746900e+00	2.267620e+01	0.0000	2.066900e+00	3.971600e+00	7.750900e+00	3.110241e+02
tax_pi_x	1999061.0	3.279000e-01	3.117000e-01	-7.2981	2.705000e-01	3.654000e-01	4.329000e-01	5.548900e+00
cash_at_x	2496082.0	1.581000e-01	2.035000e-01	0.0000	2.580000e-02	7.260000e-02	2.026000e-01	9.799000e-01
ni_emp_x	2332173.0	-1.044570e+01	1.898294e+02	-3810.3810	4.055000e-01	4.200600e+00	1.703640e+01	1.438498e+03
sale_emp_x	2328826.0	2.691786e+02	5.003031e+02	0.0000	6.301400e+01	1.411000e+02	2.763478e+02	7.782523e+03
sale_emp_gr1_x	2120715.0	1.123000e-01	4.553000e-01	-0.9563	-3.330000e-02	5.300000e-02	1.513000e-01	7.027000e+00
emp_gr1_x	2048454.0	7.670000e-02	2.504000e-01	-1.3333	-3.060000e-02	4.520000e-02	1.538000e-01	1.483100e+00
ni_inc8q_x	1837805.0	3.116800e+00	3.262400e+00	0.0000	0.000000e+00	2.000000e+00	7.000000e+00	8.000000e+00
noa_gr1a_x	2130139.0	1.277000e-01	4.002000e-01	-0.7366	-1.750000e-02	4.940000e-02	1.574000e-01	1.075230e+01
ppeinv_gr1a_x	2130674.0	1.104000e-01	2.282000e-01	-0.5663	9.400000e-03	5.870000e-02	1.436000e-01	3.078700e+00
lnoa_gr1a_x	2042945.0	3.180000e-02	9.170000e-02	-0.5778	-3.800000e-03	1.370000e-02	4.740000e-02	7.544000e-01
capx_gr2_x	1996106.0	1.219100e+00	4.305300e+00	-1.4277	-2.477000e-01	2.272000e-01	1.043000e+00	7.697220e+01
saleq_gr1_x	2256822.0	2.428000e-01	8.315000e-01	-1.0000	-1.270000e-02	9.890000e-02	2.606000e-01	1.574840e+01
niq_be_x	2153966.0	5.000000e-04	1.393000e-01	-2.0216	1.600000e-03	2.420000e-02	4.290000e-02	6.993000e-01
niq_at_x	2218680.0	-2.200000e-03	6.080000e-02	-0.6672	0.000000e+00	8.200000e-03	2.060000e-02	1.818000e-01
niq_be_chg1_x	1961181.0	-7.700000e-03	1.339000e-01	-2.0038	-1.650000e-02	-6.000000e-04	1.090000e-02	1.227600e+00
niq_at_chg1_x	2044996.0	3.000000e-04	5.400000e-02	-0.4547	-7.100000e-03	-0.000000e+00	5.600000e-03	8.413000e-01
dsale_dinv_x	1796036.0	-4.380000e-02	8.780000e-01	-19.4778	-1.460000e-01	2.150000e-02	1.949000e-01	5.598300e+00
dsale_drec_x	2136436.0	-3.080000e-02	6.202000e-01	-7.3996	-1.418000e-01	1.500000e-03	1.418000e-01	7.637700e+00
dgp_dsale_x	2120443.0	2.720000e-02	5.405000e-01	-5.9700	-7.530000e-02	2.300000e-03	8.380000e-02	1.201120e+01
dsale_dsga_x	1827645.0	2.310000e-02	3.643000e-01	-2.2251	-8.920000e-02	-1.000000e-04	9.360000e-02	6.963700e+00
saleq_su_x	1944544.0	1.618000e-01	1.699500e+00	-16.0960	-8.666000e-01	1.532000e-01	1.125000e+00	3.358810e+01
niq_su_x	1972831.0	-1.123000e-01	1.940400e+00	-50.8463	-7.565000e-01	5.100000e-03	7.529000e-01	2.019490e+01
capex_abn_x	1806456.0	1.173000e-01	9.626000e-01	-1.1469	-3.685000e-01	-6.920000e-02	2.932000e-01	1.196350e+01

	count	mean	std	min	25%	50%	75%	max
op_atl1_x	2415570.0	1.320000e-01	2.472000e-01	-6.9463	4.860000e-02	1.355000e-01	2.227000e-01	1.125400e+00
gp_atl1_x	2413733.0	3.639000e-01	3.763000e-01	-1.9036	1.162000e-01	3.032000e-01	5.356000e-01	2.788000e+00
ope_bel1_x	2010286.0	2.202000e-01	6.575000e-01	-13.6285	1.063000e-01	2.425000e-01	3.880000e-01	4.617600e+00
cop_atl1_x	2237311.0	1.409000e-01	2.863000e-01	-3.8344	4.500000e-02	1.505000e-01	2.563000e-01	1.923400e+00
pi_nix_x	1959639.0	1.615000e+00	6.861000e-01	0.1059	1.340700e+00	1.572900e+00	1.777900e+00	1.989360e+01
ocf_at_x	2449158.0	1.150000e-02	1.872000e-01	-1.8184	-2.140000e-02	4.090000e-02	1.033000e-01	5.979000e-01
op_at_x	2505194.0	1.113000e-01	1.575000e-01	-1.2330	4.200000e-02	1.205000e-01	1.892000e-01	5.662000e-01
ocf_at_chg1_x	2333855.0	2.300000e-03	1.627000e-01	-1.0782	-4.770000e-02	-1.000000e-04	4.630000e-02	1.390100e+00
at_be_x	2452393.0	3.714900e+00	4.779700e+00	1.0000	1.469000e+00	2.029600e+00	3.240900e+00	5.963100e+01
niq_saleq_std_x	1902197.0	1.360600e+00	1.149800e+01	0.0008	1.930000e-02	4.260000e-02	1.236000e-01	3.177766e+02
roe_be_std_x	1799259.0	1.611000e-01	4.732000e-01	0.0021	2.230000e-02	4.760000e-02	1.133000e-01	9.225400e+00
tangibility_x	2201788.0	6.502000e-01	1.916000e-01	0.0025	5.540000e-01	6.638000e-01	7.614000e-01	1.684700e+00
earnings_variability_x	1752776.0	8.639000e-01	1.037400e+00	0.0243	2.577000e-01	5.765000e-01	1.052900e+00	1.145280e+01
aliq_at_x	2174808.0	8.263000e-01	8.005000e-01	0.1044	5.792000e-01	6.946000e-01	8.423000e-01	2.803980e+01
f_score_x	1978727.0	4.911500e+00	1.728500e+00	0.0000	4.000000e+00	5.000000e+00	6.000000e+00	9.000000e+00
o_score_x	2127585.0	-1.902100e+00	3.035200e+00	-9.3872	-3.598500e+00	-2.309000e+00	-8.857000e-01	2.287030e+01
z_score_x	2126989.0	5.526800e+00	9.357000e+00	-37.3359	1.992200e+00	3.446700e+00	5.637300e+00	1.744239e+02
intrinsic_value_x	1899809.0	1.317903e+03	5.258077e+03	0.0982	3.489040e+01	1.295681e+02	5.706605e+02	1.130984e+05
kz_index_x	2167838.0	-1.126290e+01	5.190800e+01	-1723.5716	-6.936600e+00	-1.467200e+00	5.962000e-01	8.903350e+01
gpoa_ch5_x	1799428.0	-5.000000e-03	1.939000e-01	-1.1201	-7.080000e-02	-2.900000e-03	5.560000e-02	1.669700e+00
roe_ch5_x	1718355.0	-1.400000e-02	5.543000e-01	-7.5143	-7.570000e-02	-6.100000e-03	5.400000e-02	7.791500e+00
roa_ch5_x	1824336.0	4.900000e-03	1.917000e-01	-1.6595	-3.640000e-02	-1.800000e-03	2.650000e-02	3.283900e+00
cfoa_ch5_x	1759171.0	1.520000e-02	1.825000e-01	-0.9610	-5.570000e-02	2.500000e-03	7.100000e-02	2.175100e+00
gmar_ch5_x	1777826.0	4.200000e-02	9.593000e-01	-24.3597	-4.330000e-02	2.700000e-03	5.140000e-02	3.059480e+01
ni_ar1_x	1798398.0	2.127000e-01	6.110000e-01	-3.9640	-1.463000e-01	1.674000e-01	5.078000e-01	9.144200e+00
ni_ivol_x	1798398.0	5.090000e-02	1.054000e-01	0.0003	7.900000e-03	1.910000e-02	4.640000e-02	1.756800e+00
at_me_x	2522907.0	2.710400e+00	4.953100e+00	0.0086	5.788000e-01	1.230200e+00	2.687400e+00	1.923122e+02
be_me_x	2452453.0	7.411000e-01	7.141000e-01	0.0050	3.072000e-01	5.729000e-01	9.557000e-01	2.516310e+01
debt_me_x	2515141.0	7.136000e-01	1.647800e+00	0.0000	3.610000e-02	2.333000e-01	7.145000e-01	6.550580e+01
netdebt_me_x	2515141.0	4.707000e-01	1.480500e+00	-3.4965	-6.240000e-02	1.146000e-01	5.472000e-01	5.866260e+01
cash_me_x	2496218.0	2.459000e-01	5.843000e-01	0.0000	3.340000e-02	9.500000e-02	2.301000e-01	1.478940e+01
sale_me_x	2509790.0	1.848400e+00	3.088100e+00	0.0000	3.854000e-01	9.080000e-01	2.049100e+00	7.507530e+01
gp_me_x	2504145.0	4.729000e-01	6.845000e-01	-5.3506	1.503000e-01	2.955000e-01	5.587000e-01	1.896990e+01
ebitda_me_x	2506237.0	1.594000e-01	2.707000e-01	-5.8474	5.650000e-02	1.331000e-01	2.363000e-01	5.597900e+00
ebit_me_x	2507305.0	9.600000e-02	2.455000e-01	-7.4186	3.000000e-02	9.500000e-02	1.716000e-01	3.506600e+00
ope_me_x	2183835.0	1.085000e-01	2.516000e-01	-8.0248	3.920000e-02	1.084000e-01	1.911000e-01	3.793500e+00

	count	mean	std	min	25%	50%	75%	max
ni_me_x	2517298.0	1.200000e-03	3.459000e-01	-18.9294	5.500000e-03	4.900000e-02	8.530000e-02	9.917000e-01
nix_me_x	2517298.0	-1.200000e-03	3.693000e-01	-20.3694	4.100000e-03	4.920000e-02	8.650000e-02	1.036200e+00
cop_me_x	2259562.0	2.183000e-01	5.014000e-01	-3.5452	4.550000e-02	1.406000e-01	2.768000e-01	2.124680e+01
ocf_me_x	2450553.0	4.280000e-02	2.747000e-01	-5.6691	-1.830000e-02	5.360000e-02	1.205000e-01	5.711200e+00
fcf_me_x	2303306.0	-7.030000e-02	3.536000e-01	-8.5448	-1.065000e-01	-2.600000e-03	5.530000e-02	4.202300e+00
div_me_x	2501593.0	1.780000e-02	2.950000e-02	0.0000	0.000000e+00	3.900000e-03	2.660000e-02	1.049700e+00
eqbb_me_x	2059868.0	1.380000e-02	3.780000e-02	-0.0037	0.000000e+00	0.000000e+00	7.800000e-03	8.704000e-01
eqis_me_x	2142182.0	4.550000e-02	1.388000e-01	-0.1339	1.000000e-04	3.500000e-03	1.830000e-02	5.839400e+00
eqpo_me_x	2058263.0	3.150000e-02	5.660000e-02	-0.0013	0.000000e+00	1.120000e-02	4.150000e-02	1.725500e+00
eqnpo_me_x	2177501.0	-1.430000e-02	1.450000e-01	-6.1142	-8.100000e-03	1.200000e-03	3.130000e-02	1.442900e+00
eqnetis_me_x	2181408.0	3.130000e-02	1.401000e-01	-0.6866	-1.400000e-03	7.000000e-04	1.260000e-02	5.679700e+00
at_mev_x	2480516.0	1.759600e+00	3.280300e+00	0.0085	5.638000e-01	1.008000e+00	1.587100e+00	6.916660e+01
bev_mev_x	2404633.0	6.919000e-01	5.487000e-01	0.0009	3.194000e-01	6.308000e-01	9.482000e-01	1.692550e+01
ppen_mev_x	2459710.0	3.322000e-01	3.872000e-01	0.0000	5.950000e-02	1.893000e-01	4.753000e-01	6.654400e+00
be_mev_x	2410201.0	6.153000e-01	8.336000e-01	0.0050	2.513000e-01	4.357000e-01	7.057000e-01	2.914710e+01
cash_mev_x	2460357.0	2.333000e-01	6.611000e-01	0.0000	2.350000e-02	6.940000e-02	1.825000e-01	1.486960e+01
sale_mev_x	2472091.0	1.265200e+00	1.765600e+00	0.0000	3.146000e-01	7.343000e-01	1.550900e+00	3.775600e+01
gp_mev_x	2467238.0	3.453000e-01	4.647000e-01	-2.4081	1.209000e-01	2.305000e-01	4.284000e-01	1.314000e+01
ebitda_mev_x	2469299.0	1.012000e-01	2.101000e-01	-5.5869	5.090000e-02	1.060000e-01	1.669000e-01	2.711700e+00
ebit_mev_x	2470075.0	6.010000e-02	2.226000e-01	-6.8743	2.670000e-02	7.470000e-02	1.222000e-01	2.601300e+00
cop_mev_x	2243652.0	1.516000e-01	2.798000e-01	-2.3844	4.200000e-02	1.203000e-01	2.126000e-01	8.747500e+00
ocf_mev_x	2431339.0	3.150000e-02	1.968000e-01	-4.7377	-1.650000e-02	4.340000e-02	9.350000e-02	2.334400e+00
fcf_mev_x	2286863.0	-3.800000e-02	2.261000e-01	-6.0410	-8.630000e-02	-2.300000e-03	4.670000e-02	1.728000e+00
debt_mev_x	2480615.0	3.008000e-01	3.502000e-01	0.0000	4.020000e-02	2.106000e-01	4.607000e-01	7.224300e+00
pstk_mev_x	2479267.0	1.480000e-02	5.860000e-02	0.0000	0.000000e+00	0.000000e+00	0.000000e+00	1.220500e+00
debtlt_mev_x	2476104.0	2.224000e-01	2.443000e-01	0.0000	1.400000e-02	1.446000e-01	3.542000e-01	2.411300e+00
debtst_mev_x	2461067.0	8.090000e-02	2.125000e-01	0.0000	9.000000e-04	1.690000e-02	7.010000e-02	5.292900e+00
dltnetis_mev_x	2453443.0	-3.150000e-02	1.853000e-01	-3.5613	-3.440000e-02	-1.900000e-03	1.200000e-03	6.324000e-01
dstnetis_mev_x	2393968.0	4.100000e-03	9.390000e-02	-1.0163	-4.800000e-03	0.000000e+00	1.110000e-02	1.122900e+00
dbnetis_mev_x	2454176.0	-2.880000e-02	2.223000e-01	-4.4848	-4.210000e-02	-6.000000e-04	2.280000e-02	1.188800e+00
netis_mev_x	2164671.0	-8.300000e-03	2.729000e-01	-4.6395	-5.040000e-02	0.000000e+00	5.030000e-02	5.358400e+00
fincf_mev_x	2164802.0	3.700000e-02	2.405000e-01	-2.3006	-4.040000e-02	1.300000e-03	7.090000e-02	6.822000e+00
aliq_mat_x	2036506.0	5.016000e-01	2.661000e-01	0.0270	3.052000e-01	4.793000e-01	6.504000e-01	3.973200e+00
eq_dur_x	2193667.0	1.598720e+01	5.630900e+00	0.2861	1.413720e+01	1.612420e+01	1.764670e+01	3.430355e+02
beta_60m_x	2090801.0	1.153800e+00	6.856000e-01	-1.7467	6.897000e-01	1.081600e+00	1.528500e+00	4.912400e+00
ivol_capm_60m_x	2090801.0	1.172000e-01	6.560000e-02	0.0288	7.050000e-02	1.002000e-01	1.454000e-01	5.392000e-01

