UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA

ISCTE
INSTITUTO UNIVERSITÁRIO DE LISBOA
DEPARTAMENTO DE FINANÇAS

Análise de Risco de Crédito usando algoritmos de *Machine Learning*

Pedro Miguel Pinhal Pereira

Mestrado em Matemática Financeira

Dissertação orientada por:
Diana Aldea Mendes

2020

Agradecimentos

Aos meus pais, obrigado pelo carinho, apoio e todo o amor que nunca me faltou. Pela educação que me proporcionaram e que eu tenciono passar para os meus filhos, da mesma forma que vocês o fizeram comigo. Nem sempre tivemos as condições financeiras ideias, mas tanto eu como o meu irmão não nos podemos queixar de nos ter faltado nada. Sei o esforço financeiro e físico que fizeram, para que nós tivéssemos condições para fazer os nossos cursos.

Ao meu irmão, por nunca me deixar desistir, pelos conselhos de irmão mais velho e por ser a pessoa que sempre quis ser. Sempre foste o modelo de ser humano que quis ser e sempre serás o meu ídolo.

À minha namorada, por todo o apoio e suporte desde do primeiro dia. Pelo amor e confiança que sempre teve em mim e em nós. Por acreditar em mim e no nosso projeto de vida e por me fazer sentir sempre importante para alguém.

À professora Diana, por toda a ajuda e orientação desta dissertação. Por estar sempre disponível para falar comigo e me ajudar em tudo o que foi sendo preciso. Foi um prazer trabalhar consigo e tê-la como minha orientadora.

Ao meu grupo de amigos, que também não me deixaram desistir de escrever esta dissertação e sempre me motivaram a fazê-lo, desta forma, terminando o meu mestrado. Em especial, ao Francisco por toda a ajuda que me deu nas dúvidas relativas à programação da linguagem Python.

Resumo

A presente dissertação resulta da necessidade de se classificar empresas consoante o seu nível de risco de crédito. Para tal, será desenvolvido um modelo que tem como *input* as demonstrações financeiras de uma empresa, classificando-a através do *rating* da sua dívida com a nomenclatura da Standard & Poor's. Com o intuito de alcançar o objetivo anteriormente definido, considerei um conjunto de dados da CRSP (Center for Research in Security Prices, LLC), sendo a amostra inicial composta por 3320 observações das demonstrações financeiras anuais de diversas empresas que constituem o índice bolsista S&P500, no intervalo temporal de 2010 a 2018. Estes dados foram trabalhados na linguagem de programação Python, utilizando a aplicação Jupyter Notebook, com objetivo de criar, treinar e testar este modelo de *Credit Scoring*, procedendo à utilização de diversos algoritmos de *Machine Learning*. Para obter uma melhor *performance* no modelo, foram usados métodos para a seleção das variáveis pela importância que tinham na classificação do modelo, tendo reduzido as variáveis numéricas de 69 para 20. A capacidade de previsão/acerto dos diversos algoritmos foram comparadas e o melhor algoritmo (*Random Forest*, o que teve maior percentagem de *accuracy*) foi escolhido e utilizado para a previsão do modelo. Devido à pouca diversidade de *ratings* das empresas do S&P500, uma vez que existem poucas empresas com *ratings* baixos, próximos do nível de *default*, o modelo criado tornou-se num modelo binário e a classificação foi reduzida a *Investment grade* (de AAA até BBB-) e *Non-Investment grade* (de BB+ até CC).

Palavras-chave:

Credit Scoring, *Machine Learning*, *ratings*, *Random Forest*, Risco de Crédito, Standard & Poor's, Demonstrações Financeiras.

Abstract

This thesis results from the need to classify companies according to their level of credit risk. And, for this purpose, a model will be developed that taking as input the financial statements of a certain company will return the rating of its debt using the nomenclature from Standard & Poor's. In order to achieve the previously defined goal, a dataset from CRSP (Center for Research in Security Prices, LLC) was considered with an initial sample of 3320 values of the annual financial statements of several companies that are integrated in the S&P500 stocks index, in the time interval from 2010 to 2018. This dataset was prepared and modified in the programming language Python, using the application Jupyter Notebook, with the goal of creating, training and testing this Credit Scoring model, proceeding with the use of several Machine Learning algorithms. With the purpose of obtaining a better performance in the model, it was produced features selection models, based on their importance for the classification model, and the features were reduced from 69 to only 20 variables. The prediction/accuracy of the various algorithms were compared, and the best algorithm (Random Forest, which had the highest percentage of accuracy) was chosen and used to predict the model. Due to the little diversity of ratings of the S&P500 companies, since there are few companies with low ratings, close to the default level, the model created became a binary model, and the rating was reduced to Investment grade (from AAA to BBB-) and Non-Investment grade (from BB+ to CC).

Keywords:

Credit Scoring, Machine Learning, ratings, Random Forest, Credit Risk, Standard & Poor's, Financial Statements.

Lista de Figuras

Índice

Introdução

A evolução da Inteligência Artificial e o uso de um dos seus grandes componentes, o *Machine Learning*, tem sido exponencial e abrangente a todas as áreas científicas. Para a área financeira em particular, o *Machine Learning* tem vindo a obter uma importância enorme na criação de novos modelos de previsão e classificação, no crescimento das Instituições Financeiras e nas tomadas de decisão relativas às suas estratégias comerciais.

Devido ao extenso número de trabalhos nesta área da Inteligência Artificial tentei desenvolver um tema que, em Portugal e pela pesquisa feita, ainda não foi tratado. Esta temática diz respeito à classificação do *rating* das empresas utilizando *Machine Learning* e métodos de *Credit Scoring*. Portanto, este tema será pioneiro no sentido em que pretendo atribuir uma classificação de *rating* com base nas demonstrações financeiras de empresas e em rácios construídos através destas. É necessário realçar ainda que este trabalho foi produzido na linguagem de programação Python, utilizando a aplicação Jupyter Notebook (o código encontrar-se-á nos Anexos).

No primeiro capítulo do relatório encontra-se a revisão bibliográfica, abordando os trabalhos com maior semelhança à tese em produção, diferenciando-os da mesma. Os que se relacionam com a temática desenvolvida incidem sobre uma classificação binária com o intuito de descortinar se uma empresa se encontra em *default* ou não, ou estão direcionados a pessoas individuais (créditos à habitação) e não a clientes institucionais. O capítulo seguinte é referente aos conceitos abordados ao longo do trabalho, estando uns mais relacionados com a área financeira (Risco de Crédito e *Credit Scoring*) e outros com o tema Métodos Econométricos (*Machine Learning* e seus algoritmos). No último capítulo apresenta-se a Análise Empírica, evidenciando todos os resultados obtidos no desenvolvimento do modelo. Este capítulo está divido em três partes: a preparação dos dados, a criação do modelo e seleção dos algoritmos, e a seleção das variáveis e melhoramento do algoritmo escolhido com a respetiva análise dos resultados.

1. Capítulo I
1.1. Revisão Bibliográfica

Neste primeiro capítulo abordam-se dois *papers* e uma dissertação relacionados com o tema desta tese, apresentando-se um breve resumo de cada um deles. Para além disto, comparam-se informações que permitem encontrar semelhanças e diferenças entre esta dissertação e os documentos analisados.

O primeiro artigo escrito por Addo, P., Guegan, D e Hassani, B. em [1] foca-se, resumidamente, em desenvolver um algoritmo que responda à questão se se deve emprestar capital a uma certa empresa. Foi relevante para os criadores deste *paper* sublinhar a importância que a escolha dos algoritmos, dos parâmetros e das variáveis têm na tomada de decisão, um pouco à semelhança da tese que estou a desenvolver. Ao contrário deste artigo, que se foca num modelo de *Credit Scoring* que desenvolve diversos algoritmos de *Machine learning* e *Deep Learning* para identificar casos de *default* em empresas, esta dissertação focar-se-á em classificar uma empresa quanto ao seu risco, usando *Credit Scoring* e também algoritmos de *Machine Learning* e *Deep Learning*, tal como este *paper*. Relativamente ao resultados, os algoritmos que tiveram melhor *performance* foram *Decision Trees*, *Gradient Boosting* e *Random Forest*, enquanto que a os modelos de *Deep Learning*, não foram tão eficazes para este problema binário, de classificar a empresas como estando em *default* ou não. Isto porque os três primeiros modelos apresentaram um valor mais elevado de AUC (Area Under the Curve – ver ponto 3.1.2.) e mais reduzido de RMSE (Root Mean Square Error – ver ponto 3.1.2.).

O segundo *paper* redigido por Ha, V. e Nguyen, H. em [2] foi elaborado com um intuito semelhante ao primeiro, no que toca à questão de usar um modelo *Credit Scoring*. Porém, este aplica-se ao o risco de crédito para pessoas individuais, e não ao risco de crédito para empresas, focando assim o seu estudo nos métodos de seleção das *features*, isto é, otimizando o número de variáveis com um valor aceitável de *accuracy*; e no uso de modelos *Deep Learning* para classificar dois conjuntos de dados - um que contém 1000 empréstimos de indíviduos na Alemanha, 700 que foram aprovados e 300 que não foram; e o outro 690 empréstimos de pessoas na Austrália, 383 aprovados e 307 não aceites. Em relação aos resultados, a *accuracy* dos dois conjuntos de dados subiu depois de ter sido feita a seleção das variáveis, ou seja, com menos *features* o modelo consegue prever melhor. Para além disto, o método foi processado pelo computador muito mais rápido, em ambos os casos, produzindo uma melhor previsão e sendo mais eficaz em termos de tempo de previsão.

Por sua vez, o tema da dissertação desenvolvida em 2019 por Hani, M., Mohamed, A. e ElMasry, T. em [3] tem pontos em comum com o segundo artigo, uma vez que desenvolve um modelo que avalia e prevê a adequabilidade de um crédito à habitação, e classifica os créditos binariamente, como *default loans* e *non-default loeans*. De acordo com a amostra deste trabalho foi utilizada uma base de dados fornecida pela Freddie Mac, sendo desenvolvidas diversas metodologias de *Machine Learning*, nomeadamente: Regressão Logística, *Regression Trees*, *Random Forest*, *K-Nearest Neighbors*, e *Support Vector Machine*. Além disto, foi ainda introduzida uma abordagem de meta-algoritmos, também designada por *stacking*, que aprende a escolher as previsões dos vários modelos acima descritos. Em

termos de resultados, esta última técnica foi a que teve melhor *accuracy* (89%) comparando com os restantes modelos de *Machine Learning*.

Concluindo, devido à pesquisa realizada, importa referir que não existem trabalhos em Portugal, pela pesquisa que foi feita, que sejam idênticos ao que irá ser realizado nesta dissertação. Isto é, não existem artigos ou teses que façam a classificação de uma empresa no que toca ao seu *rating* de dívida (utilizando a nomenclatura da Standard and Poor's) com *Credit Scoring* e usando algoritmos de *Machine Learning*.

2. Capítulo II

2.1. Conceitos Base

2.1.1. Risco de Crédito

Risco é a incerteza que um acontecimento futuro carrega, significa estar exposto a algum tipo de incerteza. Na área financeira, por seu turno, é a volatilidade de um resultado futuro e inesperado. Apesar de ter uma conotação negativa, no que toca à população em geral, eventos com incerteza podem levar a resultados mais positivos, do que se não estivéssemos expostos a esse risco. É um preço a pagar e uma relação muito difícil de conjugar, no sentido de quanto risco estou disposto a correr com expectativa de retorno. Esta relação depende da personalidade dos investidores e da sua aversão ao risco.

Existem dois tipos de risco: o risco financeiro e o risco não financeiro. Dentro do primeiro, temos outros três grandes grupos: Risco de Mercado, que se define pela a possibilidade de perda da carteira de ativos de um banco, resultando das variações dos preços de mercado desses ativos; Risco de Liquidez, que reflete o risco de uma empresa não respeitar as suas responsabilidades, seja por falta de recursos ou por não ter capacidade de liquidar ativos que têm em carteira, para fazer face a dívidas existentes; e finalmente, Risco de Crédito que é a probabilidade de uma contraparte (devedor) não ser capaz de pagar as suas obrigações, entrando assim em *default*. Isto vai implicar uma perda de valor da entidade credora por não receber os pagamentos que o devedor teria de pagar.

Tendo em conta que esta dissertação aborda apenas o conceito Risco de Crédito, acrescenta-se que este diz respeito à probabilidade de uma dada contraparte entrar em *default* num pagamento relativo habitualmente a ativos como obrigações ou empréstimos bancários.

Neste sentido, é necessário esclarecer alguns conceitos, a fim de se perceber concretamente qual a perda que se espera vir a ter caso a contraparte entre em *default*:

a) *Probability of Default* (PD) - é simplestamente a probabilidade de o devedor entrar em situação de *default*, e não cumprir com as suas obrigações. Em termos mais financeiros, é a probabilidade de os ativos da carteira do devedor ser menor que o seu passivo, fazendo os capitais próprios serem negativos (Equação Fundamental da Contabilidade: Ativos = Capital Próprio + Passivo).
b) *Exposure at Default* (EAD) - é o valor que o credor pode vir a perder se o devedor entrar em *default* sem o credor ter qualquer recuperação do montante investido.
c) *Loss Given Default* (LGD) - trata-se da percentagem do montante que o credor irá perder. Visto que, habitualmente, quando uma contraparte entra em default, uma parte do valor investido é reavido pela contraparte credora. A percentagem que é recuperada define-se por *Recovery Rate* (RR = 1 - LGD).

Com os conceitos apresentados e definidos estamos em condições de calcular a perda esperada (*Expected Loss*, EL), através da seguinte equação:

$$EL = PD \times EAD \times LGD \tag{2.1}$$

No sentido da contraparte credora obter o valor da perda esperada, é preciso estimar os *inputs* mencionados anteriormente: a EAD e o LGD são fáceis de conhecer, visto que a EAD é apenas o valor que o credor pode vir a perder no caso de *default* da contraparte devedora, e sabendo a *Recovery Rate* a LGD fica simples de calcular. Por outro lado, a PD é o pressuposto desta equação mais complexo e com diferentes métodos de obter. É aqui que se encontra o problema de obter o Risco de Crédito associado a um ativo ou a uma carteira de ativos, pois a forma como se obtém esta probabilidade difere bastante, existindo diversas formas de a calcular.

A nomenclatura habitualmente usada para classificar o Risco de Crédito que uma empresa possuí é definido por três agências de *rating*: a Moody's, a Fitch e a Standard & Poor's. A Moody's é a única que tem uma nomenclatura diferente (Figura 2.1). Apesar das diferentes classes entre si, o valor intrínseco é o mesmo, por exemplo, uma classificação de AAA na Fitch é equivalente a Aaa na Moody's. Mas, se a Fitch define que uma certa empresa tem AAA, a Standard & Poor's não tem de dar a mesma classificação, nem a Moody's tem de dar Aaa. Os modelos de cálculo são diferentes, apesar da nomenclatura ser igual, e no caso da Moody's ser equivalente. Esta classificação está dividida para todas as agências em duas classes: *Investment grade* e *Non-Investment grade*. Dentro da primeira classe, temos ainda três subclasses: *Low-medium grade*, *Upper-medium grade* e *High-quality grade*. Na segunda classe, existem apenas duas subclasses: *Low grade* e *Default*.

Para esta dissertação o modelo de avaliação de Risco de Crédito será *Credit Scoring*, usando uma base de dados do S&P500 com a classificação de ratings da Moody's que depois será convertida na nomenclatura da Standard & Poor's.

TABLE 3.1.: Add caption

		Moody's	S\&P	Fitch
Investment grade	High-quality grade	Aaa	AAA	AAA
		Aa1	AA+	AA+
		Aa2	AA	AA
		Aa3	AA−	AA−
	Upper-medium grade	A1	A+	A+
		A2	A	A
		A3	A−	A−
	Low-medium grade	Baa1	BBB+	BBB+
		Baa2	BBB	BBB
		Baa3	BBB−	BBB−
Non-investment grade	Low grade or speculative grade	Ba1	BB+	BB+
		Ba2	BB	BB
		Ba3	BB−	BB−
		B1	B+	B+
		B2	B	B
		B3	B−	B−
		Caa1	CCC+	CCC+
		Caa2	CCC	CCC
		Caa3	CCC−	CCC−
		Ca	CC	CC
		C		C
	Default	C	D	D

Figura 2.1 – Agências de rating e suas nomenclaturas – Credit Risk Lecture Notes – Professor José Carlos Dias

2.1.2. *Credit Scoring*

No âmbito da criação de um modelo de avaliação de Risco de Crédito existem diversos caminhos que podem ser escolhidos pelo analista, isto é, diferentes tipos de modelos que podem ser desenvolvidos mediante o objetivo da análise. Os métodos estruturais enquadram o modelo de Merton, o modelo da Moody's KMV[1], o modelo *Creditgrades* e, o que irá ser usado nesta dissertação, *Credit Scoring*.

O modelo de *Credit Scoring* é obtido por uma análise de vários métodos econométricos que usam diversos parâmetros, variáveis explicativas, que neste caso são dados finaceiros de empresas. Para desenvolver um modelo de avaliação de Risco de Crédito usando *Credit Scoring* há que seguir cinco passos: seleção da amostra, recolha de dados financeiros, seleção e cálculo de rácios relevantes para o desenvolvimento do modelo, e no fim, análise de resultados empíricos e principais conclusões.

A seleção da amostra é uma tarefa bastante complexa, uma vez que é difícil encontrar empresas em estado de falência e que contenham as demonstrações financeiras que comprovam esse estado. É também importante conter no conjunto de dados empresas que não estejam em *default*, apesar de esta informação ser mais abundante e fácil de encontrar. Algumas empresas possuem estas informações para desenvolver modelos de *rating*, como é o exemplo da Moody's, da Fitch e da Standard & Poor's, sendo que estes não estão disponíveis para a população em geral.

Na recolha de dados financeiros é importante ter uma base de dados diversficada que tenha empresas em *default*, empresas em mau estado financeiro mas que ainda não estão em falência e empresas que estejam com saúde financeira. Os dados financeiros da base de dados são das Demonstrações Financeiras: Balanço e Demonstração de Resultados das diversas empresas escolhidas na seleção da amostra. Para a minha dissertação a base de dados que usarei é da CRSP (*Center for Research in Security Prices*, LLC) e são as demonstrações financeiras das empresas que constituem o S&P500 de 2010 até 2018.

A seguinte etapa é o cálculo de rácios relevantes para análise e desenvolvimento do modelo de *Credit Scoring*. Não existe nenhum critério definido para quais os rácios mais relevantes, depende sempre do âmbito e objetivo do modelo a criar. Por exemplo, Beaver em [10] selecionou uma lista de 30 rácios enquanto que Altman em [11] usou apenas 22.

Nesta tese os rácios calculados têm por base os dados das demonstrações financeiras. Inicialmente irei utilizar 72 variáveis (contendo diversas contas das Demonstrações Financeiras e alguns rácios financeiros), que na etapa da escolha de *features* pelo modelo desenvolvido ficarão apenas 20. Os rácios foram escolhidos com base na literatura recolhida e analisada ([1] [4]) e construídos através das variáveis.

Às duas últimas fases - desenvolvimento do modelo de *Credit Scoring* e obtenção dos resultados empíricos e principais conclusões – corresponde a criação e análise do *software*, a obtenção dos

[1] A Moody's adquiriu a empresa KMV (sigla do nome dos seus fundadores: Kealhofer, McQuown, and Vasicek) que era líder em fornecer ferramentas quantitativas de análises de crédito para investidores.

resultados, a análise crítica e as diversas conclusões dos mesmos, que se encontram aprofundadas no próximo capítulo.

2.2. Métodos Econométricos

2.2.1. Introdução à Econometria dos Mercados Financeiros

A Econometria dos Mercados Financeiros e as suas técnicas foram desenvolvidas no âmbito de problemas práticos e do quotidiano das pessoas, para conseguir soluções para estes problemas que provinham dos vários ramos da economia e das finanças. A econometria está dividida numa parte teórica, onde se criam novos métodos e onde são realizados estudos das propriedades que a compõe; e numa parte mais aplicada a situações reais, onde são desenvolvidos e aplicados mecanismos para solucionar as questões provenientes destes problemas. Tal como o nome indica, a primeira direciona-se a questões teóricas e problemas académicos, enquanto a segunda está mais relacionada com a procura de soluções para problemas baseados em dados empíricos.

Para fazer uma análise econométrica é necessário passar por um processo definido, onde primeiro há a especificação do modelo. Depois formulamos as condições sobre os parâmetros intrínsecos ao modelo. De seguida recolhemos a amostra e refinamos os dados que a compõem. O passo seguinte é identificar e estimar os parâmetros para o modelo. Continuamos para os testes estatísticos, onde trabalhamos os parâmetros com os demais testes, e por fim, concluímos com a interpretação económica, onde fazemos a análise dos resultados obtidos nos testes com uma base económica, respondendo aos problemas que foram identificados.

A Econometria Financeira é muito utilizada atualmente nas Finanças empíricas, tais como, *forecasting*, integração financeira, microestrutura do mercado e avaliação de risco. É usada para explicar e eventualmente prever os efeitos dinâmicos dos vários choques financeiros que provêm dos mecanismos de transmissão das crises financeiras.

No sentido de realizar estas análises, previsões ou estudos econométricos temos de conhecer os dados que temos, qual o seu tipo ou natureza. O tipo de dados a utilizar varia consoante o problema em análise e os objetivos que são delineados para o estudo. Existem diversos tipos de dados, nomeadamente, quantitativos (exemplo: preços de ações de uma empresa cotada em bolsa), qualitativos (exemplo: o *rating* da dívida de uma empresa), séries temporais, dados *cross-section* (dados que são caracterizados por pelo menos uma variável observada em apenas um momento do tempo), e finalmente, dados em painel (dados *cross-section* observados em diversos momentos do tempo).

Para a análise de séries temporais existe um conjunto de metodologias e modelos fundamentais que são atualmente utilizados, como por exemplo: modelos de regressão linear e não linear (com estimação pelo método dos Mínimos Quadrados ou Máxima Verosimilhança), modelos de séries temporais autoregressivos (captam padrões como tendência, ciclos, volatilidade, não-linearidade, entre outros) e outros modelos dinâmicos (processos de transferência).

2.2.2. Regressão e Correlação

No intuito de realizar modelos de regressão é necessário ter presentes e bem definidos os conceitos de regressão e correlação. Regressão é simplesmente o resultado que se obtém pela estimação de uma equação linear que descreve o relacionamento da variável dependente (y) com as variáveis independentes ou também designadas de variáveis explicativas (x_i). Enquanto que a Correlação é a medida que descreve a relação entre as variáveis, ou seja, quantifica a força da relação entre as variáveis, sem nos indicar qual é a dependente ou independente.

A regressão linear é uma ferramenta usada na estatística para modelar e prever a relação entre variáveis e está baseada num modelo probabilístico que diz que uma variável dependente (y) é composta por uma componente determinística ($\beta_0 + \beta_1 x + \cdots + \beta_n x$) e um erro aleatório ($\varepsilon$). Este erro aleatório tem uma distribuição normal com média 0 e variância σ^2. O β_0 é o ponto de interseção da equação do modelo com o eixo das ordenadas, enquanto que os restantes β são os coeficientes e parâmetros da regressão linear. Os betas indicam a contribuição que a variável explicativa tem na variável dependente (y), ou seja, representam a sensibilidade que este fator tem quando a respetiva variável explicativa varia com o pressuposto de que as restantes se mantiveram fixas. A regressão linear simples contém apenas um regressor que explica a variável dependente, enquanto que regressão linear múltipla tem *n* variáveis independentes e *n* parâmetros que poderão explicar com melhor exatidão a variabilidade da variável dependente.

Para conduzir a elaboração de um modelo de regressão linear é necessário recorrer a um processo com quatro etapas: desenvolve-se o modelo, estimam-se os parâmetros, valida-se o modelo e por fim usa-se o mesmo para responder ao problema inicial. Primeiramente, há que fazer uma análise gráfica, onde se colocam as variáveis da amostra num diagrama de dispersão, observando posteriormente a correlação entre elas. Ainda nesta etapa assumimos que existe uma relação linear ($y = \beta_0 + \beta_1 x + \cdots + \beta_n x + \varepsilon$). O próximo passo é usar a amostra para estimar os parâmetros que não são conhecidos no modelo, fazendo assim ajustes ao mesmo (utilizando, por exemplo, o método dos mínimos quadrados – OLS). A terceira fase é a identificação da distribuição de probabilidades do termo aleatório (erro), da estimação da volatilidade ou desvio padrão desta distribuição, da validação do modelo (estudando o quão adequável o modelo é do ponto de vista estatístico) e da validação dos pressupostos inerentes. Na última fase utiliza-se o modelo para fazer a estimação e predição dos resultados, analisando-os e tirando conclusões do problema que foi encontrado, cuja solução é o objetivo do desenvolvimento deste modelo.

Na análise gráfica o processo é bastante simples, pois apenas há que dispor de pares de variáveis num gráfico de dispersão e perceber se existe correlação ou não entre as variáveis. Se em teoria as variáveis são independentes, então gráfico deve estar muito disperso na direção horizontal, quase como se tratasse de uma mancha de pontos aleatórios. Na prática encontra-se frequentemente correlação entre variáveis explicativas (multicolinearidade), conduzindo à eliminação de algumas com o objetivo de uma estimação não-enviesada e eficiente, relativamente aos seus parâmetros.

No passo da estimação de parâmetros, se estivermos numa regressão linear múltipla (que depois se pode estender para a regressão linear simples, visto que esta última difere no número de variáveis explicativas) podemos usar o Método dos Mínimos Quadrados. O objetivo deste método é definir os estimadores dos parâmetros que irão minimizar a soma dos quadrados dos desvios (SSE, resíduos da

regressão), originando um sistema de *k+1* equações lineares ou normais. Esta minimização resulta da derivada do SSE em ordem a cada beta do modelo que estamos a produzir. A solução deste sistema ($X^T X\beta = X^T Y$) fornece os estimadores dos mínimos quadrados do vetor beta (solução: $\widehat{\beta} = (X^T X)^{-1} X^T Y$).

Na terceira fase realizam-se testes de hipótese para determinar quais as variáveis explicativas relevantes para o modelo. Se existirem variáveis que tenham pouca importância ou até nenhuma, a solução é eliminá-las. Para este efeito, definimos a hipótese nula (H_0) e a hipótese alternativa (H_1) da seguinte forma: $H_0: \beta_i = 0$ e $H_1: \beta_i \neq 0$. Isto é, a hipótese nula indica que o coeficiente da variável é 0, enquanto que a hipótese alternativa diz-nos que este coeficiente não é nulo. Assim, se rejeitarmos a hipótese nula, então significa que a variável testada é significativa para explicar *y*, logo tem relevância para o modelo. Se não rejeitarmos, então esta não tem valor contributivo para explicar *y*, e por isso, elimina-se da análise do modelo que estamos a implementar.

De seguida validam-se os pressupostos definidos para o modelo: a média da distribuição de probabilidades do resíduo ser zero, os resíduos serem homocedásticos (têm a variância constante), não existir autocorrelação entre os resíduos, as variáveis explicativas serem independentes dos erros, e finalmente, os resíduos terem distribuição normal. Para a verificação do primeiro pressuposto o erro tem de ter média igual a 0 e é sempre validado se o modelo que estamos a desenvolver tiver um termo constante diferente de 0. Para a validação da homocedasticidade existem vários testes: o teste de Goldfeld-Quandt, o teste de Breusch-Pagan e o teste de White, todos contendo a mesma hipótese nula H_0: erros homocedásticos. Assim, o resultado pretendido é não rejeitar esta hipótese para que os erros do modelo sejam homocedásticos, ou seja, tenham variância constante. Para o seguinte pressuposto também existem testes estatísticos para validar a não existência de autocorrelação entre resíduos, nomeadamente, o Teste de Durbin-Watson e o teste de Breusch-Godfrey. Caso o primeiro teste falhe, procedemos à realização do segundo, pois este é mais completo e mais consistente do que o primeiro. Por fim, para verificar se os resíduos têm distribuição normal, desenvolve-se o teste de Jarque-Bera, que tem como Hipótese nula H_0: distribuição normal.

Para finalizar a regressão linear, na última etapa utilizam-se os estimadores, prevêem-se os resultados e analisamos os mesmos. O valor indicativo da qualidade do modelo é o coeficiente de determinação R^2, que varia entre 0 e 1. Quanto maior for este coeficiente melhor é o modelo e melhores serão os resultados ($R^2 = 1 - \frac{SSE}{SS_{yy}}$). O R^2 ajustado considera a perda de graus de liberdade quando são adicionadas outras variáveis ao modelo (R^2 ajustado = $1 - \left[\frac{n-1}{n-(k+1)}(1 - R^2)\right]$), e tal como, os critérios de informação, ajuda na escolha de um melhor modelo, sempre que tivermos a comparar dois semelhantes.

2.2.3. Análise de Séries Temporais

As séries temporais (ou sucessões cronológicas) são conjuntos de observações em certos instantes do tempo caracterizados por intervalos idênticos. Em termos matemáticos são definidas por uma função que contém uma variável *y* nos tempos t_1, t_2,… ($y = F(t)$). Os movimentos das séries

temporais são classificados em quatros tipos principais, normalmente designados de componentes de uma sucessão cronológica, sendo estes:

1) Movimentos de tendência, que representam o sentido ou direção que a série leva num intervalo de tempo longo. Podemos determinar as retas e curvas destas tendências através da regressão linear.
2) Movimentos ou variações cíclicas, que se referem a oscilações ou desvios da reta. São ciclos, periódicos ou não, que poderão seguir padrões em intervalos de tempo iguais.
3) Movimentos ou variações por estações, ou conventualmente designados por sazonalidade, são padrões idênticos a que uma série obedece durante um período de tempo semelhante, durante meses ou anos. Resultam de eventos que ocorrem anualmente ou mensalmente, como as compras durante o natal ou estadias em hotéis no verão, por exemplo.
4) Movimentos irregulares ou aleatórios. Como o nome indica, são séries que se deslocam esporadicamente, provocadas por eventos casuais. Estes movimentos são normalmente curtos mas intensos e podem acarretar novos movimentos cíclicos ou de outros tipos.

Uma série temporal pode ser composta por uma tendência determinística (componente do movimento geral), uma componente de sazonalidade determinística e a componente irregular (residual). Podemos determinar a série por forma aditiva, ou seja, somando estas três componentes, ou pela forma multiplicativa, multiplicando as componentes.

Para realizar uma análise econométrica para as séries temporais é necessário definir o objetivo principal: desenvolver um modelo que seja capaz de prever, interpretar e testar hipóteses relativas a dados económicos e financeiros. Os modelos utilizados habitualmente são AR(p) e MA(q), ARMA(p,q), ARCH(p) e GARCH(p,q). Para um conhecimento mais vasto, e visto que não irei abordar estes modelos na programação do algoritmo desta dissertação, pode ser consultado o livro escrito por William H. Greene em [5]. No entanto, irei fazer uma brevíssima apresentação de cada um dos modelos.

O modelo AR(p) ou modelo Autoregressivo de ordem *p* tem a seguinte forma geral: $y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + \ldots + a_p y_{t-p} + \varepsilon_t$, em que o *p* designa a ordem de desfasamento. O modelo MA(q) ou modelo Média Móvel de ordem *q* é representado pela equação: $y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$, tendo *q* o mesmo significado de *p*. Por sua vez, o modelo ARMA(p,q), que agrega os modelos AR(p) e MA(q), é descrito pela fórmula $y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + \ldots + a_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$. O modelo ARCH(q) ou modelo Autoregressivo com Heterocedasticidade Condicional representa-se da seguinte forma: $y_t = \beta_1 + \beta_2 x_{2t} + \ldots + \beta_k x_{tk} + u_t$. Por fim, o modelo GARCH(p,q), que é o modelo ARCH generalizado, define-se por: $y_t = \beta_1 + \beta_2 y_{t-1} + \ldots + \beta_k y_{t-k} + u_t$.

2.3. *Machine Learning*

2.3.1. Introdução ao *Machine Learning*

No século XXI, a utilização e o desenvolvimento do *Machine Learning* e das restantes áreas da Inteligência Artificial tem vindo a crescer devido ao relacionamento destas técnicas com modelos de previsão ou classificação de certos ativos. Estes métodos são aplicados em diversas áreas do mercado de trabalho, desde da área financeira empresarial, como ao marketing e, obviamente, aos mercados financeiros.

Nesta dissertação, a utilização do *Machine Learning* focar-se-á no desenvolvimento de um modelo de classificação dos *ratings* da dívida das empresas constituintes do índice S&P500, e a nomenclatura utilizada nestes *ratings* será também o da Standard & Poor's. Este último ponto do capítulo 2 irá ser constituído por uma breve descrição do mundo do *Machine Learning* e dos algoritmos que o englobam.

Existem diversas definições relativas ao que é o *Machine Learning*. Arthur Samuel em [6], diz-nos que o *Machine Learning* é a área de estudo que fornece aos computadores a habilidade para aprenderem sem serem especificamente programados para tal. Tom Mitchell em [7] descreve-o como a resolução de um problema, dizendo ao programa do computador para aprender com a experiência E, devido a uma tarefa T e com uma medida de rendimento P (P de *Performance*), isto é, o rendimento do computador a T, medido por P, melhora com E.

Na programação do modelo que foi desenvolvido nesta tese foram utilizados diversos algoritmos para treinar o modelo, nomeadamente: Regressão Logística com regularização L1 e L2 (ou algoritmo de Lasso e Ridge, respetivamente), *Linear Discriminant Analysis* (LDA), *Gaussian Naive Bayes* (GNB), *Decision Tree, Random Forest, Support Vector Machine e K-Nearest Neighbours*.

2.3.2. Regressão Logística com regularização L1 e L2

A Regressão Logística é um modelo muito utilizado e bastante popular nesta área. É um algoritmo que aprende um modelo a partir das *features*, que são combinações lineares do exemplo que servem de *input* ao algoritmo. Este método é aplicado a modelos de classificação, sendo o *output* uma variável categórica, isto é, calcula-se uma probabilidade cujo valor classifica o *output* como pertencente a uma categoria ou classe, sendo este condicionado por X, ou seja, P(Y|X).

Este modelo é dado pela função logística:

$$P(Y=1) = \frac{1}{1+e^{-f(x)}}, \tag{2.2}$$

sendo o $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_n x + \varepsilon$, a equação que irá ser prevista e classificada no modelo. Esta função logística é uma curva em forma de "S", querendo dizer que se obtém um valor entre 0 e 1. Se a probabilidade for superior a um valor barreira (por exemplo, 0.5) o modelo classifica Y como "sim" ou

1 ou uma certa classe. Caso aconteça o contrário, Y irá ser registado como "não" ou 0 ou outra classe. Este método é estimado pelo uso do modelo de Máxima Verosimilhança.

Na Regressão Logística pode-se recorrer à regularização para tratar problemas de *overfitting*, isto é, o modelo consegue prever muito bem os dados de treino, mas quando se depara com os dados de teste ou outros conjuntos de dados, a previsão é pouco eficiente. Existem diversas razões para haver *overfitting*: o modelo que estamos a criar é demasiado complexo para os dados que temos; considerar demasiadas *features* para um número muito curto de exemplos de treino; os dados terem uma dimensão muito elevada, enquanto que os exemplos de treino têm uma dimensão diminuta. Para resolver este problema há várias formas de o fazer, por exemplo: tentar usar um algoritmo mais simples, não tão complexo; reduzir a dimensão dos exemplos no conjunto de dados; adicionar dados de treino; e por fim, regularizar o modelo. A Regularização é a abordagem mais comum e que dá melhores resultados quando nos deparamos com o problema do *overfitting*.

Andriy Burkov em [8] definiu Regularização como uma abordagem cujo objetivo é forçar o algoritmo de aprendizagem a criar um modelo muito menos complexo, ou seja, o modelo reduz significativamente a sua variância e aumenta ligeiramente o seu enviesamento. Os tipos de regularização mais utilizados são a Regularização L1 e a Regularização L2, testadas no modelo criado nesta tese.

Relativamente à Regularização L1 (também designada por Regularização de Lasso), esta produz um modelo mais disperso, no qual a maioria dos seus parâmetros são iguais a zero. Portanto, a L1 faz uma seleção das *features*, decidindo as que são relevantes para o modelo e eliminando as que não o são. A regularização L1 é usada quando o nosso objetivo é aumentar o quão explicável é o nosso modelo. No que diz respeito à Regularização L2 (também designada por Regularização de Ridge), esta relaciona-se mais com o crescimento da *performance* do modelo para os dados de teste. Estas duas regularizações são frequentemente usadas também noutros algoritmos, em Redes Neuronais, por exemplo.

2.3.3. *Linear Discriminat Analysis* (LDA) e *Gaussian Naive Bayes* (GNB)

O LDA pode ser descrito como um método de classificação supervisionado. Este algoritmo encontra uma combinação linear entre as variáveis, separando-as em classes diferentes. Este método é paramétrico, pois assume que os dados têm distribuição Gaussiana, tendo um bom comportamento em conjuntos de dados que tenham esta distribuição, uma vez que não consegue preservar a complexidade dos dados. Em termos práticos, esta metodologia tenta separar os dados em diversas classes e desenha uma região entre as mesmas. Em simultâneo, facilita a perceção da distribuição das variáveis.

O modelo em análise tem duas abordagens diferentes: *Class-dependent transformation* e *Class-independent transformation.* O objetivo da primeira é maximizar o rácio da variância entre classes, afim de se obter uma separação adequada entre as classes. Por sua vez, a segunda pretende maximizar o rácio entre a variância global e a de cada classe.

O GNB é um método probabilístico que se baseia no Teorema de Bayes[2], assume uma distribuição Gaussiana e considera as varíaveis independentes entre si. O objetivo deste algoritmo é estimar os parâmetros necessários para o modelo de classificação, tendo a vantagem de conseguir trabalhar bem com poucos dados. Esta metodologia pode ser treinada com grande eficiência em aprendizagem supervisionada, mas também em situações complexas, como problemas da vida real. Apesar dos pressupostos simplificarem muito o método, o mesmo apresenta bons resultados em problemas de classificação.

Em termos práticos, o modelo descrito acima classifica primeiramente os dados em diversas classes; e, de seguida, calcula a probabilidade de cada variável, sabendo a que classe esta se encontra. Para o cálculo desta probabilidade é necessário determinar a média e a variância da classe associada à mesma *feature*.

2.3.4. *Decision Tree*

O algortimo de previsão seguinte é o *Decision* Tree, definido como um gráfico acíclico utilizado para tomar decisões. Em cada nodo do gráfico o exemplo vai sendo testado segundo uma *feature* específica e esta será avaliada sobre um critério qualquer. Se o valor passar nesse critério, essa *feature* segue para um nodo seguinte. Se não passar segue para outro e assim sucessivamente, até termos a classe a que o exemplo está associado. Este modelo aprende com os dados que lhe são fornecidos, os dados de treino.

O objetivo deste modelo é prever a que classe o exemplo de teste que estamos a usar se identifica. Primeiro, dá-se ao algoritmo os exemplos já classificados e, de seguida cria-se a "árvore" com os vários nodos, para que o modelo possa prever com melhor exatidão a classe do exemplo. Este é um modelo não-paramétrico, visto que ao longo da árvore de decisão o exemplo vai sofrendo avaliações e vai sendo testado em cada nodo pelas *features* definidas.

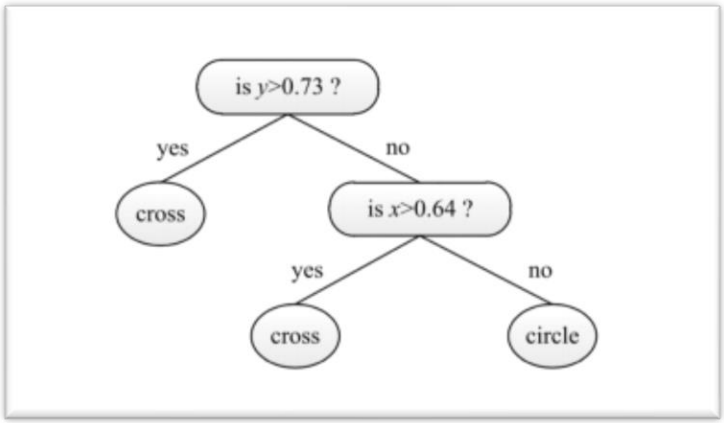


Figura 2.2 – Exemplo de uma *Decision Tree*. Fonte: [9]

[2] O Teorema de Bayes diz-nos que a $P(A|B) = \frac{P(A \cap B)}{P(B)}$, ou seja, a probabilidade de um acontecimento A se realizar conhecendo um B, é ígual a fazer a interseção dos dois e dividir pelo B.

2.3.5. *Random Forest*

A *Random Forest* é um modelo utilizado para classificar ou prever a classe de um exemplo dado, semelhante à *Decision Tree*. Neste sentido, este algoritmo é construído através de diversas árvores de decisão e o resultado que o modelo fornece é a classe que ocorre com maior frequência, ou seja, é a moda entre as classes. Em cada nodo, o exemplo é avaliado por um conjunto de *features* que foram selecionadas aleatoriamente. Esta seleção é feita para que não exista correlação entre as várias *Decision Trees*. Se existirem "árvores" na nossa "floresta" que estão correlacionadas, as mesmas não estarão a ajudar o modelo a prever e a classificar o exemplo.

Atualmente, este algoritmo é um dos mais utilizados, uma vez que recorre a várias amostras do conjunto dos dados originais, resultando na diminuição considerável da variância do modelo. Deste modo, a redução da variância fará com que não ocorra *overfitting*. A *Random Forest* tem uma vantagem significativa em relação aos restantes modelos: a avaliação da importância de cada variável através de uma classificação das *performances* de cada *feature*. Esta vantagem decorre do modelo conseguir estimar o quanto uma variável tem de valor de previsão e quanto acrescenta ao modelo pela medição da *performance* da mesma variável.

2.3.6. *Support Vector Machine* (SVM)

O SVM é um algoritmo de classificação que coloca todas as variáveis num gráfico e traça um hiperplano, ou seja, uma linha/plano (dependendo da dimensão do vetor das variáveis) que separa as *features* numa classificação binária (dando um valor de +1 ou -1, respetivamente *positive label* e *negative label*). O modelo faz esta separação com o fim de maximizar a margem, sendo que esta corresponde à distância entre as observações e o hiperplano (Figura 2.3).

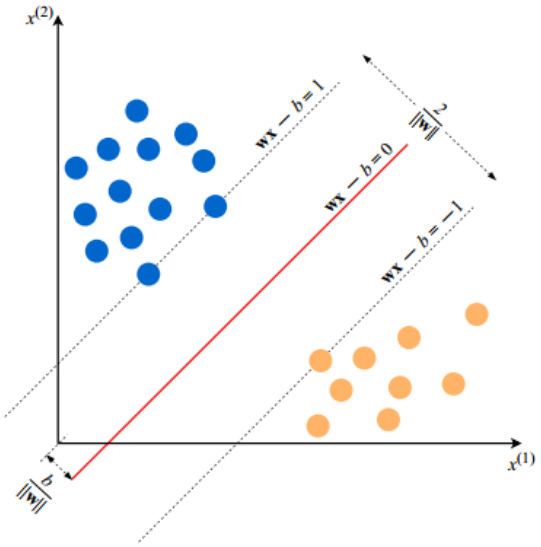


Figura 2.3 – Exemplo de um SVM para um vetor de variáveis com duas dimensões – fonte: [8]

A Figura 2.3 representa a versão linear do SVM, uma vez que o hiperplano é uma linha/plano num gráfico e a decisão resulta dessa linha/plano. É aí que os exemplos são classificados e separados. Existe também uma versão não-linear que incorpora *kernels*, tomando esta decisão arbitrariamente. O maior problema neste algoritmo é que em muitos casos não é possível classificar corretamente as observações em apenas dois grupos, devido ao ruído, a erros de *labeling* ou ainda *outliers*.

Um hiperplano ótimo maximiza a margem, mas a grande questão é como encontrar um hiperplano ótimo. Este determina-se com base nas observações das diferentes classes (*Support Vectors)*, que originam a maior margem. As observações que estão afastadas dos *Support Vectors* (que não resultam numa margem maximizada) não têm influência na determinação do hiperplano.

2.3.7. *K-Nearest Neighbours* (kNN)

Finalmente, o último algoritmo utilizado no modelo criado para esta dissertação é o *K-Nearest Neighbours*. Este modelo é um algoritmo não-paramétrico à semelhança do *Decision Tree* e do *Random Forest*. Ao contrário dos outros modelos, que vão descartando partes dos dados de treino que não servem para o algoritmo, o kNN guarda todos os exemplos dos dados de treino na memória. Deste modo, quando um exemplo não visto pelo modelo aparece, o kNN classifica-o com a *label* mais próxima e com a qual se assemelha mais.

Este modelo é visto como um algoritmo não-paramétrico, pois não especifica pressupostos sobre a forma e a distribuição dos parâmetros. O kNN classifica os exemplos pela proximidade, ou seja, mede a distância entre as observações e aglomera-as. As métricas de distância que são usadas para este modelo podem ser: *Euclidean, Manhattan, Chebyshev e Hamming distance*.

3. Capítulo III
3.1. Análise Empírica

O modelo desenvolvido no âmbito desta dissertação é uma análise de risco de crédito a empresas, avaliando as suas demonstrações financeiras e alguns rácios financeiros. Para treinar este modelo utilizaram-se algoritmos de *Machine Learning* com o intuito final de se classificar empresas quanto ao seu nível de risco, dando-se um *rating* com a nomenclatura da Standard & Poor's. Neste último capítulo está presente uma descrição do modelo produzido e serão especificados todos os passos que foram necessários para a criação do mesmo, nomeadamente, preparação dos dados, criação do modelo e teste dos vários algoritmos descritos no tópico do *Machine Learning*, seleção das variáveis e melhoramento do algoritmo. Todo este processo foi desenvolvido na linguagem de programação Python, mais concretamente usando a aplicação Jupyter Notebook. Foi aqui que todo o código foi criado e testado, sendo dividido em três ficheiros: "I. *Data Preparation* (*equal bin*)", "II. *Model creation and Algorithm Selection*" e "III. *Feature selection, Algorithm improvement*".

No primeiro ficheiro apresenta-se a preparação dos dados, a exploração e a separação dos mesmos. No segundo desenvolve-se o modelo, importando os dados já preparados no primeiro ficheiro. Depois testam-se e analisam-se os algoritmos, comparando a *accuracy* de cada um, com o objetivo de escolher o melhor (com maior *accuracy*). Por fim, no último ficheiro, selecionam-se as *features* no sentido de melhorar a *performance* do modelo, analisando-a posteriormente.

3.1.1. Preparação dos dados

Neste primeiro ponto do capítulo, relativo à Análise Empírica do modelo, descrevem-se primeiramente os dados e todas as operações realizadas para os preparar para o desenvolvimento do modelo, estando esta informação no ficheiro: "I. *Data Preparation* (*equal bin*)". Este trabalho foi desenvolvido utilizando a linguagem de programação Python, na aplicação Jupyter Notebook.

A base de dados que irá ser usada, como já foi referido no ponto relativo ao *Credit Scoring*, é da CRSP (*Center for Research in Security Prices*, LLC), constituída inicialmente por 3320 observações de demonstrações financeiras relativamente a empresas do S&P 500 de 2010 a 2018, e por mais de 500 variáveis para cada observação. Dentro deste conjunto foram escolhidas 20 variáveis, desde do *ticker* da empresa ("tic") até ao inventário ("invt"), entre outras. Com base nestas variáveis, escolhidas e analisadas com base em literatura, foram criadas mais 5 variáveis e 46 rácios financeiros. Como exemplo de uma das variáveis desenvolvidas com base nas que estavam inicialmente na base dados, temos o cash-flow ("cf"), que será igual ao *Net Income* ("ni") mais as depreciações e amortizações ("dp") e os itens extraordinários ("xi"). A título de exemplo de um rácio criado temos o EBITDA/SALES ("ebitdasales") que, como o próprio nome indica, é o rácio entre o EBITDA e as vendas de cada empresa.

Assim, a base de dados é composta por 3320 observações e 72 variáveis, uma vez que foi necessário acrescentar o "spcrf" (*Credit rating* do S&P 500) às 71 já mencionadas no parágrafo anterior.

No modelo criado esta variável corresponde ao resultado de cada observação, visto ser a variável objetivo, isto é, a *feature* que o programa consegue prever e classificar tendo apenas como base estes dados financeiros e respetivos rácios. Esta variável foi obtida através da pesquisa dos *ratings* no site da Moody's e depois convertida para a nomenclatura da Standard & Poor's. Visto que a base de dados representa empresas da S&P500 optei por manter a coerência.

Após a descrição dos dados, passei para a eliminação das observações. Estas foram retiradas por algumas variáveis não terem valores (*missing values*), resultando na diminuição da amostra para 1755 observações. De seguida, averiguei os tipos de variáveis que os dados continham e verifiquei que existiam 69 variáveis númericas (agregam todos rácios e variáveis financeiras), 1 discreta ("spcrf"), 1 categórica ("tic") e 1 temporal ("fyear") que indica o ano da observação.

A etapa seguinte foi observar os *outliers* da variáveis númericas. A conclusão retirada foi que existem muitos *outliers*, uma vez que as empresas têm dimensões completamente diferentes, enviesando o modelo. Para solucionar este problema calculei a *skewness*, isto é, a assimetria de cada variável, com o intuito de saber se será utilizado o *Robust Scaling*, *MinMax Scaling* ou *Normal Standardization*. Se o valor absoluto da *skewness* fosse superior a 2 usaria *Robust Scaling*; entre 1 e 2 utilizaria *MinMax Scaling*; inferior a 1 seria usado *Normal Standardization*. Estes três métodos são aplicados quando a *dataset* tem observações com grande disparidade de magnitudes (neste caso, relativamente ao tamanho da empresa), sendo os valores definidos como *outliers*. Sem estes métodos o modelo daria mais importância, em certos algoritmos, às grandes empresas e menos às pequenas. Deste modo, estas técnicas são importantes para melhorar a *performance* do modelo, que seria enviesada se estas não fossem aplicadas.

No que diz respeito ao *Robust Scaling*, matematicamente, este executa o seguinte rácio:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)} \tag{3.1}$$

ou seja, subtraí cada valor de uma variável pelo primeiro quartil da mesma e depois divide pela diferença entre o terceiro e o primeiro quartil.

Em relação ao *MinMax Scaling*, este é operacionalmente muito simples. O maior número de uma variável em todas as observações fica como 1, o menor como 0 e os restantes valores são obtidos proporcionalmente entre 0 e 1, sendo a sua fórmula a seguinte:

$$\frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{3.2}$$

Por fim, a *Normal Standardization* faz o *scaling* através da média dos valores de cada variável, a diferença entre o valor de cada *feature* e a sua média, e divide pelo desvio padrão da respetiva variável. Matematicamente este modelo é dado por:

$$\frac{x_i - \text{mean}(x)}{\sigma(x)} \tag{3.3}$$

Deste modo, utilizei o *Robust Scaling* para valores mais distantes da média, ou seja, os maiores *outliers*; o *MinMax Scaling* para os *outliers* intermédios; e para os valores mais próximos da média usei a *Normal Standardization*.

Em último lugar, a base de dados foi dividida numa parte de treino (85%, ou seja, 1491 observações) e noutra de teste (15%, isto é, 264), sendo que a variável "spcrf" foi separada das restantes visto ser o "Y" ou *target* do modelo (variável explicada pelas outras *features*). Assim, obtive os ficheiros: "X_SP500_train", "X_SP500_test", "Y_SP500_train" e "Y_SP500_test" que constituem, respetivamente, as variáveis explicativas de treino e teste e as variáveis *target* de treino e teste.

3.1.2. Criação do Modelo e Seleção dos Algoritmos

Neste segundo ponto da Análise Empírica é desenvolvido o modelo com a base de dados já preparada. Depois este será treinado utilizando diversos algoritmos que serão testados, no sentido de encontrar o melhor a ser aplicado no modelo. Na criação do modelo apliquei a técnica SMOTE (*Synthetic Minority Over-sampling Technique)* para lidar com os dados sub-representados, que muitas vezes têm uma importância vital no modelo. Se esta técnica não for utilizada, a *performance* do modelo pode ser imensamente afetada. De seguida, testei os algoritmos descritos no Capítulo II. Finalmente, comparei os resultados e escolhi o melhor algoritmo com o objetivo de o usar no último ficheiro, onde farei a seleção das variáveis e o melhoramento do algoritmo.

O SMOTE é utilizado para contrariar os conjuntos de dados que não estão balanceados, ou seja, que possuem certos dados que estão em classes sub-representadas. O modelo iria dar pouca importância a estas classes, quando estas podem ser observações fulcrais para o modelo. Esta técnica irá produzir novos exemplos semelhantes a estas minorias, criando assim mais observações para que o modelo dê a mesma relevância em relação a outros valores. No caso desta dissertação existiam poucas observações classificadas como *Non-Investment grade* e o SMOTE criou mais valores nesta classe.

Em primeiro lugar, antes da aplicação da técnica SMOTE (utilizada apenas para os dados de treino) existiam 1491 observações, algumas classificadas como *Investment grade* e outras como *Non-Investment grade*. Depois de aplicada a técnica passei a ter 2246 observações, 1123 classificadas como *Investment grade* e 1123 como *Non-Investment grade*.

No passo seguinte testei os algoritmos e observei a *accuracy* que cada um deles obteve. Comecei pela Regressão Logística com regularização L1, e de seguida, testei com regularização L2. Na primeira obtive 81.44% de *accuracy* e na segunda 80.30%, sendo estes valores pouco díspares. Depois, testei os algoritmos *Linear Discriminant Analysis* e o *Gaussian Naive Bayes*, que tiveram como resultados 79.92% e 35.98%, respetivamente. Os algoritmos testados de seguida foram o *Decision Tree* e o *Random Forest* que obtiveram como resultados 87.12% para o primeiro e 95.08% para o segundo. Por fim, verifiquei as *accuracies* dos algoritmos SVM e kNN, que resultaram em valores percentuais de 59.85% e 83.71%, respetivamente.

Concluindo, o melhor algoritmo foi claramente o *Random Forest*. Como foi explicado no ponto 2.3.5., este modelo é um dos mais utilizados para problemas de classificação e habitualmente é o que

agrega melhores resultados para este tipo de modelos. Isto deve-se muito ao facto do próprio algoritmo selecionar as melhores *features*, estatisticamente significativas, ou seja, as variáveis que têm mais relevância para a explicação do modelo. Perante o resultado deste algoritmo pode-se concluir que o modelo tem uma boa *performance*, devido à elevada percentagem de *accuracy* obtida no *Random Forest* (95.08%) e na *Cross Validation* (técnica que avalia a *performance* do modelo), cujo resultado foi 93.77%.

3.1.3. Seleção das variáveis e Melhoramento do Algoritmo

No último ficheiro "*III. Feature Selection, Algorithm improvement*" realizei a seleção das *features* para obter um modelo mais eficiente, com menos variáveis e, se possível, com melhor *performance*. De seguida, tentei explicar as escolhas das variáveis feitas pelo modelo, ajustei os hiperparâmetros do modelo e testei-os para observar as melhorias trazidas à *performance* do modelo. Por fim, analisei os resultados através do AUC, da matriz de confusão e da curva AUC.

No processo da seleção das *features* procedi à realização de um método híbrido, ou seja, testei o modelo e obtive um AUC de 97.4835%. Este resultado foi conseguido através de 69 variáveis numéricas, uma vez que retirei as variáveis "tic" e "fyear" por não serem relevantes para o modelo, pois representam apenas o *ticker* da empresa e o ano da observação.