	count	mean	std	min	25%	50%	75%	max
resff3_12_1_x	2274040.0	-2.210000e-02	2.736000e-01	-1.1550	-1.908000e-01	-8.900000e-03	1.610000e-01	7.899000e-01
resff3_6_1_x	2273172.0	-5.420000e-02	5.396000e-01	-2.9537	-3.435000e-01	-2.040000e-02	2.734000e-01	1.925800e+00
mispricing_mgmt_x	2414716.0	4.896000e-01	1.856000e-01	0.0147	3.610000e-01	5.047000e-01	6.284000e-01	9.427000e-01
mispricing_perf_x	2649116.0	5.208000e-01	2.065000e-01	0.0099	3.773000e-01	5.270000e-01	6.749000e-01	9.881000e-01
zero_trades_21d_x	2568596.0	9.102000e-01	2.670500e+00	0.0000	1.800000e-03	3.700000e-03	7.200000e-03	2.100980e+01
dolvol_126d_x	2527407.0	1.272436e+07	5.041472e+07	36.1000	6.023594e+04	3.756701e+05	3.493927e+06	1.038495e+09
dolvol_var_126d_x	2527340.0	1.275800e+00	7.751000e-01	0.2622	7.587000e-01	1.088500e+00	1.545100e+00	8.289100e+00
turnover_126d_x	2527415.0	4.300000e-03	6.800000e-03	0.0000	9.000000e-04	2.200000e-03	5.300000e-03	2.857000e-01
turnover_var_126d_x	2527348.0	1.251900e+00	7.609000e-01	0.2796	7.459000e-01	1.058700e+00	1.509700e+00	7.678300e+00
zero_trades_126d_x	2527415.0	9.170000e-01	2.511100e+00	0.0000	1.900000e-03	4.000000e-03	1.771000e-01	1.949730e+01
zero_trades_252d_x	2472485.0	9.236000e-01	2.470600e+00	0.0001	2.000000e-03	4.300000e-03	2.625000e-01	1.910030e+01
bidaskhl_21d_x	2474735.0	1.470000e-02	1.810000e-02	0.0011	5.600000e-03	9.400000e-03	1.710000e-02	5.318000e-01
rvolhl_21d_x	2474735.0	2.130000e-02	1.570000e-02	0.0000	1.100000e-02	1.720000e-02	2.680000e-02	1.854000e-01
beta_21d_x	2469080.0	8.736000e-01	1.205700e+00	-11.1429	2.238000e-01	8.042000e-01	1.458000e+00	1.276490e+01
ivol_capm_21d_x	2469080.0	2.710000e-02	1.960000e-02	0.0018	1.400000e-02	2.160000e-02	3.380000e-02	2.415000e-01
iskew_capm_21d_x	2469046.0	2.407000e-01	8.745000e-01	-3.5665	-2.542000e-01	2.053000e-01	7.097000e-01	3.715300e+00
coskew_21d_x	2469074.0	-1.530000e-02	3.111000e-01	-1.4678	-2.232000e-01	-2.070000e-02	1.886000e-01	1.347500e+00
beta_dimson_21d_x	2469080.0	9.503000e-01	1.950600e+00	-19.3713	4.290000e-02	8.515000e-01	1.798400e+00	2.341690e+01
ivol_ff3_21d_x	2469080.0	2.640000e-02	1.930000e-02	0.0018	1.360000e-02	2.100000e-02	3.300000e-02	2.340000e-01
iskew_ff3_21d_x	2469068.0	1.990000e-01	7.943000e-01	-3.1203	-2.632000e-01	1.696000e-01	6.344000e-01	3.455800e+00
ivol_hxz4_21d_x	2332649.0	2.680000e-02	1.960000e-02	0.0018	1.370000e-02	2.130000e-02	3.350000e-02	2.397000e-01
iskew_hxz4_21d_x	2332643.0	1.777000e-01	7.585000e-01	-3.0805	-2.681000e-01	1.513000e-01	6.005000e-01	3.275600e+00
rmax5_21d_x	2469033.0	3.860000e-02	2.910000e-02	0.0022	1.960000e-02	3.050000e-02	4.810000e-02	3.544000e-01
rmax1_21d_x	2469033.0	6.730000e-02	5.830000e-02	0.0035	3.110000e-02	5.000000e-02	8.280000e-02	8.996000e-01
rvol_21d_x	2469080.0	2.970000e-02	2.060000e-02	0.0018	1.590000e-02	2.400000e-02	3.690000e-02	2.515000e-01
rskew_21d_x	2469038.0	2.439000e-01	8.740000e-01	-3.5810	-2.529000e-01	2.077000e-01	7.136000e-01	3.808400e+00
ami_126d_x	2427976.0	2.294900e+00	1.277990e+01	0.0000	6.300000e-03	8.310000e-02	7.621000e-01	7.242321e+02
beta_252d_x	2434576.0	8.972000e-01	6.011000e-01	-1.8325	4.682000e-01	8.481000e-01	1.259200e+00	4.013900e+00
ivol_capm_252d_x	2434576.0	2.910000e-02	1.710000e-02	0.0050	1.700000e-02	2.480000e-02	3.650000e-02	1.684000e-01
betadown_252d_x	2406390.0	1.001300e+00	7.817000e-01	-3.9821	5.127000e-01	9.352000e-01	1.414000e+00	5.699200e+00
prc_highprc_252d_x	2434268.0	7.724000e-01	1.997000e-01	0.0167	6.610000e-01	8.272000e-01	9.318000e-01	1.000000e+00
rvol_252d_x	2434576.0	3.110000e-02	1.740000e-02	0.0052	1.870000e-02	2.680000e-02	3.870000e-02	1.690000e-01
corr_1260d_x	1904407.0	3.603000e-01	1.650000e-01	-0.0374	2.362000e-01	3.573000e-01	4.786000e-01	8.219000e-01
betabab_1260d_x	1893789.0	1.075600e+00	5.871000e-01	-0.3259	6.475000e-01	1.000500e+00	1.410600e+00	4.274900e+00
rmax5_rvol_21d_x	2343331.0	1.232900e+00	5.049000e-01	0.1125	8.720000e-01	1.159900e+00	1.512600e+00	4.328700e+00
age_x	2739928.0	2.184690e+02	1.888040e+02	1.0000	7.900000e+01	1.590000e+02	3.000000e+02	1.115000e+03

	count	mean	std	min	25%	50%	75%	max
qmj_x	1825615.0	8.990000e-02	9.763000e-01	-1.7027	-7.318000e-01	1.204000e-01	9.350000e-01	1.701100e+00
qmj_prof_x	2502382.0	9.110000e-02	9.846000e-01	-1.7036	-7.339000e-01	1.300000e-01	9.456000e-01	1.698800e+00
qmj_growth_x	1825622.0	3.610000e-02	9.739000e-01	-1.7018	-7.911000e-01	4.900000e-02	8.716000e-01	1.702100e+00
qmj_safety_x	2579701.0	8.730000e-02	9.713000e-01	-1.7012	-7.189000e-01	1.215000e-01	9.239000e-01	1.708800e+00
r	2739928.0	6.400000e-03	1.555000e-01	-1.0113	-6.530000e-02	-1.400000e-03	6.680000e-02	1.988170e+01
ri	2739928.0	1.770830e+01	6.890425e+02	-0.2196	6.348000e-01	1.290100e+00	3.413100e+00	1.527087e+05
r_f001m	2717410.0	-3.000000e-04	1.452874e+03	-11994.7451	-6.730875e+02	-6.488730e+01	5.622922e+02	1.975796e+05
r_f002m	2694932.0	-4.000000e-04	1.453405e+03	-12093.2324	-6.719280e+02	-6.400750e+01	5.623110e+02	1.340182e+05
r_f003m	2672377.0	-5.000000e-04	1.452500e+03	-12181.5869	-6.704744e+02	-6.279780e+01	5.626841e+02	1.023323e+05
r_f004m	2649956.0	3.000000e-04	1.467237e+03	-12221.2090	-6.701777e+02	-6.287600e+01	5.621040e+02	1.259151e+05
r_f005m	2627466.0	1.000000e-04	1.471883e+03	-12192.0312	-6.696848e+02	-6.289280e+01	5.614439e+02	1.259212e+05
r_f006m	2604896.0	3.000000e-04	1.479795e+03	-12242.7471	-6.699737e+02	-6.314070e+01	5.606614e+02	1.259316e+05
r_f007m	2582271.0	6.000000e-04	1.481432e+03	-12249.9131	-6.683008e+02	-6.239580e+01	5.602472e+02	1.895877e+05
r_f008m	2559645.0	-4.000000e-04	1.494461e+03	-12270.5273	-6.684138e+02	-6.289780e+01	5.595248e+02	1.975297e+05
r_f009m	2536940.0	-7.000000e-04	1.497932e+03	-12302.8760	-6.676816e+02	-6.255130e+01	5.583124e+02	1.975135e+05
r_f010m	2514233.0	0.000000e+00	1.505680e+03	-12306.7148	-6.674777e+02	-6.308580e+01	5.573524e+02	1.975094e+05
r_f011m	2491745.0	0.000000e+00	1.511348e+03	-12250.5898	-6.675006e+02	-6.366780e+01	5.557687e+02	1.975246e+05
r_f012m	2469229.0	-4.000000e-04	1.506349e+03	-12207.8350	-6.657420e+02	-6.324470e+01	5.550534e+02	1.975047e+05
r_f013m	2446715.0	-1.000000e-04	1.515108e+03	-12176.1465	-6.655490e+02	-6.377990e+01	5.530358e+02	1.974958e+05
r_f014m	2424394.0	2.000000e-04	1.510473e+03	-11440.9531	-6.651825e+02	-6.383700e+01	5.523057e+02	1.974974e+05
r_f015m	2402314.0	-1.000000e-04	1.512811e+03	-11449.6279	-6.642017e+02	-6.469670e+01	5.507740e+02	1.974900e+05
r_f016m	2380363.0	2.000000e-04	1.515042e+03	-11449.5361	-6.646672e+02	-6.581010e+01	5.496704e+02	1.974964e+05
r_f017m	2358474.0	-1.000000e-04	1.518632e+03	-11473.9346	-6.650385e+02	-6.704440e+01	5.478585e+02	1.974688e+05
r_f018m	2336696.0	1.000000e-04	1.522247e+03	-11924.3223	-6.646155e+02	-6.711870e+01	5.465416e+02	1.974814e+05
r_f019m	2315092.0	1.000000e-04	1.522928e+03	-11477.0908	-6.638971e+02	-6.765280e+01	5.454422e+02	1.974768e+05
r_f020m	2293683.0	-1.000000e-04	1.526165e+03	-11837.1934	-6.631345e+02	-6.773940e+01	5.443864e+02	1.974707e+05
r_f021m	2272416.0	-0.000000e+00	1.525184e+03	-11830.5957	-6.630684e+02	-6.768750e+01	5.435944e+02	1.974645e+05
r_f022m	2251211.0	1.000000e-04	1.521066e+03	-11822.4795	-6.620583e+02	-6.732170e+01	5.424589e+02	1.974786e+05
r_f023m	2230142.0	-0.000000e+00	1.521688e+03	-11815.7969	-6.616403e+02	-6.751140e+01	5.414223e+02	1.974742e+05
r_f024m	2209316.0	-0.000000e+00	1.520726e+03	-11948.9941	-6.596912e+02	-6.725050e+01	5.403683e+02	1.974745e+05
r_f025m	2188719.0	1.000000e-04	1.521428e+03	-11828.5732	-6.593975e+02	-6.767100e+01	5.392651e+02	1.974657e+05
r_f026m	2168242.0	0.000000e+00	1.522714e+03	-11837.6758	-6.583010e+02	-6.773110e+01	5.379360e+02	1.974702e+05
r_f027m	2147901.0	1.000000e-04	1.511179e+03	-11839.1934	-6.568350e+02	-6.728920e+01	5.372294e+02	1.895889e+05
r_f028m	2127702.0	1.000000e-04	1.507530e+03	-11834.7500	-6.557527e+02	-6.652340e+01	5.364533e+02	1.895897e+05
r_f029m	2107711.0	1.000000e-04	1.502109e+03	-11846.0547	-6.544728e+02	-6.637620e+01	5.357737e+02	1.896120e+05
r_f030m	2087915.0	-1.000000e-04	1.498496e+03	-11853.5107	-6.531980e+02	-6.626840e+01	5.346227e+02	1.896178e+05

	count	mean	std	min	25%	50%	75%	max
r_f031m	2068400.0	0.000000e+00	1.492586e+03	-11866.2324	-6.525869e+02	-6.630400e+01	5.336425e+02	1.596513e+05
r_f032m	2049032.0	0.000000e+00	1.489238e+03	-11872.9951	-6.517037e+02	-6.576680e+01	5.334434e+02	1.596468e+05
r_f033m	2029771.0	1.000000e-04	1.488224e+03	-11889.0117	-6.510807e+02	-6.590900e+01	5.329744e+02	1.596441e+05
r_f034m	2010873.0	1.000000e-04	1.484454e+03	-11889.4648	-6.505530e+02	-6.600180e+01	5.321952e+02	1.339924e+05
r_f035m	1991985.0	0.000000e+00	1.481871e+03	-11888.5801	-6.497461e+02	-6.637680e+01	5.308284e+02	1.339871e+05
r_f036m	1973277.0	-0.000000e+00	1.480547e+03	-11901.7588	-6.487253e+02	-6.560970e+01	5.307343e+02	1.339843e+05
r_f037m	1954719.0	0.000000e+00	1.474733e+03	-11808.9707	-6.477959e+02	-6.532040e+01	5.303346e+02	1.105830e+05
r_f038m	1936335.0	0.000000e+00	1.470288e+03	-11795.4795	-6.465491e+02	-6.499620e+01	5.294399e+02	1.105891e+05
r_f039m	1918045.0	-0.000000e+00	1.466709e+03	-11661.7285	-6.456418e+02	-6.549740e+01	5.287186e+02	1.105892e+05
r_f040m	1900119.0	0.000000e+00	1.464523e+03	-11641.5117	-6.446006e+02	-6.542780e+01	5.281598e+02	1.105945e+05
r_f041m	1882204.0	-0.000000e+00	1.461618e+03	-11620.1201	-6.434552e+02	-6.541130e+01	5.274347e+02	1.105990e+05
r_f042m	1864406.0	0.000000e+00	1.461010e+03	-11649.9248	-6.426281e+02	-6.453170e+01	5.271555e+02	1.105945e+05
r_f043m	1846808.0	0.000000e+00	1.458799e+03	-11647.5039	-6.419841e+02	-6.478650e+01	5.264048e+02	1.105988e+05
r_f044m	1829434.0	1.000000e-04	1.456109e+03	-11657.0537	-6.405219e+02	-6.464430e+01	5.260392e+02	1.105983e+05
r_f045m	1812186.0	0.000000e+00	1.455769e+03	-11657.0459	-6.401905e+02	-6.447610e+01	5.255322e+02	1.105999e+05
r_f046m	1795157.0	-0.000000e+00	1.457390e+03	-11652.4551	-6.403130e+02	-6.455670e+01	5.250655e+02	1.105960e+05
r_f047m	1778282.0	-0.000000e+00	1.460243e+03	-11658.8438	-6.401207e+02	-6.554230e+01	5.242169e+02	1.105929e+05
r_f048m	1761604.0	-0.000000e+00	1.455848e+03	-11682.4463	-6.391160e+02	-6.477180e+01	5.234120e+02	1.105901e+05
r_f049m	1745155.0	1.000000e-04	1.455978e+03	-11670.7920	-6.382848e+02	-6.515990e+01	5.229612e+02	1.105818e+05
r_f050m	1728738.0	0.000000e+00	1.454388e+03	-11632.7188	-6.375092e+02	-6.496860e+01	5.226442e+02	1.105831e+05
r_f051m	1712469.0	1.000000e-04	1.452841e+03	-11632.8320	-6.363624e+02	-6.464440e+01	5.219601e+02	1.105796e+05
r_f052m	1696257.0	-0.000000e+00	1.447684e+03	-11634.2344	-6.354518e+02	-6.441250e+01	5.216530e+02	1.105797e+05
r_f053m	1680291.0	0.000000e+00	1.446640e+03	-11599.9922	-6.352231e+02	-6.461350e+01	5.207815e+02	1.105815e+05
r_f054m	1664527.0	-0.000000e+00	1.447881e+03	-11610.1670	-6.343800e+02	-6.448220e+01	5.200620e+02	1.105856e+05
r_f055m	1648942.0	-0.000000e+00	1.441350e+03	-11593.4238	-6.328156e+02	-6.362880e+01	5.197030e+02	1.105916e+05
r_f056m	1633612.0	0.000000e+00	1.439244e+03	-11559.6719	-6.322285e+02	-6.335630e+01	5.195420e+02	1.105924e+05
r_f057m	1618435.0	-0.000000e+00	1.437575e+03	-11520.1182	-6.320952e+02	-6.346460e+01	5.184686e+02	1.105943e+05
r_f058m	1603369.0	0.000000e+00	1.432364e+03	-11498.3047	-6.313463e+02	-6.371530e+01	5.178313e+02	1.105877e+05
r_f059m	1588459.0	0.000000e+00	1.432379e+03	-11524.8418	-6.294853e+02	-6.328710e+01	5.172325e+02	1.105811e+05
r_f060m	1573646.0	-1.000000e-04	1.425208e+03	-11522.6631	-6.281742e+02	-6.214360e+01	5.171097e+02	1.105770e+05
r_f061m	1558831.0	0.000000e+00	1.423086e+03	-11512.3076	-6.275454e+02	-6.188950e+01	5.169286e+02	1.105791e+05
r_f062m	1544091.0	0.000000e+00	1.421962e+03	-11506.7881	-6.266454e+02	-6.163530e+01	5.167087e+02	1.105750e+05
r_f063m	1529580.0	-0.000000e+00	1.421226e+03	-11490.2002	-6.257752e+02	-6.118430e+01	5.159457e+02	1.105775e+05
r_f064m	1515298.0	0.000000e+00	1.420464e+03	-11447.7783	-6.246249e+02	-6.092020e+01	5.155374e+02	1.105721e+05
r_f065m	1501145.0	0.000000e+00	1.414488e+03	-11448.2666	-6.226816e+02	-6.009660e+01	5.149099e+02	1.105906e+05
r_f066m	1487126.0	0.000000e+00	1.409900e+03	-11429.1436	-6.217508e+02	-6.041790e+01	5.136979e+02	1.105791e+05