No passo seguinte observei a importância de cada variável para a explicação do modelo (figura 3.1) e criei um modelo apenas com a *feature* que tinha maior importância – "dvt", dividendos que a empresa distribuiu no ano daquela observação. Este novo modelo teve como resultado um AUC de 70.1421%, representando assim grande parte da explicação do modelo e indicando que os dividendos se relacionam bastante com o facto da empresa ter um *rating* de dívida elevado. Uma vez que as empresas que distribuem muitos dividendos têm de estar em boas condições financeiras para o fazer têm de ter, teoricamente, as suas dívidas controladas, resultando num *rating* da dívida muito favorável.

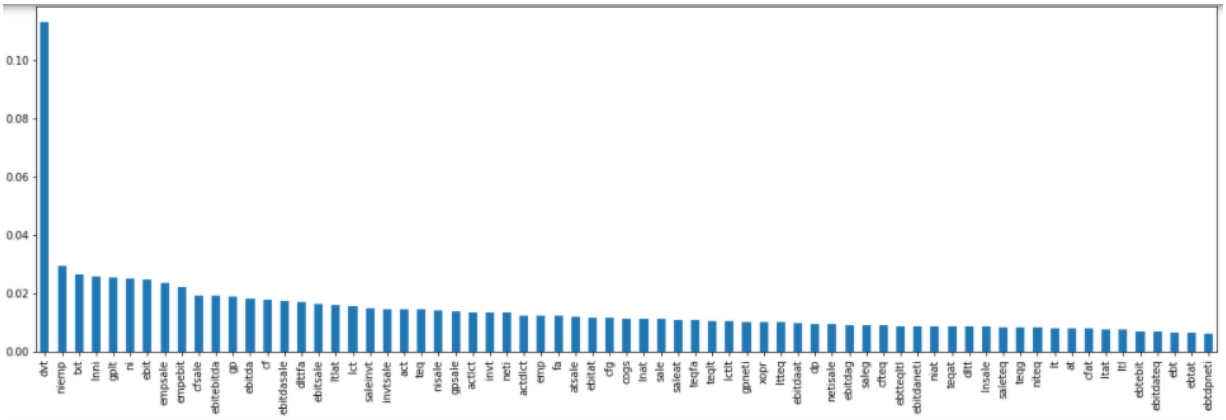


Figura 3.1 – Significância inidividual das 69 variáveis numéricas iniciais do modelo.

Finalmente criei um método que vai acrescentando uma variável de cada vez no novo modelo e verifica qual é o aumento no AUC. Se este aumento for superior 0.1% a variável é mantida no modelo, caso contrário a *feature* será removida.

Features	Metric
CURRENT ASSETS	act
TOTAL ASSETS	at
CASH-FLOW/SALES	cfsale
DIVIDENDS	dvt
EBIT	ebit
EBIT/EBITDA	ebitebitda
EBITDA/TOTAL ASSETS	ebitdaat
EBT/EBIT	ebtebit
EMPLOYEES/SALES	empsale
INVENTORIES	invt
INVENTORIES/SALES	invtsale
GROSS PROFIT/LIABILITIES	gplt
CURRENT LIABILITIES	lct
LONG-TERM LIABILITIES/TOTAL ASSETS	ltlat
LN (NET INCOME)	lnni
NET INCOME/EMPLOYEES	niemp
NET INCOME/EQUITY	niteq
SALES/INVENTORIES	saleinvt
SALES/EQUITY	saleteq
INCOME TAXES	txt

Figura 3.2 – *Features* selecionadas pelo modelo

Este método híbrido selecionou as 20 variáveis indicadas acima, na figura 3.2, sendo que estas fazem parte das 69 variáveis apresentadas inicialmente na figura 3.1. Estas 20 variáveis fizeram aumentar cada uma delas pelo menos 0.1% no AUC. De seguida, irei abordar cada uma das variáveis adicionadas pelo modelo.

No que diz respeito à primeira variável a ser adicionada ao novo modelo, esta designa-se por "niemp" e representa o rácio entre o *net income* e o número de trabalhadores. Este rácio dá-nos o lucro que a empresa está a ter por trabalhador e, quanto maior for este rácio, melhor será para a empresa e o seu *rating*. A mesma fez aumentar o modelo em 2.01%. Outra *feature* acrescentada pelo modelo foi o logaritmo do *net income* ("lnni"). Esta variável fez aumentar o modelo em 0.4%, sendo utilizado o logaritmo e não o valor absoluto do *net income* ("ni", variável removida pelo modelo por não aumentar pelo menos 0.1%). O "ni" foi eliminado uma vez que, sem o logaritmo, as empresas mais pequenas teriam menos *net income*. Contudo, não significa que tenham uma pior situação financeira. Deste modo, o valor absoluto não contempla a disparidade do tamanho das empresas, enquanto que logaritmo aproxima mais os valores e não beneficia tanto as empresas grandes.

Para além das referidas, outras *features* adicionadas nomeiam-se "niteq" e "txt". A primeira representa a divisão entre o *net income* e o capital próprio, contribuindo com um aumento de 0.22%. Este rácio é importante pelo facto de quanto mais a empresa se financiar por Capital Próprio e não por Capital Alheio (Passivo), melhor será o seu *rating* e menor será este rácio. A segunda variável retrata os impostos que a empresa teve que pagar no ano da observação. Esta *feature* fez aumentar consideravelmente o AUC em 8.84%. Este acontecimento pode ser explicado por as empresas terem muitos impostos para pagar, ou seja, significa que tiveram valores elevados de lucro, uma vez que os

impostos são percentagens dos lucros das empresas. Desta forma, quanto maior os impostos, mais lucros a empresa tem e melhor será o *rating* da dívida da mesma companhia.

Outras variáveis incluídas no novo modelo são o "gplt" (rácio entre os rendimentos brutos e o passivo da empresa), o "lct" (representa o valor do passivo corrente) e o "ltlat" (rácio entre o passivo de longo prazo e os ativos). O "gplt" contribuiu para um aumento de 5.83% e quanto maior o rácio, melhor *rating* da dívida, uma vez que um grande valor deste rácio pode ter como contribuição um valor elevado de rendimentos brutos ou um baixo valor do passivo da empresa. Por seu turno, o "lct" fez aumentar o AUC apenas 0.17%. Esta variável é importante para o modelo por representar a dívida de curto prazo. Relativamente ao "ltlat", quanto menor este rácio, melhor será o *rating* da empresa. O rácio pode ser baixo devido à companhia ter pouco passivo de longo prazo ou um valor elevado de ativos, comparando com o valor do passivo de longo prazo. Esta variável fez aumentar o AUC em 1.05%.

No que concerne às próximas variáveis ("act" e "at"), estas são muito semelhantes por uma delas representar os ativos correntes de curto prazo (com duração inferior a 1 ano), consumidos rapidamente, e por outra representar o total dos ativos da empresa. Estas duas *features* são muito relevantes para este modelo e para definir a dificuldade que uma empresa pode ter em pagar as suas dívidas, uma vez que, pela fórmula da Equação Fundamental da Contabilidade (Ativos = Passivo + Capital Próprio), quanto maior forem os ativos, mais capacidade e menos dificuldade a empresa terá de pagar as dívidas que acarreta. Estas variáveis aumentaram o modelo em 0.41% e 0.18%, respetivamente.

As variáveis do conjunto de *features* que se segue são também muito semelhantes, pois ou representam rácios de *earnings* ("ebitebitda", "ebitdaat" e "ebtebit") ou os valores desses rendimentos ("ebit"). O "ebit" retrata os *earnings* antes dos impostos e custos financeiros, tratando-se de uma variável importante para o modelo, pois indica a *performance* que a empresa teve no ano da observação. Neste modelo "ebit" fez crescer o AUC em 0.59%. As restantes variáveis são rácios. O "ebitebitda" é o rácio entre o EBIT e o EBITDA e representa o impacto das amortizações e depreciações nos resultados da empresa. O "ebitdaat" é a fração entre o EBITDA e o total dos ativos, produzindo os resultados consoante os ativos que a empresa apresenta, ou seja, os rendimentos que os ativos adquiridos estão a proporcionar. O "ebtebit" é o rácio entre o EBT sobre o EBIT e indica o peso dos custos financeiros para a empresa. Estes rácios fizeram aumentar o AUC do novo modelo em 1.71%, 0.11% e 0.20%, respetivamente.

As seguintes variáveis estão todas relacionadas com as vendas ou prestação de serviços das empresas: "cfsale", "empsale" e "saleteq. Estas *features* representam três rácios: o primeiro indica a percentagem das vendas que entra como cash-flow para a empresa; o segundo retrata a fração entre o número de empregados e as vendas; e o último especifica a divisão entre as vendas e o Capital Próprio, que nos dá a perceção do rendimento que o investimento dos acionistas está a produzir na empresa. Mais uma vez, estes são rácios que representam a *performance* económica da empresa e têm impacto direto no possível *rating* que será dado à mesma. Estas *features* aumentaram o AUC do novo modelo em 1.22%, 3.44% e 0.15%, correspondentemente.

Por fim, foram selecionadas e adicionadas ao modelo três variáveis que estão relacionadas com os inventários da empresa: "invt", "invtsale" e "saleinvt". A primeira é constituída pelos valores dos inventários; a segunda pelo rácio entre os inventários e as vendas; e a última pelo rácio entre as vendas e os inventários. Os inventários são uma medida importante para a análise da *performance* da empresa pois, se uma empresa tiver muitos inventários pode significar que não os está a conseguir escoar

(depende dos setores de atividade), isto é, não está a conseguir vender os seus produtos. Esta questão influencia diretamente o *rating* de uma empresa, uma vez que se esta não está a conseguir vender o que produz, não consegue pagar as suas dívidas ou então poderá ter mais dificuldades em fazê-lo. Estas *features* fizeram acrescer o AUC, respetivamente, em 0.49%, 0.41% e 0.22%.

Concluindo o processo de seleção das *features*, foram analisados os AUCs obtidos antes e depois deste processo. Como já foi referido anteriormente neste ponto, o AUC antes do processo de seleção apresentava o valor de 97.4835%. Depois de selecionadas as variáveis o valor do AUC passou a ser então 97.5698%. Portanto, apesar de ter retirado 49 variáveis do modelo, acabei por ter uma melhor *performance* do mesmo. Deste modo, o modelo foi melhorado em termos de *accuracy* e tornou-se mais eficiente.

Relativamente à etapa seguinte, esta retrata o ajuste dos hiperparâmetros do modelo. Os hiperparâmetros são os parâmetros que o modelo não consegue estimar a partir do conjunto de dados e serão utilizados depois para estimar os parâmetros do modelo. O processo de ajustar os hiperparâmetros serve para determinar a combinação correta destes, no sentido de maximar a *performance* do modelo. Este ajuste foi feito automaticamente usando uma estratégia designada por *Random Search*, que cria uma rede de possíveis valores para os hiperparâmetros e vai testando a *performance* do modelo a cada combinação. A que tive melhor *performance* é a combinação que permanece.

Em relação ao modelo desta dissertação, a *accuracy* melhorou para 97.5748% depois de ter sido utilizado o método anteriormente mencionado. Na figura 3.3 podemos observar a curva AUC deste mesmo modelo.

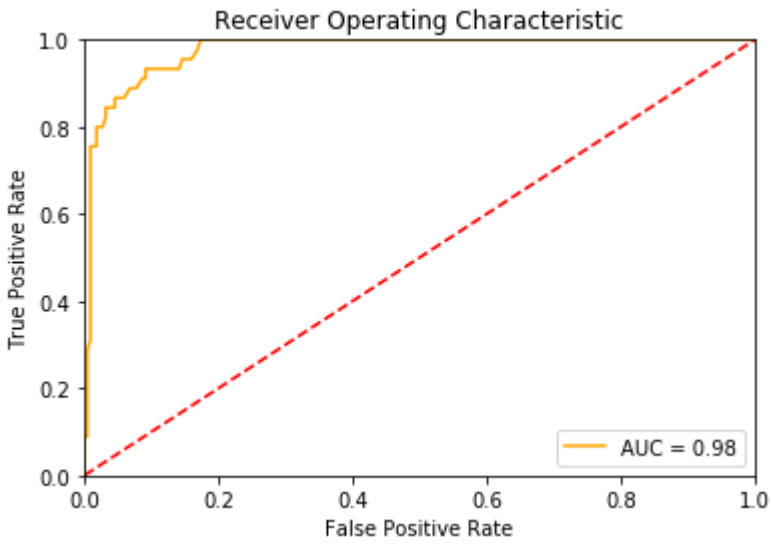


Figura 3.3 – Curva AUC do modelo

Finalmente analisei a Matriz de Confusão, análise esta importante para perceber se o modelo tem sucesso a prever e a classificar a variável *target* do conjunto de teste, classificando as empresas como *Investment grade* ou *Non-Investment grade*. Como é possível observar na Figura 3.4, o modelo apenas errou em 13 exemplos: 2 classificou como *Non-Investment grade* e devia ter associado à classe *Investment grade*; e para as restantes 11 entradas classificou de forma errada, mas em situações contrárias, ou seja, classificou como *Investment grade* e deveria ser *Non-Investment grade*. O modelo

ajustado sobre o conjunto de treino conseguiu acertar na classe em 251 exemplos das 264 observações do conjunto de teste, permitindo concluir que a *accuracy* provada no treino e o ajustamento se mantêm na amostra de teste.

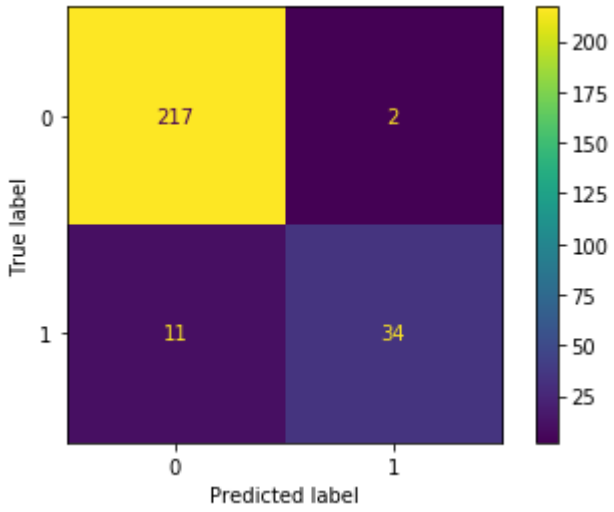


Figura 3.4 – Matriz de Confusão do modelo

Conclusão

O principal objetivo deste trabalho é conseguir classificar empresas quanto ao *rating* da sua dívida, baseando esta classificação nas diversas variáveis que compõem este modelo, sendo estas constituídas por algumas contas das Demonstrações Financeiras e alguns rácios construídos através das mesmas. Para este efeito foram utilizadas Demonstrações Financeiras das empresas presentes no Índice Bolsista S&P500, inerentes ao intervalo temporal de 2010 a 2018. Estes dados foram seguidamente trabalhados no sentido de compreender que observações eram válidas, por exemplo, na procura de *missing values* e outras estratégias para uniformizar os dados, dando relevância a todos os cenários diferenciados deste conjunto de dados (por exemplo, a utilização dos métodos de *Robust Scaling*, *Standard Normalization* e *MinMax Scaling*, que tiveram o intuito de retirar a ordem de grandeza que separa as empresas grandes das mais pequenas presentes neste índice).

De seguida, criou-se o modelo com base em 69 variáveis numéricas que agregam contas das Demonstrações Financeiras das empresas e alguns rácios financeiros. Diversos algoritmos foram testados no sentido de perceber qual é que produzia melhor *accuracy* e melhor se adequava ao desenvolvimento deste modelo. Analisados os resultados, concluiu-se que o melhor algortimo é o *Random Forest* (95.08%). O modelo criado nesta dissertação transitou para a etapa seguinte, selecionando as *features*, no último ficheiro da aplicação Jupyter Notebook, com o propósito de melhorar o mesmo tanto a nível de *performance* (através do incremento da *accuracy*) como a nível da eficiência, reduzindo variáveis. Esta ação resultou na redução das variáveis de 69 para apenas 20 e no aumento do AUC de 97.4835% para 97.5698%, tendo sido feito ainda um ajuste aos hiperparâmetros do modelo que resultou no crescimento do AUC para 97.5748%, o que significa que o modelo criado prevê com sucesso.

Posto isto, constata-se que o objetivo principal desta dissertação não foi totalmente alcançado, uma vez que o modelo teve que ser reduzido a um modelo de classificação binária devido à falta de dados reunidos. Tendo em conta que o conjunto de dados agregava poucos exemplos de empresas com baixos níveis de *rating*, inicialmente, o modelo não conseguia prever e classificar de forma correta este tipo de empresas. Portanto, a solução encontrada foi dividir os *ratings* das empresas em *Investment grade* (desde da classificação AAA até BB+) e *Non-Investment grade* (de BB+ até CC). Esta limitação acabou por influenciar os resultados, pois, se o modelo inicialmente pensado tivesse sido produzido, estes resultados teriam sido bastante diferentes e este trabalho poderia ter sido muito mais interessante, uma vez que seria um trabalho pioneiro nesta área a nível nacional.

Apesar das limitações apresentadas, há que salientar os pontos positivos desta dissertação, nomeadamente, o facto de abordar um tema complexo, utilizando métodos de pré-processamento e algoritmos atuais de *Machine Learning*, abrindo uma linha de investigação recente ao nível nacional, pelas metodologias e linguagem de programação usadas e pela performance dos resultados obtidos.

Bibliografia

[1] P. M. Addo, D. Guegan, and B. Hassani, “Credit risk analysis using machine and deep learning models,” *Risks*, vol. 6, no. 2, pp. 1–20, 2018, doi: 10.3390/risks6020038.

[2] V.-S. Ha and H.-N. Nguyen, “Credit scoring with a feature selection approach based deep learning,” vol. 08003, pp. 1–7, 2016.

[3] M. Hani, A. Mohamed, and T. Elmasry, “Machine Learning approach for Credit Score Analysis: A case study of predicting mortgage loan defaults,” 2019.

[4] J. Fernandes, “Corporate credit risk modeling,” Instituto Superior de Ciências do Trabalho e da Empresa, 2006.

[5] W. Greene, *Econometric Analysis*, Pearson Ed. 2012.

[6] A. L. Samuel, “Some Studies in Machine Learning,” *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, 1959, [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5392560.

[7] T. Mitchell, *Machine Learning*, McGraw-Hil. 1997.

[8] A. Burkov, *The Hundred-Page Machine Learning Book*, vol. 45, no. 13. 2017.

[9] Z.-H. Zhou, *Ensemble Methods, Foundations and Algorithms*, Taylor & F. 2012.

[10] Beaver, William H., 1966, Financial Ratios as Predictors of Failure, *Empirical Research in Accounting: Selected Studies*, 1966, *Supplement, Journal of Accounting Research* 4, 71-111.

[11] Altman, Edward I., 1968, Financial Ratios, Discriminant Analysis and the Prediction of Corporate Bankrupty, Journal of Finance 23, 589-609.

Anexos

In [1]:

```
# to handle datasets
import pandas as pd
import numpy as np

# for plotting
import matplotlib.pyplot as plt
%matplotlib inline

# to divide train and test set
from sklearn.model_selection import train_test_split

# feature scaling
from sklearn.preprocessing import MinMaxScaler

# for tree binarisation
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score


# to build the models
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xgb
# Quando precisar de algum package escrever: "anaconda nome do package install" e ver n
a anaconda cloud

# to evaluate the models
from sklearn.metrics import mean_squared_error
from math import sqrt

pd.pandas.set_option('display.max_columns', None)

import warnings
warnings.filterwarnings('ignore')


#Scaling
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

2. Import and merge datasets

In [2]:

```
variables=['cogs','act','lct','dp','dvt','ebit','ebitda','emp','teq','txt',
           'gp','invt','lt','dltt','ni','xopr','sale','tic','at','fyear','spcrf']
df=pd.read_excel('D:/ISCTE/Mestrado/Tese/python/Pedro/X_SP500.xlsx')[variables]
#quando der erro de utf-8, por df = pd.read_csv('etc.csv', dtype='unicode')[fields]
#df=df.set_index('tic')
```

In [3]:

```
df['cf']=df['ni']+df['dp']
df['ebt']=df['ni']-df['txt']
df['neti']=df['ebt']-df['ebit']
df['fa']=df['at']-df['act']
df['ltl']=df['lt']-df['lct']
df['ltlat']=df['ltl']/df['at']
df['ebtat']=df['ebt']/df['at']
df['ebitdasale']=df['ebitda']/df['sale']
df['actdlct']=df['act']/df['lct']
df['ebitsale']=df['ebit']/df['sale']
df['empsale']=df['emp']/df['sale']
df['ebtdpneti']=(df['ebt']+df['dp'])/df['at']
df['ebtteqltl']=df['ebt']/(df['teq']/df['ltl'])
df['ebitdaneti']=df['ebitda']/df['neti']
df['ebitat']=df['ebit']/df['at']
df['ebitdaat']=df['ebitda']/df['at']
df['ebitdateq']=df['ebitda']/df['teq']
df['ebtebit']=df['ebt']/df['ebit']
df['cfat']=df['cf']/df['at']
df['lctlt']=df['lct']/df['lt']
df['actlct']=df['act']-df['lct']
df['ebitebitda']=df['ebit']/df['ebitda']
df['empebit']=df['emp']/df['ebit']
df['cfsale']=df['cf']/df['sale']
df['niemp']=df['ni']/df['emp']
df['cfteq']=df['cf']/df['teq']
df['dlttfa']=df['dltt']/df['fa']
df['teqfa']=df['teq']/df['fa']
df['teqlt']=df['teq']/df['lt']
df['teqat']=df['teq']/df['at']
df['netisale']=df['neti']/df['sale']
df['gpneti']=df['gp']/df['neti']
df['gplt']=df['gp']/df['lt']
df['gpsale']=df['gp']/df['sale']
df['invtsale']=df['invt']/df['sale']
df['ltteq']=df['lt']/df['teq']
df['ltat']=df['lt']/df['at']
df['lnni']=np.log(df['ni'])
df['lnsale']=np.log(df['sale'])
df['lnat']=np.log(df['at'])
df['niteq']=df['lt']/df['at']
df['niat']=df['ni']/df['at']
df['nisale']=df['ni']/df['sale']
df['saleteq']=df['sale']/df['teq']
df['saleinvt']=df['sale']/df['invt']
df['saleat']=df['sale']/df['at']
df['atsale']=df['at']/df['sale']
df['ebitdag'] = df.groupby(['tic'])['ebitda'].pct_change(periods=1)
df['teqg'] = df.groupby(['tic'])['teq'].pct_change(periods=1)
df['cfg'] = df.groupby(['tic'])['cf'].pct_change(periods=1)
df['saleg'] = df.groupby(['tic'])['sale'].pct_change(periods=1)

df.shape
```

Out[3]:

```
(3320, 72)
```

3. Drop duplicate rows and columns

In [4]:

```
#column list
columns=[]

for var in df.columns:
    columns.append(var)
#drop duplicate rows (using subset, I drop raws where values of columns mentioned matc
h)
df.drop_duplicates(subset=columns, inplace = True)
#replace inf values to missing values
df=df.replace([np.inf,-np.inf],np.nan)
df.shape
```

Out[4]:

```
(3320, 72)
```

In [5]:

```
#features_to_keep = ['dvt','gp','act','lctlt','ebit', 'sale', 'gplt', 'saleinvt', 'em
p','ebitat','ebitebitda', 'ebitsale','lct','spcrf']
#df=df[features_to_keep]

#drop rows that contain missing values
df=df.dropna()
df.shape
```

Out[5]:

```
(1755, 72)
```

There were no duplicate rows

In [6]:

```
# remove duplicate columns
_, i = np.unique(df.columns, return_index=True)
df=df.iloc[:, i]
df.shape
```

Out[6]:

```
(1755, 72)
```

In [7]:

```
df.head()
```

Out[7]:

	act	actdlct	actlct	at	atsale	cf	cfat	cfg	cfsale	cfteq
1	5569.0	3.031573	3732.0	9057.0	1.366888	1265.0	0.139671	0.430995	0.190915	0.293(
2	4629.0	2.445325	2736.0	10536.0	1.536308	1460.0	0.138573	0.154150	0.212890	0.281[
3	4983.0	3.110487	3381.0	10686.0	1.575641	1104.0	0.103313	-0.243836	0.162784	0.208[
4	5500.0	3.231492	3798.0	10831.0	1.551497	895.0	0.082633	-0.189312	0.128205	0.168[
5	3686.0	3.776639	2710.0	7479.0	1.852155	655.0	0.087579	-0.268156	0.162209	0.157(

There were no duplicate columns

4. Types of Variables

In [8]:

```
# let's inspect the type of variables in pandas
df.dtypes
```

Out[8]:

```
act           float64
actdlct       float64
actlct        float64
at            float64
atsale        float64
cf            float64
cfat          float64
cfg           float64
cfsale        float64
cfteq         float64
cogs          float64
dltt          float64
dlttfa        float64
dp            float64
dvt           float64
ebit          float64
ebitat        float64
ebitda        float64
ebitdaat      float64
ebitdag       float64
ebitdaneti    float64
ebitdasale    float64
ebitdateq     float64
ebitebitda    float64
ebitsale      float64
ebt           float64
ebtat         float64
ebtdpneti     float64
ebtebit       float64
ebtteqltl     float64
               ...
lctlt         float64
lnat          float64
lnni          float64
lnsale        float64
lt            float64
ltat          float64
ltl           float64
ltlat         float64
ltteq         float64
neti          float64
netisale      float64
ni            float64
niat          float64
niemp         float64
nisale        float64
niteq         float64
sale          float64
saleat        float64
saleg         float64
saleinvt      float64
saleteq       float64
spcrf           int64
teq           float64
teqat         float64
teqfa         float64
teqg          float64
teqlt         float64
tic            object
```

```
txt           float64
xopr          float64
Length: 72, dtype: object
```

There are a mixture of numerical and categorical variables. Normally object type determines categorical.

4.1 Find categorical variables

In [9]:

```
# find categorical variables
categorical = [var for var in df.columns if df[var].dtype=='O']
print('There are {} categorical variables'.format(len(categorical)))
categorical
```

```
There are 1 categorical variables
```

Out[9]:

```
['tic']
```

4.2 Find temporal variables

In [10]:

```
# make a list of the numerical variables first
numerical = [var for var in df.columns if df[var].dtype!='O']

# list of variables that contain year information (IT'S MISSING 'date', BECAUSE OF ebit
dateq)
temporal = [var for var in numerical + categorical if 'yr' in var or 'year' in var or
'day'in var or 'month'in var or 'time' in var]

print('There are {} temporal variables'.format(len(temporal)))
temporal
```

```
There are 1 temporal variables
```

Out[10]:

```
['fyear']
```

4.3 Find Discrete Varibles

In [11]:

```
# let's visualise the values of the discrete variables
discrete = []

for var in numerical:
    if len(df[var].unique())<20 and var not in temporal:
        print(var, ' values: ', df[var].unique())
        discrete.append(var)
print()
print('There are {} discrete variables'.format(len(discrete)))
discrete
```

```
spcrf  values:  [0 1]

There are 1 discrete variables
```

Out[11]:

```
['spcrf']
```

In [12]:

```
### Find Continuous
# let's remember to skip the Id variable and the target variable SalePrice, which are b
oth also numerical

numerical = [var for var in numerical if var not in discrete and var not in ['Id', 'Sal
ePrice'] and var not in temporal]
print('There are {} numerical and continuous variables'.format(len(numerical)))
numerical
```

```
There are 69 numerical and continuous variables
```

Out[12]:

```
['act',
 'actdlct',
 'actlct',
 'at',
 'atsale',
 'cf',
 'cfat',
 'cfg',
 'cfsale',
 'cfteq',
 'cogs',
 'dltt',
 'dlttfa',
 'dp',
 'dvt',
 'ebit',
 'ebitat',
 'ebitda',
 'ebitdaat',
 'ebitdag',
 'ebitdaneti',
 'ebitdasale',
 'ebitdateq',
 'ebitebitda',
 'ebitsale',
 'ebt',
 'ebtat',
 'ebtdpneti',
 'ebtebit',
 'ebtteqltl',
 'emp',
 'empebit',
 'empsale',
 'fa',
 'gp',
 'gplt',
 'gpneti',
 'gpsale',
 'invt',
 'invtsale',
 'lct',
 'lctlt',
 'lnat',
 'lnni',
 'lnsale',
 'lt',
 'ltat',
 'ltl',
 'ltlat',
 'ltteq',
 'neti',
 'netisale',
 'ni',
 'niat',
 'niemp',
 'nisale',
 'niteq',
 'sale',
 'saleat',
```

```
 'saleg',
 'saleinvt',
 'saleteq',
 'teq',
 'teqat',
 'teqfa',
 'teqg',
 'teqlt',
 'txt',
 'xopr']
```

In [13]:

```
print('So there are:','\n','\n','-',len(discrete),'discrete variable','\n'
      ,'-',len(categorical),'categorical variables','\n'
      ,'-',len(temporal),'temporal variables','\n','-',len(numerical),'numerical variab
les')
```

```
So there are:

 - 1 discrete variable
 - 1 categorical variables
 - 1 temporal variables
 - 69 numerical variables
```

5.Types of problems within variables

5.1 Missing values

In [14]:

```
# let's now determine how many variables we have with missing information

vars_with_na = [var for var in df.columns if df[var].isnull().sum()>0]
print('Total variables that contain missing information: ', len(vars_with_na))
```

```
Total variables that contain missing information:  0
```

In [15]:

```
# let's inspect how many variables have above 80% of values missing information
missingvalues = []
for var in df.columns:
    if df[var].isnull().mean()>0.80:
        missingvalues.append(var)

print('There are',len(missingvalues),'variables with above 80% of values missing inform
ation.')
```

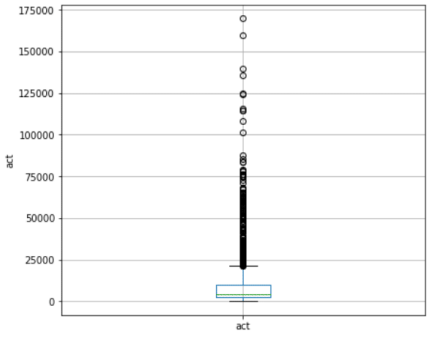
```
There are 0 variables with above 80% of values missing information.
```

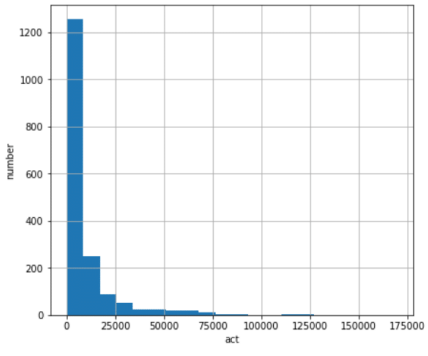
In [16]:

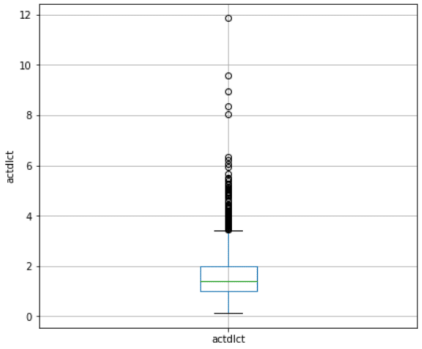
```
# let's visualise the percentage of missing values for each variable
for var in df.columns:
    if df[var].isnull().sum()>0:
        print(var, df[var].isnull().mean())
```

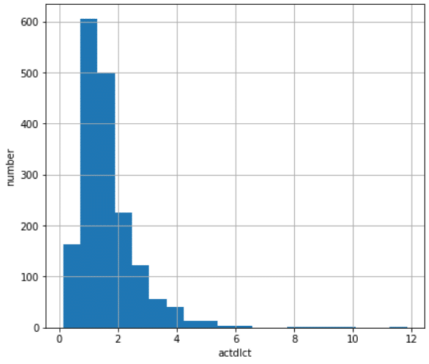
In [18]:

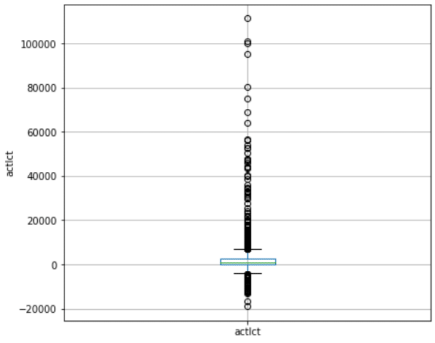
```
# let's make boxplots to visualise outliers in the continuous variables
# and histograms to get an idea of the distribution

for var in numerical:
    plt.figure(figsize=(15,6))
    plt.subplot(1, 2, 1)
    fig = df.boxplot(column=var)
    fig.set_title('')
    fig.set_ylabel(var)

    plt.subplot(1, 2, 2)
    fig = df[var].hist(bins=20)
    fig.set_ylabel('number')
    fig.set_xlabel(var)