	count	mean	std	min	25%	50%	75%	max
r_f067m	1473347.0	-0.000000e+00	1.410359e+03	-11395.8105	-6.213913e+02	-5.986390e+01	5.135056e+02	1.105783e+05
r_f068m	1459783.0	-0.000000e+00	1.408494e+03	-11416.6279	-6.212914e+02	-5.952610e+01	5.132261e+02	1.105899e+05
r_f069m	1446378.0	-0.000000e+00	1.407508e+03	-11403.6064	-6.206971e+02	-6.046170e+01	5.122022e+02	1.105804e+05
r_f070m	1433053.0	-0.000000e+00	1.407457e+03	-11369.1240	-6.195969e+02	-6.033520e+01	5.112055e+02	1.105780e+05
r_f071m	1419806.0	-0.000000e+00	1.405462e+03	-11368.4258	-6.189065e+02	-6.046740e+01	5.107696e+02	1.105753e+05
r_f072m	1406604.0	-0.000000e+00	1.400772e+03	-11371.8594	-6.173835e+02	-5.969520e+01	5.105945e+02	1.105796e+05
r_f073m	1393602.0	-0.000000e+00	1.400693e+03	-11396.2529	-6.175958e+02	-6.000100e+01	5.098098e+02	1.105802e+05
r_f074m	1380833.0	0.000000e+00	1.402398e+03	-11372.6416	-6.169977e+02	-6.035980e+01	5.092791e+02	1.105740e+05
r_f075m	1368264.0	0.000000e+00	1.403089e+03	-11405.9805	-6.166624e+02	-6.033900e+01	5.087950e+02	1.105671e+05
r_f076m	1355788.0	0.000000e+00	1.397425e+03	-11382.0527	-6.149069e+02	-5.957630e+01	5.083461e+02	1.105749e+05
r_f077m	1343377.0	0.000000e+00	1.395999e+03	-11382.4336	-6.135413e+02	-5.895680e+01	5.081946e+02	1.105644e+05
r_f078m	1331081.0	0.000000e+00	1.391616e+03	-11378.7529	-6.125250e+02	-5.866520e+01	5.075152e+02	1.105711e+05
r_f079m	1318964.0	-0.000000e+00	1.388824e+03	-11362.2051	-6.116483e+02	-5.816850e+01	5.073903e+02	1.105640e+05
r_f080m	1307052.0	-0.000000e+00	1.386874e+03	-11357.9014	-6.107534e+02	-5.763730e+01	5.068637e+02	1.105745e+05
r_f081m	1295317.0	0.000000e+00	1.381056e+03	-11383.5312	-6.103754e+02	-5.766870e+01	5.064513e+02	1.105647e+05
r_f082m	1283667.0	0.000000e+00	1.383347e+03	-11365.5000	-6.103617e+02	-5.777960e+01	5.060737e+02	1.105658e+05
r_f083m	1272047.0	-0.000000e+00	1.374520e+03	-11384.9004	-6.086416e+02	-5.797640e+01	5.042909e+02	1.105673e+05
r_f084m	1260498.0	0.000000e+00	1.373337e+03	-11363.2959	-6.076294e+02	-5.762950e+01	5.040967e+02	1.105768e+05
r_f085m	1249073.0	-0.000000e+00	1.374082e+03	-11402.5264	-6.065834e+02	-5.695480e+01	5.032051e+02	1.105750e+05
r_f086m	1237781.0	0.000000e+00	1.370850e+03	-11400.8906	-6.059218e+02	-5.657530e+01	5.023136e+02	1.105727e+05
r_f087m	1226562.0	-1.000000e-04	1.370446e+03	-11414.5410	-6.048359e+02	-5.619380e+01	5.021124e+02	1.105748e+05
r_f088m	1215483.0	0.000000e+00	1.369352e+03	-11404.8936	-6.043073e+02	-5.630440e+01	5.014727e+02	1.105904e+05
r_f089m	1204483.0	0.000000e+00	1.370709e+03	-11400.6182	-6.037288e+02	-5.706160e+01	5.004432e+02	1.105700e+05
r_f090m	1193700.0	0.000000e+00	1.367878e+03	-11385.3594	-6.028849e+02	-5.712190e+01	4.993200e+02	1.105744e+05
r_f091m	1183030.0	-0.000000e+00	1.363581e+03	-11358.1553	-6.016176e+02	-5.622980e+01	4.984420e+02	1.105727e+05
r_f092m	1172486.0	0.000000e+00	1.360579e+03	-11309.1123	-6.002883e+02	-5.573990e+01	4.975717e+02	1.105593e+05
r_f093m	1162055.0	0.000000e+00	1.358095e+03	-11357.9609	-5.987974e+02	-5.531470e+01	4.968514e+02	1.105644e+05
r_f094m	1151736.0	0.000000e+00	1.352453e+03	-11317.3789	-5.971700e+02	-5.476540e+01	4.963483e+02	1.105683e+05
r_f095m	1141522.0	0.000000e+00	1.348188e+03	-11309.7559	-5.962652e+02	-5.471390e+01	4.951203e+02	1.105691e+05
r_f096m	1131442.0	-0.000000e+00	1.342258e+03	-11309.1016	-5.944086e+02	-5.403380e+01	4.948370e+02	1.105662e+05
r_f097m	1121493.0	-0.000000e+00	1.338318e+03	-11247.9277	-5.932616e+02	-5.396110e+01	4.937179e+02	1.105643e+05
r_f098m	1111606.0	-0.000000e+00	1.338118e+03	-11261.6719	-5.919281e+02	-5.361060e+01	4.932585e+02	1.105735e+05
r_f099m	1101788.0	-0.000000e+00	1.334361e+03	-11223.1240	-5.905834e+02	-5.352240e+01	4.921118e+02	1.105733e+05
r_f100m	1092121.0	-0.000000e+00	1.331728e+03	-11206.9795	-5.900486e+02	-5.297250e+01	4.916311e+02	1.105714e+05
r_f101m	1082539.0	0.000000e+00	1.332157e+03	-11283.8125	-5.892725e+02	-5.268540e+01	4.909461e+02	1.105668e+05
r_f102m	1072947.0	-0.000000e+00	1.332334e+03	-11282.4170	-5.881101e+02	-5.258020e+01	4.907911e+02	1.105829e+05

	count	mean	std	min	25%	50%	75%	max
r_f103m	1063428.0	0.000000e+00	1.329495e+03	-11285.6172	-5.869788e+02	-5.248610e+01	4.906705e+02	1.105661e+05
r_f104m	1053983.0	0.000000e+00	1.325444e+03	-11298.3848	-5.862903e+02	-5.239650e+01	4.899225e+02	1.105618e+05
r_f105m	1044601.0	0.000000e+00	1.323975e+03	-11301.4951	-5.851770e+02	-5.231380e+01	4.887940e+02	1.105658e+05
r_f106m	1035321.0	0.000000e+00	1.318717e+03	-11297.5840	-5.832391e+02	-5.184850e+01	4.882903e+02	1.105617e+05
r_f107m	1026131.0	-0.000000e+00	1.311274e+03	-11285.4033	-5.818715e+02	-5.133990e+01	4.878159e+02	1.105708e+05
r_f108m	1017036.0	0.000000e+00	1.305877e+03	-11274.5352	-5.799500e+02	-5.048810e+01	4.874828e+02	1.105686e+05
r_f109m	1007990.0	1.000000e-04	1.294919e+03	-11275.3604	-5.771514e+02	-4.921990e+01	4.869372e+02	1.105662e+05
r_f110m	999001.0	-0.000000e+00	1.289170e+03	-11268.4707	-5.755809e+02	-4.920890e+01	4.857903e+02	1.105673e+05
r_f111m	990052.0	-0.000000e+00	1.290836e+03	-11281.2275	-5.746790e+02	-4.955900e+01	4.840651e+02	1.105661e+05
r_f112m	981156.0	0.000000e+00	1.287458e+03	-11282.5312	-5.735778e+02	-4.994070e+01	4.832928e+02	1.105664e+05
r_f113m	972370.0	-1.000000e-04	1.284069e+03	-11264.9014	-5.725287e+02	-4.965830e+01	4.823773e+02	1.105705e+05
r_f114m	963718.0	0.000000e+00	1.278476e+03	-11253.8916	-5.716486e+02	-4.986240e+01	4.817620e+02	1.105646e+05
r_f115m	955145.0	0.000000e+00	1.275283e+03	-11264.3701	-5.702315e+02	-4.919280e+01	4.812133e+02	1.105725e+05
r_f116m	946626.0	-0.000000e+00	1.272883e+03	-11263.4824	-5.686965e+02	-4.945870e+01	4.802682e+02	1.105707e+05
r_f117m	938160.0	0.000000e+00	1.274644e+03	-11257.9434	-5.681894e+02	-5.015010e+01	4.790787e+02	1.105761e+05
r_f118m	929764.0	0.000000e+00	1.270645e+03	-11259.9062	-5.671390e+02	-5.087480e+01	4.773846e+02	1.105775e+05
r_f119m	921481.0	-0.000000e+00	1.270319e+03	-11240.6992	-5.655049e+02	-5.112130e+01	4.756199e+02	1.105739e+05
r_f120m	913287.0	-0.000000e+00	1.257670e+03	-11251.2500	-5.629036e+02	-4.977740e+01	4.754564e+02	1.105857e+05
train	2739928.0	3.998000e-01	4.994000e-01	0.0000	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00
test	2739928.0	6.002000e-01	4.994000e-01	0.0000	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
dev	2739928.0	3.998000e-01	4.994000e-01	0.0000	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00

Table 2: Summary Statistics

11.2 Technical Details

11.2.1 Organisation

This research essay uses data science best practise (**J:10**). Data and results saved regularly and reproducible. Data retention in all forms receives high levels of attention. Project files synchonises continuously to Google Drive (**Google˙Drive**). Git (**Git**) manages version control protocols for source code, data, documents, and results. Git stores a complete history of versions using Git hashes. These hashes are strings unique to each state of the publicly available finance-honours repository[1]. Git hashes enable discretisation of finance-honours development, enabling the accessibility and recollection of all previous states given a unique git hash. This functionality enables reproducibility, error correction, and the ability to revert to previous models.

11.2.2 Version Control

Git, hosted by GitHub, provides a comprehensive set of version control technologies and range of benefits. Firstly, Git enables collaborative functionalities. The master version of a project is accessible for all who have access to the repository. Each contributor can create custom copies of branches through pull requests on the master branch. Contributors can commit changes to custom branches and push these changes to the master branch through push requests. Product managers can review push requests, approving valid requests for integrating changes to the master branch. Collaborative efforts are possible with commit messages describing contributions from each contributor. This research essay has only one contributor, rendering collaborative functionalities redundant in this instance. Git ensures the storage of code, work, and author histories. The descriptive nature of commit logs ensures journal accuracy.

11.2.3 Directories

This research essay follows directory structure recommendations from Wilson et al (**J:10**). Organisation is crucial as the modelling of artificial neural networks involves integrating a range of optimisation models, data files and documents. Directory management is most efficient and comprehensive. **finance-honours** is the root directory containing the following sub directories: bin, data, doc, src, and results. The **bin** sub directory contains external scripts and compiled programmes. The **data** sub directory contains all raw data associated with the project. The **doc** sub directory stores user guides, academic resources, research reports and project deliverables. The **results** sub directory contains the outputs from project analysis. The **src** sub directory stores the source code for preparing datasets, partitioning sets of geographies with varying granularities. All files were continuously backed up using Google Drive and Git.

11.2.4 Python

Python 3.9.7 is the primary programming language for this research essay. The language is omnipresent, widespread in software development. Python's language design makes the language highly productive and simple to use. Python can hand off computationally straining tasks to C/C++ using supporting first-class integration capabilities. The language also has a very active and supportive community. Python is the most popular coding language on the planet defined by the PYPL PopularitY of Programming Language Index. As at December 2021, Python has 30.21% of all language tutorial search instances on Google (**PYPL˙Pop**). Python's dynamic, low cost, and open source nature makes programming quick.

11.2.5 Package Management

The Anaconda package management platform for Python (**Anaconda**) is the chosen coding environment. Anaconda is a well defined, free platform, with known versions of python packages such as matplotlib, numpy, and pip. The use of this environment ensures reproducibility and consistency across infrastructure. Pip is the default package manager for Python, included in the Anaconda package. Pip manages package installation and updates.

[1]https://github.com/CMCD1996/finance-honours

11.2.6 Code Style

The PEP8 style for Python Code is formatting style for development code **PEP8**. Yapf, a formatter maintained by Google, manages formatting. Standardised formatting is important as makes supports readability, optimisation, and consistency. Docstrings and rigourous commenting are important in documentation. A docstring is a Python inline comment describing function use, inputs, and outputs. An unique docstring belongs to each Python class and function. The Google style docstring is most appropriate because of it's readability, writing ease, and consistency with Google's Style Guide. The parsing of yapf docstrings enables automated documentation generators to create docstring documents describing functions and classes.

11.2.7 Infrastructure

This research essay deploys variations in artificial neural networks of changing size and complexity. Analysis either took place locally, or remotely, depending on the computational requirements for the particular analysis. An Apple MacBook Pro 13 Inch 2019 with 8 GB 2133 MHz LPDDR3 memory and 1.4 GHz Quad-Core Intel Core i5 processor handles simple tasks locally. A Virtual Machine Instance on the Google Cloud Platform **Insert specification before submission** handles complex tasks remotely.

11.2.8 Documentation

The research essay documentation keeps an accurate record of key design decisions. Commit histories (**??**) is the most important form of documentation. Application of auxiliary documentation methods are supplementary.

11.3 Code

All files, resources, and code is available for download from Github. The document listing function and class docstring is available for download here. Furthermore, the coding listings for this research essay follow.

```python
################################################################################
# Module Imports
################################################################################
# System
import psutil as ps  # Monitor CPU usage
import nvidia_smi  # Monitor GPU usage
import os  # Change/manipulate operating systems
import datetime as dt  # Manipulate datetime values
import random as rd  # Random functionality
import csv as csv  # Read and write csvs
import itertools as it  # Create iterators for efficient looping
# Analytical
from pandas.core.base import NoNewAttributesMixin
import sympy as sym  # Symbolic package for calculus
# Machine Learning/AI/Statistics
import numpy as np
from numpy.core.fromnumeric import transpose  # Arithmetic operations
import pandas as pd  # Data analysis package
import dask as ds  # Data importing for very large software packages.
import seaborn as sb  # Imports seaborn library for use
import sklearn as skl  # Simple statistical models
from sklearn.model_selection import train_test_split
import tensorflow as tf  # Tensorflow (https://www.tensorflow.org/)
from tensorflow.keras import layers
from tensorflow.python.ops.gen_array_ops import split  # Find combinations of lists
# Keras backend functions to design custom metrics
import tensorflow.keras.backend as K
import linearmodels as lp  # Ability to use PooledOLS
from statsmodels.regression.rolling import RollingOLS  # Use factor loadings
from keras.callbacks import Callback  # Logging training performance
import neptune.new as neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor
# APIs
import wrds as wrds  # Wharton Research Data Services API
import pydatastream as pds  # Thomas Reuters Datastream API
import yfinance as yf  # Yahoo Finance API
import finance_byu as fin  # Python Package for Fama-MacBeth Regressions
import saspy as sas  # Use saspy functionality in python
import statsmodels.api as sm  # Create Stats functionalities
# Formatting/Graphing
import tabulate as tb  # Create tables in python
import pydot as pyd  # Dynamically generate graphs
import matplotlib.pyplot as plt  # Simple plotting
import scipy as sc  # Scipy packages
# Stargazor package to lm latex tables
from stargazer.stargazer import Stargazer
#################################################################################
# Function Calls
#################################################################################
# System Functions
#################################################################################


def monitor_memory_usage(units, cpu=False, gpu=False):
    """ Function to monitor both CPU & GPU memory consumption

    Args:
        units (int): Memory units (0 = Bytes, 1 = KB, 2 = MB, 3 = GB, 4 = TB, 5 = PB)
        cpu (bool, optional): CPU Information. Defaults to False.
        gpu (bool, optional): GPU Information. Defaults to False.
    """
    # Set unit conversion for readability
    convertor = (1024**units)
    # Shows CPU information using psutil
    if cpu:
        cpu_f = (ps.virtual_memory().available)/convertor
        cpu_t = (ps.virtual_memory().total)/convertor
        cpu_u = (ps.virtual_memory().used)/convertor
        cpu_fp = (ps.virtual_memory().available *
                  100 / ps.virtual_memory().total)
        print("CPU - Memory : ({:.2f}% free): {}(total), {} (free), {} (used)".format(
            cpu_fp, cpu_t, cpu_f, cpu_u))
        # Shows GPU information using nvidia-ml-py3
    if gpu:
        print("GPU Memory Summary")
        nvidia_smi.nvmlInit()
        deviceCount = nvidia_smi.nvmlDeviceGetCount()
        for i in range(deviceCount):
            # Gets device handle
            handle = nvidia_smi.nvmlDeviceGetHandleByIndex(i)
            # Uses handle to get GPU device info
            info = nvidia_smi.nvmlDeviceGetMemoryInfo(handle)
            # Prints GPU information
            print("GPU - Device {}: {}, Memory : ({:.2f}% free): {}(total), {} (free),
    {} (used)".format(i, nvidia_smi.nvmlDeviceGetName(
                handle), 100*info.free/info.total, info.total/convertor, info.free//
    convertor, info.used/convertor))
        nvidia_smi.nvmlShutdown()
    return


def reconfigure_gpu(restrict_tf, growth_memory):
    # Check the number of GPUs avaiable to Tensorflow and in use
    print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
    # Limit tf to a specfic set of GO devices
    gpus = tf.config.list_physical_devices('GPU')
    # Restrict TensorFlow to only use the first GPU
    if gpus and restrict_tf:
        try:
            tf.config.set_visible_devices(gpus[0], 'GPU')
            logical_gpus = tf.config.list_logical_devices('GPU')
            print(len(gpus), "Physical GPUs,",
                  len(logical_gpus), "Logical GPU")
        except RuntimeError as e:
            # Visible devices must be set before GPUs have been initialized
            print(e)
    # Limit GPU Memory Growth
    if gpus and growth_memory:
        try:
            # Currently, memory growth needs to be the same across GPUs
            for gpu in gpus:
                tf.config.experimental.set_memory_growth(gpu, True)
            logical_gpus = tf.config.list_logical_devices('GPU')
            print(len(gpus), "Physical GPUs,", len(
                logical_gpus), "Logical GPUs")
        except RuntimeError as e:
            # Memory growth must be set before GPUs have been initialized
            print(e)
    return


def configure_training_ui(project, api_token):
    # Monitor Keras loss using callback
    # https://app.neptune.ai/common/tf-keras-integration/e/TFK-35541/dashboard/metrics-
    b11ccc73-9ac7-4126-be1a-cf9a3a4f9b74
    # Initialise neptune with credientials
    run = neptune.init(project=project, api_token=api_token)
    # project - 'connormcdowall/finance-honours')
    # api_token  = '
    eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vYXBwLm5lcHR1bmUuYWkiLCJhcGlfdXJsIjoiaHR0cHM6Ly9hcHAubmVwdHVuZS5haSI
    =='
    # Define the custom class for the function

    class NeptuneCallback(Callback):
        def on_batch_end(self, batch, logs=None):
            for metric_name, metric_value in logs.items():
                run[f"{metric_name}"].log(metric_value)

        def on_epoch_end(self, epoch, logs=None):
            for metric_name, metric_value in logs.items():
                run[f"{metric_name}"].log(metric_value)
    # Find the call back
    neptune_cbk = NeptuneCallback(run=run, base_namespace='metrics')
    # Example to set paramters
    # run["JIRA"] = "NPT-952"
    # run["parameters"] = {"learning_rate": 0.001,
    #                      "optimizer": "Adam"}
    # run["f1_score"] = 0.66
    # Example in using in model callback
    # model.fit(x_train, y_train,
    #         validation_split=0.2,
    #         epochs=10,
    #         callbacks=[neptune_cbk])
    # Returns Callback APIs
    return neptune_cbk
#################################################################################
# Data Processing
#################################################################################


def partition_data(data_location, data_destination):
    """ Converts dta  format to a series of 100k line csvs

    Args:
        data_location (str): directory of dta file
        data_destination (str):
    """

    # Converts dta file to chunks
    dflocation = data_destination
    data = pd.read_stata(data_location, chunksize=100000)
    num = 1
    for chunk in data:
        # Saves chunck to seperate csvs given dataset size
        df = pd.DataFrame()
        df = df.append(chunk)
        df.to_csv(dflocation + str(num) + '.csv')
        num_convert = num*100000
        print('Number of rows converted: ', num_convert)
        num = num + 1
    return
```

```python


def create_dataframes(csv_location, multi_csv):
    """ Function to create
    """
    # Creates list of dataframes
    num_csvs = list(range(1, 29, 1))
    if multi_csv == False:
        df = pd.read_csv(csv_location + "1.csv")
        # Show frame information
        show_info = False
        if show_info == True:
            # Prints df head, info, columns
            print('information on Dataframe')
            print(df.info())
            print('Dataframe Head')
            print(df.head())
            print('Dataframe Columns')
            print(df.columns)
            # Saves columns as list in txt file
            np.savetxt(r'/Users/connor/Google Drive/Documents/University/Courses
    /2020-21/Finance 788/finance-honours/data/dataframe-columns.txt', df.columns, fmt='%
    s')
        # Save summary statistics to dataframe
        data_stats = df.describe().round(4)
        data_stats.T.to_latex('results/tables/subset-summary-statistics.txt')
        return df
        # Pre-process dataframe for suitability (Remove empty rows, columns etc.)
    else:
        df_list = []
        for num in num_csvs:
            df = pd.read_csv(csv_location + str(num) + ".csv")
            # Append all the dataframes after reading the csv
            df_list.append(df)
            # Concantenate into one dataframe
            df = pd.concat(df_list)
        # Save summary statistics to dataframe
        data_stats = df.describe().round(4)
        data_stats.T.to_latex('results/tables/subset-summary-statistics.txt')
        return df


def sass_access(dataframe):
    # Two files are accessed once for reference
    # sascfg_personal is a configuration file for accessing SAS Ondemand Academic
    Packages
    '/opt/anaconda3/lib/python3.7/site-packages/saspy'
    # SAS User credientials for granting access
    '/Users/connor/.authinfo'
    # Enable SAS Connection
    session = sas.SASsession()
    # Create sass data
    data = session.dataframe2sasdata(dataframe)
    # Display summary statistics for the data
    data.means()
    return


def replace_nan(df, replacement_method):
    """ Replace/Remove nan files in a dataframe

    Args:
        df (dataframe): Pandas Dataframe
        replacement_method (int): Specify replacement methods
                                : 0 - remove rows with nan values
                                : 1 - remove columns with nan values
                                : 2 - fill nan with column mean
                                : 3 - fill nan with column median
    Returns:
        dataframe: Updated pandas dataframe
    """
    nan_total = df.isnull().sum().sum()
    print('Number of nan values before processing: ', nan_total)
    if nan_total > 0:
        # Replace dataframe level nan (rows or columns)
        # Replacement methods (0: remove rows with nan values, medium, remove, none)
        if replacement_method == 0:
            df.dropna(axis=0, how='any', inplace=True)
        # Caution: Change to dataframe-columns.txt and features list required (Do not
    use)
        if replacement_method == 1:
            df.dropna(axis=1, how='any', inplace=True)
        # Replace column level nan
        for column in df.columns:
            if df[column].isnull().sum() > 0:
                if replacement_method == 2:
                    df[column].fillna(df[column].mean(), inplace=True)
                elif replacement_method == 3:
                    df[column].fillna(df[column].median(), inplace=True)
    nan_total = df.isnull().sum().sum()
    print('Number of nan values after processing: ', nan_total)
    return df


def reduce_mem_usage(props):
    """ Function reducing the memory size of a dataframe from Kaggle
        https://www.kaggle.com/arjanso/reducing-dataframe-memory-size-by-65

    Args:
        props (dataframe): Pandas Dataframe

    Returns:
        props (dataframe): Resized Pandas Dataframe
    """
    # Begin the resizing function
    start_mem_usg = props.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :", start_mem_usg, " MB")
    NAlist = []  # Keeps track of columns that have missing values filled in.
    for col in props.columns:
        if props[col].dtype != object:  # Exclude strings

            # Print current column type
            print("******************************")
            print("Column: ", col)
            print("dtype before: ", props[col].dtype)

            # make variables for Int, max and min
            IsInt = False
            mx = props[col].max()
            mn = props[col].min()

            # Integer does not support NA, therefore, NA needs to be filled
            if not np.isfinite(props[col]).all():
                NAlist.append(col)
                props[col].fillna(mn-1, inplace=True)

            # test if column can be converted to an integer
            asint = props[col].fillna(0).astype(np.int64)
            result = (props[col] - asint)
            result = result.sum()
            if result > -0.01 and result < 0.01:
                IsInt = True

            # Make Integer/unsigned Integer datatypes
            if IsInt:
                if mn >= 0:
                    if mx < 255:
                        props[col] = props[col].astype(np.uint8)
                    elif mx < 65535:
                        props[col] = props[col].astype(np.uint16)
                    elif mx < 4294967295:
                        props[col] = props[col].astype(np.uint32)
                    else:
                        props[col] = props[col].astype(np.uint64)
                else:
                    if mn > np.iinfo(np.int8).min and mx < np.iinfo(np.int8).max:
                        props[col] = props[col].astype(np.int8)
                    elif mn > np.iinfo(np.int16).min and mx < np.iinfo(np.int16).max:
                        props[col] = props[col].astype(np.int16)
                    elif mn > np.iinfo(np.int32).min and mx < np.iinfo(np.int32).max:
                        props[col] = props[col].astype(np.int32)
                    elif mn > np.iinfo(np.int64).min and mx < np.iinfo(np.int64).max:
                        props[col] = props[col].astype(np.int64)

            # Make float datatypes 32 bit
            else:
                props[col] = props[col].astype(np.float32)

            # Print new column type
            print("dtype after: ", props[col].dtype)
            print("******************************")

    # Print final result
    print("___MEMORY USAGE AFTER COMPLETION:___")
    mem_usg = props.memory_usage().sum() / 1024**2
    print("Memory usage is: ", mem_usg, " MB")
    print("This is ", 100*mem_usg/start_mem_usg, "% of the initial size")
    return props, NAlist


def resizing_dataframe(dataframe, resizing_options):
    print(dataframe.head())
    # Remove both micro
    if resizing_options[0]:
        print('Reducing number of size_grp entries')
        indexNames = dataframe[(dataframe['size_grp'] == 'micro') | (
            dataframe['size_grp'] == 'nano')].index
        dataframe.drop(indexNames, inplace=True)
        print(dataframe.info())
        monitor_memory_usage(units=3, cpu=True, gpu=True)
    # Reduce the number of factors to the original ~178 from JKP
    if resizing_options[1]:
        print('Reducing number of factors to original ~178 from JKP')
        # Extract new columns to the dataframe
        new_columns = []
        list_of_columns = '/home/connormcdowall/finance-honours/data/178-factors.txt'
        file = open(list_of_columns, 'r')
        lines = file.readlines()
        for line in lines:
            line = line.rstrip('\n')
            new_columns.append(line)
        # Only collect column in both lists
        cols = dataframe.columns
        extract_columns = []
        for column in new_columns:
            if column in cols:
                extract_columns.append(column)
        # Extract the old columns
        dataframe = dataframe[extract_columns]
        # Rewrite new working file for numerical encoding
        file = open(
            "/home/connormcdowall/finance-honours/data/working-columns.txt", "r+")
        file.truncate(0)
        file.close()
        textfile = open(
            "/home/connormcdowall/finance-honours/data/working-columns.txt", "w")
        for element in extract_columns:
            textfile.write(element + "\n")
        textfile.close()
        monitor_memory_usage(units=3, cpu=True, gpu=True)
    # Optimises Variable Type
    if resizing_options[2]:
        print('Optimise variable type configuration')
        dataframe, NAlist = reduce_mem_usage(dataframe)
        monitor_memory_usage(units=3, cpu=True, gpu=True)
    return dataframe
```

```python
def split_vm_dataset(data_vm_directory, create_statistics, split_new_data,
    create_validation_set):
    """ Creates summmary statistics from unprocessed dataset

    Args:
        data_vm_directory (str): Directory location of data stored on the VM instance.
    """
    # Create Dataframe from the entire dataset
    # total_df = pd.read_stata(data_vm_directory + 'combined_predictors_filtered_us.dta
    ')
    # Create summary statisitics for the entire dataset
    if create_statistics == True:
        # Read data into one dataframe on python
        total_df = pd.read_stata(
            data_vm_directory + 'combined_predictors_filtered_us.dta')
        data_stats = total_df.describe().round(4)
        data_stats.T.to_latex('results/tables/summary-statistics.txt')
    # Create training and testing dataframes for Tensorflow
    if split_new_data == True:
        train_df = pd.DataFrame()
        test_df = pd.DataFrame()
        total_df = pd.read_stata(
            data_vm_directory + 'combined_predictors_filtered_us.dta', chunksize=100000)
        for chunk in total_df:
            test_df = test_df.append(chunk[chunk["test"] == 1])
        # Split training set into training and validation
        if create_validation_set == True:
            train_new_df, val_df = train_test_split(train_df, test_size=0.2)
            print(train_df.info())
            print(val_df.info())
            train_new_df.to_stata(data_vm_directory + 'train.dta')
            val_df.to_stata(data_vm_directory + 'val.dta')
        else:
            train_df.to_stata(data_vm_directory + 'train.dta')
        test_df.to_stata(data_vm_directory + 'test.dta')
    return


def process_vm_dataset(data_vm_dta, size_of_chunks, resizing_options, save_statistics=
    False, sample=False):
    """ This script processes the training and testing datasets for Tensorflow
    following the classify structured data with feature columns tutorial
    """
    # Load the test and train datasets into dataframes in chunks
    #df = pd.read_stata(data_vm_dta)
    subset = pd.read_stata(data_vm_dta, chunksize=size_of_chunks)
    df_full = pd.DataFrame()
    for df in subset:
        print('Number of instances: ', len(df))
        print('Excess Return')
        print(df['ret_exc'])
        # Find the dtypes of the dataframe and save them to a data column
        if save_statistics:
            # Saves dtypes for column dataframe
            np.savetxt(
                r'/home/connormcdowall/finance-honours/results/statistics/factor-types.
    txt', df.dtypes, fmt='%s')
            # Saves information on missing values in the dataframe
            np.savetxt(
                r'/home/connormcdowall/finance-honours/results/statistics/missing-values
    .txt', df.isna().sum(), fmt='%s')
        # Gets list of dataframe column values
        column_list = list(df.columns.values)
        # Gets list of unique dataframe dtype
        data_type_list = list(df.dtypes.unique())
        # Gets unique list of size_grp
        size_grp_list = list(df['size_grp'].unique())
        # Removes the mth column/factor from the dataframe given datatime format
        df['mth'] = pd.to_numeric(df['mth'], downcast='float')
        df_full = df_full.append(df)
        # Prints memory usage after the process
        monitor_memory_usage(units=3, cpu=True, gpu=True)
        if sample:
            # Process nan options in the dataframe
            df_full = replace_nan(df_full, replacement_method=3)
            # Resizes the dataframe base on memory options
            df_full = resizing_dataframe(
                dataframe=df_full, resizing_options=resizing_options)
            # Print size and shape of dataframe
            print('The dataframe has {} entries with {} rows and {} columns.'.format(
                df_full.size, df_full.shape[0], df_full.shape[1]))
            return df_full