    plt.show()
```

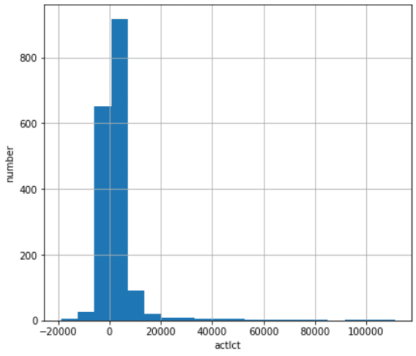

act
175000
150000
125000
100000
75000
50000
25000
0
act

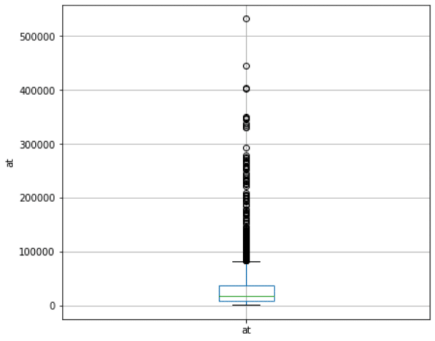

number
1200
1000
800
600
400
200
0
0
25000
50000
75000
100000
125000
150000
175000
act

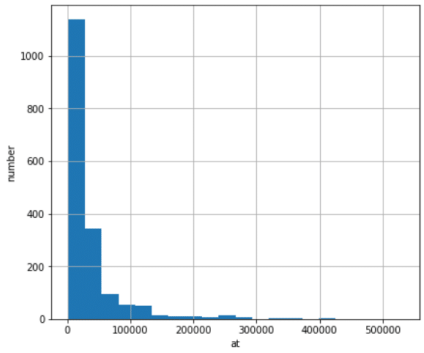

actdlct
12
10
8
6
4
2
0
actdlct

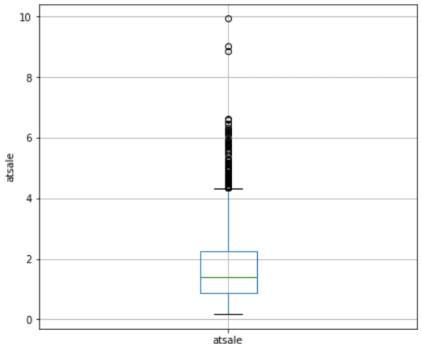

number
600
500
400
300
200
100
0
0
2
4
6
8
10
12
actdlct

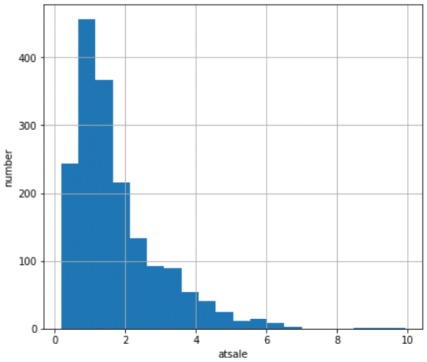

actlct
100000
80000
60000
40000
20000
0
−20000
actlct

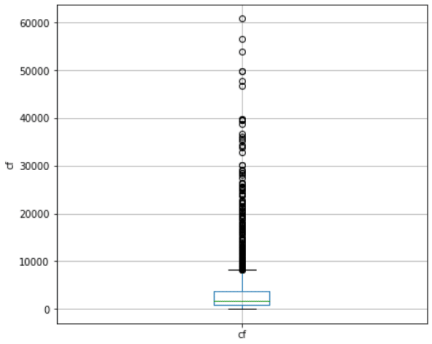

number
800
600
400
200
0
−20000
0
20000
40000
60000
80000
100000
actlct

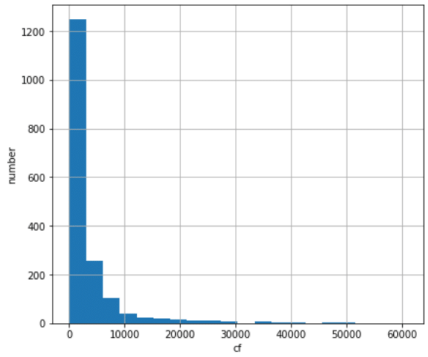

500000
400000
300000
200000
100000
0
at
at

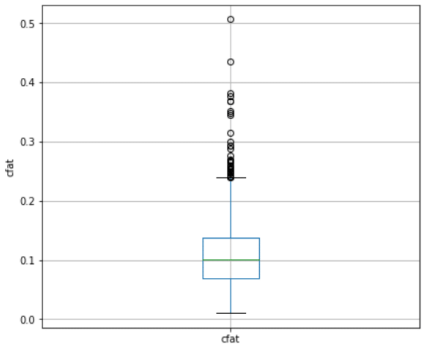

1000
800
600
400
200
0
number
0
100000
200000
300000
400000
500000
at

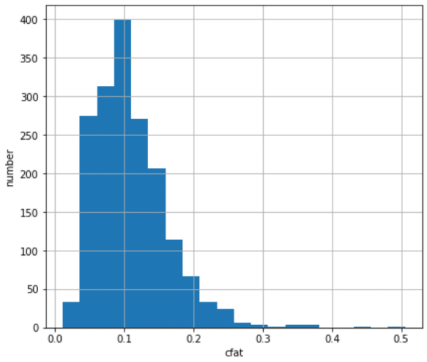

10
8
6
4
2
0
atsale
atsale

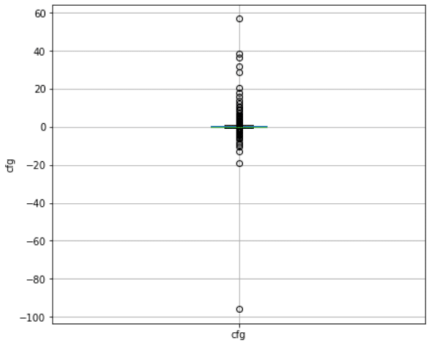

400
300
200
100
0
number
0
2
4
6
8
10
atsale

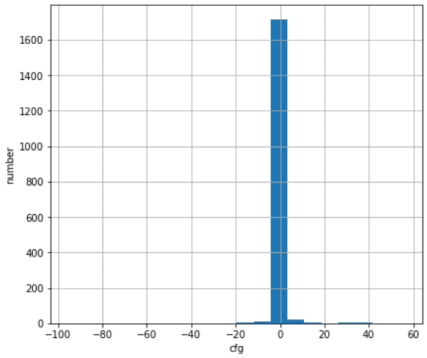

60000
50000
40000
30000
20000
10000
0
cf
cf

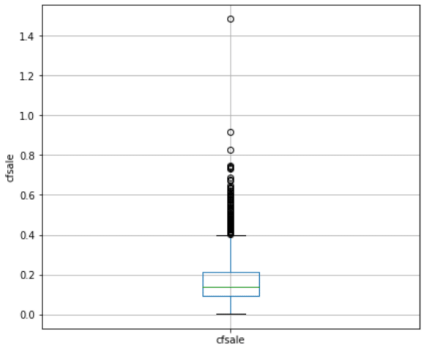

1200
1000
800
600
400
200
0
number
0
10000
20000
30000
40000
50000
60000
cf

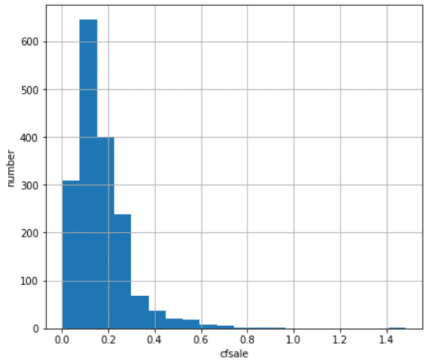

cfat

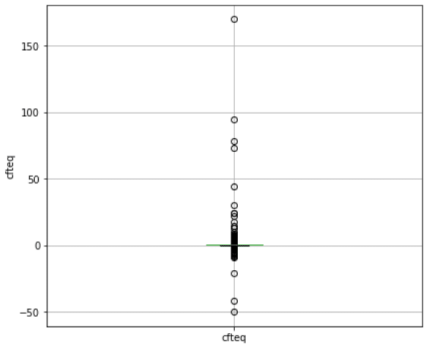

number
cfat

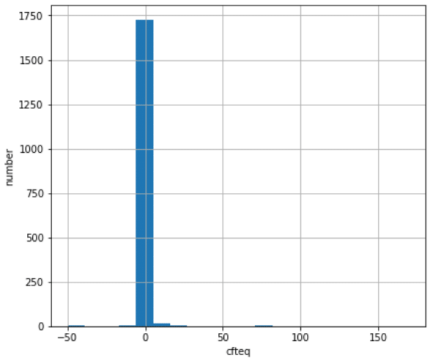

cfg

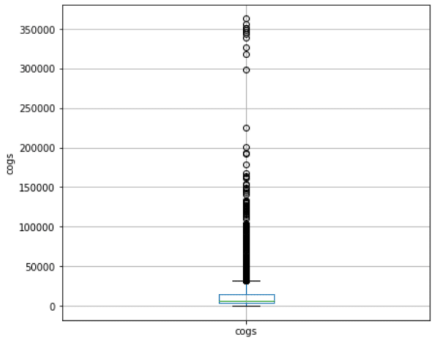

number
cfg

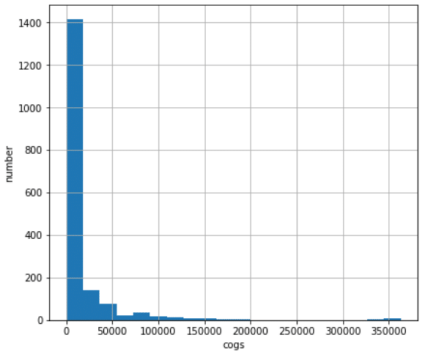

cfsale

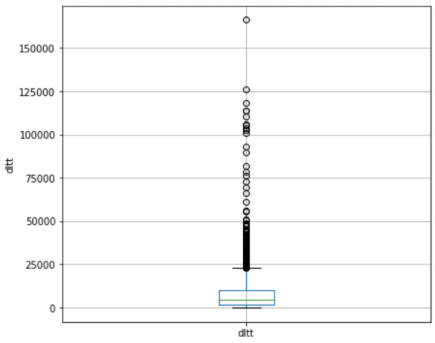

number
cfsale

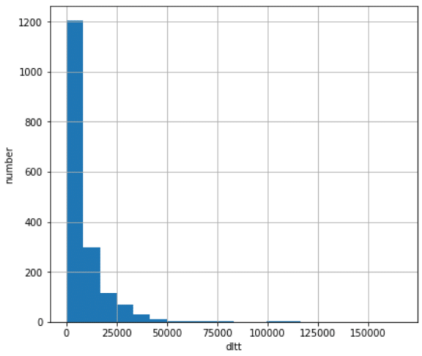

cfteq
150
100
50
0
−50
cfteq

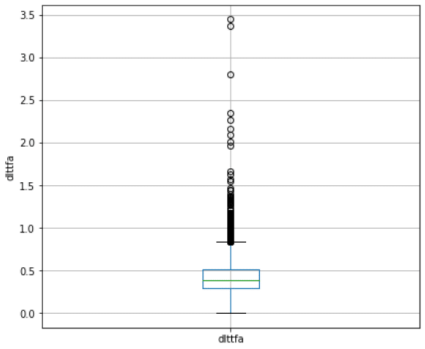

number
1750
1500
1250
1000
750
500
250
0
−50
0
50
100
150
cfteq

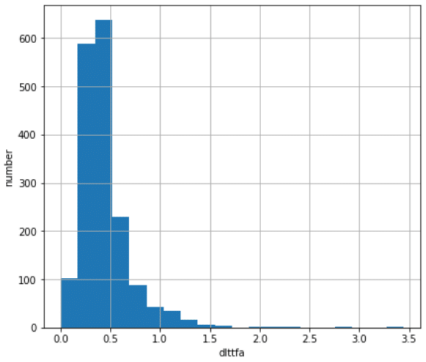

cogs
350000
300000
250000
200000
150000
100000
50000
0
cogs

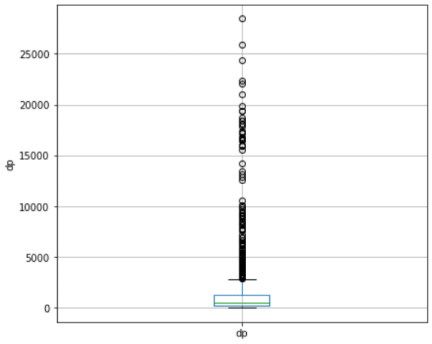

number
1400
1200
1000
800
600
400
200
0
0
50000
100000
150000
200000
250000
300000
350000
cogs

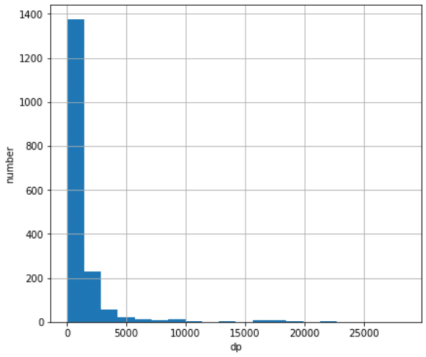

dltt
150000
125000
100000
75000
50000
25000
0
dltt

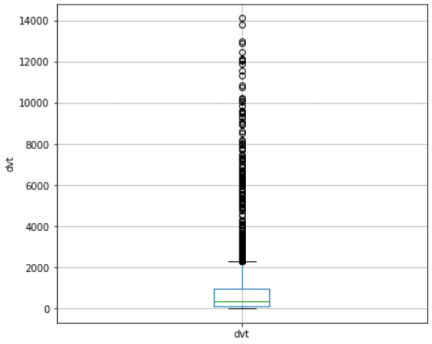

number
1200
1000
800
600
400
200
0
0
25000
50000
75000
100000
125000
150000
dltt

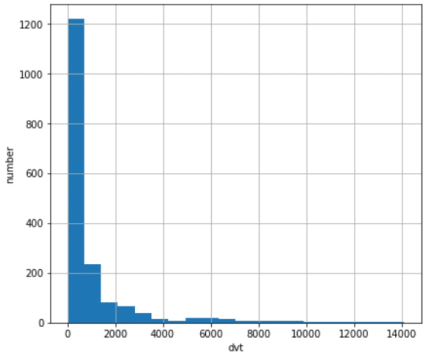

dlttfa

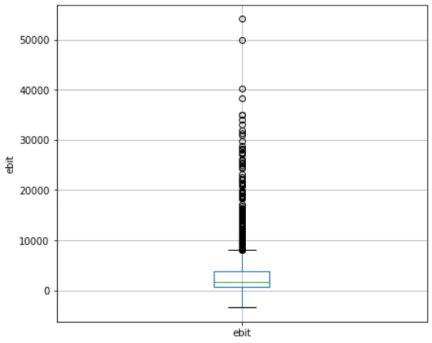

number
dlttfa

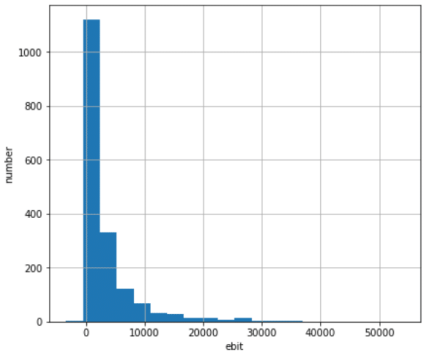

dp

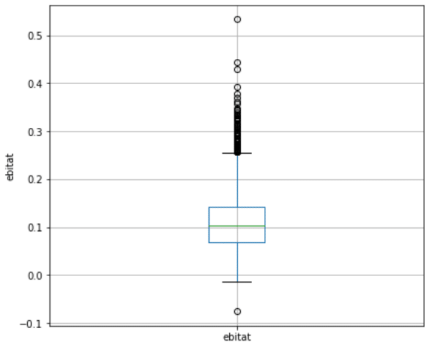

number
dp

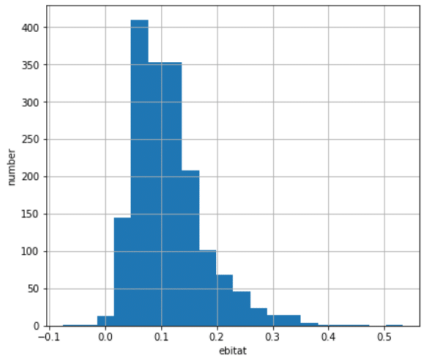

dvt

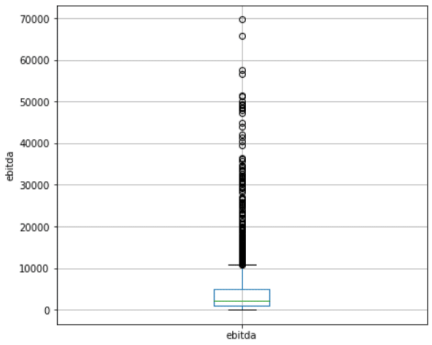

number
dvt

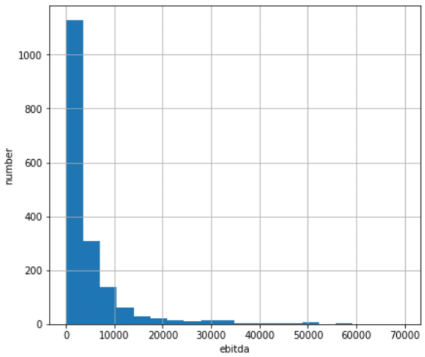

ebit
50000
40000
30000
20000
10000
0

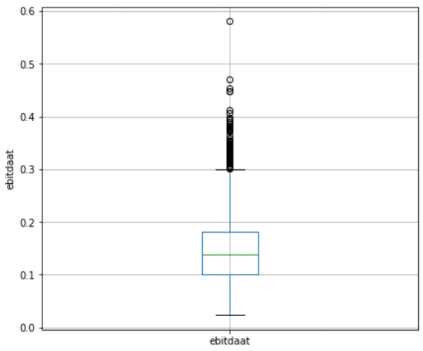

number
ebit
1000
800
600
400
200
0
0
10000
20000
30000
40000
50000

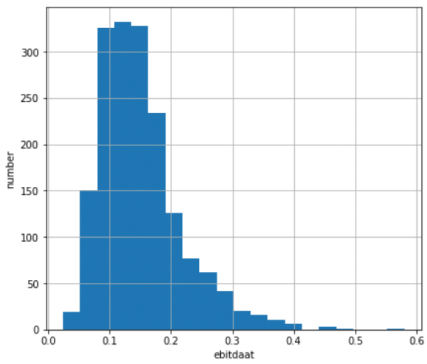

ebitat
0.5
0.4
0.3
0.2
0.1
0.0
−0.1

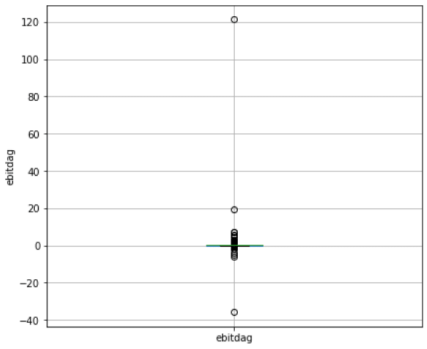

number
ebitat
400
350
300
250
200
150
100
50
0
−0.1
0.0
0.1
0.2
0.3
0.4
0.5

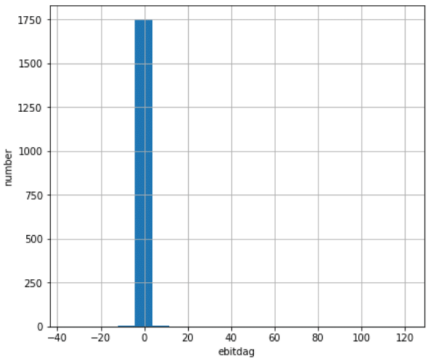

ebitda
70000
60000
50000
40000
30000
20000
10000
0

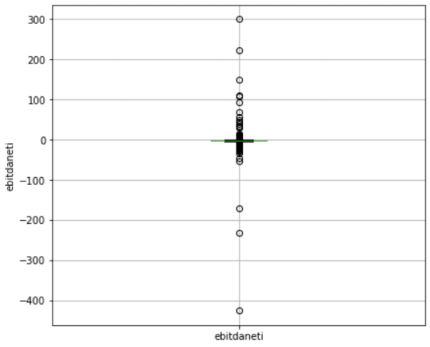

number
ebitda
1000
800
600
400
200
0
0
10000
20000
30000
40000
50000
60000
70000

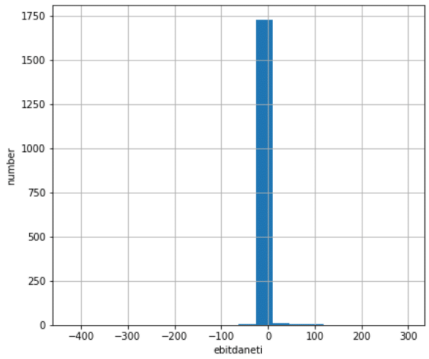

ebitdaat
0.0
0.1
0.2
0.3
0.4
0.5
0.6

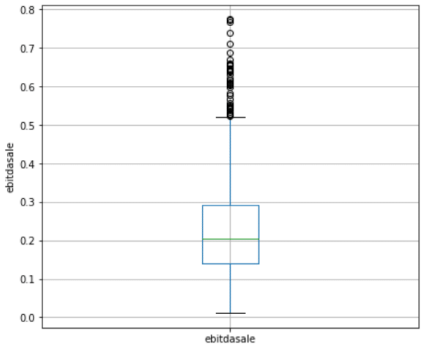

number
ebitdaat
0
50
100
150
200
250
300
0.0
0.1
0.2
0.3
0.4
0.5
0.6

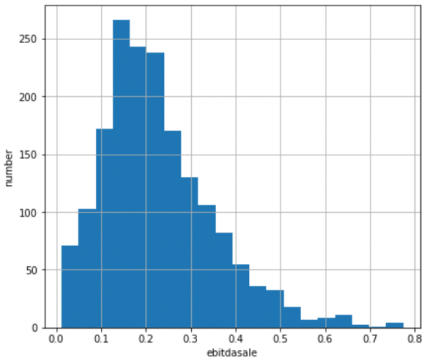

ebitdag
−40
−20
0
20
40
60
80
100
120

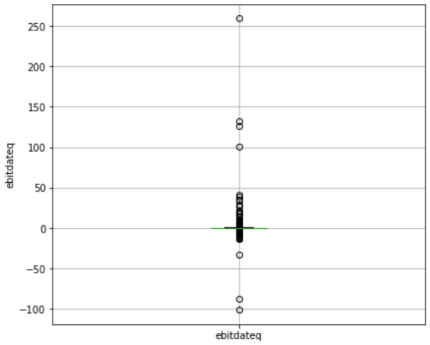

number
ebitdag
0
250
500
750
1000
1250
1500
1750
−40
−20
0
20
40
60
80
100
120

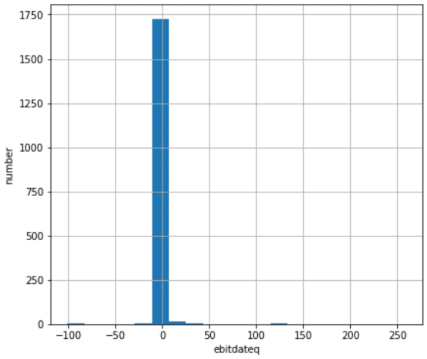

ebitdaneti
−400
−300
−200
−100
0
100
200
300

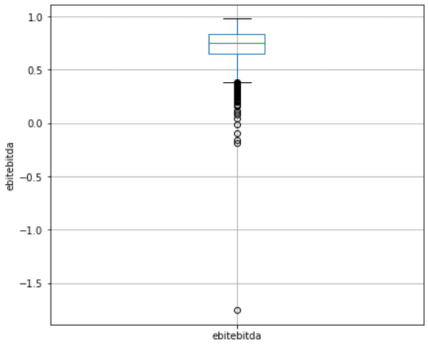

number
ebitdaneti
0
250
500
750
1000
1250
1500
1750
−400
−300
−200
−100
0
100
200
300

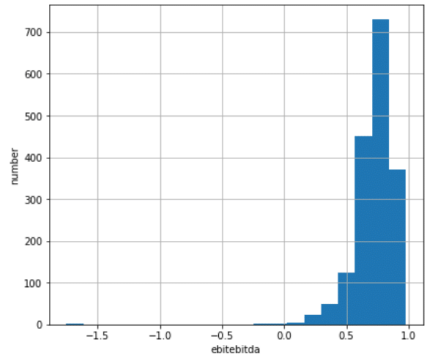

ebitdasale
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

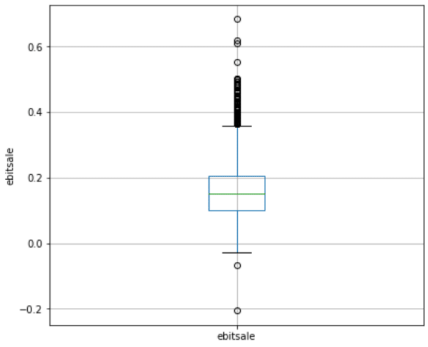

number
ebitdasale
0
50
100
150
200
250
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

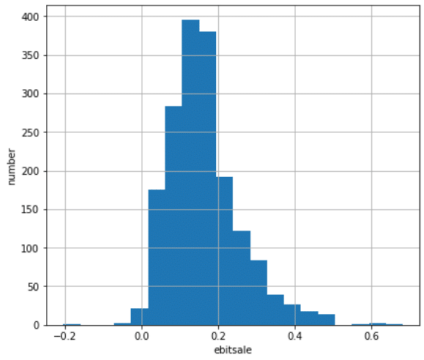

ebitdateq
−100
−50
0
50
100
150
200
250

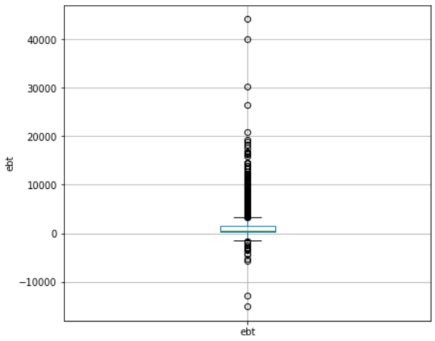

number
ebitdateq
0
250
500
750
1000
1250
1500
1750
−100
−50
0
50
100
150
200
250

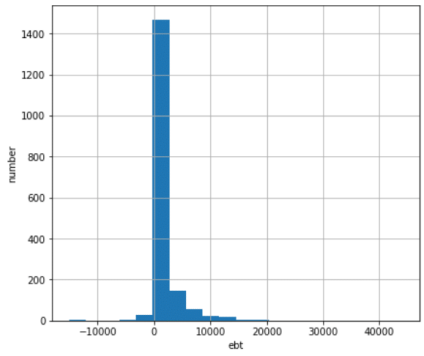

ebitebitda
−1.5
−1.0
−0.5
0.0
0.5
1.0

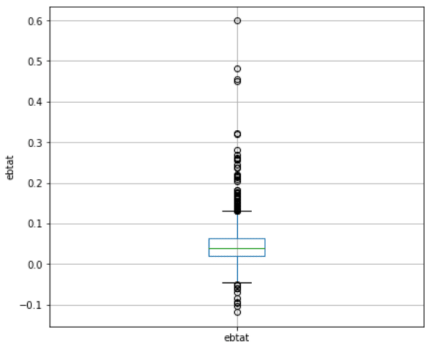

number
ebitebitda
0
100
200
300
400
500
600
700
−1.5
−1.0
−0.5
0.0
0.5
1.0

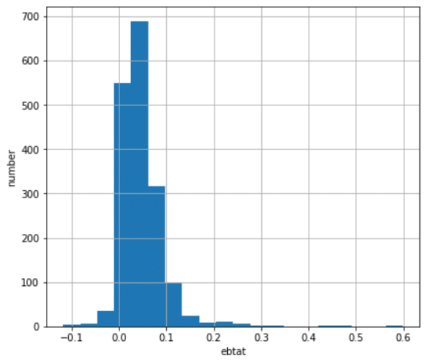

ebitsale
0.6
0.4
0.2
0.0
−0.2
ebitsale

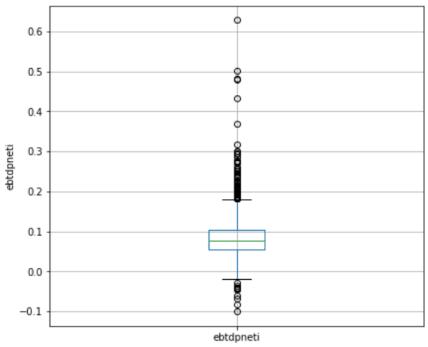

number
400
350
300
250
200
150
100
50
0
−0.2
0.0
0.2
0.4
0.6
ebitsale

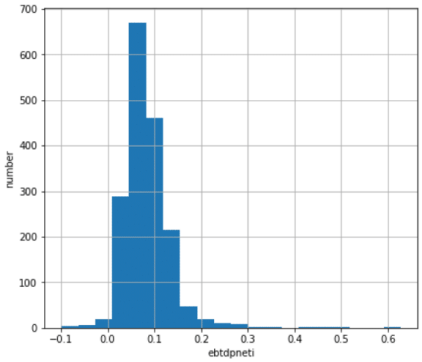

ebt
40000
30000
20000
10000
0
−10000
ebt

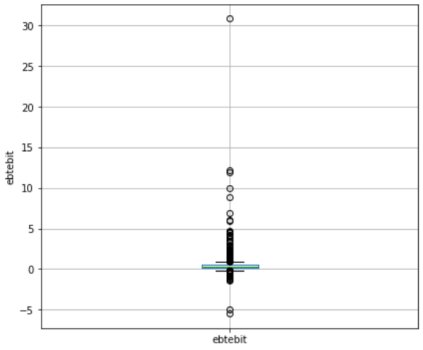

number
1400
1200
1000
800
600
400
200
0
−10000
0
10000
20000
30000
40000
ebt

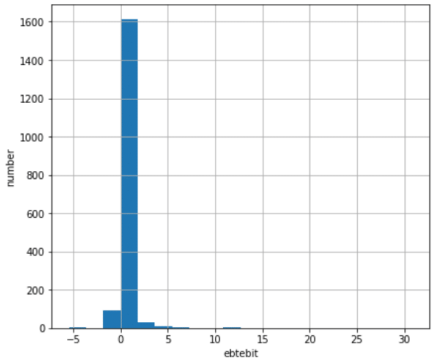

ebtat
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
ebtat

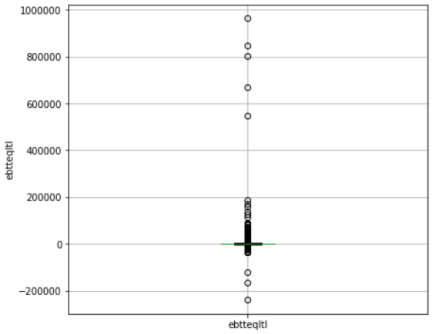

number
700
600
500
400
300
200
100
0
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
ebtat

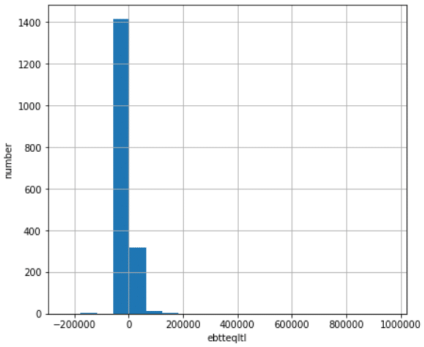

ebtdpneti
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
ebtdpneti

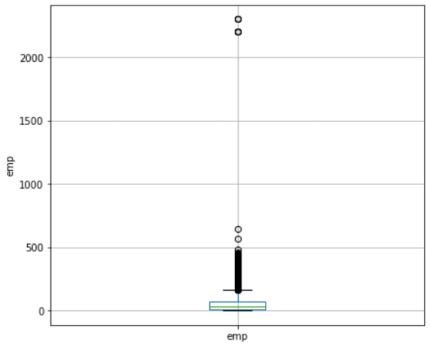

number
700
600
500
400
300
200
100
0
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
ebtdpneti

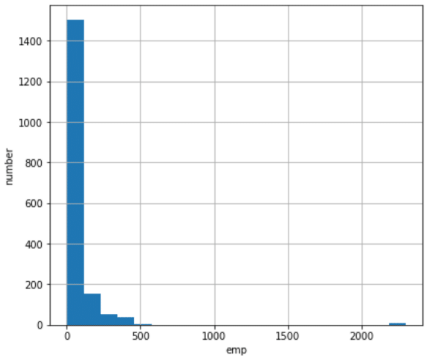

ebtebit
30
25
20
15
10
5
0
−5
ebtebit

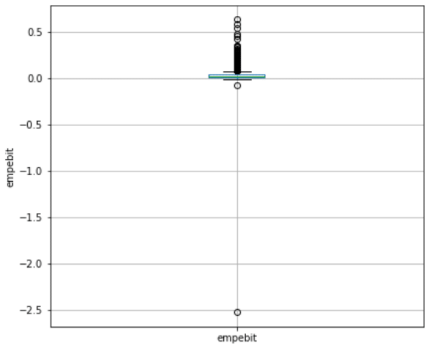

number
1600
1400
1200
1000
800
600
400
200
0
−5
0
5
10
15
20
25
30
ebtebit

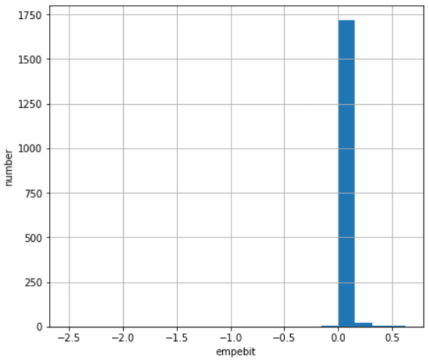

ebtteqltl
1000000
800000
600000
400000
200000
0
−200000
ebtteqltl

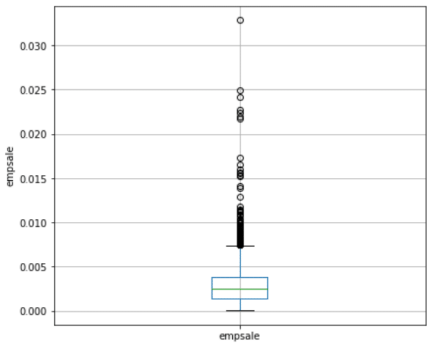

number
1400
1200
1000
800
600
400
200
0
−200000
0
200000
400000
600000
800000
1000000
ebtteqltl

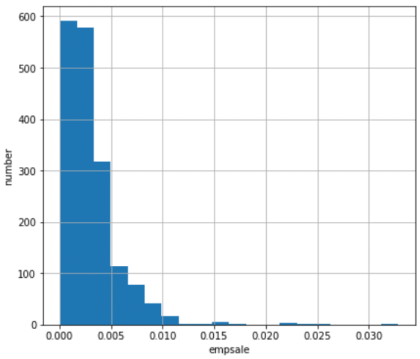

emp
2000
1500
1000
500
0
emp

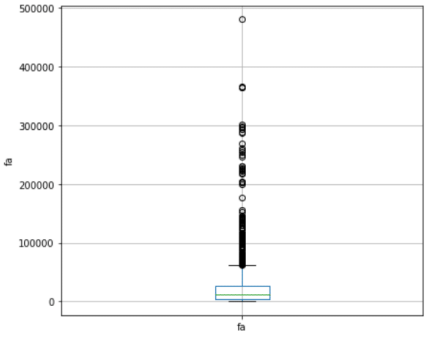

number
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
emp

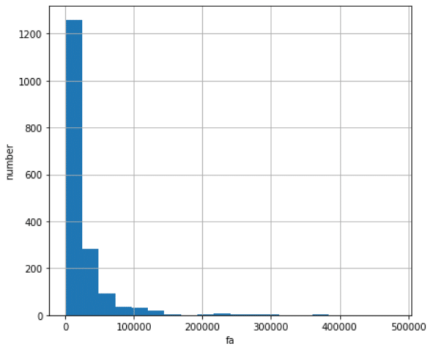

empebit
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
empebit

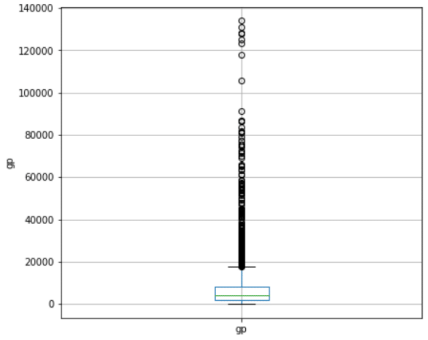

number
1750
1500
1250
1000
750
500
250
0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
empebit

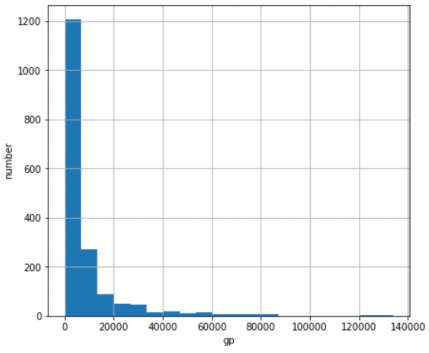

empsale
0.030
0.025
0.020
0.015
0.010
0.005
0.000
empsale

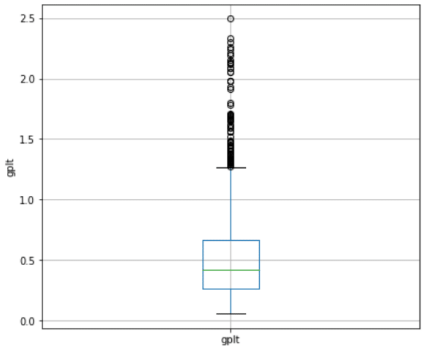

number
600
500
400
300
200
100
0
0.000
0.005
0.010
0.015
0.020
0.025
0.030
empsale

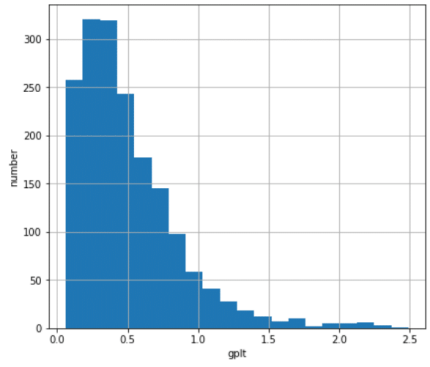

fa
fa

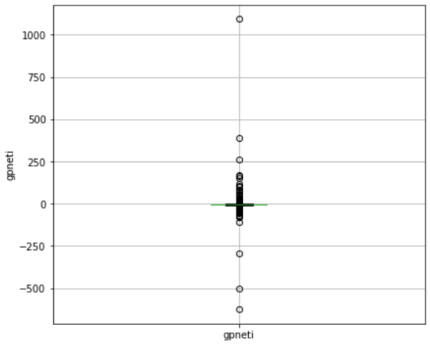

number
fa

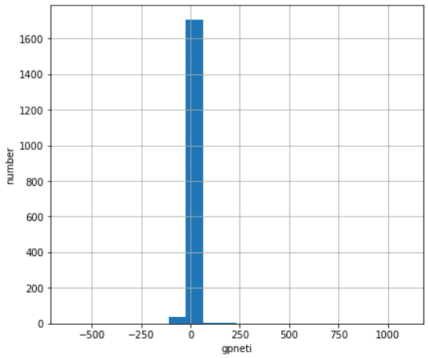

gp
gp

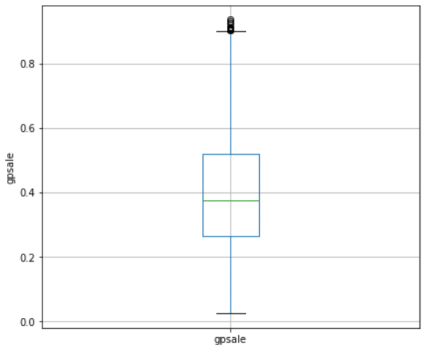

number
gp

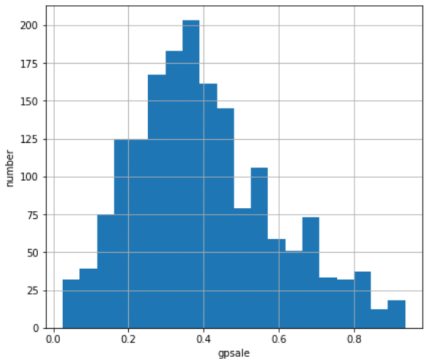

gplt
gplt

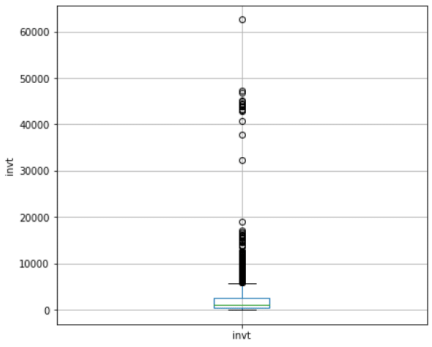

number
gplt

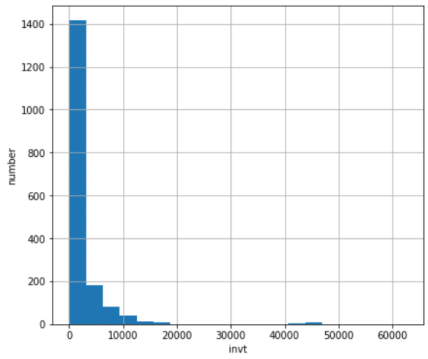

gpneti
1000
750
500
250
0
−250
−500
gpneti

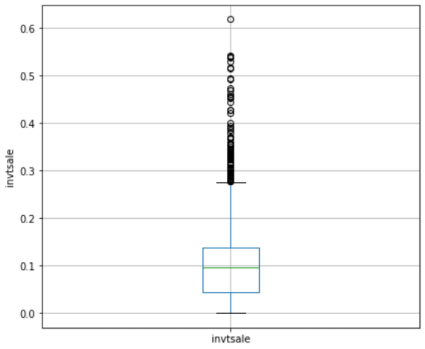

number
1600
1400
1200
1000
800
600
400
200
0
−500
−250
0
250
500
750
1000
gpneti

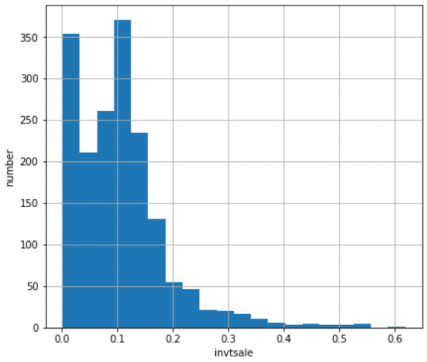

gpsale
0.8
0.6
0.4
0.2
0.0
gpsale

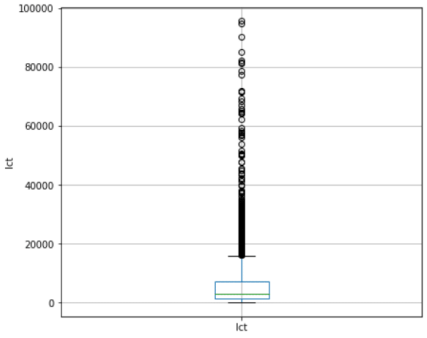

number
200
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
gpsale

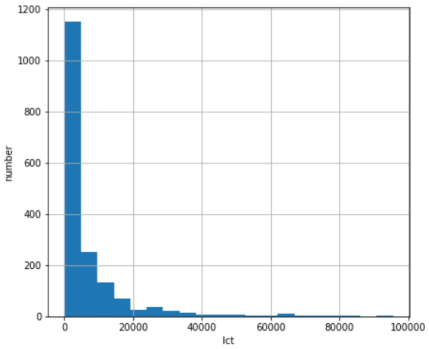

invt
60000
50000
40000
30000
20000
10000
0
invt

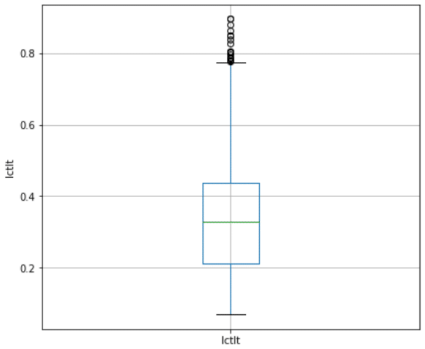

number
1400
1200
1000
800
600
400
200
0
0
10000
20000
30000
40000
50000
60000
invt

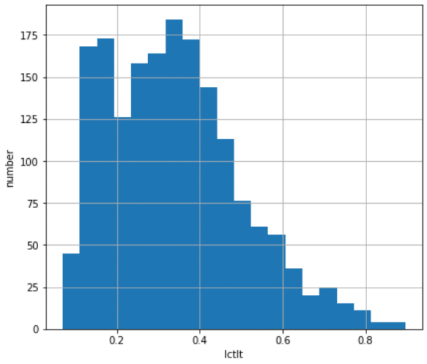

invtsale
0.0
0.1
0.2
0.3
0.4
0.5
0.6

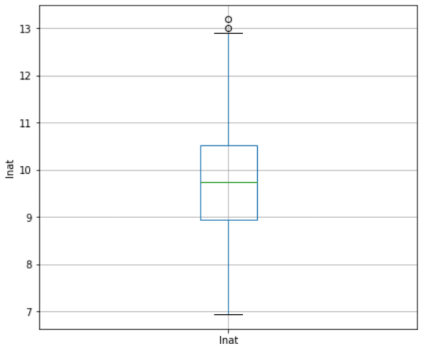

number
invtsale
0
50
100
150
200
250
300
350
0.0
0.1
0.2
0.3
0.4
0.5
0.6

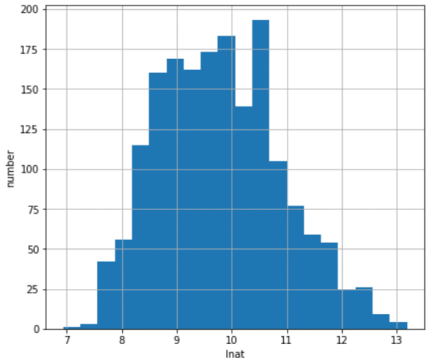

lct
0
20000
40000
60000
80000
100000

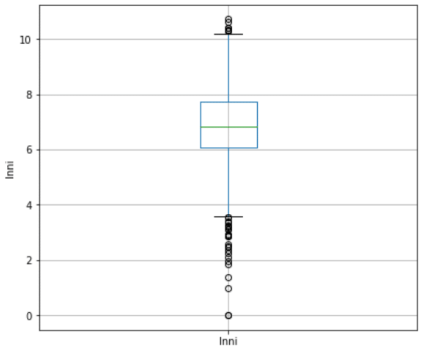

number
lct
0
200
400
600
800
1000
1200
0
20000
40000
60000
80000
100000

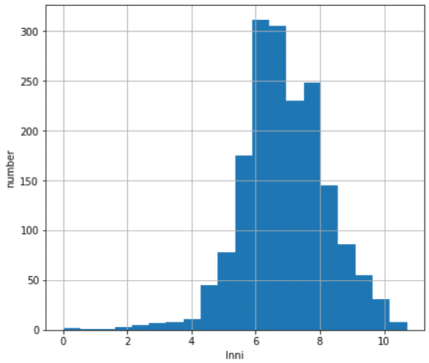

lctlt
0.2
0.4
0.6
0.8

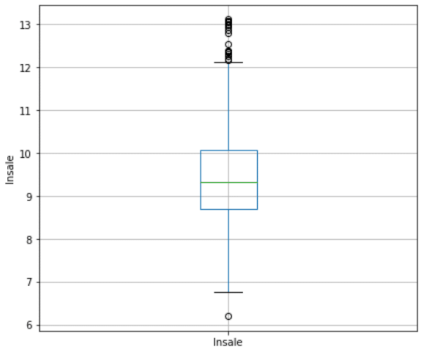

number
lctlt
0
25
50
75
100
125
150
175
0.2
0.4
0.6
0.8

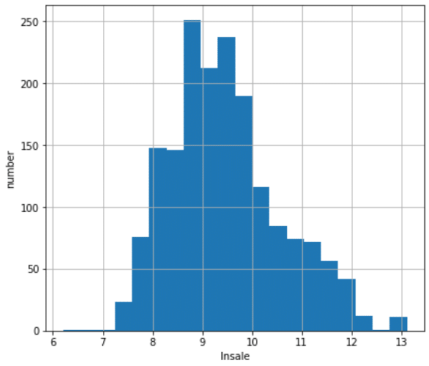

Inat
13
12
11
10
9
8
7
Inat

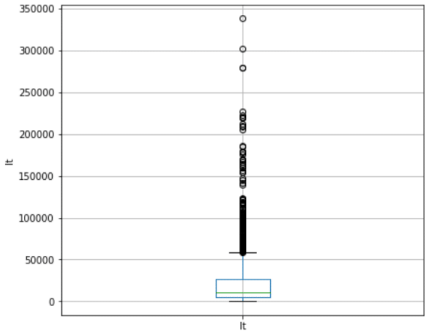

number
200
175
150
125
100
75
50
25
0
7
8
9
10
11
12
13
Inat

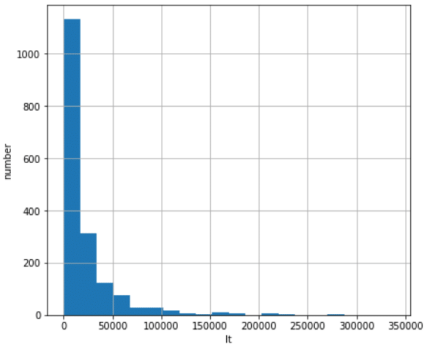

Inni
10
8
6
4
2
0
Inni

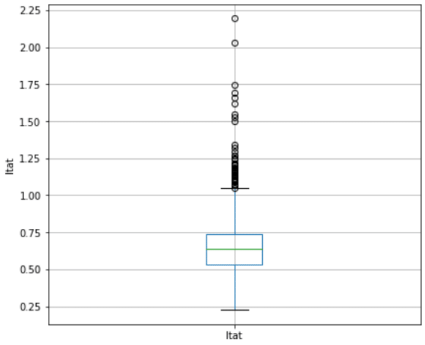

number
300
250
200
150
100
50
0
0
2
4
6
8
10
Inni

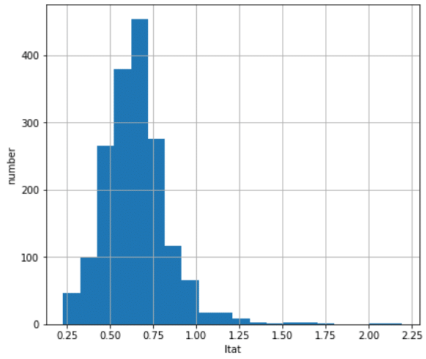

Insale
13
12
11
10
9
8
7
6
Insale

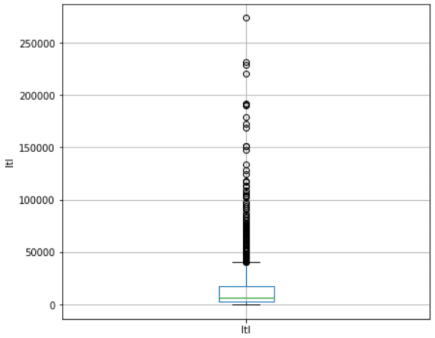

number
250
200
150
100
50
0
6
7
8
9
10
11
12
13
Insale

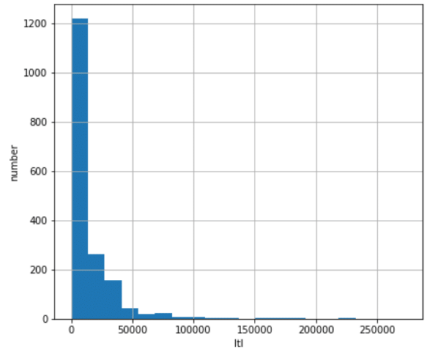

350000
300000
250000
200000
150000
100000
50000
0
lt
lt

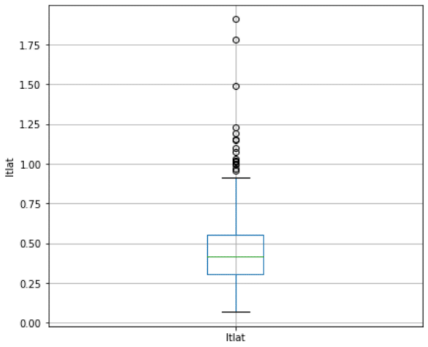

1000
800
600
400
200
0
number
0
50000
100000
150000
200000
250000
300000
350000
lt

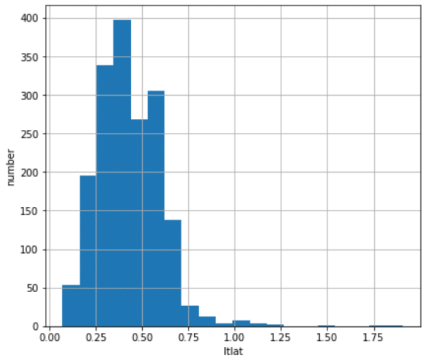

2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
ltat
ltat

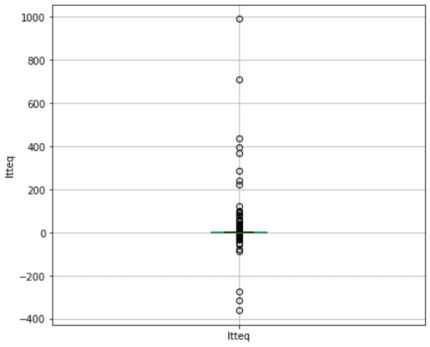

400
300
200
100
0
number
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
ltat

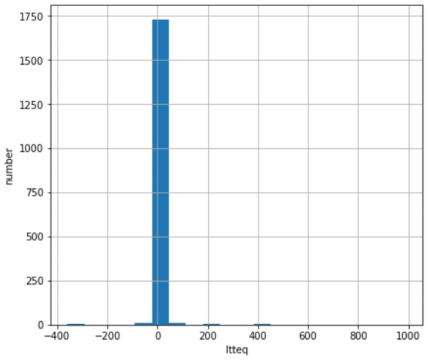

250000
200000
150000
100000
50000
0
ltl
ltl

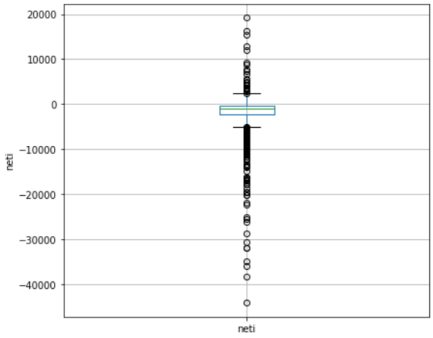

1200
1000
800
600
400
200
0
number
0
50000
100000
150000
200000
250000
ltl

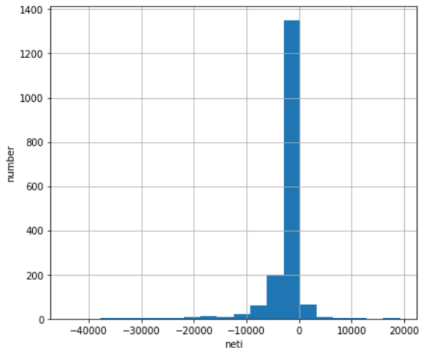

ltlat

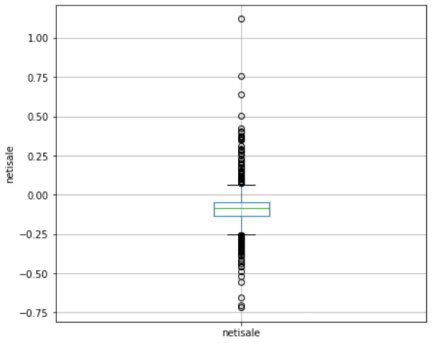

number
ltlat

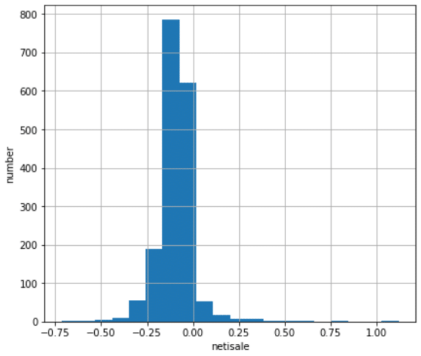

ltteq

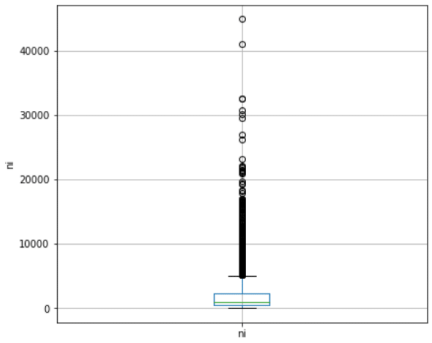

number
ltteq

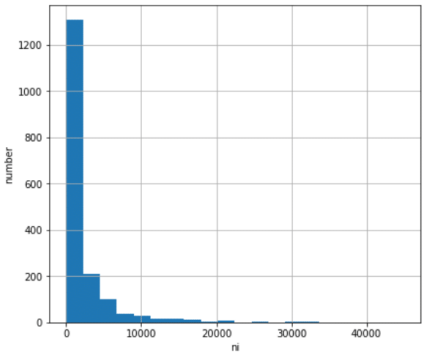

neti

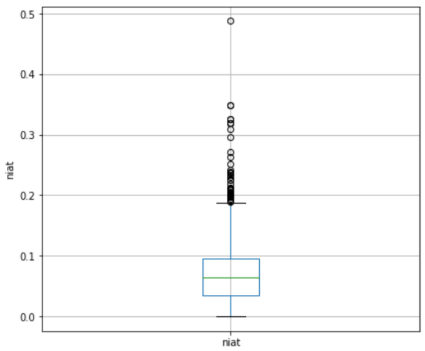

number
neti

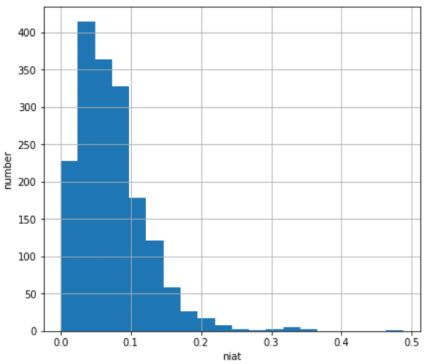

netisale
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
netisale

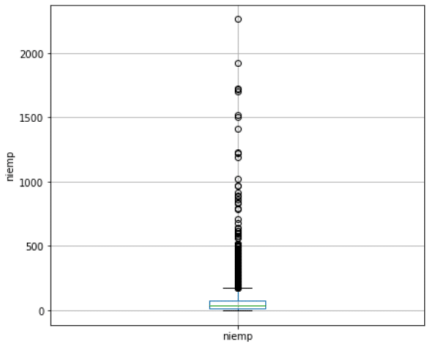

number
800
700
600
500
400
300
200
100
0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
netisale

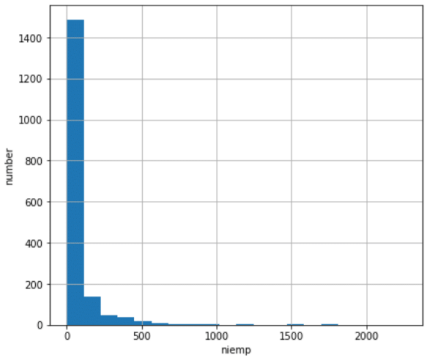

ni
40000
30000
20000
10000
0
ni

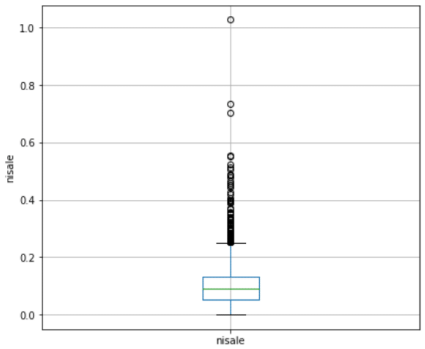

number
1200
1000
800
600
400
200
0
0
10000
20000
30000
40000
ni

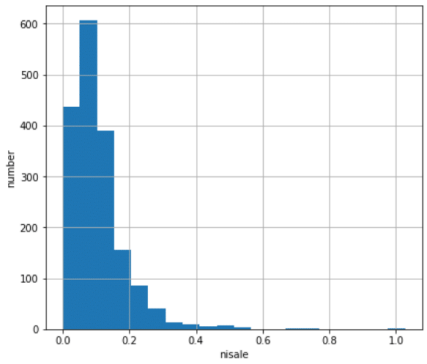

niat
0.5
0.4
0.3
0.2
0.1
0.0
niat

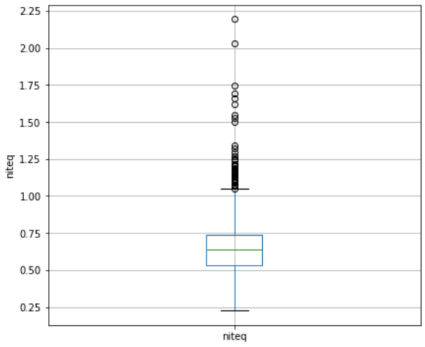

number
400
350
300
250
200
150
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5
niat

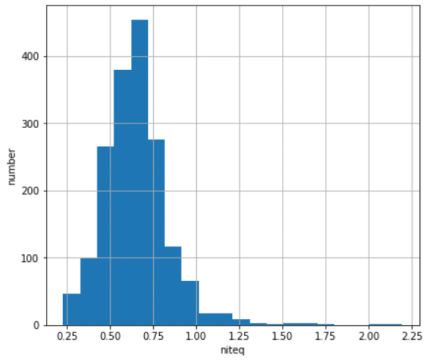

niemp
2000
1500
1000
500
0
niemp

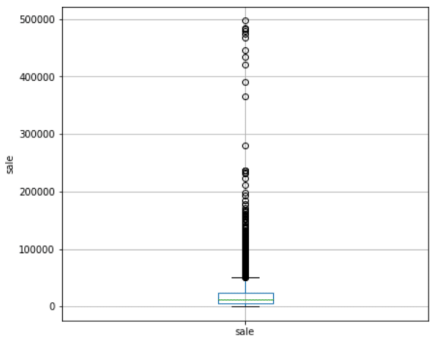

number
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
niemp

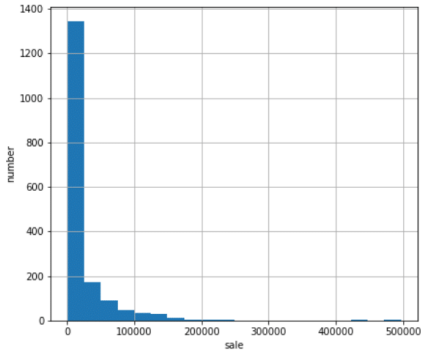

nisale
1.0
0.8
0.6
0.4
0.2
0.0
nisale

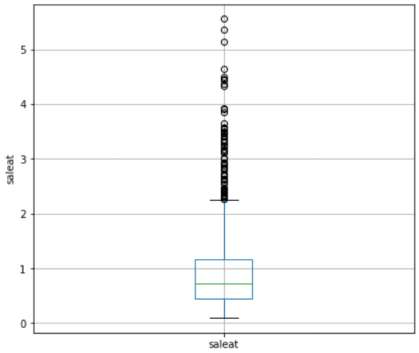

number
600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0
nisale

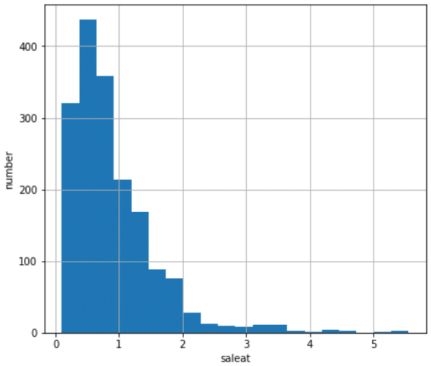

niteq
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
niteq

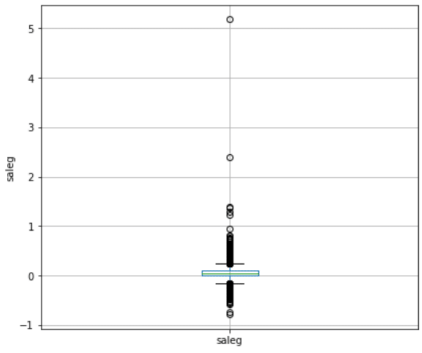

number
400
300
200
100
0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
niteq

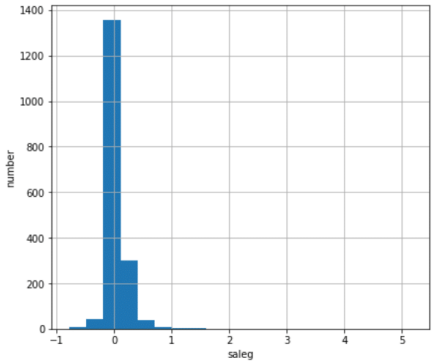

sale
500000
400000
300000
200000
100000
0
sale

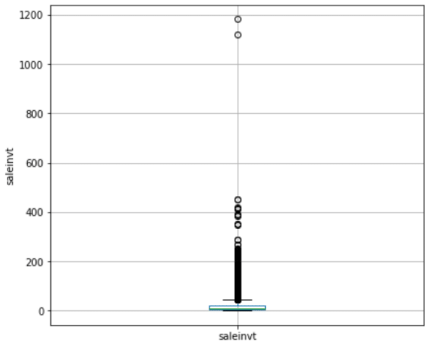

number
1400
1200
1000
800
600
400
200
0
0
100000
200000
300000
400000
500000
sale

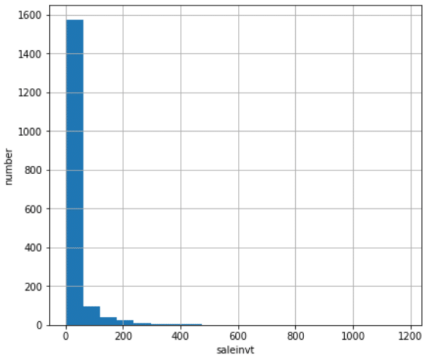

saleat
5
4
3
2
1
0
saleat

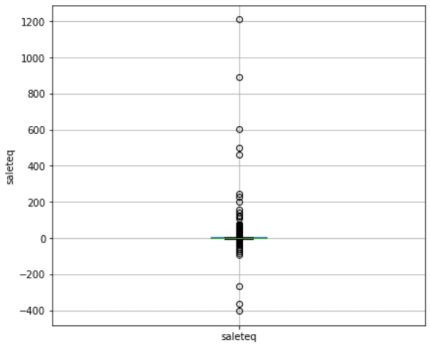

number
400
300
200
100
0
0
1
2
3
4
5
saleat

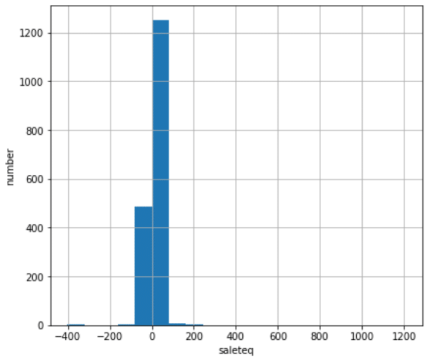

saleg
5
4
3
2
1
0
−1
saleg

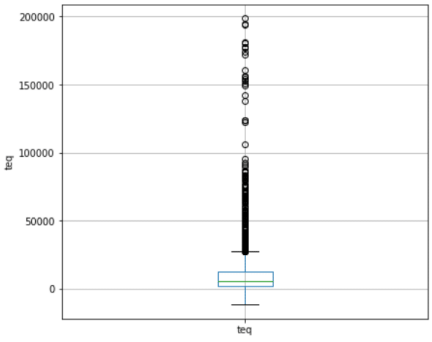

number
1400
1200
1000
800
600
400
200
0
−1
0
1
2
3
4
5
saleg

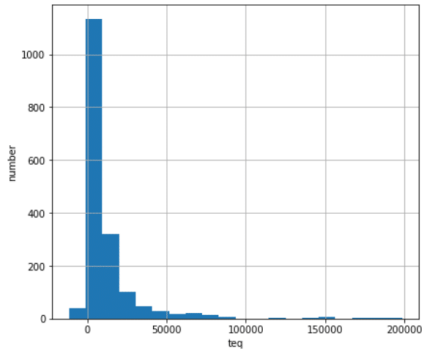

saleinvt
1200
1000
800
600
400
200
0
saleinvt

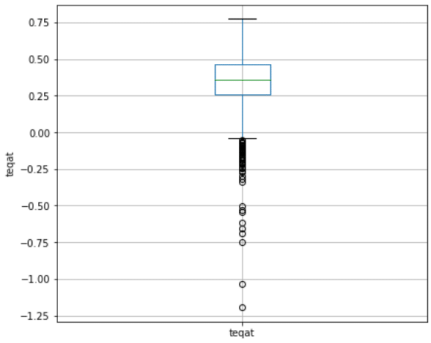

number
1600
1400
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200
saleinvt

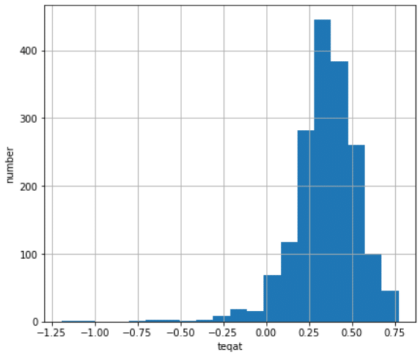

saleteq
1200
1000
800
600
400
200
0
−200
−400
saleteq

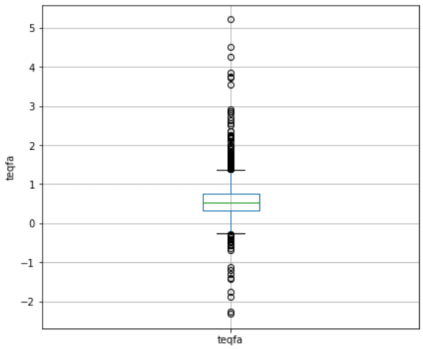

number
1200
1000
800
600
400
200
0
−400
−200
0
200
400
600
800
1000
1200
saleteq

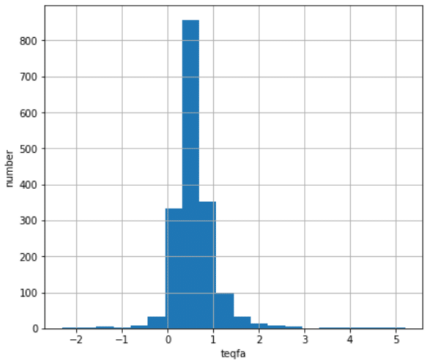

teq
200000
150000
100000
50000
0
teq

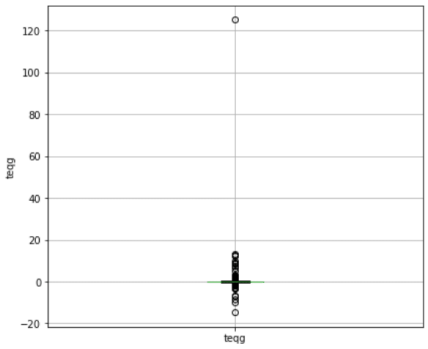

number
1000
800
600
400
200
0
0
50000
100000
150000
200000
teq

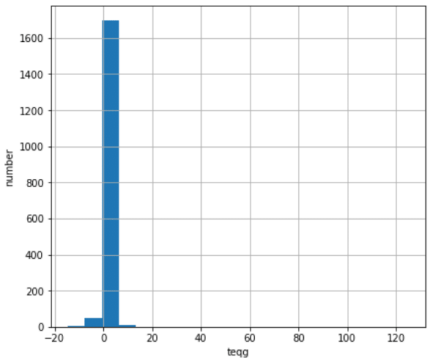

teqat

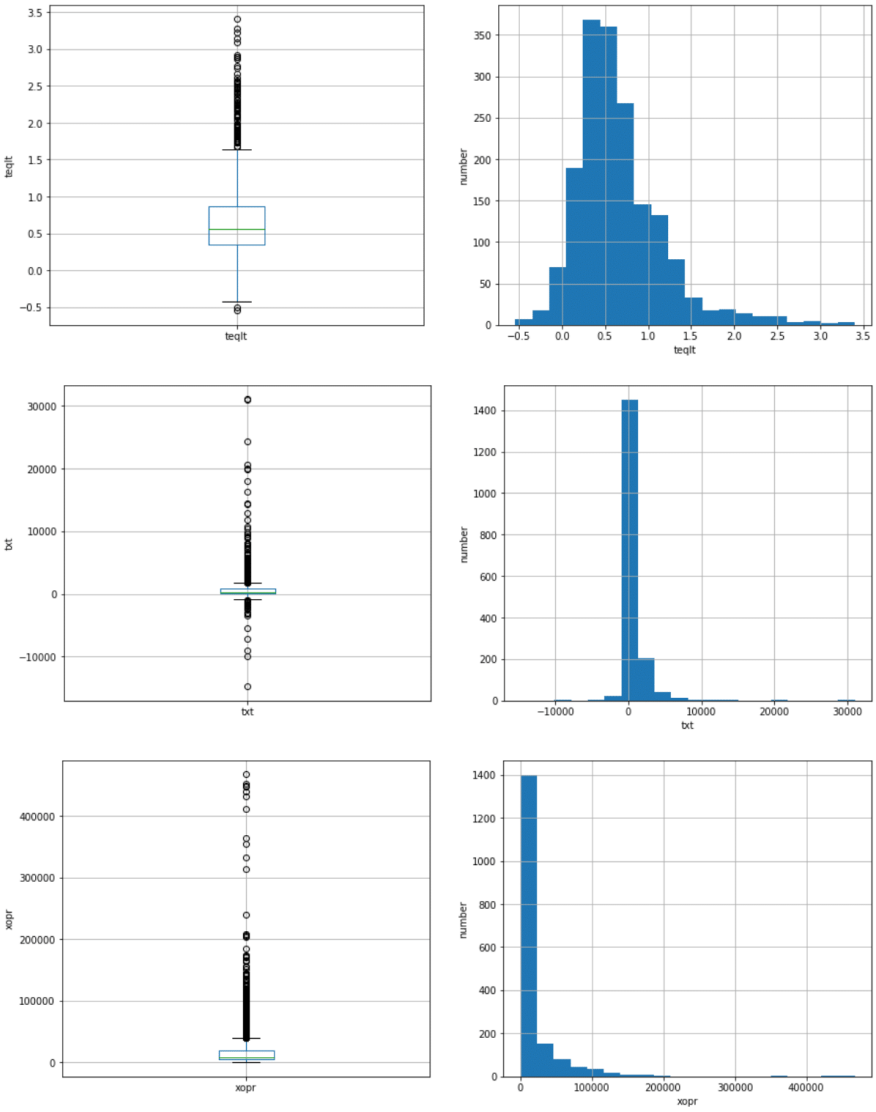

number
teqat


teqfa


number
teqfa


teqg


number
teqg



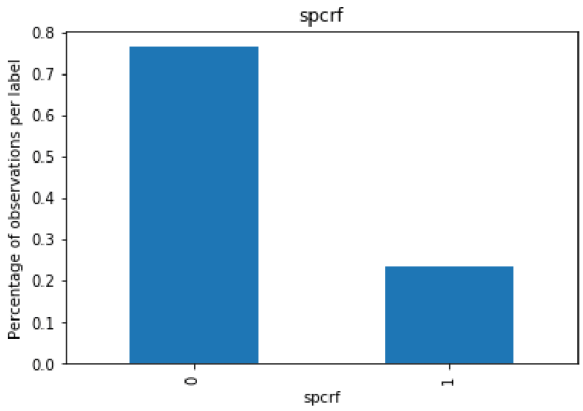
We can see the following variables present outliers:

5.2.2 Continue with discrete variables

We'll consider outliers those labels in discrete variables that are pressent in less than 1% of observations

In [19]:

```
# outlies in discrete variables
for var in discrete:
    (df.groupby(var)[var].count() / np.float(len(df))).plot.bar()
    plt.ylabel('Percentage of observations per label')
    plt.title(var)
    plt.show()
    #print(data[var].value_counts() / np.float(len(data)))
    print()
```



spcrf has outliers.

5.2.3 Number of labels: Cardinality

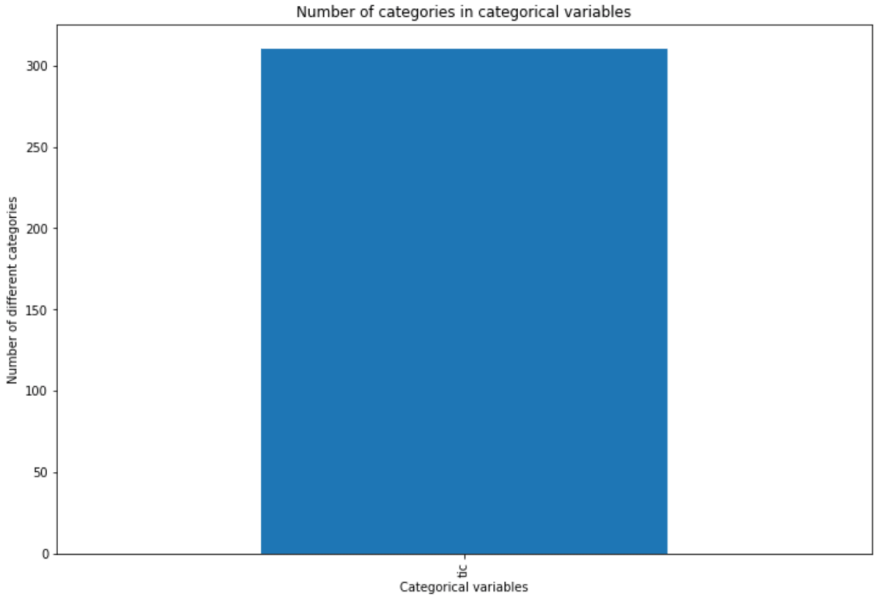
In [20]:

```
no_labels_ls = []
for var in categorical:
    no_labels_ls.append(len(df[var].unique()))
print(categorical, '\n',no_labels_ls)

tmp = pd.Series(no_labels_ls)
tmp.index = pd.Series(categorical)
tmp.plot.bar(figsize=(12,8))
plt.title('Number of categories in categorical variables')
plt.xlabel('Categorical variables')
plt.ylabel('Number of different categories')
```

```
['tic']
 [310]
```

Out[20]:

```
Text(0, 0.5, 'Number of different categories')
```



tic have a reasonable number of categories.

6. Split into train and test sets

In [21]:

```
# Let's separate into train and test set


X_SP500_train, X_SP500_test, Y_SP500_train, Y_SP500_test = train_test_split(df.loc[:,df
.columns!='spcrf'],
                                                                             df.spcrf, t
est_size=0.15, random_state=0)
X_SP500_train.shape, X_SP500_test.shape
```

Out[21]:

```
((1491, 71), (264, 71))
```

In [22]:

```
X_SP500_train.columns
```

Out[22]:

```
Index(['act', 'actdlct', 'actlct', 'at', 'atsale', 'cf', 'cfat', 'cfg',
       'cfsale', 'cfteq', 'cogs', 'dltt', 'dlttfa', 'dp', 'dvt', 'ebit',
       'ebitat', 'ebitda', 'ebitdaat', 'ebitdag', 'ebitdaneti', 'ebitdasal
e',
       'ebitdateq', 'ebitebitda', 'ebitsale', 'ebt', 'ebtat', 'ebtdpneti',
       'ebtebit', 'ebtteqltl', 'emp', 'empebit', 'empsale', 'fa', 'fyear',
       'gp', 'gplt', 'gpneti', 'gpsale', 'invt', 'invtsale', 'lct', 'lctl
t',
       'lnat', 'lnni', 'lnsale', 'lt', 'ltat', 'ltl', 'ltlat', 'ltteq', 'n
eti',
       'netisale', 'ni', 'niat', 'niemp', 'nisale', 'niteq', 'sale', 'sale
at',
       'saleg', 'saleinvt', 'saleteq', 'teq', 'teqat', 'teqfa', 'teqg',
       'teqlt', 'tic', 'txt', 'xopr'],
      dtype='object')
```

In [23]:

```
Y_SP500_train.head()
```

Out[23]:

```
1085    0
824     1
1557    0
1119    0
2439    0
Name: spcrf, dtype: int64
```

Part II: Feature engineering

7. Engineering missing values

7.1 Continuous variables

In [24]:

```
# print variables with missing data
# keep in mind that now that we created those new temporal variables, we
# are going to treat them as numerical and continuous as well:


# examine percentage of missing values
for col in numerical+temporal:
    if X_SP500_train[col].isnull().mean()>0:
        print(col, X_train[col].isnull().mean())
```

7.3 Engineering Missing data in Categorical Variables

In [25]:

```
categorical
```

Out[25]:

```
['tic']
```

In [26]:

```
# print variables with missing data
for col in categorical:
    if X_SP500_train[col].isnull().mean()>0:
        print(col, X_train[col].isnull().mean())
```

No missing values in Categorical variables

Sanity check to make sure I have no more nulls

In [27]:

```
# check absence of null values
for var in X_SP500_train.columns:
    if X_SP500_train[var].isnull().sum()>0:
        print(var, X_train[var].isnull().sum())
```

In [28]:

```
# check absence of null values
for var in X_SP500_test.columns:
    if X_SP500_test[var].isnull().sum()>0:
        print(var, X_test[var].isnull().sum())
```

Well done! I have no more NA in the variables.

8. Outlier engineering

8.1 Outlier identification and strategy setting

All the variables present outliers.

To decide how to handle these outliers, I shall check for their distribution skewness.

In [29]:

```
X_SP500_train.isnull().sum()
X_SP500_train.shape
```

Out[29]:

```
(1491, 71)
```

In [30]:

```
X_SP500_test.isnull().sum()
X_SP500_test.shape
```

Out[30]:

```
(264, 71)
```

11.Feature Scaling

In [31]:

```
# Check before it is all numerical
X_SP500_train.dtypes
```

Out[31]:

```
act           float64
actdlct       float64
actlct        float64
at            float64
atsale        float64
cf            float64
cfat          float64
cfg           float64
cfsale        float64
cfteq         float64
cogs          float64
dltt          float64
dlttfa        float64
dp            float64
dvt           float64
ebit          float64
ebitat        float64
ebitda        float64
ebitdaat      float64
ebitdag       float64
ebitdaneti    float64
ebitdasale    float64
ebitdateq     float64
ebitebitda    float64
ebitsale      float64
ebt           float64
ebtat         float64
ebtdpneti     float64
ebtebit       float64
ebtteqltl     float64
               ...
lct           float64
lctlt         float64
lnat          float64
lnni          float64
lnsale        float64
lt            float64
ltat          float64
ltl           float64
ltlat         float64
ltteq         float64
neti          float64
netisale      float64
ni            float64
niat          float64
niemp         float64
nisale        float64
niteq         float64
sale          float64
saleat        float64
saleg         float64
saleinvt      float64
saleteq       float64
teq           float64
teqat         float64
teqfa         float64
teqg          float64
teqlt         float64
tic            object
```

```
txt           float64
xopr          float64
Length: 71, dtype: object
```

In [32]:

```
# Check before it is all numerical
X_SP500_test.dtypes
```

Out[32]:

```
act           float64
actdlct       float64
actlct        float64
at            float64
atsale        float64
cf            float64
cfat          float64
cfg           float64
cfsale        float64
cfteq         float64
cogs          float64
dltt          float64
dlttfa        float64
dp            float64
dvt           float64
ebit          float64
ebitat        float64
ebitda        float64
ebitdaat      float64
ebitdag       float64
ebitdaneti    float64
ebitdasale    float64
ebitdateq     float64
ebitebitda    float64
ebitsale      float64
ebt           float64
ebtat         float64
ebtdpneti     float64
ebtebit       float64
ebtteqltl     float64
               ...
lct           float64
lctlt         float64
lnat          float64
lnni          float64
lnsale        float64
lt            float64
ltat          float64
ltl           float64
ltlat         float64
ltteq         float64
neti          float64
netisale      float64
ni            float64
niat          float64
niemp         float64
nisale        float64
niteq         float64
sale          float64
saleat        float64
saleg         float64
saleinvt      float64
saleteq       float64
teq           float64
teqat         float64
teqfa         float64
teqg          float64
teqlt         float64
tic            object
```

```
txt           float64
xopr          float64
Length: 71, dtype: object
```

In [33]:

```
#Find column names
X_SP500_train.columns
```

Out[33]:

```
Index(['act', 'actdlct', 'actlct', 'at', 'atsale', 'cf', 'cfat', 'cfg',
       'cfsale', 'cfteq', 'cogs', 'dltt', 'dlttfa', 'dp', 'dvt', 'ebit',
       'ebitat', 'ebitda', 'ebitdaat', 'ebitdag', 'ebitdaneti', 'ebitdasal
e',
       'ebitdateq', 'ebitebitda', 'ebitsale', 'ebt', 'ebtat', 'ebtdpneti',
       'ebtebit', 'ebtteqltl', 'emp', 'empebit', 'empsale', 'fa', 'fyear',
       'gp', 'gplt', 'gpneti', 'gpsale', 'invt', 'invtsale', 'lct', 'lctl
t',
       'lnat', 'lnni', 'lnsale', 'lt', 'ltat', 'ltl', 'ltlat', 'ltteq', 'n
eti',
       'netisale', 'ni', 'niat', 'niemp', 'nisale', 'niteq', 'sale', 'sale
at',
       'saleg', 'saleinvt', 'saleteq', 'teq', 'teqat', 'teqfa', 'teqg',
       'teqlt', 'tic', 'txt', 'xopr'],
      dtype='object')
```

In [34]:

```
# let's find the skewness of above variables
training_vars = numerical
for var in training_vars:
    print(var, 'skewness is', df[var].skew() )
```

```
act skewness is 4.088890504717582
actdlct skewness is 2.5598634976262526
actlct skewness is 6.362430874780734
at skewness is 3.775928088305178
atsale skewness is 1.642465571799068
cf skewness is 4.2921742001173255
cfat skewness is 1.4256162637847662
cfg skewness is -7.532965422060111
cfsale skewness is 2.4719869634442064
cfteq skewness is 18.386582196927424
cogs skewness is 5.565472753530459
dltt skewness is 4.71571594957898
dlttfa skewness is 3.511913312795555
dp skewness is 5.170197154922581
dvt skewness is 3.4972679742474884
ebit skewness is 3.7525105563511465
ebitat skewness is 1.358908011027437
ebitda skewness is 3.8974430637944746
ebitdaat skewness is 1.3315168935809427
ebitdag skewness is 33.69336380255187
ebitdaneti skewness is -5.51836200808925
ebitdasale skewness is 1.012144183850165
ebitdateq skewness is 16.732119785886933
ebitebitda skewness is -3.343633828049272
ebitsale skewness is 1.0713213346119053
ebt skewness is 5.186326086917846
ebtat skewness is 3.217599605624505
ebtdpneti skewness is 2.648047033272759
ebtebit skewness is 16.466837011103657
ebtteqltl skewness is 16.63881618224545
emp skewness is 10.349328101442863
empebit skewness is -18.923643156188493
empsale skewness is 3.210527697334079
fa skewness is 4.599748846821065
gp skewness is 4.076125709932687
gplt skewness is 1.7730587523533945
gpneti skewness is 11.726057411525657
gpsale skewness is 0.5530283957322638
invt skewness is 6.515672850841117
invtsale skewness is 1.7469557362484378
lct skewness is 3.684809752475689
lctlt skewness is 0.6485391287475231
lnat skewness is 0.3251285302941392
lnni skewness is -0.2899607953339182
lnsale skewness is 0.5644226783898939
lt skewness is 3.8825255991013514
ltat skewness is 1.4113009344811183
ltl skewness is 4.629490132511626
ltlat skewness is 1.211297406391993
ltteq skewness is 14.820610867189206
neti skewness is -4.120393436128353
netisale skewness is 1.531689791700061
ni skewness is 4.370468311845299
niat skewness is 1.668685795110482
niemp skewness is 6.2613225571329165
nisale skewness is 2.6818962974656815
niteq skewness is 1.4113009344811183
sale skewness is 5.416038823088144
saleat skewness is 2.2341356113788406
saleg skewness is 10.27962009252688
saleinvt skewness is 8.801585463023489
```

```
saleteq skewness is 15.794537980388375
teq skewness is 4.509053245441113
teqat skewness is -1.3981077744366357
teqfa skewness is 1.8947602308117701
teqg skewness is 35.99737452608082
teqlt skewness is 1.5287686928517246
txt skewness is 6.391265017399306
xopr skewness is 5.798016321355908
```

OK, I shall do :

- Robust Scaling for variables with an absolute skew above 2
- MinMax Scaling for variables with an absoulute skew between 1 and 2
- Standard for the rest

In [35]:

```
robustscalling= []
for var in training_vars:
    max_abs_value = 2
    if abs(df[var].skew()) > max_abs_value:
        robustscalling.append(var)
robustscalling
```

Out[35]:

```
['act',
 'actdlct',
 'actlct',
 'at',
 'cf',
 'cfg',
 'cfsale',
 'cfteq',
 'cogs',
 'dltt',
 'dlttfa',
 'dp',
 'dvt',
 'ebit',
 'ebitda',
 'ebitdag',
 'ebitdaneti',
 'ebitdateq',
 'ebitebitda',
 'ebt',
 'ebtat',
 'ebtdpneti',
 'ebtebit',
 'ebtteqltl',
 'emp',
 'empebit',
 'empsale',
 'fa',
 'gp',
 'gpneti',
 'invt',
 'lct',
 'lt',
 'ltl',
 'ltteq',
 'neti',
 'ni',
 'niemp',
 'nisale',
 'sale',
 'saleat',
 'saleg',
 'saleinvt',
 'saleteq',
 'teq',
 'teqg',
 'txt',
 'xopr']
```

In [36]:

```
# I Will start with Robust Scaling
scalerR = RobustScaler() # call the object
X_SP500_train_scaledR = scalerR.fit_transform(X_SP500_train[robustscalling]) # fit the
 scaler to the train set, and then scale it
X_SP500_test_scaledR= scalerR.transform(X_SP500_test[robustscalling]) # scale the test
 set
```

In [37]:

```
minmaxscalling=[]
for var in training_vars:
    max_abs_value = 2
    if (1 < abs(df[var].skew()) <=2):
        minmaxscalling.append(var)

minmaxscalling
```

Out[37]:

```
['atsale',
 'cfat',
 'ebitat',
 'ebitdaat',
 'ebitdasale',
 'ebitsale',
 'gplt',
 'invtsale',
 'ltat',
 'ltlat',
 'netisale',
 'niat',
 'niteq',
 'teqat',
 'teqfa',
 'teqlt']
```

In [38]:

```
#Continue with MinMax Scaling
scalerMM = MinMaxScaler() # create an instance

X_SP500_train_scaledMM= scalerMM.fit_transform(X_SP500_train[minmaxscalling]) #  fit  t
he scaler to the train set and then transform it
X_SP500_test_scaledMM= scalerMM.transform(X_SP500_test[minmaxscalling]) # transform (sc
ale) the test set
```

In [39]:

```
normalstandarization=[]
for var in training_vars:
    max_abs_value = 2
    if (abs((df[var].skew())) <=1):
        normalstandarization.append(var)

normalstandarization
```

Out[39]:

```
['gpsale', 'lctlt', 'lnat', 'lnni', 'lnsale']
```

In [40]:

```
# Normal Standarization
scalerN = StandardScaler() # create an object
X_SP500_train_scaledN = scalerN.fit_transform(X_SP500_train[normalstandarization]) # fi
t the scaler to the train set, and then transform it
X_SP500_test_scaledN = scalerN.transform(X_SP500_test[normalstandarization]) # transfor
m the test set
```

In [41]:

```
print(robustscalling,'\n','\n',minmaxscalling,'\n','\n',normalstandarization)
```

```
['act', 'actdlct', 'actlct', 'at', 'cf', 'cfg', 'cfsale', 'cfteq', 'cogs',
'dltt', 'dlttfa', 'dp', 'dvt', 'ebit', 'ebitda', 'ebitdag', 'ebitdaneti',
'ebitdateq', 'ebitebitda', 'ebt', 'ebtat', 'ebtdpneti', 'ebtebit', 'ebtteq
ltl', 'emp', 'empebit', 'empsale', 'fa', 'gp', 'gpneti', 'invt', 'lct', 'l
t', 'ltl', 'ltteq', 'neti', 'ni', 'niemp', 'nisale', 'sale', 'saleat', 'sa
leg', 'saleinvt', 'saleteq', 'teq', 'teqg', 'txt', 'xopr']

 ['atsale', 'cfat', 'ebitat', 'ebitdaat', 'ebitdasale', 'ebitsale', 'gpl
t', 'invtsale', 'ltat', 'ltlat', 'netisale', 'niat', 'niteq', 'teqat', 'te
qfa', 'teqlt']

 ['gpsale', 'lctlt', 'lnat', 'lnni', 'lnsale']
```

12. Save selected variables for features predictions

In [42]:

```
X_SP500_train_scaledR = pd.DataFrame(X_SP500_train_scaledR, columns = [robustscalling])
X_SP500_train_scaledMM = pd.DataFrame(X_SP500_train_scaledMM, columns = [minmaxscalling
])
X_SP500_train_scaledN = pd.DataFrame(X_SP500_train_scaledN, columns = [normalstandariza
tion])

######
X_SP500_test_scaledR = pd.DataFrame(X_SP500_test_scaledR, columns = [robustscalling])
X_SP500_test_scaledMM = pd.DataFrame(X_SP500_test_scaledMM, columns = [minmaxscalling])
X_SP500_test_scaledN = pd.DataFrame(X_SP500_test_scaledN, columns = [normalstandarizati
on])
```

In [43]:

```
X_SP500_train = pd.concat([X_SP500_train_scaledR, X_SP500_train_scaledMM, X_SP500_train
_scaledN], axis=1)
X_SP500_test = pd.concat([X_SP500_test_scaledR, X_SP500_test_scaledMM, X_SP500_test_sca
ledN], axis=1)
X_SP500_train.shape
X_SP500_train.head()
```

Out[43]:

	act	actdlct	actlct	at	cf	cfg	cfsale	cfteq	cogs
0	0.212587	0.213857	0.424675	0.016286	0.100690	0.646290	-0.164745	-0.016969	0.3
1	-0.162177	0.311398	0.059997	-0.387366	-0.408298	-2.045715	-0.749989	0.252928	-0.0
2	0.361525	0.094955	0.465516	0.534607	0.566013	-0.372379	0.035369	0.961412	0.9
3	0.760858	-0.106289	0.482503	2.019232	0.470537	-0.882937	-0.076570	-0.587693	1.0
4	-0.259802	1.758152	0.147281	-0.356811	-0.145690	0.018869	0.207345	0.855768	-0.1

In [44]:

```
X_SP500_train.shape
X_SP500_train.head()
X_SP500_train.dtypes
```

Out[44]:

```
act           float64
actdlct       float64
actlct        float64
at            float64
cf            float64
cfg           float64
cfsale        float64
cfteq         float64
cogs          float64
dltt          float64
dlttfa        float64
dp            float64
dvt           float64
ebit          float64
ebitda        float64
ebitdag       float64
ebitdaneti    float64
ebitdateq     float64
ebitebitda    float64
ebt           float64
ebtat         float64
ebtdpneti     float64
ebtebit       float64
ebtteqltl     float64
emp           float64
empebit       float64
empsale       float64
fa            float64
gp            float64
gpneti        float64
               ...
sale          float64
saleat        float64
saleg         float64
saleinvt      float64
saleteq       float64
teq           float64
teqg          float64
txt           float64
xopr          float64
atsale        float64
cfat          float64
ebitat        float64
ebitdaat      float64
ebitdasale    float64
ebitsale      float64
gplt          float64
invtsale      float64
ltat          float64
ltlat         float64
netisale      float64
niat          float64
niteq         float64
teqat         float64
teqfa         float64
teqlt         float64
gpsale        float64
lctlt         float64
lnat          float64
```

```
lnni          float64
lnsale        float64
Length: 69, dtype: object
```

In [45]:

```
X_SP500_train.to_csv('X_SP500_train.csv', index=False)
```

In [46]:

```
X_SP500_test.to_csv('X_SP500_test.csv', index=False)

X_SP500_test.head()
```

Out[46]:

	act	actdlct	actlct	at	cf	cfg	cfsale	cfteq	cog
0	1.397474	1.177996	2.920318	0.170201	0.352903	-1.663718	-0.536510	0.075971	1.1
1	0.622740	2.013688	1.909411	0.347222	1.073045	2.660095	3.146921	0.071688	-0.2
2	-0.214115	0.924112	0.133366	-0.430871	-0.411883	-1.403333	-0.690409	0.709063	-0.1
3	0.354524	-0.145037	0.159750	0.430402	0.392005	-0.314222	0.024993	-0.349430	0.5
4	-0.446344	-0.585325	-0.407037	-0.273011	-0.175311	-0.235639	2.569528	0.306445	-0.4

In [47]:

```
Y_SP500_train = pd.DataFrame(Y_SP500_train)
Y_SP500_train.to_csv('Y_SP500_train.csv', index=False)

Y_SP500_train.head(-1)
```

Out[47]:

	spcrf
1085	0
824	1
1557	0
1119	0
2439	0
1764	0
3044	0
1735	0
1150	1
2681	1
5	0
448	1
1120	0
465	0
1833	0
1621	0
2122	1
1412	0
950	0
2434	0
636	0
557	1
1116	0
40	0
2142	0
1666	0
1029	0
2388	0
2412	0
2529	0
...	...
1465	0
1871	0
259	0
1493	0
171	0
1641	0

	spcrf
1037	0
3240	0
1645	0
3075	1
378	0
206	0
1073	0
2864	0
1373	0
681	0
2669	0
1156	1
2844	0
2047	0
1152	1
568	0
1962	0
2616	0
3285	0
1477	0
1615	1
2278	0
3140	0
1091	0

1490 rows × 1 columns

In [48]:

```
Y_SP500_test = pd.DataFrame(Y_SP500_test)
Y_SP500_test.to_csv('Y_SP500_test.csv', index=False)
Y_SP500_test.head(-1)
```

Out[48]:

	spcrf
2232	0
1314	0
822	1
1090	0
2551	1
2481	0
2860	0
1914	0
3196	0
3319	0
309	0
518	0
3089	0
335	0
505	0
932	0
1469	0
1650	0
354	0
269	0
2141	0
1312	0
7	0
2040	0
1724	0
2352	0
2640	0
389	0
728	0
187	0
...	...
2635	0
1963	0
569	0
2033	1
1752	0
349	0

	spcrf
1897	0
352	0
855	0
65	0
775	1
1038	0
1880	0
1311	0
2138	0
1877	0
865	0
1025	0
751	0
548	0
1813	0
3305	0
3110	1
1491	0
2242	0
1961	0
1007	0
523	0
2540	0
1640	0

263 rows × 1 columns

In [49]:

```
Y_SP500_test.shape
```

Out[49]:

```
(264, 1)
```

In [ ]:

```

```

In [1]:

```
# import libraries
import pandas as pd
from sklearn import preprocessing
import sklearn.model_selection as ms
from sklearn import linear_model
import sklearn.metrics as sklm
import numpy as np
import numpy.random as nr
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as ss
import math
from glob import glob
from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier
from numpy import loadtxt
import xgboost as xgb

import warnings
warnings.filterwarnings('ignore')


%matplotlib inline
```

2. Import datasets

In [2]:

```
# Merge the two datasets
X_SP500_train=pd.read_csv('X_SP500_train.csv')
X_SP500_train.shape
```

Out[2]:

```
(1491, 69)
```

In [3]:

```
X_SP500_test=pd.read_csv('X_SP500_test.csv')
X_SP500_test.shape
```

Out[3]:

```
(264, 69)
```

In [4]:

```
# Merge the two datasets
Y_SP500_train=pd.read_csv('Y_SP500_train.csv',sep= ',')
Y_SP500_train.shape
```

Out[4]:

```
(1491, 1)
```

In [5]:

```
# Merge the two datasets
Y_SP500_test=pd.read_csv('Y_SP500_test.csv',sep= ',')
Y_SP500_test.shape
```

Out[5]:

```
(264, 1)
```

3. Data balancing and collinearity

3.3 Smote for data imbalance

In [6]:

```
train_input = X_SP500_train
train_output = Y_SP500_train
train_output
```

Out[6]:

	spcrf
0	0
1	1
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	1
10	0
11	1
12	0
13	0
14	0
15	0
16	1
17	0
18	0
19	0
20	0
21	1
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
...	...
1461	0
1462	0
1463	0
1464	0
1465	0
1466	0

	spcrf
1467	0
1468	0
1469	1
1470	0
1471	0
1472	0
1473	0
1474	0
1475	0
1476	0
1477	1
1478	0
1479	0
1480	1
1481	0
1482	0
1483	0
1484	0
1485	0
1486	1
1487	0
1488	0
1489	0
1490	1

1491 rows × 1 columns

In [16]:

```
from imblearn.over_sampling import SMOTE
from collections import Counter
print('Original dataset shape {}'.format(Counter(train_output)))
smt = SMOTE(random_state=123)
train_input_new, train_output_new = smt.fit_sample(train_input, train_output)
print('New dataset shape {}'.format(Counter(train_output_new)))
```

```
Original dataset shape Counter({'spcrf': 1})
New dataset shape Counter({0: 1123, 1: 1123})
```

4. Algorithm Testing

Logistic Regression

LR classifier (l1) penalty

In [45]:

```
#Given smote, we have to do a little adjustment
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
logreg = LogisticRegression(fit_intercept=True, penalty='l1',solver='liblinear',random_
state=123)
logreg.fit(X_SP500_train, Y_SP500_train)

logregprediction=logreg.predict(X_SP500_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Lasso Accuracy:",metrics.accuracy_score(logregprediction,Y_SP500_test))
```

```
Lasso Accuracy: 0.8143939393939394
```

LR Classifier (l2) penalty

In [47]:

```
#Given smote, we have to do a little adjustment
X__SP500rain, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_ne
w, train_output_new, test_size=0.20, random_state=123)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
logreg2 = LogisticRegression(fit_intercept=True, penalty='l2',random_state=123)
logreg2.fit(X_SP500_train, Y_SP500_train)
logreg2prediction=logreg2.predict(X_SP500_test)
#evaluation(Accuracy)
print("Ridge Accuracy:",metrics.accuracy_score(logreg2prediction,Y_SP500_test))
```

```
Ridge Accuracy: 0.803030303030303
```

Linear Discriminant Analysis

In [48]:

```
#Given smote, we have to do a little adjustment
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)


from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(X_SP500_train, Y_SP500_train)
lda.fit(X_SP500_train, Y_SP500_train)


lda_prediction=lda.predict(X_SP500_test)
#evaluation(Accuracy)
print("LDA Accuracy:",metrics.accuracy_score(lda_prediction,Y_SP500_test))
```

```
LDA Accuracy: 0.7992424242424242
```

Gaussian Naive Bayes

In [49]:

```
#Given smote, we have to do a little adjustment
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)


from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_SP500_train, Y_SP500_train)

gnbprediction=gnb.predict(X_SP500_test)
print("GNB Accuracy:",metrics.accuracy_score(gnbprediction,Y_SP500_test))
```

```
GNB Accuracy: 0.35984848484848486
```

Decision Tree Classifier

In [50]:

```
#remember becaus of the new data after imbalance adjustment, we use the following
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)

#importing module
from sklearn.tree import DecisionTreeClassifier
#making the instance
dtc= DecisionTreeClassifier(random_state=123)
#Learning
dtc.fit(X_SP500_train,Y_SP500_train)
#Prediction
dtcprediction=dtc.predict(X_SP500_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Decision Tree Classifier Accuracy:",metrics.accuracy_score(dtcprediction,Y_SP500
_test))
```

```
Decision Tree Classifier Accuracy: 0.8712121212121212
```

Random Forest Classifier

In [51]:

```
#remember becaus of the new data after imbalance adjustment, we use the following
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)

#importing module
from sklearn.ensemble import RandomForestClassifier
#making the instance
rfc=RandomForestClassifier(n_jobs=-1,random_state=123)
#Learning
rfc.fit(X_SP500_train,Y_SP500_train)
#Prediction
rfcprediction=rfc.predict(X_SP500_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Random Forest Classifier Accuracy:",metrics.accuracy_score(rfcprediction,Y_SP500
_test))
```

```
Random Forest Classifier Accuracy: 0.9507575757575758
```

SVM Classifier

In [52]:

```
#remember becaus of the new data after imbalance adjustment, we use the following
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)

#importing module
from sklearn import svm
#making the instance
svc = svm.SVC(random_state=123)
#learning
svc.fit(X_SP500_train,Y_SP500_train)
#Prediction
svcprediction=svc.predict(X_SP500_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy of SVM classifier:",metrics.accuracy_score(svcprediction,Y_SP500_test))
```

```
Accuracy of SVM classifier: 0.5984848484848485
```

K-NearestNeighbours Classifier

In [53]:

```
#remember becaus of the new data after imbalance adjustment, we use the following
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)

#importing module
from sklearn.neighbors import KNeighborsClassifier
#making the instance
knn = KNeighborsClassifier()
#learning
knn.fit(X_SP500_train,Y_SP500_train)
#Prediction
knnprediction=knn.predict(X_SP500_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("KNN classifier Accuracy:",metrics.accuracy_score(knnprediction,Y_SP500_test))
```

```
KNN classifier Accuracy: 0.8371212121212122
```

Ada Boost

In [54]:

```
#remember becaus of the new data after imbalance adjustment, we use the following
X_SP500_train, X_SP500_dev, Y_SP500_train, Y_SP500_dev = train_test_split(train_input_n
ew, train_output_new, test_size=0.20, random_state=123)

ada_model = AdaBoostClassifier(n_estimators=200, random_state=123)