    # Prints size categories in dataframe
    size_grp_list = list(df['size_grp'].unique())
    print('List of size_grp variables')
    print(size_grp_list)
    # Checks Nan in dataframe
    df_full = replace_nan(df_full, replacement_method=3)
    # Memory resizing to prevent excessive memory consumption
    df_full = resizing_dataframe(
        dataframe=df_full, resizing_options=resizing_options)
    # Print size and shape of dataframe
    print('The dataframe has {} entries with {} rows and {} columns.'.format(
        df_full.size, df_full.shape[0], df_full.shape[1]))
    # Prints memory usage after the process
    monitor_memory_usage(units=3, cpu=True, gpu=True)
    return df_full

################################################################################
# Machine Learning
################################################################################
# Utility method to use pandas dataframe to create a tf.data dataset
# Adapted from https://www.tensorflow.org/tutorials/structured_data/feature_columns#
    use_pandas_to_create_a_dataframe
# Adapted from https://www.tensorflow.org/tutorials/structured_data/preprocessing_layers


def download_test_data():
    dataset_url = 'http://storage.googleapis.com/download.tensorflow.org/data/petfinder-
    mini.zip'
    csv_file = 'datasets/petfinder-mini/petfinder-mini.csv'
    tf.keras.utils.get_file('petfinder_mini.zip', dataset_url,
                            extract=True, cache_dir='.')
    dataframe = pd.read_csv(csv_file)

    # Creates the target variable for the assignment
    dataframe['target'] = np.where(dataframe['AdoptionSpeed'] == 4, 0, 1)
    # Drop unused features.
    dataframe = dataframe.drop(columns=['AdoptionSpeed', 'Description'])
    # Split the dataset into training, validation and testing sets
    train, val, test = np.split(dataframe.sample(
        frac=1), [int(0.8*len(dataframe)), int(0.9*len(dataframe))])
    # Returns the dataframe and the three subsets
    return dataframe, train, val, test


def create_feature_lists(list_of_columns, categorical_assignment):
    # Assignn variables
    categorical_features = []
    numerical_features = []
    file = open(list_of_columns, 'r')
    lines = file.readlines()
    for line in lines:
        line = line.rstrip('\n')
        if line in categorical_assignment:
            categorical_features.append(line)
        else:
            numerical_features.append(line)
    # Returns numerical and categorical features
    return numerical_features, categorical_features


def create_tf_dataset(dataframe, target_column, shuffle=True, batch_size=32):
    """Set target variable and converts dataframe to tensorflow dataset

    Args:
        df (dataframe): dataframe
        target_column (str): Column used to predict for labels
        shuffle (bool, optional): [description]. Defaults to True.
        batch_size (int, optional): Sets batch size. Defaults to 32.

    Returns:
        [type]: [description]
    """
    df = dataframe.copy()
    print(df[target_column].head())
    labels = df.pop(target_column)
    df = {key: value[:, tf.newaxis] for key, value in dataframe.items()}
    ds = tf.data.Dataset.from_tensor_slices((dict(df), labels))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(dataframe))
    ds = ds.batch(batch_size)
    ds = ds.prefetch(batch_size)
    print('Create Dataset: Successful')
    return ds


def get_normalization_layer(name, dataset):
    # Create a Normalization layer for the feature.
    # Layer Normalization normalizes each feature of the activations
    # to zero mean and unit variance.
    normalizer = layers.Normalization(axis=None)
    # Prepare a Dataset that only yields the feature.
    feature_ds = dataset.map(lambda x, y: x[name])
    # Learn the statistics of the data.
    normalizer.adapt(feature_ds)
    return normalizer


def get_category_encoding_layer(name, dataset, dtype, max_tokens=None):
    # Create a layer that turns strings into integer indices.
    if dtype == 'string':
        index = layers.StringLookup(max_tokens=max_tokens)
    # Otherwise, create a layer that turns integer values into integer indices.
    else:
        index = layers.IntegerLookup(max_tokens=max_tokens)
    # Prepare a `tf.data.Dataset` that only yields the feature.
    feature_ds = dataset.map(lambda x, y: x[name])
    # Learn the set of possible values and assign them a fixed integer index.
    index.adapt(feature_ds)
    # Encode the integer indices.
    encoder = layers.CategoryEncoding(num_tokens=index.vocabulary_size())
    # Apply multi-hot encoding to the indices. The lambda function captures the
    # layer, so you can use them, or include them in the Keras Functional model later.
    return lambda feature: encoder(index(feature))


def encode_tensor_flow_features(train_df, val_df, test_df, target_column,
    numerical_features, categorical_features, categorical_dictionary, size_of_batch=256)
    :
    """ size of batch may vary, defaults to 256
    """
    # Creates the dataset
    train_dataset = create_tf_dataset(
        train_df, target_column, shuffle=True, batch_size=size_of_batch)
    val_dataset = create_tf_dataset(
        val_df, target_column, shuffle=False, batch_size=size_of_batch)
    test_dataset = create_tf_dataset(
        test_df, target_column, shuffle=False, batch_size=size_of_batch)

    # Display a set of batches
    [(train_features, label_batch)] = train_dataset.take(1)
    print('Every feature:', list(train_features.keys()))
    print('A batch of size groups:', train_features['size_grp'])
    print('A batch of targets:', label_batch)

    # Initilise input and encoded feature arrays
    all_inputs = []
    encoded_features = []
    numerical_count = 0
    categorical_count = 0

    # Encode the remaicategorical features
    for header in categorical_features:
        try:
            print('Start: ', header)
            categorical_col = tf.keras.Input(
                shape=(1,), name=header, dtype=categorical_dictionary[header])
            print('Processing: Input Categorical Column')
            encoding_layer = get_category_encoding_layer(name=header,
                                                         dataset=train_dataset,
                                                         dtype=categorical_dictionary[
    header],
                                                         max_tokens=5)
            print('Processing: Sourced Encoding Layer')
            encoded_categorical_col = encoding_layer(categorical_col)
            print('Processing: Encoded Categorical Column')
            all_inputs.append(categorical_col)
            encoded_features.append(encoded_categorical_col)
            print('Passed: ', header)
            categorical_count = categorical_count + 1
            print('Number of Categorical Features Encoded: ', categorical_count)
        except RuntimeError as e:
            print(e)
        # Monitor memory usage
        monitor_memory_usage(units=3, cpu=True, gpu=True)
    # Normalise the numerical features
    for header in numerical_features:
        try:
            print('Start: ', header)
            numeric_col = tf.keras.Input(shape=(1,), name=header)
            print('Processing: Input Numeric Column')
            normalization_layer = get_normalization_layer(
                header, train_dataset)
            print('Processing: Sourced Normalization Layer')
            encoded_numeric_col = normalization_layer(numeric_col)
            print('Processing: Encoded Numerical Column')
            all_inputs.append(numeric_col)
            encoded_features.append(encoded_numeric_col)
            print('Passed: ', header)
            numerical_count = numerical_count + 1
            print('Number of Numerical Features Encoded: ', numerical_count)
        except RuntimeError as e:
            print(e)
        # Monitor memory usage
        monitor_memory_usage(units=3, cpu=True, gpu=True)
    # Concatenate all encoded layers
    all_features = tf.keras.layers.concatenate(encoded_features)
    print('All Features')
    print(all_features)
    print('Encoding: Successful')
    # Monitor memory usage
    monitor_memory_usage(units=3, cpu=True, gpu=True)
    return all_features, all_inputs, train_dataset, val_dataset, test_dataset


def build_tensor_flow_model(train_dataset, val_dataset, test_dataset, model_name,
    all_features, all_inputs, selected_optimizer, selected_loss, selected_metrics,
    finance_configuration=True):
    # Information pertaining to the tf.keras.layers.dense function
    if finance_configuration:
        # Note: The combination of optimizer. loss function and metric must be
    compatible
        # https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
        # Generalised Artificial Neural Network
        # Input features (One per feature)
        # Hidden Layers (1-5)
        # Neurons per input layer (10-100)
        # Output neurons (1 per prediction dimension)
        # Hidden activations (Relum Tanh, Sigmoid)
        # Output layer (sigmoid)

        # List of activation functions:
        # 'relu' = Rectified linear unit activation
        # 'sigmoid' = Sigmoid activation function, sigmoid(x) = 1 / (1 + exp(-x)).
        # 'softmax' = Softmax converts a vector of values to a probability distribution
        # 'softplus' = Softplus activation function, softplus(x) = log(exp(x) + 1)
        # 'softsign' = Softsign activation function, softsign(x) = x / (abs(x) + 1).
        # 'tanh' = Hyperbolic tangent activation function.
        # 'selu' = Scaled Exponential Linear Unit (SELU) activation function is defined
    as:
        #   if x > 0: return scale * x
        #   if x < 0: return scale * alpha * (exp(x) - 1)
        # 'elu' = The exponential linear unit (ELU) with alpha > 0 is:
        # x if x > 0 and alpha * (exp(x) - 1) if x < 0
        # Note: The ELU hyperparameter alpha controls the value to which an ELU
    saturates
        # for negative net inputs. ELUs diminish the vanishing gradient effect.
        # https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout
        # Dropout layer to randomly set input units to zero with a deterministic rate
        # during each step of training to help prevent overfitting. Note:
        # inputs not set to zero are scaled by 1/(1-rate) so the sum of all inputs is
    unchanged.

        # Configure the neural network layers
        print('Start: Configuration of Deep Network Layers')
        # Binary variables to control network construction
        complex_model = True
        # Simple configuration, only a handful of layers
        if complex_model:
            # Initial Layer
            layer_1 = tf.keras.layers.Dense(
                32, activation="relu")(all_features)
            # Dropout layer
            layer_2 = tf.keras.layers.Dropout(
                rate=0.5, noise_shape=None, seed=None)(layer_1)
            layer_3 = tf.keras.layers.Dense(64, activation='relu')(layer_2)
            layer_4 = tf.keras.layers.Dense(128, activation='sigmoid')(layer_3)
            # Creates the output layer
            output = tf.keras.layers.Dense(1)(layer_4)
            print('End: Configuration of Deep Network Layers')
            # Configure the model (https://www.tensorflow.org/api_docs/python/tf/keras/
    Model)
            model = tf.keras.Model(all_inputs, output)
            print('Model Summary')
            print(model.summary)
        # Deploy a sequential model
        else:
            # Initial Layer
            x = tf.keras.layers.Dense(
                units=32, activation="relu", use_bias=True,
                kernel_initializer='glorot_uniform',
                bias_initializer='zeros', kernel_regularizer=None,
                bias_regularizer=None, activity_regularizer=None, kernel_constraint=None
    ,
                bias_constraint=None)(all_features)
            # Dropout layer
            x = tf.keras.layers.Dropout(
                rate=0.5, noise_shape=None, seed=None)(x)
            # Creates the output layer
            output = tf.keras.layers.Dense(1)(x)
            print('End: Configuration of Deep Network Layers')
            # Configure the model (https://www.tensorflow.org/api_docs/python/tf/keras/
    Model)
            model = tf.keras.Model(all_inputs, output)
        # Initilises optimizer variables
        lr = 0.001
        eps = 1e-07
        rh = 0.95
        mom = 0.0
        b0 = 0.0
        b1 = 0.9
        b2 = 0.999
        iav = 0.1
        lrp = -0.5
        l1rs = 0.0
        l2rs = 0.0
        l2srs = 0.0
        ams = False
        cen = False
        nes = False
    # ################################################################################
        # Optimizer (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers)
        # ################################################################################
        if selected_optimizer == 'Adagrad':
            opt = tf.keras.optimizers.Adagrad(
                learning_rate=lr, initial_accumulator_value=iav, epsilon=eps, name='Adagrad')
        if selected_optimizer == 'Adadelta':
            opt = tf.keras.optimizers.Adadelta(
                learning_rate=lr, rho=rh, epsilon=eps, name='Adadelta')
        if selected_optimizer == 'Adam':
            opt = tf.keras.optimizers.Adam(
                learning_rate=lr, beta_1=b1, beta_2=b2, epsilon=eps, amsgrad=ams, name='Adam')
        if selected_optimizer == 'Adamax':
            opt = tf.keras.optimizers.Adamax(
                learning_rate=lr, beta_1=b1, beta_2=b2, epsilon=eps, name='Adamax')
        if selected_optimizer == 'Ftrl':
            opt = tf.keras.optimizers.Ftrl(
                learning_rate=lr, learning_rate_power=lrp, initial_accumulator_value=iav,
                l1_regularization_strength=l1rs, l2_regularization_strength=l2rs,
                name='Ftrl', l2_shrinkage_regularization_strength=l2srs, beta=b0)
        if selected_optimizer == 'Nadam':
            opt = tf.keras.optimizers.Nadam(
                learning_rate=lr, beta_1=b1, beta_2=b2, epsilon=eps, name='Nadam')
        if selected_optimizer == 'RMSprop':
            opt = tf.keras.optimizers.RMSprop(
                learning_rate=lr, rho=rh, momentum=mom, epsilon=eps, centered=cen, name='RMSprop')
        if selected_optimizer == 'SGD':
            opt = tf.keras.optimizers.SGD(
                learning_rate=lr, momentum=mom, nesterov=nes, name='SGD')
    # ################################################################################
        # Losses
    # ################################################################################
        # Loss variables
        red = 'auto'
        flt = True
        ls = 0.0
        ax = -1
        dta = 1.0
        # Loss classes
        if selected_loss == 'binary_crossentropy':
            lf = tf.keras.losses.BinaryCrossentropy(
                from_logits=flt, label_smoothing=ls, axis=ax, reduction=red, name='binary_crossentropy')
        if selected_loss == 'categorical_crossentropy':
            lf = tf.keras.losses.CategoricalCrossentropy(
                from_logits=flt, label_smoothing=ls, axis=ax, reduction=red, name='categorical_crossentropy')
        if selected_loss == 'cosine_similarity':
            lf = tf.keras.losses.CosineSimilarity(
                axis=-1, reduction=red, name='cosine_similarity')
        if selected_loss == 'hinge':
            lf = tf.keras.losses.Hinge(reduction=red, name='hinge')
        if selected_loss == 'huber_loss':
            lf = tf.keras.losses.Huber(
                delta=dta, reduction=red, name='huber_loss')
        # loss = y_true * log(y_true / y_pred)
        if selected_loss == 'kl_divergence':
            lf = tf.keras.losses.KLDivergence(
                reduction=red, name='kl_divergence')
        # logcosh = log((exp(x) + exp(-x))/2), where x is the error y_pred - y_true.
        if selected_loss == 'log_cosh':
            lf = tf.keras.losses.LogCosh(reduction=red, name='log_cosh')
        if selected_loss == 'loss':
            lf = tf.keras.losses.Loss(reduction=red, name=None)
        # loss = abs(y_true - y_pred)
        if selected_loss == 'mean_absolute_error':
            lf = tf.keras.losses.MeanAbsoluteError(
                reduction=red, name='mean_absolute_error')
        # loss = 100 * abs(y_true - y_pred) / y_true
        if selected_loss == 'mean_absolute_percentage_error':
            lf = tf.keras.losses.MeanAbsolutePercentageError(
                reduction=red, name='mean_absolute_percentage_error')
        # loss = square(y_true - y_pred)
        if selected_loss == 'mean_squared_error':
            lf = tf.keras.losses.MeanSquaredError(
                reduction=red, name='mean_squared_error')
        # loss = square(log(y_true + 1.) - log(y_pred + 1.))
        if selected_loss == 'mean_squared_logarithmic_error':
            lf = tf.keras.losses.MeanSquaredLogarithmicError(
                reduction=red, name='mean_squared_logarithmic_error')
        if selected_loss == 'poisson':  # loss = y_pred - y_true * log(y_pred)
            lf = tf.keras.losses.Poisson(reduction=red, name='poisson')
        if selected_loss == 'sparse_categorical_crossentropy':
            lf = tf.keras.losses.SparseCategoricalCrossentropy(
                from_logits=flt, reduction=red, name='sparse_categorical_crossentropy')
        # loss = square(maximum(1 - y_true * y_pred, 0))
        if selected_loss == 'squared_hinge':
            lf = tf.keras.losses.SquaredHinge(
                reduction=red, name='squared_hinge')
        # Custom loss classes
        # loss = square(maximum(1 - y_true * y_pred, 0))
        if selected_loss == 'custom_l2_mse':
            lf = custom_l2_mse
        # loss = square(maximum(1 - y_true * y_pred, 0))
        if selected_loss == 'custom_hedge_portfolio_returns':
            lf = custom_hedge_portfolio_returns
        # loss = square(maximum(1 - y_true * y_pred, 0))
        if selected_loss == 'custom_sharpe_ratio':
            lf = custom_sharpe_ratio
        # loss = square(maximum(1 - y_true * y_pred, 0))
        if selected_loss == 'custom_information_ratio':
            lf = custom_information_ratio
        # if selected_loss == 'multi_layer_loss':
        #     lf = multi_layer_loss
        if selected_loss == 'custom_loss':
            lf = custom_loss(layer=layer_3, reduction=red, name='custom_loss')
        #
    ################################################################################
        # Metrics
        #
    ################################################################################
        # Metric variables
        metrics_list = []
        meaniou_num_classes = 2

        def mean_metric_wrapper_function(y_true, y_pred):
            return tf.cast(tf.math.equal(y_true, y_pred), tf.float32)
        # Must be the same size as predictions
        mean_relative_error_normalizer = [1, 2, 3, 4]
        recall = 0.5  # A scalar value in range [0, 1]
        precision = 0.5  # A scalar value in range [0, 1]
        specificity = 0.5  # A scalar value in range [0, 1]
        sensitivity = 0.5  # A scalar value in range [0, 1]
        # Metric Classes
        if 'Auc' in selected_metrics:
            metrics_list.append(tf.keras.metrics.AUC(
                num_thresholds=200, curve='ROC',
                summation_method='interpolation', name=None, dtype=None,
                thresholds=None, multi_label=False, num_labels=None, label_weights=None,
                from_logits=False))
        if 'accuracy' in selected_metrics:
            met = metrics_list.append(tf.keras.metrics.Accuracy(
                name='accuracy', dtype=None))
        if 'binary_accuracy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.BinaryAccuracy(
                name='binary_accuracy', dtype=None, threshold=0.5))
        if 'binary_crossentropy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.BinaryCrossentropy(
                name='binary_crossentropy', dtype=None, from_logits=False,
                label_smoothing=0))
        if 'categorical_accuracy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CategoricalAccuracy(
                name='categorical_accuracy', dtype=None))
        if 'categorical_crossentropy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CategoricalCrossentropy(
                name='categorical_crossentropy', dtype=None, from_logits=False,
                label_smoothing=0))
        if 'categorical_hinge' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CategoricalHinge(
                name='categorical_hinge', dtype=None))
        if 'cosine_similarity' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CosineSimilarity(
                name='cosine_similarity', dtype=None, axis=-1))
        if 'Fn' in selected_metrics:
            metrics_list.append(tf.keras.metrics.FalseNegatives(
                thresholds=None, name=None, dtype=None))
        if 'Fp' in selected_metrics:
            metrics_list.append(tf.keras.metrics.FalsePositives(
                thresholds=None, name=None, dtype=None))
        if 'hinge' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Hinge(
                name='hinge', dtype=None))
        if 'kullback_leibler_divergence' in selected_metrics:
            metrics_list.append(tf.keras.metrics.KLDivergence(
                name='kullback_leibler_divergence', dtype=None))
        if 'logcosh' in selected_metrics:
            metrics_list.append(tf.keras.metrics.LogCoshError(
                name='logcosh', dtype=None))
        if 'mean' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Mean(
                name='mean', dtype=None))
        if 'mean_absolute_error' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanAbsoluteError(
                name='mean_absolute_error', dtype=None))
        if 'mean_absolute_percentage_error' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanAbsolutePercentageError(
                name='mean_absolute_percentage_error', dtype=None))
        if 'meaniou' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanIoU(
                num_classes=meaniou_num_classes, name=None, dtype=None))
        if 'mean_metric_wrapper' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanMetricWrapper(
                fn=mean_metric_wrapper_function, name=None, dtype=None))
        if 'mean_relative_error' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanRelativeError(
                normalizer=mean_relative_error_normalizer, name=None, dtype=None))
        if 'mean_squared_error' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanSquaredError(
                name='mean_squared_error', dtype=None))
        if 'mean_squared_logarithmic_error' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanSquaredLogarithmicError(
                name='mean_squared_logarithmic_error', dtype=None))
        if 'mean_tensor' in selected_metrics:
            metrics_list.append(tf.keras.metrics.MeanTensor(
                name='mean_tensor', dtype=None, shape=None))
        if 'metric' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Metric(
                name=None, dtype=None))
        if 'poisson' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Poisson(
                name='poisson', dtype=None))
        if 'precision' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Precision(
                thresholds=None, top_k=None, class_id=None, name=None, dtype=None))
        if 'precision_at_recall' in selected_metrics:
            metrics_list.append(tf.keras.metrics.PrecisionAtRecall(
                recall, num_thresholds=200, class_id=None, name=None, dtype=None))
        if 'recall' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Recall(
                thresholds=None, top_k=None, class_id=None, name=None, dtype=None))
        if 'recall_at_precision' in selected_metrics:
            metrics_list.append(tf.keras.metrics.RecallAtPrecision(
                precision, num_thresholds=200, class_id=None, name=None, dtype=None))
        if 'root_mean_squared_error' in selected_metrics:
            metrics_list.append(tf.keras.metrics.RootMeanSquaredError(
                name='root_mean_squared_error', dtype=None))
        if 'sensitivity_at_specificity' in selected_metrics:
            metrics_list.append(tf.keras.metrics.SensitivityAtSpecificity(
                specificity, num_thresholds=200, class_id=None, name=None, dtype=None))
        if 'sparse_categorical_accuracy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.SparseCategoricalAccuracy(
                name='sparse_categorical_accuracy', dtype=None))
        if 'sparse_top_k_categorical_accuracy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.SparseTopKCategoricalAccuracy(
                k=5, name='sparse_top_k_categorical_accuracy', dtype=None))
        if 'specificty_at_sensitivity' in selected_metrics:
            metrics_list.append(tf.keras.metrics.SpecificityAtSensitivity(
                sensitivity, num_thresholds=200, class_id=None, name=None, dtype=None))
        if 'squared_hinge' in selected_metrics:
            metrics_list.append(tf.keras.metrics.SquaredHinge(
                name='squared_hinge', dtype=None))
        if 'sum' in selected_metrics:
            metrics_list.append(tf.keras.metrics.Sum(
                name='sum', dtype=None))
        if 'top_k_categorical_accuracy' in selected_metrics:
            metrics_list.append(tf.keras.metrics.TopKCategoricalAccuracy(
                k=5, name='top_k_categorical_accuracy', dtype=None))
        if 'Tn' in selected_metrics:
            metrics_list.append(tf.keras.metrics.TrueNegatives(
                thresholds=None, name=None, dtype=None))
        if 'Tp' in selected_metrics:
            metrics_list.append(tf.keras.metrics.TruePositives(
                thresholds=None, name=None, dtype=None))
        # Custom Metrics
        if 'hedge_portfolio_mean' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CustomHedgePortolfioMean(
                num_classes=None, batch_size=None,
                name='hedge_portfolio_mean'))
        if 'hedge_portfolio_alphas' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CustomHedgePortolfioAlphas(
                num_classes=None, batch_size=None,
                name='hedge_portfolio_alphas'))
        if 'sharpe_ratio' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CustomSharpeRatio(
                num_classes=None, batch_size=None,
                name='sharpe_ratio'))
        if 'information_ratio' in selected_metrics:
            metrics_list.append(tf.keras.metrics.CustomInformationRatio(
                num_classes=None, batch_size=None,
                name='information_ratio'))
        #
    ################################################################################
        # Loss weights
        #
    ################################################################################
        # Optional list or dictionary specifying scalar coefficients (Python floats) to
        # weight the loss contributions of different model outputs. The loss value that
        # will be minimized by the model will then be the weighted sum of all individual
        # losses, weighted by the loss_weights coefficients. If a list, it is expected
        # to have a 1:1 mapping to the model's outputs. If a dict, it is expected to map
        # output names (strings) to scalar coefficients.
        lw = None
        #
    ################################################################################
        # Weighted Metrics
        #
    ################################################################################
        # List of metrics to be evaluated and weighted by sample_weight or class_weight
        # during training and testing.
        wm = None
    # ################################################################################
        # Run eagerly
    # ################################################################################
        # Bool. Defaults to False. If True, this Model's logic will not be wrapped in a
        # tf.function. Recommended to leave this as None unless your Model cannot be run
        # inside a tf.function. run_eagerly=True is not supported when using
        # tf.distribute.experimental.ParameterServerStrategy.
        regly = None
    # ################################################################################
        # Steps_per_execution
    # ################################################################################
        # Int. Defaults to 1. The number of batches to run during each tf.function call.
        # Running multiple batches inside a single tf.function call can greatly improve
        # performance on TPUs or small models with a large Python overhead. At most,
        # one full epoch will be run each execution. If a number larger than the size
        # of the epoch is passed, the execution will be truncated to the size of the
        # epoch. Note that if steps_per_execution is set to N, Callback.on_batch_begin
        # and Callback.on_batch_end methods will only be called every N batches
        # (i.e. before/after each tf.function execution).
        spe = None
    # ################################################################################
        # Compiler
    # ################################################################################
        # Compiler variables
        # Establishes the compiler
        print('Start: Model Compilation')
        model.compile(
            optimizer=opt, loss=lf, metrics=metrics_list, loss_weights=lw,
            weighted_metrics=wm, run_eagerly=regly, steps_per_execution=spe)
        print('End: Model Compilation')
    # ################################################################################
        # Visualise model (https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model)
    # ################################################################################
        # Visualisation variables
        to_file = '/home/connormcdowall/finance-honours/results/plots/tensorflow-visualisations/' + \
            model_name + '.png'
        show_shapes = True
        show_dtype = False
        show_layer_names = True
        rankdir = 'TB'  # TB (Top Bottom), LR (Left Right)
        expand_nested = False
        dpi = 96
        layer_range = None
        show_layer_activations = False
        # Creates a plot of the model
        tf.keras.utils.plot_model(model, to_file, show_shapes, show_dtype,
                                   show_layer_names, rankdir, expand_nested, dpi, layer_range, show_layer_activations)
        # Prints a summary of the model
        print('Model Summary')
        print(model.summary())
    # ################################################################################
        # Model.fit (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit)
    # ################################################################################
        # Fit variables
        x_train = train_dataset
        y = None  # If x is a dataset, generator, or keras.utils.Sequence instance, y should
        # not be specified (since targets will be obtained from x).
        batch_size = None  # Defaults to 32
        eps = 10  # Integer. Number of epochs to train the model. An epoch is an iteration over
        # the entire x and y data provided (unless the steps_per_epoch flag is set to something other than None).
        verbose = 'auto'
        callbacks = None
        validation_split = 0.0  # Not support when x is a dataset
        validation_data = val_dataset
        # Ignored when x is a generator or an object of tf.data.Dataset (This case)
        shuffle = True
        # Optional dictionary mapping class indices (integers) to a
        class_weight = None
        # continued: weight (float) value, used for weighting the loss function (during training only)
        sample_weight = None  # This argument is not supported when x is a dataset
        # Integer. Epoch at which to start training (useful for resuming a previous training run).
        initial_epoch = 0
        # If x is a tf.data dataset, and 'steps_per_epoch' is None, the epoch will run until the input dataset is exhausted.
        steps_per_epoch = None
        # Only relevant if validation_data is provided and is a tf.data dataset.
        validation_steps = None
        # Continued: If 'validation_steps' is None, validation will run until the validation_data dataset is exhausted.
        # Do not specify the validation_batch_size if your data is in the form of datasets
        validation_batch_size = None
        validation_freq = 1
        # Integer. Used for generator or keras.utils.Sequence input only.
        max_queue_size = 10
        # Continued: Maximum size for the generator queue. If unspecified, max_queue_size will default to 10.
        # Integer. Used for generator or keras.utils.Sequence input only (Not this case)
        workers = 1
        # Boolean. Used for generator or keras.utils.Sequence input only.
        use_multiprocessing = False
        # Fit the model
        print('Start: Model Fitting')
        model.fit(x=x_train, batch_size=32, epochs=eps,
                  verbose='auto', validation_data=val_dataset)
        # model.fit(x=x_train, batch_size=32, epochs=eps, verbose='auto',
        #     callbacks=None, validation_data=val_dataset, shuffle=True,
        #     class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
        #     validation_steps=None, max_queue_size=10, workers=1, use_multiprocessing=False)
        print('End: Model Fitting')
        # model.fit(x, batch_size, epochs=eps, verbose='auto',
        # callbacks, validation_data, shuffle,
        # class_weight, sample_weight, initial_epoch, steps_per_epoch,
        # validation_steps, validation_batch_size, validation_freq,
        # max_queue_size, workers, use_multiprocessing)
    # ################################################################################
        # Model.evaluate (https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate)
    # ################################################################################
        # Evaluation variables
        x_test = test_dataset
        # Only use if target variables not specified in dataset, must align with x.
        y = None
        batch_size = None  # Defaults to 32
        verb = 1  # 0 or 1. Verbosity mode. 0 = silent, 1 = progress bar.
        sample_weight = None  # Optional, This argument is not supported when x is a dataset
        steps = None  # If x is a tf.data dataset and steps is None, 'evaluate' will run until the dataset is exhausted
        callbacks = None
        mqs = 10  # Max queue size. If unspecified, max_queue_size will default to 10
        workers = 1  # Integer. Used for generator or keras.utils.Sequence
```

```python
        # use_multiprocessing, boolean. Used for generator or keras.utils.Sequence input only.
        ump = False
        # Continued: If True, use process-based threading. If unspecified, use_multiprocessing will default to False.
        rd = False  # If True, loss and metric results are returned as a dict,
        # with each key being the name of the metric. If False, they are returned as a list.
        # Model evaluation
        print('Start: Model Evaluation')
        loss, metrics = model.evaluate(x_test, batch_size=None, verbose=verb, steps=None, callbacks=None,
                                       max_queue_size=mqs, workers=1, use_multiprocessing=ump, return_dict=rd)
        # ################################################################################
        print('End: Model Evaluation')
        print("Loss: ", loss)
        print("Metric Descriptions: ", model.metrics_names)
        print("Metric Values: ", metrics)
        # Save the model
        model.save(
            '/home/connormcdowall/finance-honours/results/model/tensorflow-models/'+model_name+'.pb')
        # Monitor memory usage
        monitor_memory_usage(units=3, cpu=True, gpu=True)
        # Return the model, loss and accuracy
        return model, loss, metrics
    else:
        # Exemplar implementation prior to finance adaptation
        # Set up neural net layers
        x = tf.keras.layers.Dense(32, activation="relu")(all_features)
        x = tf.keras.layers.Dropout(rate=0.5, noise_shape=None, seed=None)(x)
        output = tf.keras.layers.Dense(1)(x)
        # Configure the model
        model = tf.keras.Model(all_inputs, output)
        model.compile(optimizer='adam',
                      loss=tf.keras.losses.BinaryCrossentropy(
                          from_logits=True),
                      metrics=["accuracy"])
        # Visualise the model via a connectivity graph
        tf.keras.utils.plot_model(model, show_shapes=True, rankdir="LR")
        # Train the model
        model.fit(train_dataset, epochs=10, validation_data=val_dataset)
        # Test the model
        loss, accuracy = model.evaluate(test_dataset)
        print("Loss: ", loss)
        print("Accuracy: ", accuracy)
        # Save the model
        model.save('results/plots/tensorflow-models/'+model_name+'.pb')
        # Return the model, loss and accuracy
        return model, loss, accuracy


def perform_tensorflow_model_inference(model_name, sample):
    """ Perform evaluations from model (must be configured)

    Args:
        model_name ([type]): [description]
        sample ([type]): [description]

    Returns:
        [type]: [description]
    """
    reloaded_model = tf.keras.models.load_model(model_name)
    input_dict = {name: tf.convert_to_tensor(
        [value]) for name, value in sample.items()}
    predictions = reloaded_model.predict(input_dict)
    prob = tf.nn.sigmoid(predictions[0])
    return prob


def implement_test_data(dataframe, train, val, test, full_implementation=False):
    # Sets the batch size
    target_column = 'target'
    batch_size = 5
    train_ds = create_tf_dataset(
        train, target_column, shuffle=True, batch_size=batch_size)
    # See arrangement of the data
    [(train_features, label_batch)] = train_ds.take(1)
    print('Every feature:', list(train_features.keys()))
    print('A batch of ages:', train_features['Age'])
    print('A batch of targets:', label_batch)
    # Test the get_normalisation function
    photo_count_col = train_features['PhotoAmt']
    layer = get_normalization_layer('PhotoAmt', train_ds)
    layer(photo_count_col)
    # Test the get category encoding layer function
    test_type_col = train_features['Type']
    test_type_layer = get_category_encoding_layer(name='Type',
                                                  dataset=train_ds,
                                                  dtype='string')
    test_type_layer(test_type_col)
    test_age_col = train_features['Age']
    test_age_layer = get_category_encoding_layer(name='Age',
                                                 dataset=train_ds,
                                                 dtype='int64',
                                                 max_tokens=5)
    test_age_layer(test_age_col)
    # Continues with a full implementation if necessary
    if full_implementation:
        print("Continues with full implementation")
        numerical_features = ['PhotoAmt', 'Fee']
        categorical_features = ['Age', 'Type', 'Color1', 'Color2', 'Gender', 'MaturitySize',
                                'FurLength', 'Vaccinated', 'Sterilized', 'Health', 'Breed1']
        # Create categorical type dictionary
        categorical_dictionary = dict.fromkeys(categorical_features, 'string')
        categorical_dictionary["Age"] = 'int64'
        model_name = 'pets_test'
        selected_optimizer = 'adam'
        selected_loss = 'binary_crossentropy'
        selected_metrics = ['accuracy']
        all_features, all_inputs, train_dataset, val_dataset, test_dataset = encode_tensor_flow_features(
            train, val, test, target_column, numerical_features, categorical_features, categorical_dictionary, size_of_batch=256)
        model, loss, metrics = build_tensor_flow_model(train_dataset, val_dataset, test_dataset, model_name,
                                                       all_features, all_inputs, selected_optimizer, selected_loss, selected_metrics, finance_configuration=False)
        # Test model inference
        sample = {
            'Type': 'Cat',
                    'Age': 3,
                    'Breed1': 'Tabby',
                    'Gender': 'Male',
                    'Color1': 'Black',
                    'Color2': 'White',
                    'MaturitySize': 'Small',
                    'FurLength': 'Short',
                    'Vaccinated': 'No',
                    'Sterilized': 'No',
                    'Health': 'Healthy',
                    'Fee': 100,
                    'PhotoAmt': 2,
        }
        prob = perform_tensorflow_model_inference(
            'results/plots/tensorflow-models/'+model_name+'.pb', sample)
    else:
        print('Test functions complete')
    return