#learning
ada_model.fit(X_SP500_train,Y_SP500_train)
#Prediction
ada_modelprediction=ada_model.predict(X_SP500_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("NN classifier Accuracy:",metrics.accuracy_score(ada_modelprediction,Y_SP500_test
))
```

```
NN classifier Accuracy: 0.8787878787878788
```

In [55]:

```
### 6. Compare models (baseline vs tuned models)
print("KNN classifier Accuracy:",metrics.accuracy_score(knnprediction,Y_SP500_test))
print("Random Forest Classifier Accuracy:",metrics.accuracy_score(rfcprediction,Y_SP500
_test))
print("Accuracy of SVM classifier:",metrics.accuracy_score(svcprediction,Y_SP500_test))
print("Decision Tree Classifier Accuracy:",metrics.accuracy_score(dtcprediction,Y_SP500
_test))
print("GNB Accuracy:",metrics.accuracy_score(gnbprediction,Y_SP500_test))
print("LDA Accuracy:",metrics.accuracy_score(lda_prediction,Y_SP500_test))
print("Ridge Accuracy:",metrics.accuracy_score(logreg2prediction,Y_SP500_test))
print("Lasso Accuracy:",metrics.accuracy_score(logregprediction,Y_SP500_test))
```

```
KNN classifier Accuracy: 0.8371212121212122
Random Forest Classifier Accuracy: 0.9507575757575758
Accuracy of SVM classifier: 0.5984848484848485
Decision Tree Classifier Accuracy: 0.8712121212121212
GNB Accuracy: 0.35984848484848486
LDA Accuracy: 0.7992424242424242
Ridge Accuracy: 0.803030303030303
Lasso Accuracy: 0.8143939393939394
```

5. Conclusion and Cross Validation

Sector Random Forest Classifier as the best classifier.

Now in the next notebook, I shall try to improve it by:

- Selecting the best feature set
- Tuning its parameters

In [56]:

```
# Cross Validation
print("Cross Validation Score: {:.2%}".format(np.mean(cross_val_score(rfc, X_SP500_trai
n, Y_SP500_train, cv=10))))
lda.fit(X_SP500_train, Y_SP500_train)
print("Dev Set score: {:.2%}".format(rfc.score(X_SP500_dev, Y_SP500_dev)))
```

```
Cross Validation Score: 93.77%
Dev Set score: 96.00%
```

In [ ]:

III. Feature selection, algorithm improvement

In this notebook, I shall run improve the best model from the preivous notebook (Random Forest Classificator) by undertaking the following steps:

```
- Feature selection (Hybrid method: recursive feature addition)
- Tune hyper paramters
```

Then I shall check and prove the improvement.

ROC visualization is provided at the end of the notebook

1 Importing libraries

In [8]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler

from sklearn import svm

from sklearn.metrics import roc_curve

from sklearn.ensemble import RandomForestClassifier


import warnings
warnings.filterwarnings('ignore')


%matplotlib inline
%matplotlib inline
```

2. Import datasets

In [9]:

```
# Merge the two datasets
X_SP500_train=pd.read_csv('X_SP500_train.csv',sep= ',')
X_SP500_train.shape
```

Out[9]:

```
(1491, 69)
```

In [10]:

```
X_SP500_test=pd.read_csv('X_SP500_test.csv',sep= ',')
X_SP500_test.shape
```

Out[10]:

```
(264, 69)
```

In [11]:

```
# Merge the two datasets
Y_SP500_train=pd.read_csv('Y_SP500_train.csv',sep= ',')
Y_SP500_train.shape
```

Out[11]:

```
(1491, 1)
```

In [12]:

```
Y_SP500_test=pd.read_csv('Y_SP500_test.csv',sep= ',')
Y_SP500_test.shape
```

Out[12]:

```
(264, 1)
```

3. Model creation & data balancing

3.2 Smote for data imbalance

In [13]:

```
train_input = X_SP500_train
train_output = Y_SP500_train
```

In [14]:

```
from imblearn.over_sampling import SMOTE
from collections import Counter
print('Original dataset shape {}'.format(Counter(train_output)))
smt = SMOTE(random_state=123)
train_input_new, train_output_new = smt.fit_sample(train_input, train_output)
print('New dataset shape {}'.format(Counter(train_output_new)))
```

```
Original dataset shape Counter({'spcrf': 1})
New dataset shape Counter({0: 1123, 1: 1123})
```

In [15]:

```
train_input_new = X_SP500_train
train_ouput_new = Y_SP500_train
```

Now both outcomes of the label are balanced

4. Feature selection (Feature addition)

In [16]:

```
# the first step of this procedure  consists in building
# a machine learning algorithm using all the available features
# and then determine the importance of the features according
# to the algorithm

# set the seed for reproducibility
seed_val = 1000000000
np.random.seed(seed_val)

# build initial model using all the features
model_all_features = RandomForestClassifier()
model_all_features.fit(X_SP500_train, Y_SP500_train)

# calculate the roc-auc in the test set
Y_SP500_pred_test = model_all_features.predict_proba(X_SP500_test)[:,1]

auc_score_all = roc_auc_score(Y_SP500_test, Y_SP500_pred_test)
print('Test all features RFC ROC AUC=%f' % (auc_score_all))
```

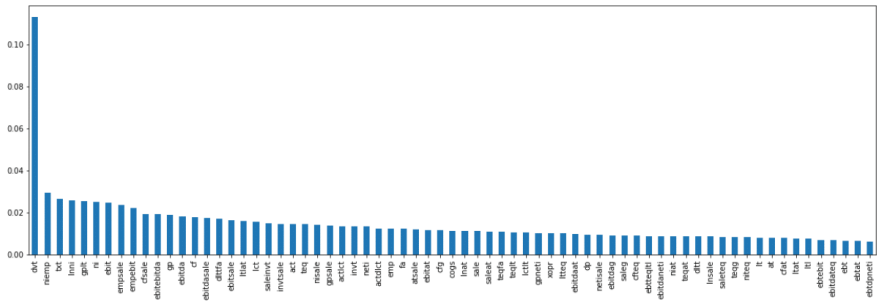
```
Test all features RFC ROC AUC=0.974835
```

In [17]:

```
# the second step consist of deriving the importance of
# each feature and ranking them from the most to the least
# important

# get feature name and importance
features = pd.Series(model_all_features.feature_importances_)
features.index = X_SP500_train.columns

# sort the features by importance
features.sort_values(ascending=False, inplace=True)

# plot
features.plot.bar(figsize=(20,6))
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27a92b33ba8>
```



```
 'lt',
 'at',
 'cfat',
 'ltat',
 'ltl',
 'ebtebit',
 'ebitdateq',
 'ebt',
 'ebtat',
 'ebtdpneti']
```

In [19]:

```
# next, we need to build a machine learning
# algorithm using only the most important feature

# set the seed for reproducibility
seed_val = 1000000000
np.random.seed(seed_val)

# build initial model using all the features
model_one_feature = RandomForestClassifier()

# train using only the most important feature
model_one_feature.fit(X_SP500_train[features[0]].to_frame(), Y_SP500_train)

# calculate the roc-auc in the test set
Y_SP500_pred_test = model_one_feature.predict_proba(X_SP500_test[features[0]].to_frame
())[:,1]
auc_score_first = roc_auc_score(Y_SP500_test, Y_SP500_pred_test)
print('Test one feature xgb ROC AUC=%f' % (auc_score_first))
```

```
Test one feature xgb ROC AUC=0.701421
```

In [20]:

```
# the final step consists in adding one at a time
# all the features, from the most to the least
# important, and build an xgboost at each round.

# once we build the model, we calculate the new roc-auc
# if the new roc-auc is bigger than the original one
# (with one feature), then that feature that was added
# was important, and we should keep it.
# otherwise, we should remove the feature

# recursive feature addition:

# first we arbitrarily set the increase in roc-auc
# if the increase is above this threshold,
# the feature will be kept
tol = 0.001

print('doing recursive feature addition')

# we initialise a list where we will collect the
# features we should keep
features_to_keep = [features[0]]

# set a counter to know how far ahead the loop is going
count = 1

# now we loop over all the features, in order of importance:
# remember that features is the list of ordered features
# by importance
for feature in features[1:]:
    print()
    print('testing feature: ', feature, ' which is feature ', count,
          ' out of ', len(features))
    count = count + 1

    # initialise model
    model_int = RandomForestClassifier()

    # fit model with the selected features
    # and the feature to be evaluated
    model_int.fit(
        X_SP500_train[features_to_keep + [feature] ], Y_SP500_train)

    # make a prediction over the test set
    Y_SP500_pred_test = model_int.predict_proba(
        X_SP500_test[features_to_keep + [feature] ])[:,1]

    # calculate the new roc-auc
    auc_score_int = roc_auc_score(Y_SP500_test, Y_SP500_pred_test)
    print('New Test ROC AUC={}'.format((auc_score_int)))

    # print the original roc-auc with one feature
    print('All features Test ROC AUC={}'.format((auc_score_first)))

    # determine the increase in the roc-auc
    diff_auc = auc_score_int - auc_score_first

    # compare the increase in roc-auc with the tolerance
    # we set previously
```

```
    if diff_auc >= tol:
        print('Increase in ROC AUC={}'.format(diff_auc))
        print('keep: ', feature)
        print
        # if the increase in the roc is bigger than the threshold
        # we keep the feature and re-adjust the roc-auc to the new value
        # considering the added feature
        auc_score_first = auc_score_int

        # and we append the feature to keep to the list
        features_to_keep.append(feature)
    else:
        # we ignore the feature
        print('Increase in ROC AUC={}'.format(diff_auc))
        print('remove: ', feature)
        print


# now the loop is finished, we evaluated all the features
print('DONE!!')
print('total features to keep: ', len(features_to_keep))
```

```
doing recursive feature addition

testing feature:  niemp  which is feature  1  out of  69
New Test ROC AUC=0.7215626585489598
All features Test ROC AUC=0.7014205986808726
Increase in ROC AUC=0.02014205986808726
keep:  niemp

testing feature:  txt  which is feature  2  out of  69
New Test ROC AUC=0.8099949264332826
All features Test ROC AUC=0.7215626585489598
Increase in ROC AUC=0.08843226788432279
keep:  txt

testing feature:  lnni  which is feature  3  out of  69
New Test ROC AUC=0.8143581938102487
All features Test ROC AUC=0.8099949264332826
Increase in ROC AUC=0.004363267376966018
keep:  lnni

testing feature:  gplt  which is feature  4  out of  69
New Test ROC AUC=0.8726534753932014
All features Test ROC AUC=0.8143581938102487
Increase in ROC AUC=0.05829528158295272
keep:  gplt

testing feature:  ni  which is feature  5  out of  69
New Test ROC AUC=0.8712328767123287
All features Test ROC AUC=0.8726534753932014
Increase in ROC AUC=-0.0014205986808726312
remove:  ni

testing feature:  ebit  which is feature  6  out of  69
New Test ROC AUC=0.8785895484525622
All features Test ROC AUC=0.8726534753932014
Increase in ROC AUC=0.005936073059360836
keep:  ebit

testing feature:  empsale  which is feature  7  out of  69
New Test ROC AUC=0.91298833079655
All features Test ROC AUC=0.8785895484525622
Increase in ROC AUC=0.034398782343987744
keep:  empsale

testing feature:  empebit  which is feature  8  out of  69
New Test ROC AUC=0.9135464231354643
All features Test ROC AUC=0.91298833079655
Increase in ROC AUC=0.0005580923389143511
remove:  empebit

testing feature:  cfsale  which is feature  9  out of  69
New Test ROC AUC=0.9252156265854896
All features Test ROC AUC=0.91298833079655
Increase in ROC AUC=0.012227295788939663
keep:  cfsale

testing feature:  ebitebitda  which is feature  10  out of  69
New Test ROC AUC=0.9423642820903094
All features Test ROC AUC=0.9252156265854896
Increase in ROC AUC=0.01714865550481981
keep:  ebitebitda
```

```
testing feature:  gp  which is feature  11  out of  69
New Test ROC AUC=0.9348046676813799
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=-0.007559614408929494
remove:  gp

testing feature:  ebitda  which is feature  12  out of  69
New Test ROC AUC=0.930593607305936
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=-0.011770674784373436
remove:  ebitda

testing feature:  cf  which is feature  13  out of  69
New Test ROC AUC=0.9316590563165905
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=-0.010705225773718907
remove:  cf

testing feature:  ebitdasale  which is feature  14  out of  69
New Test ROC AUC=0.9348554033485541
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=-0.0075088787417553204
remove:  ebitdasale

testing feature:  dlttfa  which is feature  15  out of  69
New Test ROC AUC=0.9386098427194316
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=-0.0037544393708778268
remove:  dlttfa

testing feature:  ebitsale  which is feature  16  out of  69
New Test ROC AUC=0.9292237442922374
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=-0.013140537798072005
remove:  ebitsale

testing feature:  ltlat  which is feature  17  out of  69
New Test ROC AUC=0.9528665651953323
All features Test ROC AUC=0.9423642820903094
Increase in ROC AUC=0.01050228310502288
keep:  ltlat

testing feature:  lct  which is feature  18  out of  69
New Test ROC AUC=0.9545915778792491
All features Test ROC AUC=0.9528665651953323
Increase in ROC AUC=0.0017250126839167823
keep:  lct

testing feature:  saleinvt  which is feature  19  out of  69
New Test ROC AUC=0.9567732115677321
All features Test ROC AUC=0.9545915778792491
Increase in ROC AUC=0.002181633688483009
keep:  saleinvt

testing feature:  invtsale  which is feature  20  out of  69
New Test ROC AUC=0.9608828006088279
All features Test ROC AUC=0.9567732115677321
Increase in ROC AUC=0.004109589041095818
keep:  invtsale
```

```
testing feature:  act  which is feature  21  out of  69
New Test ROC AUC=0.9649416539827499
All features Test ROC AUC=0.9608828006088279
Increase in ROC AUC=0.004058853373921978
keep:  act

testing feature:  teq  which is feature  22  out of  69
New Test ROC AUC=0.9656012176560121
All features Test ROC AUC=0.9649416539827499
Increase in ROC AUC=0.0006595636732622534
remove:  teq

testing feature:  nisale  which is feature  23  out of  69
New Test ROC AUC=0.9605783866057838
All features Test ROC AUC=0.9649416539827499
Increase in ROC AUC=-0.004363267376966129
remove:  nisale

testing feature:  gpsale  which is feature  24  out of  69
New Test ROC AUC=0.9620497209538306
All features Test ROC AUC=0.9649416539827499
Increase in ROC AUC=-0.0028919330289193246
remove:  gpsale

testing feature:  actlct  which is feature  25  out of  69
New Test ROC AUC=0.9593099949264332
All features Test ROC AUC=0.9649416539827499
Increase in ROC AUC=-0.005631659056316685
remove:  actlct

testing feature:  invt  which is feature  26  out of  69
New Test ROC AUC=0.96986301369863
All features Test ROC AUC=0.9649416539827499
Increase in ROC AUC=0.004921359715880147
keep:  invt

testing feature:  neti  which is feature  27  out of  69
New Test ROC AUC=0.9639776763064435
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0058853373921865515
remove:  neti

testing feature:  actdlct  which is feature  28  out of  69
New Test ROC AUC=0.9645357686453577
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0053272450532723115
remove:  actdlct

testing feature:  emp  which is feature  29  out of  69
New Test ROC AUC=0.9670218163368849
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0028411973617451514
remove:  emp

testing feature:  fa  which is feature  30  out of  69
New Test ROC AUC=0.967427701674277
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.002435312024352987
remove:  fa

testing feature:  atsale  which is feature  31  out of  69
```

```
New Test ROC AUC=0.9680365296803654
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0018264840182646847
remove:  atsale

testing feature:  ebitat  which is feature  32  out of  69
New Test ROC AUC=0.9690512430238457
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.000811770674784329
remove:  ebitat

testing feature:  cfg  which is feature  33  out of  69
New Test ROC AUC=0.9666159309994925
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.003247082699137538
remove:  cfg

testing feature:  cogs  which is feature  34  out of  69
New Test ROC AUC=0.9682902080162354
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0015728056823945957
remove:  cogs

testing feature:  lnat  which is feature  35  out of  69
New Test ROC AUC=0.9672247590055809
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0026382546930491246
remove:  lnat

testing feature:  sale  which is feature  36  out of  69
New Test ROC AUC=0.9672247590055809
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0026382546930491246
remove:  sale

testing feature:  saleat  which is feature  37  out of  69
New Test ROC AUC=0.9676813800101471
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.002181633688482898
remove:  saleat

testing feature:  teqfa  which is feature  38  out of  69
New Test ROC AUC=0.9692541856925418
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0006088280060881912
remove:  teqfa

testing feature:  teqlt  which is feature  39  out of  69
New Test ROC AUC=0.969558599695586
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.0003044140030440401
remove:  teqlt

testing feature:  lctlt  which is feature  40  out of  69
New Test ROC AUC=0.9632673769660072
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.006595636732622867
remove:  lctlt

testing feature:  gpneti  which is feature  41  out of  69
New Test ROC AUC=0.9638762049720954
```

```
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.005986808726534676
remove:  gpneti

testing feature:  xopr  which is feature  42  out of  69
New Test ROC AUC=0.9648909183155758
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.004972095383054209
remove:  xopr

testing feature:  ltteq  which is feature  43  out of  69
New Test ROC AUC=0.9668188736681887
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=-0.003044140030441289
remove:  ltteq

testing feature:  ebitdaat  which is feature  44  out of  69
New Test ROC AUC=0.9709284627092846
All features Test ROC AUC=0.96986301369863
Increase in ROC AUC=0.001065449010654529
keep:  ebitdaat

testing feature:  dp  which is feature  45  out of  69
New Test ROC AUC=0.9675799086757991
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.0033485540334854402
remove:  dp

testing feature:  netisale  which is feature  46  out of  69
New Test ROC AUC=0.9702688990360223
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.0006595636732622534
remove:  netisale

testing feature:  ebitdag  which is feature  47  out of  69
New Test ROC AUC=0.9602739726027397
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.010654490106544845
remove:  ebitdag

testing feature:  saleg  which is feature  48  out of  69
New Test ROC AUC=0.965398274987316
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.00553018772196856
remove:  saleg

testing feature:  cfteq  which is feature  49  out of  69
New Test ROC AUC=0.9718924403855911
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=0.0009639776763065155
remove:  cfteq

testing feature:  ebtteqltl  which is feature  50  out of  69
New Test ROC AUC=0.9660071029934043
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.004921359715880258
remove:  ebtteqltl

testing feature:  ebitdaneti  which is feature  51  out of  69
New Test ROC AUC=0.9693556570268899
All features Test ROC AUC=0.9709284627092846
```

```
Increase in ROC AUC=-0.0015728056823947067
remove:  ebitdaneti

testing feature:  niat  which is feature  52  out of  69
New Test ROC AUC=0.9675291730086251
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.0033992897006595024
remove:  niat

testing feature:  teqat  which is feature  53  out of  69
New Test ROC AUC=0.9707255200405885
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.00020294266869602673
remove:  teqat

testing feature:  dltt  which is feature  54  out of  69
New Test ROC AUC=0.9678335870116692
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.0030948756976153513
remove:  dltt

testing feature:  lnsale  which is feature  55  out of  69
New Test ROC AUC=0.9675291730086251
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=-0.0033992897006595024
remove:  lnsale

testing feature:  saleteq  which is feature  56  out of  69
New Test ROC AUC=0.9724505327245053
All features Test ROC AUC=0.9709284627092846
Increase in ROC AUC=0.0015220700152207556
keep:  saleteq

testing feature:  teqg  which is feature  57  out of  69
New Test ROC AUC=0.9724505327245053
All features Test ROC AUC=0.9724505327245053
Increase in ROC AUC=0.0
remove:  teqg

testing feature:  niteq  which is feature  58  out of  69
New Test ROC AUC=0.9746321664129883
All features Test ROC AUC=0.9724505327245053
Increase in ROC AUC=0.002181633688483009
keep:  niteq

testing feature:  lt  which is feature  59  out of  69
New Test ROC AUC=0.9734652460679858
All features Test ROC AUC=0.9746321664129883
Increase in ROC AUC=-0.0011669203450025423
remove:  lt

testing feature:  at  which is feature  60  out of  69
New Test ROC AUC=0.9764586504312532
All features Test ROC AUC=0.9746321664129883
Increase in ROC AUC=0.0018264840182649067
keep:  at

testing feature:  cfat  which is feature  61  out of  69
New Test ROC AUC=0.9741755454084221
All features Test ROC AUC=0.9764586504312532
Increase in ROC AUC=-0.0022831050228311334
```

```
remove:  cfat

testing feature:  ltat  which is feature  62  out of  69
New Test ROC AUC=0.976103500761035
All features Test ROC AUC=0.9764586504312532
Increase in ROC AUC=-0.0003551496702182133
remove:  ltat

testing feature:  ltl  which is feature  63  out of  69
New Test ROC AUC=0.9764586504312531
All features Test ROC AUC=0.9764586504312532
Increase in ROC AUC=-1.1102230246251565e-16
remove:  ltl

testing feature:  ebtebit  which is feature  64  out of  69
New Test ROC AUC=0.9784373414510401
All features Test ROC AUC=0.9764586504312532
Increase in ROC AUC=0.0019786910197868712
keep:  ebtebit

testing feature:  ebitdateq  which is feature  65  out of  69
New Test ROC AUC=0.9749365804160325
All features Test ROC AUC=0.9784373414510401
Increase in ROC AUC=-0.003500761035007627
remove:  ebitdateq

testing feature:  ebt  which is feature  66  out of  69
New Test ROC AUC=0.9745306950786403
All features Test ROC AUC=0.9784373414510401
Increase in ROC AUC=-0.003906646372399791
remove:  ebt

testing feature:  ebtat  which is feature  67  out of  69
New Test ROC AUC=0.9723997970573313
All features Test ROC AUC=0.9784373414510401
Increase in ROC AUC=-0.006037544393708849
remove:  ebtat

testing feature:  ebtdpneti  which is feature  68  out of  69
New Test ROC AUC=0.9752409944190765
All features Test ROC AUC=0.9784373414510401
Increase in ROC AUC=-0.0031963470319635867
remove:  ebtdpneti
DONE!!
total features to keep:  20
```

We go down from 69 to 20 featues, and our ROC increases.

In [21]:

```
# capture the 20 selected features
seed_val = 1000000000
np.random.seed(seed_val)

# build initial model
final_xgb = RandomForestClassifier()

# fit the model with the selected features
final_xgb.fit(X_SP500_train[features_to_keep], Y_SP500_train)

# make predictions
Y_SP500_pred_test = final_xgb.predict_proba(X_SP500_test[features_to_keep])[:,1]

# calculate roc-auc
auc_score_final = roc_auc_score(Y_SP500_test, Y_SP500_pred_test)
print('Test selected features ROC AUC=%f' % (auc_score_final))
```

```
Test selected features ROC AUC=0.975698
```

5. Tune mode hyperparamters

In [30]:

```
from sklearn.model_selection import RandomizedSearchCV
# number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# number of features at every split
max_features = ['auto', 'sqrt']


# max depth
max_depth = [int(x) for x in np.linspace(100, 500, num = 11)]
max_depth.append(None)
# create random grid
random_grid = {
 'n_estimators': n_estimators,
 'max_features': max_features,
 'max_depth': max_depth
 }
from sklearn.ensemble import RandomForestClassifier
#making the instance
rfc=RandomForestClassifier(random_state=123)
# Random search of parameters
rfc_model = RandomizedSearchCV(estimator = rfc, param_distributions = random_grid, n_it
er = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the model
rfc_model.fit(X_SP500_train[features_to_keep], Y_SP500_train)
# print results
print(rfc_model.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  6.5min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 12.9min finished

{'n_estimators': 200, 'max_features': 'sqrt', 'max_depth': None}
```

Now we test best parameters below

In [31]:

```
seed_val = 1000

np.random.seed(seed_val)

rfc2 = RandomForestClassifier(n_estimators=rfc_model.best_params_['n_estimators'],
                              max_depth=rfc_model.best_params_['max_depth'],
                              max_features=rfc_model.best_params_['max_features'])
rfc2.fit(X_SP500_train[features_to_keep],Y_SP500_train)
rfc2_predict = rfc2.predict_proba(X_SP500_test[features_to_keep])[:,1]

auc_score_final2 = roc_auc_score(Y_SP500_test, rfc2_predict)
print('Test selected features ROC AUC=%f' % (auc_score_final2))
```

```
Test selected features ROC AUC=0.975748
```

Check for improvement

In [32]:

```
print ('Model ROC AUC improvement is',(auc_score_final2-auc_score_final))
```

```
Model ROC AUC improvement is 5.0735667174173216e-05
```

If we compare original ROC AUC with the latest one, we find:

In [33]:

```
print('Test all features RFC ROC AUC=%f' % (auc_score_all))
print('Test selected features ROC AUC=%f' % (auc_score_final2))
print ('Model ROC AUC change is',(auc_score_final2-auc_score_all))
```

```
Test all features RFC ROC AUC=0.974835
Test selected features ROC AUC=0.975748
Model ROC AUC change is 0.0009132420091325644
```

We have improved our model accuracy by 0.0079 and reduced the featuer space from 69 to 20!

7.Save the model

In [34]:

```
from sklearn.externals import joblib
from joblib import dump, load
dump(rfc2,'S&P500_credit_rating_model.joblib')
```

Out[34]:

```
['S&P500_credit_rating_model.joblib']
```

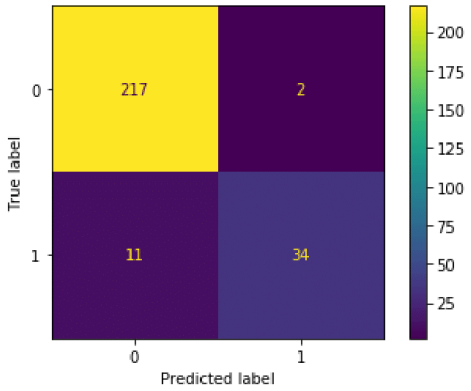
8. Evidence that the model will generalize well

In [35]:

```
import sklearn.metrics as sklm

sklm.plot_confusion_matrix(rfc2,X_SP500_test[features_to_keep],Y_SP500_test,values_format='.4g')
```

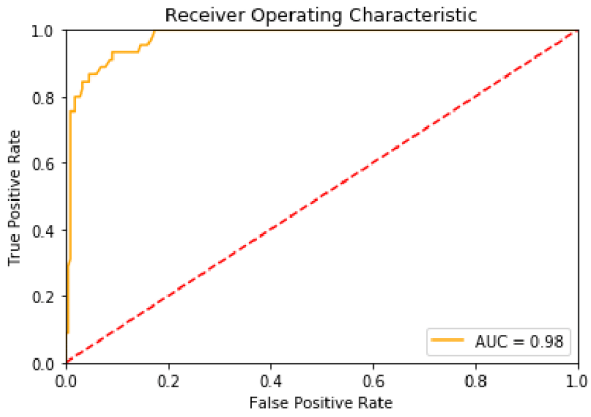
Out[35]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27a9136f4e0>
```



In [36]:

```
df = pd.concat([X_SP500_train.append(X_SP500_test), Y_SP500_train.append(Y_SP500_test
)], axis = 1)
labels = np.array(df['spcrf'])
probabilities = rfc2.predict_proba(X_SP500_test[features_to_keep])

def plot_auc(labels, probs):
    ## Compute the false positive rate, true positive rate
    ## and threshold along with the AUC
    fpr, tpr, threshold = sklm.roc_curve(labels, probs[:,1])
    auc = sklm.auc(fpr, tpr)

    ## Plot the result
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, color = 'orange', label = 'AUC = %0.2f' % auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

plot_auc(Y_SP500_test, probabilities)
```



In [ ]:

```
```