```

```python
def project_analysis(data_vm_directory, list_of_columns, categorical_assignment,
    target_column, chunk_size, resizing_options, batch_size, model_name,
    selected_optimizer, selected_loss, selected_metrics, split_data=False, trial=False,
    sample=False):
    # Prints memory usage before analysis
    monitor_memory_usage(units=3, cpu=True, gpu=True)
    # Reset working textfile if resizing used for numerical encoding
    # Clear the working file
    file = open(
        "/home/connormcdowall/finance-honours/data/working-columns.txt", "r+")
    file.truncate(0)
    file.close()
    # Tranfer file lines
    with open("/home/connormcdowall/finance-honours/data/dataframe-columns.txt", "r") as
     f1:
        with open("/home/connormcdowall/finance-honours/data/working-columns.txt", "w")
    as f2:
            for line in f1:
                f2.write(line)
    # Split the initial vm dataset
    if split_data:
        split_vm_dataset(data_vm_directory, create_statistics=False,
                         split_new_data=True, create_validation_set=True)
    # Creates the training, validation and testing dataframes
    test_df = process_vm_dataset(data_vm_directory + 'test.dta', chunk_size,
                                 resizing_options, save_statistics=False, sample=sample)
    train_df = process_vm_dataset(data_vm_directory + 'train.dta',
                                  chunk_size, resizing_options, save_statistics=False,
    sample=sample)
    val_df = process_vm_dataset(data_vm_directory + 'val.dta', chunk_size,
                                resizing_options, save_statistics=False, sample=sample)
    # Use trial to test the dataframe when functions not as large
    if trial:
        # Trial run takes 5% of dataframe produced from processed vm datasets
        test_df, test_discard_df = train_test_split(test_df, test_size=0.95)
        train_df, train_discard_df = train_test_split(train_df, test_size=0.95)
        val_df, val_discard_df = train_test_split(val_df, test_size=0.95)
    # Create feature lists for deep learning
    numerical_features, categorical_features = create_feature_lists(
        list_of_columns, categorical_assignment)
    # Creates the categorical dictonary (must specify the variables types of each)
    categorical_dictionary = dict.fromkeys(categorical_features, 'string')
    category_dtypes = {'size_grp': 'string', 'permno': 'int32', 'permco': 'int32', '
    crsp_shrcd': 'int8',
                       'crsp_exchcd': 'int8', 'adjfct': 'float64', 'sic': 'float64', '
    ff49': 'float64'}
    for key in category_dtypes:
        categorical_dictionary[key] = category_dtypes[key]
    # Encodes the tensorflow matrix
    all_features, all_inputs, train_dataset, val_dataset, test_dataset =
    encode_tensor_flow_features(
        train_df, val_df, test_df, target_column, numerical_features,
    categorical_features, categorical_dictionary, size_of_batch=batch_size)
    # Note: Keep Stochastic Gradient Descent as Optimizer for completeness
    # Buids tensorflow model
    model, loss, metrics = build_tensor_flow_model(train_dataset, val_dataset,
    test_dataset, model_name,
                                                   all_features, all_inputs,
    selected_optimizer, selected_loss, selected_metrics, finance_configuration=True)
    return
#################################################################################
# Custom Loss Functions, Metrics and Autodiff Testing
#################################################################################
# Loss Functions
#################################################################################
# Key:
# 0 = Matrix of Parameters (Theta)
# X = Feature Matrix
# f_(0)(X) = Target (e.g., Excess Returns)
# V = All-Ones=Vector

# Use Tensorlow backend functions

# 0: Custom Example for reference
# Loss Function (Class Example, not as efficient)


class CustomLossFunctionExample(tf.keras.losses.Loss):
    # Example from Youtube (https://www.youtube.com/watch?v=gcwRjM1nZ4o)
    def __init__(self):
        # Initialise the function
        super().__init__()

    def call(self, y_true, y_pred):
        mse = tf.reduce_mean(tf.square(y_true, y_pred))
        rmse = tf.math.sqrt(mse)
        return rmse / tf.reduce_mean(tf.square(y_true)) - 1

# 1: In-Built MSE Loss Function / Metric
# Call MSE Loss Function/Metric with SGD in build_tensorflow_model()

# 2: Custom L2 (Mean Square Error Function)


@tf.function  # Decorate the function
def custom_l2_mse(y_true, y_pred):
    mse = K.mean(K.square(y_true - y_pred))
    return mse

# 3: Custom Hedge Portfolio Returns


@tf.function
def custom_hedge_portfolio_returns(y_true, y_pred):
    # Analytical Derivation
    # f_(0)(X) = ((X^T(0)/V(X^T))^T)X^T(0)
    # Derivitive of Function
    # df_(0)(X)/d(0) = (1/((0^T)X1)(X)(X^T)(0)
    #                + (1/((VX^T)(0))(X)(X^T)(0)
    #                - (1/((0^T)(X)(V))**2)(0^T)(X)(X^T)(0)(X)(V)

    # Empirical Derivation(s)
    # Sets boolean to select weighting scheme
    equally_weighted = False
    # Sets up predicted value
    # Get the shape of a tensor
    print('y_pred is of shape: ', y_true.shape)
    print('y_true is of type: ', type(y_true))
    sp_pred = y_true.shape[0]
    print(sp_pred)
    # Implments Equally Weighted Monotonic Weighting Function
    if equally_weighted:
        # Initialise equally-weighted array
        weights = np.linspace(1, -1, sp_pred)
        # Alternative method of calculating weights
        # weights = np.empty([sp_pred,1])
        # weights[0] = 1
        # weights[1] = -1
        # # Sets remaining weights via a loop
        # for i in range(len(weights)):
        #     if i > 0:
        #         weights[i] = weights[i-1] - 2/(len(weights)-1)

        # Sorts the returns to descending_order
        y_pred_sorted = tf.sort(y_pred, axis=-1, direction='DESCENDING')
        y_true_sorted = tf.sort(y_true, axis=-1, direction='DESCENDING')
        # Calculates weighted Tensors
        weighted_returns_pred = tf.math.multiply(weights, y_pred_sorted)
        weighted_returns_true = tf.math.multiply(weights, y_true_sorted)
        # Calculates MSE equivalent between the hedge portfolios
        loss = K.mean(K.square(weighted_returns_true - weighted_returns_pred))
    else:
        # Gets the mean of the top 10% of predicted returns
        print('sp_pred is ', type(sp_pred))
        print('y_pred is ', type(y_pred))

        long_mean_pred = K.mean(tf.math.top_k(y_pred, k=0.1*sp_pred))
        # Creates a negative
        neg_y_pred = tf.math.scalar_mul(-1, y_pred)
        # Gets the mean of the top 10% of predicted returns
        short_mean_pred = -1*K.mean(tf.math.top_k(neg_y_pred, k=0.1*sp_pred))
        # Gets the value of the hedge portfolio
        hedge_pred = long_mean_pred - short_mean_pred
        # Sets up true value
        # Get the shape of a tensor
        sp_true = y_true.shape[0]
        # Gets the mean of the top 10% of predicted returns
        long_mean_true = K.mean(tf.math.top_k(y_true, k=0.1*sp_true))
        # Creates a negative
        neg_y_true = tf.math.scalar_mul(-1, y_true)
        # Gets the mean of the top 10% of predicted returns
        short_mean_true = -1*K.mean(tf.math.top_k(neg_y_true, k=0.1*sp_true))
        # Gets the value of the hedge portfolio
        hedge_true = long_mean_true - short_mean_true
        # Calculate a MSE based on a hedge portfolio opposed to predicted returns
        loss = K.mean(K.square(hedge_true - hedge_pred))
    return loss

# 4: Custom Sharpe Ratio (# Negative to maximise)


@tf.function
def custom_sharpe_ratio(y_true, y_pred):
    # Finds Sharpe ratios of both true and predicted returns
    sr_pred = -1*(K.mean(y_pred)/K.std(y_pred))
    sr_true = -1*(K.mean(y_true)/K.std(y_true))
    # Finds MSE between predited and true MSE
    loss = K.mean(K.square(sr_true - sr_pred))
    return loss

# 5: Custom Information Ratio (E(R) - E(BM))/SD(R-BM))
# Note: This instance uses the true results as the benchmanr


@tf.function
def custom_information_ratio(y_true, y_pred):
    loss = -1*((K.mean(y_pred) - K.mean(y_true))/K.std(y_pred - y_true))
    return loss

# @tf.function
# def multi_layer_loss(self):
#         """" Wrapper function which calculates auxiliary values for the complete loss
    function.
#          Returns a *function* which calculates the complete loss given only the input
    and target output """
#         # KL loss
#         kl_loss = self.calculate_kl_loss
#         # Reconstruction loss
#         md_loss_func = self.calculate_md_loss
#         # KL weight (to be used by total loss and by annealing scheduler)
#         self.kl_weight = K.variable(self.hps['kl_weight_start'], name='kl_weight')
#         kl_weight = self.kl_weight
#         def seq2seq_loss(y_true, y_pred):
#             """ Final loss calculation function to be passed to optimizer"""
#             # Reconstruction loss
#             md_loss = md_loss_func(y_true, y_pred)
#             # Full loss
#             model_loss = kl_weight*kl_loss() + md_loss
#             return model_loss
#         return seq2seq_loss

# Note: Symbolic Tensors do not work in function calls as require eager tensors.
# Subsequently, must create custom class with call function
#
# Utilisation of function closure to pass multiple inputs into the function.


class custom_loss(tf.keras.losses.Loss):
```

```python
    def __init__(self, layer=None, reduction=tf.keras.losses.Reduction.AUTO, name='custom_loss'):
        super().__init__(reduction=reduction, name=name)
        self.layer = layer
        # self.layer = layer

    def call(self, y_true, y_pred):
        layer = self.layer
        mse = K.mean(K.square(y_true - y_pred))
        rmse = K.sqrt(mse)
        # return (rmse / K.mean(K.square(y_true)) - 1)
        return K.mean(K.square(y_pred - y_true) + K.square(layer), axis=-1)

    # def custom_loss(layer):
    #     # Create a loss function that adds the MSE loss to the mean of all squared activations of a specific layer
    #     def loss(y_true,y_pred):
    #         return K.mean(K.square(y_pred - y_true) + K.square(layer), axis=-1)
    #     # Return a function
    #     return loss

################################################################################
# Metrics
################################################################################
# 1: HP Mean


class CustomHedgePortolfioMean(tf.keras.metrics.Metric):
    # Initialisation
    def __init__(self, num_classes=None, batch_size=None,
                 name='hedge_portfolio_mean', **kwargs):
        super(CustomHedgePortolfioMean, self).__init__(name=name, **kwargs)
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.hedge_portflio_mean = self.add_weight(
            name='hedge_portfolio_mean', initializer="zeros")
        # Core componnent of the update state
    # Update State

    def update_state(self, y_true, y_pred, sample_weight=None):
        # Returns the index of the maximum values along the last axis in y_true (Last layer)
        y_true = K.argmax(y_true, axis=-1)
        # Returns the index of the maximum values along the last axis in y_true (Last layer)
        y_pred = K.argmax(y_pred, axis=-1)
        # Flattens a tensor to reshape to a shape equal to the number of elements contained
        # Removes all dimensions except for one.
        y_true = K.flatten(y_true)
        # Defines the metric for assignment
        true_poss = K.sum(K.cast((K.equal(y_true, y_pred)), dtype=tf.float32))
        self.hedge_portflio_mean.assign_add(true_poss)
    # Metric

    def result(self):
        return self.hedge_portflio_mean

# 2: HP Alphas in CAPM, FF3, FF5 ()


class CustomHedgePortolfioAlphas(tf.keras.metrics.Metric):
    # Initialisation
    def __init__(self, num_classes=None, batch_size=None,
                 name='hedge_portfolio_alphas', **kwargs):
        super(CustomHedgePortolfioAlphas, self).__init__(name=name, **kwargs)
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.custom_hedge_portfolio_alphas = self.add_weight(
            name='hedge_portfolio_alphas', initializer="zeros")
    # Update State

    def update_state(self, y_true, y_pred, sample_weight=None):
        # Returns the index of the maximum values along the last axis in y_true (Last
    layer)
        y_true = K.argmax(y_true, axis=-1)
        # Returns the index of the maximum values along the last axis in y_true (Last
    layer)
        y_pred = K.argmax(y_pred, axis=-1)
        # Flattens a tensor to reshape to a shape equal to the number of elements
    contained
        # Removes all dimensions except for one.
        y_true = K.flatten(y_true)
        # Defines the metric for assignment
        true_poss = K.sum(K.cast((K.equal(y_true, y_pred)), dtype=tf.float32))
        self.custom_hedge_portfolio_alphas.assign_add(true_poss)
    # Metric

    def result(self):
        return self.custom_hedge_portfolio_alphas

# 3: Sharpe Ratio (SR = E[R - Rf]/SD Excess Return)


class CustomSharpeRatio(tf.keras.metrics.Metric):
    # Initialisation
    def __init__(self, num_classes=None, batch_size=None,
                 name='sharpe_ratio', **kwargs):
        super(CustomSharpeRatio, self).__init__(name=name, **kwargs)
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.custom_sharpe_ratio = self.add_weight(
            name="csr", initializer="zeros")
    # Update State

    def update_state(self, y_true, y_pred, sample_weight=None):
        # Returns the index of the maximum values along the last axis in y_true (Last
    layer)
        y_true = K.argmax(y_true, axis=-1)
        # Returns the index of the maximum values along the last axis in y_true (Last
    layer)
        y_pred = K.argmax(y_pred, axis=-1)
        # Flattens a tensor to reshape to a shape equal to the number of elements
    contained
        # Removes all dimensions except for one.
        y_true = K.flatten(y_true)
        # Defines the metric for assignment
        true_poss = K.sum(K.cast((K.equal(y_true, y_pred)), dtype=tf.float32))
        self.custom_sharpe_ratio.assign_add(true_poss)
    # Metric

    def result(self):
        return self.custom_sharpe_ratio

# 4: Information Ratio (IR = [R - Rf]/SD[R-Rf])


class CustomInformationRatio(tf.keras.metrics.Metric):
    # Initialisation
    def __init__(self, num_classes=None, batch_size=None,
                 name='information_ratio', **kwargs):
        super(CustomHedgePortolfioAlphas, self).__init__(name=name, **kwargs)
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.custom_information_ratio = self.add_weight(
            name="cir", initializer="zeros")
    # Update State

    def update_state(self, y_true, y_pred, sample_weight=None):
        # Returns the index of the maximum values along the last axis in y_true (Last
    layer)
        y_true = K.argmax(y_true, axis=-1)
        # Returns the index of the maximum values along the last axis in y_true (Last
    layer)
        y_pred = K.argmax(y_pred, axis=-1)
        # Flattens a tensor to reshape to a shape equal to the number of elements
    contained
        # Removes all dimensions except for one.
        y_true = K.flatten(y_true)
        # Defines the metric for assignment
        true_poss = K.sum(K.cast((K.equal(y_true, y_pred)), dtype=tf.float32))
        self.custom_information_ratio.assign_add(true_poss)
    # Metric

    def result(self):
        return self.custom_information_ratio
################################################################################
# Autodiff Testing
################################################################################
# Information:
# TensorFlow provides the tf.GradientTape API for automatic differentiation;
# that is, computing the gradient of a computation with respect to some inputs,
# usually tf.Variables. TensorFlow "records" relevant operations executed inside
# the context of a tf.GradientTape onto a "tape". TensorFlow then uses that tape
# to compute the gradients of a "recorded" computation using reverse mode
    differentiation.
# (https://en.wikipedia.org/wiki/Automatic_differentiation)

# Function to test loss functions and metrics using autodiff


def loss_function_testing(custom_loss_function):
    """ Uses tensorflow autodifferientiation functionality
        to confirm differientable nature and feasibility
        of custom loss functions.
        Note: code verbatim from tensorflow guide.
        Merely for illustration purposes
    """
    layer = tf.keras.layers.Dense(32, activation='relu')
    x = tf.constant([[1., 2., 3.]])
    # Sets loss functions

    # Set Metrics
    with tf.GradientTape() as tape:
        # Forward pass
        y = layer(x)

        loss = tf.reduce_mean(y**2)
    # Calculate gradients with respect to every trainable variable
    try:
        grad = tape.gradient(loss, layer.trainable_variables)
    except:
        print('Gradient Function Failed')
    # Print the outcomes of the simple model analysis
    for var, g in zip(layer.trainable_variables, grad):
        print(f'{var.name}, shape: {g.shape}')
    return

# Function for implementing autodiff


def autodiff_guide(example):
    """ Execute autodiff examples from Tensorflow resources.
        Used to help gain an understanding of different
        functionalities (Demonstration Purposes Only)

    Args:
        example (int): Example to implement
                     : 1 - 'simple'
                     : 2 - 'simple_tensor'
                     : 3 - 'simple_model'
                     : 4 - 'control_tape'
                     : 5 - 'control_tensor_tape'
                     : 6 - 'stop_recording'
                     : 7 - 'watch_multiple_variables'
                     : 8 - 'higher_order_derivatives'
                     : 9 - 'jacobian'
                     : 10- 'hessian_newton'

    """
    # Uses the autodiff functionality to test custom gradients with gradient tape
    # Extracted from
    if example == 1:
        # Simple example
        print('Starting Simple Example')
        x = tf.Variable(3.0)
        with tf.GradientTape() as tape:
            y = x**2
        # dy = 2x * dx
        dy_dx = tape.gradient(y, x)
        print(dy_dx.numpy())
    if example == 2:
        w = tf.Variable(tf.random.normal((3, 2)), name='w')
        b = tf.Variable(tf.zeros(2, dtype=tf.float32), name='b')
        x = [[1., 2., 3.]]
        with tf.GradientTape(persistent=True) as tape:
            y = x @ w + b
        loss = tf.reduce_mean(y**2)
        [dl_dw, dl_db] = tape.gradient(loss, [w, b])
        print(w.shape)
        print(dl_dw.shape)
    if example == 3:
        layer = tf.keras.layers.Dense(2, activation='relu')
        x = tf.constant([[1., 2., 3.]])
        with tf.GradientTape() as tape:
            # Forward pass
            y = layer(x)
            loss = tf.reduce_mean(y**2)
        # Calculate gradients with respect to every trainable variable
        grad = tape.gradient(loss, layer.trainable_variables)
        # Print the outcomes of the simple model analysis
        for var, g in zip(layer.trainable_variables, grad):
            print(f'{var.name}, shape: {g.shape}')
    if example == 4:
        # A trainable variable
        x0 = tf.Variable(3.0, name='x0')
        # Not trainable
        x1 = tf.Variable(3.0, name='x1', trainable=False)
        # Not a Variable: A variable + tensor returns a tensor.
        x2 = tf.Variable(2.0, name='x2') + 1.0
        # Not a variable
        x3 = tf.constant(3.0, name='x3')
        with tf.GradientTape() as tape:
            y = (x0**2) + (x1**2) + (x2**2)
        grad = tape.gradient(y, [x0, x1, x2, x3])
        for g in grad:
            print(g)
        [var.name for var in tape.watched_variables()]
    if example == 5:
        x = tf.constant(3.0)
        with tf.GradientTape() as tape:
            tape.watch(x)
            y = x**2
        # dy = 2x * dx
        dy_dx = tape.gradient(y, x)
        print(dy_dx.numpy())
    if example == 6:
        # Sets the variables
        x = tf.Variable(2.0)
        y = tf.Variable(3.0)
        # Starts the graident tape
        with tf.GradientTape() as t:
            x_sq = x * x
            with t.stop_recording():
                y_sq = y * y
            z = x_sq + y_sq
        # Compute the gradient
        grad = t.gradient(z, {'x': x, 'y': y})
        # Shows tape starting and stopping with the reporting
        print('dz/dx:', grad['x'])  # 2*x => 4
        print('dz/dy:', grad['y'])
    if example == 7:
        # Set the variables
        x0 = tf.constant(0.0)
        x1 = tf.constant(0.0)
        # Establish gradient tape
        with tf.GradientTape() as tape0, tf.GradientTape() as tape1:
            tape0.watch(x0)
            tape1.watch(x1)
            # Establish sin & sigmoid functions
            y0 = tf.math.sin(x0)
            y1 = tf.nn.sigmoid(x1)
            # Create combined function, tracking multiple components
            y = y0 + y1
            ys = tf.reduce_sum(y)
    if example == 8:
        # Higher order derivatives
        x = tf.Variable(1.0)  # Create a Tensorflow variable initialized to 1.0
        with tf.GradientTape() as t2:
            with tf.GradientTape() as t1:
                y = x * x * x
        # Compute the gradient inside the outer ‘t2‘ context manager
        # which means the gradient computation is differentiable as well.
            dy_dx = t1.gradient(y, x)
        d2y_dx2 = t2.gradient(dy_dx, x)
        # Prints the result from the gradient outputs
        print('dy_dx:', dy_dx.numpy())  # 3 * x**2 => 3.0
        print('d2y_dx2:', d2y_dx2.numpy())  # 6 * x => 6.0
    if example == 9:
        # Jacobian Matrices
        x = tf.random.normal([7, 5])
        layer = tf.keras.layers.Dense(10, activation=tf.nn.relu)
        # Shape of the gradient tape
        with tf.GradientTape(persistent=True) as tape:
            y = layer(x)
        # Output Layer Shape
        y.shape
        # Shape of the kernal
        layer.kernel.shape
        # The shape of the Jacobian of the output with respect to the kernel
        # is the combination of the two shapes
        j = tape.jacobian(y, layer.kernel)
        j.shape
        # Summing over the targtes dimensions gives you the amount calculated
        # a scaler gradient
        g = tape.gradient(y, layer.kernel)
        print('g.shape:', g.shape)
        j_sum = tf.reduce_sum(j, axis=[0, 1])
        delta = tf.reduce_max(abs(g - j_sum)).numpy()
        assert delta < 1e-3
        print('delta:', delta)
    if example == 10:
        # Construction of Simple Hessian Matrix
        # A Hessian Matrix is a square matrix of 2nd order PDEs of a scaler
        # valued function, or scaler field, describing the local curvature of
        # a multivariate function
        x = tf.random.normal([7, 5])
        layer1 = tf.keras.layers.Dense(8, activation=tf.nn.relu)
        layer2 = tf.keras.layers.Dense(6, activation=tf.nn.relu)
        with tf.GradientTape() as t2:
            with tf.GradientTape() as t1:
                x = layer1(x)
                x = layer2(x)
                loss = tf.reduce_mean(x**2)
            g = t1.gradient(loss, layer1.kernel)
        h = t2.jacobian(g, layer1.kernel)
        print(f'layer.kernel.shape: {layer1.kernel.shape}')
        print(f'h.shape: {h.shape}')
        # Flatten axes into matrix and flatten to gradient vector
        n_params = tf.reduce_prod(layer1.kernel.shape)
        g_vec = tf.reshape(g, [n_params, 1])
        h_mat = tf.reshape(h, [n_params, n_params])
        # Define function to display hessian matrix

        def imshow_zero_center(image, **kwargs):
            lim = tf.reduce_max(abs(image))
            plt.imshow(image, vmin=-lim, vmax=lim, cmap='seismic', **kwargs)
            plt.colorbar()
        # Shows the hessian matrix
        imshow_zero_center(h_mat)
        # Newton's Method Update Step
        eps = 1e-3
        eye_eps = tf.eye(h_mat.shape[0])*eps
        # X(k+1) = X(k) - (   f   (X(k)))^-1 @   f  (X(k))
        # h_mat =    f   (X(k))
        # g_vec =   f  (X(k))
        update = tf.linalg.solve(h_mat + eye_eps, g_vec)
        # Reshape the update and apply it to the variable.
        _ = layer1.kernel.assign_sub(tf.reshape(update, layer1.kernel.shape))
    return
################################################################################
# Analytical/Calculus
################################################################################
# Writes functions


def analytical_analysis():
    # Test simple functionality
    print(sym.sqrt(8))
    theta, x = sym.symbols('O X')
    return


def ranking_function():
    """ Ranking function to produce charts for demonstration purposes

    Args:
        type ([type]): String for desired ranking functions
    """
    # Creates an ordered, random array of proxy returns (%)
    num = 100
    returns_uniform = np.sort(np.arange(-10, 10, -0.2))
    print('returns', returns_uniform)
    print('returns size', np.size(returns_uniform))
    returns = np.sort(np.random.uniform(low=-10.0, high=10.0, size=(num,)))
    # returns = returns[::-1].sort
    base = np.zeros(num)
    ones = np.ones(num)
    # Creates rank array
    rank = np.linspace(num, 1, num)
    # Sets thresholds
    u = np.zeros((rank.shape))
    u[:] = 20
    v = np.zeros((rank.shape))
    v[:] = 80
    # rank = np.array(list(range(1,len(returns)+ 1)))
    # Create weights
    weights = returns/transpose(ones)
    print('weights', weights)
    print('Sum of weights', np.sum(weights))
    weights = weights*returns
    print('weights', weights)
    print('Sum of weights', np.sum(weights))
    # Plots the functions
    plt.plot(returns, rank, 'r.', base, rank, 'k.',
             returns, u, 'g--', returns, v, 'b--')
    # Invert the y-axis
    plt.gca().invert_yaxis()
    plt.gca().invert_xaxis()
    plt.legend('Returns', 'Baseline')
    plt.xlabel('Excess Return (y(i,t), %)')
    plt.ylabel('Rank (R(y(i,t)))')
    plt.title('Monotonic Ranking Function')
    plt.savefig(
        '/home/connormcdowall/finance-honours/results/plots/monotonic-ranking.png')
    return

```

```python

###############################################################################
# Variables
###############################################################################
# Integers
batch_size = 256  # Batch size for creating tf dataset
chunk_size = 100000  # chunk size for reading stata files
# Targets
targets_dictionary = {1: 'ret_exc', 2: 'ret_exc_lead1m'}
# Sets the intended target column (test multiple configurations)
target_column = targets_dictionary[2]
# Lists and arrays
# 1: , 2: , 3:
resizing_options = [True, True, True]
categorical_assignment = ['size_grp', 'permno', 'permco',
                          'crsp_shrcd', 'crsp_exchcd', 'adjfct', 'sic', 'ff49']
# Tensorflow configurations (listed for completeness/reference)
# Optimizers
optimizers = ['Adagrad', 'Adadelta', 'Adam',
              'Adamax', 'Ftrl', 'Nadam', 'RMSprop', 'SGD']
# Losses
binary_classification_losses = ['binary_crossentropy']
multiclass_classfication_losses = ['categorical_crossentropy',
                                   'sparse_categorical_crossentropy', 'poisson', 'kl_divergence']
regression_losses = ['cosine_similarity', 'mean_absolute_error', 'mean_absolute_percentage_error',
                     'mean_squared_logarithmic_error', 'mean_squared_error', 'huber_loss']
extra_losses = ['hinge', 'log_cosh', 'loss', 'squared_hinge']
custom_losses = ['custom_l2_mse', 'custom_hedge_portfolio_returns', 'custom_sharpe_ratio',
                 'custom_information_ratio', 'custom_loss']  # List names here when created
losses = binary_classification_losses + multiclass_classfication_losses + \
    regression_losses + extra_losses + custom_losses
# Metrics (Functions used to judge model performance,similar to a loss function but results are not used when training a model)
accuracy_metrics = ['accuracy', 'binary_accuracy', 'categorical_accuracy',
                    'top_k_categorical_accuracy', 'sparse_top_k_categorical_accuracy', 'sparse_categorical_accuracy']
probabilistic_metrics = ['binary_crossentropy',
                         'categorical_crossentropy', 'kullback_leibler_divergence']
regression_metrics = ['root_mean_squared_error', 'mean_absolute_percentage_error', 'mean_metric_wrapper', 'sum',
                      'mean_relative_error', 'mean_squared_error', 'mean_squared_logarithmic_error', 'cosine_similarity', 'logcosh', 'mean', 'mean_absolute_error', 'mean_tensor', 'metric']
classification_tf_pn = ['Auc', 'Fn', 'Fp', 'poisson', 'precision', 'precision_at_recall',
                        'recall', 'recall_at_precision', 'sensitivity_at_specificity', 'Tn', 'Tp']
images_segementation_metrics = ['meaniou']
hinge_metrics = ['categorical_hinge', 'squared_hinge', 'hinge']
custom_metrics = ['hedge_portfolio_mean', 'hedge_portfolio_alphas',
                  'sharpe_ratio', 'information_ratio']  # Add when create the metrics
metrics = accuracy_metrics + probabilistic_metrics + regression_metrics + \
    classification_tf_pn + images_segementation_metrics + hinge_metrics + custom_metrics
# Tensorflow congifuration
optimisation_dictionary = {1: 'SGD', 2: 'SGD',
                           3: 'SGD', 4: 'SGD', 5: 'SGD', 6: 'SGD'}
loss_function_dictionary = {1: 'mean_squared_error', 2: 'custom_l2_mse', 3: 'custom_hedge_portfolio_returns',
                            4: 'custom_sharpe_ratio', 5: 'custom_information_ratio', 6: 'custom_loss'}
metrics_dictionary = {1: ['mean_squared_error'], 2: ['mean_squared_error'], 3: [
    'mean_squared_error'], 4: ['mean_squared_error'], 5: ['mean_squared_error'], 6: ['mean_squared_error']}
# Selected Tensorflow Configuration
###############################################################################
tf_option = 6  # Change to 1,2,3,4,5 for configuration
selected_optimizer = optimisation_dictionary[tf_option]
selected_loss = loss_function_dictionary[tf_option]
```

```
selected_metrics = metrics_dictionary[tf_option]
################################################################################
# Strings
model_name = 'finance-honours-test'
data_source = 'data/combined_predictors_filtered_us.dta'
csv_location = '/Volumes/Seagate/dataframes/'
data_vm_directory = '/home/connormcdowall/local-data/'
data_vm_dta = '/home/connormcdowall/local-data/combined_predictors_filtered_us.dta'
results_tables = '/home/connormcdowall/finance-honours/results/tables'
list_of_columns = '/home/connormcdowall/finance-honours/data/working-columns.txt'
# Binary (Set to True or False depending on the functions to run)
# System Checks
sys_check = False
# Data processing
source_data = False
split_vm_data = False
process_vm_data = False
use_sass = False
need_dataframe = False
# Tensorflow
assign_features = False
extract_test_data = False
test_implementation = False
example_autodiff = True
test_loss_function = False
# Analytical
analytical = False
rank_functions = False
# Research Proposal Analysis
begin_analysis = True
################################################################################
# Function Calls - Testing
################################################################################
# System Checks
################################################################################
if sys_check:
    reconfigure_gpu(restrict_tf=False, growth_memory=True)
################################################################################
# Data processing
################################################################################
# Source data from local drive
if source_data:
    partition_data(data_source, csv_location)
# Source data from VM Instance
if split_vm_data:
    split_vm_dataset(data_vm_directory, create_statistics=False,
                     split_new_data=False, create_validation_set=False)
# Process vm data for Tensorflow
if process_vm_data:
    process_vm_dataset(data_vm_dta, save_statistics=False, sample=False)
if need_dataframe:
    data = create_dataframes(csv_location, False)
    print(data.info())
    print(data.head())
if use_sass:
    sass_access(data)
################################################################################
# Tensorflow
################################################################################
if assign_features:
    numerical_features, categorical_features = create_feature_lists(
        list_of_columns, categorical_assignment)
if extract_test_data:
    df, train_data, val_data, test_data = download_test_data()
    if test_implementation:
        implement_test_data(df, train_data, val_data,
                            test_data, full_implementation=True)
if example_autodiff:
    autodiff_guide(example=5)
if test_loss_function:
    print('Add Function Here')
################################################################################
# Analytical
```

```
##############################################################################
# Analytical function
# Do analytical function
if analytical:
    analytical_analysis()
# Creates monotonic ranking function plots
if rank_functions:
    ranking_function()
###############################################################################
# Function Call - Analysis
###############################################################################
if begin_analysis:
    project_analysis(data_vm_directory, list_of_columns, categorical_assignment,
    target_column, chunk_size, resizing_options,
                     batch_size, model_name, selected_optimizer, selected_loss,
    selected_metrics, split_data=False, trial=True, sample=True)
```